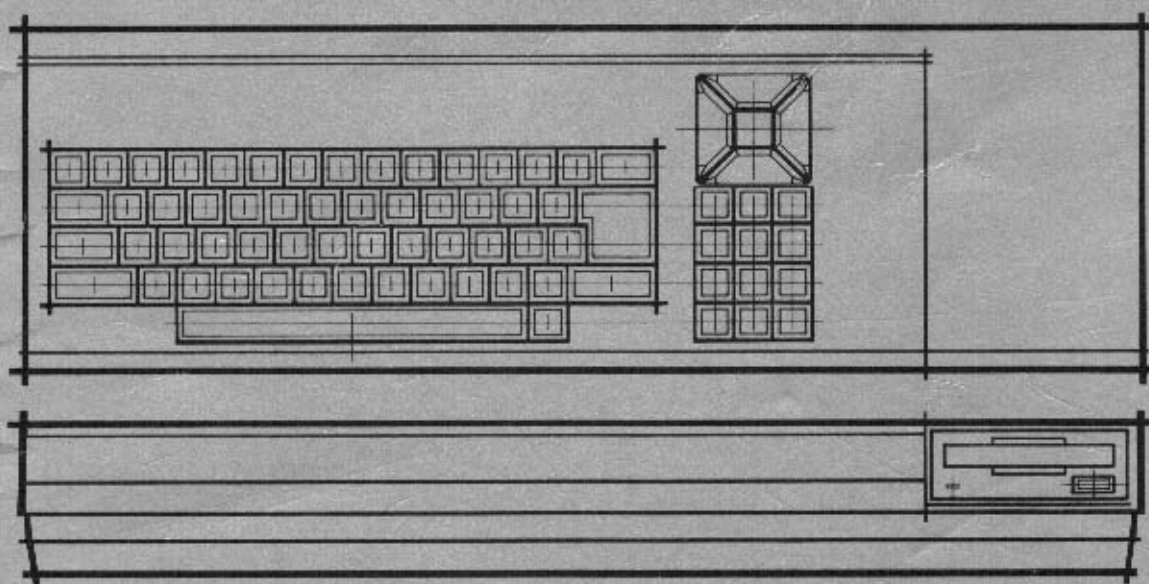


AMSTRAD



USER INSTRUCTIONS

Introduction

AMSTRAD CPC664

Integrated Computer/Disc System

About us....

The AMSTRAD CPC664 follows in the tradition of the successful CPC464 computer system. The widespread acclaim given to this computer has encouraged us to take the next logical step, and to combine the best elements of the CPC464 and DDI1 disc drive, together with many new features and enhancements. The resulting package - the CPC664, offers a performance and wealth of facilities unrivalled in the world of personal computers today.

Your product is supported by one of the largest consumer electronics organisations in the country, and the AMSTRAD computer users' club with its authoritative monthly magazine is already firmly established as the leading source of news and information.

Available software and compatibility....

The CPC664 runs virtually all CPC464 software, giving the owner an instant and enviable choice of software from AMSOFT's extensive range, as well as the products of many independent vendors.

Why disc?

The disc is undoubtedly replacing the cassette as the data and program storage medium for all but the most casual computer user, and since the power and friendliness of the CPC464 quickly transformed 'casual' users into enthusiasts, the '664 will be the natural choice for many.

Thank you Digital Research....

The power of Digital Research's CP/M operating system has not previously been available at such low cost, and we look forward to the innovative software that can be written in an environment which provides over 160K of memory on each side of a disc.

Digital Research's Dr. LOGO has emerged as the most universal educational and teaching medium. Combining the unique user friendliness of 'turtle graphics' with sophisticated processing power, Dr. LOGO is acclaimed as the most comprehensive implementation of LOGO available.

Thank you Amstrad!

Both CP/M and Dr. LOGO are provided free with your CPC664 system.

What's new?

The CPC664 also takes the opportunity to refine and add some features and BASIC commands not previously available on the CPC464 (area fills, dotted line drawing, character copying from the screen, frame flyback synchronisation, switchable cursor, etc.) Programs using these new features will not be downwards compatible to the CPC464, although transportability from the CPC464 upwards to the CPC664 is not compromised in any way.

AMSOF
A division of
AMSTRAD
CONSUMER ELECTRONICS PLC

© Copyright 1985 AMSOF, AMSTRAD Consumer Electronics plc

Neither the whole nor any part of the information contained herein, nor the product described in this manual may be adapted or reproduced in any material form except with the prior written approval of AMSTRAD Consumer Electronics plc ('AMSTRAD').

The product described in this manual and products for use with it are subject to continuous development and improvement. All information of a technical nature and particulars of the product and its use (including the information and particulars in this manual) are given by AMSTRAD in good faith. However, it is acknowledged that there may be errors or omissions in this manual. A list of details of any amendments or revisions to this manual can be obtained by sending a stamped, self addressed envelope to AMSOF Technical Enquiries. We ask that all users take care to submit their reply paid user registration and guarantee cards.

You are also advised to complete and send off your Digital Research User registration card.

AMSOF welcome comments and suggestions relating to the product or this manual.

All correspondence should be addressed to:

AMSOF
Brentwood House,
169 Kings Road,
Brentwood,
Essex CM14 4EF

All maintenance and service on the product must be carried out by AMSOF authorised dealers. Neither AMSOF nor AMSTRAD can accept any liability whatsoever for any loss or damage caused by service or maintenance by unauthorised personnel. This manual is intended only to assist the reader in the use of the product, and therefore AMSOF and AMSTRAD shall not be liable for any loss or damage whatsoever arising from the use of any information or particulars in, or any error or omission in, this manual or any incorrect use of the product.

Dr LOGO and CP/M are trademarks of Digital Research Inc.

Z80 is the trademark of Zilog Inc.

IBM and IBM PC are trademarks of International Business Machines Inc.

AMSDOS, CPC664, and CPC464 are trademarks of AMSTRAD Consumer Electronics plc.

First Published 1985

Compiled by Ivor Spital

Written by Roland Perry, Ivor Spital, William Poel, Cliff Lawson,
with acknowledgements to Locomotive Software Ltd.

Published by AMSOF

Typeset by KAMSET typesetting graphics (Brentwood)

AMSTRAD is a registered trademark of AMSTRAD Consumer Electronics plc.

Unauthorised use of the trademark or word AMSTRAD is strictly forbidden.

IMPORTANT

You must read this....

Installation Notes

1. Always connect the Mains Lead to a 3-pin plug following the instructions contained in part 1 of the Foundation course.
2. Never attempt to connect the system to any Mains Supply other than 220-240V ~ 50Hz.
3. There are no user serviceable parts inside the system. Do not attempt to gain access into the equipment. Refer all servicing to qualified service personnel.
4. To avoid eye-strain, the monitor should be placed as far away as possible from the keyboard and operated in an adequately lit room. The monitor BRIGHTNESS control should be kept to as low a setting as possible.
5. The computer should be placed centrally in front of the monitor, but as far away from the screen as possible. For maximum data reliability, the disc drive section of the computer should NOT be placed directly in front of the monitor, but to the right of it. Do not place the computer close to any source of electrical interference.
6. Always keep disc drives and discs away from magnetic fields.
7. If you are operating a 2-drive system, keep the interconnecting ribbon cable to the 2nd drive, away from Mains Leads.
8. Do not block or cover any ventilation holes.
9. Do not use or store the equipment in excessively hot, cold, damp, or dusty areas.

Operation Notes

(Don't worry if you are a little baffled by the technical jargon in this section; the importance of these warnings will become clearer as you work through this manual.)

1. Never switch the system on or off with a disc in the drive. Doing so will corrupt your disc, losing valuable programs or data.

-
2. Always make back-up (duplicate) copies of discs which contain valuable programs. It is especially important to make a back-up copy of the master CP/M system disc provided with the CPC664. Should you otherwise accidentally lose or corrupt your disc, replacing it could prove expensive.
 3. Make sure that you do not accidentally overwrite your master CP/M system disc, by ensuring that the Write Protect holes on the disc are always open.
 4. If you are operating a 2-drive system, i.e. if you have purchased an additional AMSTRAD FD1, always switch on the 2nd disc drive before switching on the computer.
 5. Never touch the floppy disc surface itself, inside its protective casing.
 6. Do not eject a disc while it is being read from or written to.
 7. Always remember that formatting a disc will erase any previous contents.
 8. The internal disc interface occupies a small portion of the memory that in some cases, was used by commercial writers of cassette based software for the AMSTRAD model CPC464. These cassettes will not operate properly with the CPC664 + Cassette unit. If you have any queries regarding cassette based software compatibility, contact AMSOFT on Brentwood (0277) 230222. Note however, that most major AMSOFT software titles are available on disc for the CPC664.
 9. The licence agreement for your CP/M system disc, (which is electronically serial-number encoded) permits its use on a single computer system only. In particular this means that you are prohibited from giving any other person a disc with YOUR serial-numbered copy of CP/M on it. Carefully read the End User Licence Agreement towards the end of this manual.

CONTENTS

Chapter 1 **Foundation Course**

Setting up
Connecting peripherals
Floppy discs
Keyboard familiarisation
Loading software, and the 'Welcome' program
Introduction to BASIC keywords
Introduction to disc operations
Modes Colours and Graphics
Sound
Introduction to AMSDOS and CP/M

Chapter 2 **Beyond Foundations**

Writing a simple program
Evolution and afterthoughts
Using an array
Introducing a menu
Loading and saving variables to disc

Chapter 3 **Complete List of AMSTRAD CPC664 BASIC Keywords**

Description of notation used
Alphabetical listing of keywords comprising:
 Keyword
 Formal syntax
 Example
 Description
 Special notes (where applicable)
 Associated keywords

Chapter 4

Using Discs and Cassettes

- Backup master disc
- A working SYSTEM/UTILITY disc
- A BASIC only disc
- Turnkey discs and packages
- Configuring discs
- Starting and autostarting a Turnkey CP/M package
- Cassette considerations

Chapter 5

AMSDOS and CP/M

AMSDOS:

- Disc directory
- Changing discs
- AMSDOS filenames and filetypes
- Filename construction, headers and wild cards
- Examples of using AMSDOS commands in a program
- Saving variables and performing a screen dump
- Summary of AMSDOS external commands
- Copying files
- Reference guide to AMSDOS error messages

CP/M:

- CP/M system tracks
- Configuration sector
- Console control codes
- Logging in a disc
- Direct console commands
- Transient commands
- File and disc copying
- System management

Chapter 6

Introduction to LOGO

What is LOGO?

Dr. LOGO procedures

Editing programs and procedures

Operating hints

Summary of Dr. LOGO primitives covering:

- Word and list processing

- Arithmetic operations

- Logical operations

- Variables

- Procedures

- Editing

- Text screen

- Graphic screen

- Turtle Graphics

- Workspace management

- Property lists

- Disc files

- Keyboard and joystick

- Sound

- Flow of control

- Exception handling

- System primitives

- System variables

- System properties

Chapter 7

For Your Reference....

Cursor locations and control code extensions

Interrupts

ASCII and graphics characters

Key references

Sound

Chapter 7 continued....

Error messages
BASIC keywords
Planners
Connections
Printers
Joysticks
Disc organisation
Memory

Chapter 8 At Your Leisure....

General:

- The world of microcomputers
- Hardware and software
- Comparing computers
- Some popular misconceptions
- How a computer deals with your instructions
- The digital world
- Bits and bytes
- The BINARY number system
- The HEXADECIMAL number system

CPC664 Specific functions:

- Character set
- Variables
- Logic
- User defined characters
- Print formatting
- Windows
- Interrupts
- Data
- Sound
- Graphics
- Hardware

Appendices

Appendix 1 End User Program Licence Agreement

Appendix 2 Glossary of Terms

Appendix 3 Some Programs For You....

 Telly tennis

 Electric fencing

 Pontoon

 Bomber

 Amthello

 Raffles

Appendix 4 Index

AMSTRAD CPC664 FOUNDATION COURSE

Part 1: Setting Up....

The CPC664 can be set up with either:

1. The AMSTRAD GT65 Green Tube Monitor
2. The AMSTRAD CTM644 Colour Monitor
3. The AMSTRAD MP2 Modulator/Power Supply and a domestic (UHF) colour TV receiver.

Fitting a Mains Plug

The CPC664 operates from a 220-240Volt~ 50Hz Mains supply. Fit a proper Mains Plug to the Mains Lead of either the GT65, CTM644, or MP2. If a 13 Amp (BS1363) Plug is used, a 5 Amp Fuse must be fitted. The 13 Amp Fuse supplied in a new plug must NOT be used. If any other type of Plug is used, a 5 Amp Fuse must be fitted either in the Plug or Adaptor or at the Distribution Board.

Important

The wires in the Mains Lead are coloured in accordance with the following code:

Blue : Neutral
Brown : Live

As the colours of the wires in the Mains Lead of this apparatus may not correspond with the coloured markings identifying the terminals in your plug, proceed as follows:

The wire which is coloured BLUE must be connected to the terminal which is marked with the letter 'N' or coloured Black.

The wire which is coloured BROWN must be connected to the terminal which is marked with the letter 'L' or coloured Red.

Disconnect the Mains Plug from the Supply Socket when not in use.

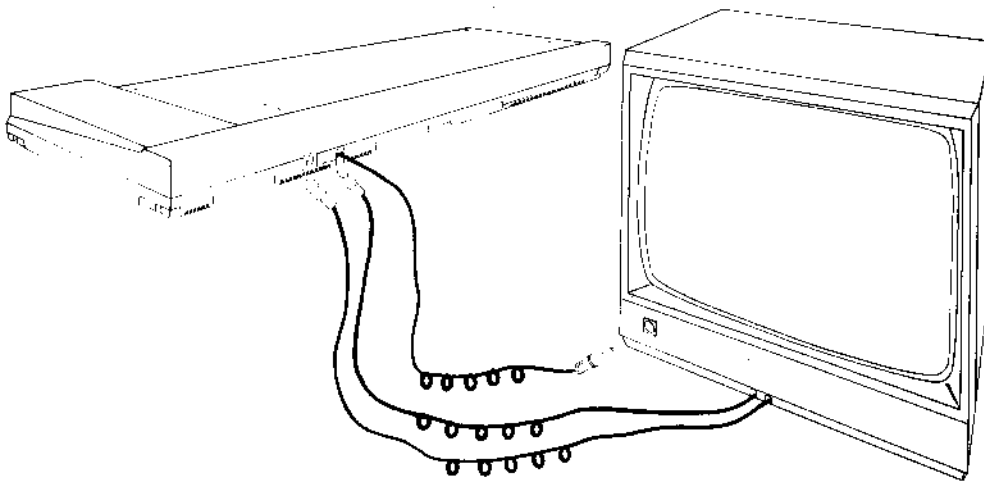
Do not attempt to remove any screws, nor open the casing of the computer, monitor, or Modulator/Power supply unit. Always obey the warning on the rating label which is located underneath the case of the CPC664 and MP2, and on the rear cabinet of the GT65 and CTM644:

WARNING - LIVE PARTS INSIDE. DO NOT REMOVE ANY SCREWS

Connecting the computer to a monitor

(If you are using the CPC664 with the MP2 Modulator/Power supply, skip to the next section.)

1. Make sure that the monitor is not plugged into the Mains supply socket.
2. Connect the lead from the front of the monitor, which is fitted with the larger (6-pin DIN) plug, into the rear socket of the computer marked **MONITOR**.
3. Connect the lead from the front of the monitor, which is fitted with the smaller (5V DC) plug, into the rear socket on the computer marked **5V DC**.
4. Connect the lead from the back of the computer, which is fitted with a small (12V DC) plug, into the socket at the front of the monitor.

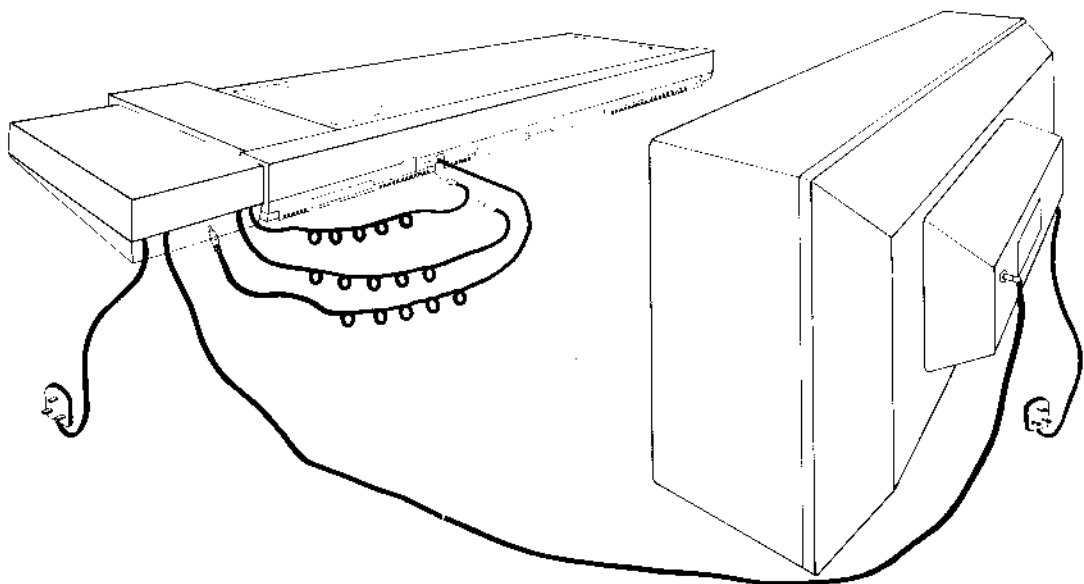


Connecting the computer to an MP2 Modulator/Power supply unit

The MP2 is an additional item that you may wish to purchase if you are currently using your CPC664 computer with the GT65 green tube monitor. The MP2 enables you to use the computer with your domestic colour TV and thereby enjoy the full colour facilities of your CPC664 computer.

The MP2 should be positioned next to the right hand end of the CPC664.

1. Make sure that the MP2 is not plugged into the Mains supply socket.
2. Connect the lead from the MP2 which is fitted with the larger (6-pin DIN) plug, into the rear socket of the computer marked **MONITOR**.
3. Connect the lead from the MP2 which is fitted with the smaller (5V DC) plug, into the rear socket on the computer marked **5V DC**.
4. Connect the lead from the MP2 which is fitted with an aerial plug, into the **AERIAL** or **ANTENNA** socket of your TV set.
5. Connect the lead from the back of the computer, which is fitted with a small (12V DC) plug, into the socket at the rear of the MP2.



Switching on - CPC664 and GT65/CTM644 system

(If you are using the CPC664 with the MP2 Modulator/Power supply, skip to the next section.)

Having connected up the system as shown previously, insert the Mains Plug into the Supply socket and switch on. Press the **POWER** button at the bottom right hand corner of the monitor so that it is set to the IN position. When the button is set to the OUT position, the Mains supply to the system is OFF.

Switch on the computer using the slide switch marked **POWER** at the right hand end.

The red **ON** lamp at the top centre of the keyboard should be illuminated, and the monitor will display the following picture:

```
Amstrad 64K Microcomputer  (v2)
©1984 Amstrad Consumer Electronics plc
and Locomotive Software Ltd.

BASIC 1.1

Ready
■
```

To avoid unnecessary eye-strain, adjust the control marked **BRIGHTNESS** until the display is adequately bright for comfortable viewing, without glare or blurring of the characters on the screen.

The **BRIGHTNESS** control will be found on the lower front panel of the GT65, or at the right hand side of the CTM644.

If you are using the GT65 green monitor, you may need to adjust the **CONTRAST** and **Vertical HOLD** controls on the lower front panel.

The **CONTRAST** should be adjusted to the minimum setting consistent with comfortable viewing.

The **Vertical HOLD** control on the GT65 is marked **V-HOLD**, and should be adjusted so that the display is correctly positioned in the middle of the screen, without 'jitter' or 'roll'.

Switching on - CPC664 and MP2 Modulator/Power supply system

Having connected up the system as shown previously, insert the Mains Plug into the Supply socket.

Switch on the computer using the slide switch marked **POWER** at the right hand end.

The red **ON** lamp at the top centre of the keyboard should be illuminated, and you must now tune in your TV set to receive the signal from the computer.

If you have a TV with push-button channel selection, press a channel button to select a spare or unused channel. Adjust the corresponding tuning control in accordance with the TV set manufacturer's instructions (the signal will be approximately at channel 36 if your TV has a marked tuning scale), until you receive a picture that looks like:

```
Amstrad 64K Microcomputer  (v2)
©1984 Amstrad Consumer Electronics plc
and Locomotive Software Ltd.

BASIC 1.1
Ready
■
```

Tune in the TV set accurately until the clearest picture is seen. The writing will be gold/yellow on a deep blue background.

If your TV has a rotary programme selector knob, turn the tuning knob until the above picture appears and remains perfectly steady. (Again, at approximately channel 36).

Other Connections....

If you wish to connect any other peripherals to the standard system, namely:

- Joystick(s)
- Cassette unit
- Printer
- 2nd disc drive
- External amplifier/speakers
- Expansion device(s)

....details will be found in part 2 of this Foundation course.

Finally, check that you have observed the following warnings given at the beginning of this manual, in the section entitled 'IMPORTANT':

INSTALLATION NOTES 1,2,4,5,6,7,8
OPERATION NOTE 1

AMSTRAD CPC664 FOUNDATION COURSE

Part 2: Connecting your peripherals....

This section explains how the various peripherals, or add-ons, are connected to the CPC664 system. Details concerning the use of these devices will be found in the appropriate sections of this manual.

Joystick

The AMSOFT joystick model JY2 is an additional item that you may wish to purchase if you are using the CPC664 computer with games software which incorporates the facility for joystick control and 'firing'.

Connect the plug fitted to the JY2 lead into the socket marked **JOYSTICK** on the computer. The CPC664 can be used with two joysticks; the second joystick should be plugged into the socket on the first joystick.

The AMSOFT joystick model JY1 is also suitable for use with this computer.

Further information on joysticks will be found later in this manual.

Cassette unit

Programs may be loaded from, or saved to tape instead of disc. The commands which instruct the computer to direct data to and from disc or tape are explained later in this manual.

To connect your cassette unit to the CPC664, you will require the AMSOFT CL1 lead, or any other equivalent standard cassette-interconnecting lead.

Insert the end of the lead with the larger (5-pin DIN) plug, into the socket marked **TAPE** on the computer.

Insert the plug at the end of the Blue cable into the socket on your cassette unit marked **REMOTE** or **REM**.

Insert the plug at the end of the Red cable into the socket on your cassette unit marked **MIC**, **COMPUTER IN**, or **INPUT**.

Insert the plug at the end of the White cable into the socket on your cassette unit marked **EAR**, **COMPUTER OUT**, or **OUTPUT**.

It is important to remember that the successful transfer of data between the CPC664 and cassette is largely dependent upon the correct setting of the LEVEL or VOLUME control on your cassette unit. If you cannot seem to load or save programs properly, experiment with different LEVEL control positions until the optimum setting is found.

Printer

The CPC664 may be used with any Centronics compatible parallel printer. If you intend to connect the AMSTRAD DMP1 to the CPC664, simply use the interconnecting lead provided with the printer.

If you wish to use any other Centronics compatible printer, you will require the AMSOFT PL1 printer interconnecting lead.

Connect the end of the lead which is fitted with the flat edge-connector plug, into the socket marked **PRINTER** at the rear of the computer.

Connect the other end of the lead which is fitted with a Centronics style plug, into the socket at the rear of the printer. If the printer is equipped with security clips at each side of the socket, these may be clipped into the cut-outs at the side of the printer plug.

Details of printer operation will be found later in this manual.

A 2nd disc drive (AMSTRAD FD1)

The AMSTRAD FD1 may be added to the system as a 2nd disc drive. The advantages of a 2-drive system will be particularly apparent to the regular CP/M user, since many programs are configured to run with the library program disc inserted in one drive, and the working data files stored on a disc in the second drive.

Operation under CP/M always requires that a program be loaded from disc (there is no access to the ROM BASIC). Since CP/M allows the use of multiple files by an overlay technique that permits programs that are larger than the RAM memory, the actual library disc may contain so many program files that there is little workspace left for the data.

Thanks to the versatility of the utilities provided with your CPC664 system disc, you can do all necessary file maintenance; copying, erasing etc, on a single disc drive. However a second drive will certainly speed up these processes and reduce the scope for accidents.

To connect the FD1 to the CPC664, you will require the AMSOFT DI2 disc interconnecting lead.

Connect the end of the lead which is fitted with the larger edge-connector plug, into the socket marked **DISC DRIVE 2** at the rear of the computer.

Connect the other end of the lead which is fitted with a smaller plug, into the socket at the rear of the FD1 disc drive.

DON'T FORGET - Before connecting or disconnecting the 2nd disc drive, make sure that any discs are removed from both drives, and that the system is switched off. If connections are altered while the system is on, it is likely that the program in the computer's memory will be corrupted. Always save any valuable programs before meddling with connections!

When the FD1 is connected to the CPC664, **FIRST** switch on the FD1 using the slide switch on the rear panel of the disc drive, **THEN** switch on the CPC664 using the slide switch at the right hand side of the computer. Both the green and red indicators on the front panel of the FD1 should be illuminated. The 2-drive system will then be ready to operate.

Details of 2-drive operation will be found later in this manual.

External amplifier/speakers

The CPC664 may be connected to a stereo amplifier and speakers to enjoy the full 3-channel capabilities of the computer.

The input lead to your stereo amplifier should be terminated with a 3.5mm stereo jack plug, which should be inserted into the socket marked **STEREO** on the computer.

The connections to the jack plug are:

- Plug tip - Left channel
- Plug inner ring - Right channel
- Plug rear shaft - Ground (Common)

The CPC664 will provide a fixed voltage signal out of the **STEREO** socket, and you should therefore use the controls on the amplifier itself to regulate volume, balance and tone.

High impedance headphones will also operate with the CPC664, however the volume will not be adjustable by the **VOLUME** control on the computer. Low impedance headphones, such as those usually used with stereo systems, will not operate directly plugged into the computer.

Details concerning the directing of sound to the required output channel will be found later in this manual.

Expansion devices

Expansion devices such as serial interfaces, modems, light pens, ROMs etc may be connected to the CPC664, using the socket marked **EXPANSION** at the rear of the computer.

The AMSOFT speech synthesiser/amplifier model SSA2 may also be connected to this socket.

Details of connections to the **EXPANSION** socket will be found in the chapter entitled 'For your reference....'.

Finally, check that you have observed the following warnings given at the beginning of this manual, in the section entitled 'IMPORTANT':

INSTALLATION NOTES 6,7

OPERATION NOTES 4,8

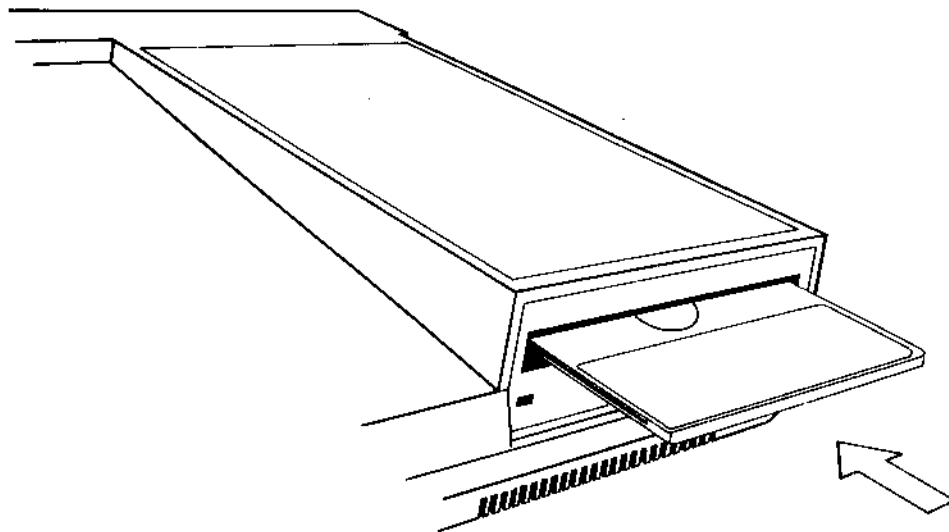
AMSTRAD CPC664 FOUNDATION COURSE

Part 3: About Discs....

The AMSTRAD CPC664 uses 3 inch compact floppy discs. We strongly recommend that for reliable data-to-disc transfer, you use only AMSOFT CF2 compact floppy discs. Discs made by leading manufacturers however, may also be used.

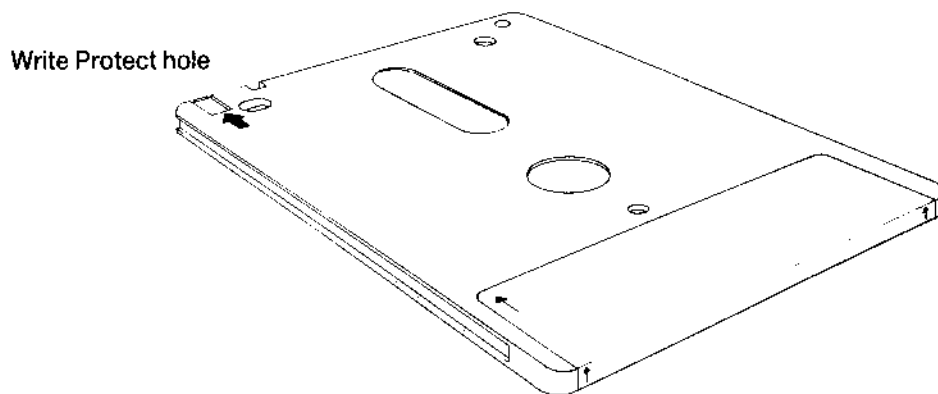
Insertion

Each side of a disc may be used separately. A disc should be inserted with its label facing outward from the drive, and with the side that you wish to use face up:



Write Protection

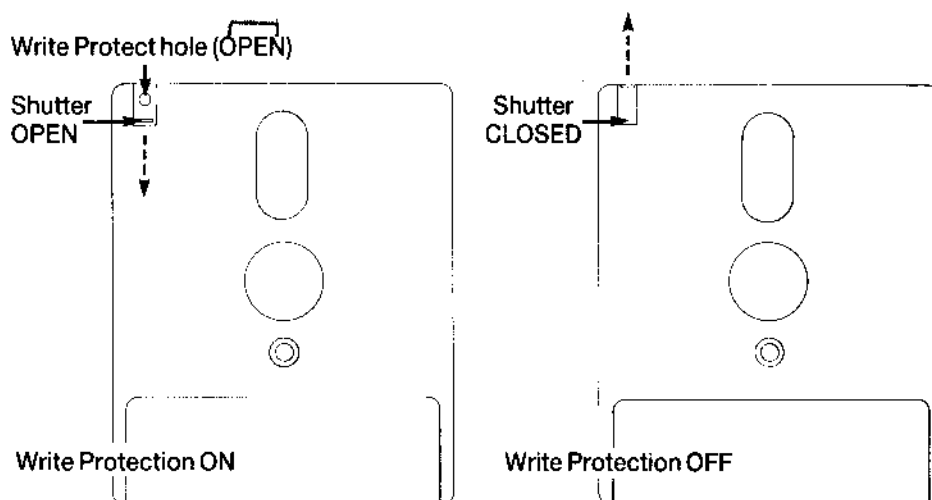
In the left hand corner of each side of a blank disc, you will see an arrow pointing to a small shuttered hole. This is called the Write Protect hole, and facilitates protection against erasure or 'overwriting':



When the hole is closed, data can be 'written' onto the disc by the computer. When the hole is open however, the disc will not allow data to be written onto it, thus enabling you to avoid accidental erasure of valuable programs.

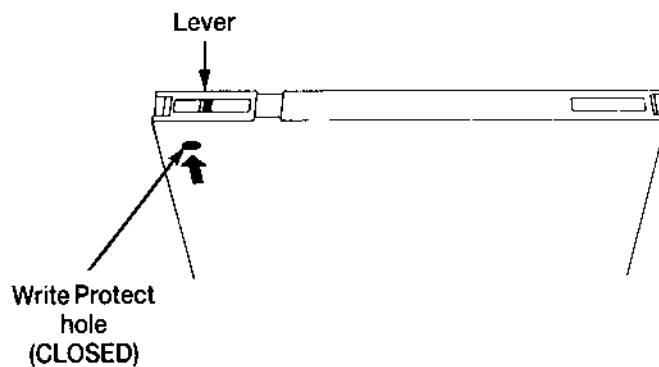
Various compact floppy disc manufacturers employ different mechanisms for opening and closing the Write Protect hole. The operation may be carried out on the AMSOFT CF2 compact floppy disc as follows:

To open the Write Protect hole, slide the small shutter located at the left hand corner of the disc, and the hole will be opened:

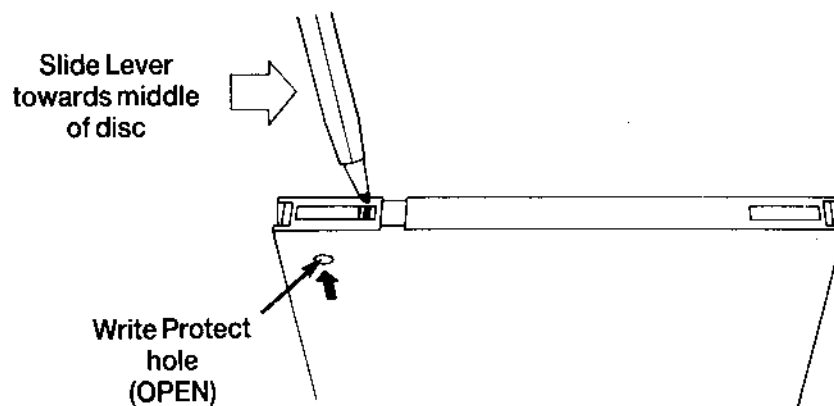


To close the Write protect hole, simply slide the shutter to its closed position.

Some other compact floppy discs employ a small plastic lever located in a slot at the left hand corner:



To open the Write Protect hole on this type of disc, slide the lever towards the middle of the disc, using the tip of a ball-point pen or similar object:



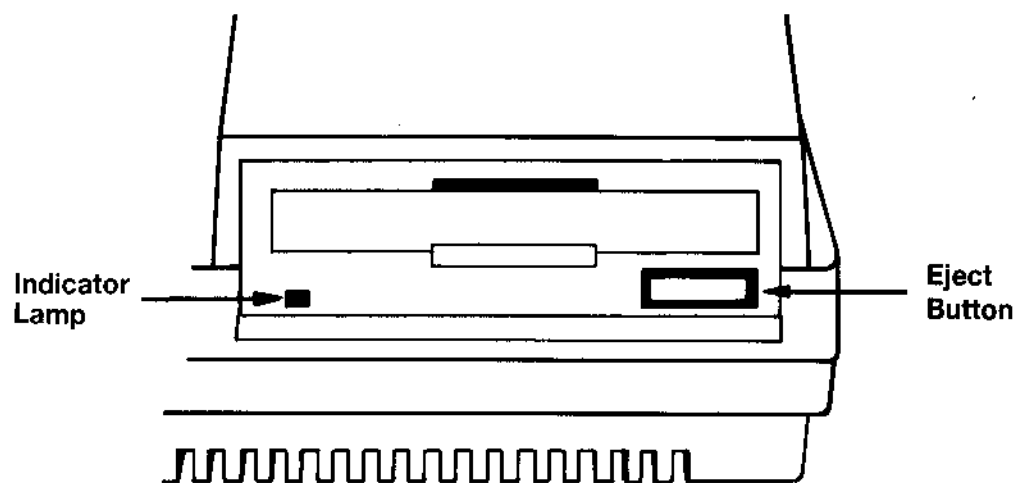
Note that regardless of the method employed to open and close the Write Protect hole, opening the hole in all cases facilitates protection against overwriting.

IMPORTANT

Always ensure that the Write Protect holes on your master CP/M system disc are open.

When Your Disc is in

At the front of the computer's disc drive, you will see a red indicator lamp, and a push button for Eject:



Indicator Lamp

This indicates that data is being read from, or written to the disc.

If a 2nd disc drive is connected, the red indicator on the 2nd disc drive (Drive B) will illuminate constantly. It will extinguish when the main disc drive within the computer (Drive A) is reading or writing to disc.

Eject Button

Pressing in the Eject button allows you to remove your disc from the drive.

Finally, check that you have observed the following warnings given at the beginning of this manual, in the section entitled 'IMPORTANT':

OPERATION NOTES 1, 3, 4, 5, 6

AMSTRAD CPC664 FOUNDATION COURSE

Part 4: Getting Started....

Before we start loading software and saving programs to disc, let's get familiar with some of the keys on the computer. Those of you who are experienced in using computers may skip this section.

With the computer switched on and the opening message on the screen, we're going to find out what the various keys do....

LARGE ARROW KEYS

The four large arrow keys (next to the **[COPY]** key), are called the 'cursor keys'. The keys move the position of the cursor (the small solid block) on the screen.

Press each of the cursor keys in turn, and you will see the cursor move about the screen.

[ENTER]

There are two **[ENTER]** keys. Either of these keys enter the information that you have typed into the computer. After the **[ENTER]** key is pressed, a new line is started on the screen. Each instruction that you type into the computer should be followed by pressing the **[ENTER]** key.

From now on, we will show **[ENTER]** as meaning press the **[ENTER]** key after each instruction or program line.

[DEL]

This key is used to delete a character to the left of the cursor on the screen (for example a letter or a number) which is not required.

Type in a b c d and you will see that the letter d is positioned to the left of the cursor. If you decide that you do not want the letter d, press **[DEL]** once and you will see the d removed. If you press **[DEL]** and continue to hold it down, the letters a b c will also be removed.

[SHIFT]

There are two **[SHIFT]** keys. If you press either of these and hold it down whilst typing a character, a capital letter or upper case symbol will appear on the screen.

Type in the letter **e** then hold down the **[SHIFT]** key and type in the letter **e** again. On the screen you will see:

e E

Now type in a few spaces by holding down the space bar. Try the following using the number keys which are on the top line of the keyboard, above the letter keys. Type in the number 2, then hold down the **[SHIFT]** key and type in the number 2 again. On the screen you will see:

2"

You can now see what happens when the **[SHIFT]** key is held down whilst pressing a character key. Experiment by pressing any of the character keys, either on their own, or together with the **[SHIFT]** key.

[CAPS LOCK]

This has a similar operation to **[SHIFT]** except that you only have to press it once. From then on each letter that you type in will be in capitals, although the number keys will not be shifted.

Press **[CAPS LOCK]** once, then type in:

abcdefghijklmnopqrstuvwxyz

On the screen you will see:

ABCDEFGHIJKLMNOPQRSTUVWXYZ

You will notice that although all the letters are shifted to capitals, the numbers have not been shifted to symbols. If you wish to type in a shifted symbol while **[CAPS LOCK]** is in operation, simply hold down the **[SHIFT]** key before pressing a number key. Type in the following while holding down the **[SHIFT]** key:

abcdefghijklmnopqrstuvwxyz

On the screen you will see:

ABCDEFGHIJKLMNOPQRSTUVWXYZ!"#\$%&

If you wish to return to small (lower case) characters again, press the **[CAPS LOCK]** key once again.

If you wish to type in capital letters and shifted upper case symbols without having to constantly hold down the **[SHIFT]** key, this can be carried out as follows:

Holding down the **[CTRL]** key, then press the **[CAPS LOCK]** key once. This performs the function of a 'SHIFT LOCK'. Now type in:

abc def 123456

On the screen you will see:

ABCDEF!"#\$%&

Note that it is still possible to type in numbers while **[CTRL]** and **[CAPS LOCK]** are in operation, by using the number keys to the right of the main keyboard.

Holding down the **[CTRL]** key and pressing **[CAPS LOCK]** once, will return you to the mode that you were previously in, (i.e. either lower case or capital letters-only). If you have returned to the capital letters-only mode, simply press **[CAPS LOCK]** once again to return to the lower case mode.

[CLR]

This key is used to clear a character within the cursor.

Type in **ABCDEF GH**. The cursor will be positioned to the right of the last letter typed (H). Now press the cursor left key **◀** four times. The cursor will have moved four places to the left, and will be superimposed over the top of the letter E.

Notice how the letter E is still visible within the cursor. Press the **[CLR]** key once and you will see that the letter E has been cleared and the letters **FGH** have each moved one space to the left with the letter F now appearing within the cursor. Now press the **[CLR]** key again and hold it down. You will see how the letter F is cleared followed by the letters G and H.

[ESC]

This key is used to **[ESC]**ape from a function that the computer is in the process of carrying out. Pressing the **[ESC]** key once will cause the computer to temporarily pause in its function, and will continue again if any other key is pressed.

Pressing the **[ESC]** key twice will cause the computer to completely **[ESC]**ape from the function which it is carrying out. The computer is then ready for you to type in some more instructions.

Important

When you reach the right hand edge of the screen by entering 40 characters on a line, the next character will automatically appear on the following line at the left edge of the screen. This means that you should **NOT** press **[ENTER]** as those of you accustomed to typewriters might press a carriage return towards the right edge of a page.

The computer does this automatically for you, and will react to an unwanted **[ENTER]** by printing an error message - usually a `Syntax error`, either there and then, or when the program is run.

Syntax Error

If the message: `Syntax error` appears on the screen, the computer is telling you that it does not understand an instruction that you have entered.

For example type in:

```
printt [ENTER]
```

On the screen you will see the message:

```
Syntax error
```

The message appears because the computer has not understood the instruction:
`printt`

If you type a mistake in the line of a program, such as:

```
10 printt "abc" [ENTER]
```

The `Syntax error` message will not appear until the instruction is processed by the computer when the program is run.

Type in:

```
run [ENTER]
```

(This command tells the computer to carry out the program that you have just typed into the memory.)

On the screen you will see:

```
Syntax error in 10  
10 printt "abc"
```

This message tells you in which line the error has occurred, and displays the program line, together with the editing cursor so that you can correct the mistake.

Press the cursor-right key $\diamond$ until the cursor is over the `t` in `print t`. Now press the **[CLR]** key to remove the unwanted `t`, then press the **[ENTER]** key to enter the corrected line into the computer.

Now type in:

```
run [ENTER]
```

....and you will see that the computer has accepted the instruction, and has printed:

```
abc
```

Finally, check that you have observed the following warnings given at the beginning of this manual, in the section entitled 'IMPORTANT':

INSTALLATION NOTES 4,5
OPERATION NOTE 1

AMSTRAD CPC664 FOUNDATION COURSE

Part 5: Loading Software and games....

Welcome to those of you who skipped here from the previous section!

As a quick demonstration of the speed of loading disc software, switch on the system, then insert the master CP/M disc supplied, into the drive with Side 1 uppermost.

Type in:

```
run "rointime.dem" [ENTER]
```

After a few seconds the program will have loaded into the memory. Answer the question on the screen as to whether you are using a green monitor; (Type Y for Yes, or N for No), and you will then see a continuous demonstration of the game 'Roland in Time' on the screen. It may even persuade you to go out and buy a copy of the game!

When you have finished watching the demonstration, you may 'escape' from the program by simultaneously holding down the [CTRL] and [SHIFT] keys then pressing the [ESC] key. This has the effect of completely resetting the computer, and may be used whenever you wish to start afresh. (You do not need to remove any disc from the drive when resetting the computer in this way).

If the program hasn't loaded, study any error message on the screen to see where you went wrong:

```
Drive A: disc missing
Retry, Ignore or Cancel?
```

....means that you have either not inserted your disc correctly, or, if you have a 2-drive system, that you have inserted it into Drive B.

```
ROINTIME.DEM not found
```

....means that you have either inserted the wrong disc, (or the wrong side of the disc), or you have not carefully and precisely typed in the name, ROINTIME.DEM

```
Bad command
```

....means that you have probably mis-typed ROINTIME.DEM by introducing an unwanted space or punctuation mark.

Type mismatch

....means that you have omitted the quotation marks "

Syntax error

....means that you mistyped the word run

Drive A: read fail
Retry, Ignore or Cancel?

....means that the computer has failed to read data from your disc. Check that you have inserted the correct disc and press **R** to Retry. This message will appear if ever you corrupt your disc by switching the system on or off while the disc is in the drive.

Once you have learned how to make back-up copies of discs, always do so for valuable programs, and especially for your master CP/M system disc.

Loading AMSOFT software and the WELCOME program

Having hopefully whetted your appetite for the lighter side of computing, let's load a game....

Insert your software disc in the drive, and type in:

run "disc" [ENTER]

After a few seconds, your game will be loaded and ready to play.

Try typing in: run "disc" with side 1 of your master CP/M system disc inserted into the drive, and you will see and hear the CPC664 continuous 'Welcome' demonstration.

When you have finished watching 'Welcome', reset the computer using the [CTRL] [SHIFT] and [ESC] keys.

The above instruction (run "disc") will load most of the AMSOFT range of disc software, but you may come across the odd occasion where you have to type in something else. In all cases however, the loading instructions will be found on the label of the software disc, which should always be carefully followed.

Finally, check that you have observed the following warnings given at the beginning of this manual, in the section entitled 'IMPORTANT':

INSTALLATION NOTE 6
OPERATION NOTES 1,5,6

AMSTRAD CPC664 FOUNDATION COURSE

Part 6: Let's Compute....

So far, we know what we must and mustn't do with the computer, and how to set it up and connect peripherals. We know what some of the keys on the computer do, and how to load software. Now we'll look at some of the instructions that you can type in to the computer to make things happen....

Like you or I, the computer can only understand instructions in a language that it knows, and that language is called BASIC, (short for Beginners' All-purpose Symbolic Instruction Code). The words in BASIC's vocabulary are called 'keywords' and each of them tell the computer to perform a specific function. All languages must conform to the rules of grammar, and BASIC is no exception. Here, grammar is referred to as 'Syntax', and the computer will always be kind enough to tell you if you've made a `Syntax error`!

An introduction to AMSTRAD BASIC keywords

In the chapter entitled 'Complete list of AMSTRAD CPC664 BASIC keywords', you will find a description of the keywords found in AMSTRAD BASIC. We will introduce some of the more commonly used BASIC keywords in this section.

CLS

To clear the screen, type in:

```
cls [ENTER]
```

You will notice that the screen clears and the word `Ready` with the cursor **■** will appear at the top left of the screen.

Note that you can use upper case (CAPITAL) or lower case (small) letters to enter any BASIC keyword into the computer.

PRINT

This is used whenever you want characters, words or figures in a program to be printed. Type in the following instruction line:

```
print "hello" [ENTER]
```

On the screen you will see:

```
hello
```

The quotation marks "" are used to tell the computer what should be printed. `hello` appeared on the screen as soon as the [ENTER] key was pressed. Type in:

```
cls [ENTER]
```

....to clear the screen.

RUN

The previous example showed a single instruction line. However, as soon as the [ENTER] key was pressed, the instruction was carried out then forgotten. It is possible to store a series of instructions in the computer to be carried out in a specified order. This is achieved by writing a 'program'. The sort of BASIC instructions that you write in a program are the same as just shown, but in front of each instruction line, a line number is typed in. If there is more than one line in the program, these line numbers tell the computer the order in which to carry out or 'run' the program. When [ENTER] is pressed the line is stored in the memory until the program is run. Now type in:

```
10 print "hello" [ENTER]
```

Notice that when [ENTER] was pressed, `hello` was NOT printed on the screen, but instead was entered into the computer's memory as a one-line program. To carry out that program, the word `run` must be used. Type in:

```
run [ENTER]
```

You will now see `hello` printed on the screen.

Note that instead of continually typing in: `print`, you can use the ? question-mark symbol, for example:

```
10 ? "hello" [ENTER]
```

LIST

After a program has been stored in the memory, it is possible to check what has been typed in by 'listing' the program. Type in:

```
list [ENTER]
```

On the screen you will see:

```
10 PRINT "hello"
```

....which is the program stored in the memory.

Notice how the word **PRINT** is now in capitals. This means that the computer has accepted **PRINT** as a known BASIC keyword.

Type in: **cls [ENTER]** to clear the screen. Note that although the screen is cleared when you type in: **cls [ENTER]**, your program is not erased from the computer's memory.

GOTO

The **GOTO** keyword tells the computer to go from one line to another in order to either miss out a number of lines or to form a loop. Type in:

```
10 print "hello" [ENTER]
20 goto 10 [ENTER]
```

Now type:

```
run [ENTER]
```

....and you will see **hello** printed continuously on the screen, one under another on the left side. The reason for this, is that line **20** of the program is telling the computer to go to line **10** and carry on processing the program from there.

To pause the running of this program, press **[ESC]** once. To start it again, press any other key. To stop it running so that other instructions can be typed in, press **[ESC]** twice.

Now type in:

```
cls [ENTER]
```

....to clear the screen.

To see the word **hello** printed continuously on each line, one next to another filling the whole of the screen, type in the previous program but with a semi-colon **;** after the quotation marks **"**

Type in:

```
10 print "hello"; [ENTER]
20 goto 10 [ENTER]
run [ENTER]
```

Note that the semicolon **;** tells the computer to print the next group of characters immediately following the previous one, (unless the next group of characters is too large to fit on the same line).

Escape from this program by pressing **[ESC]** twice. Now type in line 10 again, but this time, use a comma , instead of a semicolon ;

```
10 print "hello", [ENTER]
run [ENTER]
```

You will now see that the comma , has told the computer to print the next group of characters 13 columns away from the first group of characters. This feature is useful for displaying information in separate columns. Note however, that if the number of characters in a group exceeds 12, the next group of characters will be displaced forwards by another 13 columns, so as to always maintain a space between columns.

This figure of 13 columns is adjustable by use of the **ZONE** command, described later in this manual.

Again, to escape from this program, press **[ESC]** twice. To clear the computer's memory completely, hold down the **[CTRL]** **[SHIFT]** and **[ESC]** keys in that order, and the computer will reset.

INPUT

This command is used to let the computer know that it is expecting something to be typed in, for example, the answer to a question.

Type the following:

```
10 input "how old are you";age [ENTER]
20 print "you look younger than";age;
   "years old." [ENTER]
run [ENTER]
```

On the screen you will see:

```
what is your age?
```

Type in your age then **[ENTER]**. If your age was 18, the screen would then show:

```
you look younger than 18 years old.
```

This example shows the use of the **input** command and a number variable. The word **age** was put into the memory at the end of line 10 so that the computer would associate the word **age** with whatever numbers were typed in and would print these numbers where the word **age** is on line 20. Although we used the expression **age** in the above for the variable, we could have just as easily used a letter, for example **b**.

Reset the computer to clear the memory, ([CTRL] [SHIFT] and [ESC] keys). If you had wanted an input made up of any characters, (letters or letters and numbers), the dollar sign \$ must be used at the end of the variable. This type of variable is called a 'string variable'.

Type in the following program: (Note that in line 20 you must put a space after the o in hello and before the m in my).

```
10 input "what is your name";name$ [ENTER]
20 print "hello ";name$;" my name is Roland" [ENTER]
run [ENTER]
```

On the screen you will see:

```
What is your name?
```

Type in your name then [ENTER]

If the name that you entered was Fred, you will see on the screen:

```
 Hello Fred my name is Roland
```

Although we used name\$ in the above example for the name string variable, we could have just as easily used a letter, for example a\$. Now we will combine the above 2 examples into one program.

Reset the computer by pressing [CTRL] [SHIFT] and [ESC]. Type in the following:

```
5  cls [ENTER]
10 input "what is your name";a$ [ENTER]
20 input "what is your age";b [ENTER]
30 print "I must say ";a$;" you dont
    look";b;"years old" [ENTER]
run [ENTER]
```

In this program we have used 2 variables, a\$ for the name and b for the age. On the screen you will see:

```
what is your name?
```

Now type in your name (e.g. Fred), then [ENTER].

You will then be asked:

```
what is your age?
```

Now type in your age (e.g. 18), then [ENTER].

If your name were Fred and your age 18, on the screen you will see:

```
I must say Fred you dont look 18 years old
```

Editing a Program

If any of the lines in the program had been typed incorrectly, resulting in a **Syntax error** or other error message, it would be possible to edit that line, rather than type it out again. To demonstrate this, let's type in the previous program incorrectly:

```
5  cls [ENTER]
10 input "what is you name";a$ [ENTER]
20 input "what is your age";b [ENTER]
30 print "I must say";a$;" you dont
    look";b;"years old" [ENTER]
```

There are 3 mistakes in the above program:

In line 5 we typed in `cls` instead of `cls`.

In line 10 we typed in `you` instead of `your`.

In line 30 we forgot to put a space between `say` and the quotation marks "

There are 3 main methods of editing a program. The first is to simply type in the new line again. When a line is retyped and entered, it replaces the same numbered line currently in the memory.

Secondly, there is the editing cursor method.

Lastly, there is the copy cursor method.

Editing Cursor Method

To correct the mistake in line 5

Type:

```
edit 5 [ENTER]
```

Line 5 will then appear under line 30 with the cursor superimposed over the `c` in `cls`

To edit out the extra `s` in `cls`, press the cursor right key `→` until the cursor appears over the last `s`, then press the **[CLR]** key. You will see the `s` disappear.

Now press **[ENTER]** and line 5 will now be corrected in the memory. Type in:

```
list [ENTER]
```

....to check line 5 is now correct.

The **AUTO** command, described later in this manual, may be used to edit a number of successive lines in a similar manner to that described in the Editing Cursor Method.

Copy Cursor Method

The copy cursor is another cursor (in addition to the one already on the screen) which comes into view when you hold down **[SHIFT]** and press one of the cursor keys. It then detaches itself from the main cursor and can then be moved around the screen independently.

To correct the mistakes in line 10 and 30, hold down the **[SHIFT]** key then press the cursor up key $\uparrow$ until the copy cursor is positioned over the very beginning of line 10. You will notice the main cursor has not moved, so there are now two cursors on the screen. Now press the **[COPY]** key until the copy cursor is positioned over the space between **you** and **name**. You will notice that line 10 is being re-written on the last line and the main cursor stops at the same place as the copy cursor. Now type in the letter **r**. This will appear on the bottom line only.

The main cursor has moved but the copy cursor stayed where it was. Now press the **[COPY]** key until the whole of line 10 is copied. Press **[ENTER]** and this new line 10 will be stored in the memory. The copy cursor disappears and the main cursor positions itself under the new line 10. To correct the second mistake, hold down **[SHIFT]** and press the cursor up key $\uparrow$ until the copy cursor appears over the very beginning of line 30.

Press **[COPY]** until the copy cursor is positioned over the quotation marks next to **say**. Now press the space bar once. A space will be inserted on the bottom line. Hold down the **[COPY]** key until the whole of line 30 is copied, then press **[ENTER]**.

You can now list the program to check that it is corrected in the memory by typing in:

```
list [ENTER]
```

NOTE: To move the cursor quickly (during editing) to the left or right hand end of a line, hold down the **[CTRL]** key then press the left $\leftarrow$ or right $\rightarrow$ cursor key once.

Now reset the computer by pressing **[CTRL]** **[SHIFT]** and **[ESC]** keys.

IF

The **IF** and **THEN** commands ask the computer to test for a specified condition, then take action depending upon the result of that test. For example, in the instruction:

```
if 1+1=2 then print "correct" [ENTER]
```

...the computer will test for the specified condition, and then take action accordingly.

The keyword **ELSE** can be used to inform the **IF THEN** command as to what alternative action the computer should take if the tested condition is false, for example:

```
if 1+1=0 then print "correct" else print "wrong" [ENTER]
```

We will now extend the previous program with the use of the **if** and **then** commands.

Type in the following: Notice that we have added two symbols. **<** means less than, and is next to the **M** key. **>** means greater than, and is next to the **<** (less than) key.

```
5  cls [ENTER]
10 input "what is your name";a$ [ENTER]
20 input "what is your age";age [ENTER]
30 if age < 13 then 60 [ENTER]
40 if age < 20 then 70 [ENTER]
50 if age > 19 then 80 [ENTER]
60 print "So ";a$," you are not quite a
   teenager at";age;"years old":end [ENTER]
70 print "So ";a$," you are a teenager
   at";age;"years old":end [ENTER]
80 print "Oh well ";a$," you are no
   longer a teenager at";age;"years old" [ENTER]
```

To check this program is correct, type:

```
list [ENTER]
```

Now type in:

```
run [ENTER]
```

Now answer the questions prompted by the computer and see what happens.

You can now see what effect the **IF** and **THEN** commands have in a program. We have also added the word **END** at the end of lines **60** and **70**. The keyword **END** is used literally to end the running of a program. If **END** wasn't there in line **60**, the program would continue to run, and carry out lines **70** and **80**.

Likewise, if **END** wasn't there in line 70, the program would continue to run, and carry out line 80. The colon **:** before the word **END** separates it from the previous instruction. Colons **:** can be used to separate two or more instructions on one program line. We have also added line 5 to clear the screen at the start of the program. We will do this from now on in the following programs to make things look neater.

Reset the computer by pressing **[CTRL]** **[SHIFT]** and **[ESC]** keys.

FOR and NEXT

The **FOR** and **NEXT** commands are used when you want to carry out a specified operation a number of times. The instructions for the specified operation must be enclosed by the **FOR NEXT** loop.

Type in:

```
5 cls [ENTER]
10 for a=1 to 10 [ENTER]
20 print "operation number";a [ENTER]
30 next a [ENTER]
run [ENTER]
```

You will see that the operation in line 20 has been carried out 10 times, as instructed by the **FOR** command in line 10. Note also that the value of the variable **a** is increased by 1 each time.

The keyword **STEP** may be used to inform the **FOR NEXT** command how much the variable should be 'stepped' per operation. For example, change line 10 to:

```
10 for a=10 to 50 step 5 [ENTER]
run [ENTER]
```

Negative steps may also be used, for example:

```
10 for a=100 to 0 step -10 [ENTER]
run [ENTER]
```

REM

REM is short for **REMark**. The instruction tells the computer to ignore anything that follows it on the line. Hence you can use **REMark**s to inform you of, for example, the title of a program, or the use of a variable, as follows:

```
10 REM Zap the invaders [ENTER]
20 L=5 :REM number of lives [ENTER]
```

The single quote mark ' (which can be typed by holding down [SHIFT] and pressing the 7 key) can be used as a substitute for : REM. For example:

```
10 'Zap the invaders [ENTER]
20 L=5 'number of lives [ENTER]
```

GOSUB

If there are a set of instructions within a program which are to be carried out a number of times, these instructions need not be typed in repeatedly every time they are needed in the program; instead, they can be made into a 'sub-routine' which when required, can be called into action by the command GOSUB followed by the line number. The end of the GOSUB-routine is marked by typing in the instruction RETURN. At this point the computer will return to the instruction that immediately followed the GOSUB command which it had just obeyed.

(The following two programs will not actually 'do' anything other than print words on the screen, and you therefore need not bother typing them in. They have been included to demonstrate the way that sub-routines can perform repeated tasks.)

For example, in the following program:

```
10 MODE 2 [ENTER]
20 PRINT "This old man he played one" [ENTER]
30 PRINT "He played knick-knack on my drum" [ENTER]
40 PRINT "With a knick-knack paddy wack" [ENTER]
50 PRINT "Give a dog a bone" [ENTER]
60 PRINT "This old man came rolling home" [ENTER]
70 PRINT [ENTER]
80 PRINT "This old man he played two" [ENTER]
90 PRINT "He played knick-knack on my shoe" [ENTER]
100 PRINT "With a knick-knack paddy wack" [ENTER]
110 PRINT "Give a dog a bone" [ENTER]
120 PRINT "This old man came rolling home" [ENTER]
130 PRINT [ENTER]
140 PRINT "This old man he played three" [ENTER]
150 PRINT "He played knick-knack on my knee" [ENTER]
180 PRINT "With a knick-knack paddy wack" [ENTER]
190 PRINT "Give a dog a bone" [ENTER]
200 PRINT "This old man came rolling home" [ENTER]
210 PRINT [ENTER]
run [ENTER]
```

...you can see a number of lines which have to be repeated at various points in the program, for example the chorus at line 180. Let's make that chorus into a sub-routine and add the instruction RETURN at the end. Then, we'll call the sub-routine using the command GOSUB 180 whenever we want to use it. The program now looks like this:

```
10 MODE 2 [ENTER]
20 PRINT "This old man he played one" [ENTER]
30 PRINT "He played knick-knack on my drum" [ENTER]
40 GOSUB 180 [ENTER]
80 PRINT "This old man he played two" [ENTER]
90 PRINT "He played knick-knack on my shoe" [ENTER]
100 GOSUB 180 [ENTER]
140 PRINT "This old man he played three" [ENTER]
150 PRINT "He played knick-knack on my knee" [ENTER]
160 GOSUB 180 [ENTER]
170 END [ENTER]
180 PRINT "With a knick-knack paddy wack" [ENTER]
190 PRINT "Give a dog a bone" [ENTER]
200 PRINT "This old man came rolling home" [ENTER]
210 PRINT [ENTER]
220 RETURN [ENTER]
run [ENTER]
```

See how much tedious typing we'd have saved ourselves? Well designed sub-routines are a principal part of computing. They lead to 'structured' programs, and develop good programming habits.

Always bear in mind when writing sub-routines, that you do not necessarily have to 'jump into' the sub-routine at the same point, i.e. its beginning. A sub-routine written from lines 500 to 800 can be called by: GOSUB 500, or GOSUB 640, or GOSUB 790.

Note in the above program, that the instruction END is used in line 170. Otherwise the program would naturally continue after line 160, and would carry out line 180, which is NOT required unless called by GOSUB.

Simple Arithmetic

Your computer can be used as a calculator quite easily.

To understand this, carry out the following examples. We will use the ? symbol instead of typing `print` in this section. The answer will be printed as soon as the [ENTER] key is pressed.

Addition

(use [SHIFT] and ; keys for plus)

Type:

```
? 3 + 3 [ENTER]
6
```

(Note that you do NOT type in the equals sign =)

Type:

```
? 8 + 4 [ENTER]
12
```

Subtraction

(use unshifted = key for minus)

Type:

```
? 4 - 3 [ENTER]
1
```

Type:

```
? 8 - 4 [ENTER]
4
```

Multiplication

(Use [SHIFT] and : keys for multiply (* means x))

Type:

```
? 3 * 3 [ENTER]
9
```

Type:

```
? 8 * 4 [ENTER]
32
```

Division

(use unshifted ? key for divide (/ means ÷)

Type:

? 3 / 3 [ENTER]

1

Type:

? 8 / 4 [ENTER]

2

Integer Division

(use \ for division with removal of the remainder)

Type:

? 10 \ 6 [ENTER]

1

Type:

? 20 \ 3 [ENTER]

6

Modulus

(use MOD to obtain the remainder portion after integer division)

Type:

? 10 MOD 4

2

Type:

? 9 MOD 3

0

Square Root

To find the square root of a number, use `sqr ( )`. The number that you want the square root of should be typed inside the brackets.

Type:

?sqr(16) [ENTER] (this means $\sqrt{16}$)

4

Type:

?sqr (100) [ENTER]
10

Exponentiation

(use unshifted $\wedge$ key for exponentiation)

Exponentiation is when a number is raised to a power of another. For example 3 squared (3^2), 3 cubed (3^3) etc.

Type:

?3 $\uparrow$ 3 [ENTER] (this means 3^3)
27

Type:

?8 $\uparrow$ 4 [ENTER] (this means 8^4)
4096

Cube Root

You can quite easily calculate cube roots by using a similar method to the last example.

To find the cube root of 27 ($\sqrt[3]{27}$)

Type:

?27 $\uparrow$ (1/3) [ENTER]
3

To find the cube root of 125

Type:

?125 $\uparrow$ (1/3) [ENTER]
5

Mixed Calculations

(+, -, *, /)

Mixed calculations are understood by the computer, but they are calculated within certain priorities.

First priority is given to multiplication and division, then addition and subtraction. These priorities apply only to calculations containing only these four operations.

If the calculation was:

$$3+7-2*7/4$$

You may think this would be calculated as:

$$\begin{aligned} &3+7-2*7/4 \\ &= 8*7/4 \\ &= 56/4 \\ &= 14 \end{aligned}$$

In fact it is calculated as:

$$\begin{aligned} &3+7-2*7/4 \\ &= 3+7-14/4 \\ &= 3+7-3.5 \\ &= 10-3.5 \\ &= 6.5 \end{aligned}$$

Prove this by typing in this calculation as it is written:

Type:

$$\begin{aligned} &?3+7-2*7/4 \text{ [ENTER]} \\ &6.5 \end{aligned}$$

You can change the way the computer calculated this by adding brackets. The computer will deal with the calculation inside brackets prior to the multiplication etc, outside the brackets. Prove this by typing in the calculation including brackets.

Type:

$$\begin{aligned} &?(3+7-2)*7/4 \text{ [ENTER]} \\ &14 \end{aligned}$$

The priority of ALL mathematical operators is as follows:

↑	Exponentiation
MOD	Modulus
-	Unary minus (declares a number as negative)
* and /	Multiplication and division
\	Integer division
+ and -	Addition and Subtraction

Further Exponents

If you want to use very large or very small numbers in calculations, it is sometimes useful to use scientific notation. The letter E is used for the exponent of numbers to the base 10. You may use either lower case e or upper case E.

For example 300 is the same as 3×10^2 . In scientific notation, this is 3E2. Similarly, 0.03 is the same as 3×10^{-2} . In scientific notation this is 3E-2. Try the following examples.

You can type in:

```
?30*10 [ENTER]  
300
```

or you can type:

```
?3E1*1E1 [ENTER]  
300
```

```
?3000*1000 [ENTER] ....or.... ?3E3*1E3 [ENTER]  
3000000
```

```
?3000*0.001 [ENTER] ....or.... ?3E3*1E-3 [ENTER]  
3
```


AMSTRAD CPC664 FOUNDATION COURSE

Part 7: Save It....

Now that you've exercised your fingers by typing in a few instructions, you'll probably want to know how to save a program from the computer onto disc, and how to load it back from disc into the computer.

Even if you are familiar with cassette saving and loading, it is worth pointing out some important information which must be observed when dealing with disc program files.

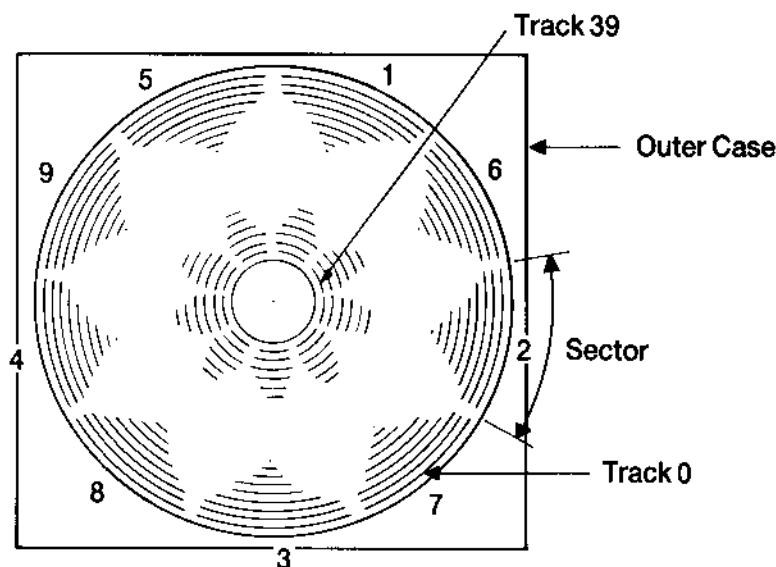
The 2 most apparent differences will be that firstly, a new blank disc cannot simply be taken out of its wrapper and recorded onto, as is the case with cassettes. A new disc must be first be 'formatted', and this process will be explained in a moment.

The other point worth mentioning here, is the importance of correctly 'naming' disc files. Cassette filenames generally conform to very loose standards, varying greatly in length, being at times omitted. Not so with discs. Disc filenames must conform strictly to CP/M standards, and will be explained later in this section.

Formatting discs for use

Before writing any data onto a new blank disc, the disc itself must first be formatted. Formatting can be likened to building a series of shelves and dividers onto a disc prior to the storage of information on those shelves; in other words, laying down an organised framework around which data can be put in or taken out.

Formatting divides the disc into 360 distinctly separate areas:



There are 40 tracks from the outside of the disc (Track 0), to the inside (Track 39), and the circumference of the disc is divided into 9 sectors.

Each track in a sector can store up to 512 bytes of data; hence the total available space on each side of a disc is 180Kbytes.

First Steps Using the CP/M Disc

To prepare a new blank disc for reading and writing your own programs onto, you will need to format using the CP/M disc.

Switch the system on, and insert the CP/M disc supplied, into the drive.

If you are operating 2 disc drives, always insert the CP/M disc into the main disc drive within the computer (Drive A.)

Type in:

```
lcpm [ENTER]
```

(You will find the bar symbol | by holding down [SHIFT] and pressing the @ key.)

After a few seconds you will see the following message on the screen:

```
CP/M 2.2 - Amstrad Consumer Electronics plc  
A>
```

This is a 'Sign on' message indicating that the operating system is under the control of CP/M.

The displayed A> on the screen is a prompt, (similar to Ready during normal BASIC operation) indicating that the computer is awaiting your instructions.

Once you are operating CP/M, you cannot enter BASIC commands into the computer, as these will not be understood.

If for example, you type in the BASIC command:

```
cls [ENTER]
```

The Computer will return your entry, together with a question mark:

```
CLS?
```

indicating that it does not understand your command.

To briefly look at some of the CP/M commands, type in:

`dir [ENTER]`

On the screen you will see a directory of CP/M and utility COMmands, one of which is `format`. Type in:

`format [ENTER]`

On the screen you will see:

Please insert disc to be formatted into drive A
then press any key.

Remove the CP/M disc and insert your new blank disc, then press any key (any grey key). Formatting will start, commencing at Track 0 and ending at track 39, after which you will be asked on the screen:

Do you want to format another disc (Y/N):

If you wish to format the other side of your blank disc, or format another disc, type Y (for Yes) and you will receive the initial message once again.

The formatting process can be repeated any number of times until you answer the repeat question N (for No), whereupon the system will ask you to:

Please insert a CP/M system disc into drive A
then press any key:

After doing so, the computer will return you to the direct CP/M mode, and will await your next command. Other CP/M commands will be dealt with later in this manual but for now, having learnt to format with CP/M, reset the computer using **[CTRL][SHIFT][ESC]**.

Always keep a master copy of your CP/M disc in a safe place, as it is literally the key to your system. Later in this manual, you will be shown how to make a 'working copy' of your CP/M disc, so that you can keep your master copy safely locked away!

BEWARE

FORMATTING A PREVIOUSLY RECORDED DISC WILL ERASE ITS CONTENTS.

You will not be able to format a disc which has its write protection hole open. Attempting to do so will result in the message on the screen:

Drive A: disc is write protected
Retry, Ignore or Cancel?

Press **C** to cancel, then follow the instructions on the screen. The format mode will then be abandoned.

Now that we have a formatted blank disc (or two), we can start to manipulate BASIC programs to and from disc.

Saving a Program in Memory onto Disc

Having typed a program into the computer's memory, save it onto disc by typing in:

```
save "filename" [ENTER]
```

Note that the naming of the program is obligatory.

A filename on disc consists of 2 parts (fields). The first part is obligatory and can contain up to 8 characters. Letters and numbers may be used but NO spaces or punctuation marks. This first field usually contains the name of the program.

The second field is optional. You can use up to 3 characters, but again no spaces or punctuation. The 2 fields are separated by a dot .

If you do not specify a second field, the system will automatically label it with a token of its own, such as .BAS for BASIC files or .BIN for binary (machine code) files.

As an example of saving to disc, write a short program into the memory, insert a formatted disc, then type in:

```
save "example" [ENTER]
```

After a few seconds, the prompt **Ready** will appear on the screen, and the program will have been saved onto disc. (If not, check any error message on the screen to establish whether you either forgot to insert your disc into the correct drive, forgot to close the write protect hole, or mistyped the command.)

Catalog

After saving the above program, type in:

```
cat [ENTER]
```

On the screen you will see:

```
Drive A: user 0
EXAMPLE.BAS 1K
168K free
```

The filename will be displayed, including any specified or token second field, together with the file length (to the nearest higher Kbyte). The amount of free space on the disc will also be displayed.

Loading from Disc

Programs may be loaded from disc then run, using the commands:

```
load "filename" [ENTER]
run [ENTER]
```

....or they may be run directly using the command:

```
run "filename" [ENTER]
```

Note that protected programs may be run directly only.

| A and | B

If you are operating an additional disc drive, you may specify which Drive (A or B) that you require a function to be performed on by typing in:

```
| a [ENTER]
```

.....OR.....

```
| b [ENTER]
```

....before issuing the SAVE, CAT, or LOAD commands.

Copying Programs from Disc to Disc

Using the commands already learnt in this section, it can be seen that disc to disc program copying is performed simply by loading the program into the memory from the original (source) disc, removing the source disc, and saving the program onto the new (destination) disc.

To save a program from disc to disc using 2 disc drives, you may prefer to insert your source disc into, for example, Drive B, and your destination disc into Drive A. To copy a program in this way, type in:

```
lb [ENTER]
load "filename" [ENTER]
la [ENTER]
save "filename" [ENTER]
```

There are four ways in which files may be **SAVED** by the CPC664. In addition to ordinary BASIC file saving, by:

```
save "filename" [ENTER]
```

....there are three alternative methods, for more specialised purposes. .

```
save "filename",a [ENTER]
```

Adding the suffix **,a** instructs the computer to save the program or data in the form of an ASCII text file. This method of saving data applies to files created by wordprocessors and other applications programs, and its use will be further discussed as applications are encountered.

```
save "filename",p [ENTER]
```

Adding the suffix **,p** tells the computer to protect the data so that the program cannot be **LIST**ed after **LOAD**ing it, or **RUN**ning it then stopping its execution using the **[ESC]** key function.

Programs saved in this way can only be run directly, using the commands:

```
run "filename" [ENTER]
```

....or....

```
chain "filename" [ENTER]
```

If you anticipate wanting to edit or alter the program, you should also keep a copy for yourself in unprotected form, i.e. without the **,p** suffix.

```
save "filename",b, <starting address> , <length in bytes>
                                [, <optional entry point>] [ENTER]
```

This option allows you to perform a binary save where a complete block of data in the computer's RAM is stored onto disc exactly as it occurs in the memory. It is necessary to instruct the computer where the section of memory you need to save starts, how long it is, and if required, the memory address at which to start execution should the file be run as a program.

This binary save feature allows data from the screen memory to be stored directly onto disc in the form of a screen dump. The contents of the screen will be saved exactly as it is seen, using the command:

```
save "scrndump",b,49152,16384 [ENTER]
```

....where 49152 is the starting address of the screen memory, and 16384 is the length of the screen memory that you wish to save.

To call it back onto the screen, type in:

```
load "scrndump" [ENTER]
```

More information on using the system to manipulate program files between discs (and cassette), will be found later in this manual.

Finally, check that you have observed the following warnings given at the beginning of this manual, in the section entitled 'IMPORTANT':

INSTALLATION NOTES 5,6,7

OPERATION NOTES 1,2,3,4,5,6,7,9

AMSTRAD CPC664 FOUNDATION COURSE

Part 8: Understanding Modes, Colours and Graphics....

The Amstrad CPC664 Colour Personal Computer has three modes of screen display operation: Mode 0, Mode 1, and Mode 2.

When the computer is first switched on, it is automatically in Mode 1.

To understand the different modes, switch on the computer and press the number **1** key. Hold it down until two lines are full of 1's. If you now count the number of 1's on a line, you will see that there are 40. This means that in Mode 1, there are 40 columns. Press **[ENTER]** - you will get a **Syntax error** message, but don't worry, this is just a quick way of getting back to the **Ready** message that tells you the computer is waiting for your next instruction.

Now type in:

```
mode 0 [ENTER]
```

You will see that the characters on the screen are now larger. Press the number **1** key again and hold it down until two lines are full of 1's. If you count the number of 1's on a line, you will see there are 20. This means that in Mode 0, there are 20 columns. Press **[ENTER]** again.

Now type in:

```
mode 2 [ENTER]
```

You will see that this is the smallest mode, and if you again type in a line of 1's, you will count 80. This means that in Mode 2 there are 80 columns.

To recap:

Mode 0 = 20 columns
Mode 1 = 40 columns
Mode 2 = 80 columns

Finally, press **[ENTER]** once again.

Colours

There is a choice of 27 colours. These are shown on a green monitor (GT65) as various shades of green. If you purchased the GT65 monitor, you can buy the AMSTRAD MP2 Modulator/Power supply in order to use the computer's colour facilities on your domestic colour T.V.

In Mode 0, up to 16 of the 27 available colours can be put onto the screen at any time.

In Mode 1, up to 4 of the 27 colours can be put onto the screen at any time.

In Mode 2, up to 2 of the 27 colours can be put onto the screen at any time.

You are able to change the colour of the **BORDER**, the **PAPER** (the area where the characters can appear) or the **PEN** (the character itself), all independently of each other.

The 27 colours available are listed in Table 1, each with their **INK** colour reference number.

For convenience, this table also appears on the panel at the top right hand side of the computer.

MASTER COLOUR CHART

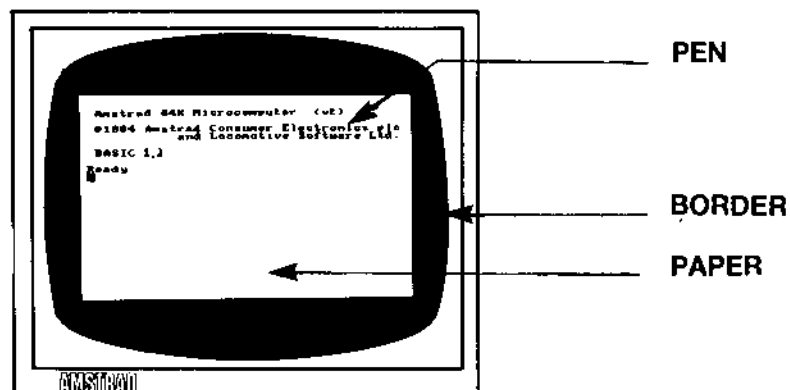
Ink No.	Colour/Ink	Ink No.	Colour/Ink	Ink No.	Colour/Ink
0	Black	9	Green	18	Bright Green
1	Blue	10	Cyan	19	Sea Green
2	Bright Blue	11	Sky Blue	20	Bright Cyan
3	Red	12	Yellow	21	Lime Green
4	Magenta	13	White	22	Pastel Green
5	Mauve	14	Pastel Blue	23	Pastel Cyan
6	Bright Red	15	Orange	24	Bright Yellow
7	Purple	16	Pink	25	Pastel Yellow
8	Bright Magenta	17	Pastel Magenta	26	Bright White

Table 1: The INK numbers and colours

As explained earlier, when the computer is first switched on, it is in Mode 1. To return to Mode 1 from a different mode, type in:

```
mode 1 [ENTER]
```

The screen display



The **BORDER** is the area surrounding the **PAPER**. (Note that when the computer is first switched on, the **BORDER** and **PAPER** are both blue). The characters on the screen can only appear inside the border. The **PAPER** is the background area behind a character, while the **PEN** writes the character itself.

Now we will explain how the colours that you see on the screen are selected, and how you can change these to your own choice.

When you first switch on, or reset the computer, the **BORDER** is always set to colour number 1. Look up number 1 on the master colour chart, and you will see that colour number 1 is blue. The colour of the border can be changed by using the command: **BORDER** followed by the colour number. To change the border to white, type in:

```
border 13 [ENTER]
```

So far, so good. Now for the tricky bit....

When you first switch on, or reset the computer, **PAPER** number 0, and **PEN** number 1 are always automatically selected. This does NOT mean that you look up numbers 0 and 1 on the master colour chart and away you go....

The important thing to remember is that 0 and 1 are **PAPER** and **PEN** numbers; they are NOT ink colour numbers. To understand this, imagine having 4 pens on your desk numbered 0, 1, 2, and 3, and being able to fill each of these pens with any colour of your choice out of 27 bottles of ink, numbered 0 to 26. It can be seen therefore, that pen number 1 is not necessarily always the same colour; as it can be filled with a different ink, in fact you could fill each of the 4 pens with the same ink.

So it is with the computer. Using the **PEN** and **INK** commands, you can select the pen number, and then the ink colour for that pen.

Remembering that we are operating in Mode 1 (40 columns), look at Table 2 below, and you will see from the first and third columns, that **PEN** number 1 corresponds to **INK** colour number 24. Now look up **INK** number 24 on the master colour chart (Table 1) and you will see that the colour listed is bright yellow, i.e. the colour of the characters on the screen when you first switch on.

DEFAULT SETTINGS

Paper/Pen No.	Ink Colour Mode 0	Ink Colour Mode 1	Ink Colour Mode 2
0	1	1	1
1	24	24	24
2	20	20	1
3	6	6	24
4	26	1	1
5	0	24	24
6	2	20	1
7	8	6	24
8	10	1	1
9	12	24	24
10	14	20	1
11	16	6	24
12	18	1	1
13	22	24	24
14	Flashing 1,24	20	1
15	Flashing 16,11	6	24

Table 2: PAPER/PEN/MODE/INK reference

Chapter 1 Page 49

Now let's change to a new paper. When a new paper is selected, the previous background colour behind the characters will NOT change because that colour was 'printed' by a different PAPER. To see this, type in:

```
paper 2 [ENTER]
```

Once again use Tables 1 and 2 to see why the background colour for PAPER number 2 is bright cyan. Change it to black by typing in:

```
ink 2,0 [ENTER]
```

On the screen now, we have characters written by PEN numbers 1 and 3, on a background of PAPER numbers 0 and 2. INKs can be changed in a PEN or PAPER that you are not currently using. For example, type in:

```
ink 1,2 [ENTER]
```

....which changes the colour of all the previous characters typed in with PEN number 1.

Type in:

```
cls [ENTER]
```

....to clear the screen.

It should now be possible for you to instruct the computer to return to its original colours (blue border and background with bright yellow characters) using the BORDER, PAPER, PEN, and INK commands. See if you can do so. If you can't, then reset the computer using the [CTRL] [SHIFT] and [ESC] keys.

Flashing Colours

It is possible to make the colour of the characters flash between one colour and another. This can be achieved by adding an extra colour number to the INK command of the PEN.

To see the characters on the screen flashing between bright white and bright red, reset the computer using [CTRL] [SHIFT] [ESC], and type in:

```
ink 1,26,6 [ENTER]
```

In this case, 1 is the PEN number, while 26 is the colour bright white, and 6 is the alternate colour, bright red.

It is also possible to make the colour of the PAPER behind the characters flash between one colour and another. This can be achieved by adding an extra colour number to the INK command for the PAPER.

To see the PAPER flashing between green and bright yellow behind the characters, type in:

```
ink 0,9,24 [ENTER]
```

In this case 0 is the PAPER number, while 9 is the colour green, and 24 is the alternate colour bright yellow.

Now reset the computer using [CTRL] [SHIFT] [ESC]

Note from Table 2 that in mode 0, two of the PENs (numbers 14 and 15), together with two of the PAPERS (numbers 14 and 15) are default flashing colours. In other words, their INK commands have been pre-programmed with an extra colour parameter.

Type in the following:

```
mode 0 [ENTER]  
pen 15 [ENTER]
```

on the screen you will see the word Ready flashing between sky blue and pink.

Now type in:

```
paper 14 [ENTER]  
cls [ENTER]
```

You will now see that in addition to the word Ready flashing between sky blue and pink, the background PAPER is also flashing between yellow and blue.

PEN and PAPER numbers 14 and 15 may be re-programmed using the INK command to other flashing colours, or to one steady colour.

Finally, it is possible to make the BORDER flash between two colours by adding an extra colour number to the BORDER command. Type in:

```
border 6,9 [ENTER]
```

You will now see that the BORDER is flashing between bright red and green. Note that the border may be set to any one, or pair of the 27 colours, regardless of whether you are operating in mode 0, 1, or 2.

Now reset the computer using [CTRL] [SHIFT] [ESC]

For further demonstration of the colours available, type in the following program, then run it.

```
10 MODE 0 [ENTER]
20 rate=600: REM sets speed of program [ENTER]
30 FOR b=0 TO 26 [ENTER]
40 LOCATE 3,12 [ENTER]
50 BORDER b [ENTER]
60 PRINT "border colour";b [ENTER]
70 FOR t=1 TO rate [ENTER]
80 NEXT t,b [ENTER]
90 CLG [ENTER]
100 FOR p=0 TO 15 [ENTER]
110 PAPER p [ENTER]
120 PRINT "paper";p [ENTER]
130 FOR n=0 TO 15 [ENTER]
140 PEN n [ENTER]
150 PRINT "pen";n [ENTER]
160 NEXT n [ENTER]
170 FOR t=1 TO rate*2 [ENTER]
180 NEXT t,p [ENTER]
190 MODE 1 [ENTER]
200 BORDER 1 [ENTER]
210 PAPER 0 [ENTER]
220 PEN 1 [ENTER]
230 INK 0,1 [ENTER]
240 INK 1,24 [ENTER]
run[ENTER]
```

IMPORTANT

In the above program, and in later chapters and listings in this manual, BASIC keywords will appear in upper case (CAPITAL) letters. This is how keywords appear when a program is LISTed by the computer. In general it is preferable that you type instructions or programs using lower case (small) letters, since it will help you spot typing mistakes when LISTing the program - (because the mis-typed BASIC keyword will NOT be converted to upper case).

For the remainder of this Foundation course, we will list programs in both upper and lower case, so that you get accustomed to this aspect of operation.

A variable's name, such as x or a \$, will NOT be converted to upper case when the program is LISTed although the computer will recognise the name regardless of whether it appears in upper or lower case in the program.

Attention

From this point in the manual, you will not be instructed to press the [ENTER] key after each line. Therefore it is assumed that you will do it automatically.

Graphics

There are a number of character symbols in the computer's memory. To print any one of these, we use the keyword:

```
chr$( )
```

Inside the brackets should be the symbol number, which is in the range from 32 to 255.

Reset the computer, **[CTRL] [SHIFT] [ESC]**, then type in:

```
print chr$(250)
```

Don't forget to press **[ENTER]**. On the screen you will see character number 250, which is a man walking to the right.

To see all the characters and symbols appear on the screen together with their associated numbers, type in the following program, once again remembering to press **[ENTER]** after each line.

```
10 for n=32 to 255
20 print n;chr$(n);
30 next n
run
```

For your reference, the range of characters together with their respective numbers appear in the chapter entitled 'For your reference....'.

LOCATE

This command is used to reposition the character cursor to a specified part of the screen. Unless changed by the locate command, the character cursor starts at the top left corner of the screen, which corresponds to x,y co-ordinates 1,1 (x is the horizontal position and y is the vertical position). In mode 1 there are 40 columns and 25 lines. Therefore, to position a character in the centre of the top line in mode 1, we would use 20,1 as the x,y co-ordinates.

To see this, type in: (remember to **[ENTER]** each line)

mode 1screen clears, cursor moves to top left.

```
10 locate 20,1
20 print chr$(250)
run
```

Just to prove that this is on the top line, type in:

```
border 0
```

The **BORDER** will now be black and you will see the man at the middle of the top line of the screen.

In mode 0, there are only 20 columns, but the same 25 lines. If you now type in:

```
mode 0  
run
```

...you will see that the man now appears at the top right corner of the screen. This happens because the x co-ordinate 20, is the last column in mode 0.

In mode 2, there are 80 columns and 25 lines. Using the same program, you will probably be able to guess where the man will appear. Type in:

```
mode 2  
run
```

Return to mode 1 by typing in:

```
mode 1
```

Now experiment for yourself, modifying the `locate` and `chr$( )` numbers to position various characters anywhere on the screen. Just for example, type in:

```
locate 20,12:print chr$(240)
```

You will see an arrow in the centre of the screen. Note that in this instruction:

20 was the horizontal (x) co-ordinate (in the range 1 to 40)

12 was the vertical (y) co-ordinate (in the range 1 to 25)

240 was the character symbol number (in the range 32 to 255)

To get the character symbol 250 to be repeated across the screen, type in the following program:

```
10 CLS  
20 FOR x=1 TO 39  
30 LOCATE x,20  
50 PRINT CHR$(250)  
60 NEXT x  
70 GOTO 10  
run
```

Press **[ESC]** key twice to break.

In order to remove the previous character from the screen before printing the next character, type in:

```
50 print " ";chr$(250)
```

(This new line 50 automatically replaces the previously typed in line 50.)

Now type in:

```
run
```

FRAME

To improve the movement of the character across the screen, add the following line:

```
40 frame
```

The **FRAME** command synchronises the movement of objects on the screen to the display frame scanning frequency. If that's a bit technical for you, just remember that the command should be used whenever you want to move characters or graphics around the screen smoothly.

This program can be further enhanced to improve the movement by adding some delay loops and by using a different returning character symbol.

Type in:

```
list
```

Now add the following lines to the program:

```
70 FOR n=1 TO 300:NEXT n
80 FOR x=39 TO 1 STEP -1
90 LOCATE x,20
100 FRAME
110 PRINT CHR$(251);" "
120 NEXT x
130 FOR n=1 TO 300:NEXT n
140 GOTO 20
run
```

PLOT

Unlike the `LOCATE` command, `PLOT` may be used to determine the position of the graphics cursor, using pixel co-ordinates. (A pixel is the smallest possible segment of the screen).

Note that the graphics cursor is not visible and is different from the character cursor.

There are 640 horizontal pixels by 400 vertical pixels. The x,y co-ordinates are positioned with respect to the bottom left corner of the screen, which has x,y co-ordinates of 0,0. Unlike the `LOCATE` command used for characters, the co-ordinates do not differ between modes 0,1, or 2.

To see this, first reset the computer using `[CTRL][SHIFT][ESC]`, then type in:

```
plot 320,200
```

A small dot will appear in the centre of the screen.

Now change the mode by typing in:

```
mode 0  
plot 320,200
```

You will see the dot is still in the centre but is now larger. Change the mode again and type in the same command to see the effect in mode 2:

```
mode 2  
plot 320,200
```

The dot is still in the centre, but is now much smaller.

Plot several dots over the screen in various modes, in order to accustom yourself with this command. When you have finished, return to mode 1 and clear the screen by typing in:

```
mode 1
```

DRAW

First reset the computer using `[CTRL][SHIFT][ESC]`. The `DRAW` command draws a line from the current graphics cursor position. To see this in more detail, draw a rectangle on the screen by using the following program.

We start by repositioning the graphics cursor with a `PLOT` command, then `DRAW`ing a line from the graphics cursor position, up towards the top left corner, then from there to the right corner etc, etc. Type in:

```
5 cls
10 plot 10,10
20 draw 10,390
30 draw 630,390
40 draw 630,10
50 draw 10,10
60 goto 60
run
```

Press **[ESC]** twice to break from this program.

(Notice line 60 of this program; the computer is told to 'loop' around line 60 again and again, until you 'break' from the program by pressing **[ESC]** twice. This sort of instruction is useful if you do not want the computer to automatically 'break' at the end of a program and display the **Ready** prompt that usually appears on the screen.)

Now add the following lines to the program, to draw a second rectangle inside the first. Type in:

```
60 plot 20,20
70 draw 20,380
80 draw 620,380
90 draw 620,20
100 draw 20,20
110 goto 110
run
```

Again, press **[ESC]** twice to break from this program.

MOVE

The **MOVE** command operates in a similar manner to **PLOT**, in that the graphics cursor is moved to the position specified by the x,y co-ordinates; however the pixel (dot) at the new graphics cursor location is **NOT** plotted.

Type in:

```
cls
move 639,399
```

Although we can see no sign of it on the screen, we have moved the graphics cursor to the top right corner.

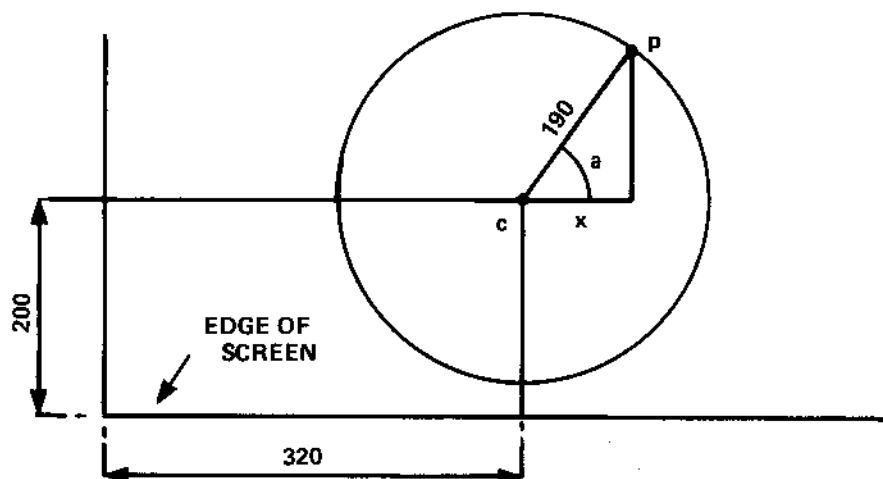
Let's prove it by drawing a line from that position to the centre of the screen, by typing in:

```
draw 320,200
```

Circles

Circles can either be plotted or drawn. One method of forming a circle is to plot the x,y, co-ordinates of each point on the circumference of a circle. Refer to the diagram below and you will see that point 'p' on the circumference can be plotted using x and y co-ordinates. These are:

$$x = 190 * \cos(a)$$
$$y = 190 * \sin(a)$$



Plotting the points of a circle.

In previous programs we have plotted points with respect to the bottom left corner of the screen. If we wanted to position a circle in the centre of the screen we would have to plot the centre of the circle at co-ordinates 320,200 then position all points of the circle relative to the centre position, by adding on the centre position co-ordinates.

A program to plot a circle would then be like this. Type in:

```
new
10 CLS
20 DEG
30 FOR a=1 TO 360
40 MOVE 320,200
50 DRAW 320+190*COS(a),200+190*SIN(a)
60 NEXT
run
```

Note the use of the keyword **NEW** before typing in the program above. This tells the computer to clear any program in the memory (in a similar manner to **[CTRL][SHIFT][ESC]**). However the screen itself is not cleared.

The radius of the circle can be reduced by lowering the 190 figure (190 refers to pixels).

To see the effect of the circle being plotted differently (in radians), delete line 20 from the program by typing in:

```
20
```

To see a solid circle drawn by lines from the centre, edit line 50, replacing the word **plot** with the word **draw**. (Line 50 will then be):

```
50 draw 320+190*cos(a),200+190*sin(a)
```

Try this with and without line 20 again.

You will note that line 60 of this program is **NEXT** instead of **NEXT a**. It is permissible to simply type '**NEXT**' on its own; the computer will work out which **FOR** expression the **NEXT** is to be associated with. In programs where there are numerous **FOR** and **NEXT** loops however, you may wish to add the variable's name after the word **NEXT** in order to identify the **NEXT** statement when studying the program.

ORIGIN

In the previous program we used the **MOVE** command to establish the centre of a circle, then added the x,y co-ordinates to this centre position. Instead of adding these centre co-ordinates to the point plotted, we can use the **ORIGIN** command. This will position each of the x,y co-ordinates relative to the **ORIGIN**. To see this type in:

```
new
10 cls
20 for a=1 to 360
30 origin 320,200
40 plot 190*cos(a),190*sin(a)
50 next
run
```

To plot four smaller circles on the screen, type in the following program:

```
new
10 CLS
20 FOR a=1 TO 360
30 ORIGIN 196,282
40 PLOT 50*COS(a),50*SIN(a)
50 ORIGIN 442,282
60 PLOT 50*COS(a),50*SIN(a)
70 ORIGIN 196,116
80 PLOT 50*COS(a),50*SIN(a)
90 ORIGIN 442,116
100 PLOT 50*COS(a),50*SIN(a)
110 NEXT
run
```

To see a different way of creating a circle, type in the following program:

```
new
10 MODE 1
20 ORIGIN 320,200
30 DEG
40 MOVE 0,190
50 FOR a=0 TO 360 STEP 10
60 DRAW 190*SIN(a),190*COS(a)
70 NEXT
run
```

This time a line is **DRAWn** from co-ordinate to co-ordinate around the circumference of the circle. Note how the circle is drawn much quicker than it is plotted.

Once again observe the effect of removing the **DEG** command, by deleting line 30, then **RUN**ning the program again.

FILL

The **FILL** command is used to fill an area of the screen which is enclosed by drawn or edge of screen/graphics window boundaries.

Reset the computer, **[CTRL][SHIFT][ESC]**, then type in:

```
new
10 cls
20 move 20,20
30 draw 620,20
40 draw 310,380
50 draw 20,20
run
```

On the screen you will see a triangle. Move the graphics cursor to the centre of the screen by typing in:

```
move 320,200
```

Using the keyword **FILL** followed by a pen number, for example 3, we will now **FILL** the screen using the specified pen, from the current graphics cursor position (centre screen) to the drawn boundaries. Type in:

```
fill 3
```

Now move the graphics cursor outside the triangle by typing in:

```
move 0,0
```

See what happens when you type in:

```
fill 2
```

The computer has used pen number 2 to **FILL** the area bounded by the drawn lines and by the edges of the screen.

Now alter the program by typing in the following lines, and see what happens:

```
50 draw 50,50
60 move 320,200
70 fill 3
run
```

You will note that any gaps in the drawn boundaries let the ink from the pen 'seep' through!

This point is further demonstrated by **FILL**ing first a plotted circle, then a drawn circle. Type in:

```
new
10 CLS
20 FOR a=1 TO 360
30 ORIGIN 320,200
40 PLOT 190*COS(a),190*SIN(a)
50 NEXT
60 MOVE -188,0
70 FILL 3
run
```

Now try:

```
new
10 MODE 1
20 ORIGIN 320,200
30 DEG
40 MOVE 0,190
50 FOR d=0 TO 360 STEP 10
60 DRAW 190*SIN(d),190*COS(d)
70 NEXT
80 MOVE -188,0
90 FILL 3
run
```

We can make the outline of the circle to be filled invisible, by setting the pen ink to the same colour as the paper ink. Add:

```
45 GRAPHICS PEN 2:INK 2,1
run
```

The **GRAPHICS PEN** command selects the pen to be used for drawing graphics on the screen. The **INK** command then specifies the ink colour for that pen, which in this case, is the same as for the paper (i.e. colour number 1).

Further details....

For a more comprehensive guide to graphics on the CPC664, see the section 'Graphically speaking' in the chapter entitled 'At your leisure....'.

To conclude this section, here are a few graphics demonstration programs which incorporate a lot of the programming commands and keywords that you should now understand. Each of the programs draws continuous patterns on the screen.

```
10 BORDER 0:GRAPHICS PEN 1
20 m=CINT(RND*2):MODE m
30 i1=RND*26:i2=RND*26
40 IF ABS(i1-i2)<10 THEN 30
50 INK 0,i1:INK 1,i2
60 s=RND*5+3:ORIGIN 320,-100
70 FOR x= -1000 TO 0 STEP s
80 MOVE 0,0:DRAW x,300:DRAW 0,600
90 MOVE 0,0:DRAW -x,300:DRAW 0,600
100 NEXT:FOR t=1 TO 2000:NEXT:GOTO 20
run
```

```

10 MODE 1:BORDER 0:PAPER 0
20 GRAPHICS PEN 2:INK 0,0:i=14
30 EVERY 2200 GOSUB 150
40 flag=0:CLG
50 INK 2,14+RND*12
60 b%=RND*5+1
70 c%=RND*5+1
80 ORIGIN 320,200
90 FOR a=0 TO 1000 STEP PI/30
100 x%=100*COS(a)
110 MOVE x%,y%
120 DRAW 200*COS(a/b%),200*SIN(a/c%)
130 IF flag=1 THEN 40
140 NEXT
150 flag=1:RETURN
run

```

```

10 MODE 1:BORDER 0:DEG
20 PRINT "Please wait"
30 FOR n=1 TO 3
40 INK 0,0:INK 1,26:INK 2,6:INK 3,18
50 IF n=1 THEN sa=120
60 IF n=2 THEN sa=135
70 IF n=3 THEN sa=150
80 IF n=1 THEN ORIGIN 0,-50,0,640,0,400 ELSE ORIGIN 0,0,0,640,0,400
90 DIM cx(5),cy(5),r(5),lc(5)
100 DIM np(5)
110 DIM px%(5,81),py%(5,81)
120 st=1:cx(1)=320:cy(1)=200:r(1)=80
130 FOR st=1 TO 4
140 r(st+1)=r(st)/2
150 NEXT st
160 FOR st=1 TO 5
170 lc(st)=0:np(st)=0
180 np(st)=np(st)+1
190 px%(st,np(st))=r(st)*SIN(lc(st))
200 py%(st,np(st))=r(st)*COS(lc(st))
210 lc(st)=lc(st)+360/r(st)
220 IF lc(st) < 360 THEN 180

```

This program continues on the next page

```

230 px%(st,np(st)+1)=px%(st,1)
240 py%(st,np(st)+1)=py%(st,1)
250 NEXT st
260 CLS:cj=REMAIN(1):cj=REMAIN(2)
270 cj=REMAIN(3):INK 1,2:st=1
280 GOSUB 350
290 LOCATE 1,1
300 EVERY 25,1 GOSUB 510
310 EVERY 15,2 GOSUB 550
320 EVERY 5,3 GOSUB 590
330 ERASE cx,cy,r,lc,np,px%,py%:NEXT
340 GOTO 340
350 cx%=cx(st):cy%=cy(st):lc(st)=0
360 FOR x%=1 TO np(st)
370 MOVE cx%,cy%
380 DRAW cx%+px%(st,x%),cy%+py%(st,x%),1+(st MOD 3)
390 DRAW cx%+px%(st,x%+1),cy%+py%(st,x%+1),1+(st MOD 3)
400 NEXT x%
410 IF st=5 THEN RETURN
420 lc(st)=0
430 cx(st+1)=cx(st)+1.5*r(st)*SIN(sa+lc(st))
440 cy(st+1)=cy(st)+1.5*r(st)*COS(sa+lc(st))
450 st=st+1
460 GOSUB 350
470 st=st-1
480 lc(st)=lc(st)+2*sa
490 IF (lc(st) MOD 360)<>0 THEN 430
500 RETURN
510 ik(1)=1+RND*25
520 IF ik(1)=ik(2) OR ik(1)=ik(3) THEN 510
530 INK 1,ik(1)
540 RETURN
550 ik(2)=1+RND*25
560 IF ik(2)=ik(1) OR ik(2)=ik(3) THEN 550
570 INK 2,ik(2)
580 RETURN
590 ik(3)=1+RND*25
600 IF ik(3)=ik(1) OR ik(3)=ik(2) THEN 590
610 INK 3,ik(3)
620 RETURN

```

AMSTRAD CPC664 FOUNDATION COURSE

Part 9: Using Sound....

Sound is generated by a loudspeaker within the computer itself. If you are using an MP2 Modulator/Power supply and a domestic television, set the TV's volume control to a minimum.

The level of sound can be adjusted by use of the **VOLUME** control on the right hand side of the computer. The sound can also be fed to the auxiliary input socket of your stereo system, using the socket on the computer marked **STEREO**. This will enable you to listen to the sound generated by the computer in stereo, through your hi-fi loudspeakers or headphones. Instructions on connecting to the computer's **STEREO** socket will be found in part 2 of this Foundation course.

The SOUND Command

The **SOUND** command has 7 parts ('parameters'). The first two of these must be used; the rest are optional. The command is typed in as:

```
SOUND <channel status> , <tone period> , <duration> , <volume> , <volume envelope> ,  
      <tone envelope> , <noise period>
```

It looks pretty complicated, but if we analyse each parameter, we can soon get to grips with it. Let's look at the parameters one by one....

Channel Status

To keep things simple at the moment, regard the <channel status> as the reference number for the sound channel. There are 3 sound channels, and for now we will use the <channel status> number 1.

Tone Period

<Tone period> is a technical way of defining the pitch of the sound, or in other words, 'what note it is' (i.e. do re mi fa so, etc). Each note has a set number, and this number is the <tone period>. Refer to the chapter entitled 'For your reference....', and you will see that the note middle c (do), has a tone period of 478.

Now reset the computer [CTRL] [SHIFT] [ESC], and type in:

```
10 sound 1,478
run
```

You will hear a short note which is middle c lasting 0.2 second.

If you don't hear anything, make sure that the **VOLUME** control on the computer is not set to zero, then type RUN again.

Duration

This parameter sets the length of the sound, in other words, 'how long it lasts'. The parameter works in units of 0.01 (one hundredth) of a second, and if you don't specify the <duration>, the computer will assume a figure of 20, which is why the note you just heard lasted 0.2 second, i.e. 0.01 multiplied by 20.

To make the note last for 1 second, a <duration> of 100 would be used; to make it last 2 seconds, 200 would be used. Type in:

```
10 sound 1,478,200
run
```

You will hear the note middle c lasting 2 seconds.

Volume

This parameter specifies the starting volume of a note. The number is in the range 0 to 15. A <volume> figure of 0 is minimum, while 15 is maximum. If no number is used, 12 is assumed. Type in:

```
10 sound 1,478,200,5
run
```

Note the volume of this sound. Now type it in using a higher volume number:

```
10 sound 1,478,200,15
run
```

You will hear that this is much louder.

Volume Envelope

To make the volume vary within the duration of the note, you can specify a volume envelope using the separate command `ENV`. You can in fact make a number of different volume envelopes, and like the `SOUND` command, each has its own reference number. If you have created a volume envelope with a reference number of 1, and you wish to use it in a `SOUND` command, then where the parameter `<volume envelope>` is required, type in 1. Creating a volume envelope will be explained shortly.

Tone Envelope

To make the tone or pitch vary within the duration of the note, you can specify a tone envelope using the separate command `ENT`. You can in fact make a number of different tone envelopes, and like the `SOUND` command, each has its own reference number. If you have created a tone envelope with a reference number of 1, and you wish to use it in a `SOUND` command, then where the parameter `<tone envelope>` is required, type in 1. Creating a tone envelope will be explained shortly.

Noise

`<Noise>` is the last parameter of the `SOUND` command. A range of noise is available by varying the `<noise>` parameter between 1 and 31. Add a `<noise>` parameter of 2 at the end of the `SOUND` command and listen to the effect. Then change the `<noise>` parameter to 27 and listen to the difference. Type in:

```
10 sound 1,478,200,15,,,2
```

Note the two 'blank' parameters (,,,) before the `<noise>` parameter of 2. This is because we haven't created a `<volume envelope>` nor a `<tone envelope>`.

Creating a Volume Envelope

The volume envelope command is `ENV`. In its simplest form, the command has 4 parameters. The command is typed in as:

```
ENV <envelope number> , <number of steps> , <size of step> , <time per step>
```

As before, let's look at the parameters one by one.

Envelope Number

This is the reference number (between 0 and 15) given to a particular volume envelope so that it can be called up in the **SOUND** command.

Number of Steps

This parameter specifies how many different steps of volume you want the sound to pass through before it ends. For example, in a note which lasts 10 seconds, you may wish to have 10 volume steps of 1 second each. In such a case, the 'number of steps' parameter used should be 10.

The available range of 'number of steps' is 0 to 127.

Size of Step

Each step can vary in size from a volume level of 0 to 15 with respect to the previous step. The 16 different volume levels are the same as those you will hear in the **SOUND** command. However, the 'size of step' parameter used can be between -128 and +127; the volume level re-cycling to 0 after each 15.

Time per Step

This parameter specifies the time between steps in 0.01 second (hundredths of a second) units. The range of 'time per step' numbers is 0 to 255, which means that the longest time between steps is 2.56 seconds (0 is treated as 256).

Note therefore, that the 'number of steps' parameter multiplied by the 'time per step' parameter shouldn't be greater than the 'duration' parameter in the **SOUND** command, otherwise the sound will finish before all the volume steps have been passed through.

(In such a case, the remaining contents of the volume envelope are discarded.)

Likewise, if the 'duration' parameter in the **SOUND** command is longer than the 'number of steps' multiplied by the 'time per step', the sound will continue after all of the volume steps have been passed through, and will remain constant at the final level.

To experiment with the volume envelope, type in the following program:

```
10 env 1,10,1,100
20 sound 1,284,1000,1,1
run
```

Line 20 specifies a **SOUND** with a tone period of 284 (international a), lasting for 10 seconds with a start volume of 1, and using volume envelope number 1, consisting of 10 steps, raising the volume of each step by 1, every 1 second (100 x 0.01 second).

Change line 10 in each of the following ways and then run to hear the effect of changing the envelope:

```
10 env 1,100,1,10
10 env 1,100,2,10
10 env 1,100,4,10
10 env 1,50,20,20
10 env 1,50,2,20
10 env 1,50,15,30
```

And finally try this:

```
10 env 1,50,2,10
```

You will notice that half way through the sound, the level remains constant. This is because the number of steps was 50 and the time between each step was 0.1 second. Therefore the length of time during which the volume varied was only 5 seconds, but the 'duration' parameter in the **SOUND** command in line 20 was 1000, i.e. 10 seconds.

Try experimenting yourself, to see what type of sounds you can create.

If you wish to create a more intricate volume envelope, the 3 parameters: 'number of steps', 'size of step', 'time per step' may be repeated at the end of the **ENV** command up to 4 more times, to specify a different 'section' of the same envelope.

Creating a Tone Envelope

The tone envelope command is **ENT**. In its simplest form, the command has 4 parameters: The command is typed in as:

```
ENT <envelope number> , <number of steps> , <tone period of step> , <time per step>
```

Once again, let's look at the parameters one by one.

Envelope Number

This is the reference number (between 0 and 15) given to a particular tone envelope so that it can be called up in the **SOUND** command.

Number of Steps

This parameter specifies how many different steps of tone (pitch) you want the sound to pass through before it ends. For example, in a note which lasts 10 seconds, you may wish to have 10 tone steps of 1 second each. In such a case, the 'number of steps' parameter used should be 10.

The available range of 'number of steps' is 0 to 239.

Tone Period of Step

Each tone step can vary in the range -128 to +127 with respect to the previous step. Negative steps make the pitch of the note higher, Positive steps make the pitch of the note lower. The shortest tone period is 0. This must be remembered when formulating the tone envelope. The full range of tone periods is shown in the chapter entitled 'For your reference....'.

Time per Step

This parameter specifies the time between steps in 0.01 second (hundredths of a second) units. The range of 'time per step' numbers is 0 to 255, which means that the longest time between steps is 2.56 seconds (0 is treated as 256).

Note therefore, that the 'number of steps' parameter multiplied by the 'time per step' parameter shouldn't be greater than the 'duration' parameter in the SOUND command, otherwise the sound will finish before all the tone steps have been passed through. (In such a case, the remaining contents of the tone envelope are discarded.)

Likewise, if the 'duration' parameter in the SOUND command is longer than the 'number of steps' multiplied by the 'time per step', the sound will continue after all of the tone steps have been passed through, and will remain constant at the final tone pitch.

To experiment with the tone envelope, type in the following program:

```
10 ent 1,100,2,2
20 sound 1,284,200,15,,1
run
```

Line 20 specifies a SOUND with a tone period of 284 (international a) lasting for 2 seconds with a start volume of 15 (max), without a volume envelope (represented by a blank parameter , ,) and with tone envelope number 1.

Line 10 is tone envelope number 1 consisting of 100 steps, increasing the tone period (reducing the pitch) by 2, every 0.02 second (2 x 0.01 second).

Now change line 10 in each of the following ways, and then **RUN** to hear the effect of changing the tone envelope:

```
10 ent 1,100,-2,2
10 ent 1,10,4,20
10 ent 1,10,-4,20
```

Now replace the **sound** command and the tone envelope by typing in:

```
10 ent 1,2,17,70
20 sound 1,142,140,15,,1
30 goto 10
run
```

Press **[ESC]** twice to break.

Now you can put the volume envelope, tone envelope, and **sound** command together to create various sounds. Start with:

```
10 env 1,100,1,3
20 ent 1,100,5,3
30 sound 1,284,300,1,1,1
run
```

Then replace line 20 by typing in:

```
20 ent 1,100,-2,3
run
```

Now replace all the lines by typing in:

```
10 env 1,100,2,2
20 ent 1,100,-2,2
30 sound 1,284,200,1,1,1
run
```

If you wish to create a more intricate tone envelope, the 3 parameters: 'number of steps', 'tone period of step', 'time per step' may be repeated at the end of the **ENT** command up to 4 more times, to specify a different 'section' of the same envelope.

Try some more variations for yourself. Add some 'noise' to the **SOUND** command, and try adding some extra sections to the volume and tone envelopes.

The chapter entitled 'Complete list of AMSTRAD CPC664 BASIC Keywords' contains full details of the various sound commands. If you are interested in the more melodious aspects of sound, see the section, 'The Sound of Music' which you will find in the chapter entitled 'At your leisure....'.

AMSTRAD CPC664 FOUNDATION COURSE

Part 10: Introducing AMSDOS and CP/M....

What is AMSDOS?....

When the computer is switched on or reset, the system defaults to operation under 'AMSDOS'. AMSDOS is an abbreviation of AMStrad Disc Operating System, and offers the following file handling commands:

```
load "filename"
run "filename"
save "filename"
chain "filename"
merge "filename"
chain merge "filename"
openin "filename"
openout "filename"
closein
closeout
cat
input #9
line input #9
list #9
print #9
write #9
```

In addition, AMSDOS provides a number of extra commands for disc management.

These commands are called external commands, and are preceded with a bar symbol |. (You will find the | symbol by holding down [SHIFT] and pressing the @ key.)

Some of the more common external commands that you will use are:

```
| a
| b
| tape (which can be sub-divided into | tape.in and | tape.out)
| disc (which can be sub-divided into | disc.in and | disc.out)
```

The commands | a and | b are used on a 2-drive system, to tell the computer which drive to direct any subsequent disc command.

Typing in for example:

```
l a  
load "filename"
```

will tell the computer to load the specified program from a disc placed in Drive A.

If neither `l a` nor `l b` is entered or the computer is reset, the system will default to Drive A.

If you are using only the disc drive within the computer, this can be regarded as Drive A, and `l a` or `l b` commands need not be issued. Entering `l b` when no additional disc drive is connected, will result in the message on the screen:

```
Drive B: disc missing  
Retry, Ignore or Cancel
```

to which you should respond `C` (to cancel).

What if I want to use cassette?....

The command `l tape` tells the computer to perform all loading and saving etc. commands to an external cassette unit instead of to disc. Unless `l tape` is entered, the computer will always default to disc operation when switched on or reset.

The return to disc operation after `l tape` has been specified, type in: `l disc`

Alternatively, you may for example wish to load in from cassette and save out to disc. You may then use the command:

```
l tape.in
```

....this command tells the computer to read data in from cassette, but continue to write data out onto disc (by default).

Similarly, to read data in from disc and save out onto cassette, you will first need to type in: `l disc.in` to countermand the previously issued `l tape.in` (above), then: `l tape.out` to tell the computer to write data out onto cassette.

It can be seen therefore that `l tape.in` and `l tape.out` countermand `l disc.in` and `l disc.out` respectively, and vice versa.

Further information on directing data to and from discs and cassette will be found later in this manual in the chapters entitled 'Using discs and cassettes' and 'AMSDOS and CP/M'.

In part 7 of the Foundation course, you learned how to format a new blank disc using your master CP/M system disc, and how to copy programs from disc to disc. From the above description of the commands:

```
ltape ldisc ltape.in ltape.out ldisc.in ldisc.out la lb
```

....you should now be able to perform loading and saving to either disc or cassette (if connected), placed in either Drive A or Drive B (if connected).

Other external commands:

```
ldir ldrive lera lren luser
```

....are dealt with later in this manual, in the chapter entitled 'AMSDOS and CP/M'.

Copying Using the CP/M System Disc

The entire contents of a disc can be copied from one to another using the CP/M system disc. If you do NOT have an additional disc drive connected to the computer, use the DISCCOPY facility.

If you HAVE connected an additional disc drive, you will find it easier and faster to use the COPYDISC facility.

Copying Using DISCCOPY

Insert the CP/M system disc and type in:

```
lcpm
```

After the prompt A> appears on the screen, type in: `discopy`

The computer will ask you to:

```
Please insert source disc into drive A then press any key
```

Remove the CP/M system disc from the drive, and insert the disc that you wish to copy. If you wish to make a back-up copy of the CP/M disc itself, simply leave the CP/M disc inserted in the drive.

When you have inserted your source disc and pressed a key, the computer will display the message:

```
Copying started
Reading track 0 to 7
```

After which, you will be asked to:

Please insert destination disc into drive A
then press any key

Whereupon you should remove your source disc and insert a disc for copying onto.

Me Bear in mind that any previous data on your destination disc will be overwritten during copying.

If you insert a wrongly formatted or new unformatted disc into the drive, it will be correctly formatted automatically during copying.

After you have inserted your destination disc and pressed a key, the computer will display the message:

Writing track 0 to 7

....and will then invite you to insert the source disc once again for reading tracks 8 to 15. The reading/writing process will continue, 8 tracks at a time, until the last track (39) is completed.

You will then be asked:

Do you want to copy another disc (Y/N):

If you have finished copying, answer N, then follow the instructions on the screen to exit from the DISCCOPY mode.

Copying Using COPYDISC

(2 drives)

COPYDISC can only be used if you have connected and additional disc drive. The facility enables you to copy the entire contents of one disc to another, in a similar manner to DISCCOPY previously described. The advantage of COPYDISC however, is that you do not have to repeatedly insert and remove the source and destination discs.

Having read and understood the principles of DISCCOPY, COPYDISC may then be performed as follows:

Insert the CP/M system disc and type in:

lcpm

After the prompt A> type in:

copydisc

Follow the instructions on the screen, and the contents of your source disc will be copied onto your destination disc 8 tracks at a time, until the last track (39) is completed. Like DISCCOPY, COPYDISC incorporates automatic formatting if required. COPYDISC can also be used to produce a back-up copy of the CP/M system disc itself.

Checking Discs

The CP/M system disc provides facilities for comparing one disc against another, to check for error free data copying.

If you have NOT connected an additional disc drive to the computer, select CP/M then type in:

```
discchk
```

Follow the instructions on the screen, and the destination disc will be checked against the source disc. If the computer detects a difference between the discs, it will display the message:

```
Failed to verify destination disc correctly:
(track x sector y)
```

Otherwise, the computer will continue to check through the discs, 8 tracks at a time, until checking is completed. If an error was detected, a final **WARNING:** will be displayed on the screen, before asking whether you wish to check another disc.

If you HAVE connected an additional disc drive, disc checking may be carried out with more speed and ease, using the CHKDISC facility. This operates in a similar manner to DISCHK previously described. The advantage of CHKDISC however, is that you do not have to repeatedly insert and remove the source and destination discs. To use CHKDISC, select CP/M then type in:

```
chkdisc
```

and follow the instructions on the screen.

Aborting

Note that functions performed using the CP/M system disc can be aborted by holding down [CTRL] and pressing the C key. Doing so will return you to direct mode CP/M (This is called 'Direct Console Mode').

As an example, select CP/M and type in:

`disccopy`

When the computer invites you to insert a disc, press **[CTRL]C**. The function will then be aborted.

Further information on **DISCCOPY**, **COPYDISC**, **DISCCHK** and **CHKDISC** together with **FORMAT** and other CP/M commands will be found later in this manual in the chapter entitled 'AMSDOS and CP/M'.

Finally, check that you have observed the following warnings given at the beginning of this manual, in the section entitled 'IMPORTANT':

INSTALLATION NOTES 5,6,7

OPERATION NOTES 1,2,3,4,5,6,7,8,9

Well, that concludes this 10-part Foundation course on the CPC664. By now you should know what most of the keys on the keyboard do, how to use some of the simpler BASIC commands, how to format a brand new disc so that it's ready to use, and how to perform some of the most elementary disc functions, i.e. **LOADing**, **SAVEing**, **CATaloguing** etc, together with a few simple AMSDOS and CP/M commands. The next part of this manual looks at the more specialised aspects of computing and AMSTRAD BASIC. It goes in-depth into the disc drive section of the unit, with sections on AMSDOS and CP/M, and starts you off on a new language - Dr. LOGO from Digital Research.

Chapter 2

Beyond Foundations....

So you've read the Foundation course and you have the computer switched on in front of you. You've already learnt how to make it carry out an operation several times over using a **FOR NEXT** loop, and how to make it test **IF** a condition is true **THEN** do something.

Well, you're soon going to tire of seeing your name printed all over the screen, and will want to get on with some serious computing - something useful or entertaining. The chapter after this, lists each of the AMSTRAD BASIC keywords at your disposal, together with a description of the 'syntax', and what the keyword is used for. Armed with this list, the scope of what you can then make the computer do is limited only by your imagination.

If you have never used a computer before, the idea of actually 'programming' may fill you with apprehension. Fear not! It's a lot easier than you think, and certainly a lot easier than the technology and jargon would have you believe. Think of BASIC not as a new language, but as a variation of English with some words abbreviated to speed things up. In other words, don't think of **C L S** as 3 letters of magic code, but instead as, **C**LEAR **S**CREEN.

Try not to be afraid of BASIC, and you'll soon find yourself enjoying the business of programming, as well as the fruits of your endeavours. Programming can be a very rewarding exercise, especially when you're a beginner experimenting with the machine and the language. Always remember that as long as you make sure that you don't accidentally write onto your master CP/M system disc, nothing you type in can actually harm the computer, and it's always worth trying something new.

So where do I start?

Starting is often the most difficult part of the program for the beginner. However, what you should avoid doing, is plunging straight in and hacking away at the keyboard without any forethought.

One of the first things you should establish, is what exactly you want the program to do, and how you want the results to be presented to you (in other words, what the screen should look like as the program runs).

Having decided this, you can then start writing a program to fulfil your requirements, all the time thinking of how to make the program flow smoothly from beginning to end, with the minimum of jumping around using **GOTO** here and **GOTO** there. A good program will be easy to follow when listed, and will not land you in a hopeless muddle when you try to fault find, or 'de-bug' as they say in computer talk.

Fortunately, BASIC is a very forgiving language and will often help you by producing error messages on the screen when you go wrong. BASIC also allows you to have 'afterthoughts', and new lines of program can be sandwiched in between the existing lines with a minimum of fuss.

Writing a simple program

OK let's get going. We'll write a program to keep a record of our friends' names and telephone numbers. We'll call the program 'Telephone book'. Now let's apply the above rules: 'What should the program do?' and 'How should the results be presented?'

Well, the program should let you enter up to say, 100 names and 'phone numbers for storage. When you want a number, you should be able to type in the name and get back the number. In addition, if you're not sure how one of the names was originally entered, you should be able to display a complete list of all the information on the screen. Notice by the way, that we're automatically starting to consider the question of how the results of the program are to be presented.

Right, let's put finger to keyboard! We'll start off with the title at the beginning:

```
10 REM telephone book
```

You don't have to put a title in a program, but when you start to accumulate quite a few programs, it helps to be able to know at a glance which is which.

Next, we know that we want to be able to INPUT (put in) a string of characters (somebody's name) into a variable; we'll call that variable NAME\$. The same applies to the telephone number, and we'll call that variable TEL\$.

Remember those example programs from the Foundation course? They used the INPUT command for you to enter the value of the variable, so if we type in:

```
20 INPUT "enter the name";NAME$
30 INPUT "enter the telephone number";TEL$
run
```

You could enter the name, for example: Joe. You could then enter the 'phone number, for example: 0277 230222

The program has stored the information, but hasn't produced any printed results on the screen. A section of program is therefore needed to retrieve the information, then display it. To get the values of NAME\$ and TEL\$ at the moment, we'd use commands like:

```
PRINT NAME$....and....PRINT TEL$
```

But hang on! We said that we want to be able to store up to 100 names and 100 'phone numbers in the program. Surely we don't have to write a program with two hundred **INPUT** commands, each with a different variable name, and then two hundred **PRINT** commands to produce a list on the screen??? Don't worry, computers make provision for this with what's known as an 'array'. An array allows you to use one variable (such as **NAME\$**) which can have any number of 'dimensions' (in our program, we require 100). Then, when you want to get at the contents of the variable, you use the variable's name followed by its reference number (inside brackets). This reference number is called a 'subscript' and the expression **NAME\$(27)** for example, is called a 'subscripted variable'. Now if we use a number-variable as the subscript, for example **NAME\$(x)**, we can then process the whole list of variables from 1 to 100 by changing the value of **x** in a **FOR NEXT** loop, i.e. **FOR x=1 TO 100**. As the value of **x** increases by 1, so the subscript value changes and refers to a different 'element' or name in the **NAME\$** array.

We want two arrays; one for **NAME\$**, and one for **TEL\$**, each with a dimension of 100 elements. Before we can start using an array, we must declare its **DIMensions** at the outset. We'll overwrite lines 10 and 20 with these statements:

```
20 DIM NAME$(100)
30 DIM TEL$(100)
```

Having established our variables, let's write some program that will firstly enable us to enter the names and numbers into the arrays, for later retrieval. Add:

```
40 FOR x=1 TO 100
50 INPUT;" name";NAME$(x)
60 INPUT " phone";TEL$(x)
70 NEXT
run
```

This is all very well, but we may not want to enter all 100 names at once. What's more, the way that the program presents itself on the screen is most unsatisfactory. What's needed here is a bit of tidying up. Firstly, before taking input of each new name and number, let's rid the screen of all the previous superfluous text, by **C**Learing the **S**creen each time. Add:

```
45 CLS
```

Now, how do we get the computer to know that we've finished inputting information for the moment? Pressing **[ESC]** would stop the program alright, but as soon as you typed **RUN** again, you'd lose all the values of your carefully entered variables!

Here's what we can do instead. As the program takes input of a new name, we'll make the program check that at least something has been typed in for the value of `NAME$(x)`, in other words, check whether the value of `NAME$(x)` is an 'empty string' or not. IF it is, THEN we'll make the program stop taking any more input. Did you get the clue on how we can do this? Add:

```
55 IF NAME$(x)="" THEN 80
80 PRINT "no more input"
```

Also, so that the program tells you how to stop inputting, add:

```
47 PRINT "[ENTER] nothing to end input"
```

Now let's write some program to PRINT the information you've stored, firstly in the form of a list. Add:

```
90 FOR x=1 TO 100
100 PRINT NAME$(x); " "; TEL$(x)
110 NEXT
```

Once again the program doesn't know when to stop, before reaching the 100th element of the array, so let's add:

```
95 IF NAME$(x)="" THEN 120
120 PRINT "list finished"
```

Line 95 detects whether `NAME$(x)` is an empty string, and IF so, THEN stops printing by bypassing lines 100 and 110.

And so to the next of our requirements. We'll now write some program that searches for a particular name that you enter. Add:

```
130 INPUT "find"; SEARCH$
140 FOR x=1 TO 100
150 IF INSTR(NAME$(x), SEARCH$)=0 THEN 180
160 PRINT NAME$(x); " "; TEL$(x)
170 END
180 NEXT
190 PRINT "name not found"
run
```

There's a new command in line 150 - `INSTR`. It tells the computer to `INterrogate` the first `STRing` expression to find the first occurrence of the second string expression, in other words, it searches `NAME$(x)` for `SEARCH$` (which is the variable you input in line 130 containing the name that you're looking for). If `INSTR` doesn't find it (or any part of it), it gives out a value of 0, which is used here to make the program pass to line 180 and try again with the `NEXT` value of `x`. If the program has passed through all values of `x` up to 100, it then continues to line 190 and tells you that it hasn't found the name. If however it does find the name, `INSTR` will not produce a value of 0, and the program will then pass from line 150 to 160 and print the name and phone number, then `END` at line 170.

As you can see, our program is developing quite rapidly now, but there's still so much to be done. Let's sit back for a moment and consider some drawbacks of this program, starting with the way in which the program runs: First you type in the information, then you get a list back, then you search for a specific name.

What if?

Well, what if you don't want to do it in that order? What if you want to start by searching for a name that you stored in the program yesterday? And what if you want to add some more names and numbers to those already there? These are all aspects of the program that you have to think about and find solutions for; it's what programming is all about. As previously mentioned, BASIC is kind enough to let you sandwich afterthoughts into the program, but a good programmer will have anticipated these problems beforehand.

Another major problem with this program is that the values of the variables in the arrays are all stored in a part of the computer's memory that is cleared whenever you `RUN` a program. You'll not want to have to type in all the information every time you use the program, so you'll need the option of being able to save the values of all the `NAME$` and `TEL$` variables before you switch off, together with the option to load in the values of the variables whenever you run the program.

Solutions

The first of these problems (i.e. the order in which things are carried out) can be dealt with by writing the program so that when it runs, you get a choice of the various functions that it can perform. This type of program is called 'menu-driven', and in effect displays a menu on the screen from which you can select an option. If you've ever used one of those cash dispensers outside the bank, then you'll have already operated a menu-driven computer program! Let's add a menu to this program:

```
32 PRINT "1. enter info"
33 PRINT "2. list info"
34 PRINT "3. search"
35 PRINT "4. save info"
36 PRINT "5. load info"
37 INPUT "[ENTER] menu selection";ms
38 ON ms GOSUB 40,90,130

85 RETURN
125 RETURN
170 RETURN
200 RETURN
```

As you can see, we've made the program print the menu of options, then take INPUT of your selection, putting it into the variable `ms`. The command `ON ms GOSUB` in line 38 tells the program that if `ms=1` then GO to the first SUB-routine line number (40); if `ms=2` then GO to the second SUB-routine line number (90), and so on.

As each of the functions are now sub-routines called by the `ON ms GOSUB` command, they must each have a RETURN command at the end, hence we have added all those RETURN commands above.

Do you remember what the RETURN command does? It makes BASIC return from the sub-routine to the point in the program immediately following the appropriate GOSUB command, so in this case it returns to the instruction after line 38 (which means that the program will continue at line 40 - the 'enter info' point!) We don't want that to happen, so we must add:

```
39 GOTO 32
```

....to make the program loop back and display the menu once again. Now RUN the program again to see how far we've progressed.

OK let's have a look at the listing of the program for a moment. (If the program is still running, press [ESC] twice.) Type in:

```
LIST
```

This is what you should have so far:

```
10 REM telephone book
20 DIM NAMES(100)
30 DIM TEL$(100)
32 PRINT "1. enter info"
33 PRINT "2. list info"
34 PRINT "3. search"
35 PRINT "4. save info"
```

```

36 PRINT "5. load info"
37 INPUT "[ENTER] menu selection";ms
38 ON ms GOSUB 40,90,130
39 GOTO 32
40 FOR x=1 TO 100
45 CLS
47 PRINT "[ENTER] nothing to end input"
50 INPUT;" name";NAME$(x)
55 IF NAME$(x)="" THEN 80
60 INPUT " phone";TEL$(x)
70 NEXT
80 PRINT "no more input"
85 RETURN
90 FOR x=1 TO 100
95 IF NAME$(x)="" THEN 120
100 PRINT NAME$(x);" ";TEL$(x)
110 NEXT
120 PRINT "list finished"
125 RETURN
130 INPUT "find";SEARCH$
140 FOR x=1 TO 100
150 IF INSTR(NAME$(x),SEARCH$)=0 THEN 180
160 PRINT NAME$(x);" ";TEL$(x)
170 RETURN
180 NEXT
190 PRINT "name not found"
200 RETURN

```

You'll see that in certain parts of the program, we're starting to run out of lines to insert instructions, so let's create some more space and tidy things up by RENUMbering the lines. Type in:

```

RENUM
LIST

```

You should now see:

```

10 REM telephone book
20 DIM NAME$(100)
30 DIM TEL$(100)
40 PRINT "1. enter info"
50 PRINT "2. list info"
60 PRINT "3. search"
70 PRINT "4. save info"
80 PRINT "5. load info"
90 INPUT "[ENTER] menu selection";ms

```

```

100 ON ms GOSUB 120,210,270
110 GOTO 40
120 FOR x=1 TO 100
130 CLS
140 PRINT "[ENTER] nothing to end input"
150 INPUT;" name";NAME$(x)
160 IF NAME$(x)="" THEN 190
170 INPUT " phone";TEL$(x)
180 NEXT
190 PRINT "no more input"
200 RETURN
210 FOR x=1 TO 100
220 IF NAME$(x)="" THEN 250
230 PRINT NAME$(x);" ";TEL$(x)
240 NEXT
250 PRINT "list finished"
260 RETURN
270 INPUT "find";SEARCH$
280 FOR x=1 TO 100
290 IF INSTR(NAME$(x),SEARCH$)=0 THEN 320
300 PRINT NAME$(x);" ";TEL$(x)
310 RETURN
320 NEXT
330 PRINT "name not found"
340 RETURN

```

That's better. Now on with the program! We'll now add an instruction, so that whenever you enter some new names and 'phone numbers, the computer will add them to the existing entries, by placing them in the first element of the array that it finds to be empty. This time, we'll use the new command LEN to tell us the LENGTH of the string. We'll specify the following:

IF the LENGTH of NAME\$(x) is greater than 0, i.e. if there's already an entry in that element of the array, THEN jump to line 180 (which steps to the NEXT element in the array).

Notice again how similar the above instruction in English is, compared to the equivalent BASIC instruction just below. I told you that BASIC isn't really a different language!!!

```

135 IF LEN(NAME$(x))>0 THEN 180

```

Such a simple solution isn't it? Problems like these can always be sorted out with your list of BASIC keywords and a little thought. There's nearly always at least one command that will satisfy your programming needs, and the more you program, the more you'll be able to find instant solutions 'off the top of your head'.

Now to the matter of saving the contents of the variables so that they can be loaded back in when the program is run. Part 7 of the Foundation course explained how to save the program itself, using the **SAVE** command. However the program itself is only a framework which lets the variable values be put in (at the keyboard) and taken out (at the screen). When you **SAVE** the program, you are saving only that framework, not the values of the variables.

Hence we must write a section of program that will save the values of the variables onto disc. We do this by creating a separate 'data file'.

First we **OPEN** an **OUT**put file and specify a name for it such as "data". Then we **WRITE** the values of the variables **NAME\$(x)** and **TEL\$(x)** from 1 to 100, into the file, and finally we **CLOSE** the **OUT**put file and **RETURN** to the menu. Let's add it to the program from line 350 onwards. To save us typing in each new line number, we'll use the command:

```
AUTO 350
```

....which will start **AUTO**matic line numbering from the required line:

```
350 OPENOUT "data"
360 FOR x=1 TO 100
370 WRITE #9,NAME$(x),TEL$(x)
380 NEXT
390 CLOSEOUT
400 PRINT "data saved"
410 RETURN
```

After you have typed in line 410 and **[ENTER]**ed it, press **[ESC]** to stop the **AUTO**matic line numbering.

Now we need to add an extra number to the list of numbers in the **ON ms GOSUB** command in line 100. This is because we have added another option for the menu to select. Therefore **EDIT** line 100 to add this extra number:

```
100 ON ms GOSUB 120,210,270,350
```

Now, whenever you select menu option number 4, the program will save all the information that you entered, onto disc.

Notice that in line 370, where the program **WRITEs** the values of **NAME\$(x)** and **TEL\$(x)** onto disc, the expression **#9** is used after the word **WRITE**. The **#** sign is a 'stream director', in other words, it tells the computer which 'stream' to send the data to. The computer has 10 streams:

Directing data to streams 0 to 7 (**#0** to **#7**) will result in it appearing on the screen because streams **#0** to **#7** are 'screen streams' or **WINDOWS**.

Directing data to stream #8 sends data to the printer (if connected).

Finally, directing data to stream #9 sends it to the disc drive, which is what we have done in line 370.

If I may digress....

A few words now about the AUTO command which we used a moment ago. If you leave out the line number and just type in:

```
AUTO
```

....the computer will commence line numbering from 10, advancing by 10 each time the line is [ENTER]ed. If you have previously used lines 10, 20, 30 etc, each line's contents will be displayed on the screen as you pass through it (by pressing [ENTER] each time). When each line appears on the screen, it may be edited before pressing [ENTER], thus providing a quick method of continuously editing regularly successive program lines.

Back to the program....

We've now added the instructions to save the information to disc, so the final main section of this program will load the data back from disc, ready for use. Therefore, we must add yet another menu option to the list of numbers in line 100. Edit line 100 again, as follows:

```
100 ON ms GOSUB 120,210,270,350,420
```

Now for the instructions to load the information. We start by OPENing the INput file from disc called "data". Then we take INPUT from disc (stream #9) of all the values of the variables NAMES(x) and TEL\$(x) from 1 to 100, and lastly we CLOSE the INput file and RETURN to the menu. Type in:

```
420 OPENIN "data"
430 FOR x=1 TO 100
440 INPUT #9,NAMES(x),TEL$(x)
450 NEXT
460 CLOSEIN
470 PRINT "data loaded"
480 RETURN
```

The end of the beginning....

So now we have written a program which fulfils the requirements that we set out to achieve when we decided 'what the program has to do'. All that remains now, is to improve the way that 'the results are presented' on the screen.

The beginning of the end....

Let's add some instructions to tidy up the presentation of the program:

```
34 MODE 1
```

This establishes the screen mode, and clears the screen at the start of the program. Now add:

```
36 WINDOW #1,13,30,10,14
```

Don't be put off by this seemingly complicated instruction. What we're doing here is creating a small window on the screen to put the menu into. After the word `WINDOW`, we first specify which stream number this window is for, remember we can use 8 screen streams from `#0` to `#7`. Now bearing in mind that all items printed on the screen use stream `#0` unless otherwise instructed, we won't use stream `#0` for our little window, otherwise everything that the program prints will be sent to it. Instead we'll specify another stream between `#1` and `#7`, and as you can see, we've chosen `#1`. The four numbers that follow `#1`, tell the computer what size the `WINDOW` should be, and it couldn't be easier; the numbers specify the left, right, top, and bottom edges of the window, and refer to text column and row numbers (the same as those used in the `LOCATE` command). So in our example, after specifying that it's stream `#1` we're using, we say that the left edge of the window starts at column `13`, the right edge ends at column `30`, the top edge starts at row `10`, and the bottom edge ends at row `14`.

Now to make all our menu options print in stream (`WINDOW`) `#1`, we'll have to edit lines `40` to `80` as follows:

```
40 PRINT #1,"1. enter info"  
50 PRINT #1,"2. list info"  
60 PRINT #1,"3. search"  
70 PRINT #1,"4. save info"  
80 PRINT #1,"5. load info"
```

Let's now add:

```
85 LOCATE 7,25
```

This LOCATEs where the menu INPUT statement in line 90 will appear, so that it looks neater.

So that the screen is always cleared when the menu is returned to, edit line 110 to:

```
110 GOTO 34
```

So that the screen is always cleared when one of the menu options is selected, add:

```
95 CLS
```

Finally, we'll just add the following three lines to make the program pause before returning to the menu:

```
103 LOCATE 9,25
105 PRINT "press any key for menu"
107 IF INKEY$="" THEN 107
```

Line 103 tells the computer where to print the message given in line 105. Line 107 INTERrogates the KEYboard to see what \$tring variable is being input (by a key being pressed). IF it detects that an empty string is being input (because no key is pressed), THEN the program loops back to the same instruction again and again, until INKEY\$ finds that the string is not empty because a key has been pressed. This is a useful way of creating a pause in a program, as BASIC will not pass to next line until a key is pressed.

So there it is; the finished program. Or is it? Well, you could make it have the facility to amend and delete names and 'phone numbers, to sort the list into alphabetical order, to 'print-out' the list on a printer, or if you're really ambitious, to make the program produce the signals to automatically dial the number after you've typed in the name - with of course, the permission of British Telecom to connect your 'phone to the computer! Nevertheless all these enhancements to the program are possible, and in truth you can go on forever improving and streamlining a program, especially when you've got a computer system as powerful as the CPC664. No, we've got to draw the line somewhere, and this is where we'll leave our 'telephone book', hopefully having learned a thing or two about the art of writing a program from scratch. Tidy up the program by typing in:

```
RENUM
```

....and then save it to disc, or throw it away. You never know though, it might come in handy for keeping a record of your friends' names and telephone numbers!

Final listing:

```
10 REM telephone book
20 DIM NAMES(100)
30 DIM TEL$(100)
40 MODE 1
50 WINDOW #1,13,30,10,14
60 PRINT #1,"1. enter info"
70 PRINT #1,"2. list info"
80 PRINT #1,"3. search"
90 PRINT #1,"4. save info"
100 PRINT #1,"5. load info"
110 LOCATE 7,25
120 INPUT "[ENTER] menu selection";ms
130 CLS
140 ON ms GOSUB 190,290,350,430,500
150 LOCATE 9,25
160 PRINT "press any key for menu"
170 IF INKEY$="" THEN 170
180 GOTO 40
190 FOR x=1 TO 100
200 CLS
210 IF LEN(NAMES(x))>0 THEN 260
220 PRINT "[ENTER] nothing to end input"
230 INPUT;" name";NAMES(x)
240 IF NAMES(x)="" THEN 270
250 INPUT " phone";TEL$(x)
260 NEXT
270 PRINT "no more input"
280 RETURN
290 FOR x=1 TO 100
300 IF NAMES(x)="" THEN 330
310 PRINT NAMES(x);" ";TEL$(x)
320 NEXT
330 PRINT "list finished"
340 RETURN
350 INPUT "find";SEARCH$
360 FOR x=1 TO 100
370 IF INSTR(NAMES(x),SEARCH$)=0 THEN 400
380 PRINT NAMES(x);" ";TEL$(x)
390 RETURN
400 NEXT
410 PRINT "name not found"
420 RETURN
430 OPENOUT "data"
```

```
440 FOR x=1 TO 100
450 WRITE #9,NAMES$(x),TEL$(x)
460 NEXT
470 CLOSEOUT
480 PRINT "data saved"
490 RETURN
500 OPENIN "data"
510 FOR x=1 TO 100
520 INPUT #9,NAMES$(x),TEL$(x)
530 NEXT
540 CLOSEIN
550 PRINT "data loaded"
560 RETURN
run
```

Chapter 3 Complete list of Amstrad CPC664 BASIC keywords

IMPORTANT

It is vital that you understand the terminology and notation that we use in this chapter. You will see various types of brackets used when explaining how a particular command is typed in; each of these types of brackets has a specific meaning, and you should note them well.

Any part of a command not shown enclosed by brackets is required as given. For example, the command **END** takes the form:

END

....and you must type in the word **END** literally.

Where an item is enclosed in angled brackets < > for example:

<line number>

....you are NOT required to type the brackets, nor the words within them. The example above shows you the type of data required in the command. For example:

EDIT <line number>

....means that you should type in:

EDIT 100

Round brackets () MUST be typed in literally. For example:

COS (<numeric expression>)

....requires that brackets be typed around the <numeric expression> of which the **COS**ine is required, e.g:

PRINT COS(45)

Finally, square brackets enclose optional items in a command or function. For example:

`RUN [⟨line number⟩]`

....means that you do not have to follow the keyword `RUN` with a parameter, but that you can expand the command by adding the optional parameter `⟨line number⟩`. Hence, the command could be typed in as:

`RUN ....or.... RUN 100`

Special Characters

<code>& or &H</code>	Prefix for hexadecimal constant
<code>&X</code>	Prefix for binary constant
<code>#</code>	Prefix for stream director

Data types

Strings may be from 0 to 255 characters long. A `⟨string expression⟩` is an expression which yields a value of type string. Strings may be appended to one another using the `+` operator, as long as the resulting string is no greater than 255 characters long.

Numeric data can be either integer or real. Integer data is held in the range -32768 to 32767 and real data is held to a little over nine digits of precision in the range $\pm 1.7E+38$ with the smallest value above zero approximately $2.9E-39$.

A `⟨numeric expression⟩` is any expression that results in a numeric value. It may simply be numbers, or it may be a numeric variable, or it may be numbers operated on by variables; just about anything that is not a `⟨string expression⟩`.

A `⟨stream expression⟩` refers to a `⟨numeric expression⟩` which identifies a screen window, printer, or disc, where the text is required to 'stream'.

A `⟨list of:⟨item⟩⟩` describes a parameter which comprises a list of items separated by commas. The list may contain one, or any number of items, limited by line length.

Type markers are:

<code>%</code>	Integer
<code>!</code>	Real (The default)
<code>\$</code>	String

Please note that AMSTRAD CPC664 BASIC keywords are listed here using the form:

KEYWORD

Syntax

Example

Description

Associated keywords

Keywords are either:

COMMANDS : operations that are executed directly.

FUNCTIONS : operations that are brought into action as arguments
in an expression.

OPERATORS : acting upon mathematical arguments.

BASIC converts all keywords entered in lower case letters to UPPER CASE when a program is **L I S T**ed. The examples shown in this chapter use UPPER CASE, since this is how the program will appear when **L I S T**ed. Hence you should enter using lower case, as you will be able to spot typing errors more readily since the mis-typed keyword will still be displayed in lower case when **L I S T**ed.

For a comprehensive guide to AMSTRAD CPC664 BASIC, see the Concise BASIC specification SOFT 945.

Keywords....

ABS

ABS (<numeric expression>)

```
PRINT ABS(-67.98)
67.98
```

FUNCTION: Returns the **ABS**olute value of the given expression. This means that negative numbers are returned as positive.

Associated keywords: **S G N**

```

10 AFTER 200 GOSUB 60:CLS
20 PRINT "Guess a letter in 5 seconds"
30 a$=INKEY$:IF flag=1 THEN END
40 IF a$<>CHR$(INT(RND*26+97)) THEN 30
50 PRINT a$;" is correct. You win"
55 SOUND 1,478:SOUND 1,358:END
60 PRINT "too late. I win"
70 SOUND 1,2000:flag=1:RETURN
run

```

AFTER

AFTER <timer delay>[, <timer number>] **GOSUB** <line number>

```

10 AFTER 250 GOSUB 60:CLS
20 PRINT "Guess a letter in 5 seconds"
30 a$=INKEY$:IF flag=1 THEN END
40 IF a$<>CHR$(INT(RND*26+97)) THEN 30
50 PRINT a$;" is correct. You win"
55 SOUND 1,478:SOUND 1,358:END
60 PRINT "too late. I win"
70 SOUND 1,2000:flag=1:RETURN
run

```

*in line 5 has
in line 6
return
back to line 30*

COMMAND: Calls a BASIC sub-routine after a given period of time has elapsed. The <timer delay> parameter specifies the period of time in units of 0.02 (fiftieths) second.

The <timer number> (in the range 0 to 3) specifies which of the four delay timers are to be used. Timer 3 has the highest priority; timer 0 (the default timer) has the lowest.

Each of the timers may have a sub-routine associated with it

Further information concerning interrupts will be found in part 2 of the chapter entitled 'At your leisure....'.

Associated keywords: **EVERY**, **REMAIN**, **RETURN**

AND

<argument> **AND** <argument>

```

IF "alan"<"bob" AND "dog">"cat" THEN PRINT "correct" ELSE PRINT "wrong"
correct
IF "bob"<"alan" AND "cat">"dog" THEN PRINT "correct" ELSE PRINT "wrong"
wrong
IF "alan"<"bob" AND "cat">"dog" THEN PRINT "correct" ELSE PRINT "wrong"
wrong
....
PRINT 1 AND 1
1
PRINT 0 AND 0
0
PRINT 1 AND 0
0

```

OPERATOR: Performs bit-wise boolean operation on integers. Result is 0 unless both argument bits are 1.

Further information concerning logic will be found in part 2 of the chapter entitled 'At your leisure....'.

Associated keywords: **OR**, **NOT**, **XOR**

ASC

ASC (⟨string expression⟩)

```
PRINT ASC("x")
120
```

FUNCTION: Returns the numeric value of the first character in the ⟨string expression⟩.

Associated keywords: CHR\$

ATN

ATN (⟨numeric expression⟩)

```
PRINT ATN(1)
0.785398163
```

FUNCTION: Calculates the Arc-TaNgent of the ⟨numeric expression⟩.

Note that DEG and RAD can be used to force the result of the above calculation to degrees or radians respectively.

Associated keywords: COS, DEG, RAD, SIN, TAN

AUTO

AUTO [⟨line number⟩][, ⟨increment⟩]

```
AUTO 100,50
```

COMMAND: Generates line numbers AUTOMATICALLY. The optional ⟨line number⟩ parameter sets the first line to be generated in case you wish to generate lines from a particular point in the program. If the parameter is omitted, line numbers will be generated from line 10 onwards.

The optional ⟨increment⟩ sets the number of lines to leave before generating the following line number. If the parameter is omitted, the line numbers will increase by 10 each time.

If a line number is generated which is already in use, then the contents of the line will be displayed on the screen and may be edited if required. The displayed line will be replaced in the memory when [ENTER] is pressed.

To stop automatic line numbering, press [ESC].

Associated keywords: none

BIN\$

BIN\$ (⟨unsigned integer expression⟩[,⟨integer expression⟩])

```
PRINT BIN$(64,8)
01000000
```

FUNCTION: Produces a string of **BIN**ary digits representing the value of the ⟨unsigned integer expression⟩, using the number of binary digits instructed by the second ⟨integer expression⟩ (in the range 0 to 16). If the number of digits instructed is too great, the resulting expression will be filled with leading zeros; if the number of digits instructed is too small, the resulting expression will **NOT** be shortened to the instructed number of digits, but will be produced in as many digits as are required.

The ⟨unsigned integer expression⟩ to be converted into binary form must yield a value in the range -32768 to 65535.

Associated keywords: **DEC\$**, **HEX\$**, **STR\$**

BORDER

BORDER ⟨colour⟩[,⟨colour⟩]

```
10 REM 729 border combinations!
20 SPEED INK 5,5
30 FOR a=0 TO 26
40 FOR b=0 TO 26
50 BORDER a,b:CLS:LOCATE 14,13
60 PRINT "border";a;" ";b
70 FOR t=1 TO 500
80 NEXT t,b,a
run
```

COMMAND: Changes the colour of the border on the screen. If two colours are specified, the border alternates between the two at a rate determined in the **SPEED INK** command. The range of border colours is 0 to 26.

Associated keywords: **SPEED INK**

BREAK

(See **ON BREAK CONT**, **ON BREAK GOSUB**, **ON BREAK STOP**)

CALL

CALL <address expression>[, <list of:parameter>]

CALL Ø

COMMAND: Allows an externally developed sub-routine to be called from BASIC. The above call completely resets the computer.

Not a command to be used by the unwary.

Associated keywords: UNT

CAT

CAT

CAT

COMMAND: CAT alogs the disc. Displays in alpha-numeric order, the full names of all files found, together with each file's length (to the nearest higher Kbyte). The free space left on the disc is also displayed, together with Drive and User identification.

Cataloguing does not affect the program currently in memory.

Associated keywords: LOAD, RUN, SAVE

CHAIN

CHAIN <filename>[, <line number expression>]

CHAIN "testprog.bas",35Ø

COMMAND: Loads a program from disc into the memory, replacing the existing program. The new program then commences running, either from the beginning, or from a line specified in the optional <line number expression>.

Protected files, (SAVED by ,p) can be loaded and run by CHAINing.

Associated keywords: CHAIN MERGE, LOAD, MERGE

CHAIN MERGE

CHAIN MERGE <filename>[, <line number expression>]
[, DELETE <line number range>]

CHAIN MERGE "newrun.bas", 750, DELETE 400-680

COMMAND: Loads a program from disc, merges it into the current program in the memory, then commences running the resultant program, either from the beginning, or from a line specified in the optional <line number expression>. If before a program is CHAIN MERGE'd, it is required to delete part of the original program, the DELETE <line number range> option may be used.

Note that line numbers in the old program which exist in the new program to be CHAIN MERGE'd, will be over-written by the new program lines.

Protected files, (SAVE'd by ,p) can NOT be loaded and run by CHAIN MERGEing.

Associated keywords: CHAIN, DELETE, LOAD, MERGE

CHR\$

CHR\$ (<integer expression>)

```
10 FOR x=32 TO 255
20 PRINT x;CHR$(x),
30 NEXT
run
```

FUNCTION: Converts an <integer expression> in the range 0 to 255, to its CHaRacter \$tring equivalent, using the AMSTRAD CPC664 character set shown in part 3 of the chapter entitled 'For your reference....'.

Note that 0 to 31 are control characters; hence the above example prints CHR\$(x) in the range 32 to 255.

Associated keywords: ASC

CINT

CINT (<numeric expression>)

```
10 n=1.9999
20 PRINT CINT(n)
run
2
```

FUNCTION: Returns the value of the <numeric expression>, Converting it to a rounded INTeger in the range -32768 to 32767.

Associated keywords: CREAL, FIX, INT, ROUND, UNT

CLEAR

CLEAR

CLEAR

COMMAND: Clears all variables to zero or null. All open files are abandoned, all arrays and user functions are erased, and BASIC is set to radians mode of calculation.

Associated keywords: **none**

CLEAR INPUT

CLEAR INPUT

```
10 CLS
20 PRINT "Type in Letters now!"
30 FOR t=1 TO 3000
40 NEXT
50 CLEAR INPUT
run
```

COMMAND: Discards all previously typed input from the keyboard, still in the keyboard buffer.

To see the effect of this command, RUN the above program and type in letters when asked to do so. Then delete line 50 of the program and RUN again, noting the difference.

Associated keywords: INKEY, INKEY\$, JOY

CLG

CLG [ink]

```
LOCATE 1,20
CLG 3
```

COMMAND: CLears the Graphics screen to the graphics paper colour. If the ink is specified, the graphics paper is set to that value.

Associated keywords: CLS, GRAPHICS PAPER, INK, ORIGIN

CLOSEIN

CLOSEIN

CLOSEIN

COMMAND: C L O S E s any I N p u t file from disc. (See O P E N I N).

Associated keywords: E O F, O P E N I N

CLOSEOUT

CLOSEOUT

CLOSEOUT

COMMAND: C L O S E s any O U T p u t file to disc. (See O P E N O U T).

Associated keywords: O P E N O U T

CLS

CLS[#stream expression]

```
10 PAPER#2,3
20 CLS#2
run
```

COMMAND: C L e a r s the given S c r e e n stream (window) to its paper ink. If no stream expression is given, screen stream #0 is cleared.

Associated keywords: C L G, I N K, P A P E R, W I N D O W

CONT

CONT

CONT

COMMAND: C O N T i n u e s program execution, either after the [ESC] key has been pressed twice, or after a S T O P command has been encountered in the program. C O N T will only continue to execute the program if it has not been altered, and if it is not a protected program.

Direct commands may be entered before C O N T i n u i n g.

Associated keywords: S T O P

COPYCHR\$

COPYCHR\$ (# <stream expression>)

```
10 CLS
20 PRINT "top corner"
30 LOCATE 1,1
40 a$=COPYCHR$(#0)
50 LOCATE 1,20
60 PRINT a$
run
```

FUNCTION: COPIes a CHaRacter from the current position in the stream (which MUST be specified). The above program copies a character from location 1,1 (top left), and reproduces it at location 1,20.

If the character read is not recognised, a null string is returned.

Associated keywords: LOCATE

COS

COS (<numeric expression>)

```
DEG
PRINT COS(45)
0.707106781
```

FUNCTION: Calculates the COSine of the <numeric expression>.

Note that DEG and RAD can be used to force the result of the above calculation to degrees or radians respectively.

Associated keywords: ATN, DEG, RAD, SIN

CREAL

CREAL (<numeric expression>)

```
10 a=PI
20 PRINT CINT(a)
30 PRINT CREAL(a)
run
3
3.14159265
```

FUNCTION: Returns the value of the <numeric expression>, Converting it to REAL.

Associated keywords: CINT

CURSOR

CURSOR[<system switch>][,<user switch>]

```
10 CURSOR 1
20 PRINT "question?";
30 a$=INKEY$:IF a$="" THEN 30
40 PRINT a$
50 CURSOR 0
run
```

COMMAND: Sets the system switch or the user switch to the cursor, on or off. The <system switch> and <user switch> parameters must be either 0 (off) or 1 (on). In the above INKEY\$ command, where the cursor is not normally visible, the cursor has been turned on by the <system switch> setting of 1 (in line 10).

The cursor is displayed whenever both the <system switch> and the <user switch> are on (1). The <system switch> is automatically turned on for the command INPUT, but is turned off for INKEY\$.

It is recommended that the cursor be turned off when printing text to the screen.

Either switch parameter may be omitted, but not both. If a switch parameter is omitted, that particular switch state is not changed.

Associated keywords: LOCATE

DATA

DATA <list of: constant>

```
10 FOR x=1 TO 4
20 READ name$,surname$
30 PRINT "Mr. ";name$;" ";surname$
40 NEXT
50 DATA Ken,Barlow,Alf,Roberts
60 DATA Mike,Baldwin,Jack,Duckworth
run
```

COMMAND: Declares constant data for use within a program. Data may be read into the variable by the READ command, after which the 'pointer' moves on to the next item in the DATA list. The RESTORE command may be used to move the pointer to a specified DATA position.

Further information concerning data will be found in part 2 of the chapter entitled 'At your leisure....'.

Associated keywords: READ, RESTORE

DEC\$

DEC\$ (<numeric expression> , <format template>)

```
PRINT DEC$(10↑7,"££#####,.##")
£10,000,000.00
```

FUNCTION: Returns a DE Cimal string representation of the <numeric expression>, using the specified <format template> to control the print format of the resulting string.

The format template may contain ONLY the characters:

+ - £ \$ * # , . ↑

The use of these 'format field specifiers' is described under the keyword PRINT USING.

Associated keywords: BIN\$, HEX\$, PRINT USING, STR\$

DEF FN

DEF FN <function name> [(<formal parameters>)] = <expression>

```
10 t=TIME/300
20 DEF FNClock=INT(TIME/300-t)
30 EVERY 100 GOSUB 50
40 GOTO 40
50 PRINT "program was run";
60 PRINT FNClock;"seconds ago"
70 RETURN
run
```

COMMAND: DEFines a FuNction. BASIC allows the program to define and use simple value returning functions. DEF FN is the definition part of this mechanism and creates a program-specific function which works within the program in the same way that for example COS operates as a built-in function of BASIC.

(Note in the above example how the value of the function FNClock is continually updated even if the program is paused by pressing [ESC] once, or stopped by pressing [ESC] twice, then CONTinuing.)

Associated keywords: none

DEFINT

DEFINT <list of: <letter range>

```
10 DEFINT n
20 number=123.456
30 PRINT number
run
123
```

COMMAND: Sets the DEFault for a variable to type INTeger. When a variable is encountered without an explicit type marker (! % \$), the default type is assumed. This command sets the default for variables with the specified first letter(s) to type INTeger. There may be a list of first letters such as:

```
DEFINT a,b,c
```

....or there may be an inclusive range of first letters such as:

```
DEFINT a-z
```

Associated keywords: DEFREAL, DEFSTR

DEFREAL

DEFREAL <list of: <letter range>

```
DEFREAL x,a-f
```

COMMAND: Sets the DEFault for a variable to type REAL. When a variable is encountered without an explicit type marker (! % \$), the default type is assumed. This command sets the default for variables with the specified first letter(s) to type REAL. There may be a list of first letters such as:

```
DEFREAL a,b,c
```

....or there may be an inclusive range of first letters such as:

```
DEFREAL a-z
```

Associated keywords: DEFINT, DEFSTR

DEFSTR

DEFSTR <list of> <letter range>

```
10 DEFSTR n
20 name="Amstrad"
30 PRINT name
run
Amstrad
```

COMMAND: Sets the DEFault for a variable to type STRING. When a variable is encountered without an explicit type marker (! % \$), the default type is assumed. This command sets the default for variables with the specified first letter(s) to type STRING. There may be a list of first letters such as:

```
DEFSTR a,b,c
```

....or there may be an inclusive range of first letters such as:

```
DEFSTR a-z
```

Associated keywords: DEFINT, DEFREAL

DEG

DEG

```
DEG
```

COMMAND: Sets DEGrees mode of calculation. The default condition for the functions SIN, COS, TAN, and ATN is radians. DEG resets BASIC to degrees until instructed otherwise by the commands RAD, and NEW, CLEAR, LOAD, RUN etc.

Associated keywords: ATN, COS, RAD, SIN, TAN

DELETE

DELETE <line number range>

DELETE 100-200

COMMAND: Deletes part of the current program as defined in the <line number range> expression.

The first or last number in the <line number range> may be omitted to indicate '...from the beginning of the program', or '...to the end of the program', i.e:

DELETE -200

....or....

DELETE 50-

....or....

DELETE

....which deletes the whole program.

Associated keywords: CHAIN MERGE, RENUM

DERR

DERR

LOAD "xyz.abc"

XYZ .ABC not found

Ready

PRINT DERR

146

FUNCTION: Reports the last error code returned by the disc filing system. The value of DERR may be used to ascertain the particular Disc ERROR that occurred. See the listing of error messages given in the chapter entitled 'For your reference....'.

Associated keywords: ERL, ERR, ERROR, ON ERROR GOTO, RESUME

DI

DI

```
10 CLS:TAG:EVERY 10 GOSUB 90
20 X1=RND*320:X2=RND*320
30 Y=200+RND*200:CS=CHR$(RND*255)
40 FOR X=320-X1 TO 320+X2 STEP 4
50 DI
60 MOVE 320,0,1:MOVE X-2,Y:MOVE X,Y
70 PRINT " ";CS;:FRAME
80 EI:NEXT:GOTO 20
90 MOVE 320,0:DRAW X+8,Y-16,0:RETURN
run
```

COMMAND: Disables Interrupts (other than the **[ESC]** interrupt) until re-enabled explicitly by **EI** or implicitly by the **RETURN** at the end of an interrupt sub-routine.

Note that entering an interrupt sub-routine automatically disables interrupts of an equal or lower priority.

The command is used to make the program literally execute without interruption - for example when two routines within a program are competing for use of resources. In the example above, the main program and the interrupt sub-routine are competing for use of the graphics display.

Further information concerning logic will be found in part 2 of the chapter entitled 'At your leisure....'.

Associated keywords: **AFTER**, **EI**, **EVERY**, **REMAIN**

DIM

DIM <list of <subscripted variable>

```
10 CLS
20 DIM friend$(5),phone$(5)
30 FOR n=1 TO 5
40 PRINT "Friend number";n
50 INPUT "Enter name";friend$(n)
60 INPUT "Enter tel.number";phone$(n)
70 PRINT
80 NEXT
90 FOR n=1 TO 5
100 PRINT n;friend$(n),phone$(n)
110 NEXT
run
```

COMMAND: DIMensions an array. DIM allocates space for arrays and specifies maximum subscript values. BASIC must be advised of the space to be reserved for an array, or it will default to 10.

An array is identified by a <subscripted variable> where one variable name is used with a range of subscript numbers, so that each 'element' of the array has its own individual value. Control of the array can then be achieved by for example FOR NEXT loops, which can step through the array, processing each element in turn.

Note that the lowest value of the subscript is zero (i.e. the first available element in the array).

Arrays can be multi-dimensional, and each element of such an array is referenced by its position within the framework of the array. For example, in an array dimensioned by:

```
DIM position$(20,20,20)
```

....an element of the array would be referenced for example:

```
position$(4,5,6)
```

Associated keywords: ERASE

DRAW

DRAW <x co-ordinate> , <y co-ordinate> [, [<ink>] [, <ink mode>]]

```
10 MODE 0:BORDER 0:PAPER 0:INK 0,0
20 x=RND*640:y=RND*400:z=RND*15
30 DRAW x,y,z
40 GOTO 20
run
```

COMMAND: Draws a line on the graphics screen, from the current graphics cursor position to the absolute position specified in the x,y co-ordinates. The <ink> in which to draw the line may be specified (in the range 0 to 15).

The optional <ink mode> determines how the ink being written interacts with that already on the graphics screen. The 4 <ink mode>s are:

0: Normal
1: XOR (eXclusive OR)
2: AND
3: OR

Associated keywords: DRAW, GRAPHICS PEN, MASK

DRAWR

DRAWR <x offset> , <y offset> [, [<ink>] [, <ink mode>]]

```
10 CLS:PRINT "coming upstairs?"
20 MOVE 0,350:FOR n=1 TO 8
30 DRAWR 50,0
40 DRAWR 0,-50
50 NEXT:MOVE 348,0:FILL 3
60 GOTO 60
run
```

COMMAND: Draws a line on the graphics screen, from the current graphics cursor position to the relative position specified in the <x and y offset>s. The <ink> in which to draw the line may be specified (in the range 0 to 15).

The optional <ink mode> determines how the ink being written interacts with that already on the graphics screen. The 4 <ink mode>s are:

0: Normal
1: XOR (eXclusive OR)
2: AND
3: OR

Associated keywords: DRAW, GRAPHICS PEN, MASK

EDIT

EDIT <line number>

EDIT 20

COMMAND: Displays the program line specified in the <line number> on the screen together with the cursor, ready for editing.

Associated keywords: AUTO, LIST

EI

EI

EI

COMMAND: Enables I nterrupts which have been disabled by the D I command.

If interrupts are disabled in an interrupt sub-routine, they are automatically re-enabled when BASIC encounters the RETURN command at the end of the sub-routine.

Further information concerning interrupts will be found in part 2 of the chapter entitled 'At your leisure....'.

Associated keywords: AFTER, DI, EVERY, REMAIN

ELSE

(See I F)

END

END

END

COMMAND: Ends the execution of a program, and returns to direct mode. Any number of **END** commands may appear in a program, and one is automatically assumed after the final line of a program.

Associated keywords: **STOP**

ENT

ENT <envelope number> [, <envelope section>] [, <envelope section>]
[, <envelope section>] [, <envelope section>]
[, <envelope section>]

```
10 ENT 1,10,-50,10,10,50,10
20 SOUND 1,500,200,10,,1
run
```

COMMAND: Sets the Tone ENvelope specified in the <envelope number> (in the range 1 to 15), which is used in conjunction with the **SOUND** command. If the <envelope number> is negative (in the range -1 to -15), the envelope repeats until the end of the duration of the **SOUND** command.

Each of the <envelope section>s may contain either 2 or 3 parameters:

If 3 parameters are used, these are:

<number of steps> , <step size> , <pause time>

Parameter 1: <number of steps>

This parameter specifies how many different steps of tone (pitch) you want the sound to pass through during the envelope section. For example, in a section of a note which lasts 10 seconds, you may wish to have 10 tone steps of 1 second each. In such a case, the <number of steps> parameter used should be 10.

The available range of <number of steps> is 0 to 239.

Parameter 2: <step size>

This parameter must be in the range -128 to +127. Negative steps make the pitch of the note higher; positive steps make the pitch of the note lower. The shortest tone period is 0. The full range of tone periods is shown in the chapter entitled 'For your reference....'.

Parameter 3: <pause time>

This parameter specifies the time between steps in 0.01 second (hundredths of a second) units. The range of <pause time> numbers is 0 to 255 (where 0 is treated as 256), which means that the longest time between steps is 2.56 seconds.

If 2 parameters are used, these are:

<tone period> , <pause time>

Parameter 1: <tone period>

This parameter gives a new absolute setting for the tone period. (See Parameter 2 of the SOUND command.)

Parameter 2: <pause time>

This parameter specifies the pausing time in 0.01 second (hundredths of a second) units. The range of <pause time> numbers is 0 to 255 (where 0 is treated as 256), or 2.56 seconds.

General

Note that the total length of all the <pause time>s should not be greater than the <duration> parameter in the SOUND command, otherwise the sound will finish before all the tone steps have been passed through. (In such a case, the remaining contents of the tone envelope are discarded.)

Likewise, if the <duration> parameter in the SOUND command is longer than the total length of all the <pause time>s, the sound will continue after all of the tone steps have been passed through, and will remain constant at the final tone pitch.

Up to 5 different <envelope section>s, (each made up of the above 2 or 3 parameters) may be used in an ENT command.

The first step of a tone envelope is executed immediately.

Each time a given tone envelope is set, its previous value is lost.

Specifying an <envelope number> with no <envelope section>s cancels any previous setting.

Further information concerning sound will be found in part 2 of the chapter entitled 'At your leisure....'.

Associated keywords: ENV, SOUND

ENV <envelope number>[, <envelope section>][, <envelope section>
[, <envelope section>][, <envelope section>
[, <envelope section>]

COMMAND: Sets the Volume **EN**velope specified in the <envelope number> (in the range 1 to 15), which is used in conjunction with the **SOUND** command.

If 3 parameters are used, these are:

Parameter 1: `<number of steps>`

The available range of <number of steps> is 0 to 127.

Each step can vary in size from a volume level of 0 to 15 with respect to the previous step. The 16 different volume levels are the same as those you will hear in the **SOUND** command. However, the `<step size>` parameter used can be between -128 and +127; the volume level re-cycling to 0 after each 15.

This parameter specifies the time between steps in 0.01 second (hundredths of a second) units. The range of 'pause time' numbers is 0 to 255 (where 0 is treated as 256), which means that the longest time between steps is 2.56 seconds.

If 2 parameters are used, these are:

⟨hardware envelope⟩ , ⟨envelope period⟩

Parameter 1: ⟨hardware envelope⟩

This parameter specifies the value to send to the envelope shape register of the the sound chip.

Parameter 2: ⟨envelope period⟩

This parameter specifies the value to send to the envelope period registers of the sound chip.

Knowledge of hardware is assumed when using hardware envelopes. Unless you have such knowledge, it is suggested that you use a software envelope incorporating a suitable ⟨pause time⟩ parameter.

General

Note that the total length of all the ⟨pause time⟩s should not be greater than the ⟨duration⟩ parameter in the SOUND command, otherwise the sound will finish before all the volume steps have been passed through. (In such a case, the remaining contents of the volume envelope are discarded.)

Likewise, if the ⟨duration⟩ parameter in the SOUND command is longer than the total length of all the ⟨pause time⟩s, the sound will continue after all of the volume steps have been passed through, and will remain constant at the final level.

Up to 5 different ⟨envelope section⟩s, (each made up of the above 2 or 3 parameters) may be used in an ENV command.

The first step of a volume envelope is executed immediately.

Each time a given volume envelope is set, its previous value is lost.

Specifying an ⟨envelope number⟩ with no ⟨envelope section⟩s cancels any previous setting.

Further information concerning sound will be found in part 2 of the chapter entitled 'At your leisure.....'.

Associated keywords: ENT, SOUND

EOF

EOF

```
10 OPENIN "ex1.bas"
20 WHILE NOT EOF
30 LINE INPUT #9,a$
40 PRINT a$
50 WEND
60 CLOSEIN
run
```

FUNCTION: Tests to see if the disc input is at End Of File. Returns -1 (true) if no file is open or file is at the end, otherwise returns 0 (false).

Associated keywords: OPENIN, CLOSEIN

ERASE

ERASE <list of: <variable name>

```
DIM a(100),b$(100)
ERASE a,b$
```

COMMAND: Erases the contents of an array no longer required, reclaiming the memory for other use.

Associated keywords: DIM

ERL

ERL

```
10 ON ERROR GOTO 30
20 GOTO 1000
30 PRINT "error is in line";ERL
40 END
run
```

FUNCTION: Reports the Line number of the last ERRor encountered. In the above example you will see that the error is in line 20, and has been reported so by the ERL function.

Associated keywords: DERR, ERR, ERROR, ON ERROR GOTO, RESUME

ERR

ERR

```
GOTO 500
Line does not exist
Ready
PRINT ERR
8
```

FUNCTION: Reports the number of the last **ERR** encountered. See the listing of error messages given in the chapter entitled 'For your reference....'. In the above example you will see that **ERR** number 8 is a 'Line does not exist' error.

Associated keywords: **DERR, ERL, ERROR, ON ERROR GOTO, RESUME**

ERROR

ERROR <integer expression>

```
10 IF INKEY$="" THEN 10 ELSE ERROR 17
run
```

COMMAND: Invokes the error specified in the <integer expression>. A listing of error messages 1 to 32 is given in the chapter entitled 'For your reference....'. **BASIC** will treat the **ERROR** as if it had been detected as genuine, and will jump to any error handling routine, as well as reporting the appropriate values of **ERR** and **ERL**.

ERROR accompanied by an <integer expression> in the range 33 to 255 can be used to create customised error messages, as shown in the following example:

```
10 ON ERROR GOTO 100
20 INPUT "enter one character";a$
30 IF LEN(a$)<>1 THEN ERROR 100
40 GOTO 20
100 IF ERR=100 THEN 110 ELSE 130
110 PRINT CHR$(7)
120 PRINT "I said ONE character!"
130 RESUME 20
run
```

Associated keywords: **ERL, ERR, ON ERROR GOTO, RESUME**

EVERY

EVERY <time period>[, <timer number>] GOSUB <line number>

```
10 EVERY 50,1 GOSUB 30
20 GOTO 20
30 SOUND 1,20
40 RETURN
run
```

COMMAND: Calls a BASIC sub-routine at regular intervals. The <time period> specifies the interval in units of 0.02 (fiftieths) second.

The <timer number> (in the range 0 to 3) specifies which of the four delay timers are to be used. Timer 3 has the highest priority; timer 0 (the default timer) has the lowest.

Each of the timers may have a sub-routine associated with it.

Further information concerning interrupts will be found in part 2 of the chapter entitled 'At your leisure....'.

Associated keywords: AFTER, REMAIN

EXP

EXP(<numeric expression>)

```
PRINT EXP(6.876)
968.743625
```

FUNCTION: Calculates 'E' to the power given in the <numeric expression>, where 'E' is approximately 2.7182818 - the number whose natural logarithm is 1.

Associated keywords: LOG

FILL

FILL <ink>

```
10 MODE 1:sea=2:MOVE 0,200,3
20 DRAWR 100,-100:DRAWR 220,0
30 MOVER 10,0:DRAWR 220,0
40 DRAWR 100,100:MOVE 0,0
50 FILL sea
run
```

COMMAND: Fills an arbitrary area of the graphics screen. The edges of the area are bounded by lines drawn either in the current graphics pen ink or in the ink being used to fill (in the range 0 to 15).

The fill starts from the current graphics cursor position. If this position lies on an edge, nothing will be filled.

Associated keywords: GRAPHICS PEN

FIX

FIX(*<numeric expression>*)

```
PRINT FIX(9.99999)
9
```

FUNCTION: Removes the part of *<numeric expression>* to the right of the decimal point, rounding towards zero.

Associated keywords: CINT, INT, ROUND

FN

(See DEF FN)

FOR

FOR *<simple variable>* = *<start>* TO *<end>* [**STEP** *<size>*]

```
10 FOR n=2 TO 8 STEP 2
20 PRINT n;
30 NEXT n
40 PRINT ",who do we appreciate?"
run
```

COMMAND: Carries out the body of program between the **FOR** and **NEXT** commands, a given number of times, stepping the control variable between a *<start>* and *<end>* value. If the **STEP** *<size>* is not specified, 1 is assumed.

The **STEP** *<size>* may be specified as a negative *<numeric expression>* in which case the value of the *<start>* parameter must be greater than that of the *<end>* parameter, otherwise the control variable will not be stepped.

FOR **NEXT** loops may be 'nested', i.e. one may be carried out within another, within another, and so on.

Assigning the variable's name to the **NEXT** command is optional as BASIC will automatically find which **FOR** command is to be associated with an 'anonymous' **NEXT**.

Associated keywords: **NEXT**, **STEP**, **TO**

FRAME

FRAME

```
10 MODE 0
20 PRINT "FRAME off"
30 TAG
40 MOVE 0,200
50 FOR x=0 TO 500 STEP 4
60 IF f=1 THEN FRAME
70 MOVE x,200
80 PRINT " ";CHR$(143);
90 NEXT
100 IF f=1 THEN RUN
110 CLS
120 TAGOFF
130 PRINT "FRAME on"
140 f=1
150 GOTO 30
run
```

COMMAND: Synchronises the writing of graphics on the screen, with the frame flyback of the display. The overall effect of this is that character or graphics movement on the screen will appear to be smoother, without 'flickering' or 'tearing'.

Associated keywords: TAG, TAGOFF

FRE

FRE (<numeric expression>)

FRE (<string expression>)

```
PRINT FRE(0)
PRINT FRE("")
```

FUNCTION: Establishes how much FREe memory remains unused by BASIC. The form FRE ("") forces a 'garbage collection' before returning a value for available space.

Associated keywords: HIMEM, MEMORY

GOSUB

GOSUB <line number>

```
GOSUB 210
```

COMMAND: GOes to a BASIC SUB-routine by branching to the specified <line number>. The end of the sub-routine itself is marked by the command **RETURN**, whereupon the program continues execution from the instruction after the invoked **GOSUB** command.

Associated keywords: **RETURN**

GOTO

GOTO <line number>

```
GOTO 90
```

COMMAND: GOes TO a specified line number.

Associated keywords: **none**

GRAPHICS PAPER

GRAPHICS PAPER <ink>

```
10 MODE 0
20 MASK 15
30 GRAPHICS PAPER 3
40 DRAW 640,0
run
```

COMMAND: Sets the <ink> of the graphics paper, i.e. the area behind graphics drawn on the screen. When drawing continuous lines, the graphics paper will not be seen. In the above example, the **MASK** command enables a broken line to be drawn, and the graphics paper to be seen.

The graphics paper's ink (in the range 0 to 15) is used for the 'paper' area of characters written when **TAG** is in operation, and as the default when clearing the graphics window, using **CLG**.

Associated keywords: **CLG, GRAPHICS PEN, INK, MASK, TAG, TAGOFF**

GRAPHICS PEN

GRAPHICS PEN [*ink*][, *background mode*]

```
10 MODE 0
20 GRAPHICS PEN 15
30 MOVE 200,0
40 DRAW 200,400
50 MOVE 639,0
60 FILL 15
run
```

COMMAND: Sets the *ink*, (in the range 0 to 15) to be used for drawing lines and plotting points. The graphics *background mode* can also be set to either:

0: Opaque background
1: Transparent background

(Transparent background affects the graphics paper of characters written with TAG on, and the gaps in dotted lines.)

Either parameter may be omitted, but not both. If a parameter is omitted, that particular setting is not changed.

Associated keywords: GRAPHICS PAPER, INK, MASK, TAG, TAGOFF

HEX\$

HEX\$ (*unsigned integer expression* [, *field width*])

```
PRINT HEX$(255,4)
00FF
```

FUNCTION: Produces a \$string of HEXadecimal digits representing the value of the *unsigned integer expression*, using the number of hexadecimal digits instructed by the *field width* (in the range 0 to 16). If the number of digits instructed is too great, the resulting expression will be filled with leading zeros; if the number of digits instructed is too small, the resulting expression will NOT be shortened to the instructed number of digits, but will be produced in as many digits as are required.

The *unsigned integer expression* to be converted into hexadecimal form must yield a value in the range -32768 to 65535.

Associated keywords: BIN\$, DEC\$, STR\$, UNT

HIMEM

HIMEM

```
PRINT HIMEM
42619
```

FUNCTION: Returns the address of the Highest byte of MEMory used by BASIC, (which may be altered by the MEMORY command).

Associated keywords: FRE, MEMORY, SYMBOL, SYMBOL AFTER

IF

IF <logical expression> THEN <option part> [ELSE <option part>]

```
10 MODE 1
20 x=CINT(RND*100)
30 PRINT "Guess my number (0 to 100)"
40 INPUT n
50 IF n<x THEN PRINT n;"is too low..."
60 IF n>x THEN PRINT n;"is too high..."
70 IF n=x THEN 80 ELSE c=c+1:GOTO 40
80 PRINT "Well done, you got it in";
90 PRINT c+1;"guesses!"
run
```

COMMAND: Determines whether the <logical expression> is true, in which case the first <option part> is executed. If the <logical expression> is false, any <option part> specified in the ELSE clause is executed, otherwise BASIC passes onto the next line.

IF THEN commands may be nested to any depth, and are terminated by end of line. Therefore it is NOT possible to have further statements which are independent of the IF THEN command, on the same line.

Where the result of the <logical expression> requires that a line should be jumped to, the command may be given as either:

Examples....

```
IF a=1 THEN 100
```

....or....

```
IF a=1 GOTO 100
```

....or....

```
IF a=1 THEN GOTO 100
```

Associated keywords: ELSE, GOTO, THEN

mode 0

ink 0 = ~~black~~ ink

ink 1 = ~~red~~ red pen 1 pen 1 pen 1 pen 1

paper #1, 1 → paper window 1 window 1

pen #1, 0 → pen 1 pen 1 pen 1 pen 1

INK

INK <ink>, <colour>[, <colour>]

```
10 MODE 1:PAPER 0:PEN 1
20 FOR p=0 TO 1
30 FOR i=0 TO 26
40 INK p,i
50 LOCATE 16,12:PRINT "ink";p;"",i
60 FOR t=1 TO 400:NEXT t,i,p
70 INK 0,1:INK 1,24:CLS
run
```

window #1,
paper #1,
pen #1,0
cls #1

COMMAND: Assigns colour(s) to a given ink. The <ink> parameter describes the ink reference, which must be an integer expression in the range 0 to 15, for use in the appertaining PEN or PAPER command. The first <colour> parameter should be an integer expression yielding a colour value in the range 0 to 26. If an optional second <colour> is specified, the ink alternates between the two colours, at a rate determined by the SPEED INK command.

Depending upon the current screen mode, a number of INKs are available. See the table of colour values given in Foundations 8, table 1.

Associated keywords: GRAPHICS PAPER, GRAPHICS PEN, PAPER, PEN, SPEED INK

INKEY

INKEY (<integer expression>)

```
10 IF INKEY(55)<>32 THEN 10
20 PRINT "You've pressed [SHIFT] and V"
30 CLEAR INPUT
run
```

FUNCTION: I Nterrogates the KEYboard to report which keys are being pressed. The keyboard is scanned every 0.02 (fiftieth) second.

The function is useful for spotting whether a certain key is down or up, by detecting the returned value of -1 (which occurs regardless of [SHIFT] and [CTRL] key status).

The above example detects when [SHIFT] and V (key number 55) are pressed together, then ends the program. An illustration of key numbers will be found in the diagram at the top right hand side of the computer, and in the chapter entitled 'For your reference....'.

The state of **[SHIFT]** and **[CTRL]** in conjunction with the key specified in the integer expression is identified as follows:

Value returned	[SHIFT]	[CTRL]	specified key
-1	UP/DOWN	UP/DOWN	UP
0	UP	UP	DOWN
32	DOWN	UP	DOWN
128	UP	DOWN	DOWN
160	DOWN	DOWN	DOWN

Associated keywords: **CLEAR INPUT, INKEY\$, JOY**

INKEY\$

*call 24131306 = watch up
feels in drink*

INKEY\$

```

10 CLS
20 PRINT "Select Yes or No (Y/N)?"
30 a$=INKEY$
40 IF a$="" THEN 30
50 IF a$="y" OR a$="Y" THEN 80
60 IF a$="n" OR a$="N" THEN 90
70 GOTO 30
80 PRINT "You have selected YES":END
90 PRINT "You have selected NO"
run

```

FUNCTION: **IN**terrogates the **KEY**board, returning the current \$tring reflecting any key that is pressed. If no key is pressed, **INKEY\$** returns an empty string. In the above example, lines 40 and 70 tell the program to loop back to line 30 after interrogating the keyboard string.

Associated keywords: **CLEAR INPUT, INKEY**

INP

INP (port number)

```

PRINT INP(&FF77)
255

```

*inp(-2600) = 94 => Printer int
inp(-1000) = 30
of inp(&F500) => 94
=> 30*

FUNCTION: Returns the **IN**Put value from the **I/O** address specified in the port number.

Associated keywords: **OUT, WAIT**


```
INPUT[#⟨stream expression⟩, ||; ||⟨quoted string⟩, separator |
                                     ⟨list of:⟨variable⟩⟩]
```

COMMAND: Accepts data input from the stated stream, (from stream #0 if not specified).

The `separator` must either be a semicolon or comma. A semicolon causes a question mark to be displayed; a comma suppresses the question mark.

If an entry is made that is of the wrong type such as if the letter o is typed instead of a 0 (zero) when INPUTing a numeric variable, then BASIC will respond with:

....and any original prompt text that you programmed.

All responses from the keyboard must be terminated by pressing the **[ENTER]** key.

Associated keywords: LINE INPUT

INSTR

INSTR ([start position], searched string, searched for string)

```
10 CLS:FOR n=1 TO 26
20 alphabet$=alphabet$+CHR$(n+64)
30 NEXT
40 INPUT "Enter a letter";a$
50 b$=UPPER$(a$)
60 PRINT b$;" is number";
70 PRINT INSTR(alphabet$,b$);
80 PRINT "in the alphabet.":PRINT
90 GOTO 40
run
```

FUNCTION: Searches the first <searched string> expression to find the <searched for string> expression, and reports the position of its first occurrence within the <searched string>. If the <searched for string> does not occur within the <searched string>, then 0 is reported.

The position at which to start searching the <searched string> is optionally specifiable using the <start position> parameter which must yield an integer number in the range 1 to 255.

Associated keywords: **none**

INT

INT (<numeric expression>)

```
PRINT INT(-1.995)
-2
```

FUNCTION: Rounds the number to the nearest smaller INTeger, removing any fractional part. Returns the same value as FIX for positive numbers, but returns one less than FIX for negative numbers which are not already integers.

Associated keywords: CINT, FIX, ROUND

JOY

JOY(⟨integer expression⟩)

```
10 PRINT "To stop the program - ";
20 PRINT "operate joystick"
30 IF JOY(0)<>0 THEN END
40 GOTO 10
run
```

FUNCTION: Reads a bit-significant result from the JOYstick specified in the ⟨integer expression⟩ (either 0 or 1).

Bit	Decimal
0: Up	1
1: Down	2
2: Left	4
3: Right	8
4: Fire 2	16
5: Fire 1	32

Hence for example, if the main 'fire' button (Fire 2) on the first joystick is pressed while the joystick handle is being moved left, the function JOY(0) returns a decimal value of 20, corresponding to 16 (Fire 2) + 4 (Left).

Further information concerning joysticks will be found in the chapter entitled 'For your reference....'.

Associated keywords: CLEAR INPUT, INKEY

KEY

KEY ⟨expansion token number⟩, ⟨string expression⟩

```
KEY 11,"border 13:paper 0:pen 1:ink 0,13:
ink 1,0:mode 2:list"+CHR$(13)
```

....now press the small [ENTER] key.

COMMAND: Assigns the ⟨string expression⟩ to the key's ⟨expansion token number⟩ specified. Thirty-two expansion tokens are supported, in the range 0 to 31, these occupying the key values 128 to 159. Keys 128 (0 on numeric keypad) to 140 ([CTRL] [ENTER] on numeric keypad) are by default assigned to print numbers 0 to 9, a decimal point, [ENTER] and RUN" [ENTER] - (for cassette operation), but may be re-assigned to other ⟨string expression⟩s as required. Expansion tokens 13 to 31 (key values 141 to 159) are empty strings by default, but may be expanded and assigned to keys, using the KEY DEF command described in the next example.

in key number 159 is the first of the 160 possible values.
 1) each key number, and the shifted keys, be
 if the 1 link is on key nr 68 (the one F80)
 2) give the value of the normal key value the key
 nr 68, the other 141 keys 159, the key nr 141
 the key nr Key def 68, 0, 141
 3) the key 141, " " " "

The <expansion token number> given in the KEY command may be in the range 0 to 31, or optionally 128 to 159 to reflect the key values. (See the key illustration in the chapter entitled 'For your reference....')

A total of 120 characters may be expanded into the <string expression>s. Attempting to over-expand will produce an 'Improper argument' error (5).

Associated keywords: KEY DEF

KEY DEF

1 = aan
 2 = uit

KEY DEF <key number>, <repeat>[, <normal>[, <shifted>[, <control>]]]

KEY 159, this is the tab key"

KEY DEF (68, 1, 159) key value dus key nr 68
now press the [TAB] key. krijg de waarde 159

COMMAND: DE Fines the KEY values to be returned by the specified <key number> in the range 0 to 79 (for an illustration of key numbers, refer to the diagram at the top right hand side of the computer, or to the chapter entitled 'For your reference....'). The <normal>, <shifted>, and <control> parameters should contain the values required to be returned when the key is pressed, alone, together with [SHIFT], and together with [CTRL], respectively. Each of these parameters is optional. zie (7.2) 141

The <repeat> parameter enables you to set the key auto-repeat function on or off (1 or 0), the rate of auto-repeat being adjustable by use of the SPEED KEY command.

In the above example, key 159 (equivalent to expansion token 31) is first assigned to an expansion string, then the KEY DEF command defines key 68 (the [TAB] key) to auto-repeat (1), and to return the <normal> value 159 when pressed alone.

In the above example, normal action would be restored by:

KEY DEF 68, 0, 9

....where 9 is the normal ASCII value for [TAB] (zie 7.2)

Associated keywords: KEY, SPEED KEY

by *SAITZ* *Ctrl* *12* *52* *72*
key 144, "Run" + chr(15) = 05 Def 50, 1, & 72, & 52, 144
159

LEFT\$

LEFT\$ (string expression, required length)

```
10 CLS
20 a$="AMSTRAD"
30 FOR n=1 TO 7
40 PRINT LEFT$(a$,n)
50 NEXT
run
```

FUNCTION: Returns the number of characters (in the range 0 to 255) specified in the 'required length' parameter, after extracting them from the LEFT of the 'string expression'. If the 'string expression' is shorter than the 'required length', the whole 'string expression' is returned.

Associated keywords: MID\$, RIGHT\$

LEN

LEN (string expression)

```
10 LINE INPUT "Enter a phrase";a$
20 PRINT "The phrase is";
30 PRINT LEN(a$);"characters long."
run
```

FUNCTION: Returns the total number of characters (i.e. the LENGTH) of the 'string expression'.

Associated keywords: none

LET

LET variable = expression

```
LET x=100
```

COMMAND: Assigns a value to a variable. A remnant from early BASICs where variable assignments had to be 'seen coming'. Has no use in AMSTRAD BASIC apart from providing compatibility with the programs supplied in early BASIC training manuals. The above example need only be typed in:

```
x=100
```

Associated keywords: none

of,



25

•

110

de

part

er i

....or....

LIST 50-

....or....

LIST

....which lists the whole program.

Associated keywords: **none**

LOAD

LOAD <filename>[, <address expression>]

LOAD "discfile.xyz",&2AF8

COMMAND: Loads a BASIC program from disc into memory, replacing any existing program. Specifying the optional <address expression> will cause a binary file to be loaded at that address, rather than the address from which it was saved.

A Protected BASIC program can NOT be loaded using the LOAD command as it will be immediately deleted from memory. Instead, use the RUN or CHAIN commands.

Associated keywords: CHAIN, CHAIN MERGE, MERGE, RUN, SAVE

LOCATE

LOCATE[#<stream expression> ,<x co-ordinate> ,<y co-ordinate>]

```
10 MODE 1
20 FOR n=1 TO 20
30 LOCATE n,n
40 PRINT CHR$(143);"location";
50 PRINT n;" ";n
60 NEXT
run
```

COMMAND: Locates the text cursor at the stream indicated, to the position specified by the x and y co-ordinates, with 1,1 being the top left corner of the stream (window). Stream #0 is the default stream.

Associated keywords: WINDOW

LOG

LOG (numeric expression)

```
PRINT LOG(9999)
9.21024037
```

FUNCTION: Calculates the natural LOGarithm of the numeric expression which must be greater than zero.

Associated keywords: EXP, LOG10

LOG10

LOG10 (numeric expression)

```
PRINT LOG10(9999)
3.99995657
```

FUNCTION: Calculates the LOGarithm to base 10 of the numeric expression which must be greater than zero.

Associated keywords: EXP, LOG

LOWERS

LOWERS\$ (string expression)

```
10 a$="SEE HOW THE LETTERS CHANGE TO "
20 PRINT LOWERS$(a$+"LOWER CASE")
run
```

FUNCTION: Returns a new string expression which is a copy of the specified string expression but in which all alphabetic characters in the range A to Z are converted to lower case. The function is useful for processing input which may come in mixed upper/lower case.

Associated keywords: UPERS

MASK

MASK [*<integer expression>*][, *<first point setting>*]

```
10 CLS:TAG
20 MASK 1:MOVE 0,250:DRAW 240,0
30 PRINT "(binary 00000001 in mask)";
40 MASK 3:MOVE 0,200:DRAW 240,0
50 PRINT "(binary 00000011 in mask)";
60 MASK 7:MOVE 0,150:DRAW 240,0
70 PRINT "(binary 00000111 in mask)";
80 MASK 15:MOVE 0,100:DRAW 240,0
90 PRINT "(binary 00001111 in mask)";
run
```

COMMAND: Sets the 'mask' or template to be used when drawing lines. The binary value of the *<integer expression>* in the range 0 to 255, sets the bits in each adjacent group of 8 pixels to ON (1), or OFF (0).

The *<first point setting>* determines whether the first point of the line is to be plotted (1) or not plotted (0).

Either of the parameters may be omitted, but not both. If a parameter is omitted, that particular setting is not changed.

Associated keywords: DRAW, DRAW, GRAPHICS PAPER, GRAPHICS PEN

MAX

MAX (*<list of: <numeric expression>*)

```
10 n=66
20 PRINT MAX(1,n,3,6,4,3)
run
66
```

FUNCTION: Returns the MAXimum value from the *<list of: <numeric expression>*s.

Associated keywords: MIN

MEMORY

MEMORY <address expression>

```
MEMORY &20AA
```

COMMAND: Allocates the amount of BASIC memory available by setting the address of the highest byte.

Associated keywords: FRE, HIMEM, SYMBOL, SYMBOL AFTER

MERGE

MERGE <filename>

```
MERGE "newload.bas"
```

COMMAND: Loads a program from disc, and adds it to the current program in the memory.

Note that line numbers in the old program which exist in the new program to be MERGED, will be over-written by the new program lines.

Protected files, (SAVED by ,p) can NOT be MERGED into the current program.

Associated keywords: CHAIN, CHAIN MERGE, LOAD

MID\$

MID\$ (<string expression> , <start position> [, <sub-string length>])

```
10 MODE 1:ZONE 3
20 a$="ENCYCLOPAEDIA"
30 PRINT "Show me how to spell ";a$
40 PRINT "OK....":PRINT
50 FOR n=1 TO LEN(a$)
60 PRINT MID$(a$,n,1),
70 FOR t=1 TO 700:NEXT t,n
80 PRINT:PRINT
90 INPUT "Now enter another word";a$
100 GOTO 50
run
```

FUNCTION: Returns a new sub-string, commencing at the `start position` of the `string expression`, being `sub-string length` characters long. If the `sub-string length` parameter is not specified, the remainder of the `string expression` after the `start position` is returned.

If the `start position` is greater than the total length of the `string expression`, then an empty string is returned. The range of `start position` is 1 to 255. The range of `sub-string length` is 0 to 255.

Associated keywords: `LEFT$, RIGHT$`

MID\$

MID\$ (`string variable`, `insertion position`[, `new string length`])
= `new string expression`

```
10 a$="hello"
20 MID$(a$,3,2)="XX"
30 PRINT a$
run
heXXo
```

COMMAND: Inserts the `new string expression` into the string specified by the `string variable`, commencing at the `insert position`, and occupying `new string length` number of characters.

Note that when using **MID\$** as a **COMMAND**, a `string variable` such as `a$` must be used, and NOT a string constant such as `"hello"`.

Associated keywords: `LEFT$, RIGHT$`

MIN

MIN (`list of: numeric expression`)

```
PRINT MIN(3,6,2.999,8,9,)
2.999
```

FUNCTION: Returns the **MIN**imum value from the `list of: numeric expressions`.

Associated keywords: `MAX`

MOD

⟨argument⟩ MOD ⟨argument⟩

```
PRINT 10 MOD 3
1
PRINT 10 MOD 5
0
```

OPERATOR: Returns the remainder after dividing the first ⟨argument⟩ by the second ⟨argument⟩ and removing any integer component - **MODulus**.

Associated keywords: **none**

MODE

MODE ⟨integer expression⟩

```
10 m=1
20 PRINT "this is mode";m
30 PRINT "now press a key"
40 IF INKEY$="" THEN 40
50 m=m+1:IF m>2 THEN m=0
60 MODE m
70 GOTO 20
run
```

COMMAND: Changes the screen mode (0,1 or 2), and clears the screen to ink 0 (which may not be the current paper ink). All text and graphics windows are reset to the whole screen, and text and graphics cursors are homed to their respective origins.

Associated keywords: **ORIGIN, WINDOW**

MOVE

MOVE <x co-ordinate> , <y co-ordinate> [, <ink>] [, <ink mode>]

```
10 MODE 1:TAG
20 x=RND*800-100:y=RND*430
30 MOVE x,y
40 PRINT "I moved here";
50 GOTO 20
run
```

COMMAND: Moves the graphics cursor to the absolute point specified by <x co-ordinate> and <y co-ordinate>. The optional <ink> parameter may be used to change the graphics pen ink, in the range 0 to 15.

The optional <ink mode> determines how the ink to be written next, will interact with that already on the graphics screen. The 4 <ink mode>s are:

- 0: Normal
- 1: XOR (eXclusive OR)
- 2: AND
- 3: OR

Associated keywords: MOVE, ORIGIN, XPOS, YPOS

MOVER

MOVER <x offset> , <y offset> [, <ink>] [, <ink mode>]

```
10 MODE 1:TAG:MOVE 0,16
20 PRINT "Life has its";
30 FOR n=1 TO 10
40 MOVER -32,16
50 PRINT "ups";:NEXT:PRINT " and";
60 FOR n=1 TO 10
70 MOVER -64,-16
80 PRINT "downs";:NEXT
run
```

COMMAND: Moves the graphics cursor to a point relative to its current position. The relative position is specified by <x offset> and <y offset>. The optional <ink> parameter may be used to change the graphics pen ink, in the range 0 to 15.

The optional <ink mode> determines how the ink to be written next, will interact with that already on the graphics screen. The 4 <ink mode>s are:

- 0: Normal
- 1: XOR (eXclusive OR)
- 2: AND
- 3: OR

Associated keywords: MOVE, ORIGIN, XPOS, YPOS

NEW

NEW

NEW

COMMAND: Deletes the current program and variables in the memory. Key definitions are not lost, display characteristics i.e. MODE, PEN, PAPER, INK etc, are not changed, and the screen is not cleared.

Associated keywords: none

NEXT

NEXT [list of variable]

```
10 FOR a=1 TO 3
20 FOR b=0 TO 26
30 MODE 1
40 PEN a:BORDER b
50 PRINT "pen";a;"border";b
60 FOR c=1 TO 500
70 NEXT c,b,a
run
```

COMMAND: Marks the end of a FOR loop. The NEXT command may be anonymous, or may refer to its matching FOR. Note from the above example that the list of variable:s must appear in reverse order to their matching FOR commands, so that 'nested' loops do not overlap.

Associated keywords: FOR, RESUME, STEP, TO

NOT

NOT argument

```
IF NOT "alan"<"bob" THEN PRINT "correct" ELSE PRINT "wrong"
wrong
IF NOT "cat">"dog" THEN PRINT "correct" ELSE PRINT "wrong"
correct
....
PRINT NOT -1
0
PRINT NOT 0
-1
```

OPERATOR: Performs bit-wise operations on integers. Inverts each bit in the argument.

Further information concerning logic will be found in part 2 of the chapter entitled 'At your leisure....'.

Associated keywords: AND, OR, XOR

ON BREAK CONT

ON BREAK CONT

```
10 ON BREAK CONT
20 PRINT "The program will CONTinue when you try to
   *Break* using [ESC]";PRINT
30 FOR t=1 TO 1000:NEXT:GOTO 20
run
```

COMMAND: Cancels the action of the **[ESC]** key from stopping the program, and instead CONTinues execution. Care should be taken when using this command, as the program will continue until the computer is completely reset; hence you should SAVE such a program before RUNning it.

ON BREAK CONT may be disabled within a program by ON BREAK STOP.

Associated keywords: ON BREAK GOSUB, ON BREAK STOP

ON BREAK GOSUB

ON BREAK GOSUB <line number>

```
10 ON BREAK GOSUB 40
20 PRINT "program running"
30 GOTO 20
40 CLS:PRINT "Pressing [ESC] ";
50 PRINT "twice calls GOSUB-routine"
60 FOR t=1 TO 2000:NEXT
70 RETURN
run
```

COMMAND: Instructs BASIC to jump to the sub-routine specified in the <line number> when the **[ESC]** key is pressed twice.

Associated keywords: ON BREAK CONT, ON BREAK STOP, RETURN

ON BREAK STOP

ON BREAK STOP

```
10 ON BREAK GOSUB 40
20 PRINT "program running"
30 GOTO 20
40 CLS:PRINT "Pressing [ESC] ";
50 PRINT "twice calls GOSUB-routine"
60 FOR t=1 TO 2000:NEXT
65 ON BREAK STOP
70 RETURN
run
```

COMMAND: Disables the ON BREAK CONT and ON BREAK GOSUB command, so that future operations of the [ESC] key stop the program. In the above example, the ON BREAK GOSUB command will operate once only, as it is then disabled by line 65 in the ON BREAK sub-routine.

Associated keywords: ON BREAK CONT, ON BREAK GOSUB

ON ERROR GOTO

ON ERROR GOTO <line number>

```
10 ON ERROR GOTO 60
20 CLS:PRINT "If error is found, ";
30 PRINT "then list the program"
40 FOR t=1 TO 4000:NEXT
50 GOTO 100
60 PRINT "Error detected in line";
70 PRINT ERL:PRINT:LIST
run
```

COMMAND: Jumps to the specified <line number> when an error is detected in the program.

The form of the command ON ERROR GOTO 0 turns off the error trap, and restores normal error processing by BASIC.

See also the RESUME command.

Associated keywords: DERR, ERL, ERR, ERROR, RESUME

ON **expression** GOSUB

ON **selector** GOSUB **list of line number**

```
10 PAPER 0: PEN 1: INK 0,1
20 CLS: PRINT "MENU OF OPTIONS": PRINT
30 PRINT "1 - Change border": PRINT
40 PRINT "2 - Change pen": PRINT
50 PRINT "3 - Change mode": PRINT
60 INPUT "Enter your selection"; x
70 ON x GOSUB 90,110,130
80 GOTO 20
90 b=b-1: IF b<0 THEN b=26
100 BORDER b: RETURN
110 p=p-1: IF p<2 THEN p=26
120 INK 1,p: RETURN
130 m=m-1: IF m<0 THEN m=2
140 MODE m: RETURN
run
```

COMMAND: Selects a sub-routine line to jump to, depending upon the value of the **selector**, which should be a positive integer expression in the range 0 to 255. The order of the **selector** values determines the **line number** to be selected from the **list of line number**s. In the above example, selecting 1 makes BASIC jump to line 90, selecting 2 jumps to line 110, and 3 jumps to line 130.

If the value of the **selector** is zero, or is higher than the amount of **line number**s listed in the command, then no sub-routine line will be selected.

Associated keywords: RETURN

ON <expression> GOTO

ON <selector> GOTO <list of: line number>

```
10 CLS:PRINT "MENU OF OPTIONS":PRINT
20 PRINT "1 - List program":PRINT
30 PRINT "2 - Edit and add":PRINT
40 PRINT "3 - Catalog disc":PRINT
50 INPUT "Enter your selection";n
60 ON n GOTO 80,90,100
70 GOTO 10
80 LIST
90 AUTO
100 CAT
run
```

COMMAND: Selects a line to jump to, depending upon the value of the <selector>, which should be a positive integer expression in the range 0 to 255. The order of the <selector> values determines the <line number> to be selected from the <list of: line number>s. In the above example, selecting 1 makes BASIC jump to line 80, selecting 2 jumps to line 90, and 3 jumps to line 100.

If the value of the <selector> is zero, or is higher than the amount of <line number>s listed in the command, then no line will be selected.

Associated keywords: none

ON SQ GOSUB

ON SQ (<channel>) GOSUB <line number>

```
10 ENV 1,15,-1,1
20 ON SQ(1) GOSUB 60
30 MODE 0:ORIGIN 0,0,200,440,100,300
40 FOR x=1 TO 13:FRAME:MOVE 330,200,x
50 FILL x:NEXT:GOTO 40
60 READ s:IF s=0 THEN RESTORE:GOTO 60
70 SOUND 1,s,25,15,1
80 ON SQ(1) GOSUB 60:RETURN
90 DATA 50,60,90,100,35,200,24,500,0
run
```

COMMAND: G0es to a BASIC SUB-routine when there is a free slot in the given Sound Queue. The <channel> should be an integer expression yielding one of the values:

- 1: for channel A
- 2: for channel B
- 4: for channel C

Further information concerning sound will be found in part 2 of the chapter entitled 'At your leisure....'.

Associated keywords: RETURN, SOUND, SQ

OPENIN

OPENIN <filename>

```
10 REM OPEN an INput file from disc
20 OPENIN "datafile":INPUT #9,a,a$
30 CLOSEIN:PRINT "The 2 values are:"
40 PRINT:PRINT a,a$
run
```

COMMAND: OPENS an INput file from disc, for use in the current program. The INput file to OPEN must be an ASCII file.

The above example will only work after you have created the file shown in the next example (under OPENOUT).

Associated keywords: CLOSEIN, EOF

OPENOUT

OPENOUT <filename>

```
10 REM OPEN an OUTput file to disc
20 INPUT "give me a number variable";a
30 INPUT "give me a string variable";a$
40 OPENOUT "datafile"
50 WRITE #9,a,a$
60 CLOSEOUT:PRINT "Data saved onto disc"
run
```

COMMAND: OPENS an OUTput file to disc.

Associated keywords: CLOSEOUT

OR

⟨argument⟩ OR ⟨argument⟩

```
IF "alan"<"bob" OR "dog">"cat" THEN PRINT "correct" ELSE PRINT "wrong"
correct
IF "bob"<"alan" OR "cat">"dog" THEN PRINT "correct" ELSE PRINT "wrong"
wrong
IF "alan"<"bob" OR "cat">"dog" THEN PRINT "correct" ELSE PRINT "wrong"
correct
....
PRINT 1 OR 1
1
PRINT 0 OR 0
0
PRINT 1 OR 0
1
```

OPERATOR: Performs bit-wise boolean operation on integers. Result is 1 unless both argument bits are 0.

Further information concerning logic will be found in part 2 of the chapter entitled 'At your leisure....'.

Associated keywords: AND, NOT, XOR

ORIGIN

ORIGIN ⟨x⟩,⟨y⟩[,⟨left⟩,⟨right⟩,⟨top⟩,⟨bottom⟩]

```
10 MODE 1:BORDER 13:TAG
20 ORIGIN 0,0,100,540,300,100
30 GRAPHICS PAPER 3:CLG
40 FOR x=550 TO -310 STEP -10
50 MOVE x,206
60 PRINT "This is a graphics window ";
70 FRAME:NEXT:GOTO 40
run
```

COMMAND: Sets the graphics origin points 0,0 to the position specified by the co-ordinates ⟨x⟩ and ⟨y⟩.

A graphics window's dimensions may also be set by specifying the last four optional parameters. If the co-ordinates specified for the graphics window describe points beyond the edge of screen, then the edge of the graphics window is taken as edge of screen.

Associated keywords: CLG

7 Centronics | out & EFOO, n $0 \leq n < 100$.
 planner | $\uparrow$
 | decimal (of 2X.....)
 | binary

$\phi \rightarrow 0$ via print Chr\$(27)"@";

OUT "Centronics" 8^{1/2} Bit OUT & F600, 32 aan
0 int

OUT *<port number>*, *<integer expression>*

```
OUT  &F8F4,&FF
```

COMMAND: Sends the value in the *integer expression* (in the range 0 to 255) OUT to the address specified in the *port number*.

Not a command to be used by the unwary.

Associated keywords: `INP`, `WAIT`

PAPER

PAPER[#<stream expression> ,]<ink>

```
10 MODE 0:PEN 0:INK 0,13
20 FOR p=1 TO 15
30 PAPER p:CLS
40 LOCATE 7,12:PRINT "paper";p
50 FOR t=1 TO 500:NEXT t,p
run.
```

COMMAND: Sets the background ink for characters. When characters are written to the text screen, the character cell is filled with the paper *ink* (in the range 0 to 15) before the character is written, (unless the transparent mode has been selected).

If the `stream expression` is omitted, the PAPER ink for stream #0 is assumed by default.

The number of different PAPER inks supported is dependent upon the screen MODE.

Associated keywords: GRAPHICS PAPER, INK, PEN

PEEK

PEEK (<address expression>)

```
10 MODE 1:ZONE 7
20 WINDOW 1,40,1,2:WINDOW #1,1,40,3,25
30 PRINT "memory-address"
40 LOCATE 20,1:PRINT "memory-contents"
50 FOR n=0 TO 65535
60 p=PEEK(n)
70 PRINT #1,n,"(&;HEX$(n);)";
80 PRINT #1,TAB(20);p,"(&;HEX$(p);)"
90 NEXT
run
```

FUNCTION: Reports the contents of the memory location specified in the <address expression> which should be in the range &0000 to &FFFF (0 to 65535). In all cases PEEK will return the value at the RAM address specified (not the ROM), and will be in the range &00 to &FF (0 to 255).

Associated keywords: POKE

PEN

PEN[#<stream expression>],[<ink>],[<background mode>]

```
10 MODE 0:PAPER 0:INK 0,13
20 FOR p=1 TO 15
30 PEN p:PRINT SPACES(47);"pen";p
40 FOR t=1 TO 500:NEXT t,p:GOTO 20
run
```

COMMAND: Sets the <ink> (in the range 0 to 15) to be used when writing to the given screen stream, (stream #0 if not specified). The <background mode> parameter can be set to transparent (1) or opaque (0).

Either of the last 2 parameters may be omitted, but not both. If a parameter is omitted, that particular setting is not changed.

Associated keywords: PAPER

PI

PI

```
PRINT PI
3.14159265
```

FUNCTION: Returns the value of the ratio between circumference and diameter of a circle.

Associated keywords: DEG, RAD

PLOT

PLOT <x co-ordinate>, <y co-ordinate>[, [<ink>][, <ink mode>]]

```
10 MODE 1:BORDER 0:PAPER 0:PEN 1
20 INK 0,0:INK 1,26:INK 2,13,26:DEG
30 FOR x=1 TO 360:ORIGIN 320,200
40 DRAW 50*COS(x),50*SIN(x),1
50 PLOT 100*COS(x),25*SIN(x):NEXT
60 ORIGIN 0,0:t=TIME+700:WHILE TIME<t
70 PLOT RND*640,RND*400:WEND
80 PLOT RND*640,RND*400,2
90 GOTO 90
run
```

COMMAND: Plots a point on the graphics screen, at the absolute position specified in the x,y co-ordinates. The <ink> in which to plot the point may be specified (in the range 0 to 15).

The optional <ink mode> determines how the ink being written interacts with that already on the graphics screen. The 4 <ink mode>s are:

0: Normal
1: XOR (eXclusive OR)
2: AND
3: OR

Associated keywords: GRAPHICS PEN, PLOT R

PLOTR

PLOTR <x offset> , <y offset> [, <ink>] [, <ink mode>]

```
10 REM use cursor keys to draw lines
20 BORDER 0:GRAPHICS PEN 1
30 MODE 1:PLOT 320,200
40 IF INKEY(0)=0 THEN PLOTR 0,1
50 IF INKEY(1)=0 THEN PLOTR 1,0
60 IF INKEY(2)=0 THEN PLOTR 0,-1
70 IF INKEY(8)=0 THEN PLOTR -1,0
80 IF INKEY(9)=0 THEN 30:REM copy=clear
90 GOTO 40
run
```

COMMAND: Plots a point on the graphics screen at the specified position <x offset> and <y offset>, relative to the current graphics cursor position. The <ink> in which to plot the point may be specified (in the range 0 to 15).

The optional <ink mode> determines how the ink being written interacts with that already on the graphics screen. The 4 <ink mode>s are:

- 0: Normal
- 1: XOR (eXclusive OR)
- 2: AND
- 3: OR

Associated keywords: GRAPHICS PEN, PLOT,

POKE

POKE <address expression> , <integer expression>

```
10 FOR m=49152 TO 65535
20 POKE m,100
30 NEXT
run
```

COMMAND: Writes the value of the <integer expression> (in the range 0 to 255) directly into the machine memory (RAM) at the specified <address expression>.

Not a command to be used by the unwary.

Associated keywords: PEEK

PRINTER status

IMP(&F5D) → 97 *immer angeschlossen*
→ 30 *wel " " "*

POS

POS(# <stream expression>)

```
10 MODE 1:BORDER 0:LOCATE 8,2
20 PRINT "use cursor left/right keys"
30 WINDOW 1,40,12,12:CURSOR 1,1
40 FOR n=1 TO 19:PRINT CHR$(9);:NEXT
50 IF INKEY(1)<>-1 THEN PRINT CHR$(9);
60 IF INKEY(8)<>-1 THEN PRINT CHR$(8);
70 LOCATE #1,2,24
80 PRINT #1,"text cursor ";
90 PRINT #1,"horizontal position =";
100 PRINT #1,POS(#0):GOTO 50
run
```

FUNCTION: Reports the current horizontal POSition of the text cursor relative to the left edge of the text window. The <stream expression> **MUST** be specified, and does NOT default to #0.

POS(#8) reports the current horizontal carriage position for the printer, where 1 is the extreme left hand edge.

POS(#9) reports the logical position in the disc file stream, i.e. the number of printing characters sent to the stream since the last carriage return.

Associated keywords: VPOS, WINDOW

PRINT

PRINT[# <stream expression> ,][<list of: print items>]

```
10 a$="small"
20 b$="this is a larger string"
30 PRINT a$;a$
40 PRINT a$,a$
50 PRINT
60 PRINT b$;b$
70 PRINT b$,b$
run
```

COMMAND: Prints the <list of: print item>s to the given stream, (to stream #0 if no <stream expression> is specified).

Note that when a semicolon ; is used to tell the computer to print the following <print item> next to the preceding item, BASIC first checks to see if the following <print item> can fit onto the same line. If not, it will be printed on a new line regardless of the semicolon.

Note also that when a comma , is used to tell the computer to print the following <print item> in the next print zone, BASIC first checks to see that the preceding item has not exceeded the length of the current zone. If it has, the following <print item> is printed in a further zone.

PRINT SPC PRINT TAB

PRINT[#<stream expression> ,][<list of: <print item>][;]
[SPC (<integer expression>)][<list of: <print item>]

PRINT[#<stream expression> ,][<list of: <print item>][;]
[TAB (<integer expression>)][<list of: <print item>]

```
10 PRINT "this is spc function"
20 FOR x=6 TO 15
30 PRINT SPC(5)"a";SPC(x)"b"
40 NEXT
50 PRINT "this is tab function"
60 FOR x=6 TO 15
70 PRINT TAB(5)"a";TAB(x)"b"
80 NEXT
run
```

SPC prints the number of spaces specified in the <integer expression>, and will print any following <print item> immediately next to the spaces, (assuming that the following <print item> will fit onto the line). Hence it is not necessary to terminate SPC with a semicolon.

TAB prints the number of spaces relative to the left edge of the text window, and will print any following <print item> immediately next to the spaces, (assuming that the following <print item> will fit onto the line). Hence it is not necessary to terminate TAB with a semicolon. If the current position is greater than the required position, then a carriage return is executed, followed by spaces to reach the required position on the next line.

PRINT USING

PRINT [# <stream expression>][<list of: <print item> < ; >]
[USING <format template>][<separator> <expression>]

```
10 FOR x=1 TO 10
20 n=100000*(RND↑5)
30 PRINT "goods";USING "#####,.##";n
40 NEXT
run
```

PRINT USING enables you to specify the print format of the expression returned by the PRINT command. This is achieved by specifying a <format template> to which the printed result must correspond. The <separator> is a comma or semicolon. The <format template> is a string expression which is constructed using the following 'format field specifiers':

Numeric Formats

Within the number:

- # Each # specifies a digit position.
Example template: #####
- . Specifies the position of the decimal point.
Example template: #####.##
- / (Specifies one digit position.) May appear BEFORE the decimal point only.
Specifies that digits before the decimal point are to be divided into groups of three (for thousands), separated by commas.
Example template: #####,.##

Around the number:

- ££ (Specifies two digit positions.) Specifies that a £ sign be printed immediately before the first digit or decimal point (after any leading sign). Note that the £ will occupy one of the digit positions.
Example template: ££#####,.##
- ** (Specifies two digit positions.) Specifies that any leading spaces be replaced by asterisks.
Example template: **#####,.##

-
- **f** (Specifies three digit positions.) Acts ** and ff options combined, i.e. leading * asterisks and f sign.
Example template: **f##### ,.##
 - \$\$** (Specifies two digit positions.) Specifies that a \$ sign be printed immediately before the first digit or decimal point (after any leading sign). Note that the \$ will occupy one of the digit positions.
Example template: \$##### ,.##
 - **\$** (Specifies three digit positions.) Acts as ** and \$\$ options combined, i.e. leading * asterisks and \$ sign.
Example template: **\$##### ,.##
 - +** Specifies that + or - is to be printed, as appropriate. If the + appears at the beginning of the template, the + sign is printed immediately before the the number (and any leading currency sign). If the + appears at the end of the template, the sign is printed after the number (and any exponent part).
Example template: +#### .####
 - The - sign may only appear at the END of a template. It specifies that - is to be printed after any negative number (and exponent part). If the number is positive, a space will be printed. A - sign is printed before a negative number by default, unless countermanded by the use of this template.
Example template: #### .#### -
 - ↑↑↑↑** Specifies that the number is to be printed using the exponent option. The ↑↑↑↑ in the template should appear AFTER the digit positions, but BEFORE any trailing + or - sign.
Example template: #.####↑↑↑↑+

The <format template> for a number may not exceed 20 characters. Numbers are rounded to the number of digits printed.

If the format template is too small for the input expression, for example:

```
PRINT USING "####";12345678
```

....the printed result is NOT shortened to fit the template, but is instead printed in its entirety, preceded by a % sign, to indicate 'format failure'.

String Formats

```
10 CLS:a$="abcdefghijklmnopqrst"
20 PRINT "input expression= ";a$
30 PRINT:PRINT "! specifier= ";
40 PRINT USING "!";a$
50 PRINT:PRINT "\spaces\ specifier= ";
60 PRINT USING "\          \";a$
70 PRINT:PRINT "& specifier= ";
80 PRINT USING "&";a$
90 GOTO 90
run
```

! Specifies that only the first character of the string is to be printed.
Example template: !

\<spaces>\
Specifies that only the first x characters of the string are to be printed, where x is equal to the length of the template (including the back-slashes).
Example template: \ \

& Specifies that the entire string is to be printed 'as is'.
Example template: &

The <format template> for a string may not exceed 255 characters.

Both numeric and string <format template>s may be represented by string variables, for example:

```
10 a$="f£#####,.##"
20 b$="!"
30 PRINT USING a$;12345.6789;
40 PRINT USING b$;"pence"
run
```

Further information concerning print formatting will be found in part 2 of the chapter entitled 'At your leisure....'.

Associated keywords: SPC, TAB, USING

RAD

RAD

RAD

COMMAND: Sets RADians mode of calculation. BASIC defaults to radians when the computer is switched on or reset, or when the commands **NEW**, **CLEAR**, or **LOAD**, **RUN**, etc, are issued.

Associated keywords: **ATN**, **COS**, **DEG**, **SIN**, **TAN**

RANDOMIZE

RANDOMIZE [numeric expression]

```
RANDOMIZE 123.456
PRINT RND
0.258852139
```

COMMAND: Randomizes the number 'seed' specified in the <numeric expression>. BASIC's random number generator produces a pseudo-random sequence in which each number depends on the previous number, commencing at a given number seed. The sequence is always the same. **RANDOMIZE** sets the new initial value for the random number generator either to the specified value, or to a value entered by the user if the <numeric expression> is omitted.

RANDOMIZE TIME produces a sequence that is difficult to repeat.

Associated keywords: **RND**

READ

READ <list of: variable>

```
10 FOR n=1 TO 8
20 READ a$,c
30 PRINT a$;" ";;SOUND 1,c:NEXT
40 DATA here,478,are,426,8,379,notes
50 DATA 358,of,319,a,284,musical,253,scale,239
run
```

COMMAND: Reads data from **DATA** statements and assigns it to variables, automatically stepping the 'pointer' to the next item in the **DATA** statement afterwards. The **RESTORE** command can be used to return the pointer to the beginning of a **DATA** statement.

Further information concerning data will be found in part 2 of the chapter entitled 'At your leisure.....'.

Associated keywords: **DATA**, **RESTORE**

RELEASE

RELEASE <sound channels>

```
10 SOUND 65,1000,100
20 PRINT "Press R to release the sound"
30 IF INKEY(50)=-1 THEN 30
40 RELEASE 1
run
```

COMMAND: Releases sound channels which are set to a 'hold' state in the SOUND command.

The parameter <sound channels> must yield an integer value in the range 1 to 7, which operates as follows:

- 1: Releases channel A
- 2: Releases channel B
- 3: Releases channel A and B
- 4: Releases channel C
- 5: Releases channel A and C
- 6: Releases channel B and C
- 7: Releases channel A and B and C

Further information concerning sound will be found in part 2 of the chapter entitled 'At your leisure....'.

Associated keywords: SOUND

REM

REM <rest of line>

```
10 REM Intergalactic Hyperspace Mega-Monster
   Invaders Deathchase by AMSOFT
20 REM Copyright AMSOFT 1985
```

COMMAND: Inserts a REMark into a program. The <rest of line> is ignored by BASIC, and may contain any characters, including colons : which normally separate statements.

A single quote character ' can be used in place of : REM in all applications EXCEPT in DATA statements, where the ' is treated as part of an unquoted string.

Associated keywords: none

REMAIN

REMAIN (<timer number>)

```
10 AFTER 500,1 GOSUB 40
20 AFTER 100,2 GOSUB 50
30 PRINT "program running":GOTO 30
40 REM this GOSUB-routine will not be called
   as it is disabled in line 80.
50 PRINT:PRINT "Timer 1 will now be ";
60 PRINT "disabled by REMAIN."
70 PRINT "Time-units remaining were:";
80 PRINT REMAIN(1)
run
```

FUNCTION: Returns the REMA INing count from the delay timer specified in <timer number> (in the range 0 to 3), and disables it.

Further information concerning interrupts will be found in part 2 of the chapter entitled 'At your leisure....'.

Associated keywords: AFTER, DI, EI, EVERY

RENUM

RENUM [<new line number>][,<old line number>][,<increment>]]

```
10 CLS
20 REM this will be line 123
30 REM this will be line 124
40 REM this will be line 125
RENUM 123,20,1
LIST
```

COMMAND: RENUMbers program lines.

The parameter <old line number> specifies the current existing line number at which renumbering is to commence. If <old line number> is omitted, renumbering will commence from the beginning of the program.

The parameter <new line number> specifies the new starting line number for the renumbered lines. If <new line number> is omitted, the renumbered program will start at line 10.

The parameter <increment> specifies the numeric step between each of the renumbered lines. If <increment> is omitted, the value of the numeric step will be 10.

RENUM takes care of all GOSUB, GOTO and other line calls. However, line number references within string expressions, such as those issued in KEY commands, are not altered; neither are line references within REM statements, nor the <line number expression> in a CHAIN or CHAIN MERGE command.

Line numbers are valid in the range 1 to 65535.

Associated keywords: DELETE, LIST

RESTORE

RESTORE [⟨line number⟩]

```
10 READ a$:PRINT a$;" ";
20 RESTORE 50
30 FOR t=1 TO 500:NEXT:GOTO 10
40 DATA restored data can be read again
50 DATA and again
run
```

COMMAND: Restores the position of the 'pointer' back to the beginning of the DATA statement specified in the optional ⟨line number⟩. Omitting this parameter restores the pointer back to the first DATA statement.

Further information concerning data will be found in part 2 of the chapter entitled 'At your leisure....'.

Associated keywords: DATA, READ

RESUME

RESUME [⟨line number⟩]

```
10 ON ERROR GOTO 60
20 FOR x=10 TO 0 STEP-1:PRINT 1/x:NEXT
30 END
40 PRINT "go here after error"
50 END
60 PRINT "error no.";ERR;"in line";ERL
70 RESUME 40
run
```

COMMAND: Resumes normal execution of a program after an error has been trapped and processed by an ON ERROR GOTO command. If the ⟨line number⟩ to RESUME at is not specified, the program will re-commence execution from the same line in which the error was first trapped. Try removing the ⟨line number⟩ parameter in the above example, then RUN again.

```
70 RESUME
run
```

Associated keywords: DERR, ERL, ERR, ERROR, ON ERROR GOTO, RESUME NEXT

RESUME NEXT

RESUME NEXT

```
10 ON ERROR GOTO 90
20 PRINT "press [ENTER] each time"
30 INPUT "1";a
40 INPUT "2";a
50 input "3";a REM syntax error!
60 INPUT "4";a
70 INPUT "5";a
80 END
90 PRINT "error no.";ERR;"in line";ERL
100 RESUME NEXT
run
```

COMMAND: Resumes normal execution of a program after an error has been trapped and processed by an ON ERROR GOTO command.

RESUME NEXT will re-commence execution from the line after that in which the error was first trapped.

Associated keywords: DERR, ERL, ERR, ERROR, ON ERROR GOTO, RESUME

RETURN

RETURN

```
10 GOSUB 50:PRINT "after the gosub":END
50 FOR n=1 TO 20
60 PRINT "sub-routine"
70 NEXT:PRINT
80 RETURN
run
```

COMMAND: Marks the end of a sub-routine. BASIC returns from the sub-routine to the statement immediately after the GOSUB command which invoked it.

Associated keywords: GOSUB

RIGHT\$

RIGHT\$(*string expression*, *required length*)

```
10 MODE 1:a$="CPC664 computer"
20 FOR n=1 TO 15:LOCATE 41-n,n
30 PRINT RIGHT$(a$,n)
40 NEXT
run
```

FUNCTION: Returns the number of characters (in the range 0 to 255) specified in the *required length* parameter, after extracting them from the RIGHT of the *string expression*. If the *string expression* is shorter than the *required length*, the whole *string expression* is returned.

Associated keywords: LEFT\$, MID\$

RND

RND[(*numeric expression*)]

```
10 RANDOMIZE
20 FOR x=1 TO -1 STEP -1
30 PRINT "rnd parameter=";x
40 FOR n=1 TO 6
50 PRINT RND(x)
60 NEXT n,x
run
```

FUNCTION: Returns the next RaNDom number in sequence if the *numeric expression* has a positive value or is not specified.

If the *numeric expression* yields a value of zero, RND returns a copy of the last random number generated.

If the *numeric expression* yields a negative value, a new random number sequence is started, the first number of which is returned.

Associated keywords: RANDOMIZE

ROUND

ROUND (<numeric expression> [, <decimals>])

```
10 FOR n=4 TO -4 STEP-1
20 PRINT ROUND (1234.5678,n),
30 PRINT "with integer expression";n
40 NEXT
run
```

FUNCTION: Rounds the <numeric expression> to a number of decimal places or power of ten specified in the <decimals> parameter. If <decimals> is less than zero, the <numeric expression> is rounded to give an absolute integer with <decimals> number of zeros before the decimal point.

Associated keywords: ABS, CINT, FIX, INT

RUN

RUN <string expression>

```
RUN "rointime.dem"
```

COMMAND: Loads a BASIC or binary program from disc and commences execution. Any previously loaded BASIC program is cleared from the memory.

Protected BASIC programs may be run directly in this manner.

Associated keywords: LOAD

RUN

RUN [<line number>]

```
RUN 200
```

COMMAND: Commences execution of the current BASIC program, from the specified <line number> parameter, or from the beginning of the program if the parameter is omitted. RUN resets the value of all current program variables to zero or null.

Protected programs may NOT be run in this manner, after loading.

Associated keywords: CONT, END, STOP

SAVE

SAVE <filename>[, <file type>][, <binary parameters>]

SAVE "discfile.xyz"

....saves the file in normal unprotected BASIC mode.

SAVE "discfile.xyz",P

....saves the file in Protected BASIC mode.

SAVE "discfile.xyz",A

....saves the file in ASCII mode.

SAVE "discfile.xyz",B,8000,3000,8001

....saves the file in Binary mode. In this example, saves the area of the computer's memory starting at address 8000; the length of the file being 3000 bytes; the optional entry point address being 8001.

COMMAND: Saves the program currently in the memory to disc. A Binary file is an area of memory saved to disc. The Binary parameters are:

<start address> , <file length> [, <entry point>]

The screen memory can be saved as a Binary file. This is known as a 'screen dump' and can be performed using the command:

SAVE "screen",B,&C000,&4000

Then, to load it back onto the screen:

LOAD "screen"

Associated keywords: CHAIN, CHAIN MERGE, LOAD, MERGE, RUN

SGN

SGN (<numeric expression>)

```
10 FOR n=200 TO -200 STEP-20
20 PRINT "SGN returns";
30 PRINT SGN(n);"for a value of";n
40 NEXT
run
```

FUNCTION: Determines the SiGN of the <numeric expression>. SGN returns -1 if <numeric expression> is less than zero, returns 0 if <numeric expression> equals zero, and returns 1 if <numeric expression> is greater than zero.

Associated keywords: ABS

SIN

SIN (<numeric expression>)

```
10 CLS:DEG:ORIGIN 0,200
20 FOR n=0 TO 720
30 y=SIN(n)
40 PLOT n*640/720,198*y:NEXT
50 GOTO 50
run
```

FUNCTION: Calculates the SIne of the <numeric expression>.

Note that DEG and RAD can be used to force the result of the above calculation to degrees or radians respectively.

Associated keywords: ATN, COS, DEG, RAD, TAN

SOUND

SOUND <channel status> [, <tone period>] [, <duration>] [, <volume>
[, <volume envelope>] [, <tone envelope>] [, <noise period>]]]]]

```
10 FOR z=0 TO 4095
20 SOUND 1,z,1,12
30 NEXT
run
```

COMMAND: Programs a sound. The command takes the following parameters:

Parameter 1: <channel status>

The <channel status> parameter must yield an integer in the range 1 to 255. The parameter is bit significant, with each bit of the binary value of <channel status> signifying the following:

Bit 0 : (decimal 1) send sound to channel A (Least significant bit)
Bit 1 : (decimal 2) send sound to channel B
Bit 2 : (decimal 4) send sound to channel C
Bit 3 : (decimal 8) rendezvous with channel A
Bit 4 : (decimal 16) rendezvous with channel B
Bit 5 : (decimal 32) rendezvous with channel C
Bit 6 : (decimal 64) hold sound channel
Bit 7 : (decimal 128) flush sound channel (Most significant bit)

Hence a <channel status> parameter of 68 for example, would mean:

Send to channel C (4), with a hold state (64).

Parameter 2: <tone period>

This parameter defines the pitch of the sound, or in other words, 'what note it is' (i.e. do re mi fa so, etc). Each note has a set number, and this number is the <tone period>. See the chapter entitled 'For your reference....'.

Parameter 3: <duration>

This parameter sets the length of the sound, in other words, 'how long it lasts'. The parameter works in units of 0.01 (one hundredth) of a second, and if you don't specify the <duration>, the computer will default to 20 (one fifth of a second).

If the `<duration>` parameter is zero, the sound will last until the end of the specified volume envelope.

If the `<duration>` parameter is negative, the specified volume envelope is to be repeated `ABS (<duration>)` times.

Parameter 4: `<volume>`

This parameter specifies the starting volume of a note. The number is in the range 0 to 15. A `<volume>` figure of 0 is off, while 15 is maximum. If no number is specified, the computer will default to 12.

Parameter 5: `<volume envelope>`

To make the volume vary within the duration of the note, you can specify a volume envelope using the separate command `ENV`. You can in fact create up to 15 different volume envelopes referenced in the range 1 to 15. The `<volume envelope>` parameter calls up the appropriate volume envelope reference number for use in the `SOUND` command.

Refer to the description of the `ENV` command.

Parameter 6: `<tone envelope>`

To make the tone or pitch vary within the duration of the note, you can specify a tone envelope using the separate command `ENT`. You can in fact create up to 15 different tone envelopes referenced in the range 1 to 15. The `<tone envelope>` parameter calls up the appropriate tone envelope reference number for use in the `SOUND` command. If you have specified a negative envelope number in the `ENT` command, use the absolute value of that number (i.e. without the negative sign) in this `<tone envelope>` parameter of the `SOUND` command.

Refer to the description of the `ENT` command.

Parameter 7: `<noise>`

A range of white noise is available, which can be switched off or added to the sound by varying the `<noise>` parameter between 0 and 31.

Further information concerning sound will be found in part 2 of the chapter entitled 'At your leisure....'.

Associated keywords: `ENT`, `ENV`, `ON SQ GOSUB`, `RELEASE`, `SQ`

SPACES

SPACES (⟨integer expression⟩)

```
10 MODE 1
20 PRINT "Put 9 spaces between me";
30 PRINT SPACES(9);
40 PRINT "and you!"
run
```

FUNCTION: Creates a string of spaces of the given length, (in the range 0 to 255) specified in the ⟨integer expression⟩.

Associated keywords: **SPC**, **STRING\$**, **TAB**

SPC

(See **PRINT SPC**)

SPEED INK

SPEED INK ⟨period 1⟩, ⟨period 2⟩

```
10 BORDER 7,18
20 FOR i=30 TO 1 STEP-1
30 SPEED INK i,i
40 FOR t=1 TO 700:NEXT t,i
run
```

COMMAND: Sets the rate of alternation between two ink colours specified in an **INK** or **BORDER** command. ⟨period 1⟩ specifies the time, in units of 0.02 (fiftieths) second for the first colour to be used; ⟨period 2⟩ sets the time for the second colour.

You must exercise careful judgement to avoid mesmeric effects when selecting colours and repeat rates!

Associated keywords: **BORDER**, **INK**

SPEED KEY

SPEED KEY <start delay>, <repeat period>

```
10 CLS:FOR k=7 TO 1 STEP-2
20 PRINT "type your name, then [ENTER]"
30 SPEED KEY k,k
40 LINE INPUT a$:NEXT
50 PRINT "That's a funny name!"
run
```

COMMAND: Sets the rate of keyboard auto repeat. The <start delay> parameter specifies the time, in units of 0.02 (fiftieths) second before auto repeat starts. The <repeat period> parameter sets the interval between each auto repeat of a key.

SPEED KEY will operate only on keys which auto repeat by default, or which have been set to auto repeat by the **KEY DEF** command.

When intending to use small values of <start delay>, it is wise to pre-program one of the numeric keys to return the keyboard to its default **SPEED KEY** setting of 30,2. This can be achieved by the command:

```
KEY 0,"SPEED KEY 30,2"+CHR$(13)
```

....which will reset **SPEED KEY** to its default values when the 0 key on the numeric keypad is pressed.

Associated keywords: **KEY DEF**

SPEED WRITE

SPEED WRITE <integer expression>

```
SPEED WRITE 1
```

COMMAND: Sets the speed at which data is to be saved or written to a cassette unit (if connected). The cassette can be written at either 2000 baud (bits per second) if the <integer expression> is 1, or at the default rate of 1000 baud if the <integer expression> is 0. When loading a file from tape, the computer automatically selects the correct reading speed as it loads.

For higher data reliability, it is recommended that you use **SPEED WRITE 0** (default).

The **SPEED WRITE** command has no effect upon disc operation.

Associated keywords: **OPENOUT, SAVE**

SQ

SQ (<channel>)

```
10 SOUND 65,100,100
20 PRINT SQ(1)
run
67
```

FUNCTION: Reports the state of the **Sound Queue** for the specified <channel> which must be an integer expression, yielding one of the values:

1 : for channel A
2 : for channel B
4 : for channel C

The **SQ** function returns an bit significant integer, comprising the following bit settings:

Bits 0, 1, and 2 : the number of free entries in the queue
Bits 3, 4, and 5 : the rendezvous state at the head of this queue
Bit 6 : the head of the queue is held
Bit 7 : the channel is currently active

....where Bit 0 is the least significant bit, and Bit 7 is the most significant bit.

It can be seen therefore, that if Bit 6 is set, Bit 7 cannot be set, and vice versa. Similarly if Bits 3, 4, or 5 are set, Bits 6 and 7 cannot be set.

Further information concerning sound will be found in part 2 of the chapter entitled 'At your leisure....'.

Associated keywords: **ON SQ GOSUB, SOUND**

SQR

SQR (<numeric expression>)

```
PRINT SQR(9)
3
```

FUNCTION: Returns the **S**quare Root of the specified <numeric expression>.

Associated keywords: **none**

STEP

(See FOR)

STOP

STOP

```
10 FOR n=1 TO 30:PRINT n:NEXT
20 STOP
30 FOR n=31 TO 60:PRINT n:NEXT
run
cont
```

COMMAND: Stops execution of a program, but leaves BASIC in a state where the program can be resumed by the CONT command. STOP may be used to interrupt the program at a particular point when de-bugging.

Associated keywords: CONT, END

STR\$

STR\$ (<numeric expression>)

```
10 a=&FF :REM 255 hex
20 b=&X1111 : REM 15 binary
30 c$="***"
40 PRINT c$+STR$(a+b)+c$
run
*** 270***
```

FUNCTION: Converts the <numeric expression> to a decimal STRing representation.

Associated keywords: BIN\$, DEC\$, HEX\$, VAL

STRING\$

STRING\$(*<length>*,*<character specifier>*)

```
PRINT STRING$(40,"*")
*****
```

FUNCTION: Returns a string expression consisting of the specified character repeated the number of times (in the range 0 to 255) specified in the *<length>*. Note that the above example could be entered as:

```
PRINT STRING$(40,42)
*****
```

....where the *<character specifier>* 42 refers to the ASCII value of the character * i.e. equivalent to PRINT STRING\$(40,CHR\$(42)).

Associated keywords: SPACE\$

SWAP

(See WINDOW SWAP)

SYMBOL

SYMBOL *<character number>*,*<list of:row>*

```
10 MODE 1:SYMBOL AFTER 105
20 row1=255:REM binary 1111111
30 row2=129:REM binary 10000001
40 row3=189:REM binary 10111101
50 row4=153:REM binary 10011001
60 row5=153:REM binary 10011001
70 row6=189:REM binary 10111101
80 row7=129:REM binary 10000001
90 row8=255:REM binary 1111111
100 PRINT "Line 110 re-defines the letter i (105).
      Type in some i's and see! Then list the program."
110 SYMBOL 105,row1,row2,row3,row4,row5,row6,row7,row8
run
```

COMMAND: Re-defines the shape of a character on the screen. Each of the parameters must yield an integer in the range 0 to 255.

To allocate space in the CPC664's memory for a newly defined character, the computer must first be prepared by issuing the command:

`SYMBOL AFTER x`

....where x is equal to or less than the character number you wish to re-define.

The command `SYMBOL` is then issued, followed firstly by the character number x.

Regardless of whether or not the value of x specifies a character which is directly typeable at the keyboard, the re-defined character can be printed on the screen by issuing the command:

`PRINT CHR$(x)`

After `SYMBOL x`, there are up to 8 parameters which specify the 8 individual horizontal rows of the character, starting from the top. Each of the parameters can be in the range 0 to 255. The binary representation of each of the 8 parameters determines the pattern of that particular row in the finished character.

For example, if the first of the 8 parameters is 1, then the top row of the character has a binary representation of 00000001. Where the 1 appears, the section of the character is printed in the `PEN` colour; where a 0 appears, the section of the character is not visible because it is printed in the `PAPER` colour. Therefore the top row of this newly defined character has a dot in the top right hand corner. Continuing this example, we will specify the other 7 parameters as 3,7,15,31,63,0,0 - the binary representation of all 8 parameters then being:

parameter(row) 1: 00000001 binary: (decimal 1)
parameter(row) 2: 00000011 binary: (decimal 3)
parameter(row) 3: 00000111 binary: (decimal 7)
parameter(row) 4: 00001111 binary: (decimal 15)
parameter(row) 5: 00011111 binary: (decimal 31)
parameter(row) 6: 00111111 binary: (decimal 63)
parameter(row) 7: 00000000 binary: (decimal 0)
parameter(row) 8: 00000000 binary: (decimal 0)

Looking at the binary representation of the above 8 parameters, it should be possible to see what the shape of the new character is going to be like. Let's assign those parameters to character number 255 using the command:

`SYMBOL 255,1,3,7,15,31,63,0,0`

Note that the value of 0 appearing in the 2 final parameters means that you need only type in:

```
SYMBOL 255,1,3,7,15,31,63
```

Note that you can enter the parameters in binary to save you converting the 'pattern' of the symbol that you've created into decimal form. (Remember to use the &X binary prefix.) For example:

```
SYMBOL 255,&X00000001,&X00000011,&X00000111,  
&X00001111,&X00011111,&X00111111
```

....Now, to see the character:

```
PRINT CHR$(255)
```

Assigning the above parameters to a typeable character on the keyboard would result in the new character appearing whenever the appropriate key is pressed, or wherever the previous character would have been printed. Furthermore, BASIC will not reject this new character as incomprehensible, but will regard it as the equivalent of the previous character.

Further information concerning user-defined characters will be found in part 2 of the chapter entitled 'At your leisure....'.

Associated keywords: HIMEM, MEMORY, SYMBOL AFTER

SYMBOL AFTER

SYMBOL AFTER <integer expression>

```
10 CLS  
20 SYMBOL AFTER 115  
30 PRINT "Line 40 re-defines the s ";  
40 SYMBOL 115,0,56,64,64,48,8,8,112  
50 PRINT "to s"  
60 PRINT "Cancel this definition of s,"  
70 PRINT "by typing: SYMBOL AFTER 240"  
run
```

COMMAND: Sets the number of permissible user defined characters (in the range 0 to 256). The default setting is 240, giving 16 user defined characters (from 240 to 255). If the <integer expression> is 32, then all characters from 32 to 255 are re-definable. SYMBOL AFTER 256 permits no characters to be re-definable.

Whenever a SYMBOL AFTER command is executed, all user defined characters are reset to their default conditions.

SYMBOL AFTER will NOT operate if invoked AFTER the value of **HIMEM** has been altered using the **MEMORY** command, or by the opening of a file buffer with **OPENIN** or **OPENOUT**. Under such circumstances, an 'Improper argument' error (5) will be reported, (unless the previous state was **SYMBOL AFTER 256**).

Further information concerning user-defined characters will be found in part 2 of the chapter entitled 'At your leisure....'.

Associated keywords: **HIMEM, MEMORY, SYMBOL**

TAB

(See **PRINT TAB**)

TAG

αβ
↓
PRINT a\$;

TAG[#<stream expression>]

```
10 INPUT "enter your name";a$:CLS
20 PRINT "You certainly get around ";a$
30 TAG
40 x=LEN(a$)*17:y=50+RND*300:MOVE -x,y
50 FOR f=-x TO 640 STEP RND*7+3
60 MOVE f,y:PRINT " ";a$;:FRAME:NEXT
70 FOR b=640 TO -x STEP-RND*7+3
80 MOVE b,y:PRINT a$;" ";:FRAME:NEXT
90 GOTO 40
run
```

COMMAND: Sends any text specified for the given <stream expression> to be printed at the graphics cursor position. This allows text and symbols to be mixed with graphics, or moved pixel by pixel as opposed to character by character. The <stream expression> defaults to #0 if omitted.

The top left of the character cell is **TAGged (Text At Graphics)** to the graphics cursor, and non-printing control characters (e.g. line feed and carriage return) will display if the **PRINT** statement is not terminated by a semicolon.

In the default stream (#0), **BASIC** will switch off **TAG** when returning to direct mode.

Associated keywords: **TAG OFF**

TAGOFF

TAGOFF [#<stream expression>]

```
10 MODE 2:TAG :REM Text At Graphics-on
20 year=1984:FOR x=1 TO 640 STEP 70
30 MOVE x,400:DRAW 0,-350
40 year=year+1:PRINT year;:NEXT
50 TAGOFF :REM Text At Graphics-OFF
60 LOCATE 34,25:PRINT "Yearly figures"
70 GOTO 70
run
```

COMMAND: Cancels TAG (Text At Graphics) for the given <stream expression> (stream #0 if not specified), and re-directs text to the previous text cursor position used before TAG was invoked.

Associated keywords: TAG

TAN

TAN (<numeric expression>)

```
PRINT TAN(45)
1.61977519
```

FUNCTION: Calculates the TAngent of the <numeric expression>, which must be in the range -200000 to +200000.

Note that DEG and RAD can be used to force the result of the above calculation to degrees or radians respectively.

Associated keywords: ATN, COS, DEG, RAD, SIN

TEST

TEST (⟨x co-ordinate⟩,⟨y co-ordinate⟩)

```
10 CLS
20 PRINT "You are using pen number";
30 PRINT TEST(10,386)
40 PRINT "Try changing PENS and MODEs";
50 PRINT "....then RUN again."
run
```

FUNCTION: Moves the graphics cursor to the absolute position specified by the ⟨x and y co-ordinate⟩s, and reports the value of the ink at the new location.

Associated keywords: MOVE, MOVER, TESTR, XPOS, YPOS

TESTR

TESTR (⟨x offset⟩,⟨y offset⟩)

```
10 MODE 0:FOR x=1 TO 15:LOCATE 1,x
20 PEN x:PRINT STRING$(10,143);:NEXT
30 MOVE 200,400:PEN 1
40 FOR n=1 TO 23:LOCATE 12,n
50 PRINT "pen";TESTR(0,-16):NEXT
run
```

FUNCTION: Moves the graphics cursor by the amount specified in the ⟨x and y offset⟩s relative to its current position, and reports the value of the ink at the new location.

Associated keywords: MOVE, MOVER, TEST, XPOS, YPOS

THEN

(See I F)

TIME

TIME

```
10 CLS:REM clock
20 INPUT "hour";hour
30 INPUT "minute";minute
40 INPUT "second";second
50 CLS:datum=INT(TIME/300)
60 WHILE hour<13
70 WHILE minute <60
80 WHILE tick<60
90 tick=(INT(TIME/300)-datum)+second
100 LOCATE 1,1
110 PRINT USING "## ";hour,minute,tick
120 WEND
130 tick=0:second=0:minute=minute+1
140 GOTO 50
150 WEND
160 minute=0:hour=hour+1
170 WEND
180 hour=1
190 GOTO 60
run
```

FUNCTION: Reports the elapsed time since the computer was last switched-on or reset, (excluding periods when reading or writing to disc).

Each second of real time is equal to the returned value: `TIME/300`.

Associated keywords: `AFTER`, `EVERY`, `WEND`, `WHILE`

TO

(See `FOR`)

TROFF

TRON

TROFF

TRON

```
10 TROFF:PRINT:PRINT "TRace-OFF"
20 FOR n=1 TO 8
30 PRINT "program running":NEXT
40 IF f=1 THEN END
50 TRON:PRINT:PRINT "TRace-ON"
60 f=1:GOTO 20
run
```

COMMAND: Traces the execution of a program by printing each line number before carrying it out. The line number appears inside square brackets [].

TRON switches T Race ON; TROFF switches T Race OFF.

The facility is particularly useful for studying the sequence of program line execution just before an error occurs.

Associated keywords: **none**

UNT

UNT (<address expression>)

```
PRINT UNT(&FF66)
-154
```

COMMAND: Returns an integer in the range -32768 to +32767 which is the twos-complement equivalent of the unsigned value of the <address expression>.

Associated keywords: CINT, FIX, INT, ROUND

UPPER\$

UPPER\$ (<string expression>)

```
10 CLS:a$="my, how you've grown!"
20 PRINT UPPER$(a$)
run
```

FUNCTION: Returns a new string expression which is a copy of the specified <string expression> but in which all alphabetic characters in the range A to Z are converted to upper case. The function is useful for processing input which may come in mixed upper/lower case.

Associated keywords: LOWER\$

USING

(See PRINT USING)

VAL

VAL (<string expression>)

```
10 CLS:PRINT "I know my times tables!"
20 PRINT:PRINT "Press a key (1 to 9)"
30 a$=INKEY$:IF a$="" THEN 30
40 n=VAL(a$):IF n<1 OR n>9 THEN 30
50 FOR x=1 TO 12
60 PRINT n;"X";x;"=";n*x
70 NEXT:GOTO 20
run
```

FUNCTION: Returns the numeric VA Lue, (including any negative sign and decimal point) of the first character(s) in the specified <string expression>.

If the first character is not a number, then 0 is returned. If the first character is a negative sign or decimal point followed by non-numeric characters, a 'Type mismatch' error (13) will be reported.

Associated keywords: STR\$

VPOS

POS ←

VPOS (#<stream expression>)

```
10 MODE 1:BORDER 0:LOCATE 8,2
20 PRINT "use cursor up/down keys"
30 WINDOW 39,39,1,25:CORSOR 1,1
40 LOCATE 1,13
50 IF INKEY(0)<>-1 THEN PRINT CHR$(11);
60 IF INKEY(2)<>-1 THEN PRINT CHR$(10);
70 LOCATE #1,3,24
80 PRINT #1,"text cursor ";
90 PRINT #1,"vertical position =";
100 PRINT #1,VPOS(#0):GOTO 50
run
```

FUNCTION: Reports the current Vertical POS ition of the text cursor relative to the top of the text window. The <stream expression> MUST be specified, and does NOT default to #0.

Associated keywords: POS, WINDOW

*XPOS graphich
cur,*

WAIT

WAIT <port number>[,<mask>[,<inversion>]

WAIT &FF34,20,25

COMMAND: Waits until the specified I/O <port number> returns a particular value in the range 0 to 255. BASIC loops whilst reading the I/O port. The value read is eXclusive ORed with the <inversion> and then ANDed with the <mask> until a non-zero result occurs.

BASIC will wait indefinitely until the required condition occurs.

Not a command to be used by the unwary.

Associated keywords: INP, OUT

WEND

WEND

WEND

COMMAND: Marks the end of the body of program which is to be executed within the WHILE loop. WEND automatically selects the WHILE command it is to be associated with.

Associated keywords: TIME, WHILE

WHILE

WHILE <logical expression>

```
10 CLS:PRINT "Ten second timer":t=TIME
20 WHILE TIME<t+3000
30 SOUND 1,0,100,15
40 WEND:SOUND 129,40,30,15
run
```

COMMAND: Repeatedly executes a body of program while a given condition is true. The WHILE command defines the head of the loop, and specifies the condition in the <logical expression>.

Associated keywords: TIME, WEND

WIDTH

WIDTH <integer expression>

WIDTH 40

COMMAND: Tells BASIC how many characters per line are to be printed when a printer is connected. BASIC will then send the extra carriage return/line feed at the appropriate time.

The computer assumes a default value of 132 unless a WIDTH command is specified.

The command WIDTH 255 suppresses the extra carriage return/line feed altogether, allowing printing to be 'line wrapped' by the printer. Note that carriage return/line feed will still be generated by a PRINT command that isn't terminated by a semicolon or comma.

Associated keywords: POS

WINDOW

WINDOW[#<stream expression> ,<left>,<right>,<top>,<bottom>]

```
10 MODE 0:BORDER 0:REM testcard
20 INK 0,0:INK 1,25:INK 2,23:INK 3,21
30 INK 4,17:INK 5,6:INK 6,2:INK 7,26
40 PAPER 0:CLS
50 PAPER 1:WINDOW 2,4,1,18:CLS
60 PAPER 2:WINDOW 5,7,1,18:CLS
70 PAPER 3:WINDOW 8,10,1,18:CLS
80 PAPER 4:WINDOW 11,13,1,18:CLS
90 PAPER 5:WINDOW 14,16,1,18:CLS
100 PAPER 6:WINDOW 17,19,1,18:CLS
110 PAPER 7:WINDOW 2,19,19,25:CLS
120 GOTO 120
run
```

COMMAND: Specifies the dimensions of a text stream (WINDOW) on the screen. The values of the parameters <left>,<right>,<top>, and <bottom> should correspond with the inclusive screen character locations consistent with the screen MODE in use.

If the <stream expression> is not specified, BASIC defaults to stream #0.

Further information concerning windows will be found in part 2 of the chapter entitled 'At your leisure....'.

Associated keywords: WINDOW SWAP

WINDOW SWAP

WINDOW SWAP <stream expression>, <stream expression>

```
10 MODE 1:INK 1,24:INK 2,9:INK 3,6
20 WINDOW 21,40,13,25:PAPER 3
30 WINDOW #1,1,20,1,12:PAPER #1,2
40 CLS:PRINT " window number 0"
50 CLS #1:PRINT #1," window number 1"
60 LOCATE 1,6
70 PRINT " red window (0)";SPC(2)
80 LOCATE #1,1,6
90 PRINT #1," green window (1)"
100 FOR t=1 TO 1000:NEXT
110 WINDOW SWAP 0,1:GOTO 60
run
```

COMMAND: Swaps the text window specified in the first <stream expression> with that specified in the second <stream expression>.

Both <stream expression>s must be specified, and in this case should NOT be preceded by a # stream director.

The command may be used to re-direct messages produced by BASIC, which are normally always sent to stream #0.

Further information concerning windows will be found in part 2 of the chapter entitled 'At your leisure....'.

Associated keywords: WINDOW

WRITE

WRITE [# <stream expression>], [(<write list>)]

```
10 REM write variables onto disc
20 INPUT "give me a number variable";a
30 INPUT "give me a string variable";a$
40 OPENOUT "datafile"
50 WRITE #9,a,a$
60 CLOSEOUT:PRINT "Data saved onto disc"
run
```

COMMAND: Writes the values of the items in the <write list> to the stream specified in the <stream expression>. Items written will be separated by commas; strings will be enclosed by double-quotes.

In this example the values of the variables which you input, are written to stream #9 (the disc stream).

(To recall the values of those variables from disc, it would be necessary to use a program such as follows:)

```
10 REM retrieve variables from disc
20 OPENIN "datafile":INPUT #9,a,a$
30 CLOSEIN:PRINT "The 2 values are:"
40 PRINT:PRINT a,a$
run
```

Associated keywords: INPUT, LINE INPUT

XOR

⟨argument⟩ XOR ⟨argument⟩

```
IF "alan"<"bob" XOR "dog">"cat" THEN PRINT "correct" ELSE PRINT "wrong"
wrong
```

```
IF "bob"<"alan" XOR "cat">"dog" THEN PRINT "correct" ELSE PRINT "wrong"
wrong
```

```
IF "alan"<"bob" XOR "cat">"dog" THEN PRINT "correct" ELSE PRINT "wrong"
correct
```

```
----
```

```
PRINT 1 XOR 1
0
PRINT 0 XOR 0
0
PRINT 1 XOR 0
1
```

OPERATOR: Performs bit-wise boolean operation on integers. Result is 1 unless both argument bits are the same - eXclusive OR.

Further information concerning logic will be found in part 2 of the chapter entitled 'At your leisure....'.

Associated keywords: AND, OR, NOT

XPOS

XPOS

```
10 MODE 1: DRAW 320,200
20 PRINT "graphics cursor X P0Sition=";
30 PRINT XPOS
run
```

FUNCTION: Reports the current horizontal (X) P0Sition of the graphics cursor.

Associated keywords: **MOVE, MOVER, ORIGIN, YPOS**

YPOS

YPOS

```
10 MODE 1: DRAW 320,200
20 PRINT "graphics cursor Y P0Sition=";
30 PRINT YPOS
run
```

FUNCTION: Reports the current vertical (Y) P0Sition of the graphics cursor.

Associated keywords: **MOVE, MOVER, ORIGIN, XPOS**

ZONE

ZONE <integer expression>

```
10 CLS: FOR z=2 TO 20
20 ZONE z
30 PRINT "X", "X ZONE =", z: NEXT
run
```

COMMAND: Changes the width of the print zone (specified in **PRINT** statements by using a comma between print items). The default setting of the print zone is 13 columns, but may be changed as specified in the <integer expression> in the range 1 to 255.

Associated keywords: **PRINT**

Chapter 4

Using discs and cassettes

Part 1: Discs

Making working discs

This section discusses how to make discs to use from day to day, and introduces some facilities of CP/M and its Utility programs.

Subjects covered:

- ★ Making a backup Master System/Utilities disc.
- ★ Constructing a Working Systems/Utility disc.
- ★ Operating with a BASIC only disc.
- ★ Installing a Turnkey AMSTRAD BASIC application.
- ★ Installing a Turnkey CP/M application.

Part 7 of the Foundation course described how to format a blank system disc, which you can use for BASIC and games, as well as CP/M.

Part 10 of the Foundation course showed you how to make exact copies of discs with the DISCCOPY or COPYDISC programs.

This section considers how to make discs with the programs that you want on them.

Backup Master Disc

It is most important to make a copy of the Master System/Utility disc provided with your computer, and keep the original safe - it will be very costly to replace if damaged! Remember that the disc supplied has two sides; the System/Utilities side and the Dr. LOGO side. Every disc, in fact, has two sides and you are free to use either side for any purposes.

Remember that if you are using a new blank disc to copy onto, the DISCCOPY and COPYDISC programs will format for you as well as doing the copying.

A working SYSTEM/UTILITY disc

You will find that, as well as making a day-to-day copy of your Master System/Utility disc and Dr. LOGO disc, it is most convenient if you make a 'working utility disc' containing a few of the programs from your Master System/Utility disc, that you use the most. This will still leave plenty of room for your programs. Should you require to run any other utility programs, then you can always obtain them from your copy of the Master System/Utility disc. All the more commonly used programs will however, be at your fingertips.

First use CP/M and the FORMAT program to create a new blank disc. Then use the CP/M program FILECOPY to copy each program that you want onto the new disc. Experience will tell you which programs you'll want to have handy. In this example we have chosen:

```
AMSDOS.COM  
FILECOPY.COM  
DISCCOPY.COM
```

Using your copy of the Master System/Utilities disc as the CP/M system disc, select CP/M, and at the A> prompt type:

```
FILECOPY FILECOPY.COM
```

....and follow the displayed instructions. (The SOURCE disc is the one initially in the drive, and the DESTINATION disc is the new disc that you are creating). When copying has finished, copy the other two programs by the commands:

```
FILECOPY AMSDOS.COM  
FILECOPY DISCCOPY.COM
```

Once you have made one working system/utilities disc with your selection upon it, you can prepare the second side of that disc or both sides of other discs simply by using the DISCCOPY or COPYDISC programs. Take note however, of the copying restrictions laid down in the End User Licence Agreement, towards the end of this manual.

A BASIC only disc

Tracks 0 and 1 of a system disc are reserved by CP/M and cannot be used by you. If you want to use all the space on the disc for games, programs or data in BASIC, and NEVER intend to use CP/M or ANY of the CP/M utilities on that disc, it can be formatted without the CP/M system tracks. It is then called a data disc.

The disc must be formatted using an option of the format program, by typing in:

FORMAT D

To copy programs onto this type of disc, you must use **FILECOPY** (loaded from a system disc) or **LOAD** and **SAVE** them from **BASIC**. In a two drive system (i.e. if you have connected an additional disc drive to the computer), it is possible to use the **CP/M** utility **PIP**.

The programs **COPYDISC** and **DISCCOPY** will format the destination disc to the same as the source.

Turnkey AMSTRAD BASIC discs

If you buy an application program written in **AMSTRAD BASIC** for the **CPC664**, it should be ready to operate when you switch on. (The expression 'turnkey' comes from the days when all small computers had a key-operated power switch). All you have to do is install it on a suitable working disc.

Turnkey BASIC using disc supplied

Simply copy the master disc with **COPYDISC** or **DISCCOPY**, preserve the master disc and use the copy. Follow the instructions provided to run the program. If you require any additional programs from your Master System/Utility disc, use **FILECOPY** to transfer them.

Turnkey BASIC onto your Working disc

In this case copy the supplied programs onto your existing disc.

Type:

FILECOPY *.*

....and follow the instructions. In particular answer **N** to the question:

Ambiguous file name: Confirm individual files (Y/N) ?

The **FILECOPY** program will inform you of the filenames as they are copied. You can then run the new application program from your upgraded working disc.

Turnkey CP/M Discs

The CP/M operating system allows you to load and run an immense library of software which has already been written for personal computers that support CP/M. The fundamental 'logic' of these programs has already been devised; all that is required to use them on your CPC664 is to establish them on a suitable disc, and maybe to inform them of the particular method that the CPC664 uses to operate the screen.

A set of programs on one disc designed to fulfil a specific application is called a 'package'. These packages are normally designed to work on a large range of different computers, each of which has its own size of screen and way of moving the cursor around. Sometimes the package you buy will have already been 'installed' for the AMSTRAD system, or cater for it by offering a menu of alternative installations. In these cases, simply follow the instructions provided with the software. If the package does not have an AMSTRAD variant built in, then the section ahead entitled 'Configuring a CP/M Program' indicates the commands that can be sent to the CPC664 screen to produce the sorts of effects that packages require. Normally the installation, or customisation, procedure will involve typing in the relevant codes when requested to. Again, follow the instructions provided with the package.

The software you have purchased must be on a disc suitable for use in this system. Almost every different computer uses a different form of disc. Although many have the same size of disc, this does not necessarily mean that there is any compatibility between one and another in the information contained on them. Ask your supplier for an AMSTRAD 3 inch version.

Creating a Turnkey CP/M System disc

It is wise to preserve the original master disc *containing the new software* and transport the programs to another disc.

Although the instructions below are for the standard system with one disc drive, it is generally simpler to follow these instructions even if you have a 2-drive system (by ignoring the second drive).

Firstly format a new system disc. Then copy all the programs from your master package disc using `FILECOPY` from your System/Utility disc. Type:

```
FILECOPY *.*
```

...and follow the instructions on the screen. In particular answer `N` to the question:

```
Ambiguous filename: Confirm individual files (Y/N) ?
```

The **FILECOPY** program will inform you of the filenames as they are copied.

When the **FILECOPY** program has finished, you will have a working copy of the turnkey disc. If you require any Utilities, copy them from your System/Utilities disc using **FILECOPY**.

Configuring a CP/M Program

The CPC664 supports a wide range of control codes suitable for customising a software package to run with CP/M. Most data-processing and many other packages require to be able to print messages at any part of the screen, to accept input from any part of the screen and to generally understand cursor controls.

If your package has already been customised for the AMSTRAD system, then you need not concern yourself further.

Configuring the Output from the package

The installation procedure for a package will normally consist of running a special program (often called **INSTAL**) which will ask a number of questions about the parameters of the CPC664 screen. The answers should be derived from the table below, which is an extract from the AMSTRAD CPC664 BASIC reference Manual SOFT 945:

Value Hex	Value Decimal	Operation
&07	7	Sound Bleeper
&08	8	Move cursor back one position.
&09	9	Move cursor forward one position.
&0A	10	Move cursor down one line.
&0B	11	Move cursor up one line.
&0C	12	Home cursor and clear screen.
&0D	13	Move cursor to left edge of window on current line.
&10	16	Delete character at cursor position.
&11	17	Clear from left edge of window to and including the cursor position.
&12	18	Clear from and including the current cursor position to right edge of window.
&13	19	Clear from start of window to and including the current cursor position.
&14	20	Clear from and including the current cursor position to end of window.
&18	24	Toggle into/out-of inverse video.
&1E	30	Home cursor.
&1F ·c· ·r·	31 ·c· ·r·	Move cursor to given position in current window. ·c· is column, normally 1 to 80, ·r· is row, normally 1 to 25.

Configuring the Input to the package

The programs in the package will expect to be able to interrogate the keyboard. Most of the keys on the CPC664 keyboard return standard values except for the cursor keys. It is possible to use the **SETUP** utility (see part 2 of the chapter entitled 'AMSDOS and CP/M' to re-define the codes produced by the keyboard, although where possible, it is preferable for each different package to be configured to accept the standard default values.

The column marked 'WP Value' in the table below indicates typical values which might be set into the keyboard via the **SETUP** utility in a word processing environment if, for example, cursor codes are required from both the cursor key cluster and a portion of the keyboard, and the Word Processing package is only capable of recognising one unique code for each operation.

The installation procedure for a package will normally consist of asking a number of questions about the parameters of the CPC664 keyboard. The answers should be derived from the table below, which is an extract from the AMSTRAD CPC664 BASIC reference Manual SOFT 945:

Key Name	Value (Hex)	Value (Decimal)	(Key number if required to use SETUP utility)	WP Value (Decimal)
Cursor up	&F0	240	0	5
Cursor right	&F3	243	1	4
Cursor down	&F1	241	2	24
Cursor left	&F2	242	8	19
Clr	&10	16	16	7
Return	&0D	13	18	13
Space	&20	32	47	32
Escape	&FC	252	66	27
Tab	&09	9	68	9
Del	&7F	127	79	127

Starting a Turnkey CP/M Package

Normally all that is required, is to type the package's main program name at the **A>** prompt. For example to run a wages program called **PAYROLL.COM** simply type:

PAYROLL

Autostarting a Turnkey CP/M Package

It is possible to arrange for the CP/M operating system to automatically run a particular program every time CP/M is entered on that particular disc. This is performed by one of the options in the **SETUP** utility program (see part 2 of the chapter entitled 'AMSDOS and CP/M' for details).

Part 2: Cassettes

If you wish to use a cassette unit connected to the system, (as described in part 2 of the Foundation course), a number of the BASIC commands will operate differently when the computer is set to cassette operation by the command **!TAPE**. Various software messages and prompts will appear on the screen which are not seen during normal disc operation.

Unlike discs, filenames on cassette do not have such strict rules concerning their form. They may be up to 16 characters long, and may contain embedded spaces and punctuation marks. In some instances, they may be omitted altogether.

The following list describes the **DIFFERENCES** in the operation of each of these BASIC commands. Descriptions of the commands themselves will be found in the chapter entitled 'Complete list of AMSTRAD CPC664 BASIC keywords'.

CAT

You will be instructed:

Press **PLAY** then any key:

...whereupon you should press the **PLAY** button on your cassette unit, followed by one of the grey keys on the computer. The tape in the cassette will start turning, and the computer will display the names of each of the files that it finds (in sequence) on the cassette.

Each of the blocks of a file will be displayed, followed by a single character which indicates what sort of file it is:

\$ is an unprotected BASIC file
% is a Protected BASIC file
* is an ASCII file
& is a Binary file

The computer displays:

Ok

...at the end of the line if it has read the file successfully, indicating that the file would have loaded into memory, had the computer attempted to do so.

The CAT function will not affect the program currently in the computer's memory.

If a cassette file has been saved without a specified name, CAT will display it as:

Unnamed file

CAT is terminated by pressing [ESC].

Read errors

If the above file-reading messages are not displayed, or you get the message:

Read error aor.... Read error b

...on the screen, this indicates either of the following:

1. Your cassette unit is not correctly connected to the computer's TAPE socket (see part 2 of the Foundation course).
2. The **VOLUME** or **LEVEL** control on your cassette unit is not correctly adjusted.
3. The tape quality is poor, or the tape is worn.
4. The tape has been subjected to a magnetic field by being placed close to a loud-speaker, television set, etc.
5. You are attempting to read a cassette which has not been created for use on AMSTRAD computer systems. Note that you may use all software written for the AMSTRAD CPC664, and most software written for the AMSTRAD CPC464.

CHAIN
CHAIN MERGE
LOAD
MERGE
RUN

You need not specify the filename if you wish the first suitable file on the cassette to be loaded. Example commands:

```
CHAIN ""  
CHAIN "",100
```

```
CHAIN MERGE ""  
CHAIN MERGE "",100  
CHAIN MERGE "",100,DELETE 30-70
```

```
LOAD ""  
LOAD "",81F40
```

```
MERGE ""
```

```
RUN ""
```

(Note that holding down the **[CTRL]** key then pressing the small **[ENTER]** key on the numeric keypad, executes this command. Use when running cassette based software, after typing **I TAPE**)

You will be instructed:

Press **PLAY** then any key:

....whereupon you should press the **PLAY** button on your cassette unit, followed by one of the grey keys on the computer. The tape in the cassette will start turning, and the computer will load the file to be processed.

The screen will display the loading messages:

```
Loading FILENAME block 1
```

....and as many other block numbers as there are in the file, until the file is loaded.

If the first character of the filename is **!** then the above messages will be suppressed, and you will not be required to 'press any key' for the file to load. (You must make sure that the **PLAY** button on your cassette unit is down.) If your programs use the **!** mark and are also required to run on disc, the **!** mark will be ignored during disc operation (when the disc filename is being read). Note that the **!** mark does **NOT** occupy one of the character positions in the cassette or disc filename.

Abandoning the command using the **[ESC]** key produces the error message on the screen:

Broken in

If the file does not successfully load, read the paragraph entitled 'Read errors' earlier in this section.

WARNING: The internal disc interface occupies a small portion of the memory that in some cases, was used by commercial writers of cassette based software for the model CPC464. These cassettes will not operate properly with the CPC664 + Cassette unit.

EOF POS(#9)

These functions operate on cassette as per disc.

INPUT #9 LINE INPUT #9 OPENIN and CLOSEIN

You need not specify the filename if you wish the first suitable file on the cassette to be loaded. Example command:

OPENIN ""

You will be instructed:

Press PLAY then any key:

....whereupon you should press the **PLAY** button on your cassette unit, followed by one of the grey keys on the computer. The tape in the cassette will start turning, and the computer loads the first 2K bytes of the file into a portion of the memory called the 'file buffer'. Input is taken from the file buffer until it is empty, and the computer again prompts:

Press PLAY then any key:

....and loads the next 2K bytes from the file.

The screen will display the loading messages:

Loading FILENAME block 1

....and other block numbers in turn, as the file is loaded.

If the first character of the filename in the **OPENIN** command is **!** then the above messages will be suppressed, and you will not be required to 'press any key' for the file to load. (You must make sure that the **PLAY** button on your cassette unit is down.) If your programs use the **!** mark and are also required to run on disc, the **!** mark will be ignored during disc operation (when the disc filename is being read). Note that the **!** mark does NOT occupy one of the character positions in the cassette or disc filename.

Abandoning file input using the **[ESC]** key produces the error message on the screen:

Broken in

If the file does not successfully load, read the paragraph entitled 'Read errors' earlier in this section.

LIST #9

OPENOUT and CLOSEOUT

PRINT #9

WRITE #9

You need not specify the filename if you wish the file to be saved as an Unnamed file. Example command:

OPENOUT ""

The first 2K bytes of the file to be saved to cassette will first be written into a portion of the memory called the 'file buffer'. When the file buffer is full, you will be instructed:

Press REC and PLAY then any key:

....whereupon you should press the **RECORD** and **PLAY** buttons on your cassette unit, followed by one of the grey keys on the computer. The tape in the cassette will start turning, and the computer saves the contents of the file buffer. The computer then refills the file buffer with the next 2K bytes of the file, and again prompts:

Press REC and PLAY then any key:

....and saves the next 2K bytes to cassette.

If the file buffer is partly full and the command **CLOSEOUT** is encountered, the computer will save the remaining contents of the file buffer to cassette, issuing the prompt:

Press **REC** and **PLAY** then any key:

The screen will display the saving message:

Saving **FILENAME** block **x**

If the first character of the filename in the **OPENOUT** command is **!** then the above messages will be suppressed, and you will not be required to 'press any key' for the file to save. (You must make sure that the **RECORD** and **PLAY** buttons on your cassette unit are down.) If your programs use the **!** mark and are also required to run on disc, the **!** mark will be ignored during disc operation (when the disc filename is being read). Note that the **!** mark does **NOT** occupy one of the character positions in the cassette or disc filename.

Abandoning file output using the **[ESC]** key produces the error message on the screen:

Broken in

Successful saving

To ensure that the file is successfully saved:

1. Check that your cassette unit is correctly connected to the computer's **TAPE** socket (see part 2 of the Foundation course).
2. Check that the **RECORD** or **LEVEL** control on your cassette unit is correctly adjusted.
3. Check that you are not using inferior quality cassettes or C120s. (AMSOFT C15 cassettes are recommended.)
4. Make sure that your cassettes are not subjected to magnetic fields by being placed close to a loud-speaker, television set, etc.
5. Before deleting a program from the memory after saving it, **CAT**alog the tape, to verify that the program has been saved successfully.
6. Ensure that your cassette unit is periodically maintained, and that the tape heads are regularly cleaned.

SAVE

You need not specify the filename if you wish the program to be saved as an Unnamed file. Example command:

```
SAVE ""
```

You will be instructed:

```
Press REC and PLAY then any key:
```

....whereupon you should press the **RECORD** and **PLAY** buttons on your cassette unit, followed by one of the grey keys on the computer. The tape in the cassette will start turning, and the computer will save the program.

The screen will display the saving messages:

```
Saving FILENAME block 1
```

....and as many other block numbers as there are in the file, until the file is saved.

If the first character of the filename is ! then the above messages will be suppressed, and you will not be required to 'press any key' for the file to be saved. (You must make sure that the **RECORD** and **PLAY** buttons on your cassette unit are down.) If your programs use the ! mark and are also required to run on disc, the ! mark will be ignored during disc operation (when the disc filename is being read). Note that the ! mark does NOT occupy one of the character positions in the cassette or disc filename.

Abandoning the command using the [ESC] key produces the error message on the screen:

```
Broken in
```

Read the paragraph entitled 'Successful saving', earlier in this section.

SPEED WRITE

The command operates only on cassette, and can be issued while the computer is set to disc operation.

Error messages

Note that error messages 7, 21, 24, 25, 27, and 32 (see the chapter entitled 'For your reference....') will be generated during cassette operation, if appropriate.

AMSDOS external commands

Input and output direction to cassette or disc, is effected by the commands:

 I TAPE (which can be sub-divided into I TAPE . IN and I TAPE . OUT)
 I DISC (which can be sub-divided into I DISC . IN and I DISC . OUT)

All external commands:

 I A
 I B
 I CPM
 I DIR
 I DRIVE
 I ERA
 I REN
 I USER

Are executed to disc, regardless of whether cassette operation has been selected.

Chapter 5

AMSDOS and CP/M

Part 1: AMSDOS

Subjects covered:

- ★ Introduction to AMSDOS
- ★ Disc Directory
- ★ Changing discs
- ★ The format of file names
- ★ AMSDOS file headers
- ★ Wild cards
- ★ An example program using AMSDOS Commands
- ★ Reference Guide to AMSDOS Commands
- ★ Manipulating files
- ★ Reference guide to Error Messages

Introduction

AMSDOS extends the AMSTRAD BASIC supplied with your computer, by the addition of a number of external commands, which are identified by the preceding | (bar) symbol.

AMSDOS allows the user to change discs freely, as long as no files are in use - in which case an error message will be displayed and there could be a loss of data if the open file was being written to.

Disc Directory

Every disc has two sections, the directory and the data area. The directory contains a list of all the filenames and a 'map' of whereabouts on the disc each file is to be found. AMSDOS or CP/M can calculate the size of a particular file by inspecting its directory entry. Calculation of the amount of space left on a disc is made by adding up all the files in the directory and seeing how much remains unused.

Whenever a file is read, its directory entry is examined, giving the disc location. When a new file is created, free space is allocated to it, and when a file is erased the space is relinquished. The directory works in units of 1K and can have up to 64 different entries. Large files will have one entry for every 16K although normally this fact is hidden from the user.

Changing Discs

Under AMSDOS a disc may be changed, or removed, whenever the drive is not being accessed, and neither the input nor output files are open on that drive. Unlike CP/M there is no need to 'log in' a disc.

Changing a disc while it is still being written to, may corrupt the data on the disc. If a disc is changed while there are still files open on it, then as soon as AMSDOS detects this, all the open files on the drive will be abandoned and an error message produced. Any data yet to be written will be lost and the latest directory entry will not be written to disc. However, AMSDOS can only detect this change when it reads the directory, which it does every 16K of the file (and whenever a file is opened or closed). Thus, potentially 16K of data could be corrupted by changing a disc while there are still files open on it.

AMSDOS filenames and filetypes.

It is standard practice to name disc files in such a way that there is an indication of which 'type' they are. This naming convention DOES NOT 'force' the computer to use the file in any particular way, however some programs will only accept a file when it has the correct type of name. BASIC will accept any type of name, but will search in preference for certain file types if not otherwise specified. (See 'AMSDOS headers' ahead.)

Construction of Filenames

The filename is constructed from two parts with a . (dot) separating them. The first part can be up to 8 characters long, and the second up to 3 characters long. Thus for example, "ROINTIME.DEM", "FORMAT.COM", and "EX1.BAS" are all legal filenames.

The second part of the filename is called the filetype. Filenames and filetypes can be composed of a mixture of letters and numbers, but cannot have embedded spaces. Some common conventional filetypes are:

- .space Unspecified type. May be a data file created by an OPENOUT "`<filename>`" or BASIC program saved by AMSDOS using SAVE "`<filename>`", A style.
- .BAS BASIC program saved by AMSDOS using SAVE "`<filename>`" or SAVE "`<filename>`", P or SAVE "`<filename>`.BAS", A styles.
- .BIN Program or area of memory saved by AMSDOS using SAVE "`<filename>`", B, `<binary parameters>` style.
- .BAK Old version of a file, where AMSDOS or a utility program has saved a newer version of a file using an existing name. This allows the user to back-track to the previous (BAcK-up) version if required.
- .COM Command file. CP/M utility programs are all of this filetype.

AMSDOS headers

AMSDOS automatically SAVES files with a suitable type identifier, so it is not normally necessary to specify one unless you wish to override the defaults described previously. BASIC program files, protected BASIC program files and binary files are saved to the disc with a header record, so that the AMSDOS command:

```
LOAD "<filename>"
```

....can recognise them and take the appropriate action. If the AMSDOS command LOAD cannot find a header, it assumes that the file is a program in ASCII, i.e. plain text.

Notwithstanding the contents of the header, when AMSDOS is asked to LOAD a file where no filetype is specified, then it first looks for a file of type:

```
.<space>
```

If that does not exist, it looks for a file of type:

```
.BAS
```

....then finally, one of type:

```
.BIN
```

This allows the user to abbreviate the filename, i.e. not needing to specify the filetype, in most instances.

A disc data file started with the command OPENOUT and subsequently written to will have no header and the contents will be in ASCII, i.e. plain text, from the BASIC WRITE, PRINT or LIST commands. The disc command OPENIN will search for files in the same order as LOAD, if no file type is specified.

Filenames on two drives

On a two drive system, i.e. if an additional FD-1 has been connected to the computer, files can exist on either drive. The computer will not automatically look for a file on both drives, so the user must specify which drive to use. You can either employ the IA or IB or I DRIVE commands (full description ahead) to select one or other drive, and then use a normal filename, or alternatively you can override the default drive assignment by specifying the drive as an A: or B: prefix to the filename. Thus, for example:

```
IB
SAVE "PROG.BAS"
IA
```

....and....

```
IA
SAVE "B:PROG.BAS"
```

....both save the program on the additional drive, Drive B

Similarly, you can override the default **USER** assignment by specifying the **USER** number (in the range 0 to 15) as a prefix to the filename. Thus, for example:

```
LOAD "15:PROG.BAS"
```

....and....

```
SAVE "15:PROG.BAS"
```

....would load and save the program to the **USER** number 15 section of the disc, whatever the default **USER** number setting. (See the **USER** command, ahead.)

Finally, it is possible to override both default **USER** and **DRIVE** settings (in that order) by specifying them together in the prefix to the filename, for example:

```
RUN "15B:PROG.BAS"
```

Wild cards

It is often required to perform some disc operation (Cataloguing, copying, erasing etc.) on a number of disc files. When a filename is specified for a particular operation, the software scans the disc directory looking for a name which exactly matches. It is possible, where the command allows, to perform the operation on a set of files where some of the characters can be 'don't care'. This is shown by using the character **?** in the 'don't care' position. If the whole block (or remainder of the whole block) of the filename or type specifier is 'don't care', then the block of **?**'s can be abbreviated to the symbol *****. Thus, for example, **FRED.*** is shorthand for **FRED.???** and **F*.BAS** is shorthand for **F??????? .BAS**

Finally, the expression ***.*** means 'all files'.

examples:

not "22223.BAS"

DIRECTORY	Match *.BAS	Match FRED?.BAS	Match F*.BA?
BERT.BAS FRED1.BAS FRED2.BAS FRED3.BAK FRED3.BAS FINISH.BAS	BERT.BAS FRED1.BAS FRED2.BAS FRED3.BAS FINISH.BAS	FRED1.BAS FRED2.BAS FRED3.BAS	FRED1.BAS FRED2.BAS FRED3.BAK FRED3.BAS FINISH.BAS

Examples of Using Amsdos Commands in a program

To give you a good understanding of the AMSDOS commands we recommend that you work through the examples, referring to the relevant sections in the rest of this chapter as you go. DO NOT operate these programs with your original Master CP/M System/Utility disc installed - the program writes to the disc and you should NEVER risk writing to the master disc. Use a working disc or a copy instead.

Saving variables and performing a Screen Dump

Note the use of .DAT and .SRN filetypes. These filetypes are used to remind us of what is in the file, rather than because they have any inherent significance. The file PARAM.DAT will be an ASCII data file without a header, whilst FLAGDUMP.SRN is an AMSDOS Binary file with a header. The programs have been provided on your Master System/Utility disc in ASCII form, within the program EX1.BAS.

This example program (EX1.BAS) draws a Union Jack flag and then saves the whole screen to disc.

Note how the program deliberately tries to read from the file PARAM.DAT before writing to it, in order to establish if the file already exists. If the file does NOT exist, then an error is reported by BASIC, the error is trapped by the program, and execution proceeds without interruption. If the file DOES already exist, then no error is reported, and the program automatically asks if you wish to overwrite the existing file.

To run the program, type:

```
RUN "EX1"
```

As we discussed, AMSDOS will search automatically adding the .BAS filetype for you. The particulars of the screen dump, namely the screen mode, palette colours and name of file containing the actual information are saved into a parameter file. This illustrates the use of a data file to WRITE program variables (dumpfile\$) and constants (1), saving them for use by another program.

```
10 dumpfile$="flagdump.srn"
20 MODE 1:BORDER 0
30 DIM colour(2)
40 FOR i=0 TO 2
50 READ colour(i): REM get colours from DATA statement
60 INK i,colour(i)
70 NEXT
80 ON ERROR GOTO 430
90 OPENIN "param.dat" ' test if file exists
100 CLOSEIN:ON ERROR GOTO 0
110 IF errnum=32 AND DERR=146 THEN CLS:
    GOTO 160 ' file doesnt exist
120 CURSOR 1:PRINT "Do you want to overwrite
    old file? Y/N ";
130 a$=INKEY$:ON INSTR(" YN",UPPER$(a$))
    GOTO 130,150,140:GOTO 130
140 PRINT a$:PRINT "Program abandoned":END
150 PRINT a$:CURSOR 0
160 OPENOUT "param.dat"
170 WRITE #9,dumpfile$,1: REM save filename and mode
180 FOR i=0 TO 2
190 WRITE #9,colour(i): REM save colours
200 NEXT i
210 CLOSEOUT
220 CLS
230 gp=1:GRAPHICS PEN gp:w=125
240 x=-65:a=240:y=400:b=-150:GOSUB 400
250 y=0:b=150:GOSUB 400
260 x=575:a=-240:y=400:b=-150:GOSUB 400
270 y=0:b=150:GOSUB 400
280 gp=2:GRAPHICS PEN gp:w=40
290 a=240:x=-40:y=400:b=-150:GOSUB 400
300 x=0:y=0:b=150:GOSUB 400
310 a=-240:x=640:y=0:b=150:GOSUB 400
320 x=600:y=400:b=-150:GOSUB 400
330 ORIGIN 0,0,256,380,0,400:CLG 1
340 ORIGIN 0,0,0,640,150,250:CLG 1
```

Continued on the next page

```

350 ORIGIN 0,0,280,352,0,400:CLG 2
360 ORIGIN 0,0,0,640,168,230:CLG 2
370 SAVE dumpfile$,b,&c000,&4000
380 DATA 2,26,6
390 END
400 MOVE x,y:DRAWR a,b:DRAWR w,0:DRAWR -a,-b
410 MOVE x+a/2+w/2,y+b/2:FILL gp
420 RETURN
430 errnum=ERR:RESUME NEXT
run

```

The second example (EX2.BAS) also provided on your master disc, is a general purpose screen dump displaying program, using a parameter file to control its action. Note how variables are INPUT from the data file, with the EOF function allowing automatic variation in the size of the file. It is important that the screen dump displayed by this program was saved with the screen in a known position in memory, otherwise the result will be 'skewed'. This is ensured by the saving program executing a MODE command and thereafter being careful not to cause the screen to scroll.

```

10 DIM colour(15): REM Provision for 16 colours
20 OPENIN "param.dat"
30 INPUT #9, filename$,screenmode
40 i=0
50 WHILE NOT EOF
60 INPUT #9,colour(i)
70 INK i,colour(i)
80 i=i+1
90 WEND
100 CLOSEIN
110 MODE screenmode:BORDER 0
120 LOAD filename$
run

```

Summary of AMSDOS external commands

| A

| A

COMMAND: Set default drive to Drive A. Equivalent to | DRIVE with parameter A. (The main drive within the computer is Drive A.)

| B

| B

COMMAND: Set default drive to Drive B. Equivalent to | DRIVE with parameter B. (The main drive within the computer is Drive A.)

| CPM

| CPM

COMMAND: Switch to alternative disc environment by loading operating system from a system disc. The operating system supplied with the computer is CP/M 2.2

This will fail if the computer's drive does not contain a system disc with CP/M.

| DIR

| DIR [, 'string expression']

| DIR, "*.BAS"

COMMAND: Display the disc directory (In CP/M style), and free space. If the string expression is omitted, the wild-card *. * is assumed.

| DISC

| DISC

COMMAND: Equivalent to the two commands | DISC . IN and | DISC . OUT

| DISC.IN

| DISC . IN

COMMAND: Use disc as file input medium.

| DISC.OUT

| DISC . OUT

COMMAND: Use disc as file output medium.

| DRIVE

| DRIVE , <string expression>

| DRIVE , "A"

COMMAND: Set the default drive. This will fail if AMSDOS is unable to read the disc in the requested drive.

| ERA

| ERA , <string expression>

| ERA , "*.BAK"

COMMAND: Erase all files which match the filename and which are not Read/Only. Wild cards are permitted.

| REN

| REN , <string expression> , <string expression>

| REN , "NEWNAME.BAS" , "OLDNAME.BAS"

COMMAND: Give a file a new name. A file with the new name must not already exist. Wild cards are not permitted.

The USER (see |USER ahead) parameter may be specified within the <string expression>s to override any default settings. For example, the command |REN , "0:NEW.BAS" , "15:OLD.BAS" will rename the file in USER 15 called "OLD.BAS" , to a file called "NEW.BAS" in USER 0, regardless of any default or previously issued settings of |USER.

| TAPE

| TAPE

COMMAND: Equivalent to the two commands |TAPE.IN and |TAPE.OUT. Used if an external cassette unit is connected.

| TAPE.IN

| TAPE.IN

COMMAND: Use tape as file input medium. Used if an external cassette unit is connected.

| TAPE.OUT

| TAPE.OUT

COMMAND: Use tape as file output medium. Used if an external cassette unit is connected.

| USER

| USER, *integer expression*

| USER, 3

COMMAND: Determines which of up to 16 individual sections of the directory (in the range 0 to 15), disc functions (e.g. **CAT**, **LOAD**, **IDIR** etc.) are to be performed on.

A file on one **USER** number may be transferred to another, by a **IREN** command. For example, **IREN**, "15:EXAMPLE.BAS", "0:EXAMPLE.BAS" transfers a file from **USER** number 0 to **USER** number 15, although the name of the file itself (**EXAMPLE.BAS**) is not changed.

Copying Files

AMSDOS files with headers

It is possible to copy this type of file in the CP/M environment using **PIP** or **FILECOPY** (See part 2 of this chapter). Any file created by AMSDOS which has a header record (see 'AMSDOS headers', previously described) will be copyable as a whole - disc to disc / disc to tape / tape to disc - but in general the contents of the file will not be understood by any CP/M programs.

ASCII files

Files created by AMSDOS without headers are generally in ASCII, and are both copyable and understood by CP/M programs. In particular it should be possible to exchange ASCII program files, ASCII data files and ASCII text files freely between AMSDOS and CP/M programs.

Read/Only files

It is possible, using CP/M, to set any file to be Read/Only, and/or invisible to directory cataloguing operations. Such attributes can only be set or reset in the CP/M environment, but are honoured by AMSDOS. For further details, see part 2 of this chapter on CP/M (STAT utility).

File Copying procedures

The tables ahead cover copying files of all sorts between disc and tape (if connected). It assumes that no additional disc drive is connected. It is not possible to copy a protected BASIC program at all, nor to copy a binary file (such as a machine code video game) unless the load addresses are known. Further details of the programs FILECOPY, CLOAD and CSAVE are given in part 2 of this chapter, which also contains information on the CP/M utility PIP. (Copying files from one disc to another in a two drive system is normally easier with the PIP utility.)

See next page for tables.

		COPY FROM:		
COPY TO:		AMSTRAD BASIC on tape *	ASCII data on tape *	Binary on tape *
AMSTRAD BASIC on tape *		ITAPE LOAD "FILE" <change tapes> SAVE "FILE" IDISC		
Binary on tape *				H=HIMEM ITAPE MEMORY <s>-1 LOAD "FILE" <change tapes> SAVE "FILE",B, <s>,<l>,<r> IDISC MEMORY H <note 2>
ASCII on tape *		ITAPE LOAD "FILE" <change tapes> SAVE "FILE",A IDISC	ICPM CLOAD "FILE",TEMP <change tapes> CSAVE TEMP,"FILE" ERATEMP AMSDOS <note 1>	
AMSTRAD BASIC on disc *		ITAPE LOAD "FILE" IDISC SAVE "FILE"		
ASCII on disc		ITAPE LOAD "FILE" IDISC SAVE "FILE",A	ICPM CLOAD "FILE" AMSDOS	
Binary on disc *				H=HIMEM ITAPE MEMORY <s>-1 LOAD "FILE" IDISC SAVE "FILE",B, <s>,<l>,<r> MEMORY H <note 2>

* File has a header

<note 1> Requires free disc space for temporary file "TEMP".

<note 2> <s> is start address of file, <l> is length, <r> is optional run address.

COPY TO:	COPY FROM:			
	AMSTRAD BASIC on disc *	ASCII data on disc	AMSDOS Binary on disc *	All other on disc
AMSTRAD BASIC on tape *	LOAD "FILE" ITAPE SAVE "FILE" IDISC			
Binary on tape *			H=HIMEM MEMORY <i>s</i> -1 LOAD "FILE" ITAPE SAVE "FILE", B, <i>s</i> , <i>l</i> , <i>r</i> IDISC MEMORY H <i>note 2</i> .	
ASCII on tape *	LOAD "FILE" ITAPE SAVE "FILE", A IDISC ICPM CSAVE FILE AMSDOS	ICPM CSAVE FILE AMSDOS		ICPM CSAVE FILE AMSDOS <i>note 3</i> .
AMSTRAD BASIC on disc *	LOAD "FILE" <i>change discs</i> SAVE "FILE" - or - ICPM FILECOPY FILE <i>follow instructions</i> AMSDOS			
ASCII on disc	LOAD "FILE" SAVE "FILE", A	ICPM FILECOPY FILE <i>follow instructions</i> AMSDOS		
AMSDOS Binary on disc *			ICPM FILECOPY FILE <i>follow instructions</i> AMSDOS	
All other on disc				ICPM FILECOPY FILE <i>follow instructions</i> AMSDOS

* File has a header

note 2 *s* is start address of file, *l* is length, *r* is optional run address.

note 3 Destination file cannot be used directly by BASIC. However this option is useful as a low cost transportation or backup medium. The file can be copied back to a disc by CLOAD "FILE".

Reference guide to AMSDOS Error Messages

When AMSDOS cannot carry out a command for some reason, it will display an error message. If there is a problem with the hardware, the error message is followed by the question:

Retry, Ignore or Cancel?

R causes the operation to be repeated, possibly after the user has taken some preventative action.

I causes the computer to continue as if the problem had not occurred, which will often lead to unexpected and possibly inconvenient results.

C causes the operation to be cancelled, which will often lead to a further error message.

For further information see part 6 of the chapter entitled 'For your reference....'.

Unknown command

7.30

....The command is not spelt correctly.

Bad command

....The command cannot be carried out for some reason. Syntax error or inappropriate hardware configuration.

<filename> already exists

....User is trying to rename a file with a name already in use.

<filename> not found

....File does not exist.

Drive <drive>: directory full

....No more room in the disc directory a new entry.

Drive <drive>: disc full

....No more room on the disc for new data.

Drive <drive>: disc changed, closing <filename>

....Disc has been changed with files still open on it.

<filename> is read only

....File cannot be operated on because it is Read/Only. Files can only be set Read/Only or Read/Write in the CP/M environment.

Drive <drive>: disc missing

....No disc in drive, or disc is not seated and spinning properly. Recommended action is to eject and re-insert the disc and type R

Drive <drive>: disc is write protected

....Attempt has been made to write on a disc with the Write Protect hole open. To use the disc, eject, close the Write Protect hole, re-insert the disc and type R

Drive <drive>: read fail

....Hardware error reading disc. Recommended action is to eject and re-insert the disc and type R

Drive <drive>: write fail

....Hardware error writing disc. Recommended action is to eject and re-insert the disc and type R

Failed to load CP/M

....Read error loading CP/M during I CPM command, or you are not using a valid system disc containing CP/M.

Part 2: CP/M

Subjects covered:

- ★ Booting CP/M
- ★ Configuration sector
- ★ Direct Console Mode
- ★ Transient programs
- ★ Managing peripherals

CP/M 2.2 is provided with your CPC664. CP/M is a disc operating system. It is a special program which gives you access to the full power of your CPC664 system. Because CP/M is available for so many different computers it means that there are thousands of applications packages available for you to choose from and a whole wealth of knowledge and experience for you to draw upon.

Full details of CP/M including information on how to write your own programs and information on the AMSTRAD implementation of CP/M are contained in SOFT159 - A Guide to CP/M.

Introduction

The CP/M operating system provides a user interface for disc hardware - a way for you to communicate with the computer and manipulate files and peripherals.

The fundamental interface that is available is called the Direct Console Mode and is identified by the A> or B> prompt. Certain built-in commands are available but the majority of the functionality is obtained by loading and running 'Transient Programs'. They are called 'transient' because they are only in the computer (rather than on the disc) while you are using them, as opposed to being built-in.

As well as standard CP/M error messages, the system also generates a number of specialised hardware error messages. Refer to the 'Reference guide to AMSDOS error messages' in part 1 of this chapter.

CP/M system tracks

The major part of CP/M resides on the outermost two tracks of the disc. The computer loads CP/M from these tracks into the memory using a two stage process.

Firstly the AMSDOS command `1 CPM` loads the first sector of track 0. On a system disc this sector has been arranged to be a program which then loads the rest of the system tracks into memory. Various checks are performed to determine that the system tracks contain valid CP/M software and to calculate where in memory to load them.

Configuration Sector

During the loading process, when CP/M is first activated, various system parameters are loaded from a special sector within the system tracks. These parameters include the Sign-On (wake-up) message, special keyboard codes required etc. The `SETUP` utility is provided to customise the configuration sector to your requirements.

Console control codes

In the CP/M environment a variety of special key operations are used to control program flow. These keystrokes replace the action of the `[ESC]`ape key used in AMSTRAD BASIC, although some applications packages may re-instate the `[ESC]`ape key with some of its former power.

- [CTRL]S** Halts the screen output from CP/M. Type any character to resume.
- [CTRL]C** Typed at the start of a line returns control to the Direct Console Mode. Many utilities and applications programs will also recognise this as a request to abandon the program.
- [CTRL]P** Hardcopy toggle. Turn on/off log of all screen output to printer.
- [CTRL]Z** End of text.

Logging in a disc

Unless special action is taken by the CP/M program (as `FILECOPY` does for example) then CP/M will not allow you to write to a disc unless it has been 'logged in'. Furthermore, the type of disc format (`SYSTEM`, `DATA` OR `IBM`) is only re-determined when a disc is logged in. For the main disc drive within the computer (Drive **A**) this takes place whenever CP/M returns to the Direct Console Mode, or when `[CTRL]C` is typed at the `A>` or `B>` prompt. For an additional drive (Drive **B**) this takes place the first time that the disc in Drive **B** is accessed, after Drive **A** has been logged in.

Should you try writing to a disc that has not been logged in, the error message:

Bdos Err on <drive>: R/O

....will be displayed. Press any key to continue. However, if the disc was also of a different format, then a read or write error will occur. Type C to continue.

Direct Console Commands

There are five Direct Console Commands which can be typed at the A> or B> prompt. These are SAVE, DIR, ERA, REN, and TYPE.

The first of these; the SAVE command, is for specialist use only.

CP/M Error messages tend to be economical and normally consist of repeating the offending command or filename followed by a question mark:

<unknown command>?

....whereupon you should repeat the command with the mistake corrected.

Filenames

Many of the commands take filenames as a parameters, and where specified, the filename may contain wild-cards (see the section entitled 'Wild cards' in part 1 of this chapter). All filenames will be forced to upper case.

Direct Console Commands and most utility programs do not require that filenames are contained in double quotes "". The CLOAD and CSAVE utilities (see ahead) do however require that double quotes be placed around the CASSETTE filename.

Remember that filenames can have an A: or B: prefix to force CP/M to use the appropriate drive.

Switching default drives

If you have an additional disc drive connected, then it is possible to switch the default drive selection between Drive A and Drive B by typing A: or B: at the B> or A> prompt. That prompt, of course, tells you the current default drive. Adding the A: and B: prefix to filenames overrides, but does not reset, the default drive setting.

DIR command

DIR lists the **DIR**ectory of the disc. The filenames are not sorted into any particular order, but the position of the filename in the **DIR** display indicates the position of that file's entry in the disc directory. Wild cards are permitted.

DIR	will list all files on the default drive
DIR B:	will list all files on Drive B
DIR *.BAS	will list all files of type .BAS
DIR B:*.BAS	will list all files of type .BAS on Drive B
DIR PIP.COM	will list only the file PIP.COM (if it exists)

ERA command

ERA is used to **ER**ase files from the directory. Only the Directory Entry is erased so the data is still in the data section of the disc until the space is re-used by another file, but the information is nevertheless not recoverable. Wild card filenames are permitted. If the filename ***.*** is specified then **ERA** will ask for confirmation that all files should be erased. **ERA** does not list the filenames that are deleted. If any file about to be erased is found to be Read/Only (see **STAT** ahead), then the command will abort.

ERA PIP.COM	will erase the file PIP.COM
ERA B:PIP.COM	will erase the file PIP.COM on Drive B
ERA *.BAS	will erase all .BAS files

REN command

REN allows you to **RE**name an existing file. The new filename is specified first, followed by **=** then the existing filename. If the new filename already exists, an error message will be displayed. Wild cards are not permitted in the filenames.

REN NEWNAME.BAS=OLDNAME.BAS	will rename the file OLDNAME.BAS to NEWNAME.BAS
REN B:NEWNAME.BAS=OLDNAME.BAS	will rename the file OLDNAME.BAS to NEWNAME.BAS , on Drive B.

TYPE command

TYPE asks for the specified file to be **TYPE**d onto the screen. If the file is not an ASCII text file, unpredictable and possibly undesirable side-effects may occur.

TYPE EX1.BAS

....will display the program file EX1.BAS

Transient commands

To perform more sophisticated file management than permitted by the Direct Console Mode you must employ one of the various utility programs provided. These are invoked merely by typing the program name, possibly followed by some parameters. You may possibly have already used **FORMAT** and **FILECOPY**.

The commands fall into a number of categories as indicated below. Full documentation of these programs is extensive and is contained in **SOFT 159 - A Guide to CP/M**.

The **SYSGEN**, **BOOTGEN**, **FILECOPY**, **COPYDISC**, **DISCCOPY**, **CHKDISC**, **DISCCHK**, **FORMAT**, **SETUP**, **CSAVE**, **CLOAD** and **AMSDOS** commands are designed by **AMSTRAD**, and work exclusively on the **AMSTRAD** system. They have no function on any other **CP/M** system, although other manufacturers may supply similar utilities (often with the same names) customised for their hardware.

Peripheral Management

The **PIP** utility (Peripheral Interchange Program) allows you to transfer information between the computer and its peripherals. In general, the form of the command is:

PIP <destination>=<source>

The <source> and <destination> can be either a filename, with wild-cards allowed in the source, or a device token. The following device tokens may be used:

As Source

CON: keyboard
RDR: serial interface

As destination

CON: screen
PUN: serial interface
LST: printer

Examples:

PIP B:=A:*.COM

....copy all **.COM** files from Drive A to Drive B

```
PIP SAV.BAS=EX1.BAS
```

....make a copy of EX1.BAS, calling it SAV.BAS

```
PIP CON:=EX1.BAS
```

....send file EX1.BAS to screen. (Similar effect to TYPE EX1.BAS)

```
PIP LST:=EX1.BAS
```

....send file EX1.BAS to printer

```
PIP TYPEIN.TXT=CON:
```

....accept keyboard input into a file called TYPEIN.TXT

Note that this operation is terminated by the **[CTRL]Z** control code, and that in order to get a new line you must type **[CTRL]J** after **[ENTER]** every time. **[CTRL]J** is the ASCII for line feed.

Note that **PIP** can NOT be used to copy files from one disc to another on a single drive system. Use **FILECOPY** instead.

File and disc copying

Single file copying

The utility **FILECOPY** allows you to copy files from one disc to another, using the single drive within the computer. It copes with disc changing and gives full instructions on the screen. If a wild-card filename is specified then **FILECOPY** asks you to confirm tht you indeed wish to copy each file on an individual basis. The program informs you of each filename as each file is copied.

```
FILECOPY *.BAS
```

....Copy all the files of type .BAS

```
FILECOPY EX1.BAS
```

....Copy the file EX1.BAS

Whole disc copiers and checkers

DISCCOPY (for the standard single drive system) and **COPYDISC** (for two drive systems) allow you to make a backup copy of the entire disc. They give full instructions on the screen.

If the destination disc is not formatted, or not of the same format as the source disc then these utilities will automatically format the disc correctly as they copy.

The companion utilities **DISCCHK** and **CHKDISC** allow you to compare two discs and verify that they have exactly the same contents.

Cassette files

Two utilities are available which transfer files between disc and cassette. Except for specialist use it is unlikely that anything than ASCII, i.e. plain text, files can usefully be transferred.

CLOAD can take two parameters, the first is the source (cassette) filename, enclosed in double quotes, and the second the destination (disc) filename. If the destination filename is omitted, the disc file will have the same name as the cassette file, which must then be a CPM compatible filename. If the source filename is omitted then **CLOAD** reads the first file encountered on the tape. If the first character of the cassette filename is ! then the normal cassette messages will be suppressed.

Example command:

```
CLOAD "MY LETTER" MYLETTER.TXT
```

CSAVE can take three parameters. The first is the source (disc) filename and the second the destination (cassette) filename, enclosed in double quotes. If the destination filename is omitted, the cassette file will have the same name as the disc file. If the first character of the cassette filename is ! then the normal cassette messages will be suppressed. If both filenames are specified, then a third parameter may be used to specify the tape write speed; 0 for nominal 1000 baud, 1 for nominal 2000 baud.

Example commands:

```
CSAVE OUTPUT.TXT "OUTPUT TEXT" 1  
CSAVE DATAFILE
```

System Management

STAT

STAT provides more sophisticated directory (and peripheral) management. All the normal rules apply to the filenames, including the use of wild-cards. Facilities provided are as follows:

```
STAT
STAT A:
STAT B:
```

....displays disc status and free space.

```
STAT *.COM
STAT EX1.BAS
```

....displays extended directory information about a particular file.

```
STAT *.COM $R/O
STAT EX1.BAS $R/O
```

....sets a file to a Read/Only status, so that it cannot be accidentally erased or overwritten.

```
STAT *.COM $R/W
STAT EX1.BAS $R/W
```

....sets a file to Read/Write status, reversing the Read/Only assignment.

```
STAT *.COM $SYS
STAT SECRET.BAS $SYS
```

....sets a file to 'System' status so that it is invisible to directory listings and file copying programs. The file will still be available for all other purposes.

```
STAT *.COM $DIR
STAT SECRET.BAS $DIR
```

....sets a file to 'Directory' status, reversing the 'System' assignment.

SETUP

This utility allows you to re-define the characteristics of the CPC664 keyboard and disc drive section, and serial interface. It also enables you to invoke various actions when CP/M is first loaded. When finished it updates the configuration sector. The program is menu-driven and when a particular screen is correct, or requires no modification, move on to the next by answering Y to the question:

Is this correct (Y/N):_

The program can be aborted by [CTRL]C keys. When all the changes have been made it will prompt:

Do you want to update your system disc (Y/N):_

....giving you the opportunity to retain the existing configuration sector by typing N. It will also prompt:

Do you want to restart CP/M (Y/N):_

....allowing you to load and try the new configuration by typing Y

To copy a configuration sector from one disc to another, either use **BOOTGEN** (see ahead) or load **SETUP** from the source disc, answer Y to **EVERY** question, inserting the destination disc before answering the penultimate question.

Characters with an ASCII value less than decimal 32 can be typed into strings by typing a ↑ followed by a suitable character from the set @, A-Z, [, \,], >, _

The following options are those more commonly requiring attention:

**** Initial command buffer:**

Any characters entered here will appear as if they had been typed into the Direct Console Mode when CP/M is first loaded. This has the effect of auto-running a particular program at that time. Remember to include the equivalent of the [ENTER] key which is represented by the two characters ↑ M

For example to auto-run **STAT**, the initial command buffer should be **STAT ↑ M**

Sign-on string:

This is the message displayed at the top of the screen when CP/M is first loaded. Note the use of ↑ J ↑ M to give a carriage return - line feed effect. The early part of the standard message sets suitable screen and border colours for working in 80 column mode, and should be copied exactly if they are to be preserved.

Keyboard translations:

This allows new ASCII values to be set into keys, effectively simulating the `KEY DEF` command in BASIC. The parameters required are the key codes, and the ASCII values to set into them. For an illustration of key numbers, refer to the diagram at the top right hand side of the computer, or to the chapter entitled 'For your reference....'.

Keyboard expansions:

Effectively simulates the `KEY` command in BASIC.

AMSDOS

This program relinquishes control from CP/M and returns to the built-in AMSTRAD BASIC, from which the AMSDOS disc commands will be available.

Disc Generation

FORMAT

The computer supports three disc formats, one of which has two variants.

The usual format is system format, obtained by using the standard `FORMAT` command. The system tracks are read from the disc containing the `FORMAT.COM` program and are automatically written to the destination disc.

The other formats are obtained by adding a single letter as a parameter to the command, separated by a single space:

For Data Format type:	<code>FORMAT D</code>
For IBM Format type:	<code>FORMAT I</code>
For Vendor Format type:	<code>FORMAT V</code>

WARNING

The licence agreement for your CP/M (which is electronically serial-number encoded) permits its use on a single computer system only. In particular this means that you are prohibited from giving any other person a disc with YOUR serial-numbered copy of CP/M on it. Because every system disc you make has your CP/M on it, you must be careful, therefore, not to sell, exchange or in any other way part with, any system format disc. Instead you must format a disc in Vendor format, which is identical to system format except that the system tracks are blank, and then copy the relevant software onto that disc using `FILE COPY` or `PIP`. Be careful that the software you copy in this way is not itself copyright or subject to a licence agreement.

If you receive software on a disc in Vendor format, in order to use it conveniently, you may either copy it to a system disc by using **FILECOPY** or **PIP**, or alternatively convert the disc to a system disc by adding your CP/M to it. This is achieved with the **BOOTGEN** and **SYSGEN** commands. Carefully read the End User Licence Agreement section of this manual.

MOVCPM

Sometimes it is required to construct a version of CP/M which does not load into memory at the standard position. This may be because you wish to reserve some memory for other purposes. Therefore CP/M itself must be moved to a lower portion of memory, and it is possible to locate CP/M at any position in memory on a 256-byte boundary. The position is specified by a size parameter in the range 64 to 179. This parameter indicates the number of 256-byte areas available for CP/M and transient programs.

The resulting relocated CP/M can either be written to the system disc using **SYSGEN** or saved using a command prompted by the **MOVCPM** program.

The syntax of the command is **MOVCPM <size> ***

MOVCPM 178 *

....will make a CP/M 256 bytes lower in memory than the standard version (which is created with the maximum possible size of 179).

SYSGEN

SYSGEN writes the result of a **MOVCPM** command onto the system tracks of a system or Vendor disc. There are three options:

SYSGEN *

....will write the CP/M generated by an immediately preceding **MOVCPM** command.

SYSGEN <filename>

....will read in the specified file, which will probably have been saved after a **MOVCPM** command, and write that to the system tracks, e.g. **SYSGEN CPM44.COM**

SYSGEN

....(with no parameters) will prompt for a source and destination disc, and therefore copies the system tracks from one disc to another. This is the option to use if upgrading a Vendor disc to a system disc.

BOOTGEN

BOOTGEN, which will prompt for source and destination discs, copies Sector 1, track 0 (the loader) and the configuration sector from one disc onto another. Use **BOOTGEN** as part of the process of upgrading a Vendor disc to a system disc, or when you want to distribute a newly designed configuration sector around a number of discs.

Advanced programming

The following programs are available on the disc for specialist use, and it is recommended that the user consult **SOFT 159 - A Guide to CP/M**, or other reference works.

ASM	8080 Assembler.
DDT	8080 Assembly code debugging aid.
DUMP	Hexadecimal file dump utility.
ED	A simple context editor.
SUBMIT	Console Command Mode batch processing.
XSUB	Transient program batch processing.

Chapter 6

Introduction to LOGO

This Section is intended to introduce the subject of LOGO, with examples, and provide a guide to the commands available. It is not intended to be an exhaustive tutorial or reference guide. Such a book is provided by 'A guide to LOGO' (SOFT160)

Subjects covered are:

- ★ Concept of LOGO
- ★ Loading and Running Dr. LOGO
- ★ Turtle Graphics
- ★ Writing your own procedures
- ★ Editing your own procedures

What is LOGO?

LOGO can help you grow as a programmer, whether or not you have ever programmed before.

LOGO is a powerful programming language that is rapidly gaining popularity because it is so easy to learn and use.

You use procedures as building blocks to create LOGO programs. Dr. LOGO itself is a collection of procedures, called primitives, that you use to build your own programs.

During the 1970's, a team of computer scientists and educators under the direction of Seymour Papert, developed LOGO with turtle graphics to allow very young children to program and use a computer.

They developed the turtle so that young learners could have, as Papert says, 'an object to think with', a tool to help them learn in new ways.

In the form of an arrow head, the turtle can be directed across the screen by the use of simple commands.

Dr. LOGO

Dr. LOGO is a thoughtful implementation of LOGO which has been specially customised for the AMSTRAD personal computer to make it even easier to program. Extensions have been included to make available the powerful sound facilities of the CPC664 and program editing is made easy by the inclusion of the cursor key cluster.

Getting Started

To operate Dr. LOGO, insert a copy of Side 2 of your master disc into the computer's disc drive. (Part 10 of the Foundation Course contains instructions on how to copy the master disc.)

Press the **[CTRL][SHIFT]** and **[ESC]** keys simultaneously to reset the computer.

Type in **! CPM [ENTER]** and LOGO will load automatically.

The following Dr. LOGO wake-up message will be displayed on your monitor:

```
      Welcome to
      Amstrad LOGO V1.1
Copyright (c) 1983, Digital Research
      Pacific Grove, California

      Dr. Logo is a trademark of
      Digital Research

      Product No. 6002-1232

      Please Wait.
```

This greeting will soon disappear and a question mark prompt **?** will appear on your screen.

The question mark tells you that Dr. LOGO is waiting for you to type something at your keyboard.

First Steps

Try typing in (using lower case letters):

```
fd 60 [ENTER]
```

....and you will see a turtle appear which then moves forward 60 units leaving a line behind it from where it started, to where it finished. The screen will clear giving a large graphics area and a smaller text area with the **?** prompt near the bottom of the screen.

Dr. LOGO will often decide to re-arrange the screen so as to give either a large text area or large graphics area, for your convenience.

Type in:

```
rt 90 [ENTER]
```

....and the turtle will move 90 degrees to the right. (From now on we will assume that you press the **[ENTER]** key after every line of command.)

Now type in:

```
fd 60
```

....and another line will be drawn the same length and at right angles to the first line.

Experiment with the simple instructions `fd`, `bk` (short for back), `rt` and `lt` (short for left) to see what happens on the screen.

Dr. LOGO Procedures

A procedure is a list of instructions that tells Dr. LOGO how to do a task.

You will probably write your first procedures by adding to those already built into Dr. LOGO, these are called 'primitives'.

`fd`, `bk`, `rt` and `lt` are all built-in primitives which you may use at any time as building blocks to write your own procedures.

Another very useful built-in primitive is `cs` which clears the screen and sends the turtle to its starting position.

Writing a Simple Procedure

It is easy to visualise that if the movements:

```
fd 60 rt 90
```

....were to be repeated 4 times each, a square with sides of 60 units would be drawn.

The same effect can be achieved by writing a simple formula:

```
repeat 4 [fd 60 rt 90]
```

Clear the screen and then try typing this in to check what happens.

To make this formula into a new procedure called 'square' type:

```
to square
repeat 4 [fd 60 rt 90]
end
```

Dr. LOGO will now understand 'square' and each time it encounters the word 'square' it will draw a square on the screen. We could have given this procedure any name, but we chose 'square' to remind us what it does.

Dr. LOGO allows us to type in a whole set of commands together so the instructions `square rt 45 square`, will draw two squares, the second at a 45 degrees angle to the first.

Procedures with parameters

It is possible to make a procedure to which we can say 'how much', in the same way that we can say 'how much' to built-in procedure. To make a procedure that will draw squares of different sized sides, the definition of 'square' can be altered to:

```
to squareanysize :side
  repeat 4 [fd :side rt 90]
end
```

This new procedure introduces the idea of a 'variable', which in this case is called `:side`

You will notice that the variable `:side` is preceded by a colon, this indicates to Dr. LOGO that `:side` is a variable rather than a command.

When we use procedure `squareanysize :side` must have a value. Hence an instruction `squareanysize 150` would produce a square with sides of 150 units. Try adding two procedures together and see what happens. For example from an instruction:

```
cs squareanysize 100 rt 45 squareanysize 150
```

....the turtle will draw two squares of differing sized sides, and one will be at a 45 degrees angle to the other.

Notice how Dr. LOGO uses an exclamation mark ! to remind you that a line of commands has split across more than one line of screen.

Using Variables to remember values

Dr. LOGO will also allow us to use variables to remember values, as well as for passing values to a procedure.

First define a new procedure called triangle:

```
to triangle
  repeat 3 [fd :edge rt 120]
end
```

We can test this by typing:

```
make "edge 100
triangle
```

If we want to know the value remembered by `:edge` we can just type `:edge` after the `?` prompt and Dr. LOGO will print the value.

Finally we can use our variable `:edge` in a new procedure to draw a pattern. Notice how the value of `:edge` is increased by adding to its previous value so that each time we draw the pattern it gets bigger.

```
to pattern
  triangle lt 60 triangle rt 60
  make "edge :edge+4
  pattern
end
make "edge 10
cs pattern
```

When you have seen enough, press **[ESC]** to stop the program.

Editing programs and procedures

Dr. LOGO allows us to correct typing mistakes and to alter procedures we have defined. The editing keys to use are:

The cursor keys `←` `→` `␣` which move the cursor by one character or line at a time.

The cursor keys `←` `→` `␣` pressed at the same time as holding down **[CTRL]** will move the cursor up and down a page, and to the left and right of a line.

[CLR] deletes the character under the cursor, **[DEL]** deletes the character to the left of the cursor.

[ENTER] tells Dr. LOGO that you have finished editing a line of commands or makes a new line if you are editing a procedure.

[ESC] means abandon and **[COPY]** tells Dr. LOGO that you have finished editing a procedure.

When typing in commands or new procedures, simply edit the text in front of you on the screen. Any characters other than those mentioned above will be inserted into the text at the cursor position.

To edit an existing procedure use the command `ed`

Dr. LOGO will display the old version of the procedure on the screen for you and you can use all the commands above to move the cursor around and change what it says.

Try editing the procedure `patterns` by typing `ed "patterns`

Experiment with the editing keys. If when you have finished if you press **[ESC]**, then Dr. LOGO will abandon what is on the screen and give you back the original unedited version.

Type `ed "patterns` again, and after changing the number 4 to 8, press **[COPY]** to exit, then re-run the procedure and see how the screen output has changed. Remember to set the initial value into `:edge`

Operating hints

The workspace used by Dr. LOGO is divided into nodes. You can see how many are left by typing:

```
nodes
```

Occasionally, when nearly all the nodes are used up, Dr. LOGO will tidy up the workspace and you may see the turtle pause while this happens. You can ask Dr. LOGO to tidy the workspace by typing the command:

```
recycle
```

This will often allow you to continue after Dr. LOGO has complained of not having any more nodes left.

Make sure that there is plenty of disc space left before starting Dr. LOGO in case you decide to save your procedures on disc. You can use the `CAT` command in AMSDOS (see part 7 of the Foundation course).

Glance through the final section ahead and try some of the examples - you won't understand everything first time! As you learn about Dr. LOGO you will be able to use more and more of the commands.

When you have finished with Dr. LOGO type:

```
bye
```

Summary of Dr. LOGO primitives

This section groups together alphabetical lists of Dr. LOGO primitives showing the inputs to use, often with an example.

WORD AND LIST PROCESSING:

(Note that prompts ? and > are shown in the following examples)

ascii

Outputs the ASCII value of the first character in the input word.

```
?ascii "G
71
?ascii "g
103
```

bf

(but first) Outputs all but the first element in the input object.

```
?bf "smiles
miles
?bf [1 2 3]
[2 3]
```

bl

(but last) Outputs all but the last element in the input object.

```
?bl "smiles
smile
?bl [1 2 3 4]
[1 2 3]
```

char

Outputs the character whose ASCII value is the input number.

```
?char 83
S
```

count

Outputs the number of elements in the input object.

```
?count "six
3
?count [0 1 2 3]
4
```

empty?

Outputs TRUE if the input object is an empty word or an empty list, otherwise outputs FALSE.

```
?empty? ""
TRUE
?empty? []
TRUE
?empty? [x]
FALSE
?make "x []
?empty? :x
TRUE
```

first

Outputs the first element of the input object.

```
?first "zebra
z
?first [1 2 3]
1
```

fput

(firstput) Outputs a new object formed by making the first input object the first element in the second object.

```
?fput "s "miles
smiles
?fput 1 [2 3]
[1 2 3]
```

item

Outputs the specified element of the input object.

```
?item 4 "dwarf  
r
```

list

Outputs a list made up of the input objects, retains list's outer brackets (compare with `se`).

```
?(list 1 2 3 4)  
[1 2 3 4]  
?list "big [feet]  
[big [feet]]  
?(list)  
[]
```

se

(sentence) Outputs a list made up of the input objects, removes list's outer brackets (compare with `list`).

```
?make "instr_list rl  
repeat 4 [fd 50 rt 90]  
?run (se "cs :instr_list "ht)
```

Note that the underline character between `instr` and `list` is obtained by pressing **[SHIFT]**.

word

Outputs a word made up of the input words.

```
?word "sun "shine  
sunshine
```

wordp

Outputs **TRUE** if the input object is a word or a number.

```
?wordp "hello  
TRUE  
?wordp []  
FALSE
```

Arithmetic Operations:

cos

Outputs the cosine of the input number of degrees.

```
?cos 60  
0.5000000000017049
```

int

Outputs the integer portion of the input number.

```
?int 4/3  
1
```

random

Outputs a random non-negative integer less than the input number.

```
?random 20
```

sin

Outputs the sine of the input number of degrees.

```
?sin 30  
0.5000000000017049
```

+

Outputs the sum of the input numbers.

```
?+ 2 2  
4  
?2+2  
4
```

—

Outputs the difference of the two input numbers.

```
?- 10 5
5
?10-5
5
```

*

Outputs the product of input numbers.

```
?* 4 6
24
?4*6
24
```

/

Outputs the quotient of the two input numbers.

```
?/ 25 5
5
?25/5
5
```

Logical Operations:

and

Outputs TRUE if the result of all input expressions are true.

```
?and (3<4) (7>4)
TRUE
```

not

Outputs TRUE if the input expression is false.

Outputs FALSE if the input expression is true.

```
?not (3=4)
TRUE
?not (3=3)
FALSE
```

or

Outputs **FALSE** if all input expressions are false.

```
?or "TRUE "FALSE
TRUE
?or (3=4) (1=2)
FALSE
```

=

Outputs **TRUE** if the two input objects are equal; otherwise outputs false.

```
?= "LOGO "LOGO
TRUE
?1=2
FALSE
```

>

Outputs **TRUE** if the first input word is greater than the second; otherwise outputs false.

```
?> 19 20
FALSE
?20>19
TRUE
```

<

Outputs **TRUE** if the first word is less than the second; otherwise outputs false.

```
?< 27 13
FALSE
?13<27
TRUE
```

Variables:

local

Makes the input-named variable(s) accessible only to the current procedure and the procedures it calls.

```
>(local "x "y "z)
```

make

Makes the input-named variable the value of the input object.

```
?make "side 50  
?:side  
50
```

Procedures:

end

Indicates the end of a procedure definition; must stand alone at the beginning of the last line.

```
?to square  
>repeat 4 [fd 50 rt 90]  
>end  
square defined  
?square
```

po

(print out) Displays the definition(s) of the specified procedure(s) or variable(s).

```
?po "square  
to square  
repeat 4 [fd 50 rt 90]  
end  
?po "x  
x is 3
```

pots

(print out titles) Displays the names and titles of all procedures in the workspace.

```
?pots
```

to

Indicates the beginning of a procedure definition.

```
?to square  
>repeat 4 [fd 50 rt 90]  
>end  
square defined
```

Editing:

ed

(edit) Loads the specified procedure(s) and/or variable(s) into the screen editor's buffer.

```
?ed "square
```

Text Screen:

ct

(clear text) Erases all text in the window that currently contains the cursor, then positions the cursor in the upper-left corner of the window.

```
?ct
```

pr

(print) Displays the input object(s) on the text screen, removes list's outer brackets, follows last input with a carriage return (compare with `show` and `type`).

```
?pr [a b c]  
a b c
```

setsplit

Sets the number of lines in the split screen.

```
?setsplit 10
```

show

Displays the input object on the text screen, retains list's outer brackets, follows input with a carriage return (compare with **pr** and **type**).

```
?show [a b c]  
[a b c]
```

ts

(text screen) Selects a full text screen.

```
?ts
```

type

Displays the input object(s) on the text screen, removes list's outer brackets, does not follow last input with a carriage return (compare with **pr** and **show**).

```
?type [a b c]  
a b c
```

Graphic Screen:

Note that the screen is in Mode 1, giving four colours, and that the same co-ordinate system is used as in AMSTRAD BASIC. In other words all screen positions will be rounded to the nearest even-numbered screen dot. Red green and blue colours can have amounts of 0, 1, or 2.

clean

Erases the graphic screen without affecting the turtle.

```
?fd 50  
?clean
```

cs

(clear screen) Erases the graphic screen and puts the turtle at [0,0] heading 0 (north) with the pen down.

```
?rt 90 fd 50  
?cs
```

dot

Plots a dot at the position specified by the input co-ordinate list in the current pen colour.

```
?dot [50 10]
```

fence

Establishes a boundary that limits the turtle to the visible graphics screen. window removes the boundary.

```
?fence  
?fd 300  
Turtle out of bounds
```

fs

(full screen) Selects a full graphic screen.

```
?fs
```

pal

(palette) Outputs numbers representing the amount of red, green, and blue colour assigned to a pen.

```
?pal 2  
[0 2 2]
```

setpal

(set palette) Sets the pen colour palette. Assign an amount of red, green, and blue to a pen.

```
?setpal 3 [1 1 2]  
?pal 3  
[1 1 2]
```

sf

(screen facts) Outputs information about the graphic screen. The format is: [**bgcolour** **screen-state** **split-size** **window-state** **scrunch**] where **bgcolour** is the background pen number, always 0. **screen-state** indicates **SS** (split screen), **FS** (full screen) or **TS** (text screen). **split-size** is the number of text lines displayed on the split screen's text window, and **window-state** indicates **WINDOW**, **WRAP**, or **FENCE** mode. **scrunch** is permanently set to 1.

```
?sf
[0 SS 5 FENCE 1]
```

ss

(split screen) Displays a window of text on the graphic screen.

```
?ss
```

window

Allows the turtle to plot outside the visible graphic screen after a wrap or fence expression.

```
?fence fd 300
Turtle out of bounds
>window
?fd 300
```

wrap

Makes the turtle reappear on the opposite side of the graphic screen when it exceeds the boundary.

```
?cs wrap
?rt 5 fd 1000
?cs window
?rt 5 fd 1000
```

Turtle Graphics:

bk

(back) Moves the turtle the input number of steps in the opposite direction of its heading.

```
?cs fd 150  
?bk 50
```

fd

(forward) Moves the turtle the input number of steps in the direction of its current heading.

```
?fd 80
```

ht

(hide turtle) Makes the turtle invisible; speeds and clarifies drawing.

```
?ht  
?cs fd 50  
?st
```

lt

(left) Rotates the turtle the input number of degrees to the left.

```
?lt 90
```

pd

(pen down) Puts the turtle's pen down; the turtle resumes drawing.

```
?fd 20 pu fd 20  
?pd  
?fd 20
```

pe

(pen erase) Changes the turtle's pen colour to 0, the background colour; the turtle erases drawn lines.

```
?fd 50
?pe
?bk 25
?fd 50
?pd fd 25
```

pu

(pen up) Picks the turtle's pen up; the turtle stops drawing.

```
?fd 30
?pu
?fd 30
?pd fd 30
```

px

(pen exchange) Makes the turtle exchange the colour of any previously coloured pixel in its trail with the reverse or logical colour compliment.

```
?fd 20 pu fd 20
?pd setpc 3 fd 20
?px
?bk 80
?fd 80
?pd bk 100
```

rt

Rotates the turtle the input number of degrees to the right.

```
?rt 90
```

seth

(set heading) Turns the turtle to the absolute heading specified by the input number of degrees; positive numbers turn the turtle clockwise; negative numbers turn the turtle counter-clockwise.

```
?seth 90
```

setpc

(set pen colour) Sets the turtle's pen to that specified by the input number. (0 is the background colour.)

```
?setpc 1
```

setpos

(set position) Moves the turtle to the position specified in the input co-ordinate list.

```
?setpos [30 20]
```

st

(show turtle) Makes the turtle visible if hidden.

```
?ht  
?fd 50  
?st
```

tf

(turtle facts) Outputs information about the turtle. The format is: [*xcor* *ycor* *heading* *penstate* *pencolour n* *shownp*] where *xcor* is the turtle's x co-ordinate, *ycor* is the turtle's y co-ordinate, *heading* indicates the compass direction the turtle is facing, *shownp* is TRUE if the turtle is visible, *penstate* indicates PD (pen down), PE (pen erase), PX (pen exchange) or PU (pen up), *pencolour n* identifies the pen's number.

```
?setpos [15 30]  
?rt 60  
?setpc 3  
?pe  
?ht  
?tf  
[15 30 60 PE 3 FALSE]
```

Workspace Management:

er

(erase) Erases the specified procedure(s) from the workspace.

```
?er "square
```

ern

(erase name) Erases the specified variable(s) from the workspace.

```
?make "side [100]
?make "angle [45]
?:side :angle
[100]
[45]
?ern [side angle]
?:side
side has no value
```

nodes

Outputs the number of free nodes in the workspace.

```
?nodes
```

recycle

Frees as many nodes as possible and reorganises the workspace.

```
?recycle
?nodes
```

Property Lists:

glist

(get list) Outputs a list of all the objects in the workspace that have the input property name in their property lists.

```
?glist ".DEF
```

gprop

(get property) Outputs the property value of the input property name of the input-named object.

```
?make "height "72
?gprop "height ".APV
72
```

plist

(property list) Outputs the property list of the input-named object.

```
?plist "height
[.APV 72]
```

pprop

(put property) Puts the input property pair into the input-named object's property list.

```
?pprop "master ".APV "Scott
?:master
Scott
```

remprop

(remove property) Removes the specified property from the input-named object's property list.

```
?remprop "master ".APV
```

Disc Files:

dir

(directory) Outputs a list of Dr. LOGO file names on the default or specified disc; accepts wild-cards.

```
?dir "a:????????
```

(Study part 1 of the chapter entitled 'AMSDOS and CP/M' for the use of ???????? wild-card characters. Note that the * wild-card is not supported in Dr. LOGO.)

load

Reads the input-named file from the disc into the workspace.

```
?load "myfile
?load "b:shapes
```

save

Writes the contents of the workspace to the input-named disc file.

```
?save "shapes
```

Keyboard and Joystick:

buttonp

(button pressed) Outputs TRUE if the button on the specified joystick is down; numbers 0 or 1 identify the two possible joysticks.

```
?to fire
>label "loop
>if (buttonp 0) [pr [fire 0!]]
>if (buttonp 1) [pr [fire 1!]]
>go "loop
>end
```

The position of the joystick is tested by `paddle`.

keyp

Outputs TRUE if a character has been typed at the keyboard and is waiting to be read.

```
?to inkey
>if keyp [op rc] [op " ]
>end
```

paddle

Returns the state of either joystick 0 or 1. The positions of the joystick are indicated as follows:

Value returned	Meaning
255	Nothing pressed
0	Up
1	Up and right
2	Right
3	Down and right
4	Down
5	Down and left
6	Left
7	Up and left

```
?paddle 0
255
```

The fire buttons are tested by `buttonp`.

rc

(read character) Outputs the first character typed at the keyboard.

```
?make "key rc
```

....then press **X** key....

```
?:key
X
```

rl

(read list) Outputs a list that contains a line typed at the keyboard; input must be followed by a carriage return.

```
?make "instr_list rl
repeat 4 [fd 50 rt 90]
?:instr_list
[repeat 4 [fd 50 rt 90]]
```

rq

(read quote) Outputs a word that contains a line typed at the keyboard; input must be followed by a carriage return.

```
?make "command rq
repeat 3 [fd 60 rt 120]
?:command
repeat 3 [fd 60 rt 120]
```

SOUND:

The sound commands are unique to the AMSTRAD implementation of Dr. LOGO and are similar to their AMSTRAD BASIC counterparts.

Refer to part 9 of the Foundation course for further information.

sound

Put a sound into the sound queue. The format is: [⟨channel status⟩ ⟨tone period⟩ ⟨duration⟩ ⟨volume⟩ ⟨volume envelope⟩ ⟨tone envelope⟩ ⟨noise⟩] The parameters after duration are optional.

```
?sound [1 20 50]
```

env

Set up a volume envelope. The format is: [⟨envelope number⟩ ⟨envelope section(s)⟩]

```
?env [1 100 2 20]
?sound [1 200 300 5 1]
```

ent

Set up a tone envelope. The format is: [⟨envelope number⟩ ⟨envelope section(s)⟩]

```
?ent [1 100 2 20]
?sound [1 200 300 5 1 1]
```

release

Releases sound channels that have been set to a hold state in a sound command. The channels to release are indicated as follows:

Input value	Channels released.
-------------	--------------------

0	None
1	A
2	B
3	A and B
4	C
5	A and C
6	B and C
7	A and B and C

`?release 1`

Flow of Control:

bye

Exits the current session of Dr. LOGO.

`?bye`

co

Ends a pause caused by **[CTRL]Z**, `pause` or `ERRACT`

`?co`

go

Executes the line within the current procedure following a label expression with the same input word.

`>go "loop`

if

Executes one of two instruction lists depending on the value of the input expression; input instructions must be literal lists enclosed in brackets.

`>if (a>b) [pr [a is bigger]]`
`>[pr [b is bigger]]`

label

Identifies the line to be executed after a go expression with the input word.

```
>label "loop
```

op

(output) Makes the input object the output of the procedure and exits the procedure at that point.

```
?op [result]
```

repeat

Executes the input instruction list the input number of times.

```
?repeat 4 [fd 50 rt 90]
```

run

Executes the input instruction list.

```
?make "instr_list [fd 40 rt 90]  
?run :instr_list
```

stop

Stops the execution of the current procedure and returns to TOPLEVEL (the ? prompt) or the calling procedure.

```
?stop
```

wait

Stops procedure execution for the amount of time specified by the input number.

The amount of time = input number * 0.22 seconds.

```
?wait 20
```

Exception Handling:

catch

Traps errors and special conditions that occur during the execution of the input instruction list.

```
>catch "error [+ [] []]  
>pr [I am here]  
I am here
```

error

Outputs a list whose elements describe the most recent error.

```
>catch "error [do.until.error]  
>show error
```

pause

Suspends the execution of the current procedure to allow interaction with the interpreter or editor.

```
>if :size>5 [pause]
```

throw

Executes the line identified by the input name in a previous `catch` expression.

```
?throw "TOPLEVEL
```

System Primitives:

.contents

Displays the contents of Dr. LOGO symbol space.

.deposit

Puts second input number into the absolute memory location specified by the first input number.

.examine

Displays the contents of the absolute memory location specified.

System Variables:**ERRACT**

When `TRUE`, causes a pause when an error occurs, then returns to `TOPLEVEL`.

FALSE

System value.

REDEFP

When `TRUE` allows redefinition of primitives.

TOPLEVEL

`throw "TOPLEVEL` will exit all pending procedures.

TRUE

System value.

System Properties:**.APV**

Associated property value; the value of a global variable.

.DEF

Definition of a procedure.

.PRM

Identifies a primitive.

Chapter 7

For your reference....

This chapter provides much of the reference information that you are likely to require as you learn to use this computer.

Subjects covered:

- ★ Cursor locations and control code extensions
- ★ Interrupts
- ★ ASCII and graphics characters
- ★ Key references
- ★ Sound
- ★ Error messages
- ★ BASIC keywords
- ★ Planners
- ★ Connections
- ★ Printers
- ★ Joysticks
- ★ Disc organisation
- ★ Memory

For a complete guide to the BASIC and firmware for the CPC664, consult AMSOFT manuals, SOFT 945 and SOFT 946 respectively.

Part 1:

Cursor locations and control code extensions

In a variety of applications programs, the text cursor may be positioned outside the current window. Various operations force the cursor to a legal position before they are performed, these being as follows:

-
1. Writing a character
 2. Drawing the cursor 'blob'
 3. Obeying the control codes marked with an asterisk in the list ahead.

The procedure for forcing the cursor to a legal position is as follows:

1. If the cursor is to the right of the right hand edge, then it is moved to the leftmost column of the next line down.
2. If the cursor is to the left of the left hand edge, then it is moved to the rightmost column of the next line up.
3. If the cursor is above the top edge, then the window is rolled down a line and the cursor is set to the top line of the window.
4. If the cursor is below the bottom edge, then the window is rolled up a line and the cursor is set to the bottom line of the window.

The tests and operations are done in the order given. The illegal cursor positions may be zero or negative, which are off to the left or above the window.

Character values in the range 0 to 31 sent to the text screen do not produce a character on the screen but are interpreted as CONTROL CODES (and should not be injudiciously applied). Some of the codes alter the meaning of one or more of the following characters, which are the code's parameters.

A control code sent to the graphics screen will merely print the conventional symbol related to its function if accessed via the keyboard (e.g. &07 'BEL' - **[CTRL] G**). It will execute its control function if addressed using the form of the command:

PRINT CHR\$(&07), or **PRINT "␣"** (where the ␣ is obtained by pressing **[CTRL] G** within the **PRINT** statement).

The codes marked * force the cursor to a legal position in the current window before they are obeyed - but may leave the cursor in an illegal position. The codes and their meanings are described with first their HEX value (&XX), then the decimal equivalent.

Control characters

CTR

J

A

B

C

D

E

F

G

H

Value	Name	Parameter	Meaning
&00 0	NUL		No effect. Ignored.
&01 1	SOH	0 to 255	Print the symbol given by the parameter value. This allows the symbols in the range 0 to 31 to be displayed.
&02 2	STX		Turn off text cursor. Equivalent to <code>CURSOR</code> command with a <code>user switch</code> parameter value of 0.
&03 3	ETX		Turn on text cursor. Equivalent to <code>CURSOR</code> command with a <code>user switch</code> parameter value of 1. Note that to display a cursor from within a BASIC program (other than the automatic cursor generated when BASIC is awaiting keyboard input), a <code>CURSOR</code> command with a <code>system switch</code> parameter value of 1 must be used.
&04 4	EOT	0 to 2	Set screen mode. Parameter taken MOD 4. Equivalent to a <code>MODE</code> command.
&05 5	ENQ	0 to 255	Send the parameter character to the graphics cursor.
&06 6	ACK		Enable Text Screen. (See &15 NAK ahead.)
&07 7	BEL	CTR G	Sound Bleeper. Note that this flushes the sound queues.
&08 8	* BS		Move cursor back one character.

	Value	Name	Parameter	Meaning
I	&09 9	* TAB		Move cursor forward one character.
J	&0A 10	* LF		Move cursor down one line.
K	&0B 11	* VT		Move cursor up one line.
L	&0C 12	FF		Clear text window and move cursor to top left corner. Equivalent to a C L S command.
M	&0D 13	* CR		Move cursor to left edge of window on current line.
N	&0E 14	SO	0 to 15	Set Paper Ink. Parameter taken MOD 16. Equivalent to P A P E R command.
O	&0F 15	SI	0 to 15	Set Pen Ink. Parameter taken MOD 16. Equivalent to P E N command.
P	&10 16	* DLE		Delete current character. Fills character cell with current Paper Ink.
Q	&11 17	* DC1		→ Clear from <u>left edge</u> of window to, and including, the current character position. Fills affected cells with the current Paper Ink.
R	&12 18	* DC2		→ Clear from, and including, the current character position to the <u>right edge</u> of window. Fills affected cells with the current Paper Ink.
S	&13 19	* DC3	CTR S	Clear from start of window to, and including, the current character position. Fills affected cells with the current Paper Ink.

CTR

	Value	Name	Parameter	Meaning
T	&14 20*	DC4	CTR T	Y Clear from, and including, the current character position to the end of window. Fills affected cells with the current Paper Ink.
U	&15 21	NAK	CTR U	X Turn off text screen. The screen will not react to anything sent to it until after an ACK (&06 6) is sent.
V	&16 22	SYN	0 to 1	Parameter MOD 2. Transparent option. 0 disables, 1 enables.
W	&17 23	ETB	0 to 3	Parameter MOD 4 0 sets normal graphics ink mode 1 sets XOR graphics ink mode 2 sets AND graphics ink mode 3 sets OR graphics ink mode
X	&18 24	CAN		Exchange Pen and Paper Inks.
	&19 25	EM	0 to 255 0 to 255 0 to 255 0 to 255 0 to 255 0 to 255 0 to 255 0 to 255	Set matrix for user definable character. Equivalent to a SYMBOL command. Takes nine parameters. The first parameter specifies which character's matrix to set. The next eight specify the matrix. The most significant bit of the first byte corresponds to the top left hand pixel of the character cell, the least significant bit of the last byte corresponds to the bottom right hand pixel of the character cell.
	&1A 26	SUB	1 to 80 1 to 80 1 to 25 1 to 25	Set Window. Equivalent to a WINDOW command. The first two parameters specify the left and right hand edges of the window - the smaller value is taken as the left edge, the larger the right. The second two parameters specify the top and bottom edges of the window - the smaller value is taken as the top edge, the larger the bottom edge.

Value	Name	Parameter	Meaning
&1B 27	ESC		No effect. Ignored.
&1C 28	FS	0 to 15 0 to 31 0 to 31	Set Ink to a pair of colours. Equivalent to an INK command. The first parameter (MOD 16) specifies the Ink, the next two (MOD 32) the required colours. (Parameter values 27 to 31 are un-defined colours.)
&1D 29	GS	0 to 31 0 to 31	Set Border to a pair of colours. Equivalent to a BORDER command. The two parameters (MOD 32) specify the two colours. (Parameter values 27 to 31 are un-defined colours.)
&1E 30	RS		Move cursor to top left hand corner of window.
&1F 31	US	1 to 80 1 to 25	Move cursor to the given position in the current window. Equivalent to a LOCATE command. The first parameter gives the column to move to, the second gives the line.

The housekeeping of the CPC664 is provided by a sophisticated real time operating system. The operating system 'directs the traffic' through the computer from the input to the output.

It primarily interfaces between the hardware and the BASIC interpreter - for example the ink flashing function, where BASIC passes the parameters - and the operating system gets on with the task, with one part determining what is required - and the other part determining the timing of these events.

The machine operating system is generally referred to as the 'firmware', and comprises the machine code routines that are called by the high level commands in BASIC.

If you are tempted to **POKE** around in the machine memory addresses, or **CALL** the sub-routines, save your program and listing before doing so, or you may regret it! The extensive operating system firmware of the CPC664 is described in **SOFT 946**, and is beyond the scope of this introductory user manual.

In order to program extensively using machine code, it will be necessary to use an assembler. The **DEVPAC** assembler from **AMSOF**T comprises a relocatable **Z80** assembler, with editor, disassembler and monitor.

Part 2: Interrupts

The **CPC664** makes extensive use of **Z80** interrupts to provide an operating system that includes several multi-tasking features, exemplified by the **AFTER** and **EVERY** structures described earlier in this manual. The precedence of the event timers is:

- Break (**[ESC][ESC]**)
- Timer 3
- Timer 2 (and the three sound channel queues)
- Timer 1
- Timer 0

Interrupts should be included after considering the consequences of possible intermediate variable states at the point of interruption. The interrupt sub-routine itself should avoid unwanted interaction with the state of variables in the main program.

The sound queues have independent interrupts of equal priority. Once a sound interrupt has started, it is not interrupted by any other sound interrupt. This enables sound interrupt routines to share variables with immunity from the effects mentioned above.

When a sound queue's interrupt is enabled (by using **ON SQ GOSUB**), it will immediately interrupt if the sound queue for that channel is not full, otherwise it will interrupt when the current sound ends and there is room for more in the queue. The action of interrupting disables the event, so the sub-routine must re-enable itself if further interrupts are required.

Both attempting to issue a sound and testing the queue status will also disable a sound interrupt.

Part 3: ASCII and graphics characters

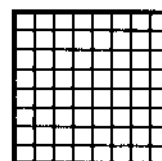
ASCII

The table below illustrates the standard ASCII reference character set using decimal, octal, and hex notation, together with the ASCII codes where appropriate. Each of the CPC664 character cells is also represented in detail in the following pages.

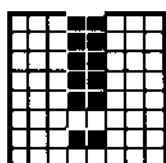
DEC	OCTAL	HEX	ASCII characters	DEC	OCTAL	HEX	ASCII	DEC	OCTAL	HEX	ASCII
0	000	00	NUL ((CTRL)@)	50	062	32	2	100	144	64	d
1	001	01	SOH ((CTRL)A)	51	063	33	3	101	145	65	e
2	002	02	STX ((CTRL)B)	52	064	34	4	102	146	66	f
3	003	03	ETX ((CTRL)C)	53	065	35	5	103	147	67	g
4	004	04	EOT ((CTRL)D)	54	066	36	6	104	150	68	h
5	005	05	ENQ ((CTRL)E)	55	067	37	7	105	151	69	i
6	006	06	ACK ((CTRL)F)	56	070	38	8	106	152	6A	j
7	007	07	BEL ((CTRL)G)	57	071	39	9	107	153	6B	k
8	010	08	BS ((CTRL)H)	58	072	3A	:	108	154	6C	l
9	011	09	HT ((CTRL)I)	59	073	3B	;	109	155	6D	m
10	012	0A	LF ((CTRL)J)	60	074	3C	<	110	156	6E	n
11	013	0B	VT ((CTRL)K)	61	075	3D	=	111	157	6F	o
12	014	0C	FF ((CTRL)L)	62	076	3E	>	112	160	70	p
13	015	0D	CR ((CTRL)M)	63	077	3F	?	113	161	71	q
14	016	0E	SO ((CTRL)N)	64	100	40	@	114	162	72	r
15	017	0F	SI ((CTRL)O)	65	101	41	A	115	163	73	s
16	020	10	DLE ((CTRL)P)	66	102	42	B	116	164	74	t
17	021	11	DC1 ((CTRL)Q)	67	103	43	C	117	165	75	u
18	022	12	DC2 ((CTRL)R)	68	104	44	D	118	166	76	v
19	023	13	DC3 ((CTRL)S)	69	105	45	E	119	167	77	w
20	024	14	DC4 ((CTRL)T)	70	106	46	F	120	170	78	x
21	025	15	NAK ((CTRL)U)	71	107	47	G	121	171	79	y
22	026	16	SYN ((CTRL)V)	72	110	48	H	122	172	7A	z
23	027	17	ETB ((CTRL)W)	73	111	49	I	123	173	7B	{
24	030	18	CAN ((CTRL)X)	74	112	4A	J	124	174	7C	
25	031	19	EM ((CTRL)Y)	75	113	4B	K	125	175	7D	}
26	032	1A	SUB ((CTRL)Z)	76	114	4C	L	126	176	7E	-
27	033	1B	ESC	77	115	4D	M				
28	034	1C	FS	78	116	4E	N				
29	035	1D	GS	79	117	4F	O				
30	036	1E	RS	80	120	50	P				
31	037	1F	US	81	121	51	Q				
32	040	20	SP	82	122	52	R				
33	041	21	!	83	123	53	S				
34	042	22	"	84	124	54	T				
35	043	23	#	85	125	55	U				
36	044	24	\$	86	126	56	V				
37	045	25	%	87	127	57	W				
38	046	26	&	88	130	58	X				
39	047	27	'	89	131	59	Y				
40	050	28	(	90	132	5A	Z				
41	051	29	)	91	133	5B	[				
42	052	2A	*	92	134	5C	\				
43	053	2B	+	93	135	5D	]				
44	054	2C	,	94	136	5E	^				
45	055	2D	-	95	137	5F	_				
46	056	2E	.	96	140	60	`				
47	057	2F	/	97	141	61	a				
48	060	30	0	98	142	62	b				
49	061	31	1	99	143	63	c				

Machine specific graphics character set

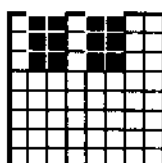
The characters reproduced here are plotted on the standard 8 x 8 cell matrix used to write the screen of the CPC664. User defined characters may be grouped for special effects, and butted next to one another. See the section 'User Defined Characters' in the chapter entitled 'At your leisure....'.



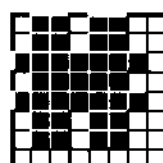
32 &H20
&X00100000



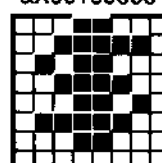
33
&H21
&X00100001



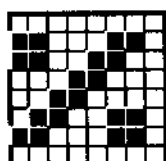
34
&H22
&X00100010



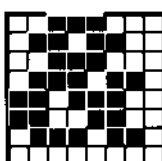
35
&H23
&X00100011



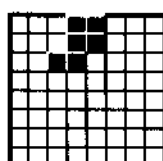
36
&H24
&X00100100



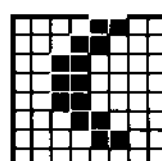
37
&H25
&X00100101



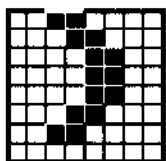
38
&H26
&X00100110



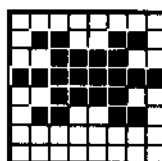
39
&H27
&X00100111



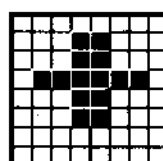
40
&H28
&X00101000



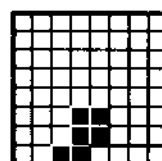
41
&H29
&X00101001



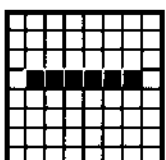
42
&H2A
&X00101010



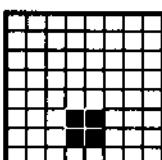
43
&H2B
&X00101011



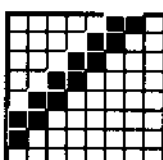
44
&H2C
&X00101100



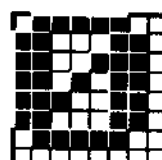
45
&H2D
&X00101101



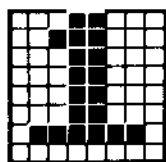
46
&H2E
&X00101110



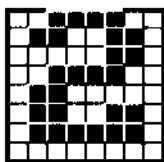
47
&H2F
&X00101111



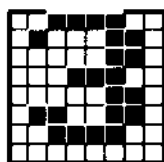
48
&H30
&X00110000



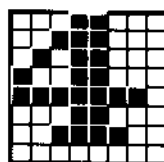
49
&H31
&X00110001



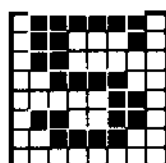
50
&H32
&X00110010



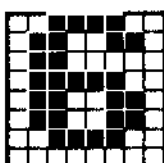
51
&H33
&X00110011



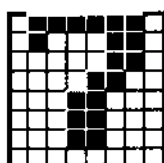
52
&H34
&X00110100



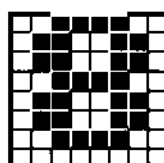
53
&H35
&X00110101



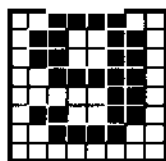
54
&H36
&X00110110



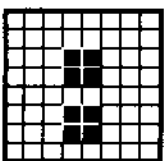
55
&H37
&X00110111



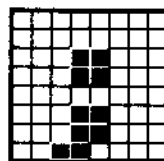
56
&H38
&X00111000



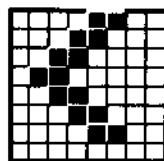
57
&H39
&X00111001



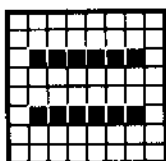
58
&H3A
&X00111010



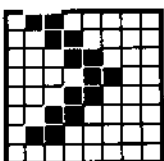
59
&H3B
&X00111011



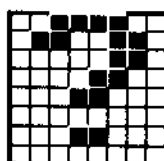
60
&H3C
&X00111100



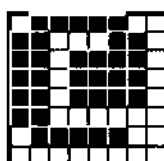
61
&H3D
&X00111101



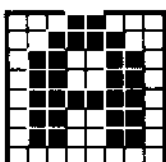
62
&H3E
&X00111110



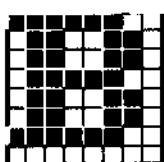
63
&H3F
&X00111111



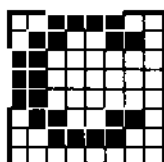
64
&H40
&X01000000



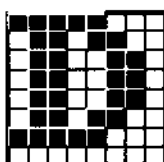
65
&H41
&X01000001



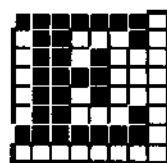
66
&H42
&X01000010



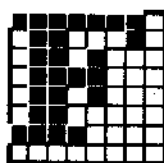
67
&H43
&X01000011



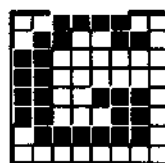
68
&H44
&X01000100



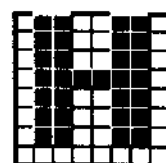
69
&H45
&X01000101



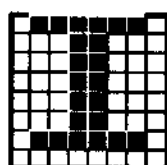
70
&H46
&X01000110



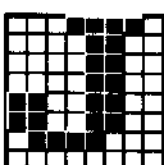
71
&H47
&X01000111



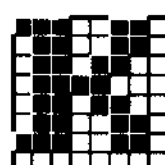
72
&H48
&X01001000



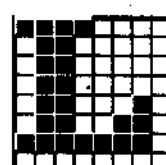
73
&H49
&X01001001



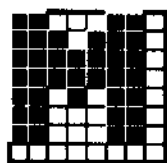
74
&H4A
&X01001010



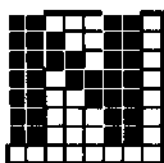
75
&H4B
&X01001011



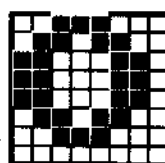
76
&H4C
&X01001100



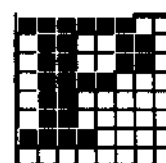
77
&H4D
&X01001101



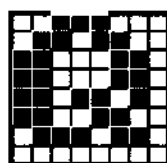
78
&H4E
&X01001110



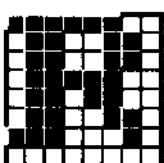
79
&H4F
&X01001111



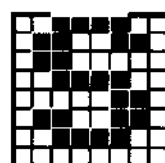
80
&H50
&X01010000



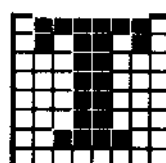
81
&H51
&X01010001



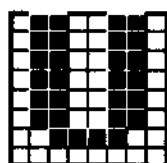
82
&H52
&X01010010



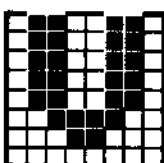
83
&H53
&X01010011



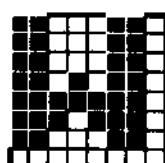
84
&H54
&X01010100



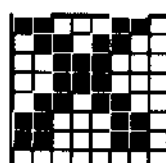
85
&H55
&X01010101



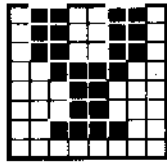
86
&H56
&X01010110



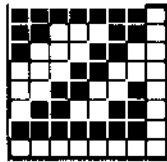
87
&H57
&X01010111



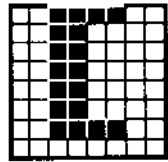
88
&H58
&X01011000



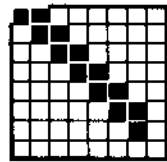
89
&H59
&X01011001



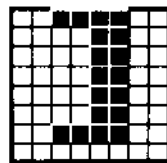
90
&H5A
&X01011010



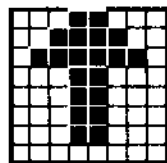
91
&H5B
&X01011011



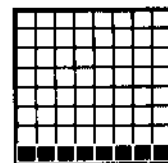
92
&H5C
&X01011100



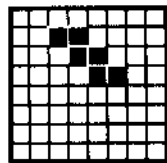
93
&H5D
&X01011101



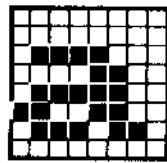
94
&H5E
&X01011110



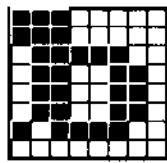
95
&H5F
&X01011111



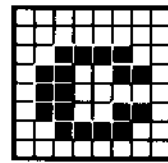
96
&H60
&X01100000



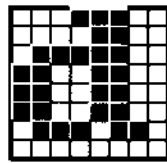
97
&H61
&X01100001



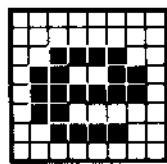
98
&H62
&X01100010



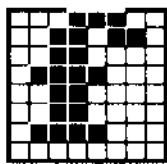
99
&H63
&X01100011



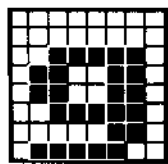
100
&H64
&X01100100



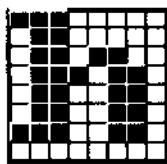
101
&H65
&X01100101



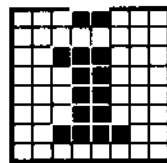
102
&H66
&X01100110



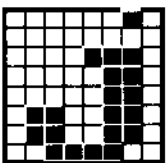
103
&H67
&X01100111



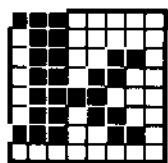
104
&H68
&X01101000



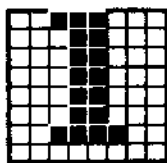
105
&H69
&X01101001



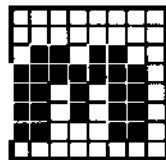
106
&H6A
&X01101010



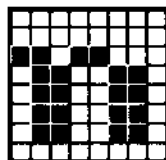
107
&H6B
&X01101011



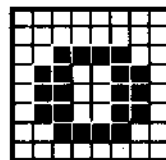
108
&H6C
&X01101100



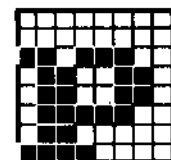
109
&H6D
&X01101101



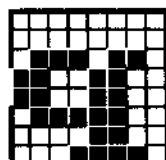
110
&H6E
&X01101110



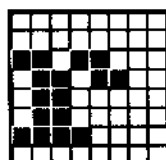
111
&H6F
&X01101111



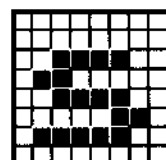
112
&H70
&X01110000



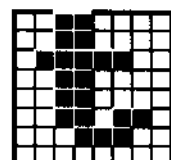
113
&H71
&X01110001



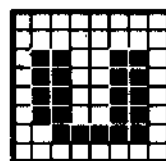
114
&H72
&X01110010



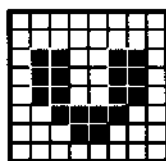
115
&H73
&X01110011



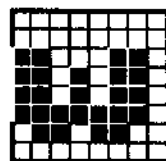
116
&H74
&X01110100



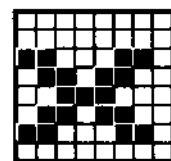
117
&H75
&X01110101



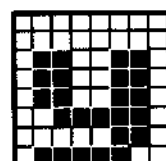
118
&H76
&X01110110



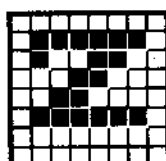
119
&H77
&X01110111



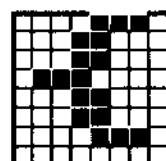
120
&H78
&X01111000



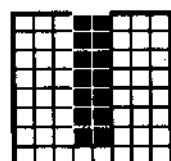
121
&H79
&X01111001



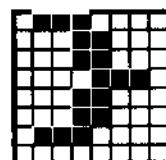
122
&H7A
&X01111010



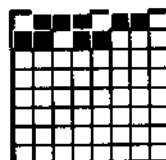
123
&H7B
&X01111011



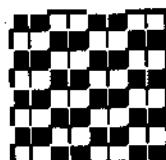
124
&H7C
&X01111100



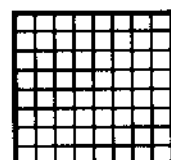
125
&H7D
&X01111101



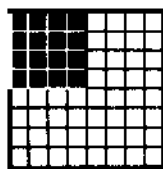
126
&H7E
&X01111110



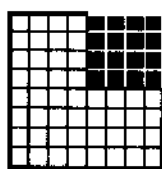
127
&H7F
&X01111111



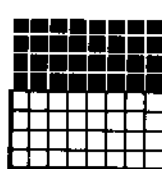
128
&H80
&X10000000



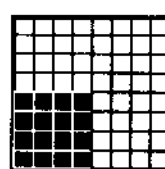
129
&H81
&X10000001



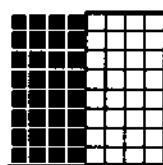
130
&H82
&X10000010



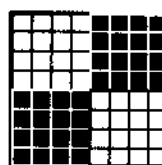
131
&H83
&X10000011



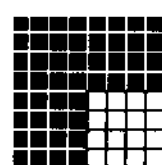
132
&H84
&X10000100



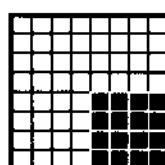
133
&H85
&X10000101



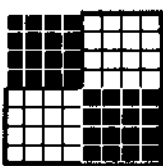
134
&H86
&X10000110



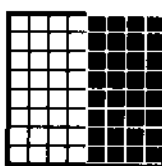
135
&H87
&X10000111



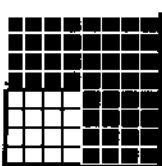
136
&H88
&X10001000



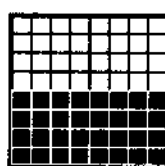
137
&H89
&X10001001



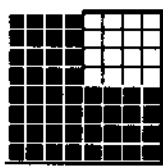
138
&H8A
&X10001010



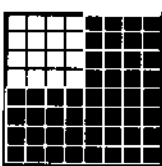
139
&H8B
&X10001011



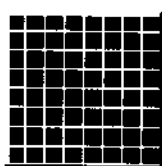
140
&H8C
&X10001100



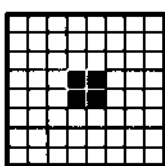
141
&H8D
&X10001101



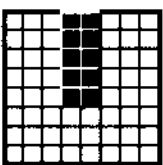
142
&H8E
&X10001110



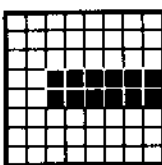
143
&H8F
&X10001111



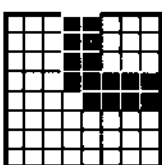
144
&H90
&X10010000



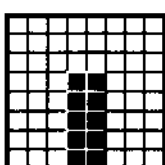
145
&H91
&X10010001



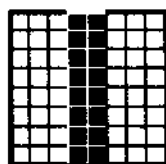
146
&H92
&X10010010



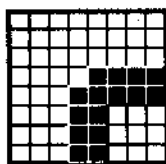
147
&H93
&X10010011



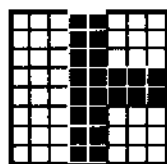
148
&H94
&X10010100



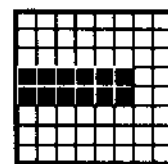
149
&H95
&X10010101



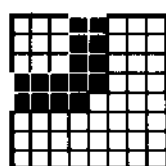
150
&H96
&X10010110



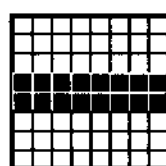
151
&H97
&X10010111



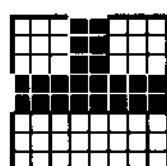
152
&H98
&X10011000



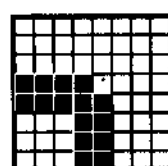
153
&H99
&X10011001



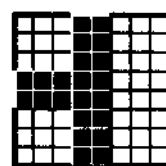
154
&H9A
&X10011010



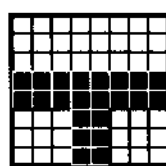
155
&H9B
&X10011011



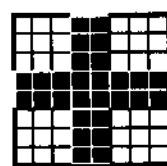
156
&H9C
&X10011100



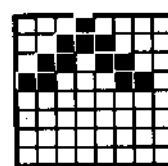
157
&H9D
&X10011101



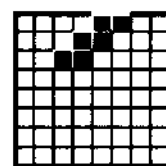
158
&H9E
&X10011110



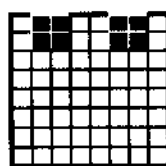
159
&H9F
&X10011111



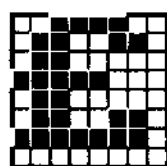
160
&HA0
&X10100000



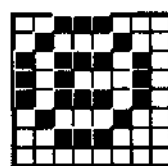
161
&HA1
&X10100001



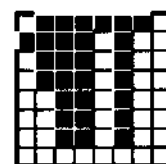
162
&HA2
&X10100010



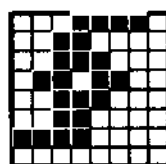
163
&HA3
&X10100011



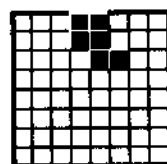
164
&HA4
&X10100100



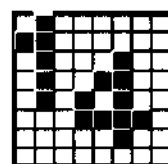
165
&HA5
&X10100101



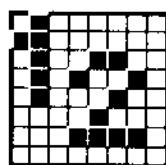
166
&HA6
&X10100110



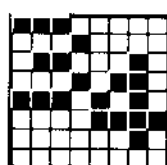
167
&HA7
&X10100111



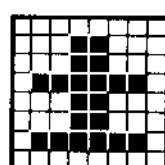
168
&HA8
&X10101000



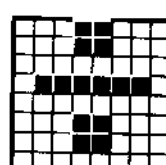
169
&HA9
&X10101001



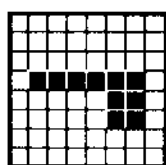
170
&HAA
&X10101010



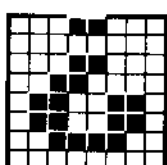
171
&HAB
&X10101011



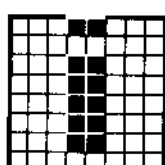
172
&HAC
&X10101100



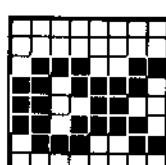
173
&HAD
&X10101101



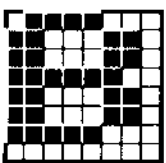
174
&HAE
&X10101110



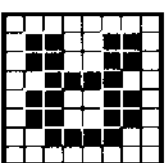
175
&HAF
&X10101111



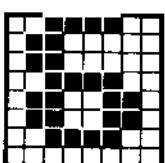
176
&HB0
&X10110000



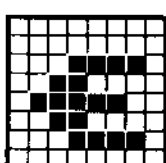
177
&HB1
&X10110001



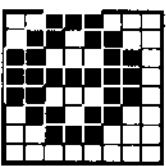
178
&HB2
&X10110010



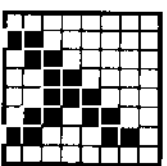
179
&HB3
&X10110011



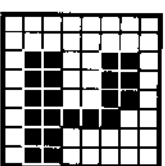
180
&HB4
&X10110100



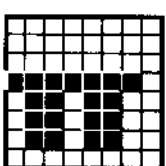
181
&HB5
&X10110101



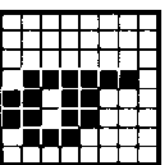
182
&HB6
&X10110110



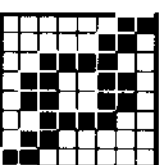
183
&HB7
&X10110111



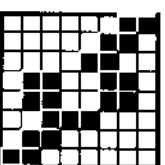
184
&HB8
&X10111000



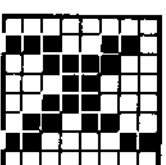
185
&HB9
&X10111001



186
&HBA
&X10111010

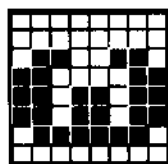


187
&HBB
&X10111011

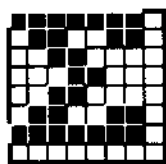


188
&HBC
&X10111100

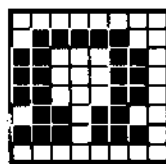
α 0, 20, 34, 66, 34, 28, 50, 66
 β 0, 1, 62, 66, 82, 82, 44, 0
 γ 0, 64, 47, 17, 46, 64, 0, 0.



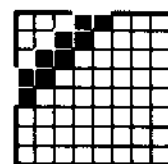
189
&HBD
&X10111101



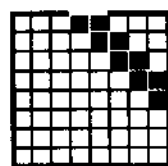
190
&HBE
&X10111110



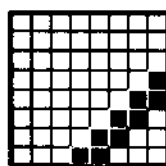
191
&HBF
&X10111111



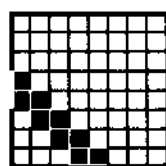
192
&HC0
&X11000000



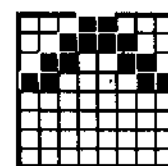
193
&HC1
&X11000001



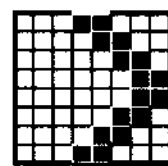
194
&HC2
&X11000010



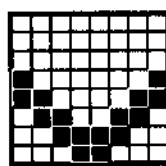
195
&HC3
&X11000011



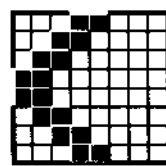
196
&HC4
&X11000100



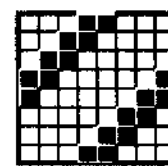
197
&HC5
&X11000101



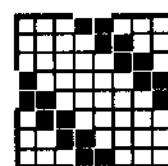
198
&HC6
&X11000110



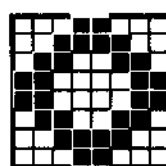
199
&HC7
&X11000111



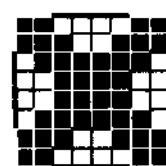
200
&HC8
&X11001000



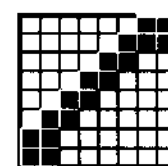
201
&HC9
&X11001001



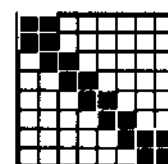
202
&HCA
&X11001010



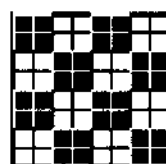
203
&HCB
&X11001011



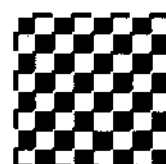
204
&HCC
&X11001100



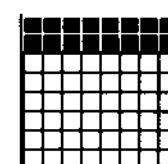
205
&HCD
&X11001101



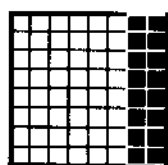
206
&HCE
&X11001110



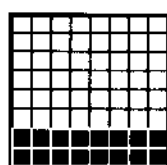
207
&HCF
&X11001111



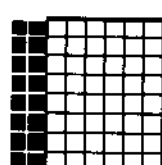
208
&HDO
&X11010000



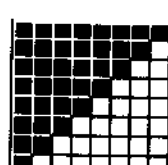
209
&HD1
&X11010001



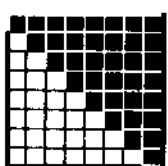
210
&HD2
&X11010010



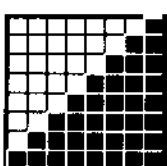
211
&HD3
&X11010011



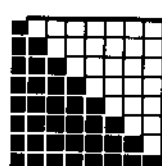
212
&HD4
&X11010100



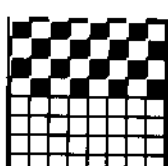
213
&HD5
&X11010101



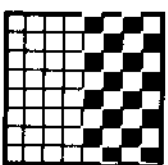
214
&HD6
&X11010110



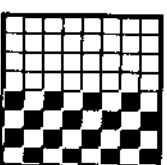
215
&HD7
&X11010111



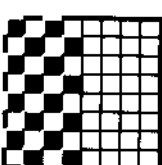
216
&HD8
&X11011000



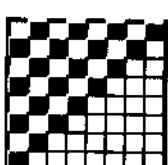
217
&HD9
&X11011001



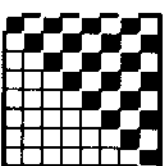
218
&HDA
&X11011010



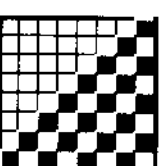
219
&HDB
&X11011011



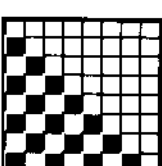
220
&HDC
&X11011100



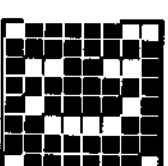
221
&HDD
&X11011101



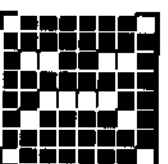
222
&HDE
&X11011110



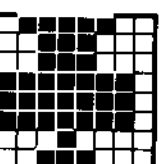
223
&HDF
&X11011111



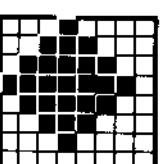
224
&HE0
&X11100000



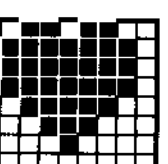
225
&HE1
&X11100001



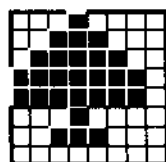
226
&HE2
&X11100010



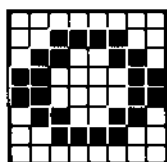
227
&HE3
&X11100011



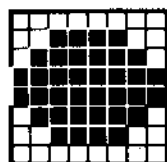
228
&HE4
&X11100100



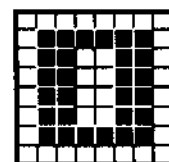
229
&HE5
&X11100101



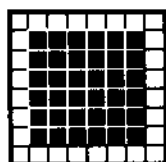
230
&HE6
&X11100110



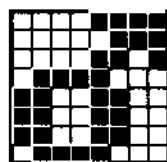
231
&HE7
&X11100111



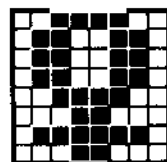
232
&HE8
&X11101000



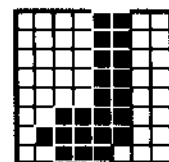
233
&HE9
&X11101001



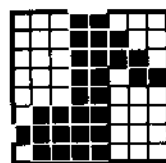
234
&HEA
&X11101010



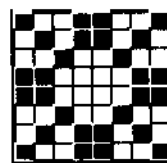
235
&HEB
&X11101011



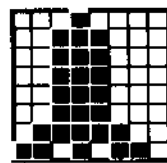
236
&HEC
&X11101100



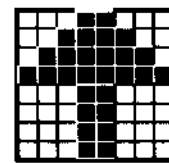
237
&HED
&X11101101



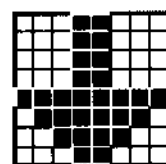
238
&HEE
&X11101110



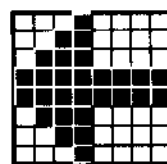
239
&HEF
&X11101111



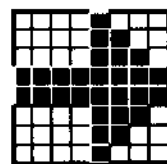
240
&HFO
&X11110000



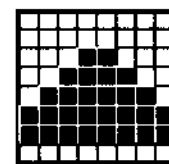
241
&HF1
&X11110001



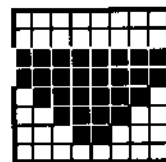
242
&HF2
&X11110010



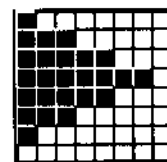
243
&HF3
&X11110011



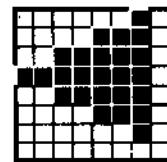
244
&HF4
&X11110100



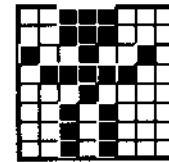
245
&HF5
&X11110101



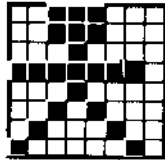
246
&HF6
&X11110110



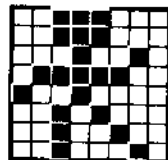
247
&HF7
&X11110111



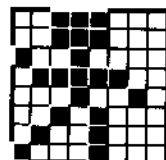
248
&HF8
&X11111000



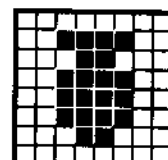
249
&HF9
&X11111001



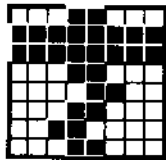
250
&HFA
&X11111010



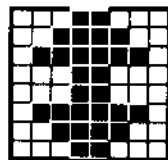
251
&HFB
&X11111011



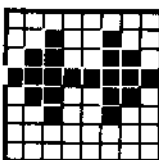
252
&HFC
&X11111100



253
&HFD
&X11111101



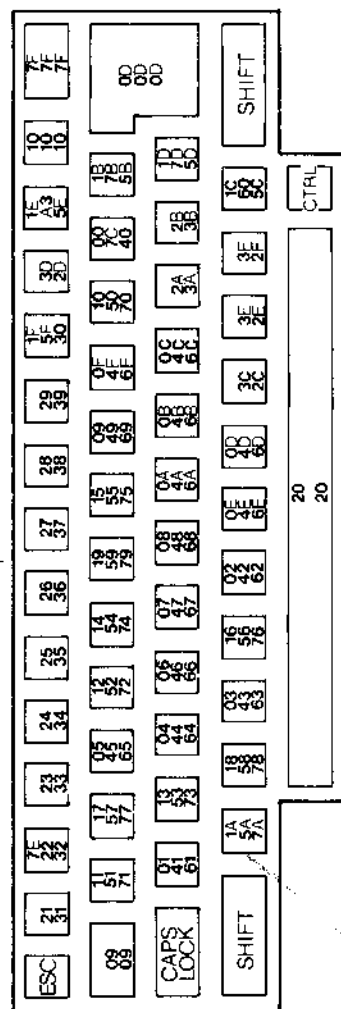
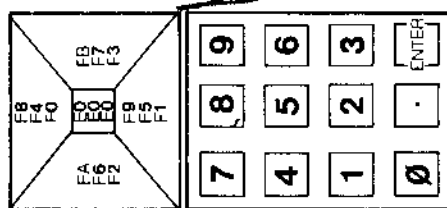
254
&HFE
&X11111110



255
&HFF
&X11111111

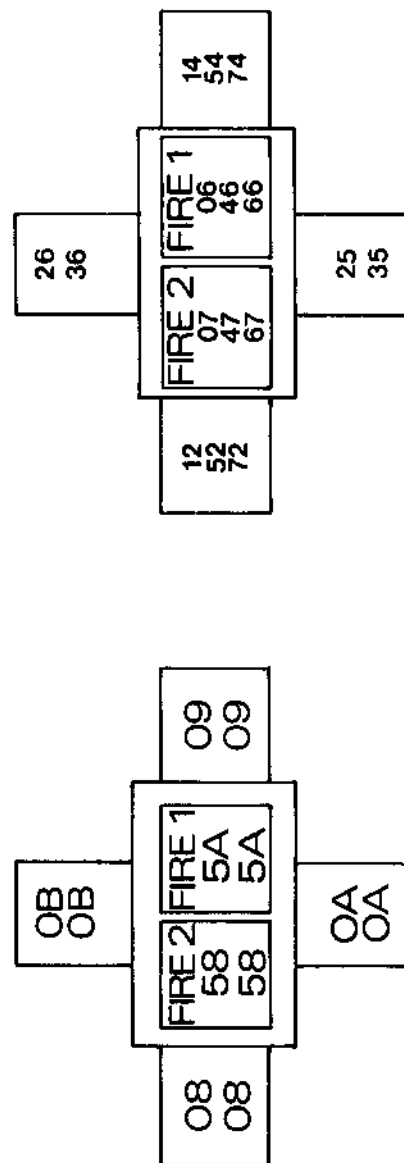
Part 4: Key references

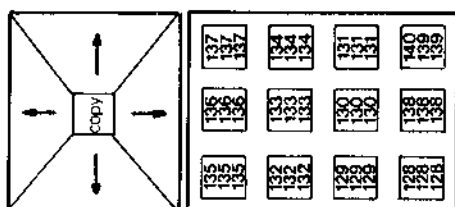
Default ASCII values



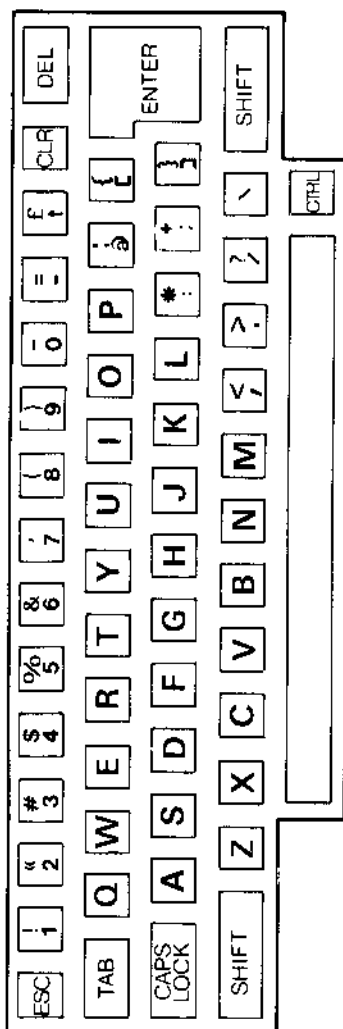
&1A = 26 CTRL
 &5A = 90 SHIFT
 &7A = 122 holding

JOYSTICK 1





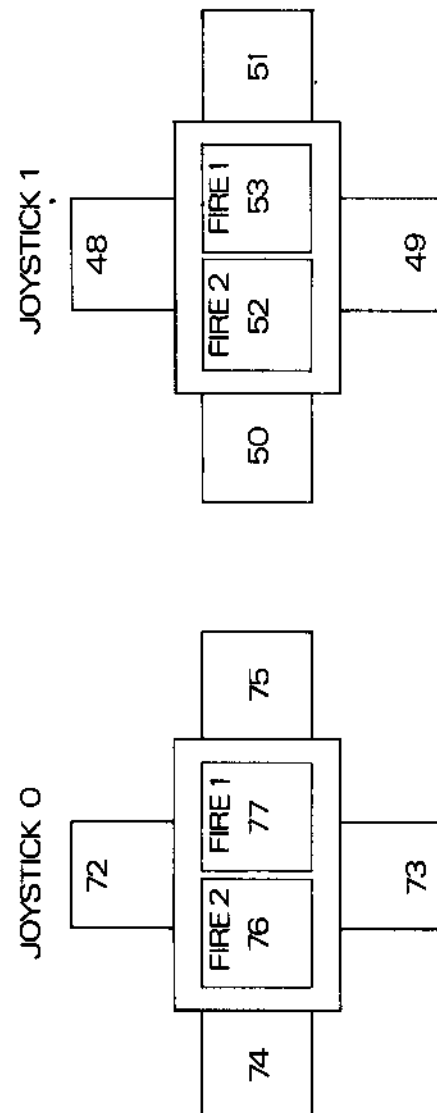
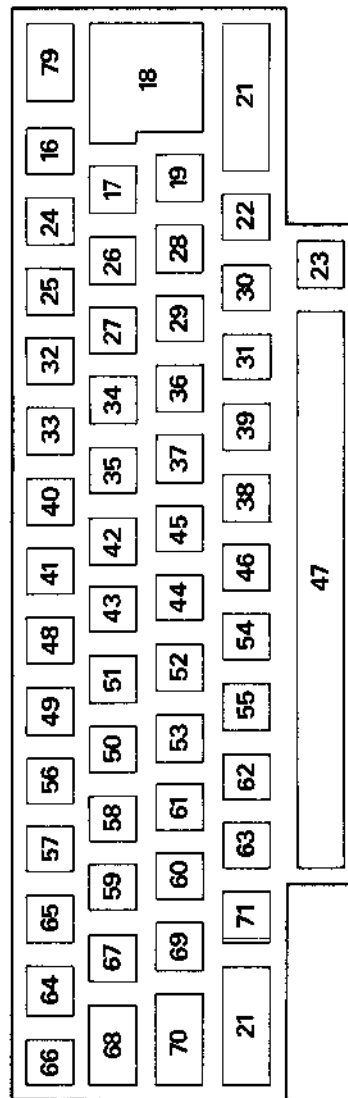
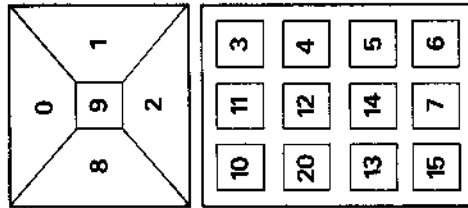
Expansion characters, default locations and values



EXPANSION CHARACTER	DEFAULT SETTING	
	CHARACTER	ASCII VALUE
0 (128)	Ø	&30
1 (129)	1	&31
2 (130)	2	&32
3 (131)	3	&33
4 (132)	4	&34
5 (133)	5	&35
6 (134)	6	&36
7 (135)	7	&37
8 (136)	8	&38
9 (137)	9	&39
10 (138)	.	&2E
11 (139)	[ENTER]	&0D
12 (140)	RUN"[ENTER]	&52 &55 &4E &22 &0D

Note: Expansion characters 13 to 31 (141 to 159) have a null value by default. They have values assigned using the BASIC command KEY, and are assigned to keys using the command KEY DEF.

Key and joystick numbers



Part 5: Sound

Notes and tone periods

The table which follows, gives the recommended <tone period> settings for notes in the usual even tempered scale, for the full eight octave range.

The frequency produced is not exactly the required frequency because the <tone period> setting has to be an integer. The RELATIVE ERROR is the percentage ratio of the difference between the required and actual frequency.

NOTE	FREQUENCY	PERIOD	RELATIVE ERROR	
C	32.703	3822	-0.007%	Octave -3
C#	34.648	3608	+0.007%	
D	36.708	3405	-0.007%	
D#	38.891	3214	-0.004%	
E	41.203	3034	+0.009%	
F	43.654	2863	-0.016%	
F#	46.249	2703	+0.009%	
G	48.999	2551	-0.002%	
G#	51.913	2408	+0.005%	
A	55.000	2273	+0.012%	
A#	58.270	2145	-0.008%	
B	61.735	2025	+0.011%	
NOTE	FREQUENCY	PERIOD	RELATIVE ERROR	
C	65.406	1911	-0.007%	Octave -2
C#	69.296	1804	+0.007%	
D	73.416	1703	+0.022%	
D#	77.782	1607	-0.004%	
E	82.407	1517	+0.009%	
F	87.307	1432	+0.019%	
F#	92.499	1351	-0.028%	
G	97.999	1276	+0.037%	
G#	103.826	1204	+0.005%	
A	110.000	1136	-0.032%	
A#	116.541	1073	+0.039%	
B	123.471	1012	-0.038%	

NOTE	FREQUENCY	PERIOD	RELATIVE ERROR	
C	130.813	956	+0.046%	
C#	138.591	902	+0.007%	
D	146.832	851	-0.037%	
D#	155.564	804	+0.058%	
E	164.814	758	-0.057%	
F	174.614	716	+0.019%	Octave -1
F#	184.997	676	+0.046%	
G	195.998	638	+0.037%	
G#	207.652	602	+0.005%	
A	220.000	568	-0.032%	
A#	233.082	536	-0.055%	
B	246.942	506	-0.038%	

NOTE	FREQUENCY	PERIOD	RELATIVE ERROR	
C	261.626	478	+0.046%	Middle C
C#	277.183	451	+0.007%	
D	293.665	426	+0.081%	
D#	311.127	402	+0.058%	
E	329.628	379	-0.057%	
F	349.228	358	+0.019%	Octave 0
F#	369.994	338	+0.046%	
G	391.995	319	+0.037%	
G#	415.305	301	+0.005%	
A	440.000	284	-0.032%	International A
A#	466.164	268	-0.055%	
B	493.883	253	-0.038%	

NOTE	FREQUENCY	PERIOD	RELATIVE ERROR	
C	523.251	239	+0.046%	
C#	554.365	225	-0.215%	
D	587.330	213	+0.081%	
D#	622.254	201	+0.058%	
E	659.255	190	+0.206%	
F	698.457	179	+0.019%	
F#	739.989	169	+0.046%	Octave 1
G	783.991	159	-0.277%	
G#	830.609	150	-0.328%	
A	880.000	142	-0.032%	
A#	932.328	134	-0.055%	
B	987.767	127	+0.356%	

NOTE	FREQUENCY	PERIOD	RELATIVE ERROR	
C	1046.502	119	-0.374%	
C#	1108.731	113	+0.229%	
D	1174.659	106	-0.390%	
D#	1244.508	100	-0.441%	
E	1318.510	95	+0.206%	
F	1396.913	89	-0.543%	Octave 2
F#	1479.978	84	-0.548%	
G	1567.982	80	+0.350%	
G#	1661.219	75	-0.328%	
A	1760.000	71	-0.032%	
A#	1864.655	67	-0.055%	
B	1975.533	63	-0.435%	

NOTE	FREQUENCY	PERIOD	RELATIVE ERROR	
C	2093.004	60	+0.462%	
C#	2217.461	56	-0.662%	
D	2349.318	53	-0.390%	
D#	2489.016	50	-0.441%	
E	2637.021	47	-0.855%	
F	2793.826	45	+0.574%	Octave 3
F#	2959.955	42	-0.548%	
G	3135.963	40	+0.350%	
G#	3322.438	38	+0.992%	
A	3520.000	36	+1.357%	
A#	3729.310	34	+1.417%	
B	3951.066	32	+1.134%	

NOTE	FREQUENCY	PERIOD	RELATIVE ERROR	
C	4186.009	30	+0.462%	
C#	4434.922	28	-0.662%	
D	4698.636	27	+1.469%	
D#	4978.032	25	-0.441%	
E	5274.041	24	+1.246%	
F	5587.652	22	-1.685%	Octave 4
F#	5919.911	21	-0.548%	
G	6271.927	20	+0.350%	
G#	6644.875	19	+0.992%	
A	7040.000	18	+1.357%	
A#	7458.621	17	+1.417%	
B	7902.133	16	+1.134%	

The above values are all calculated from International A as follows:

$\text{FREQUENCY} = 440 * (2^{\uparrow (\text{OCTAVE} + ((N - 10) / 12)))}$

$\text{PERIOD} = \text{ROUND}(125000 / \text{FREQUENCY})$

....where N is 1 for C, 2 for C#, 3 for D, etc.

Part 6: Error messages

1 Unexpected NEXT

A NEXT command has been encountered while not in a FOR loop, or the control variable in the NEXT command does not match that in the FOR.

2 Syntax Error

BASIC cannot understand the given line because a construct within it is not legal.

3 Unexpected RETURN

A RETURN command has been encountered when not in a sub-routine.

4 DATA exhausted

A READ command has attempted to read beyond the end of the last DATA.

5 Improper argument

This is a general purpose error. The value of a function's argument, or a command parameter is invalid in some way.

6 Overflow

The result of an arithmetic operation has overflowed. This may be a floating point overflow, in which case some operation has yielded a value greater than $1.7E \uparrow 38$ (approx.). Alternatively, this may be the result of a failed attempt to change a floating point number to a 16 bit signed integer.

7 Memory full

The current program or its variables may be simply too big, or the control structure is too deeply nested (nested GOSUBs, WHILEs or FORs).

A MEMORY command will give this error if an attempt is made to set the top of BASIC's memory too low, or to an impossibly high value. Note that an open file has a buffer allocated to it, and that may restrict the values that MEMORY may use.

8 Line does not exist

The line referenced cannot be found.

9 Subscript out of range

One of the subscripts in an array reference is too big or too small.

10 Array already dimensioned

One of the arrays in a DIM statement has already been declared.

11 Division by zero

May occur in real division, integer division, integer modulus or in exponentiation.

12 Invalid direct command

The last command attempted is not valid in direct mode.

13 Type mismatch

A numeric value has been presented where a string value is required or vice versa, or an invalidly formed number has been found in READ or INPUT.

14 String space full

So many strings have been created that there is no further room available, even after 'garbage collection'.

15 String too long

String exceeds 255 characters in length. May be generated by appending strings together.

16 String expression too complex

String expressions may generate a number of intermediate string values. When the number of these values exceeds a reasonable limit, this error results.

17 Cannot CONTinue

For one reason or another the current program cannot be restarted using `CONT`. Note that `CONT` is intended for restarting after a `STOP` command, **[ESC] [ESC]**, or an error, and that any alteration of the program in the meantime makes a restart impossible.

18 Unknown user function

No `DEF FN` has been executed for the `FN` just invoked.

19 RESUME missing

The end of the program has been encountered while in error processing mode (i.e. in an `ON ERROR GOTO` routine).

20 Unexpected RESUME

`RESUME` is only valid while in error processing mode (i.e. in an `ON ERROR GOTO` routine).

21 Direct command found

When loading a file, a line without a line number has been found.

22 Operand missing

BASIC has encountered an incomplete expression.

23 Line too long

A line when converted to BASIC internal-form becomes too big.

24 EOF met

An attempt has been made to read past end of the file input stream.

25 File type error

The file being read is not of a suitable type. `OPEN IN` is only prepared to open ASCII text files. Similarly, `LOAD`, `RUN`, etc, are only prepared to deal with file types produced by `SAVE`.

26 NEXT missing

Cannot find a NEXT to match a FOR command. A line number accompanying this message indicates the FOR command to which this error applies.

27 File already open

An OPENIN or OPENOUT command has been executed before the previously opened file has been closed.

28 Unknown command

BASIC cannot find a taker for an external command, i.e. a command preceded by a bar |.

29 WEND missing

Cannot find a WEND to match a WHILE command.

30 Unexpected WEND

Encountered a WEND when not in a WHILE loop, or a WEND that does not match the current WHILE loop.

31 File not open

(See the section ahead entitled 'Disc errors'.)

32 Broken in

(See the section ahead entitled 'Disc errors'.)

Disc errors

There are several errors that may occur during the processing of any filing operations. BASIC handles all such errors as ERROR number 32, however more specific information may be returned by the function DERR when this error number is detected. This returns values as follows:

AMSDOS error	DERR value	Cause of error
0	0 or 22	[ESC] has been pressed.
14	142 (128+14)	The stream is not in a suitable state.
15	143 (128+15)	Hard end of file has been reached.
16	144 (128+16)	Bad command, usually an incorrect filename.
17	145 (128+17)	File already exists.
18	146 (128+18)	File does not exist.
19	147 (128+19)	Directory is full.
20	148 (128+20)	Disc is full.
21	149 (128+21)	Disc changed while files were open.
22	150 (128+22)	File is Read/Only.
26	154 (128+26)	Soft end of file has been detected.

If AMSDOS has already reported an error, then bit 7 is set; hence the value of DERR is offset by 128. = $128 + \text{error_code}$

Other values returned by DERR originate from the disc controller and are bit significant, always with bit 6 set. Bit 7 indicates whether the error has been reported by AMSDOS (as explained above). The significance of each bit is as follows:

Bit	Significance
0	Address mark missing.
1	Not writable - disc is write protected.
2	No data - can't find the sector.
3	Drive not ready - no disc in the drive.
4	Overflow error.
5	Data error - CRC error.
6	Always set to 1 to indicate error from disc controller.
7	Set to 1 if error has already been reported by AMSDOS.

ERR may also return 31 if access was attempted when no file was open. The usual way in which one may use ERR and DERR would be to include an ON ERROR GOTO which calls a short routine that checks if ERR has the value 31 or 32, and if it is 32, DERR could be interrogated to give more detailed information regarding the nature of the error. For example:

```
10 ON ERROR GOTO 1000
20 OPENOUT "myfile.asc"
30 WRITE #9,"test-data"
40 CLOSEOUT
50 END
1000 amsdoserr=(DERR AND &7F):REM mask off bit 7
1010 IF ERR<31 THEN END
1020 IF ERR=31 THEN PRINT "are you sure you've typed
      line 20 correctly?":END
1030 IF amsdoserr=20 THEN PRINT "disc is full, suggest
      you use a new data disc":END
1040 IF amsdoserr=&X01001000 THEN PRINT "put a disc in
      the drive, then press a key":WHILE INKEY$=""
      WEND:RESUME
1050 END
```

102 = 01111111

if Disk full > 0

Part 7: BASIC Keywords

The following is a list of all AMSTRAD CPC664 BASIC keywords. As such, they are reserved and can NOT be used as variable names.

ABS, AFTER, AND, ASC, ATN, AUTO

BIN\$, BORDER

CALL, CAT, CHAIN, CHR\$, CINT, CLEAR, CLG, CLOSEIN, CLOSEOUT, CLS,
CONT, COPYCHR\$, COS, CREAL, CURSOR

DATA, DEC\$, DEF, DEFINT, DEFREAL, DEFSTR, DEG, DELETE, DERR, DI,
DIM, DRAW, DRAWR

EDIT, EI, ELSE, END, ENT, ENV, EOF, ERASE, ERL, ERR, ERROR, EVERY,
EXP

FILL, FIX, FN, FOR, FRAME, FRE

GOSUB, GOTO, GRAPHICS

HEX\$, HIMEM

IF, INK, INKEY, INKEY\$, INP, INPUT, INSTR, INT

JOY

KEY

LEFT\$, LEN, LET, LINE, LIST, LOAD, LOCATE, LOG, LOG10, LOWERS\$

MASK, MAX, MEMORY, MERGE, MIDS\$, MIN, MOD, MODE, MOVE, MOVER

NEXT, NEW, NOT

ON, ON BREAK, ON ERROR GOTO 0, ON SQ, OPENIN, OPENOUT, OR,
ORIGIN, OUT

PAPER, PEEK, PEN, PI, PLOT, PLOTR, POKE, POS, PRINT

RAD, RANDOMIZE, READ, RELEASE, REM, REMAIN, RENUM, RESTORE,
RESUME, RETURN, RIGHT\$, RND, ROUND, RUN

SAVE, SGN, SIN, SOUND, SPACES\$, SPC, SPEED, SQ, SQR, STEP, STOP,
STR\$, STRING\$, SWAP, SYMBOL

TAB, TAG, TAGOFF, TAN, TEST, TESTR, THEN, TIME, TO, TROFF, TRON

UNT, UPPER\$, USING

VAL, VPOS

WAIT, WEND, WHILE, WIDTH, WINDOW, WRITE

XOR, XPOS

YPOS

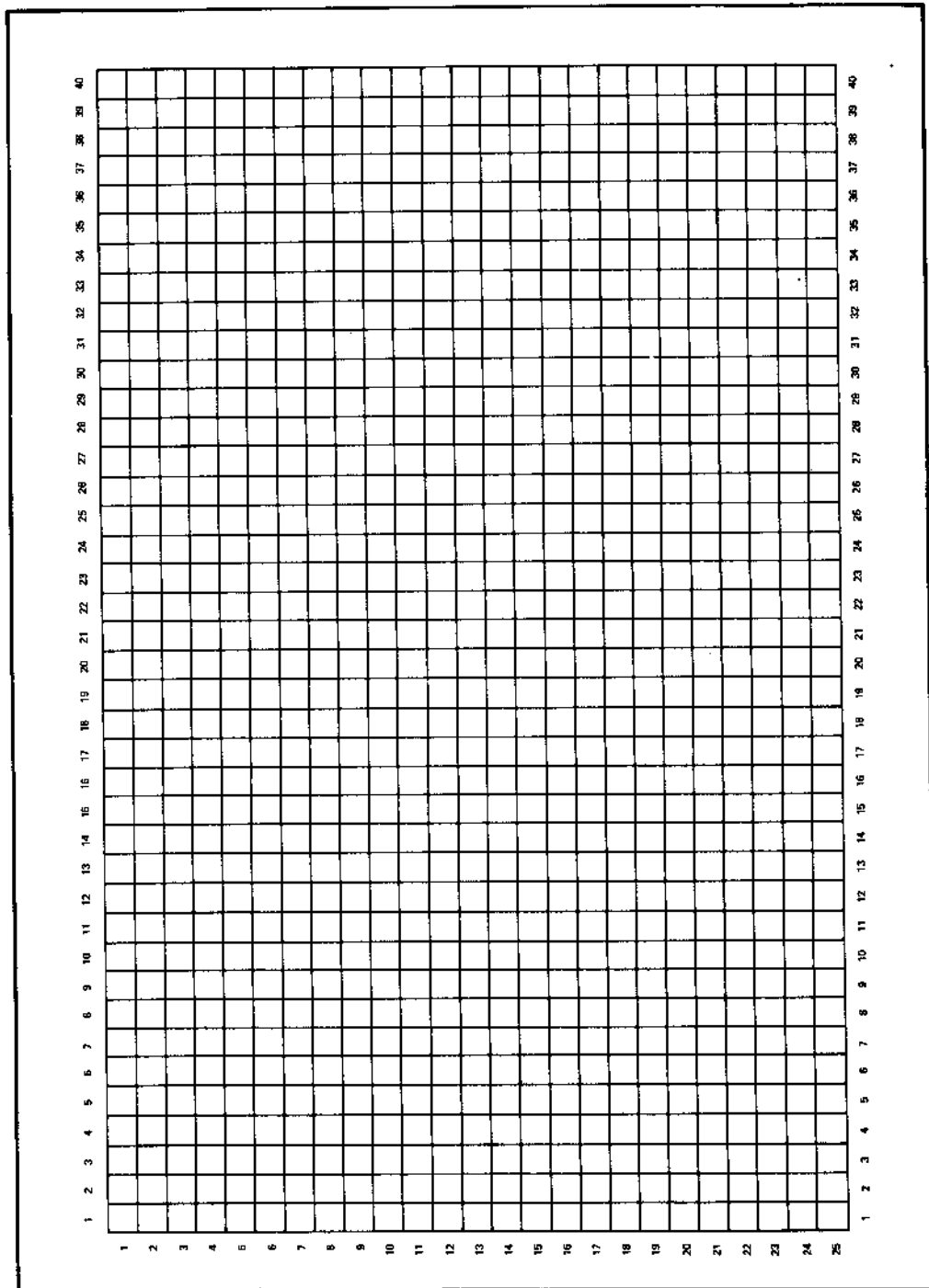
ZONE

Part 8: Planners

Text and window planner - MODE 0 (20 columns)

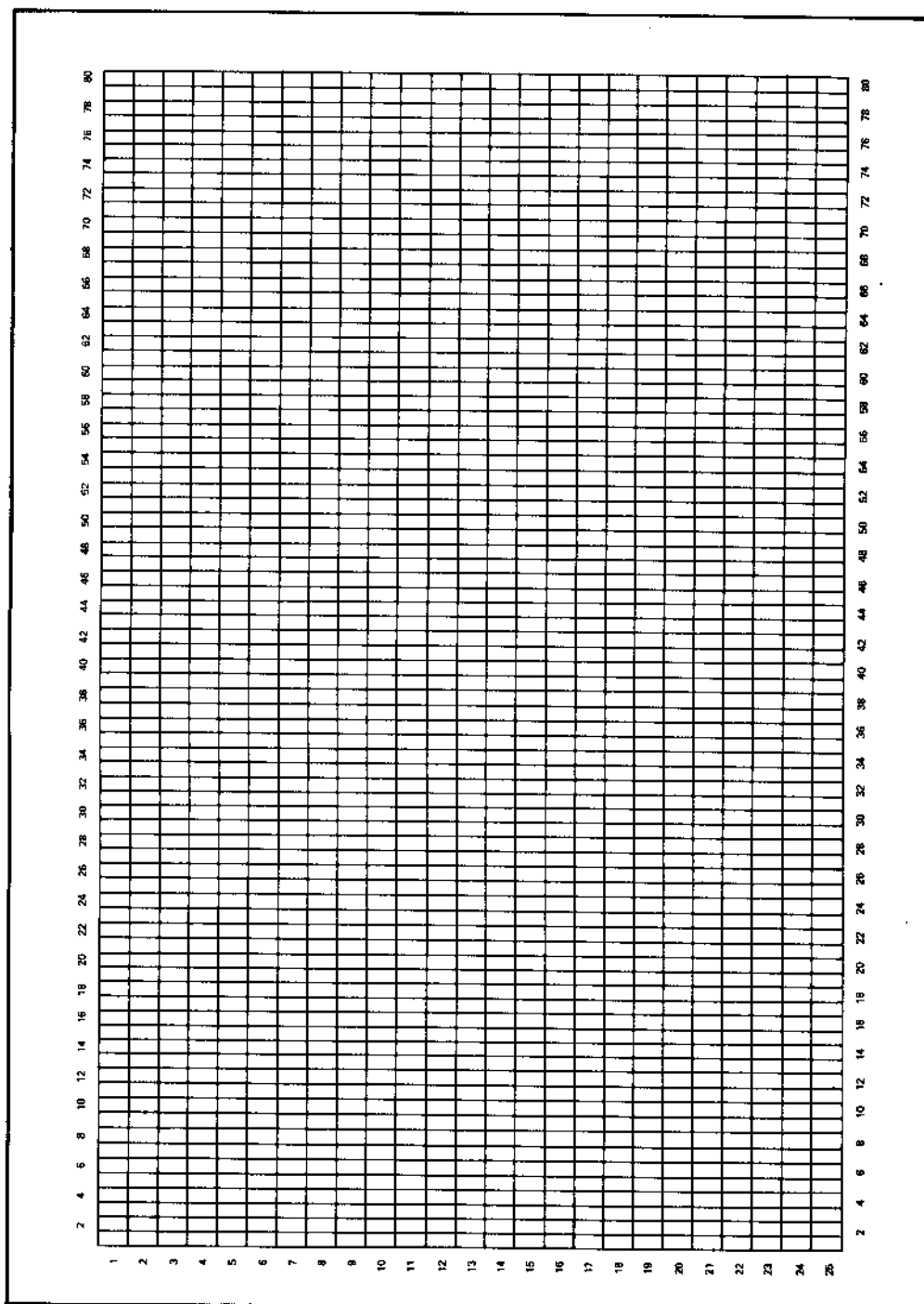
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1																			
2																			
3																			
4																			
5																			
6																			
7																			
8																			
9																			
10																			
11																			
12																			
13																			
14																			
15																			
16																			
17																			
18																			
19																			
20																			
21																			
22																			
23																			
24																			
25																			
26																			

Text and window planner - MODE 1 (40 columns)

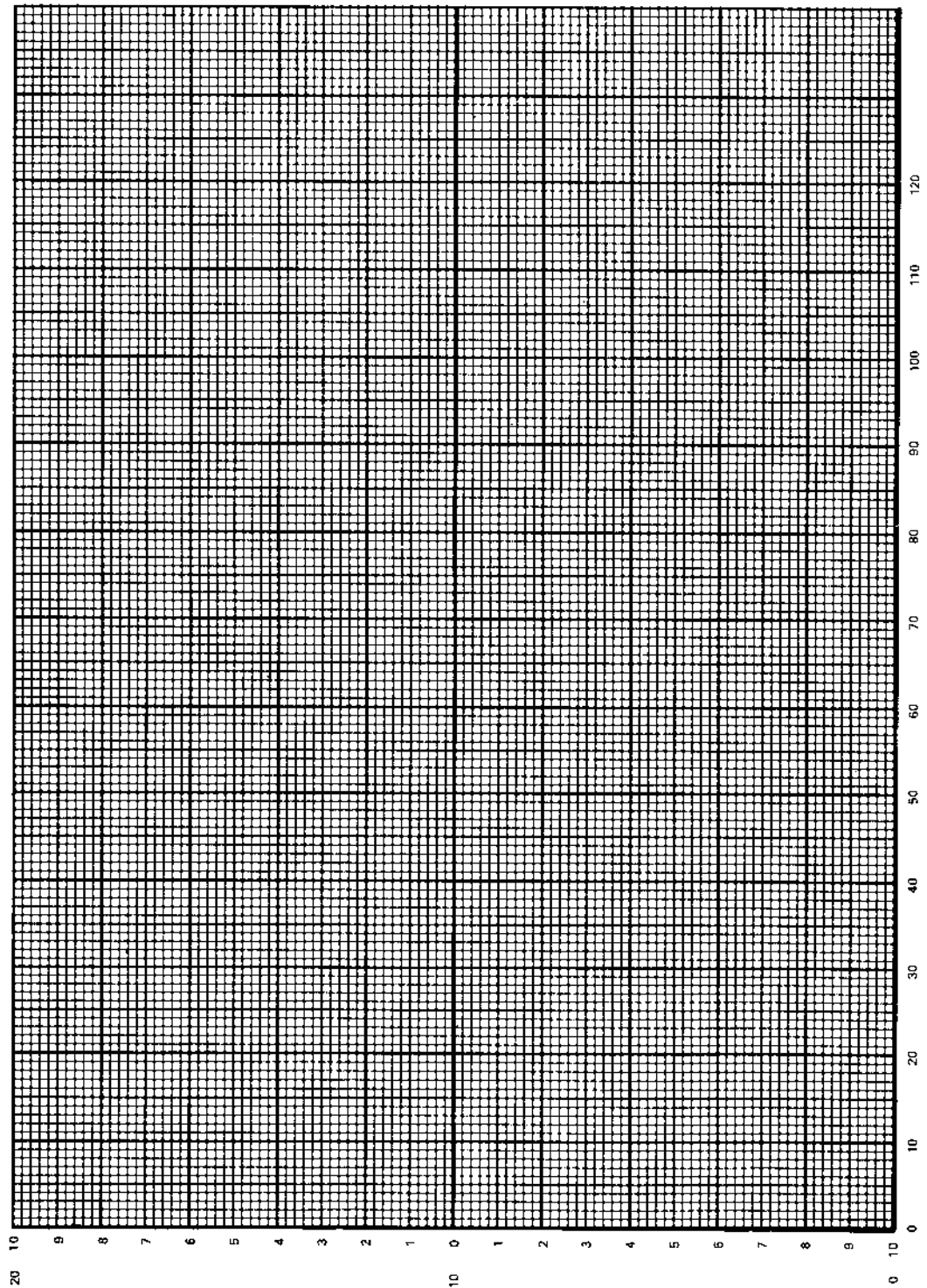


For your reference....

Text and window planner - MODE 2 (80 columns)



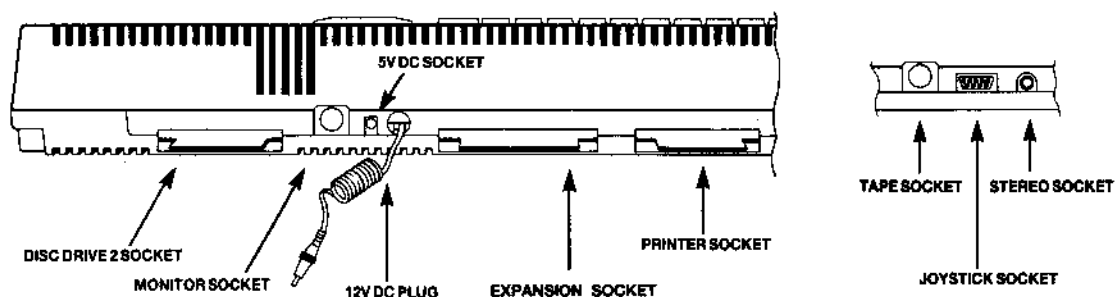
Sound envelope/music planner



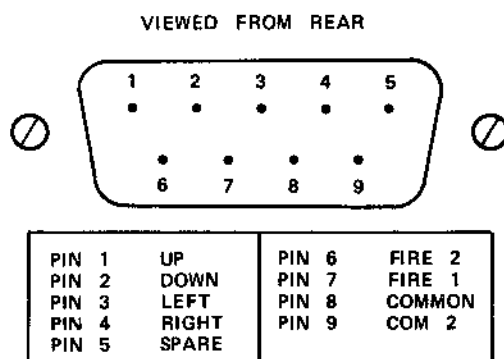
For your reference....

Part 9: Connections

CPC664 Input/Output Sockets



Joystick Socket



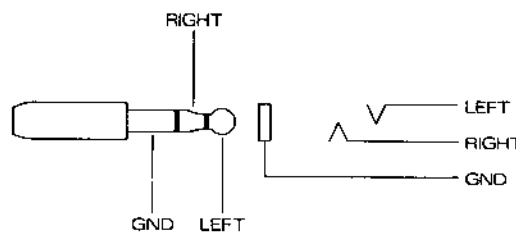
Monitor Socket

VIEWED FROM REAR



PIN 1	RED	PIN 4	SYNC
PIN 2	GREEN	PIN 5	GND
PIN 3	BLUE	PIN 6	LUM

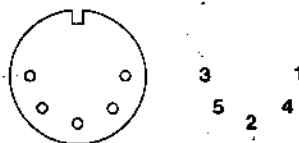
Stereo Socket



PIN 1 REMOTE SWITCH	PIN 4 DATA IN
PIN 2 GND	PIN 5 DATA OUT
PIN 3 REMOTE SWITCH	

Tape Socket

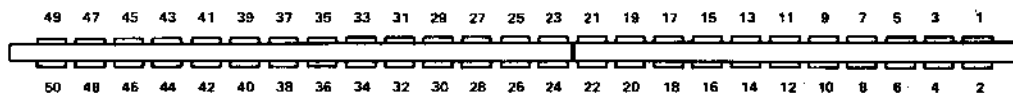
VIEWED FROM REAR



PIN 1 LEFT CHANNEL
PIN 2 RIGHT CHANNEL
PIN 3 GND

Expansion Socket

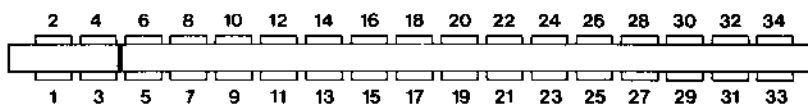
VIEWS FROM REAR



PIN 1	SOUND	PIN 18	A0	PIN 35	INT
PIN 2	GND	PIN 19	D7	PIN 36	NMI
PIN 3	A15	PIN 20	D6	PIN 37	BUSR2
PIN 4	A14	PIN 21	D5	PIN 38	BUSAK
PIN 5	A13	PIN 22	D4	PIN 39	READY
PIN 6	A12	PIN 23	D3	PIN 40	BUS RESET
PIN 7	A11	PIN 24	D2	PIN 41	RESET
PIN 8	A10	PIN 25	D1	PIN 42	ROMEN
PIN 9	A9	PIN 26	D0	PIN 43	ROMDIS
PIN 10	A8	PIN 27	+5v	PIN 44	RAMRD
PIN 11	A7	PIN 28	MREQ	PIN 45	RAMDIS
PIN 12	A6	PIN 29	M1	PIN 46	CURSOR
PIN 13	A5	PIN 30	RFSH	PIN 47	L. PEN
PIN 14	A4	PIN 31	IORQ	PIN 48	EXP
PIN 15	A3	PIN 32	RD	PIN 49	GND
PIN 16	A2	PIN 33	WR	PIN 50	φ
PIN 17	A1	PIN 34	HALT		

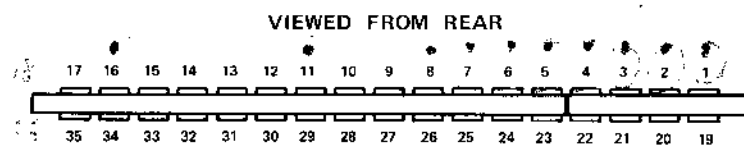
Disc Drive 2 Socket

VIEWS FROM REAR



PIN 1	READY	PIN 18	GND
PIN 2	GND	PIN 19	MOTOR ON
PIN 3	SIDE 1 SELECT	PIN 20	GND
PIN 4	GND	PIN 21	N/C
PIN 5	READ DATA	PIN 22	GND
PIN 6	GND	PIN 23	DRIVE SELECT 1
PIN 7	WRITE PROTECT	PIN 24	GND
PIN 8	GND	PIN 25	N/C
PIN 9	TRACK 0	PIN 26	GND
PIN 10	GND	PIN 27	INDEX
PIN 11	WRITE GATE	PIN 28	GND
PIN 12	GND	PIN 29	N/C
PIN 13	WRITE DATA	PIN 30	GND
PIN 14	GND	PIN 31	N/C
PIN 15	STEP	PIN 32	GND
PIN 16	GND	PIN 33	N/C
PIN 17	DIRECTION SELECT	PIN 34	GND

Printer Port



PIN 1	STROBE	PIN 19	GND
PIN 2	D0	PIN 20	GND
PIN 3	D1	PIN 21	GND
PIN 4	D2	PIN 22	GND
PIN 5	D3	PIN 23	GND
PIN 6	D4	PIN 24	GND
PIN 7	D5	PIN 25	GND
PIN 8	D6	PIN 26	GND
PIN 9	GND	PIN 27	GND
PIN 11	BUSY	PIN 28	GND
PIN 14	GND	PIN 33	GND
PIN 16	GND	All other pins	NC

D7 mbv

OUT & F600, n

Schakelen:

n = 32 8^e bit
n = 0 " " "

→ naar pin 3 van Printer

Part 10: Printers

Printer interfacing

The CPC664 allows the connection and use of an industry standard 'Centronics style interface' printer.

The printer cable is simply constructed as a one-to-one connection between the **PRINTER** socket at the rear of the computer, and the connector on the parallel printer. Note that there are two less 'fingers' on the computer printed circuit board than on the printer connector. This is to allow the use of a standard printed circuit board edge connector.

The actual pin interface details are illustrated in part 9 of this chapter.

The cable should be constructed so that pin 1 from the computer connects to pin 1 on the printer; pin 19 from the computer to pin 19 on the printer, etc, etc. However, pins 18 and 36 of the printer should NOT be connected to the computer.

Note that although there are 17 fingers on the upper row of the computer's **PRINTER** socket, the lower row of fingers is numbered 19 onwards (rather than 18 onwards). This is so that every wire used, connects between exactly the **SAME NUMBERED** finger of the computer's edge connector as pin of the socket on the printer itself.

The computer uses the **BUSY** signal (pin 11) to synchronise with the printer, and will wait if the printer is **OFF-LINE**.

There are no user set-up commands required, and the output is directed to the printer by specifying stream #8.

Although the CPC664's **PRINTER** port is envisaged for use with low cost dot-matrix printers; with a suitable interface it will support daisywheel printers, graphics plotters, and multi-colour ink-jet printers. The key to compatibility is the standard parallel interface.

The customised software in the AMSTRAD DMP1 printer facilitates dot-graphics operation, together with the printing of complete screen dumps.

Printer configuration

A facility is provided whereby special characters which may appear on the screen and which are supported by the AMSTRAD DMP1, will be printed even though the character codes for the screen and printer may be different. The majority of these symbols will only be available when the printer is switched to one of its foreign language modes. For example:

```
PRINT CHR$(&A0)
^
PRINT #8,CHR$(&A0)
^ is printed on the printer.
```

This works even though the character code for a circumflex accent on the DMP1 is &5E. In other words, the printer routine has recognised &A0 as one of the codes held in a printer translation table, and has translated it to &5E so that the same character appearing on the screen will be printed by the printer. The code &5E will produce a circumflex accent on a DMP1 no matter which language mode the printer is set to (this is not true for all the characters in the translation table). The other characters in the table are as shown in the following table:

CHR\$	Character On Screen	Printer Translation	U.K.	U.S.A.	France	Germany	Spain
&A0	^	&5E	^	^	^	^	^
&A2	..	&7B	†	†	†	†	..
&A3	£	&23	£	#	#	#	Pt
&A6	\$	&40	†	†	†	\$	†
&AE	¿	&5D	†	†	†	†	¿
&AF	¡	&5B	†	†	†	†	¡

† For the printed character, refer to page 18 of your DMP1 instruction manual.

The above is an extract from the default translations, which can be changed if required. See the Firmware manual (SOFT 946) for further details.

Part 11: Joysticks

The built-in software in the computer supports either one or two joysticks. These are treated as part of the keyboard, and as such, may be interrogated by INKEY and INKEY\$.

Note that in the majority of cases, the main 'fire' button on a joystick is interpreted as 'Fire 2' by the CPC664.

The functions JOY(0) and JOY(1) enable direct inspection of the first and second joysticks respectively. The function returns a bit-significant result which indicates the state of the joystick switches at the last keyboard scan.

The table below indicates the values returned by both joysticks. The JOY values are followed by values for use in statements which take key numbers as parameters (i.e. INKEY and KEY DEF).

STATUS	JOY COMMAND		KEY VALUES		
	BIT SET	VALUE RETURNED	FIRST JOYSTICK	SECOND JOYSTICK	EQUIVALENT KEY
Up	0	1	72	48	'6'
Down	1	2	73	49	'5'
Left	2	4	74	50	'R'
Right	3	8	75	51	'T'
Fire 2	4	16	76	52	'G'
Fire 1	5	32	77	53	'F'

Note that when key values for the SECOND joystick are returned, the computer cannot tell whether those values have been generated by the joystick or by the equivalent keyboard key (indicated in the last column of the previous table). This means that the keyboard can be used as a substitute for the second joystick.

Part 12: Disc organisation

The BIOS supports three different disc formats: SYSTEM format, DATA ONLY format and IBM format. Under AMSDOS, the format of a disc is automatically detected each time a disc with no open files is accessed. To permit this automatic detection, each format has unique sector numbers.

3 inch discs are double sided, but only one side may be accessed at a time depending on which way round the user inserts the disc. There may be a different format on each side.

Common to all formats:

Single sided (the two sides of a 3 inch disc are treated separately).
512 byte physical sector size.
40 tracks numbered 0 to 39.
1024 byte CP/M block size.
64 directory entries.

SYSTEM format

9 sectors per track numbered &41 to &49.
2 reserved tracks.

The system format is the main format supported, since CP/M can only be loaded (cold and warm boot) from a system format disc. The reserved tracks are used as follows:

Track 0 sector &41	: boot sector.
Track 0 sector &42	: configuration sector.
Track 0 sectors &43 to &47	: unused.
Track 0 sectors &48 to &49	: } CCP and BDOS.
Track 1 sectors &41 to &49	: }

Note that VENDOR format is a special version of system format which does not contain any software on the two reserved tracks. It is intended for use in software distribution.

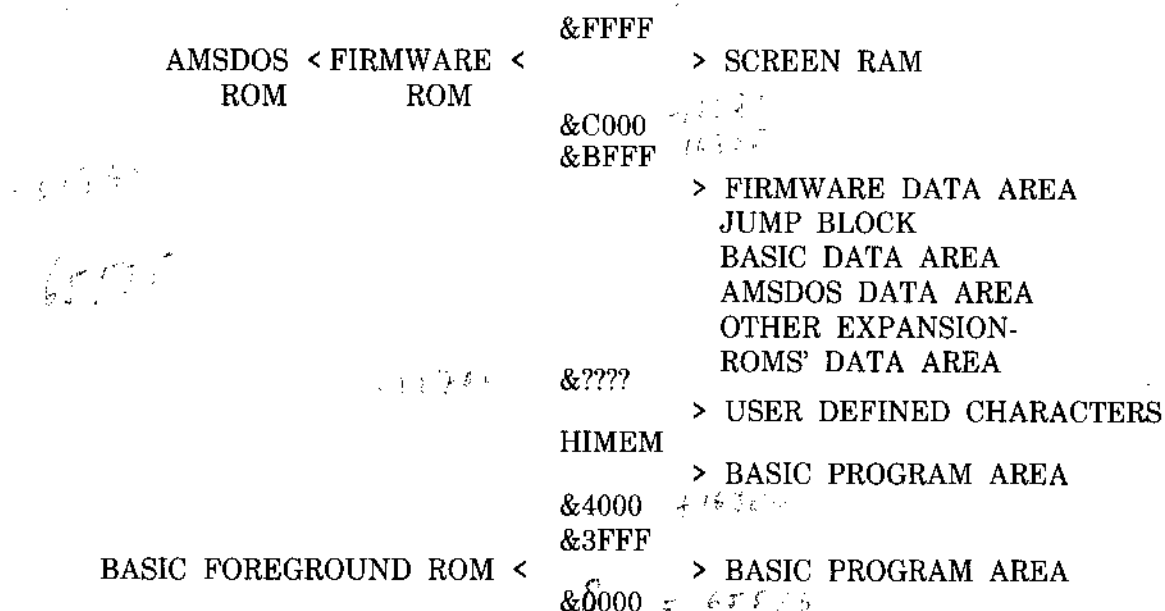
9 sectors per track numbered &C1 to &C9.
0 reserved tracks.

This format is intended for future enhancement. It is not recommended for use with CP/M since it is not possible to 'warm boot' from it. However, if only AMSDOS is to be used, the DATA ONLY format affords a little more disc space.

8 sectors per track numbered &1 to &8.
1 reserved track.

This format is logically the same as the single-sided format used by CP/M on the IBM PC. It is intended for specialist use and is not otherwise recommended.

Memory Map



...where address &???? is dependent upon expansion ROMs. (Note that &???? is &A6FC when no external expansion ROMs are fitted.)

Additional I/O

Most I/O port addresses are reserved by the computer, in particular addresses below &7FFF should not be used at all.

It is intended that the part of the address A0 - A7 should reflect the type of external I/O device, and that address lines A8 and A9 may be decoded to select registers within the I/O device. Of the remaining address lines, only A10 must be decoded (as low) whilst lines A11 to A15 are high. Thus each device may have registers addressed as &F8??, &F9??, &FA??, and &FB??, where ?? is in the range DC to DF for communications interfaces, and E0 to FE for other user peripherals.

Note that Z80 instructions which place the B register on the upper half of the address bus (A15 - A8) must be used.

Sideways ROMs

Provision is made for additional ROMs to be selected in place of any part of the on-board ROM. The address arbitration and bank selection logic will be contained in a module connected to the expansion bus, but all the signals required are brought to the expansion bus.

Chapter 8

At your leisure....

This chapter takes a leisurely look at some background information to computing in general, and at the CPC664 in particular. It is not vital that you read this chapter before operating the computer, but it may help you to understand a little of what's going on 'under the bonnet'.

Part 1: Generally speaking....

Zap the wotsit!

Even if the only reason that you bought your CPC664 was to take advantage of the sophisticated computer games available, you may still probably be wondering about several aspects of the computer that come under the heading of 'hardware'.

The hardware is what you can pick up and carry around, i.e. the main computer keyboard, the monitor, the connecting leads etc. In fact, it's just about everything that isn't specifically the 'software' - programs, manuals, and disc or cassette based information.

Certain features of the way that the computer behaves, are produced by courtesy of the hardware - things like the coloured display on the TV set (or monitor). It's up to the software to make use of these hardware capabilities to produce specifically designed characters and shapes on the screen.

The hardware actually directs the beam of electrons at the electro luminescent surface on the inside of the screen of the TV tube to make it 'light up' - the software adds order and intelligence by telling the hardware when and how to perform. It adds timing, control and sequencing to produce the effect of a spaceship taking off, or something more mundane like a letter appearing when you type at the keyboard.

So what makes one computer better than another?

Hardware without software is worthless. Software without hardware is equally worthless - the value of the computer begins when the two come together to perform various tasks. There are some very basic considerations that can be used to grade performance of both hardware and software.

The generally accepted reference points for personal computers are now:

1. The screen resolution - the smallest discernible item on the display.

This is a combination of factors, including the number of different colours available to the programmer, the number of distinctly different areas that can be resolved on the display (i.e. the pixels), and the number of text characters that can be displayed on a single screen area.

You will find that your CPC664 compares very favourably with any similarly priced machine in each of these respects.

2. The BASIC interpreter

Virtually every home computer includes with it a BASIC interpreter that allows the user to start creating programs to use the hardware features. The built in programming language (BASIC) that comes supplied with your machine is itself a program - an immensely complicated and intricate program that has been evolved over a million man-years of experience since BASIC was 'invented' in the USA. The 'Beginners' All-purpose Symbolic Instruction Code' is easily the most widely used computer language in the world, and like any language, it comes in a variety of local 'dialects'.

The version in the CPC664 is one of the most widely compatible dialects of BASIC, and will also run programs written for operation under the CP/M disc operating system. It is a very fast implementation of BASIC - in other words it performs its calculations quickly - and whilst you may not be too concerned that one computer may take 0.05 of a second to multiply 3 by 5 and display the answer, whereas another may take 0.075 second to do the same - where a program that draws graphics patterns on the screen may call for many thousands of simple repetitive calculations, the difference between 0.05 and 0.075 of a second adds up to a considerable difference in performance.

You will frequently hear the term 'machine code' being used. Machine code is the raw form of instruction code that can be passed to the processor. It takes less time to work out what it's been asked to, and gets on with producing the result some 5 to 15 times faster than an equivalent operation being passed along through the BASIC interpreter. On the other hand, it can take 5 to 50 times longer to write an equivalent program in machine code when compared to performing the same overall task using BASIC.

The BASIC in your Amstrad computer is amongst the fastest and most fully featured to be found in any home computer system, and incorporates many features that help the experienced BASIC programmer overcome some of the inherent sluggishness of a 'high level language' interpreter to perform surprisingly dynamic visual and musical effects.

3. Expansibility

Most computers pay attention to the need to 'add-on' additional items of hardware: printers, joysticks, extra disc drives. Paradoxically, some of the most successful home computers require the addition of add-on units known as 'expansion interfaces' before even a simple printer or joystick controller can be installed.

The purchaser does not always think ahead to his needs in the future, because a machine that incorporates a properly supported parallel printer (Centronics compatible) and a games joystick port may actually be cheaper in real terms.

The CPC664 computer features a built-in Centronics printer port, a port for an additional disc drive, a cassette data (+ motor control) socket, facilities for up to two joysticks, a stereo sound output - and a comprehensive expansion bus that can be used to attach serial (RS232) interfaces, MODEMs, speech synthesisers, light pens, etc.

4. Sound

The sound features of a computer determine whether or not it sounds like a bluebottle in a empty cocoa tin - or if it can produce an acceptable representation of an electronic musical instrument.

The CPC664 computer uses a 3 channel 8 octave sound generator, which can produce a very acceptable musical quality with full control of the amplitude and tone envelopes. Furthermore, the sound is divided into a stereo configuration, where one channel provides the left output, one channel provides the right output, and the third channel sits in the middle.

This provides considerable scope for writing programs that track the sound effects across the screen to follow the motion of an arcade-style game.

Ultimately, you will make up your own mind about which of these features is most important to you. We hope that you will try them all to make the most of your computer.

Why can't?

With all the power of modern technology, users frequently wonder why even a machine as advanced as the CPC664 is apparently unable to perform tasks seen on any TV set. Why for instance, can't a computer animate a picture of someone walking across the screen in a natural fashion? - why do all computers represent movement with 'matchstick' figures?

The answer is simple yet complex. The simple answer is that you must not be beguiled into believing that the screen of your computer has anything of the subtlety of the screen of a TV set. A television set operates using 'linear' information that can describe a virtually infinite range of resolution between the extremes of light and dark across all the colours of the spectrum. This process means that in computer terms, the display 'memory' of a full TV picture is some twenty times greater than the converted equivalent of a home computer display.

That's only part of the problem, since to animate this picture requires that this enormous amount of memory must be processed at high speed (around 50 times each second). It can be done - but only by machines that cost a few thousand times more than a home computer, at least for the time being!

Until the price of high speed memory falls dramatically (it will eventually), small computers have to make do with a relatively small amount of memory available to control the screen display, which results in lower resolution, and jerkier movements. Thoughtful hardware design and good programming can go a long way to making the best of this situation, but we are still a fair way from cheap computers that can reproduce flowing motions and lifelike pictures in the same way that even a moderate animated cartoon can produce.

That keyboard looks familiar....

Why can't you simply walk up to the computer type a page of simple text into the machine?

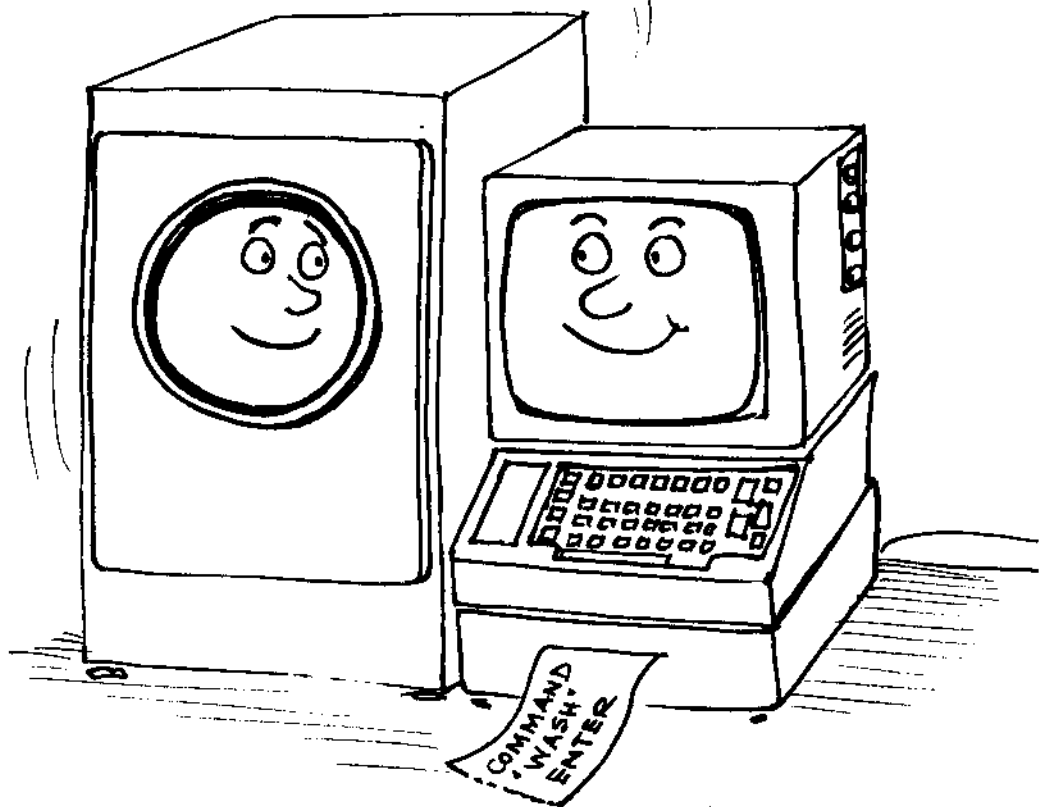
Don't be misled by the fact that the computer looks like a typewriter with an electronic display. The screen is not a piece of electronic paper - it's a 'command console' - jargon which means that it simply provides you with the means of communicating with the programming language (and the programs) in the machine memory.

Until you tell it to the contrary, the computer will try and interpret all the characters that you type at the keyboard as being program instructions. When you press the **[ENTER]** key, the computer will look through what has been typed, and if it doesn't make sense to the built in BASIC, it will reject the 'input' with the comment:

```
Syntax error
```

However, it may just happen that the program presently residing in your computer is a Word Processor system, in which case you will be able to type random words, press **[ENTER]** and carry on typing as if the system WERE operating as an electronic piece of paper in an electronic typewriter. But to do this, you must have first loaded a word processor program into the machine's memory.

The computer 'seems' to combine several items of equipment that have become familiar around the home and office such as the TV-like screen and the keyboard. You must remember that the similarities are generally strictly superficial, and that the computer is a combination of familiar looking hardware that has an entirely different personality of its own!



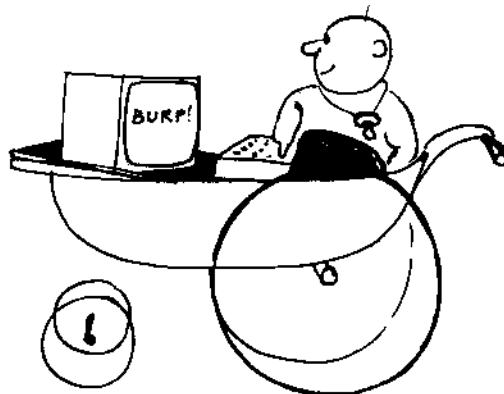
Whose afraid of the jargon?

As with all 'specialist' industries, computing has developed its own jargon as a short-hand form of communicating complicated concepts that require many words of 'plain language' explanation. It's not just the high technology business that's guilty of hiding itself behind an apparent smokescreen of 'buzz words', jargon and terminology - most of us have come up against the barriers to understanding erected by all the main professions and trades.

A major difference is that the confusion in legal jargon arises from the way the words are used - rather than the words themselves as in the case of computing. Most people who grow familiar with computing terminology will go out of their way to use the words in the most straightforward possible manner, so as to minimise the complexity of the communication. Don't be misled by the 'plain language' used in computing, it is not a literary subject, but a precise science, and apart from the 'syntax' of the wording, the structure of the communication is very straightforward, and not in the least confusing or ambiguous. Teachers of computing have not yet managed to make an art form out of trying to analyse the exact meaning intended by a programmer in his program construction.

Having said that, despite whether or not the meaning of a computer program is obvious, there are still many aspects that can be analysed as either elegant or untidy, and more emphasis is being put on a formal approach to program construction, now that the initial mayhem brought about the micro revolution is settling down.

Computing is rapidly being understood by many young people who appreciate the precision and simplicity of the ideas and the way they can be communicated - you don't find too many 10 year old lawyers - but you can find plenty of ten year old programmers!



Basics of BASIC

Virtually all home computers provide a language known as BASIC, which allows programs to be written in the nearest thing to plain language presently available. BASIC no longer has any particular significance as to the degree of the sophistication of the languages, and many extremely complex and powerful programs are written using BASIC.

However, there's no doubt that the name has attracted many newcomers for its promise of providing a starting place in the maze of computer program languages, and this has contributed significantly to its universality.

BASIC is a computer language that interprets a range of permitted commands, and then performs operations on data while the program runs. Unlike the average human vocabulary of 5000-8000 words (plus all the different ways verbs can be used etc,) BASIC has to get by with about two hundred. Computer programs written using BASIC have to follow rigid rules concerning the use of these words. The syntax is precise, and any attempt to communicate with the computer using literal or colloquial expressions (i.e. plain language) will result in the cold and clinical message:

Syntax error

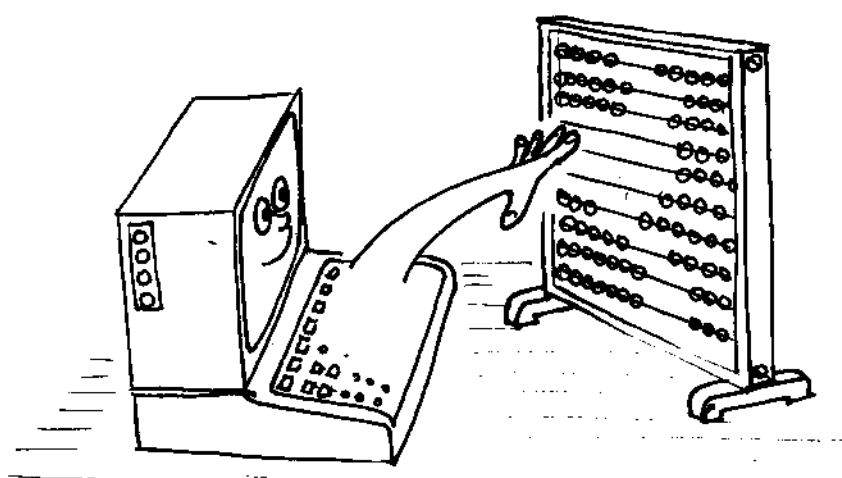
This is not as restrictive as it first appears, since the language of BASIC (the syntax) is primarily designed to manipulate numbers - the numeric data. The words are essentially an extension of the familiar mathematical operators $+/=$ etc., and the most important concept for newcomers to grasp is the fact that a computer can only work with numeric data. Information that is supplied to the CPU (Central Processor Unit) integrated circuit is only supplied in the form of numerical data.

Number please....

If a computer is used to store the complete works of Shakespeare, there will not be a single letter or word to be found anywhere in the system. Every piece of information is first converted into a number which the computer can locate and manipulate as required.

BASIC interprets the words as numbers which the computer can then manipulate using only addition, subtraction and features from Boolean logic that permits the computer to compare data and select for certain attributes - in other words, check to see if one number is greater than or the same as another, or to perform a defined task if one number or another meets certain criteria.

Through the medium of the program, the computer breaks down every task into a simple series of Yes/No operations.



If this process sounds cumbersome, then you're quite right, as you have uncovered the first and most important truth about computing. A computer is primarily a tool for performing the simplest of repetitive tasks very quickly and with absolute precision. Thus BASIC interprets the instructions as given in the form of the program, and translates them into the language that can be handled by the CPU. Only two states are understood by the logic of a computer - 'yes' or 'no', represented in binary notation as '1' and '0'. The representation in Boolean logic is simply 'true' and 'false' - there's no such thing as a 'maybe' or 'perhaps'!

The process of switching between these two distinct states is the essence of the term 'digital', and is sometimes referred to as 'toggling'. In the world of nature, most processes move gradually from one completely 'stable' state to another in a linear progression. In other words, the transition is made by following the path of a line between the two states - in an ideal digital environment the switch from one state to the next is made in no time at all -but the physics of semiconductor science dictate that there will be some minor delay, referred to as propagation delay - and it is the accumulation of many of these propagation delays that provides the reason why a computer has to spend some time processing the information before the answer comes out.

In any case, the computer would have to wait a finite time for one task to have finished before it can start work on the result of that first task - so there would need to be some artificial delay imposed anyway. The digital process is black or white, and the stages of transition via the various shades of grey have NO significance whatsoever. Conversely, the linear or 'analogue' progression IS via the shades of grey.

If the ultimate answer is either 0 or 1, then there is no possibility of it being 'nearly' correct. The fact that computers can sometimes appear to make errors when handling numeric data is due to the limitation of the size of numbers it can process, requiring 'oversized' data to be squeezed down to fit the space available, or 'truncated', leading to rounding errors. e.g. 999,999,999 becomes 1,000,000,000.

In a world where the only two numbers available are 0 or 1, how then do you count beyond 1?

Bits and Bytes

We just happen to be used to understanding numbers based on the decimal system where the reference point is the number 10 - i.e. there are ten digits available to represent quantities in range from 0 to 9 (which is used in preference to the expression 1 to 10). The system where numbers range from 0 to 1 is the binary system, and the units in which the system operates are called bits - an abbreviated form of 'Binary digiT'.

The relationship between bits and decimal notation is simple to understand:

It's conventional to declare the maximum number of binary digits being used by adding leading zeros to make up the number to the full number of bits:

e.g. decimal 7 becomes:

00111 binary

....using 5 bit notation.

In the binary system, the figures may be considered merely as indicators in columns to specify whether or not a given power of 2 is present; 1=yes, 0=no.

$$2^0 = 1$$

$$2^1 = 2 = 2(2^0)$$

$$2^2 = 4 = 2 \times 2 = 2(2^1)$$

$$2^3 = 8 = 2 \times 2 \times 2 = 2(2^2)$$

$$2^4 = 16 = 2 \times 2 \times 2 \times 2 = 2(2^3)$$

....so the columns look like:

$$\begin{array}{ccccc} 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \\ 1 & 0 & 0 & 1 & 1 \\ (16 + 0 + 0 + 2 + 1) = 19 \end{array}$$

In order to provide a shorthand method of referring to binary digit information, the term 'byte' is used to denote 8 bits of information. The maximum number that be stored in a byte is then (binary) 11111111 - or (decimal) 255. This implies 256 actual variations, including 00000000, which is still perfectly valid data to a computer.

Computers tend to manipulate data in 8 bit multiples. 256 is not a very large number, so in order to achieve an acceptable means of handling the memory, two bytes are used to provide a method of addressing memory which is in the form of array, with a horizontal and vertical address by which the elements of that array can be located:

	0	1	2	3	4	5	6	7	8	9
0										
1										
2										
3										
4										
5				1		1				
6										
7										
8										
9										

The array can locate up to (10x10) items of information using address numbers that lie in the range 0 to 9. The item stored at position 3,5 is a '1' - as is the item at 5,5.

So a binary array of 256x256 can handle 65,536 individual locations using 8 bit addresses for the vertical and horizontal axes of the array. So our '0' and '1' have progressed to being capable of identifying one of 65,536 different elements.

The next level of shorthand for binary is the kilobyte (kByte or simply 'K') which is 1024 bytes. 1024 is the nearest binary multiple to the more familiar decimal use of the term 'kilo' (1000) - and explains why a computer described as having a '64K' memory does in fact have a memory of 65,536 bytes (64 x 1024).

Thankfully, the BASIC interpreter does all the necessary conversions for you, and it is quite possible to become a proficient programmer without a complete understanding of binary, although an appreciation of the significance of binary will help you spot the many 'magic' or significant numbers that inevitably crop up as you work through the science of computing.

It's worth spending some effort to acquire an understanding of binary and the various significant numbers 255, 1024 etc, since it is very unlikely that these will change from being the bedrock of computer operation in the foreseeable future. The certainty and simplicity that comes from working in only two states will prevail over the enormously increased complexity that would result from any other number base.

However....

Simple and elegant as it is, binary notation is longwinded and prone to inaccuracy as it cannot be easily read at a glance. Binary has a number of associated counting systems that act as shorthand for programmers. One such number system widely used in microcomputing is called HEX (an abbreviation of hexadecimal).

Here the number is based on 16 (0 to 15), and is represented in a single character:

Decimal

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

Hex

0 1 2 3 4 5 6 7 8 9 A B C D E F

The hexadecimal system can break the eight bits of a byte into two blocks of four bits, since 15 is a four bit number: 1111 binary. The first block indicates the number of complete units of '15', and the second indicates the 'remainder' - and this is where the absolute elegance of binary and hex begin to emerge.

Reconsidering the table that introduced binary notation:

Decimal	Binary	Hexadecimal
0	0	0
1	1	1
2	10	2
3	11	3
4	100	4
5	101	5
6	110	6
7	111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F
16	10000	10

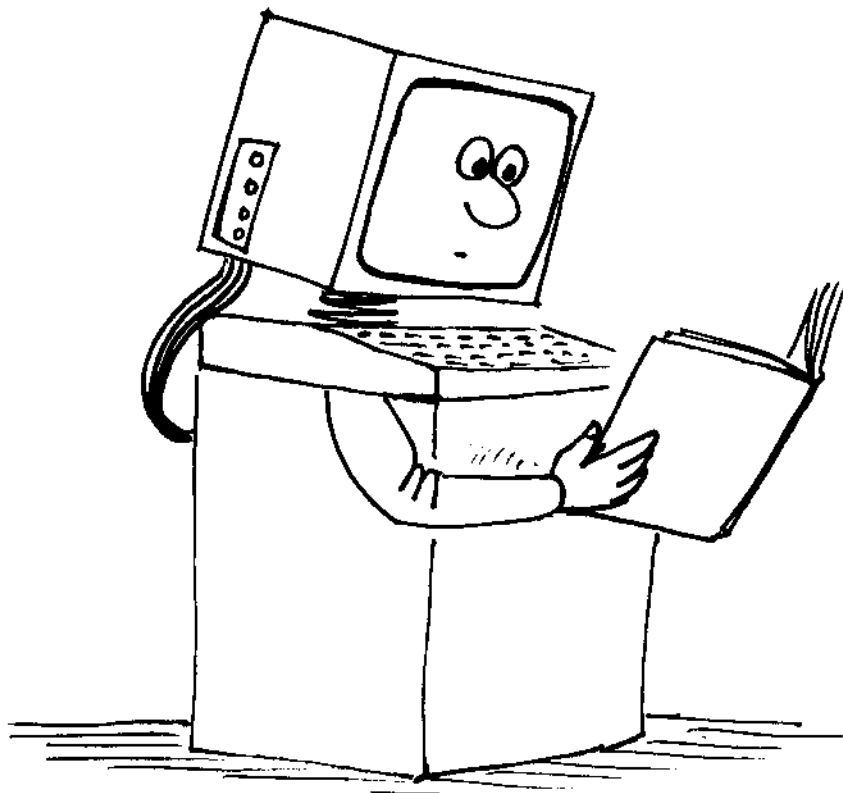
An 8-bit number 11010110 (&D6 hex) can be subdivided, and then considered as two 4-bit numbers (known as nibbles). Throughout this manual, a hex based number is introduced by the '&' symbol e.g. &D6, and this is the number base most commonly used by programmers using assembly language techniques. An assembly language program is the nearest most programmers get to programming directly in machine code, since the assembly language program allows the program to use simple letter 'mnemonics' to specify the actual machine code 'numbers'.

When using hex, you must first work out the value of the first digit to obtain the number of 16's in the final number, and then add the remainder designated by the second 'half' of the hex notation to obtain the total decimal equivalent. There's a powerful temptation to regard a number like &D6 as 13+6, or 136, but it's $(13 \times 16) + (6) = 214$.

It's the same process you use when you read a decimal number (also known as a Denary number) such as '89' - i.e. $(8 \times 10) + (9)$. It just happens that multiplying by ten is a great deal simpler, unless you've had a lot of practice at multiplying by 16.

If you've got this far without becoming too confused, then you are well on your way to getting a grasp of the basic principles of the computer. You may even be wondering what all the fuss is about - and you'd be quite correct. A computer is a device that manages very simple concepts and ideas; it just happens to perform these tasks at great speed (millions of times per second), and with a huge capacity to remember both the data that has been input, and the intermediate results of the many thousands of very simple sums along the way to the result.

If you want to pursue the theory of your computer, there are literally thousands of books available on the subject of computing. Some will tend to leave you more confused than you were when you started reading them, but a few will actually lead you along the way by revealing the simplicity and the fundamental relationships that exist between the number systems, and the way that your computer deals with them.



Part 2: More about the CPC664 specifically....

This section gently expands upon some of the machine-specific aspects of the CPC664. Background information to these matters will be found both in the Foundation course, and in the chapter entitled 'Complete list of AMSTRAD CPC664 BASIC keywords'.

Subjects covered in this section:

- ★ Character set
- ★ ASCII
- ★ Variables
- ★ Logic
- ★ User defined characters
- ★ Print formatting
- ★ Windows
- ★ Interrupts
- ★ Data
- ★ Sound
- ★ Graphics
- ★ Machine hardware

A bit of character....

As you type at your CPC664 keyboard, you should not take for granted the fact that recognisable letters and numbers etc, appear on the screen. After all, we've already discussed the fact that your computer is not a typewriter. What's actually happening is a result of you pressing a combination of electrical switches. The electrical signals produced when you press these switches are translated by the circuitry inside the equipment to produce a pattern of dots on the screen. We recognise that pattern of dots as a letter, number, or other character from the CPC664's 'character set'.

Some of the characters that you will see are not directly accessible by pressing the keys on the keyboard, but are only available for display using the `PRINT CHR$(number)` statement. This is because each element stored in the computer is stored in the unit of data known as the 'byte' - and as just discussed in part 1 of this chapter, a byte has 256 different possible combinations of value. As the computer has to use at least one whole byte per character stored (whether we want it to or not - it's the smallest denomination that the CPC664 appreciates), we might as well use all 256 possible combinations, rather than simply be satisfied with the 96 or so 'standard characters' that are printed on most typewriters - and throw away the spare 160 possibilities.

The 'standard' range of characters is known as a 'subset'. It is classified throughout the computer world as the 'ASCII' display system, a term derived from 'American Standard Code for Information Interchange'. It's primarily a system that ensures the data sent from one computer to another is in a recognisable form. The chapter entitled 'For your reference....' lists the ASCII display range, together with the additional characters available on the CPC664, and the corresponding numeric codes.

How we get there....

You are by now probably quite familiar with the program:

```
10 FOR n=32 TO 255
20 PRINT CHR$(n);
30 NEXT
```

....which makes the computer display the character set on the screen. Let's now examine the essence of this small program:

The first point to notice is that the computer has not been instructed to `PRINT "a b c d e f g h i j k l m n..... etc"`; instead it has been asked to `PRINT CHR$(n)`. `n` just happens to be a convenient shorthand note for a 'variable'. A variable is an item of computer information that 'varies' according to the instructions given in the program. (The choice of the letter `n` for the variable is arbitrary - it can be any letter(s) as long as it's not a keyword.)

How can you tell what is a variable?....

A number like 5 is fixed, it occurs between the numbers 4 and 6 - thus it is not a variable. The character `n` is also fixed - it's a letter from the alphabet.

So how did the computer know the difference? If the letter `n` had been declared to be the alphabetical character, we would have typed `n` in quotation marks, i.e. `"n"`, and the computer would have responded with the message `Syntax error` - because it does not understand the command sequence `FOR "n"=32 TO 255`.

Simply by using `n` without quotation marks, we have told the computer that `n` is a variable. The definition of the `FOR` statement in BASIC requires that it should be followed by a variable - so the computer assumes that whatever follows `FOR` is just that.

We have also told the computer that `n=32 TO 255`. Thus we have declared the range of the variable. It is in effect, a sequence starting at 32, finishing at 255.

Having declared this variable, we should then instruct the computer what it should do with it - line 20 does just that:

```
20 PRINT CHR$(n);
```

This specifies that whatever the current numeric value of `n`, the computer should look into its memory to see which character number corresponds to that value, and print the character on the screen.

The semicolon at the end of the line instructs the computer to prevent a carriage return and line feed. (Otherwise each new character will be printed in the first column of a new line.)

Line 30 tells the computer that after it has performed the task with the first value of `n` in the sequence (which is 32), it should return to the line where the `FOR` is located, and do the same again with the `NEXT` value that it assigns to the variable `n`. This process is known as 'looping', and is one of the most vital and fundamental aspects of computer programming and operation. It saves typing long repetitious sequences manually, and you will quickly come to use it in your own programming.

When this `FOR NEXT` loop reaches the limit of its declared range (255), the operation ceases and the computer then looks for the next line after line 30 - but there isn't one, so it simply ends, and returns to direct mode, displaying the `Ready` prompt. This tells you that the computer is ready to accept further instructions - or you can enter `RUN` again and repeat the execution of the program. The program is safely stored away in the memory and will remain there until you tell the computer otherwise - or turn the power off.

This program neatly illustrates a fundamental point about computing - i.e. everything the computer does is related to numbers. The computer has displayed the alphabet - and a whole range of other characters - using a number as its reference to the character required. When you type the key marked **A**, you don't ask the computer to type an **A** on the screen, but you tell the computer to look into the part of its memory that contains the numeric information to display the letter **A** on the screen. The actual location of this data is defined by the numeric code that is activated by the action of typing at the keyboard.

(Each character has a corresponding number, and these are listed in part 3 of the chapter entitled 'For your reference....'.)

Similarly the displayed character has nothing to do with 'writing' the letter on the screen; once again it's all about numbers.

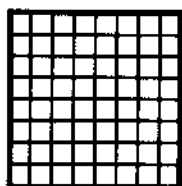
For example, the ASCII code for the letter A is 97. The computer doesn't understand 97 either (awkward blighter, eh?), and this number has to be translated from the human decimal code into a code that computer can relate to - it's generally referred to as 'machine code', and the principles underlying this aspect of the machine are discussed earlier in this chapter.

At first, the translation from the decimal number notation we are used to in everyday life, to the 'hexadecimal' notation of the computer will seem heavy going. Thinking of numbers that are based on the ten unit is so natural, that to do otherwise is like trying to eat with your knife and fork in the opposite hands.

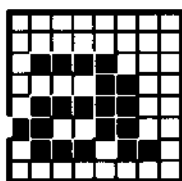
A degree of mental dexterity must be acquired to understand hex notation, but once you do, many things about computing will fall into place and the elegant structure of the numbering system will become apparent.

If you are unsure about the binary and hexadecimal numbering systems, we suggest that you thoroughly read part 1 of this chapter (if you have not already done so).

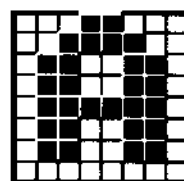
Once the computer has translated the pressing of the **A** key into the type of number it understands, it looks into that part of the memory indicated, and the result is another series of numbers that define the character. Hence the character that you see displayed on your screen, is built up from a block of data, stored in the memory as a numeric 'matrix':



A BLANK CHARACTER
MATRIX (GRID)



LOWER CASE a



UPPER CASE A

The elements of the matrix are rows and columns of dots. The character is displayed by turning the required sequence of dots on or off - each dot is determined by data stored in the computer's memory. There are 8 rows and 8 columns in each character matrix or 'cell' on the CPC664 display, and if you don't find a character you want out of the set of 255 that are provided, then you can re-define your own characters using the keyword `SYMBOL` described later in this section.

These 'user defined characters' can be made up using any combination of 0 to 64 dots, arranged in any order - so the 'complete' character set that uses all possible combinations of this matrix would comprise many more different characters. Add to this the fact that you can group blocks of characters together to form larger block characters, and the possibilities for user-defined graphics are limited only by your time and ingenuity.

Logic....

A major difference between a calculator and computer is the computer's ability to handle logical operations in applications like the conditional IF THEN sequence. To do this, the logical operators treat the values to which they are applied as bit patterns (bit-wise), and operate on the individual bits. The description and use is entirely, well ...er logical - but it is notoriously difficult to describe logic in simple terms without the precision of concise definitions.

The two halves of the logical expression are known as the arguments. A logical expression comprises:

⟨argument⟩[⟨logical operator⟩⟨argument⟩]

where:

⟨argument⟩ is: NOT ⟨argument⟩
 or: ⟨numeric expression⟩
 or: ⟨relational expression⟩
 or: (⟨logical expression⟩)

Both the arguments for a logical operator are forced to integer representation, and ERROR 6 results if an argument will not fit into the integer range.

The logical operators, in order of precedence, and their effect on each bit are:

AND Result is 0 unless both argument bits are 1
OR Result is 1 unless both argument bits are 0
XOR Result is 1 unless both argument bits are the same

AND is the most commonly employed logical operator, and does NOT mean 'add'.

PRINT 10 AND 10

Results in 10

PRINT 10 AND 12

Results in 8.

```
PRINT 10 AND 1000
```

Results in 8 again.

This is because the numbers 10 and 1000 have been converted to their binary representations:

```
    1010
1111101000
```

The AND operation checks each corresponding bit at a time, and where the bit in the top AND the bottom row is the 1, the answer is 1:

```
0000001000
```

...which is our result of 8. The logical operator AND is used to detect when two conditions are present simultaneously. Here's a self explanatory application:

```
10 INPUT "The number of the day";day
20 INPUT "The number of the month";month
30 IF day=25 AND month=12 THEN 50
40 CLS:GOTO 10
50 PRINT "Merry Christmas!"
```

OR works on bits as well, where the result is 1 unless both bits from the arguments are 0, in which case the result is 0. Using the same numbers as for the AND example:

```
PRINT 1000 OR 10
1002
```

Bit-wise:

```
    1010
1111101000
```

Resulting in the answer:

```
1111101010
```

And in a program example:

```
10 CLS
20 INPUT "The number of the month";month
30 IF month=12 OR month=1 OR month=2 THEN 50
40 GOTO 10
50 PRINT "It must be winter!"
```

The NOT operator inverts each bit in the argument (0 becomes 1, and vice versa):

```
10 CLS
20 INPUT "The number of the month";month
30 IF NOT(month=6 OR month=7 OR month=8) THEN 50
40 GOTO 10
50 PRINT "It can't be summer!"
```

Another major feature to consider is the fact that you can add together any number of logical conditions (up to the maximum line length) to distill the facts yet further.

```
10 INPUT "The number of the day";day
20 INPUT "The number of the month";month
30 IF NOT(month=12 OR month=1) AND day=29 THEN 50
40 CLS:GOTO 10
50 PRINT "This is neither December nor January, but
    this might be a leap year"
```

The result of a relational expression is either -1 or 0. The bit representation for -1 is all bits of the integer = 1; for 0 all bits of the integer = 0. The result of a logical operation on two such arguments will yield either -1 for True, or 0 for False.

Check this by adding lines 60 and to the above program:

```
60 PRINT NOT(month=12 OR month=1)
70 PRINT (month=12 OR month=1)
```

....and when the program is run, entering 29 for the day and, say, 2 for the month will produce the answer in line 50, and the actual values returned by the logical expressions in lines 60 and 70.

Finally, XOR (eXclusive OR) produces a true result as long as both arguments are different.

The following summarises all these features in what's known as a 'truth table'. It's a convenient way of illustrating what happens in a bit-wise logical operation.

Argument A	1010
Argument B	0110
AND result	0010
OR result	1110
XOR result	1100

User defined characters

One of the first applications of binary numbers that you are likely to come across will be when designing characters for use with the **SYMBOL** command. If the character is drawn on an 8 by 8 grid then each of the eight rows can be converted to a binary number by putting a 1 for each pixel that is to be inked and a zero for each that should be invisible, i.e. set to the paper colour. These eight numbers are then passed to the **SYMBOL** command. For example, to define a house character:

*	=	00001000	=	&08	=	8	=	8	
****	=	00111100	=	&3C	=	32+16+8+4	=	60	
* *	=	01000010	=	&42	=	64	2 * 2	=	66
* * * *	=	10100101	=	&A5	=	128 + 32 + 4 + 1	=	165	
* * * *	=	10000001	=	&81	=	128 + 1	=	129	
* * * *	=	10110101	=	&B5	=	128 + 32 + 16 + 4 + 1	=	181	
* * *	=	10110001	=	&B1	=	128 + 32 + 16 + 1	=	177	
*****	=	11111111	=	&FF	=	128+64+32+16+8+4+2+1	=	255	

....the command is:

```
SYMBOL 240,8,60,66,165,129,181,177,255
```

....or....

```
SYMBOL 240,&08,&3C,&42,&A5,&81,&B5,&B1,&FF
```

....or....

```
SYMBOL 240,&X00001000, &X00111100, &X01000010, &X10100101,
&X10000001, &X10110101, &X10110001, &X11111111
```

To print the user defined character, you would type:

```
PRINT CHR$(240)
```

Finally, to group blocks of characters together, you may for example specify:

```
semi$=CHR$(240)+CHR$(240)
PRINT semi$
```

.... or

```
terrace$=STRING$(15,240)
PRINT terrace$
```

Printing press....

PRINT is one of the first ever commands that you use when you start to learn computing. It's one of those BASIC commands that does what it says really.... or does it? In fact there's a lot more to **PRINT** than at first it seems, for instance **WHERE** should it print? and **HOW** should it print?....

Print formatting

The PRINT command has several ways in which it can be used. The simplest is to follow it by an item to be printed. This item can be a number, a string or a variable name.

```
PRINT 3
3
```

```
PRINT "hello"
hello
```

```
a=5
PRINT a
5
```

```
a$="test"
PRINT a$
test
```

Several items may be placed in one PRINT statement with each being separated by a separator, or TAB or SPC. The possible separators are either a semicolon or a comma. The semicolon causes direct continuation of printing, while a comma forces printing to continue in the next zone. The initial zone width is 13, but may be changed using the ZONE command.

```
PRINT 3;-4;5
3 -4 5
```

```
PRINT "hello ";"there"
hello there
```

```
PRINT "hello","there"
hello      there
```

```
PRINT 3,-4,5
3          -4          5
```

```
ZONE 4
PRINT 3,-4,5
3 -4 5
```

A point to note here is the fact that positive numbers are printed with a leading space, while negative numbers have a leading minus sign. All numbers have a trailing space. Strings are printed exactly how they appear between the quotes.

The function `SPC` takes a numeric expression as a parameter, and will print as many spaces as are specified by the expression. If the value is negative then zero is assumed, if it is greater than the current stream (window) width, then the stream width is assumed.

```
PRINT SPC(5)"hi"  
      hi  
  
x=3  
PRINT SPC(x*3)"hi"  
      hi
```

`TAB` is very similar except that it will print as many spaces as are needed so that the item to be printed will appear in the specified column.

The stream in which all printed output will appear is window 0 unless a stream director (`#`) is included before the list of items to be printed. Other streams may be used to output to the other windows. Streams 8 and 9 are special cases - anything printed on stream 8 will appear on the printer (if connected). Stream 9 directs output to a disc (or cassette) file. Note however, that the `WRITE` command should be used instead of `PRINT` for this purpose.

```
PRINT "hello"  
hello                                - window 0  
  
PRINT #0,"hello"  
hello                                - also window 0  
  
PRINT #4,"hello"  
hello                                - window 4  
                                      (At the top of the screen)  
  
PRINT #8,"hello"  
hello                                - on the printer  
                                      (If connected)
```

`TAB` and `SPC` are fine for simple print formats, but to specify a more detailed format, the `PRINT USING` command, together with a suitable format template can be used. A format template consists of a string expression containing special characters, each of which will specify a particular type of format. These characters, called 'Format field specifiers', are detailed in the description of the keyword `PRINT USING`, earlier in this manual. Some of the following examples however, may make their use a little clearer.

Firstly, here are the formats available for the printing of strings:

"\ \ " will print as many characters from a given string, as there are spaces + backslashes in the template.

```
PRINT USING "\ \ "; "test string"
test s
```

"!" can be used to print the first character of a string.

```
PRINT USING "!"; "test string"
t
```

....But probably the most useful string format is "&". This can be used to override the string wrapping feature of BASIC if required. By default, BASIC will start the printing of any string on a new line if it is too long to fit onto the current line. `PRINT USING "&";` can be used to override this.

(Use `BORDER 0`, so that you can see the edges of the paper.)

```
MODE 1:LOCATE 39,1:PRINT "too long"
```

```
too long
```

- line 1
- line 2

```
MODE 1:LOCATE 39,1:PRINT USING "&"; "too long"
```

```
o long
```

```
to
```

- line 1
- line 2

A large number of templates are available for the printing of numbers. Probably the simplest is `PRINT USING "#####"`, one digit is printed for each "#" that appears in the template.

```
PRINT USING "#####"; 123
123
```

The position of the decimal point may be included by the use of "."

```
PRINT USING "####.#####"; 12.45
12.45000
```

The digits before the decimal point may be grouped into threes, separated by commas if "," is included in the template before the decimal point.

```
PRINT USING "#####,.#####"; 123456.78
123,456.7800
```

Floating dollar and pound signs may be included in the format - i.e. a currency sign that will always be printed directly before the first digit of the number, even if it does not fill the complete format. This is achieved by the use of "\$\$" and "££" in the template.

```
PRINT USING "$$##";7
$7
```

```
PRINT USING "$$##";351
$351
```

```
PRINT USING "££####,.#";1234.567
£1,234.57
```

Note the rounding of the result.

The space before the result may be padded with floating asterisks by the use of "***" in the template.

```
PRINT USING "***###.#";12.22
***12.2
```

This may be combined with the currency symbols, (and then only one currency symbol is used) - i.e. "***\$....etc" or "***£....etc".

A "+" at the start of the template specifies to always print the sign of the result before the first digit. A "+" at the end of the template prints a trailing sign.

A "-" can only be placed at the end of the format, and specifies that a trailing minus sign be printed if the number is negative.

```
PRINT USING "+##";12
+12
```

```
PRINT USING "+##";-12
-12
```

```
PRINT USING "##+";12
12+
```

```
PRINT USING "##-";-12
12-
```

```
PRINT USING "##-";12
12
```

A "↑↑↑↑" format template can be used to print a number in exponential format.

```
PRINT USING "###.##↑↑↑↑";123.45  
12.35E+01
```

When using print formats for numbers, note that if a number is too long for the specified template, then a % symbol is printed before the result, to indicate that this has happened, and the result is NOT shortened to fit the specified template.

```
PRINT USING "####";123456  
%123456
```

Want your windows done?....

NB print #P is over line print.

The BASIC of the CPC664 provides a comprehensive method for setting up a maximum of eight text windows. Any of the text screen driving commands may then be directed to any one of these windows. *no thing*

The command that is used to set up a window is, simply enough: WINDOW. This is followed by 5 values. The first is optional and is used to specify which window is to be defined - if omitted, then window zero is assumed, all the normal BASIC prompts and messages (for example, 'Ready') are produced in window zero. The hash symbol (#) precedes this number to identify it as being a stream director. The next four numbers specify the left, right, top and bottom limits of the window. These values are column and row numbers, so they must lie in the range 1 to 80 for left/right and 1 to 25 for top/bottom.

The following example will define WINDOW (stream) number 4 to start in column 7 (left) and go on to column 31 (right), and to start at row 6 (top) and go down to line 18 (bottom). Reset the computer then type :

```
WINDOW #4,7,31,6,18
```

Nothing will appear to have happened after this command, however, try typing the following:

```
INK 3,9  
PAPER #4,3  
CLS #4
```

This will cause a large green rectangle to appear on the screen, and this is window number 4. The above also shows that PAPER and CLS may be used with any one of the eight windows by the inclusion of a stream director; its omission causes the command to operate on window 0 - the default window.

Each of the following commands may include a stream director to identify the WINDOW in which the command is to be carried out.

```
CLS
COPYCHRS
INPUT
LINE INPUT
LIST
LOCATE
PAPER
PEN
POS
PRINT
TAG
TAGOFF
VPOS
WINDOW
WRITE
```

The new green window which you have put on the screen, will have obscured some of the previous text (written on window number 0).

Text may be directed to any window by including a stream director in a PRINT statement:

```
PRINT #4,"hello there"
```

These words will appear at the top of the green rectangle, rather than on the following line as would have happened if....

```
PRINT "hello there"
```

...had been used. While typing in the earlier command, you will have noticed that part of the green window was overwritten by the text.

If you want all the normal BASIC messages to appear in window 4, then it can be swapped with the default window (0) by use of the WINDOW SWAP command:

```
WINDOW SWAP 0,4
```

The 'Ready' that follows this command will be printed in the green window. The cursor will be positioned directly beneath it. Now try typing the following:

```
PRINT #4,"hello there"
```

....and the words 'hello there' will appear directly beneath the WINDOW SWAP command in the old window 0, which is now window 4. It may also be apparent from this, that the current print position in each window is stored, so that even after a WINDOW SWAP, text is printed part way down the new stream 4 rather than starting at the top. Try the following:

```
LOCATE #4,20,1
PRINT "this is window 0"
PRINT #4,"this is window 4"
```

The 'window 0' message will appear on the line after the PRINT, while the 'window 4' message will appear at the middle of the top line of the whole screen.

Before a WINDOW command has been issued, all eight windows cover the entire screen. This is also true after a MODE command has been issued - so, if after using windows you find that the cursor ends up in a very small window, just type in MODE 1, as shown:

```
MODE 1
WINDOW 20,21,7,18
MO
DE
1
```

Don't worry about the word 'MODE' being split up - it will still work, and don't forget to leave a space between MODE and 1.

Now that you know a little about the way in which windows operate - try typing in the following short program:

```
10 MODE 0
20 FOR n=0 TO 7
30 WINDOW #n,n+1,n+6,n+1,n+6
40 PAPER #n,n+4
50 CLS #n
60 FOR c=1 TO 200:NEXT c
70 NEXT n
```

This sets up 8 overlapping windows and clears each to a different paper colour. When the program has finished running and 'Ready' appears, try pressing [ENTER] a few times to see how the scrolling of window 0 affects the coloured blocks on the screen. However, although these coloured blocks may be scrolling, the locations of the other windows do not actually move. Try the following:

```
CLS #4
```

....and you will see that the 4th window is still in the same position - the new coloured block having obscured those beneath it as one would expect. As a matter of interest, observe the differences when you type:

```
LIST
LIST #4
LIST #3
```

A further feature of the WINDOW command, demonstrated by the final program in this section, is that it does not matter if you specify the left and right window dimensions in reverse order. This means that if the value of the first parameter is greater than the second, BASIC will automatically sort the dimensions into the correct order. This also applies to the top and bottom window dimensions.

```
10 MODE 0
20 a=1+RND*19:b=1+RND*19
30 c=1+RND*24:d=1+RND*24
40 e=RND*15
50 WINDOW a,b,c,d
60 PAPER e:CLS
70 GOTO 20
```

If I may interrupt....

If you haven't already noticed, a major innovation in the software of the CPC664 is its unique ability to handle interrupts from BASIC - which means that AMSTRAD BASIC is capable of performing a number of simultaneous but separate operations within a program. Such a facility is sometimes referred to as 'multi-tasking', and it is implemented by the application of the commands AFTER and EVERY.

This facility is also clearly demonstrated in the way in which sound may be handled through facilities such as queues and rendezvous.

Every aspect of timing is referred to the master system clock, which is a quartz controlled timing system within the computer that looks after the timing and synchronisation of events that happen in the computer - things like the scanning of the display and clocking the processor. Where a function in the hardware is related to time, this can be traced back to the quartz master clock.

The software implementation is the AFTER and EVERY command, which in keeping with the user-friendly approach of AMSTRAD BASIC, do precisely what they say; i.e. AFTER the time that you have preset in the command, the program will divert to the designated sub-routine and perform the task defined therein.

The CPC664 maintains a real time clock. The **AFTER** command allows a BASIC program to arrange for sub-routines to be called at some time in the future. Four delay timers are available, each of which may have a sub-routine associated with it.

When the time specified has passed, the sub-routine is called automatically, just as if a **GOSUB** had been issued at the current position in the program. When the sub-routine finishes, using a normal **RETURN** command, the main program continues running where it was interrupted.

The **EVERY** command allows a BASIC program to arrange for sub-routines to be repeatedly called at regular intervals. Once again, four delay timers are available, and each may have a sub-routine associated with it.

The timers have different interrupt priorities. Timer 3 has the highest priority and timer 0 the lowest (see the chapter entitled 'For your reference....').

```
10 MODE 1:n=14:x=RND*400
20 AFTER x,3:GOSUB 80
30 EVERY 25,2 GOSUB 160
40 EVERY 10,1 GOSUB 170
50 PRINT "test your reflexes"
60 PRINT "press the space bar.";
70 IF flag=1 THEN END ELSE 70
80 z=REMAIN(2)
90 IF INKEY(47)=-1 THEN 110
100 SOUND 1,900:PRINT "cheat!":GOTO 150
110 SOUND 129,20:PRINT "NOW":t=TIME
120 IF INKEY(47)=-1 THEN 120
130 PRINT "you took";
140 PRINT (TIME-t)/300;"seconds"
150 CLEAR INPUT:flag=1:RETURN
160 SOUND 1,0,50:PRINT ".":RETURN
170 n=n+1:IF n>26 THEN n=14
180 INK 1,n:RETURN
```

AFTER and **EVERY** commands may be issued at any time, resetting the sub-routine and time associated with the given delay timer. The delay timers are shared by the **AFTER** and **EVERY** commands, so an **AFTER** overrides any previous **EVERY** for the given timer, and vice versa.

The **DI** and **EI** commands disable and enable timer interrupts whilst the commands between them are executed. This has the effect of delaying a higher priority interrupt from ever occurring during the processing of a lower priority interrupt. The **REMAIN** function disables, and returns the remaining count for one of the four delay timers.

Using data....

In a program that always requires the same set of information to be input at the start, it would make more sense if there were some way of entering all the values without having to ask the user to type them in every time. This facility is provided by the `READ` and `DATA` commands. The word `READ` is very similar to `INPUT` in that it can be used to assign values to variables. It differs, however, in the fact that values are read from `DATA` statements, rather than prompting for input from the keyboard. The following two examples show this:

```
10 INPUT "enter 3 numbers separated by commas";a,b,c
20 PRINT "the numbers are";a;"and";b;"and";c
run
```

```
10 READ a,b,c
20 PRINT "the numbers are";a;"and";b;"and";c
30 DATA 12,14,21
run
```

In the same way that different items in an `INPUT` statement are separated by commas, so it is with items in a `DATA` statement.

In addition to numeric values, constant strings may also be held in `DATA` statements:

```
10 DIM a$(8)
20 FOR i=0 TO 8
30 READ a$(i)
40 NEXT
50 FOR i=0 TO 8
60 PRINT a$(i);" ";
70 NEXT
80 DATA The,quick,brown,fox,jumps,over,the,lazy,dog
run
```

You may notice that although the `DATA` contains strings, the strings are not enclosed by double quotes `"`. The use of double quotes in `DATA` statements to delimit (separate) strings is optional, just as they are when typing a string in answer to an `INPUT` statement. One occasion that double quotes are useful however, is when the string `DATA` itself contains commas. If strings are not delimited by double quotes under these circumstances, the `READ` statement will use the commas to delimit the strings in the `DATA` statement.

```

10 READ a$
20 WHILE a$<>"*"
30 PRINT a$
40 READ a$
50 WEND
60 DATA The old, desolate, battered house creaked in
   the wind
70 DATA "The tall, slim, dark man coughed loudly."
80 DATA *
run

```

The string in line 60 contains commas, so each part will be READ and printed separately. The string in line 70 however, is delimited by double quotes and will be printed as a whole, as intended.

The above example illustrates the fact that data can be spread over a number of lines. READ will work down the lines in number order (60, 70, 80, etc.). Another fact that may not be obvious is that DATA statements can be placed anywhere within a program; before or after the READ statement that picks up the information.

If a program contains more than one READ statement, then the second READ will continue from the point at which the first READ stops:

```

10 DATA 123, 456, 789, 321, 654, 2343
20 FOR i=1 to 5
30 READ num
40 total=total+num
50 NEXT
60 READ total2
70 IF total=total2 THEN PRINT "the data is ok"
   ELSE PRINT "there is an error in the data"
run

```

Try editing line 10 so that one of the first 5 numbers is wrong, then run the program again. This technique of adding an extra value to the end of DATA statements which is the sum of all the other values, is a good method of detecting errors in DATA, especially if there are a large number of DATA lines - this is known as a 'checksum'.

If a program requires mixed data (strings and numbers), it is permissible to combine string and numeric items in READ and DATA statements, as long as the items are read correctly. For instance, if the DATA contained sequences of two numbers followed by a string - then it would only make sense to use a READ that was followed by two numeric variables, then a string variable:

```

10 DIM a(5),b(5),s$(5)
20 FOR i=1 TO 5
30 READ a(i),b(i),s$(i)
40 NEXT
50 DATA 1,7,fred,3,9,jim,2,2,eric,4,6,peter,9,1,alfonzo
60 FOR i=1 TO 5
70 PRINT s$(i),": ";a(i)*b(i)
80 NEXT

```

Alternatively, you may wish to separate the different types of data:

```

10 DIM a(5),b(5),s$(5)
20 FOR i=1 TO 5
30 READ a(i),b(i)
40 NEXT
50 FOR i=1 TO 5
60 READ s$(i)
70 NEXT
80 DATA 1,7,3,9,2,2,4,6,9,1
90 DATA fred,jim,eric,peter,alfonzo
100 FOR i=1 TO 5
110 PRINT s$(i),": ";a(i)*b(i)
120 NEXT

```

If the FOR loop in line 20 is now changed to:

```

20 FOR i=1 TO 4

```

....then the first two attempts to read strings in line 60 will produce '9' then '1'. These values are of course valid strings, but the result is not exactly what was planned! One method by which the program could be forced to work properly, would be to include the following commands:

```

15 RESTORE 80
45 RESTORE 90

```

The RESTORE statement will move the DATA-reading 'pointer' to the line specified, and can therefore be used in a conditional statement to pick a certain block of data to be read depending upon some criterion. For instance, in a multi-level game which has a number of different screens, the DATA for each screen may be picked according to some variable - for example 'level'. The following is just an example section of such a program:

```

1000 REM section to draw the screen
1010 IF level=1 THEN RESTORE 2010
1020 IF level=2 THEN RESTORE 2510
1030 IF level=3 THEN RESTORE 3010
1040 FOR y=1 TO 25
1050 FOR x=1 TO 40
1060 READ char
1070 LOCATE x,y: PRINT CHR$(char);
1080 NEXT x,y
:
2000 REM DATA for screen 1
2010 DATA 200,190,244,244,210 .....etc.
:
2500 REM DATA for screen 2
2510 DATA 100,103,245,243,251 .....etc.
:
3000 REM DATA for screen 3
3010 DATA 190,191,192,193,194 .....etc.

```

Another example use of DATA, READ and RESTORE might be in a program that plays a tune. Tone period values may be READ from DATA statements, and RESTORE used to make a section repeat by moving the pointer back to the beginning of a certain part of the music data:

```

10 FOR i=1 TO 3
20 RESTORE 100
30 READ note
40 WHILE note<>-1
50 SOUND 1,note,35
60 READ note
70 WEND
80 NEXT
90 SOUND 1,142,100
100 DATA 95,95,142,127,119,106
110 DATA 95,95,119,95,95,119,95
120 DATA 95,142,119,142,179,119
130 DATA 142,142,106,119,127,-1
run

```

The sound of music....

Of all the CPC664's features, the sound and envelope commands will probably seem to be the most daunting at first sight - this needn't be the case. With a little practice you should be able to make a whole range of different noises and even make the machine play a complete tune with harmonies.

Let's start by having a look at the first 4 parts of a **SOUND** command. These are: channel number, tone period, duration of note, and volume. You may be wondering what range each number has to be in.

We'll leave the first part (channel number) for a moment because it is quite complicated. The second part (tone period) can take any whole number value from 0 to 4095, but only some of these will actually produce recognisable notes from the musical scale - these are listed in part 5 of the chapter entitled 'For your reference....'. For instance the number 478 will play middle C, and 506 will play the note B below middle C, the values 479-505 will play a tone, but not one that corresponds to the piano scale. If the tone period is zero then no tone will be played - this will be useful when using 'noise' (explained later).

The third part of the **SOUND** command gives the duration of the note in units of a hundredth of a second. The value can generally be anywhere in the range 1 to 32767. However, if the value is zero, then the length of the note will be determined by the 'envelope' used (more of this later). If the value is negative then it indicates how many times the envelope should be played, so -3 would mean 'repeat the volume envelope three times' (again, this will be explained later).

The fourth part of the command is the volume. This can be from 0 to 15 (but if it is omitted, 12 is assumed). With the simple sounds that have been made so far, the volume remained constant for the whole time that the note played. When a 'volume envelope' is used to vary it, then the volume part of the **SOUND** command is taken as the starting value of the note.

Now for the channel number part of the command. You might as well know now that this is a 'bit significant' number - so you'll need to know a little about binary numbers to fully understand it (see part 1 of this chapter).

A sound can be played on one of three possible channels, if the computer is connected to a stereo amplifier, then one channel will be on the left, another on the right and the third on both (in the middle). To choose which channel(s) that a note should be played on, the following numbers are used:

1 channel A
2 channel B
4 channel C

To play on more than one channel, add up the numbers for the desired channels. For example to play on A and C use $1+4 = 5$.

SOUND 5,284

You may be wondering why channel C is given the number 4 and not 3 as you might expect. This is because each of these numbers is a power of two ($1=2^0$, $2=2^1$, $4=2^2$) so that they combine to form a binary number. If you think about a three digit binary number, then each of the three digits can be either 0 or 1, and this is used in the channel number to indicate whether the corresponding channel should be on or off. From the above example:

5 in decimal is equivalent to $1*4 + 0*2 + 1*1$ or 101 in binary. It follows then, that if each column of this binary number is labelled C, B, and A, this gives:

C	B	A
1	0	1

In other words, channel C is ON, B is OFF, and A is ON. If the note was to be played on channels A and B, then this would become:

C	B	A
0	1	1

And the binary number 011 is the same as $0*4 + 1*2 + 1*1 = 3$. So the **SOUND** command would be:

SOUND 3,248

This is, of course, the same value that would be found if you just added up the numbers for the channels to play on (remember $A=1$, $B=2$, $C=4$). So to play on A and B, the channel number is given by $1+2 = 3$.

If you didn't understand that - don't worry. As long as you can see that a combination of channels can be chosen by adding up the numbers for each of the channels to be used, then that's all you really need to know.

Unfortunately, there are yet more values that can be used in the channel number. The numbers 8, 16 and 32 are used to specify that the sound should 'rendezvous' with another channel (A, B, C respectively). You're probably wondering what is meant by the term *rendezvous*. Well, up to now the sounds that we have produced have gone straight to the specified channels. Try this:

SOUND 1,248,2000
SOUND 1,90,200

Unless you are a very slow typist, you will have noticed that you were able to type in the second of these commands before the first had finished. This is because the sound system is able to hold up to 5 sound commands for each channel in a 'queue'. If we wanted a sound to play on channel A, and then two sounds to play simultaneously on A and B, we would need some way of letting the computer know that the sound on B should not start until the second note on A was ready to play - i.e. making one channel wait for another. This is known as a *rendezvous*, and there are two ways in which this can be achieved:

```
SOUND 1,200,1000
SOUND 3,90,200
```

In this, the second note is directed to A and B, so it cannot start until the note on A finishes. The limitation with using this method (to make a note combination wait until all channels that it should play on are free) is that the same sound will be directed to each of the channels (in this case ,90,200 went to both A and B). The alternative method is to use the following:

```
SOUND 1,200,2000
SOUND 1+16,90,200
SOUND 2+8,140,400
```

Here, the second note on A is made to rendezvous with the sound on B (and the sound on B is made to rendezvous with the note on A). The advantage here is clear - although the second note on A was different to the note on B, the two were still linked so that neither could start until both channels were free - this is a *rendezvous*. Once again the values are bit-significant:

$$8 = 2 \uparrow 3, 16 = 2 \uparrow 4, \text{ and } 32 = 2 \uparrow 5$$

....so now the channel number can be seen as a binary number where the columns are headed:

Rendezvous C	Rendezvous B	Rendezvous A	Play on C	Play on B	Play on A
Add 32	Add 16	Add 8	Add 4	Add 2	Add 1

So for a note to play on C and rendezvous with A, you would use:

0	0	1	1	0	0
---	---	---	---	---	---

This is the binary number 1100, which is equal to $8 + 4 = 12$

Hence, a channel number of 12 would tell the computer to play a note on channel C, and wait for a note that has been marked to rendezvous with it on channel A.

If 64 ($2 \uparrow 6$) is added to the channel number, then this indicates that the note should be held. In effect, this means that the note will not play until the command **RELEASE** is used.

And finally, if 128 ($2 \uparrow 7$) is added to the number, then the queue of the channel specified will be cleared (or flushed). Therefore, if you start a sound that is going to continue for too long on a particular channel, then a quick way to stop it is to flush the channel:

```
SOUND 1,248,30000      (this would play for 5 minutes)
SOUND 1+128,0          (this will stop it short)
```

In direct command mode, a quicker way of stopping any long sounds is to press the **[DEL]** key at the start of a line; the short warning beep flushes all sound channels.

Now that we can hopefully send a sound to any of the three channels that we choose, (with rendezvous if necessary), it would be nice to be able to produce a little more than the rather unmelodious 'beep' that the simple **SOUND** command produces. The way to do this is to play the sound with an envelope - a pattern that defines how the note gets louder and quieter during the short time it is playing. A note produced by an instrument has an initial attack, where the volume rises very sharply. The volume of the note then falls away to a lower level which is sustained for a time, after which it fades away to zero. It is possible to give an envelope of this nature to the notes produced by the **SOUND** command. The associated **ENV** command is used to do this. First let's look at a simple example:

```
ENV 1,5,3,4,5,-3,8
SOUND 1,284,0,0,1
```

The **ENV** must come before the **SOUND** command for which it is used. To use this envelope in a **SOUND** command, its number is included as the fifth part of the **SOUND** command - in this case, the envelope is number 1. The first number in an **ENV** command is the number of the envelope that it defines. The **ENV** instruction contains information about how long the note will last and how loud it will get, so the duration and volume parts of the **SOUND** command are set to zero. The envelope defined above causes the sound to be increased in 5 steps, each step increasing the volume by 3, and each step being 4 hundredths of a second long. The volume is then to be reduced in 5 steps, each step decreasing the volume by -3, each step being 8 hundredths of a second long. In other words, the first number identifies which envelope is being defined, and this is then followed by two groups of three numbers, and in each of these groups, the first number indicates by how many steps the volume is to go up or down. The second number indicates by how much the volume is to go up or down at each of these steps, and the third number determines by how long each step of volume is to be held for.

The total length of time that each section will take is equal to the first value (number of steps) multiplied by the third value (pause time). The total increase or decrease in volume is equal to the number of steps multiplied by the step size. The overall length of an envelope with more than one section, is equal to the sum of the lengths of each section.

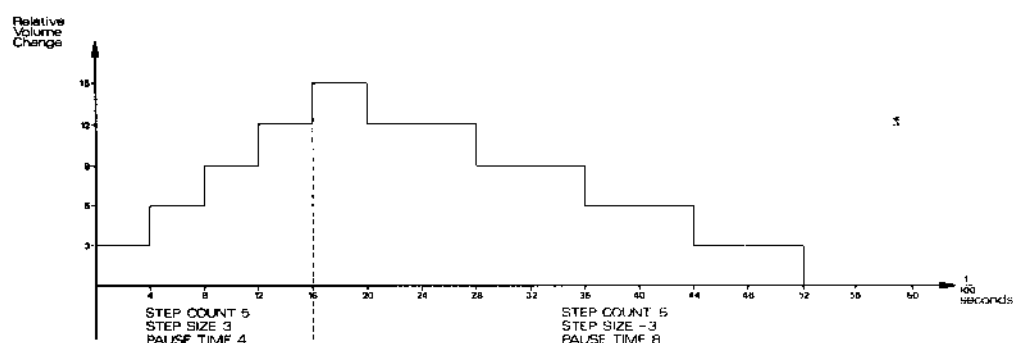
Of course, the starting volume needn't always be 0 (as set by the **SOUND** command). The above example produced a note that went up, then back down again. The following will go down, then back up:

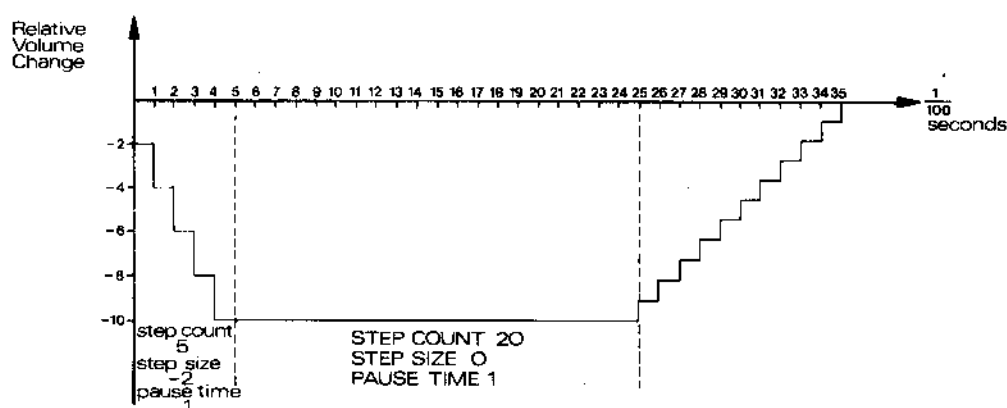
```
ENV 2,5,-2,1,20,0,1,10,1,1
SOUND 1,248,0,15,2
```

This envelope is given the number 2 and has three sections. In the first, the volume is reduced in 5 steps of -2. That is, it goes down by 2 at each step and there are 5 steps. The length of each of these steps is 1 hundredth of a second. The second section consists of 20 steps, but with 0 (zero) reduction or increase in the volume (i.e. constant volume) per step. Again, the length of each of these steps is 1 hundredth of a second. Finally, the third section has 10 steps, each increases the volume by 1, and once again each step is 1 hundredth of a second long.

The **SOUND** command has a starting volume of 15, so after the first section the volume will be reduced to 5, this is held constant for 20 hundredths of a second, then gets increased to 15 in the final section of the envelope.

It is perhaps a little difficult to visualise the shape of these envelopes. It often helps to draw them out on graph paper and take the values for the **ENV** command from this. The following show the shape of the two envelopes that have been defined so far:

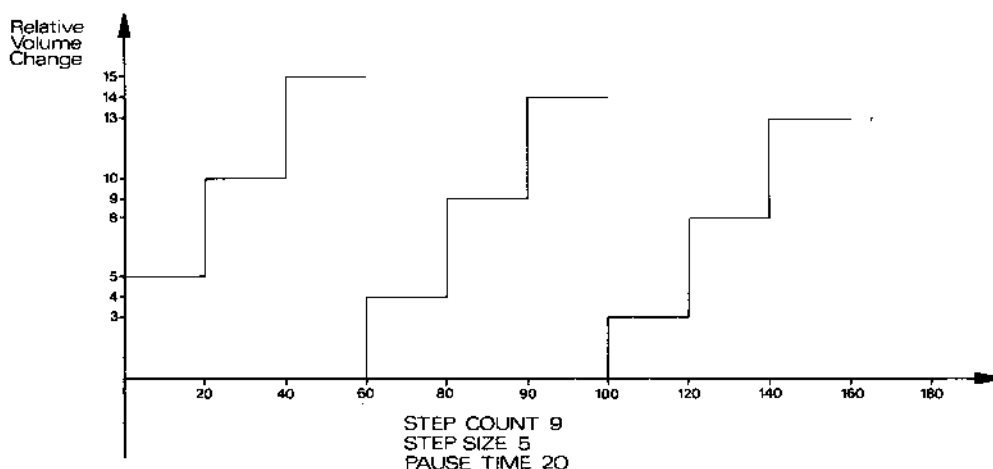




The maximum number of sections in an envelope is 5, and each section takes 3 values, so the ENV command can have up to 16 parts (including the first, that identifies which of the 15 envelopes (1 to 15) is being created). If the steps up or down cause the volume to go above 15 or below 0 then the value will wrap around; so the step above 15 is 0, and the step below 0 is 15:

```
ENV 3,9,5,20
SOUND 1,248,0,0,3
```

This simple envelope produces 9 steps, each step increasing the volume by 5, and each step lasting 20 hundredths of a second. After the first three steps therefore, the volume will be 15, so the next step will take it back round to 4, then 9 and so on. The diagram on the following page shows the effect :



The range of values for the number of steps is 0 to 127. The step size can be varied from -128 to +127 (negative values decrease the volume), and the pause time (i.e. the time between steps) can be anywhere in the range 0 to 255.

Now that we can create a volume envelope to give a characteristic shape, we may also want to define a characteristic tone pattern to include things such as vibrato - where the sound 'wavers' about the main tone of the note.

This is done in a very similar way to that in which volume envelopes are defined. Tone envelopes are defined using the ENT command. For example:

```
ENT 1,5,1,1,5,-1,1
SOUND 1,248,10,15,,1
```

A tone envelope is called into the SOUND command by entering the tone envelope's number as the sixth part of the SOUND command. Again the ENT must come before the SOUND command.

This first example of ENT specifies that in tone envelope number 1, there is to be 5 steps, each step increasing the tone period by 1, each step being 1 hundredth of a second. The next section of the envelope again specifies 5 steps, this time each step decreasing the tone period by -1, again each step being 1 hundredth of a second. This is a total length of $5 + 5 = 10$. Notice that this length was specified in the SOUND command because a tone envelope does NOT determine the length of time that a note will play (in the same way that a volume envelope does). If the length used in the SOUND command is shorter than the length of the tone envelope, then the last part of the tone envelope will be lost. If it is greater, then the last part of the note will play with a constant tone. This also applies if a volume envelope is used to determine the note length.

(Notice the absence of any number in the volume envelope part of the SOUND command - this is because we have not created a volume envelope for this sound.)

The majority of tone envelopes will probably be much shorter than the expected length of the note. For this reason, a tone envelope can be made to repeat throughout the time that the note is playing. This is done by specifying a negative envelope number, the positive equivalent of which must be used in the SOUND command:

```
ENT -5,4,1,1,4,-1,1
SOUND 1,248,100,12,,5
```

This will produce a vibrato effect on the note. When defining tone envelopes, it is usually best if they can be made to vary symmetrically about the initial tone period, so that when the note repeats, it will not go further and further off the initial frequency (pitch) - try this:

```
ENT -6,3,1,1
SOUND 1,248,90,12,,6
```

You will notice that this tone will fall in frequency quite dramatically because each time the envelope is repeated, the tone period will increase by 3 and this will happen 30 times ($90 / 3$). This sort of effect can be quite useful for 'warbling' notes and sirens:

```
ENT -7,20,1,1,20,-1,1
SOUND 1,100,400,12,,7

ENT -8,60,-1,1,60,1,1
SOUND 1,100,480,12,,8
```

The number of possible tone envelopes is 15 (in the range 1 to 15), with a negative value indicating that the envelope should repeat. The number of steps (the first value in each group of three), can be anywhere between 0 and 239. As in the volume envelope, the step size can be from -128 to 127, and the pause time should be between 0 and 255. Once again, an ENT command, like an ENV command, can have a maximum of five sections (each containing 3 values).

The final part that can be added onto a **SOUND** command is a seventh value that indicates the level of noise which should be included in the sound. One point that should be borne in mind when including noise in a sound, is the fact that there is only one noise channel, so each subsequent noise period value overrides any previous one.

The noise can be added to a tone to give it a different sound, or can be used quite separately, by setting the tone period (second part) of the **SOUND** command to 0, so that only noise is present. This is of use in making percussive type sounds. Try this

```
ENT -3,2,1,1,2,-1,1
ENV 9,15,1,1,15,-1,1
FOR a=1 TO 10: SOUND 1,4000,0,0,9,3,15: NEXT
```

This could form the basis of a train noise. You will notice that this combines both types of envelopes and noise.

The duration and volume parts of the **SOUND** command are both set to 0 as they are determined by the volume envelope.

Now that we can hopefully use the **SOUND**, **ENV** and **ENT** commands to their full, we can look at various other associated commands and functions.

In describing the channel number of the **SOUND** command, you will remember that by adding 64 to it, the sound was marked as 'held' so that it would remain in the queue, without playing until released. The way in which a sound can be released is by use of the **RELEASE** command. The word **RELEASE** is followed by a bit significant number, where each bit is used to indicate one of the three possible channels to be released. Once again, it is not important to fully understand this, as long as you realise that:

4 means channel C
2 means channel B
1 means channel A

....and a combination of channels is released by adding up the values for each of the channels. So, to release held sounds on all three channels the following would be used:

```
RELEASE 7
```

....where $7 = 1 + 2 + 4$. If no channels are held, then the **RELEASE** command is ignored. Try the following:

```
SOUND 1+64,90
SOUND 2+64,140
SOUND 4+64,215
RELEASE 3: FOR t=1 TO 1000: NEXT: RELEASE 4
```

The sounds that you might expect from these **SOUND** commands are not produced until the first **RELEASE** command which allows the sounds on channels A and B to play. After the delay, the sound on channel C is **RELEASED**.

There is yet another method by which more than one sound can be made to rendezvous. When a sound is added to a queue that has the hold bit set (64 added), then it is not just that sound which is held, but all subsequent sounds sent to that queue. If more than four further sounds are sent to the held queue, then the machine will pause until the queue is released, perhaps by using a sub-routine that is called after a fixed period of time (using **AFTER** or **EVERY**). However this is not a particularly good method for using the sound system, as the program that contains the sound commands may pause from time to time as the sound queues fill up. This is also true if a lot of long sounds are added in quick succession. Try this:

```
10 FOR a=1 TO 8
20 SOUND 1,100*a,200
30 NEXT
40 PRINT "hello"
run
```

You will notice that the word 'hello' does not appear instantly, but only after the first three sounds. This is because program execution cannot continue until there is a free space in the queue.

The BASIC contains an interrupt mechanism, rather like that used in the **AFTER** and **EVERY** commands, and in **ON BREAK GOSUB**. This enables you to specify a sound playing sub-routine that is only called when a free space appears in the required queue. Try this:

```
10 a=0
20 ON SQ(1) GOSUB 1000
30 PRINT a;
40 GOTO 30
1000 a=a+10
1010 SOUND 1,a,200
1020 IF a<200 THEN ON SQ(1) GOSUB 1000
1030 RETURN
run
```

You will notice that the program never pauses. The **SOUND** command is only called when channel A's queue (number 1) has a free slot. This condition is detected by the **ON SQ(1) GOSUB** command in line 20. The command initialises an interrupt mechanism that will call the sound sub-routine when a free slot appears in the specified queue. Once **ON SQ GOSUB** has been used, it must be re-initialised, and this is done by line 1020 in the sound sub-routine. In this example, the sound sub-routine only re-initialises itself while the value of 'a' is less than 200.

In a complete program that may be moving objects on the screen, adding up totals and the like, a background tune can be kept continually playing by having a sub-routine called to play each note only when there is a free slot in the queue. This ensures that the program does not pause while waiting for a free slot to appear. If the note values for this tune were being read from `DATA` statements, then the sound routine could be set stop re-initialising itself just before the data is exhausted.

The number within the brackets of the `ON SQ() GOSUB` command can be 1, 2 or 4 depending on which channel queue is to be tested for a free slot.

There is a function `SQ()` that may be used within a program to read the current state of any of the sound channels. The number within the brackets after `SQ` is again 1, 2, or 4, to specify for which channel the information should be returned. The function returns a bit-significant number and will again require an understanding of binary numbers to decipher it. The bits of the value returned have the following significance:

BIT	DECIMAL	MEANING
0, 1, 2	1, 2, 4	Number of free spaces in the queue
3	8	Note at head of queue is marked to rendezvous with A
4	16	Note at head of queue is marked to rendezvous with B
5	32	Note at head of queue is marked to rendezvous with C
6	64	Top note in queue has hold bit set (the queue is held)
7	128	A note is currently playing

Try this simple example:

```
10 SOUND 2,200
20 x=SQ(2)
30 PRINT BIN$(x)
run
```

This will print the binary number `10000100`, in which bit 7 is set, indicating that the channel was currently playing when the `SQ` function was used. The last three digits `100`, when converted to decimal give the value 4 indicating that there were 4 free entries in the queue. This function may be used to test a queue's status at a given point within a program - unlike `ON SQ() GOSUB` which will test and react to a queue's status at an indeterminate point.

So far, all the examples have just dealt with one or two notes of sound. Processing a whole group of unrelated notes, for example in a piece of music, can be achieved by listing the required notes in a `DATA` statement, from where they can be `READ` into a `SOUND` command:

```

10 FOR octave=-1 TO 2
20 FOR x=1 TO 7: REM notes per octave
30 READ note
40 SOUND 1,note/2↑octave
50 NEXT
60 RESTORE
70 NEXT
80 DATA 426,379,358,319,284,253,239
run

```

The final example program in this section greatly elaborates on this basic principle. The tune and rhythm play on channels A and B, using rendezvous to keep in step. The example demonstrates one of the ways in which DATA can be formatted so as to include note, octave, length and rendezvous information:

```

10 REM line 190 gives treble clef tune
20 REM line 200 gives bass clef tune
30 DIM scale%(12):FOR x%=1 TO 12:READ scale%(x%):NEXT
40 ch1%=1:READ ch1$:ch2%=1:READ ch2$
50 CLS
60 Spd%=12
70 scale$=" a-b b c+c d-e e f+f g+g"
80 ENV 1,2,5,2,8,-1,10,10,0,15
90 ENV 2,2,7,2,12,-1,10,10,0,15
100 ENT -1,1,1,1,2,-1,1,1,1,1
110 DEF FNMS(s$,s)=MID$(s$,s,1)
120 ch1%=1:GOSUB 200
130 ch2%=1:GOSUB 380
140 IF ch1%+ch2%>0 THEN 140
150 END
160 DATA &777,&70c,&6a7,&647,&5ed,&598
170 DATA &547,&4fc,&4b4,&470,&431,&3f4
180 DATA 4cr4f4f1f1g1A1-B2C2f4g2g1A1-B6A2Cr1f1g1f1g1a1-
b1A1-b2C2g2A2g2f1g1a2g2f6e2c2e2c2g2e2c1-B1A2g2f4e4d
8c4f3f1c2d4-b2fr2-B2A2g2f6e2gr4C4-B1a1f1-b1g2c2-b4a
4g4fr6A2A2-B4-B2Ar2-B2A2g2f6e2g4C4-B1A1f1-B1g2C2-B4
A4g8f.
190 DATA r4f4f8f4e4c4fr8f4e2f2e4d2e2d8c8c6e2f4g4g8e4f3f
1c4dr8g4cr4e4c6f2d4c4c8fr8-e4dr8g8c4e4c6f2d4c4c8f.
200 REM send sound to channel A
210 p1$=FNMS(ch1$,ch1%)
220 IF p1$<>"r" THEN r1%=0:GOTO 240
230 r1%=16:ch1%=ch1%+1:p1$=FNMS(ch1$,ch1%)

```

continued on the next page

```

240 IF p1$="." THEN ch1%=0:RETURN ELSE l1%=VAL(p1$)
250 ch1%=ch1%+1
260 n1$=FNMS$(ch1$,ch1%)
270 ch1%=ch1%+1
280 IF n1$="+" OR n1$="-" THEN 350
290 n1$=" "+n1$
300 nd1%=(1+INSTR(scale$,LOWER$(n1$)))/2
310 IF ASC(RIGHT$(n1$,1))>96 THEN o1%=8 ELSE o1%=16
320 SOUND 1+r1%,scale%(nd1%)/o1%,Spd%*l1%,0,1,1
330 ON SQ(1) GOSUB 200
340 RETURN
350 n1$=n1$+FNMS$(ch1$,ch1%)
360 ch1%=ch1%+1
370 GOTO 300
380 REM send sound to channel B
390 p2$=FNMS$(ch2$,ch2%)
400 IF p2$<>"r" THEN r2%=0:GOTO 420
410 r2%=8:ch2%=ch2%+1:p2$=FNMS$(ch2$,ch2%)
420 IF p2$="." THEN ch2%=0:RETURN ELSE l2%=VAL(p2$)
430 ch2%=ch2%+1
440 n2$=FNMS$(ch2$,ch2%)
450 ch2%=ch2%+1
460 IF n2$="+" OR n2$="-" THEN 530
470 n2$=" "+n2$
480 nd2%=(1+INSTR(scale$,LOWER$(n2$)))/2
490 IF ASC(RIGHT$(n2$,1))>96 THEN o2%=4 ELSE o2%=8
500 SOUND 2+r2%,scale%(nd2%)/o2%,Spd%*l2%,0,2
510 ON SQ(2) GOSUB 380
520 RETURN
530 n2$=n2$+FNMS$(ch2$,ch2%)
540 ch2%=ch2%+1
550 GOTO 480
run

```

Graphically speaking....

This section describes the graphics facilities available. The first example builds up slowly demonstrating each major feature in turn.

To start off with, we will divide the screen into a text window (line 40) and a graphics window (line 30), setting the MODE and a couple of flashing colours along the way (line 20):

```
10 REM mask and tag in window
20 MODE 1:INK 2,10,4:INK 3,4,10
30 ORIGIN 440,100,440,640,100,300
40 WINDOW 1,26,1,25
50 CLG 2
```

If you RUN this program you will see a square of flashing colour halfway down the right-hand side of the screen. This square has been cleared to ink number 2 (flashing magenta/cyan) by line 50, and the origin of co-ordinates has been moved to the bottom left hand corner of the square. The MODE command has set the graphics cursor to the origin of co-ordinates (X=0, Y=0) so we can draw a diagonal line across the square with line 60:

```
60 DRAW 200,200,3
```

RUN the new program to see the effect. Now add:

```
80 MOVE 0,2:FILL 3
```

Line 80 puts the graphics cursor just inside one of the two halves of the square and fills it with ink 3. The boundary of the fill is the edge of the graphics window (in this case also the edge of the square) and anything drawn in the current graphics pen (3) or anything drawn in the ink being used to fill (also 3).

Now RUN the resulting program.

To prove the point about 'fill' edges, add line 70 below. Note that it is only the fact that the fill is in the same ink as the diagonal line that restricts the fill to half the square.

```
70 GRAPHICS PEN 1
run
```

Edit line 80 to FILL with ink 1 then RUN to prove this last point. Then restore it back to the original (FILL 3).

Now add lines 100 to 140, which draw a box:

```
100 MOVE 20,20
110 DRAW 180,20
120 DRAW 180,180
130 DRAW 20,180
140 DRAW 20,20
run
```

The box is drawn in ink 1 because of line 70. If line 70 were omitted then we would need to add a ', 1' either as the third parameter of the `MOVE` command in line 100, or the third parameter of the `DRAW` command in line 110 in order to instruct the computer to change graphics pens.

Join the dots....

Lines need not be solid, they can be dotted. The `MASK` command allows us to specify the size of the dots. The pattern will repeat every 8 pixels and each subsequent line will continue the dotting scheme from where the last line left off. A new `MASK` command (probably with the same parameter as the current one) will reset the dotting logic to the start of the 8 pixel pattern.

The dotting pattern is actually a single byte binary number where the bits set indicate where to put pen ink. In our example we will use a binary constant (called up by the '&X') indicating that we want four pixels drawn in the middle of each 8 pixel group, with the two either side not drawn. This will give a dashed line with four pixels on and four pixels off. To do this, add:

```
90 MASK &X00111100
run
```

But hold on a moment! This program does not give us a smooth continuation of the dotting around the corners as we expected. The reason for this is that each corner point is actually plotted twice, once as the last pixel of one line, and again as the first pixel of the next line. A clumsy way around this is to type in:

```
115 MOVE 180,22
125 MOVE 178,180
135 MOVE 20,178
run
```

....which produces the desired effect. However there is a simpler way, which is to add a second parameter ', 0' to the end of the `MASK` command which tells the computer NOT to plot the first point of each line. Edit line 90 to read:

```
90 MASK &00111100,0
```

....and delete the lines you just added by typing:

```
115
125
135
```

Now RUN, and once again the dotted box is symmetrical. Note that if the second MASK parameter is ',1' the command will be reset to draw whole lines including the first pixel.

Now look in between the dashes of the line. There is something in the bottom right triangle which is not in the top left triangle. This is the graphics paper, which is set to ink 2 by the CLG 2 command in line 50, but is invisible in the top left triangle because it is the same colour as the background. Alter line 50 to read:

```
50 CLG 2:GRAPHICS PAPER 0
```

....and re-run the program. The paper now shows up clearly all round the box.

It is possible make the graphics paper invisible, or, as we call it 'transparent'. This means that a dotted line drawn over an existing picture will preserve the gaps between the dots. The graphics drawing is made transparent by adding a ',1' parameter to the GRAPHICS PEN command. (It is reset to non-transparent or 'opaque' by a ',0' parameter). Alter line 70 to read:

```
70 GRAPHICS PAPER 1,1  
run
```

....and observe the result.

As well as drawing lines (and plotting points) it is possible to write normal text characters at the position of the graphics cursor. This has the advantage that we can position the characters with much greater accuracy (within one pixel rather than within 8 pixels) and also that we can draw characters with the added flavour of graphics ink modes (see ahead).

To write characters at the graphics cursor location, simply position the graphics cursor at the top left-hand corner of where you want the character to be drawn, then issue the command TAG (or TAG #1 etc, for other text streams) followed by normal PRINTing commands. The graphics cursor is automatically stepped right by 8 pixels after each character is drawn. Add:

```
160 MOVE 64,108  
170 TAG  
180 PRINT "SALLY"  
190 TAGOFF  
run
```

(Any messages output by BASIC will be sent to the text screen irrespective of the state of the TAG/TAGOFF switch, but it is good practice to cancel the TAG assignment as soon as the TAGging is over.)

But what are the arrows after the name? Well, these are carriage-return `CHR$(13)` and line-feed `CHR$(10)` symbols. The graphics drawing software translates all of the first 32 ASCII characters into their printable versions (as if they were each preceded by a `CHR$(1)`; when sent to the text screen). The reason for this is that most of the first 32 control codes are meaningful only to the text screen. By the same token for example, if we overflow to the right of the graphics window, no wrap-around is performed.

The carriage-return and line-feed symbols can be suppressed by the normal technique of ending the `PRINT` statement with a semicolon, i.e:

```
180 PRINT "SALLY";  
run
```

Text sent to the graphics screen using `TAG` is influenced by the same `GRAPHICS PEN` commands as the line drawing. Thus currently, the name is written in `GRAPHICS PEN 1`, and is transparent. The command:

```
150 GRAPHICS PEN 1,0  
run
```

....will switch back to opaque paper, whilst:

```
150 GRAPHICS PEN 0,1  
run
```

....will write with ink number 0, transparently.

Now delete line 150 and `RUN` again. Ink number 1 + transparent mode (previously set in line 70) will be restored to the graphics pen.

Transparent characters

It is also possible to write characters to the text screen transparently, by using one of the control codes provided. Add:

```
200 PRINT #2,CHR$(22);CHR$(1)  
210 LOCATE #2,32,14:PRINT #2,"*****"  
220 LOCATE #2,32,14:PRINT #2,"-----"  
230 PRINT #2,CHR$(22);CHR$(0)  
run
```

Line 200 sets stream #2 to transparent mode. Note how the underlining appears to 'overstrike' the asterisks. This demonstrates that it is possible to build up composite characters, even in multiple colours. Line 230 switches off transparent mode, setting stream #2 back to opaque mode.

Ink modes

It is possible to draw using a graphics ink mode which combines the current drawing with the ink already on the screen. The final ink for each pixel is calculated by forming the logical combination of the old ink for the pixel with the graphics ink (pen or paper) being plotted. The logical combinations provided are XOR, AND and OR. The ink mode can be set either as the fourth parameter of DRAW/DRAWR, PLOT/PLOTR and MOVE/MOVER commands or by PRINTing a CHR\$(23); CHR\$(mode) control code sequence. In each case a value of 1 sets XOR plotting, 2 sets AND plotting and 3 sets OR plotting. A value of 0 restores the 'force' mode, where the graphics ink is used 'as is'.

The next example demonstrates the use of the XOR combination. XOR is often used in so-called Turtle Graphics because it has the property that drawing the same pattern twice restores the original image. Thus the square drawing routine is executed twice (in lines 110 and 130), and the TAGged printing is also executed twice (in lines 170 and 190). The FRAME commands cause enough delay to make the effect visible. Note the use, in line 90, of commands without a first parameter. This is quite in order in these commands, and simply leave the current settings of the first parameter unchanged.

The third parameter (,1) of the MOVE command in line 220 sets the GRAPHICS PEN to 1, overriding the '3' set in line 60. The XOR mode is set by the fourth parameter of the DRAW command in line 230. Note once more the missing parameter.

A reminder of the first-point plotting effect can be seen by removing the MASK command in line 90. The corners of the square disappear because they are drawn twice (at the end of one line and the start of the next) and are therefore cancelled out by the XORing action.

```
10 REM XOR ink modes
20 MODE 1:INK 2,10:INK 3,4
30 ORIGIN 440,100,440,640,100,300
40 WINDOW 1,26,1,25
50 CLG 2:GRAPHICS PAPER 0
60 DRAW 200,200,3
70 MOVE 2,0:FILL 3
80 ORIGIN 440,0,440,640,0,400
90 GRAPHICS PEN ,1:MASK ,0
100 FOR y=60 TO 318 STEP 2
110 GOSUB 220
120 FRAME:FRAME
130 GOSUB 220
```

continued on the next page

```

140 NEXT
150 TAG
160 FOR y=60 TO 318 STEP 2
170 MOVE 96,y:PRINT CHR$(224);
180 FRAME:FRAME
190 MOVE 96,y:PRINT CHR$(224);
200 NEXT
210 END
220 MOVE 90,y,1
230 DRAW 20,0,,1
240 DRAW 0,20
250 DRAW -20,0
260 DRAW 0,-20
270 RETURN
run

```

Animation

It is possible to produce an animation effect by switching the colours assigned to inks. Although the contents of the screen memory are unchanged, there appears to be movement. An example of this is included in the 'Welcome' program on your master system disc (type RUN "disc" to see that demonstration). The simple palette switching of that example is not enough however, if the animated patterns are required to overlap. The next example uses ORing of inks to write the numbers 1 to 4 onto the screen. (The shape is determined by scanning the character printed at the bottom left hand corner and reproducing what is found, in big block graphics.) The numbers are written in turn using inks 1,2,4 and 8 with the OR mode turned on - in this case with a control character sequence, see line 50.

Lines 160 onwards rotate the palette according to a mathematical formula which results in one block graphics number at a time being displayed. The inks are set by inspecting each ink in turn and determining if it includes the binary component that we are looking for. For example, the number 3 was drawn in ink 4 and therefore to show the number 3 we must allocate a visible colour to all inks whose number contains a binary 4. Those inks are:

4(0100), 5(0101), 6(0110), 7(0111), 12(1100), 13(1101), 14(1110), 15(1111)

In a practical application the inks which require to be changed at each stage in the animation would be calculated, and lines 180 to 200 would be replaced by a speedier section of program.

```

10 REM latch animation
20 ON BREAK GOSUB 220
30 FOR i=1 TO 15:INK i,26:NEXT
40 m(1)=1:m(2)=2:m(3)=4:m(4)=8
50 MODE 0:PRINT CHR$(23);CHR$(3);:TAG
60 FOR P=1 TO 4
70 GRAPHICS PEN m(p),1
80 LOCATE #1,1,25:PRINT#1,CHR$(48+p);
90 FOR x=0 TO 7
100 FOR y= 0 TO 14 STEP 2
110 IF TEST(x*4,y)=0 THEN 140
120 MOVE (x+6)*32,(y+6)*16:PRINT CHR$(143);
130 MOVE (x+6)*32,(y+7)*16:PRINT CHR$(143);
140 NEXT y,x,p
150 LOCATE #1,1,25:PRINT#1," ";
160 FOR p=1 TO 4
170 FOR i= 1 TO 25:FRAME:NEXT
180 FOR i=0 TO 15
190 IF (i AND m(p))=0 THEN INK i,0 ELSE INK i,26
200 NEXT i,p
210 GOTO 160
220 INK 1,26
run

```

Colour plane sprites

In the example above we have seen how, having written graphics in inks 1,2, 4 and 8, an animation effect can be produced by colour changing. If the same inks are used, but the colours set up in a different way then a completely different effect can be produced. This effect is known as 'colour planes' and is demonstrated in the example below.

```

10 REM mountains
20 DEFINT a-z
30 INK 0,1:INK 1,26
40 INK 2,6:INK 3,6
50 FOR i=4 TO 7:INK i,9:NEXT
60 FOR i=8 TO 15:INK i,20:NEXT
70 MODE 0:DEG:ORIGIN 0,150:CLG:MOVE 0,150
80 FOR x=16 TO 640 STEP 16

```

continued on the next page

```

90 DRAW x,COS(x)*150+RND*100,4
100 NEXT
110 MOVE 0,0:FILL 4
120 cx=175:GOSUB 320
130 cx=525:GOSUB 320
140 SYMBOL 252,0,0,&C,&1F,&30,&7F,&FF
150 SYMBOL 253,0,6,&E,&F2,2,&F2,&FE
160 SYMBOL 254,0,&60,&70,&7F,&7F,&7F,&7F
170 SYMBOL 255,0,0,0,&F8,&EC,&FE,&FF
180 pr$=CHR$(254)+CHR$(255)
190 pl$=CHR$(252)+CHR$(253)
200 TAG:t!=TIME
210 FOR x=-32 TO 640 STEP 4
220 x2=((608-x)*2)MOD 640:hl=RND*10:hr=50*SIN(x)
230 GRAPHICS PEN 8,1:MOVE x,100+hr,,3:PRINT pr$;
240 GRAPHICS PEN 2,1:MOVE x2,115+hl,,3:PRINT pl$;
250 IF (TEST(x2-2,115+hl-12) AND 8)=8 THEN 380
260 IF TIME-t!<30 THEN 260
270 FRAME:t!=TIME
280 GRAPHICS PEN 7,1:MOVE x,100+hr,,2:PRINT pr$;
290 GRAPHICS PEN 13,1:MOVE x2,115+hl,,2:PRINT pl$;
300 NEXT
310 GOTO 210
320 MOVE cx,100
330 FOR x=0 TO 360 STEP 10
340 DRAW cx+SIN(x)*50+10*RND,100+COS(x)*25+10*RND,1
350 NEXT
360 DRAW cx,100:MOVE cx,90:FILL 1
370 RETURN
380 ENT -1,1,1,1
390 SOUND 1,25,400,15,,1,15
400 FOR y=100+hr TO -132 STEP -2
410 GRAPHICS PEN 7,1:MOVE x,y,,2:PRINT pr$;
420 GRAPHICS PEN 8,1:MOVE x,y-2,,3:PRINT pr$;
430 NEXT
440 GOTO 70
run

```

To explain how this works we must, once again, visualise the INK number in binary. Starting with the highest INK number (15), all the INKs with the '8' bit present (15 down to 8) are set to cyan. Then all the INKs with the '4' bit present (7 down to 4) are set to green. INKs 2 and 3, each with the '2' bit present are set to red, and finally INK 1 is set to bright white, with INK 0 remaining as blue.

On the screen the graphics are ORed into place - see lines 230 and 240. The colour seen on the screen in any particular pixel is determined by the most significant bit of the resultant at that point. Therefore an image in a 'more significant' plane will always obscure an image in a 'less significant' plane, but the background will be preserved and can be seen again if the 'more-significant' image is removed. The way to remove an image is to plot it using the AND ink mode with INK numbers of 7,11,13 or 14 removing original INKs of 8,4,2 and 1 respectively - see lines 280 and 290.

Each of the above 4 programs demonstrates particular aspects of graphics on the CPC664. Graphics is potentially the most exciting aspect of computing, and much of the excitement is generated by the ingenuity of you the programmer. It would be impossible to summarise the computer's full graphics capabilities in this instruction manual, and we suggest that you continue to experiment with the many graphics commands and functions provided.

The machine itself....

To conclude this chapter, a few words about the inside of your computer.

The CPC664 is based around the Z80 microprocessor. The machine also contains a 32K ROM that is addressed in two halves, the upper half contains the BASIC interpreter which lives in the memory map from &C000 to &FFFF. The lower half is mapped from &0000 to &4000 and contains the firmware which carries out all the mundane tasks such as running the machine, reading the keyboard, driving the screen, etc.

The CPC664 also contains a ULA that enables the 8 bit CPU to address more than 64K of memory. This is why the screen and the BASIC ROM can have the same address. The top 16K of RAM is used by the 6845 video controller chip to store the pixel colour information that generates the picture. The keyboard is scanned by the 8255 parallel interface chip, which is also used to drive the AY-3-8912 sound chip.

The printer port is provided by an 8 bit latch. There are 7 data bits, while the eighth bit is used to latch the strobe signal. At the expansion port, all the standard Z80 bus signals are brought out to the edge connector, but they are not buffered.

For further technical details of this computer including circuit diagrams, PCB layouts, etc, consult the CPC664 Service Manual, available from AMSTRAD.

Appendix 1

Digital Research & AMSTRAD

End User Program Licence Agreement

NOTICE TO USER - PLEASE READ THIS NOTICE CAREFULLY. DO NOT OPEN THE DISKETTE PACKAGE UNTIL YOU HAVE READ THIS LICENCE AGREEMENT.

OPENING THE DISKETTE PACKAGE INDICATES YOUR AGREEMENT TO BE BOUND BY THESE TERMS AND CONDITIONS.

1. Definitions

- In this Licence Agreement, the terms:

1. DRI means DIGITAL RESEARCH (CALIFORNIA) INC., P.O. Box 579, Pacific Grove, California 93950, owner of the copyright in, or authorised licensor of, the program.
2. Machine means the single microcomputer on which you use the program. Multiple CPU systems require additional licences.
3. Program means the set of programs, documentation and related materials in this package, together with all ancillary updates and enhancements supplied by DRI to you regardless of the form in which you may subsequently use it, and regardless of any modification which you make to it.
4. AMSTRAD means AMSTRAD CONSUMER ELECTRONICS PLC., Brentwood House, 169 Kings Road, Brentwood, Essex CM14 4EF.

You assume responsibility for the selection of the program to achieve your intended results, and for the installation, use and results obtained from the program.

2. Licence

You may:

1. Use the program on a single machine.
2. Copy the program into any machine readable or printed form for backup or modification purposes in support of your use of the program on a single machine. You may make up to 3 (three) copies of the program for such purposes. (Certain programs, however, may include mechanisms to limit or inhibit copying. They are marked 'copy protected'). Copying of documentation and other printed materials is prohibited. Disassembly of code is prohibited.
3. Modify the program and/or merge it into another program for your use on the single machine. (Any portion of this program merged into another program will continue to be subject to the terms and conditions of this Agreement).
4. Transfer the program and licence to another party if you notify DRI of name and address of the other party and the other party agrees to a) accept the terms and conditions of this Agreement, b) sign and forward to DRI a copy of the registration card and c) pay the then current transfer fee. If you transfer the program, you must at the same time either transfer all copies, including the original, whether in printed or machine readable form to the same party, or destroy any copies not transferred; this includes all modifications and portions of the program contained or merged into other programs.

You must reproduce and include the copyright notice on any copy, modification or portion merged into another program.

EACH DISKETTE IS SERIALISED, AND YOU MAY NOT USE, COPY, MODIFY, TRANSFER, OR OTHERWISE MAKE AVAILABLE TO ANY THIRD PARTY, THE PROGRAM, OR ANY COPY, MODIFICATION OR MERGED PORTION, IN WHOLE OR IN PART, EXCEPT AS EXPRESSLY PROVIDED FOR IN THIS LICENCE AGREEMENT.

IF YOU TRANSFER POSSESSION OF ANY COPY, MODIFICATION OR MERGED PORTION OF THE PROGRAM TO ANOTHER PARTY, YOUR LICENCE IS AUTOMATICALLY TERMINATED.

3. Term

The licence is effective until terminated. You may terminate it at any other time by destroying the program together with all copies, modifications and merged portions in any form. It will also terminate upon conditions set forth elsewhere in this Agreement or if you fail to comply with any term or condition of this Agreement. You agree upon such termination to destroy the program together with all copies, modifications and merged portions in any form.

4. Limited Warranty

THE PROGRAM IS PROVIDED 'AS IS'. NEITHER DRI NOR AMSTRAD MAKE ANY WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU (AND DRI OR AMSTRAD) ASSUME THE ENTIRE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

Neither DRI nor AMSTRAD warrant that the functions contained in the program will meet your requirements or that the operation of the program will be uninterrupted or error free.

However, AMSTRAD warrants the diskette on which the program is furnished, to be free from defects in materials and workmanship under normal use for a period of ninety (90) days from the date of delivery to you as evidenced by a copy of your receipt.

5. Limitations of Remedies

AMSTRAD's entire liability and your exclusive remedy shall be the replacement of any diskette not meeting this 'Limited Warranty' and which is returned to AMSOFT with a copy of your receipt.

IN NO EVENT SHALL DRI OR AMSTRAD BE LIABLE FOR ANY DAMAGES, INCLUDING ANY LOST PROFITS, LOST SAVINGS, OR OTHER SPECIAL INDIRECT OR CONSEQUENTIAL DAMAGES, EVEN IF DRI OR AMSTRAD HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES, OR FOR ANY CLAIM BY ANY OTHER PARTY.

6. Registration Card

DRI may from time to time update its programs. Updates will be provided to you only if a properly signed registration card is on file at DRI's main office or an authorised registration card recipient. DRI is not obligated to make any program updates, or to supply any such updates to you.

7. General

You may not sublicense, assign or transfer the licence or the program except as expressly provided in this Agreement. Any attempt otherwise to sublicense, assign or transfer any of the rights, duties, or obligations hereunder is void.

This Agreement shall be governed by and construed in accordance with the laws of England.

Should you have any questions concerning this Agreement, you may contact DRI by writing to Digital Research Inc., P.O. Box 579, Pacific Grove, California 93950.

THIS AGREEMENT CANNOT AND SHALL NOT BE MODIFIED BY PURCHASE ORDERS, ADVERTISING OR OTHER REPRESENTATIONS BY ANYONE, AND MAY ONLY BE MODIFIED BY A WRITTEN AMENDMENT EXECUTED BY YOU AND AN AUTHORISED OFFICER OF DRI AND AMSTRAD.

YOU ACKNOWLEDGE THAT YOU HAVE READ THIS AGREEMENT, UNDERSTAND IT AND AGREE TO BE BOUND BY ITS TERMS AND CONDITIONS. YOU FURTHER AGREE THAT IT IS THE COMPLETE AND EXCLUSIVE STATEMENT OF THE AGREEMENT BETWEEN YOU AND DRI AND AMSTRAD WHICH SUPERSEDES ANY PROPOSAL OR PRIOR AGREEMENT, ORAL OR WRITTEN, AND ANY COMMUNICATIONS BETWEEN YOU AND DRI OR AMSTRAD RELATING TO THE SUBJECT MATTER OF THIS AGREEMENT.

THIS AGREEMENT DOES NOT AFFECT YOUR STATUTORY RIGHTS.

Appendix 2

Glossary of Terms

Some commonly used terms from the world of computing, explained for CPC664 users....

Accumulator

A memory location within the microprocessor circuit at the heart of the microcomputer that stores data temporarily while it is being processed. Used extensively in machine code programming - BASIC users need never know it exists!

Acoustic coupler

Also known as an Acoustic Modem. An electronic attachment for a computer that connects a telephone handset to the computer and enables the latter to communicate over the normal voice telephone network. In this way, a computer can communicate with public information systems such as PRESTEL, and with other users of home computers, in order to exchange software, get data and information etc.

Address

The number in an instruction that identifies the location of a 'cell' in a computer's memory. By means of its address, a particular memory location can be selected so that its contents can be found and 'read', in the case of RAM, both written and read.

Adventure game

A cult with some, and a bore to others. A text-based computer game in which the player is invited to participate in a series of pseudo random events based on trying to find a way around a maze or labyrinth.

Algorithm

A grandiose name for a complicated formula or sum. A sequence of logical and arithmetic steps to perform a defined task in computing.

Alphanumeric

The attribute that describes the difference between a letter or number and a graphics character.

ALU

Arithmetic Logic Unit. The part of a microprocessor that carries out arithmetic and logical operations - not of direct concern except in machine code programming.

Ambiguous File Name

A file name containing one or more wildcard characters. Ambiguous filenames refer to more than one specific file name and are used to refer to one or more files at a time.

AMSDOS

AMStrad Disc Operating System. The program that allows Locomotive BASIC to access disc files.

AMSOFT

AMSTRAD's specialist computer support division, supplying software, peripherals, publications, specifically to enhance the CPC664 and its many applications.

A/D

Analogue

A state where change between a start and finish point occurs gradually rather than instantaneously. Computers are digital devices - most of the natural world is based on analogue principles, thus the computer has to perform an analogue to digital (A/D) conversion before it can process data from an analogue source.

Animation

Cartoons are the best known form of animation - computer animation is based on the concept of moving graphics to simulate the idea of 'live' movement.

Applications program

A program with a specific task rather than a general purpose software 'tool' such as an assembler, printer driver etc.

Arcade Game

The type of moving action computer video game where for example, spacemen invade and are vanquished, or insatiable monsters chase around a maze and gobble up the unwary. Figures under the control of the user have to avoid all manner of unpleasant 'deaths'. Generally good for the reflexes but of little educational benefit for the computer student.

Architecture

The plan relationship of the databus, peripheral and CPU handling aspects of a microcomputer.

Argument

An independent variable, e.g. in the expression $x+y=z$, the terms x , y , and z are the arguments.

Array

A 2 dimensional matrix (grid) in which data is stored by addressing with the 'horizontal' and 'vertical' co-ordinates.

Artificial intelligence AI

A structure in program techniques that enables the program to learn from its past experience.

ASCII

American Standard Code for Information Interchange. A commonly used way of representing the numbers, letters and other symbols that can be entered from the computer's keyboard or invoked using a variety of other commands.

Assembler

The practical method for programming in machine code, where the machine code instructions are invoked by mnemonics (letters that suggest the function being performed by the corresponding machine code routine).

Backup

A duplicate copy of information used as a safeguard in case the original is lost or accidentally damaged. Making a backup refers to the process of duplicating a disc or disc file.

Bar code

A computer readable printed code that can be read by optical techniques such as scanning by a low power laser. Look at the bottom of a box of soap powder to see an example.

Base

The prime numeric consideration of any mathematician. The basis of any system of number representation. The binary system has base 2; the decimal system has base 10, and the hexadecimal system has the base 16.

BASIC

Beginners' All-purpose Symbolic Instruction Code. An interpretive programming language used in almost all home computers. BASIC was specifically designed to be easy to learn and simple to use since it allows for programs to be 'glued' together and tested at any point in their development; as opposed to compiled types where the complete program must be run before any aspect can be properly tested.

Baud

A bit per second. The unit for measuring the rate at which digital data is transmitted in serial communication systems.

BCD

Binary Coded Decimal. A coding system for decimal numbers in which each digit is represented by a group of four binary digits.

BDOS

Basic Disc Operating System. This is the part of the CP/M operating system which provides an interface for a user program to use the functions of CP/M.

Benchmark

A standard task that can be given to different computers to compare their speed, efficiency and accuracy, e.g. calculating the square root of 99.999 squared.

Binary

The number system with base 2, in which all numbers are made up from the two binary digits 0 and 1. (See part 1 of the chapter entitled 'At your leisure....')

Binary number

A number represented in binary notation. Signified in CPC664 programming by the prefix &X. e.g. &X0101 = (decimal) 5.

BIOS

Basic Input/Output System. This is the hardware dependent part of CP/M that is written specifically for one type of computer. All the input and output to the screen, keyboard, disc and so on is performed through the BIOS.

Bit

Shortform of BInary digiT. A binary digit is one of the two digits, represented by 0 and 1, that are used in the binary number system.

Bit Significant

Where the information contained in a number is extracted by considering the state of each of the eight bits that make up the complete byte. The overall decimal value may have no meaning.

Boot

The process of loading an operating system into memory. When CP/M is started from BASIC a small boot program is loaded automatically from the disc, which then loads the rest of the operating system into memory.

Bootling or Bootstrapping

Programs and operating systems don't load themselves, they are 'bootstrapped' by a small routine in ROM (usually), that initiates the loading processes at a specific location in memory.

Boolean algebra

Is the statement of logical relationships where there can only be two answers: true or false. Usually signified as 0 or 1.

Buffer

An area of memory reserved for temporarily storing, or buffering, information during an information transfer.

Bug

A problem on a scale ranging from an 'unexpected feature' based on some obscure aspect of the use of a program (e.g. if you press four keys at once, the screen changes colour), to a sequence that completely and irrevocably crashes a computer program and wipes the memory clean of all data.

Built-in commands

Commands that are part of an operating system. They are always quicker than transient commands because they are not accessed from disc.

Bus

A group of connections either within the computer, or connecting it to the outside world that carries information on the state of the CPU, the RAM and other hardware features. The CPC664 bus is presented on the circuit board connector marked **EXPANSION**, at the rear of the computer.

Byte

A group of eight bits, which forms the smallest portion of memory that an 8-bit CPU can recall from, or store in memory.

CAD

Computer Aided Design. Usually an interaction of computing power and graphics to provide an electronic drawing board, although any calculation performed on a computer in pursuance of a 'design' comes under the heading of CAD.

CAE

Computer Aided Education. Further nourishment for the buzzphrases of computing. The use of the computer to help with education. CAI (Computer Aided Instruction) and CAL (Computer Aided Learning) are two aspects of CAE.

Cartridge

A specially packaged memory integrated circuit containing software which can be plugged directly into a socket specifically provided for the purpose on the computer. Cartridge software loads and runs quickly and easily, but costs considerably more than software supplied on disc.

Cassette

Apart from the obvious recording tape variety, a generic term that encompasses a variety of 'packages' - including ROM software etc.

CCP

Console Command Processor. This is a module of CP/M that interprets and executes user input from the keyboard. Usually commands are input which the CCP loads and executes.

Character

Any symbol that can be represented in a computer and displayed by it, including letters, numbers and graphics symbols.

Character set

All the letters, numbers and symbols available on a computer or printer. The fact that a character exists on a computer does not imply it is accessible on any printer.

Character string

A piece of <variable> data comprising a sequence of characters that can be stored or manipulated as a single unit, e.g. a word or a collection of words.

Chip

A misleading but popular reference to any form of monolithic electronic integrated circuit. The 'chip' is actually a small slice of specially processed silicon material, on which the circuit is fabricated.

Clock

The reference timing system in the computer used to synchronise and schedule the operations of the computer. A real time clock is one that maintains the hour, date etc.

Code

Apart from the more literal implications, frequently used by programmers as an abbreviation of 'machine code'.

Cold start

The process of booting and initialising an operating system. A cold start of CP/M is performed when the `1 CPM` command is used.

Command

A programming instruction.

Compiler

A complex program that converts complete programs written in a high level interpretive languages like BASIC into the direct instruction code of the microprocessor, thereby enabling operation at much greater speeds.

Computer generations

Technological landmarks have delineated several distinct steps in computer technology, and the groupings within these various strata are known as the 'generations' of design technologies.

Computer literacy

Another grandiose expression meaning understanding computers.

Console mode

CP/M direct mode; the A> appears on the screen, and the system awaits input of a CP/M or utility command.

Corruption

The destruction or alteration of the contents of a disc file or memory, in an undesirable and potentially unrecoverable manner.

CP/M

Control Program for Microcomputers. A disc based operating system by Digital Research that provides a standard systems interface to software written for a wide range of microprocessor based computer systems.

CPU

Central Processing Unit. The component at the heart of any computer system that interprets instructions to the computer and causes them to be obeyed, in a microcomputer, the CPU is the microprocessor device itself.

Cursor

A movable marker, indicating where the next character is to appear on the screen.

Cursor control keys

Keys that move the Cursor around the screen, and are frequently used to control the direction of action in arcade games, indicated by arrows printed on the top.

Daisy-wheel printer

A printer that can produce high quality or 'typewriter quality' documents. Printed characters are created by the impact of letters against the ink or film ribbon.

Database

An array of any type of data in a variety of computer addressable formats.

Data capture

The term which describes the collection of data from any outside sources that are linked in some way to a central computer.

Debugging

The process of fixing the bugs in a program by a combination of 'suck it and see' and more scientific methods.

Decimal notation

Also known as the Denary system, for numbers with base 10, using the digits 0 to 9, representing numbers of units, tens, hundreds, thousands and so on.

Default

The value assumed in the absence of any user output. For example, when CP/M is started Drive A : is assumed to be the default drive.

Delimiter

(See Separator)

Diagnostic

A message automatically produced by a computer to indicate and identify an error in a program.

Digital

Describes the transition of a changing quantity in terms of discrete steps rather than by a continuous process. The opposite of analogue.

Digitiser

A means of plotting analogue information into a computer. Commonly referred to in conjunction with graphics tablets.

Directory

A section of disc containing entries for each file on the disc. A list of the contents of a disc.

Disc (or disk)

A flat, thin circular piece of plastic, coated on one or both sides with a magnetic oxide surface and used as a medium for storing data. The disk is housed in a protective envelope, with access for the reading head provided by a 'window'. On a 3" disc the window is covered by a metallic shutter, which automatically slides across when the disc is out of the disc drive.

Disc drive

The mechanisms used to spin and access the data on the surface of the disc.

Documentation

The manuals that are supplied with computers or software to explain how they are operated.

DOS

Disc Operating System. The software that controls all the operations of a disc drive.

Dot matrix

A rectangular grid of dots on which a character can be displayed by the selection of certain dots.

Double sided

A disc that can store information on both sides. A double sided disc drive can access both sides of a disc without the need to change the disc over.

Download

The transfer of information from one computer to another - the computer receiving the data is generally referred to as the machine downloading. The other end of the link is uploading.

Dr. LOGO

Digital Research's version of LOGO, a programming language with a graphics turtle.

Dumb terminal

A computer terminal that simply acts as a medium for input and output without any processing of the information passing through. Note that a mindless terminal is one where even the display drive electronics are absent, and that the screen display information is fed in as pure video.

Edit

To correct or make changes to data, a program or text.

Editor

A program that is usually in the ROM of the computer, which enables the editing process to be carried out.

EPROM

Erasable Programmable Read only Memory. Similar to the PROM, except that the data contained in the chip can be erased by ultra-violet light, and new program recorded. An EEPROM is similar, except that it may be electronically erased.

Expression

A simple or complex formula used within a program to perform a calculation on data -the expression will usually define the nature of the data it can handle. In Dr.Logo an expression consists of a procedure name followed by any necessary inputs to the procedure.

Fifth generation computers

Mainly large mainframe computers that are promised to arrive with the ability for self-programming using the developments of artificial intelligence.

File

A collection of data, generally stored on cassette or disc.

File name

The name of a file. In Dr.Logo a file name can consist of up to 8 alphabetic or numeric characters. In CP/M an additional three character file type, preceded by a dot . is allowed.

Firmware

Software contained in ROM - a cross between pure software and pure hardware.

Fixed-point number

A number represented, manipulated and stored with the decimal point in a fixed specified position.

Floating-point number

A Real number, manipulated and stored with its decimal point permitted to settle in the required position. The method is particularly useful when dealing with large numbers.

Floppy disc

(See Disc)

Flowchart

A diagrammatic representation of the progression of program steps and logical processes tracing the sequence of events during program execution.

Forth

A high speed programming language, with speed and complexity falling between a High-level language and Machine code. Not a beginners' language.

Function key

A key on the keyboard that has been assigned a specific task - which it may execute in addition to, or instead of the main purpose inscribed upon that key.

Gate

Logical gates permit the passage of data when certain conditions are fulfilled. There are many different types of gate (OR, AND, XOR etc).

Graphics

The part of the screen display of the computer that is not related to the display of 'characters'. Graphics encompasses the drawing and plotting of lines, circles, patterns, etc.

Graphics character

A shape or pattern specially designed to be useful in creating images.

Graphics cursor

Similar to the text cursor, but addressing the graphics screen. An invisible concept on the CPC664 - but nevertheless an indispensable facility for locating drawn graphics. Not to be confused with graphics characters which are still part of the 'character set', and are printed at the text cursor.

Graphics mode

Early microcomputers required to be specifically set to either handle characters or graphics. Modern personal computers are capable of mingling text and graphics simultaneously.

Graphics tablet

A device that plots the co-ordinate points of a given picture or drawing for manipulation within the computer. A form of A/D.

Handshaking

A sequence of electronic signals which initiates, checks and synchronises the exchange of data between a computer and a peripheral, or between two computers.

Hard Copy

Paper print-out of a program or other text - or of a graphics display. The transitory screen equivalent is known as 'soft copy'.

Hardware

The electronic and mechanical parts of a computer system - anything that isn't software or firmware.

Hexadecimal (or HEX)

The number system with base 16. Hexadecimal numbers are signified in CPC664 programming by the prefix & or &H. e.g. &FF = (decimal) 255. (See part 1 of the chapter entitled 'At your leisure....')

Hex file

An ASCII representation of a command or machine code file.

High-level language

Languages like BASIC, which are written in 'near literal' form, where the actual language does most of the work of interpreting. Slower than machine code orientated programs, but far simpler to understand.

IEEE-488

One of the standard Interfaces for connecting devices to a microcomputer. Similar to - but not wholly compatible with - the Centronics parallel interface.

Information technology

Anything relating to the use of electronics in the processing of information and communications: wordprocessing, data communications, PRESTEL, etc.

Initialise

Switch on a system, or declare specific values for variables before beginning to execute the body of program - e.g. dimensioning arrays, declaring variables to be integers, etc.

Input

Anything that enters the computer memory from its keyboard, disc unit, serial interface or other input source.

Instruction

A request/command to a computer to perform a particular operation. A collection or sequence of instructions form a program.

Instruction set

The prime logical and mathematical processes carried out by the microprocessor. Every high level instruction (including assembler mnemonics) have to be capable of being distilled down to an instruction that is recognised by the computer CPU. A single high level command may invoke a large number of elements from the CPU's instruction set.

Integer

A whole number with no decimal point.

Integer number

A number with no fractional part, i.e. a number with no part to the right of the decimal point - as opposed to a real number which is the integer part plus the fractional part.

Integrated circuit

A collection of electronic circuit components miniaturised and built onto a single piece of silicon. (See also Chip)

Intelligent terminal

A terminal where as well as handling the requirements of the computer's input and output, local processing power is also available when the terminal is 'off line'.

Interactive

Usually a reference to programs where the hardware computer prompts the user to provide various types of input - ranging from controlling the spaceship in an arcade game, to answering questions in educational programs. The action of the user has an effect in 'real time' on the behaviour of the program.

Interface

The way in and out of a computer, both in electrical and human terms. The CPC664 interface is the keyboard (input), and the screen (output) - as well as the facility for the connection of user peripherals to the various sockets.

Interpreter

A further extension of the analogy of computer instruction sets and language. The element of the system software that interprets the high level language to the level that can be understood by the CPU. e.g It converts BASIC code as entered via the keyboard into the computer's own internal language.

I/O

Input/Output.

Iteration

One of the elements of computing. The computer performs all tasks by breaking them down into the simple tasks that can be handled by the CPU. To do this, the computer must go to and fro' between many simple elements until a given condition is fulfilled.

Joystick

An input device that generally replaces the function of the cursor keys, and makes games playing faster and easier.

K

A short form of the metric measure prefix for 1000, 'kilo' - which in computing has come to be widely used to refer to a 'kilobyte' - which is actually 1024 (decimal) in view of the binary association of 2 raised to the power of 10.

Keyboard

The matrix of alphanumeric key switches, arranged to provide the means of typing commands and other information into the computer.

Keyword

A word whose use in the computer program or language is reserved for a specific function or command.

Least significant bit

In a binary number, the Least Significant Bit (LSB) is the bit at the extreme right hand end of the expression.

Light Pen

Another alternative input method, using a pen or 'wand'.

Line number

BASIC and some other languages use programs that are arranged in line number order.

Lisp

The acronym formed from LISt Processor language. Another high level computer language.

Logic

The electronic components that carry out the elementary logical operations and functions, from which every operation of a computer is ultimately built up.

Logical device

The representation of a device that may be different to its physical form. For example the CP/M logical device LST may be assigned to the Centronics port or perhaps the VDU.

LOGO

The name of a programming language derived from the Greek word logos, which means word. Logo is designed to teach the fundamentals of computer programming.

Loop

A process in a program that is executed repeatedly by the computer until a certain condition is satisfied.

Low-level language

Such as 'assembly language'. A programming language in which each instruction corresponds to the computer's machine code instruction.

LSI

Large Scale Integration. The development of integrated circuits, packing more functions onto ever smaller pieces of silicon.

Machine Code

The programming language that is directly understood by a microprocessor, since all its commands are represented by patterns of binary digits.

Machine readable

A medium of data or any other information that can be immediately input to a computer without additional work on keyboarding etc.

Man-machine interface

A point of interaction between the computer and the operator: keyboard, screen, sound etc.

Matrix

The arrangement of the dots that form the character cell on the screen, or on the print head of a 'dot matrix' printer. Also a term used in mathematics and computer science to encompass arrays.

Memory

The computer's parking lot for information and data, neatly arranged in logical rows with each item individually accessible. Either known as RAM (random access memory) where information can be both stored and retrieved, or ROM, where the information may be read, but not re-written in another form. Discs and tape are examples of 'bulk memory', although the term has evolved to mean the memory that is directly addressed by the CPU.

Memory map

The layout of the memory, showing the various addresses, and the allocation of the memory to specific functions, such as the screen, the disc operating system etc.

Menu

A bill of fare of the different options that may be carried out by the program in the computer, left to the user to select.

Microprocessor

An integrated circuit that sits at the heart of a microcomputer and executes the instructions that are presented to it by the BASIC interpreter, in order to control the various output devices and options.

Modem

A MOdulator DEModulator that connects the computer's I/O to a telephone line or other serial data transmission medium - including fibre optics. (See also Acoustic coupler)

Monitor

The screen section of a computer terminal system, and also a term describing a machine language program that provides access to the fundamental machine operation of the computer.

Mouse

An upside-down 'tracker' or 'roller ball'. Pushed around a table top by hand, a mouse is generally used to move a cursor around the screen. Originally designed to overcome the fear of keyboards and make software appear more 'user friendly'.

MSB

The Most Significant Bit of a binary number, i.e. the bit at the left end of the binary expression.

Network

When two or more computers are linked together to exchange data and information - either by wiring, or via MODEMs.

Nibble

Half a byte: a four bit expression. Each of the hexadecimal digits in the expression &F6 represents 'one nibble'.

Node

A unit of storage in the LOGO workspace. Typically one node consumes 4 bytes of memory space.

Noise

The CPC664 sound facilities include the ability to inject a variable amount of random noise to create effects such as explosions.

Numeric keypad

The area on the keyboard where number keys are grouped to facilitate entry of numeric data, and in the case of the CPC664, to provide the additional facility of user definable function keys.

OCR

Optical Character Recognition. A means of reading printed or written characters with an optical reader and translating them directly into computer readable data.

Octal

A number system to the base of 8, where each digit (0-7) is constructed from three Bits.

Off line

A computer peripheral - usually a display terminal or a printer - that is not actively connected to, or accessible by, the main processing unit.

On line

The opposite of Offline.

Operating system

The attendant in the 'parking lot' referred to under the entry for Memory. Software that allocates precedence and timing to the operations of the computer.

Output

Anything that comes from a computer as the result of some computational function.

Operator

The part of arithmetic expression that causes one number to operate on another, e.g. + - * / etc.

Overwrite

Erase an area of memory by replacing its contents with new data.

Paddle

An alternative name for a joystick. Also referred to as a 'games paddle'.

Page zero

This refers to the region of memory in a CP/M environment between &0000 and &0100 which is used to hold vital system parameters.

Paperware

Another description for the printed 'hardcopy' of computing. Occasionally computers launched before they have actually finished development are described as a 'paperware exercise'.

Parallel interface

The CPC664 printer interface supports a parallel printer: which means that each data line from the bus is connected to a corresponding input on the printer. Data is transferred many times more quickly using a parallel interface than it is through a serial interface, since the serial interface must first format each byte, and frame it with synchronisation information.

Pascal

A high-level structured programming language that must be compiled before it will execute - and therefore runs very quickly. Generally the next language that the keen BASIC student will pursue.

PEEK

The BASIC function that looks directly into the computer's memory, and reports the value at the specified location.

Peripheral

Printers, modems, joysticks, cassette units - anything that plugs into the computer to expand its capabilities.

Physical device

An actual device, consisting of hardware, that exists. Physical devices may be represented by logical devices.

Pixel

The smallest accessible area of the screen that can be controlled by the hardware.

Plotter

A specific type of printer that draws 'longhand' using pens rather than an impact print head. Used for technical and graphical drawing output.

POKE

The statement in BASIC that is used to place a value in a specified memory location.

Port

A specifically addressable point on the interface for input or output of data.

Portability

Other than the literal use, means the ability for software to operate on different makes of computer - usually as a result of a compatible operating system, such as Digital Research's CP/M.

Primitives

Procedures, operations or commands that make up Dr.Logo; the built-in procedures.

Printer

Any hardcopy method for printing out text.

Procedure

A series of expressions or program statements that dictate how to perform a particular task.

Program

A combination of instructions that cause the computer to execute a task. It can be anything from a simple machine code 'routine' to a complete applications program, such as a wordprocessor.

Programming language

The medium through which the program is written, being comprised of rigid rules on the use of words, numbers and the sequence in which they are implemented.

PROM

Programmable Read Only Memory. An integrated memory circuit that once written with data, cannot be erased. (See also EPROM)

Prompt

A short message or character sequence reminding the user that some type of input is expected. For example, the CP/M prompt is the > and the Dr.Logo prompt is the ? character.

PSU

Power Supply Unit. The means of converting the domestic mains electricity supply into the necessary voltages to operate the computer (and peripheral devices).

QWERTY keyboard

The colloquial term to describe a keyboard with the conventional UK or US typewriter key layout.

RAM

Random Access Memory. Memory that may be both read from, and written to, using the internal circuitry of the computer during the normal course of program execution.

Random access

The ability to read and write information in memory or on a disc in any desired order.

Random number

A number that is generated by the computer program that is neither repeatable, not predictable. The CPC664 is capable of generating a pseudo random number sequence.

Raster

A system of 'writing' on the screen where the images are built from a number of horizontal scan lines. (Raster scan).

Read only R/O

An attribute assigned to a disc, a disc file or a disc drive that prevents writing or changing of data.

Read write R/W

An attribute assigned to a disc, a disc file or a disc drive that allows both reading and writing of data.

Real number

A number that has both integer and fractional parts. i.e. both sides of the decimal point are used.

Real time

Events that occur before your eyes, as opposed to those which only become evident after the termination of the process that produced them.

Record

A group of bytes in a file. CP/M uses 128 byte records.

Recursion

The series of repeated steps (also sometimes imprecisely described as reciprocation) within a program or routine in which the result of each repeated cycle of events is related to the previous one.

Refresh

To update information, either on the screen of a VDU, or in the memory. Need not be a destructive process, but merely reinforcing whatever was already present in memory or on the screen.

Register

A transient memory location within the CPU that is used for temporary storage.

Remark

A non-executing statement in a program that is included to remind the programmer what part of the program he is doing, and to date and time stamp that particular 'edition'.

Reserved word

A word which has particular significance to the computer program, and cannot be used other than in its pre-defined context. For example, BASIC will not accept the word NEW as a variable - it is already 'reserved' for another purpose.

Resolution

The ability to determine where one element of the display ends, and the next one begins. Also loosely applied to the ability of a computer to perform arithmetic manipulations on large numbers.

Reverse Polish notation

(RPN) A method of describing arithmetic operations favoured by some calculator manufacturers, where the operators (+, -, *, /) are placed behind the values to which they apply.

RF Modulator

The means by which the video signals from the computer are encoded and 'transmitted' to the aerial of a standard TV set.

ROM

Read Only Memory. Generally with reference to semiconductor memory systems: once written, neither erasable, nor re-writable.

Routine

A part of the program that performs a 'routine' task. A 'sub' program that resides either within a main program, or may exist as a separate module for incorporation into a variety of applications programs. e.g. A program to derive a 12 hour display from the system's clock may be considered as a routine.

RS232C

A specific standard for serial data communications interfaces. Devices at both ends of the data link using an RS232 interface require to be 'configured' to the particular conditions of the RS232 data standard used. Compare this with the Centronics parallel interface where the interconnection is virtually standard everywhere.

Screen Editor

A text or program editor where the cursor may be taken to any part of the screen display in order to alter the characters appearing there.

Scrolling

The term describing the way in which the screen display 'rolls up' when the display fills up to the bottom, and needs to make space for the next line of entry or output to appear.

Sector

A block of data on a disc. The AMSTRAD disc system uses a sector size of 512 bytes.

Separator

A separator performs the same function as a delimiter, i.e. marking the boundary between reserved words and other elements of the program or data.

Serial interface

Although this term nearly always refers to an RS232 interface, other serial standards exist for the sequential transmission of computer data.

Simulation

A technique for emulation of real life interactive processes using the computer, such as flight simulation, driving simulation etc.

Single sided

Refers to a disc which has only one side available for data storage.

Soft key

(See UDK - user defined key)

Software

Programs themselves. May be based on disc, cassette, ROM, etc.

Software engineering

A grandiose expression meaning computer programming, implying a structured and considered approach, as opposed to arbitrary techniques.

Sound generator

The part of a computer (it may be either hardware or software) that creates the sound and noise.

Speech recognition

The conversion of the spoken word into machine readable instructions.

Speech synthesis

Generation of simulated speech using a combination of hardware and software.

Spreadsheet

A program that allows rows and columns of numbers to be entered and arithmetically manipulated. Changing one entry causes all the associated calculations to be re-run, and produces an updated result.

Sprite

A screen character that moves freely around the display, generated by specific hardware or software that allows it to appear and disappear apparently at random.

Stack

An area of memory allocated for 'stacking' information, but where only the last entry on the stack can be recalled.

Statement

An instruction, or sequence of instructions, in a computer program.

Stream

The route used for the output from the computer. e.g. the screen, the printer, or the disc.

String

A type of data comprising an assortment of characters that may not be treated as a numeric variable. It may be purely numeric, but it is not treated as such unless specifically converted to a corresponding numeric variable by the appropriate command.

Structured programming

A logical and premeditated programming technique that results in programs that flow from 'top to bottom', with clearly described steps.

Sub-routine

(See routine)

Syntax error

When the rules of the program are broken by the incorrect use of keywords and variables, BASIC will prompt the user with this message.

System tracks

Tracks reserved on the disc for the CP/M system.

Terminal

A keyboard input device, with either a VDU screen or teletype typewriter output system.

TPA

Transient Program Area. An area in memory commencing at &0100 where CP/M user programs run and store data.

Track

Tracks are concentric rings on a disc. Each track holds a fixed number of sectors. The tracks and sectors are written to a specific area of a disc during formatting.

Transient program

A CP/M utility program such as FILECOPY, which can be loaded into the TPA and run by typing its name at the keyboard.

Truncated

A number or string that has been shortened by removal of leading or trailing characters. Where intentional, the process may involve rounding the value. Where unintentional, the extra characters are simply discarded to enable the number or string to fit the available space.

Truth table

The results of a logical operation are either 'true' or 'false'. The computer interprets these as being either 1 or 0, and the truth table lists the possible results of a logical operation (IF A > B THEN C) accordingly.

Turnkey

A word used to describe a program which executes automatically when the system is booted. The Dr.Logo disc is an example of a turnkey program.

Turtle

A graphic symbol, in the shape of an arrow head, that functions as a graphic cursor on the Dr.Logo graphic screen.

Turtle graphics

The graphics image left on the screen by the movement of a turtle. As the turtle moves it leaves a trace of its path on the screen.

Turtle step

The smallest distance a turtle can move. Normally one pixel.

UDK

User defined keys. The CPC664 has up to 32 keys which may be redefined to perform a variety of tasks.

Unsigned number

A number with no prefix to signify whether its value is positive or negative.

Utility

Any complete program used to perform a common operation, such as sorting data or copying files.

Utility program

A program on disc that enables the user to perform certain operations. (See Transient program)

Variable

An item included in a computer program that can be identified by name, but whose actual value may be made to vary during the execution of a program.

Warm start

This is performed when [CTRL]C is pressed during CP/M. A warm start reinitialises the disc sub-system and returns control to CP/M ready for commands to be entered.

Wildcard character

Either of the characters * or ?. Dr.Logo only supports the ? character. The * wildcard character simply means any number of ?s. When referencing files wildcard characters are used to make up an ambiguous file name. Any ?s in the file name refer to any alphabetic or numeric character.

Write protection

A safeguard used to prevent re-writing of a disc or disc file. A write protected disc or file is Read Only.

XYZY

A magic word to get out of sticky corners in adventure games.

Appendix 3

Some Programs For You....

Telly tennis

The one that started it all, but still great fun! For two players, or one player against the computer. Keyboard or Joystick(s).

```
10 'TELLY TENNIS by DAVID RADISIC
20 'copyright (c) AMSOFT 1985
30 '
40 DEFINT a-z
50 comp=1
60 ENV 1,=11,20,=9,5000
70 MODE 1:INK 0,10:BORDER 10:INK 1,26:INK 2,18:INK 3,0
80 GOSUB 710
90 GOSUB 150
100 GOSUB 330
110 GOSUB 420
120 LOCATE 13,1:PRINT USING"#### ";score1;
130 LOCATE 35,1:PRINT USING"#### ";score2;
140 GOTO 100
150 PEN 2
160 x(1)=3:y(1)=5
170 x(2)=37:y(2)=22
180 edge$=CHR$(233):edge2$=STRING$(2,207)
190 LOCATE 1,3:PRINT STRING$(39,edge$):PRINT STRING$(39
,edge$)
200 FOR i=1 TO 19
210 PRINT edge2$;TAB(38);edge2$
220 NEXT
230 PRINT STRING$(39,edge$):PRINT STRING$(39,edge$);
240 WINDOW #1,3,37,5,23
250 CLS#1
260 SYMBOL 240,0,60,126,126,126,126,60,0
270 bat$="I"+CHR$(8)+CHR$(10)+"I"
280 clr$=" "+CHR$(8)+CHR$(10)+" "
290 ball$=CHR$(240)
300 PEN 3
310 LOCATE 2,1:PRINT"Player 1 :    0";:LOCATE,24,1:PRIN
T"Player 2 :    0";
320 RETURN
```

continued on the next page

```

330 n=INT(RND*2):CLS #1:scored=0
340 PEN 3
350 FOR i=1 TO 2:LOCATE x(i),y(i):PRINT bat$;:NEXT
360 ON n GOTO 390
370 xb=21:dx=1
380 GOTO 400
390 xb=19:dx=-1
400 yb=12:dy=INT(RND*3)-1
410 RETURN
420 GOSUB 600
430 oxb=xb:oyb=yb
440 GOSUB 500
450 IF note>0 THEN SOUND 129,note,50,15,1
460 LOCATE oxb,oyb:PRINT " ";
470 LOCATE xb,yb:PRINT ball$
480 IF scored=0 THEN 420
490 RETURN
500 LOCATE xb+dx,yb+dy:ch$=COPYCHR$(#0)
510 note=0
520 IF ch$=" " THEN xb=xb+dx:yb=yb+dy:RETURN
530 IF ch$="I" THEN dx=2-dx-2:dy=INT(RND*3)-1:note=200:
RETURN
540 IF ch$=LEFT$(edge$,1) THEN 570
550 IF ch$=edge$ THEN dy=2-dy-2:note=250
560 RETURN
570 IF dx>0 THEN score1=score1+1 ELSE score2=score2+1
580 scored=1:note=2000
590 RETURN
600 p(1)=(INKEY(69)>=0)+(INKEY(72)>=0)+ABS((INKEY(71)>=
0)+(INKEY(73)>=0))*2
610 IF comp=1 THEN p(2)=ABS(y(2)<yb)*2+(y(2)>yb):GOTO 6
30
620 p(2)=(INKEY(4)>=0)+(INKEY(48)>=0)+ABS((INKEY(5)>=0)
+(INKEY(49)>=0))*2
630 PEN 3
640 FOR i=1 TO 2
650 LOCATE x(i),y(i)+p(i):ch$=COPYCHR$(#0)
660 IF ch$=" " THEN LOCATE x(i),y(i):PRINT clr$;:y(i)=y
(i)+ROUND(p(i)/2)
670 LOCATE x(i),y(i):PRINT bat$;
680 NEXT
690 PEN 1
700 RETURN
710 PEN 2:PRINT:PRINT TAB(15)"Ping-Pong":PRINT TAB(15)"
-----"

```

continued on the next page

```

720 PEN 3:PRINT:PRINT TAB(14)"To move bats :
730 PRINT:PRINT:PEN 1
740 PRINT" Player 1          Player 2          Direction":PRI
    NT
750 PRINT"      A              6              UP"
760 PRINT"      Z              3              DOWN":PRINT
770 PEN 3:PRINT:PRINT TAB(14)"Or joysticks"
780 PRINT:PRINT:PRINT:PRINT
790 PEN 2
800 PRINT TAB(6)"Select <1> or <2> players"
810 i$=INKEY$:IF i$<>"1" AND i$<>"2" THEN 810
820 IF i$="1" THEN comp=1 ELSE comp=0
830 MODE 1:RETURN

```

Electric fencing

Try to 'foil' your opponent! For two players only. Keyboard or Joysticks.

```

10 'ELECTRIC FENCING by ALEXANDER MARTIN
20 'copyright (c) AMSOFT 1985
30 '
40 DEFINT a-z
50 MODE 0
60 GOSUB 980
70 GOSUB 1370
80 GOSUB 270
90 GOSUB 1520
100 GOSUB 1370
110 GOSUB 1270
120 '
130 '
140 REM start
150 IF finished THEN GOTO 100
160 GOSUB 240
170 FRAME:IF p1dir THEN GOSUB 570 ELSE FRAME:FRAME
180 FRAME:IF p2dir THEN GOSUB 620 ELSE FRAME:FRAME
190 IF p1sa=-1 THEN GOSUB 670
200 IF p2sa=-1 THEN GOSUB 720
210 GOTO 140
220 '
230 '

```

continued on the next page

```

240 IF j THEN 380 ELSE 480
250 '
260 '
270 CLS:PEN 6
280 PRINT:PRINT"  CHOOSE CONTROL"
290 PRINT:PRINT:PRINT:PRINT"press J/K then ENTER"
300 LOCATE 4,10:PRINT"JOYSTICK";TAB(5);"OR KEYS"
310 LOCATE 12,10:IF j THEN PRINT"*":ELSE PRINT" "
320 LOCATE 12,11:IF j THEN PRINT" ":ELSE PRINT"*"
330 IF NOT(INKEY(45)) THEN j=-1
340 IF NOT(INKEY(37)) THEN j=0
350 IF NOT(INKEY(18)) THEN RETURN ELSE 310
360 '
370 '
380 p1=JOY(0):p2=JOY(1)
390 p1dir=(p1 AND 1)*-1+(p1 AND 2)*0.5
400 p2dir=(p2 AND 1)*-1+(p2 AND 2)*0.5
410 IF P1 AND 16 THEN p1sa=p1sa-1:IF p1sa=-1 THEN AFTER
    15 GOSUB 770
420 IF P2 AND 16 THEN p2sa=p2sa-1:IF p2sa=-1 THEN AFTER
    15 GOSUB 770
430 IF p1sa THEN p1dir=0
440 IF p2sa THEN p2dir=0
450 RETURN
460 '
470 '
480 p2dir=((INKEY(4)=0)*1)+((INKEY(5)=0)*-1)
490 p1dir=((INKEY(69)=0)*1)+((INKEY(71)=0)*-1)
500 IF INKEY(63)=0 THEN p1sa=p1sa-1:IF p1sa=-1 THEN AFT
    ER 15 GOSUB 770
510 IF INKEY(10)=0 THEN p2sa=p2sa-1:IF p2sa=-1 THEN AFT
    ER 15 GOSUB 770
520 IF p1sa THEN p1dir=0
530 IF p2sa THEN p2dir=0
540 RETURN
550 '
560 '
570 pt=p1wp+p1dir:IF pt>25 OR pt<6 THEN RETURN ELSE p1w
    p=pt
580 p1dir=0
590 PEN 1:LOCATE 3,p1wp:CLS #3:PRINT CHR$(209);:RETURN
600 '
610 '
620 pt=p2wp+p2dir:IF pt>25 OR pt<6 THEN RETURN ELSE p2w
    p=pt

```

continued on the next page

```

630 p2dir=0
640 PEN 2:LOCATE 18,p2wp:CLS #5:PRINT CHR$(211);:RETURN
650 '
660 '
670 PAPER #4,4:WINDOW #4,4,17,p1wp,p1wp:CLS#4:FRAME:FRA
    ME
680 PAPER #4,0:CLS#4
690 GOTO 570
700 '
710 '
720 PAPER #6,5:WINDOW #6,4,17,p2wp,p2wp:CLS#6:FRAME:FRA
    ME
730 PAPER #6,0:CLS#6
740 GOTO 620
750 '
760 '
770 pwpe=(p1wp=p2wp):IF p1sa AND NOT(p2sa) AND pwpe THE
    N p1sc=p1sc+1:SOUND 132,120,10,0,1,0:PRINT#1,a$(p1s
    c);:IF p1sc=9 THEN 860
780 IF p2sa AND NOT(p1sa) AND pwpe THEN p2sc=p2sc+1:SOU
    ND 132,100,10,0,1,0:PRINT#2,a$(p2sc);:IF p2sc=9 THE
    N 860
790 IF p1sa THEN SOUND 132,40,70,0,1,1
800 IF p2sa THEN SOUND 132,56,70,0,1,1
810 p1sa=0
820 p2sa=0
830 RETURN
840 '
850 '
860 PEN 6
870 LOCATE 6,10:PRINT"GAME OVER"
880 IF p1sc=9 THEN INK 1,2,20:INK 2,0 ELSE INK 2,6,17:I
    NK 1,0
890 SOUND 129,1000,0,12,3:SOUND 130,900,0,12,3
900 WHILE INKEY$<>"":WEND
910 t!=TIME:WHILE t!+2000>TIME:WEND
920 WHILE INKEY$="":WEND
930 CLS
940 finished=-1
950 RETURN
960 '
970 '

```

continued on the next page

```

980 a$(0)="111101101101111"
990 a$(1)="001001001001001"
1000 a$(2)="111001111100111"
1010 a$(3)="111001111001111"
1020 a$(4)="100100101111001"
1030 a$(5)="111100111001111"
1040 a$(6)="111100111101111"
1050 a$(7)="111001001010010"
1060 a$(8)="111101111101111"
1070 a$(9)="111101111001001"
1080 FOR n=0 TO 9
1090 howlong=LEN(a$(n))
1100 FOR n2=1 TO howlong
1110 IF MID$(a$(n),n2,1)="1"THEN MID$(a$(n),n2,1)=CHR$(
143)ELSE MID$(a$(n),n2,1)=CHR$(32)
1120 NEXT n2,n
1130 '
1140 '
1150 b$="ELECTRIC FENCING"
1160 c$=CHR$(32)+CHR$(164)+" Alexander Martin"
1170 ENV 1,=9,2000:ENT -1,6,3,1
1180 ENV 2,127,0,0,127,0,0,127,0,0,127,0,0,127,0,0
1190 ENV 3,=9,9000
1200 '
1210 '
1220 BORDER 0
1230 INK 0,12:PEN #4,1:PEN #6,2:PEN #1,1:PEN #2,2:PAPER
#1,3:PAPER #2,3:PEN #0,6
1240 RETURN 'FROM SETTING UP CONSTANTS
1250 '
1260 '
1270 INK 0,12:INK 1,2:INK 2,6:INK 3,13:INK 4,20:INK 5,1
7:INK 6,20
1280 WINDOW #3,3,3,6,25:WINDOW #5,18,18,6,25
1290 WINDOW #1,3,5,1,5:WINDOW #2,16,18,1,5:WINDOW #7,1,
20,1,5:PAPER #7,3
1300 CLS:CLS#7:PRINT#1,a$(0);:PRINT#2,a$(0);:p1sc=0:p2s
c=0:p1wp=5:p2wp=24:p1dir=1:p2dir=1
1310 GOSUB 570:GOSUB 620
1320 SOUND 1,1000,0,12,2:SOUND 2,900,0,12,2
1330 p1sa=0:p2sa=0:finished=0
1340 RETURN 'FROM GAME SHEET RESTORE
1350 '
1360 '
1370 CLS

```

continued on the next page

```

1380 PEN 7
1390 FOR n=1 TO LEN(b$)
1400 LOCATE 2+n,10
1410 FOR n2=LEN(b$) TO n STEP-1
1420 PRINT MID$(b$,n2,1)
1430 LOCATE 2+n,10
1440 SOUND 135,20*n2,5,12,2,1
1450 NEXT n2,n
1460 SOUND 135,100,0,13,3,1,20
1470 PEN 6:PRINT:PRINT:PRINT:PRINT c$
1480 FOR n=1 TO 5000:NEXT
1490 RETURN
1500 '
1510 '
1520 IF j THEN RETURN
1530 CLS
1540 LOCATE 7,5
1550 PRINT"CONTROLS"
1560 PRINT
1570 PRINT"  PLAYER1  PLAYER2"
1580 PRINT
1590 PRINT"  A      up      6"
1600 PRINT"  Z      down    3"
1610 PRINT"  X      fire     7"
1620 t!=TIME:WHILE t!+1000>TIME:WEND
1630 RETURN

```

Pontoon

Brings out the gambler in you! For one player against the computer. Keyboard only.

```

10 'PONT00N
20 'copyright (c) AMSOFT 1984
30 '
40 INK 0,21:INK 1,0:INK 3,6
50 ENV 1,7,2,1
60 MODE 1
70 yc=2:cc=2
80 aces=0
90 caces=0
100 laces=0

```

continued on the next page

```

110 s=0
120 t=0
130 DIM suit$(4)
140 suit$(1)=CHR$(226)
150 suit$(2)=CHR$(229)
160 suit$(3)=CHR$(228)
170 suit$(4)=CHR$(227)
180 CLS
190 DIM pack(52)
200 FOR x= 1 TO 52
210 pack(x)=0
220 NEXT x
230 '
240 'Deal two cards to each player
250 '
260 LOCATE 8,3
270 PRINT "PLAYER";SPC(15)"HOUSE"
280 LOCATE 7,5
290 GOSUB 930
300 s=s+f
310 IF f=11 THEN aces=aces+1
320 LOCATE 27,5
330 GOSUB 930
340 t=t+f
350 IF f=11 THEN caces=caces+1
360 LOCATE 7,6
370 GOSUB 930
380 s=s+f
390 IF s=21 THEN 850
400 IF f=11 THEN aces=aces+1
410 IF s=22 THEN aces=aces-1
420 IF s=22 THEN s=12
430 '
440 'Input option-twist(t) or stick(s)
450 '
460 LOCATE 10,12: PRINT "TWIST (t) OR STICK (s)":FOR z=
1 TO 500:NEXT:LOCATE 10,12:PRINT SPC(25):FOR z=1 TO
500:NEXT
470 x$=LOWER$(INKEY$):IF x$<>"s" AND x$<>"t" THEN 460
480 IF x$="s" THEN LOCATE 10,12:PRINT:GOTO 680
490 LOCATE 7,yc+5
500 yc=yc+1
510 GOSUB 930
520 s=s+f

```

continued on the next page

```

530 IF f=11 THEN aces=aces+1
540 '
550 'Check score and aces
560 '
570 IF s<22 THEN 460
580 IF aces=0 THEN 620
590 aces=aces-1
600 s=s-10
610 GOTO 570
620 LOCATE 12,19
630 PRINT "YOU'RE BUST"
640 PRINT:LOCATE 12,21:PRINT"ANOTHER GAME (Y/N)"
650 x$=LOWER$(INKEY$):IF x$<>"y" AND x$<>"n" THEN 650
660 IF x$="y" THEN RUN
670 END
680 LOCATE 27,6
690 GOSUB 930
700 t=t+f
710 IF f=11 THEN caces=caces+1
720 IF t=17 OR t=18 OR t=19 OR t=20 OR t=21 THEN GOTO 8
80
730 cc=cc+1
740 LOCATE 27,cc+4
750 GOSUB 930
760 t=t+f
770 IF f=11 THEN caces=caces+1
780 IF t>6 AND t<12 AND laces>0 THEN t=t+10:GOTO 880
790 IF t<21 THEN 720
800 IF t=21 THEN 880
810 IF caces=0 THEN 850
820 caces=caces-1:laces=laces+1
830 t=t-10
840 GOTO 790
850 LOCATE 12,19
860 PRINT "YOU WIN"
870 GOTO 640
880 LOCATE 12,19
890 IF t<s THEN 850
900 PRINT "THE HOUSE WINS"
910 GOTO 640
920 '
930 'Deal card
940 '
950 card=INT(RND(1)*52+1)

```

continued on the next page

```

960 IF pack(card)=1 THEN GOTO 950
970 pack(card)=1
980 f=card-13*INT(card/13)
990 IF f=0 THEN f=13
1000 IF f=1 OR f>10 THEN GOTO 1090
1010 FOR z=1 TO 1000:z=z+1:NEXT
1020 IF f=10 THEN PRINT f;" ";
1030 IF f<10 THEN PRINT " "f" ";
1040 IF f>10 THEN f=10
1050 IF card>26 THEN PEN 3 ELSE PEN 1
1060 SOUND 1,0,0,0,1,,1:PRINT suit$(INT((card-1)/13)+1)
1070 PEN 1
1080 RETURN
1090 FOR z=1 TO 1000:z=z+1:NEXT
1100 IF f=11 THEN PRINT " J ";
1110 IF f=12 THEN PRINT " Q ";
1120 IF f=13 THEN PRINT " K ";
1130 IF f<>1 THEN GOTO 1040
1140 f=11
1150 PRINT " A ";
1160 GOTO 1060

```

Bomber

A variation on a classic theme! For one player against the computer. Keyboard or Joystick.

```

10 'BOMBER by DAVE TOWN
20 'copyright (c) AMSOFT 1984
30 '
40 MODE 1:CLS:INK 0,0:BORDER 0:INK 1,18:INK 2,6:INK 3,4
   :INK 5,15:INK 6,2:INK 7,24:INK 8,8:INK 9,26:INK 10,1
   :INK 11,20:INK 12,12:INK 13,16:INK 14,14:INK 15,21
50 SYMBOL AFTER 240:SYMBOL 241,&40,&60,&70,&7F,&7F,&3F,
   &7,&0:SYMBOL 242,&0,&32,&7A,&FE,&FA,&F2,&E0,&0
60 score=0:hiscore=0:plane$=CHR$(241)+CHR$(242):x=2:y=2
   :drop=0:a=2:b=2
70 GOSUB 480
80 CLS
90 PEN 2:LOCATE 1,15:INPUT"Enter skill : 0 (ACE) to 5 (
   NOVICE) : ",skill

```

continued on the next page

```

100 IF skill<0 OR skill>5 GOTO 90
110 skill=skill+10
120 LOCATE 1,15:PRINT CHR$(18);:LOCATE 1,15:INPUT"Enter
    speed 0 (FAST) to 100 (SLOW) : ",rate
130 IF rate>100 OR rate<0 GOTO 120
140 '
150 'Buildings
160 '
170 MODE 0:FOR base=5 TO 15:FOR height=21 TO INT(RND(1)
    *8+skill) STEP -1:LOCATE base,height:PEN base-2:PRI
    NT CHR$(143)+CHR$(8)+CHR$(11)+CHR$(244);:NEXT :NEXT
180 PLOT 0,20,4:DRAW 640,20,4
190 LOCATE 1,25:PEN 2:PRINT"SCORE";score;:LOCATE 13,25:
    PRINT"HI";hiscore;
200 '
210 'Main Game
220 '
230 LOCATE x-1,y:PRINT" ";
240 PEN 1:LOCATE x,y:PRINT plane$;:PEN 2
250 IF y=21 AND x=15 THEN GOTO 290:ELSE GOTO 340
260 '
270 'Landed
280 '
290 FOR c=0 TO 1000:NEXT
300 score=score+100-(skill*2):skill=skill-1:x=2:y=2:a=2
    :b=2:drop=0
310 IF skill<10 THEN skill=10:rate=rate-20
320 IF rate<0 THEN rate=0
330 GOTO 150
340 FOR c=0 TO rate:NEXT
350 x=x+1
360 IF x=18 THEN LOCATE x-1,y:PRINT CHR$(18);:x=2:y=y+1
    :LOCATE x,y:PEN 1:PRINT plane$;:PEN 2
370 a$=INKEY$:IF a$=" " AND drop=0 THEN drop=1:b=y+2:a=
    x
380 IF y=21 THEN drop=0
390 IF drop=1 THEN LOCATE a,b:PRINT CHR$(252);:LOCATE a
    ,b-1:PRINT" ";:b=b+1:IF b>21 THEN LOCATE a,b:PRINT"
    ";:LOCATE a,b-1:PRINT" ";:a=0:b=0:drop=0:SOUND 3,4
    000,10,12,0,0,10
400 ga=(a-0.5)*32:gb=400-(b*16):bomb=TEST(ga,gb)
410 IF bomb>0 THEN GOTO 670
420 gx=((x+1.5)*32):gy=408-(y*16):crash=TEST(gx,gy)
430 IF crash>0 GOTO 570

```

continued on the next page

```

440 GOTO 230
450 '
460 'Instructions
470 '
480 LOCATE 1,2:PEN 1:PRINT"You are piloting an aircraft
    over a des-erted city and must clear the buildings
    in order to land and refuel. Your air- craft move
    s across the screen from left to right.":PRINT
490 PRINT:PRINT"On reaching the right, the aircraft
    returns to the left A LINE FURTHER DOWN.You have a
    n unlimited supply of bombs and you can drop them
    on the buildings below by pressing the SPACE BAR.
    ";:PRINT
500 PRINT:PRINT"Each time you land, the height of the
    buildings or the speed of your aircraft increases.
    ";:PRINT:PRINT:PRINT"ONCE YOU HAVE RELEASED A BOMB,
    YOU WILL NOT BE ABLE TO RELEASE ANOTHER UNTIL THE
    FIRST HAS EXPLODED !!!!";
510 PEN 2:LOCATE 1,24:PRINT:PRINT"Press any key to star
    t.";
520 a$=INKEY$:IF a$="" GOTO 520
530 RETURN
540 '
550 'Collision
560 '
570 LOCATE x-1,y:PRINT CHR$(32)+CHR$(32)+CHR$(32)+CHR$(
    253)+CHR$(8)+CHR$(238)+CHR$(8);
580 FOR t=1 TO 10:SOUND 7,4000,5,15,0,0,5:PEN t:PRINT C
    HR$(253)+CHR$(8)+CHR$(238)+CHR$(8)+CHR$(32)+CHR$(8)
    ;:FOR tm=0 TO 50:NEXT:NEXT:PEN 2
590 CLS:LOCATE 1,5:PRINT"You scored";score;
600 IF score>hiscore THEN hiscore=score:LOCATE 1,8:PRIN
    T"TOP SCORE!!";
610 score=0:LOCATE 1,12:PRINT"Press R to restart";
620 a$=INKEY$:IF a$="r" OR a$="R" GOTO 630 ELSE GOTO 62
    0
630 PEN 1:MODE 1:x=2:y=2:a=2:b=2:GOTO 90
640 '
650 'Bombed building
660 '
670 LOCATE a,b-1:PRINT" "+CHR$(8);:PEN 4:FOR tr=1 TO IN
    T(RND(1)*3)+1:score=score+5:SOUND 3,4000,10,12,0,0,
    10:LOCATE a,b:FOR t=0 TO 4:PRINT CHR$(253)+CHR$(8)+
    CHR$(32)+CHR$(8);:NEXT:b=b+1
680 IF b=24 THEN b=b-1
690 NEXT
700 LOCATE 6,25:PRINT score;:drop=0:a=x:b=y:GOTO 230

```

Amthello

The thinking person's game. Try to surround and capture your opponent's squares without leaving your own squares open to capture! For one player against the computer. Keyboard only.

```
10 'AMTHELLO by M.J.GRIBBINS
20 'copyright (c) AMSOFT 1984
30 '
40 BORDER 14
50 CLEAR
60 MODE 1:PEN 0:PAPER 1:CLS
70 INK 0,0:INK 1,14:INK 2,18:INK 3,26
80 LOCATE 2,3:PEN 3:PRINT"A":LOCATE 3,4:PRINT"M":LOCATE
  4,5:PRINT"T":LOCATE 5,6:PRINT"H"
90 LOCATE 6,7:PRINT"E":LOCATE 7,8:PRINT"L":LOCATE 8,9:P
  RINT"L":LOCATE 9,10:PRINT"O"
100 WINDOW #1,2,39,22,25:PAPER #1,1:PEN #1,0:CLS #1
110 PEN 0
120 LOCATE #1,8,1:PRINT #1,"BLACK ALWAYS PLAYS FIRST"
130 LOCATE #1,1,3:PRINT #1,"PRESS B OR W TO CHOOSE BLAC
  K OR WHITE"
140 B$=INKEY$:IF B$="" THEN 140
150 IF B$="W" OR B$="w" THEN Q%=3:N%=0:GOTO 210
160 IF B$="B" OR B$="b" THEN Q%=0:N%=3:GOTO 210
170 CLS #1:LOCATE #1,4,3
180 PRINT #1,"      BLACK OR WHITE ONLY"
190 FOR T=0 TO 1000:NEXT T
200 GOTO 140
210 DIM C%(10,10),P%(9,9),C1%(8),C2%(8),CX%(9),CY%(9)
220 I1%=2:J1%=2:I2%=7:J2%=7
230 FOR I%=0 TO 9
240 C%(I%,0)=6:C%(0,I%)=6
250 C%(9,I%)=6:C%(I%,9)=6
260 NEXT I%
270 FOR I%=1 TO 8
280 READ C1%(I%),C2%(I%)
290 FOR J%= 1 TO 8
300 READ P%(I%,J%)
310 C%(I%,J%)=6
320 NEXT J%:NEXT I%
330 C%(4,4)=3:C%(4,5)=0:C%(5,4)=0:C%(5,5)=3
```

continued on the next page

```

340 FOR K%=1 TO 58
350 READ AR%,BR%,CR%,DR%
360 PLOT AR%,BR%:DRAW CR%,DR%,0
370 NEXT K%
380 GOSUB 1460
390 IF Q%=3 GOTO 770
400 CLS #1:INPUT #1," WHICH LINE DO YOU WANT ";E%
410 IF E% <1 OR E% >8 GOTO 400
420 LOCATE #1,1,3:INPUT #1,"WHICH COLUMN DO YOU WANT ";
D%
430 IF D% <1 OR D% >8 GOTO 420
440 IF C%(D%,E%)=6 GOTO 480
450 CLS #1:LOCATE #1,5,2:PRINT #1,"THAT SQUARE IS ALREA
DY OCCUPIED !"
460 FOR T=1 TO 1000:NEXT T
470 GOTO 400
480 PLOT 270+(30*D%),70+(30*E%):DRAW 290+(30*D%),89+(30
+E%),Q%
490 PLOT 290+(30*D%),70+(30*E%):DRAW 270+(30*D%),89+(30
+E%),Q%
500 GOTO 540
510 FOR M%= 0 TO 19 STEP 2:PLOT 270+(30*D%),70+M%+(30*E
%)
520 DRAW 290+(30*D%),70+M%+(30*E%),6:NEXT M%
530 GOTO 400
540 VRX%=0
550 FOR K%=1 TO 8
560 VR%=0:C3%=D%:C4%=E%
570 C3%=C3%+C1%(K%):C4%=C4%+C2%(K%)
580 IF C%(C3%,C4%)=N% GOTO 590 ELSE 600
590 VR%=VR%+1:GOTO 570
600 IF C%(C3%,C4%)=6 GOTO 610 ELSE 620
610 NEXT K%:GOTO 670
620 IF VR%=0 GOTO 610 ELSE 630
630 VRX%=VRX%+VR%
640 C3%=C3%-C1%(K%):C4%=C4%-C2%(K%)
650 IF C%(C3%,C4%)=6 GOTO 610 ELSE 660
660 C%(C3%,C4%)=Q%:GOTO 640
670 IF VRX%=0 GOTO 680 ELSE 710
680 CLS #1:PRINT #1," THIS IS NOT A POSSIBLE CHOICE"
690 FOR T=1 TO 1000:NEXT T
700 GOTO 510
710 E%=E%:D%=D%:VRX%=VRX%
720 CLS #1:PRINT #1,"YOU HAVE PLAYED LINE NUMBER ";E%

```

continued on the next page

```

730 PRINT #1,"                AND COLUMN NUMBER ";D%
740 LOCATE #1,2,4:PRINT #1,"THAT GIVES YOU ";VRX%;" SQU
    ARES(S)"
750 C%(D%,E%)=Q%:GOSUB 1710
760 GOSUB 1460
770 CLS #1:LOCATE #1,10,2:PRINT #1,"NOW IT'S MY TURN ..
    .!"
780 P%=0:VRX%=0:VRY%=0
790 IF I1%*J1%=1 AND I2%*J2%=64 GOTO 860
800 FOR K%=2 TO 7
810 IF C%(2,K%) <> 6 THEN I1%=1
820 IF C%(7,K%) <> 6 THEN I2%=8
830 IF C%(K%,2) <> 6 THEN J1%=1
840 IF C%(K%,7) <> 6 THEN J2%=8
850 NEXT K%
860 FOR I%=I1% TO I2%
870 FOR J%=J1% TO J2%
880 IF C%(I%,J%)=6 GOTO 1030
890 NEXT J%:NEXT I%
900 IF P% > 0 GOTO 1000
910 IF PAS%=1 GOTO 920 ELSE 940
920 CLS #1:PRINT #1," DEADLOCK ! I MUST PASS ALSO.GAME
    OVER"
930 FOR T=1 TO 1000:NEXT T:GOTO 1550
940 CLS #1:LOCATE #1,18,2:PRINT #1,"I MUST PASS"
950 GOSUB 2720
960 IF PAS%=1 GOTO 970 ELSE 990
970 CLS #1:PRINT #1,"DEADLOCK! YOU MUST PASS ALSO.GAME
    OVER"
980 FOR T=1 TO 1000:NEXT T:GOTO 1550
990 GOTO 400
1000 IF LC%=0 THEN LC%=1:RANDOMIZE LC%:RL%=RND(LC%)
1010 CX1%=CX%(RL%):CX2%=CY%(RL%)
1020 GOTO 1220
1030 VRX%=0
1040 FOR K%=1 TO 8
1050 VR%=0:C3%=I%:C4%=J%
1060 C3%=C3%+C1%(K%):C4%=C4%+C2%(K%)
1070 IF C%(C3%,C4%)=Q% GOTO 1080 ELSE 1090
1080 VR%=VR%+1:GOTO 1060
1090 IF C%(C3%,C4%)=6 GOTO 1100 ELSE 1110
1100 NEXT K%:GOTO 1130
1110 IF VR%=0% GOTO 1100 ELSE 1120
1120 VRX%=VRX%+VR%:GOTO 1100

```

continued on the next page

```

1130 IF VRX%=0 GOTO 890
1140 IF P%(I%,J%) < P% GOTO 890
1150 IF P%(I%,J%) > P% GOTO 1160 ELSE 1170
1160 P%=P%(I%,J%):VRY%=VRX%:LC%=0:CX%(0)=I%:CY%(0)=J%:G
    OTO 890
1170 IF VRY% > VRX% GOTO 890
1180 IF VRY% < VRX% GOTO 1190 ELSE 1200
1190 LC%=0:VRY%=VRX%:CX%(0)=I%:CY%(0)=J%:GOTO 890
1200 LC%=LC%+1:CX%(LC%)=I%:CY%(LC%)=J%
1210 GOTO 890
1220 CX2%=CX2%:CX1%=CX1%:VRY%=VRY%
1230 CLS #1:PRINT #1," I CHOOSE LINE NUMBER ";CX2%
1240 PRINT #1,"      AND COLUMN NUMBER ";CX1%
1250 LOCATE #1,1,4:PRINT #1,"THAT GIVES ME ";VRY%;" SQU
    ARE(S)"
1260 PLOT 270+(30*CX1%),70+(30*CX2%):DRAW 290+(30*CX1%)
    ,89+(30*CX2%),N%
1270 PLOT 290+(30*CX1%),70+(30*CX2%):DRAW 270+(30*CX1%)
    ,89+(30*CX2%),N%
1280 FOR T=1 TO 1000:NEXT T
1290 FOR K%=1 TO 8
1300 VRX%=0:C3%=CX1%:C4%=CX2%
1310 C3%=C3%+C1%(K%):C4%=C4%+C2%(K%)
1320 IF C%(C3%,C4%)=Q% GOTO 1330 ELSE 1340
1330 VRX%=VRX%+1:GOTO 1310
1340 IF C%(C3%,C4%)=6 GOTO 1350 ELSE 1360
1350 NEXT K%:GOTO 1400
1360 IF VRX%=0 GOTO 1350
1370 C3%=C3%-C1%(K%):C4%=C4%-C2%(K%)
1380 IF C%(C3%,C4%)=6 GOTO 1350
1390 C%(C3%,C4%)=N%:GOTO 1370
1400 C%(CX1%,CX2%)=N%
1410 GOSUB 2720
1420 GOSUB 1460
1430 IF PAS%=1 GOTO 1440 ELSE 1450
1440 CLS #1:PRINT #1,"      YOU MUST PASS":FOR T=1 TO 10
    00:NEXT T:GOTO 770
1450 GOTO 400
1460 FOR I%=1 TO 8
1470 FOR J%=1 TO 8
1480 FOR M%=0 TO 19 STEP 2
1490 Z%=270+(30*I%):H%=70+(30*J%):W%=H%+M%
1500 PLOT Z%,W%:DRAW Z%+20,W%,C%(I%,J%)
1510 NEXT M%:NEXT J%:NEXT I%

```

continued on the next page

```

1520 X%=X%+1
1530 IF X%=61 GOTO 1550
1540 RETURN
1550 CQ%=0:CN%=0
1560 FOR I%=1 TO 8
1570 FOR J%=1 TO 8
1580 IF C%(I%,J%)=Q% THEN CQ%=CQ%+1
1590 IF C%(I%,J%)=N% THEN CN%=CN%+1
1600 NEXT J%:NEXT I%
1610 IF CQ% > CN% GOTO 1680
1620 IF CQ%=CN% GOTO 1630 ELSE 1650
1630 CLS #1:LOCATE #1,25,2:PRINT #1,"DEADLOCK"
1640 END
1650 CLS #1:LOCATE #1,5,1:PRINT #1,"YOU HAVE ";CQ%;" SQU
    UARES;I HAVE ";CN%
1660 LOCATE #1,11,3:PRINT #1,"I HAVE WON....!!!"
1670 END
1680 CLS #1:LOCATE #1,5,1:PRINT #1,"YOU HAVE ";CQ%;" SQU
    UARES;I HAVE ";CN%
1690 LOCATE #1,5,3:PRINT #1,"WELL DONE. YOU HAVE WON !!
    "
1700 END
1710 IF C%(2,2)=Q% AND (C%(3,1)=N% OR C%(1,3)=N%) GOTO
    1720 ELSE 1730
1720 P%(3,1)=1:P%(1,3)=1
1730 IF C%(7,7)=Q% AND (C%(8,6)=N% OR C%(6,8)=N%) GOTO
    1740 ELSE 1750
1740 P%(8,6)=1:P%(6,8)=1
1750 IF C%(2,7)=Q% AND (C%(1,6)=N% OR C%(3,8)=N%) GOTO
    1760 ELSE 1770
1760 P%(1,6)=1:P%(3,8)=1
1770 IF C%(7,2)=Q% AND (C%(6,1)=N% OR C%(8,3)=N%) GOTO
    1780 ELSE 1790
1780 P%(6,1)=1:P%(8,3)=1
1790 IF D%=1 OR D%=8 OR E%=1 OR E%=8 GOTO 1820
1800 IF CX1%=1 OR CX1%=8 OR CX2%=1 OR CX2%=8 GOTO 1820
1810 RETURN
1820 FOR J%=1 TO 8 STEP 7
1830 FOR I%=2 TO 7
1840 IF C%(I%,J%)=N% GOTO 1850 ELSE 1860
1850 P%(I%+1,J%)=21:P%(I%-1,J%)=21
1860 IF C%(J%,I%)=N% GOTO 1870 ELSE 1880
1870 P%(J%,I%+1)=21:P%(J%,I%-1)=21
1880 NEXT I%

```

continued on the next page

```

1890 FOR I%=2 TO 7
1900 IF C%(I%,J%)=Q% GOTO 1910 ELSE 1920
1910 P%(I%+1,J%)=2:P%(I%-1,J%)=2
1920 IF C%(J%,I%)=Q% GOTO 1930 ELSE 1940
1930 P%(J%,I%+1)=2:P%(J%,I%-1)=2
1940 NEXT I%:NEXT J%
1950 P%(1,2)=1:P%(1,7)=1:P%(2,1)=1:P%(7,1)=1
1960 P%(2,8)=1:P%(7,8)=1:P%(8,2)=1:P%(8,7)=1
1970 FOR I%=2 TO 7
1980 IF C%(1,I%-1)=Q% AND C%(1,I%+1)=Q% THEN P%(1,I%)=2
5
1990 IF C%(8,I%-1)=Q% AND C%(8,I%+1)=Q% THEN P%(8,I%)=2
5
2000 IF C%(I%-1,1)=Q% AND C%(I%+1,1)=Q% THEN P%(I%,1)=2
5
2010 IF C%(I%-1,8)=Q% AND C%(I%+1,8)=Q% THEN P%(I%,8)=2
5
2020 NEXT I%
2030 FOR J%=1 TO 8 STEP 7
2040 FOR I%=4 TO 8
2050 IF C%(J%,I%) <> N% GOTO 2140
2060 IC%=I%-1:IF C%(J%,IC%)=6 GOTO 2140
2070 IF C%(J%,IC%)=Q% GOTO 2080 ELSE 2090
2080 IC%=IC%-1:GOTO 2070
2090 IF C%(J%,IC%)=6 GOTO 2110
2100 GOTO 2140
2110 IF IC%=0 GOTO 2140
2120 IF C%(J%,I%+1)=Q% AND C%(J%,IC%-1)=6 GOTO 2140
2130 P%(J%,IC%)=26
2140 IF C%(I%,J%) <> N% GOTO 2230
2150 IC%=I%-1:IF C%(IC%,J%)=6 GOTO 2230
2160 IF C%(IC%,J%)=Q% GOTO 2170 ELSE 2180
2170 IC%=IC%-1:GOTO 2160
2180 IF C%(IC%,J%)=6 GOTO 2200
2190 GOTO 2230
2200 IF IC%=0 GOTO 2230
2210 IF C%(I%+1,J%)=Q% AND C%(IC%-1,J%)=6 GOTO 2230
2220 P%(IC%,J%)=26
2230 NEXT I%
2240 FOR I%=1 TO 5
2250 IF C%(J%,I%) <> N% GOTO 2340
2260 IC%=I%+1:IF C%(J%,IC%)=6 GOTO 2340
2270 IF C%(J%,IC%)=Q% GOTO 2280 ELSE 2290
2280 IC%=IC%+1:GOTO 2270

```

continued on the next page

```

2290 IF C%(J%,IC%)=6 GOTO 2310
2300 GOTO 2340
2310 IF IC%=9 GOTO 2340
2320 IF C%(J%,I%-1)=Q% AND C%(J%,IC%+1)=6 GOTO 2340
2330 P%(J%,IC%)=26
2340 IF C%(I%,J%) <> N% GOTO 2430
2350 IC%=I%+1:IF C%(IC%,J%)=6 GOTO 2430
2360 IF C%(IC%,J%)=Q% GOTO 2370 ELSE 2380
2370 IC%=IC%+1:GOTO 2360
2380 IF C%(IC%,J%)=6 GOTO 2400
2390 GOTO 2430
2400 IF IC%=9 GOTO 2430
2410 IF C%(I%-1,J%)=Q% AND C%(IC%+1,J%)=6 GOTO 2430
2420 P%(IC%,J%)=26
2430 NEXT I%:NEXT J%
2440 IF C%(1,1)=N% GOTO 2450 ELSE 2460
2450 FOR I%=2 TO 6:P%(1,I%)=20:P%(I%,1)=20:NEXT I%
2460 IF C%(1,8)=N% GOTO 2470 ELSE 2480
2470 FOR I%=2 TO 6:P%(I%,8)=20:P%(1,9-I%)=20:NEXT I%
2480 IF C%(8,1)=N% GOTO 2490 ELSE 2500
2490 FOR I%=2 TO 6:P%(9-I%,1)=20:P%(8,I%)=20:NEXT I%
2500 IF C%(8,8)=N% GOTO 2510 ELSE 2520
2510 FOR I%=3 TO 7:P%(I%,8)=20:P%(8,I%)=20:NEXT I%
2520 IF C%(1,1) <> 6 THEN P%(2,2)=5
2530 IF C%(1,8) <> 6 THEN P%(2,7)=5
2540 IF C%(8,1) <> 6 THEN P%(7,2)=5
2550 IF C%(8,8) <> 6 THEN P%(7,7)=5
2560 P%(1,1)=30:P%(1,8)=30:P%(8,1)=30:P%(8,8)=30
2570 FOR I%=3 TO 6
2580 IF C%(1,I%)=N% THEN P%(2,I%)=4
2590 IF C%(8,I%)=N% THEN P%(7,I%)=4
2600 IF C%(I%,1)=N% THEN P%(I%,2)=4
2610 IF C%(I%,8)=N% THEN P%(I%,7)=4
2620 NEXT I%
2630 IF C%(7,1)=Q% AND C%(4,1)=N% AND C%(6,1)=6 AND C%(
5,1)=6 THEN P%(6,1)=26
2640 IF C%(1,7)=Q% AND C%(1,4)=N% AND C%(1,6)=6 AND C%(
1,5)=6 THEN P%(1,6)=26
2650 IF C%(2,1)=Q% AND C%(5,1)=N% AND C%(3,1)=6 AND C%(
4,1)=6 THEN P%(3,1)=26
2660 IF C%(1,2)=Q% AND C%(1,5)=N% AND C%(1,3)=6 AND C%(
1,4)=6 THEN P%(1,3)=26
2670 IF C%(8,2)=Q% AND C%(8,5)=N% AND C%(8,3)=6 AND C%(
8,4)=6 THEN P%(8,3)=26

```

continued on the next page

```

2680 IF C%(2,8)=Q% AND C%(5,8)=N% AND C%(3,8)=6 AND C%(
    4,8)=6 THEN P%(3,8)=26
2690 IF C%(8,7)=Q% AND C%(8,4)=N% AND C%(8,5)=6 AND C%(
    8,6)=6 THEN P%(8,6)=26
2700 IF C%(7,8)=Q% AND C%(4,8)=N% AND C%(5,8)=6 AND C%(
    6,8)=6 THEN P%(6,8)=26
2710 RETURN
2720 PAS%=0
2730 FOR I%=1 TO 8
2740 FOR J%=1 TO 8
2750 IF C%(I%,J%)=Q% GOTO 2780
2760 NEXT J%:NEXT I%
2770 PAS%=1:RETURN
2780 FOR K%=1 TO 8
2790 VR%=0:C3%=I%:C4%=J%
2800 C3%=C3%+C1%(K%):C4%=C4%+C2%(K%)
2810 IF C3% < 1 OR C3% > 8 GOTO 2820 ELSE 2830
2820 NEXT K%:GOTO 2760
2830 IF C4% < 1 OR C4% > 8 GOTO 2820 ELSE 2840
2840 IF C%(C3%,C4%)=N% GOTO 2850 ELSE 2860
2850 VR%=VR%+1:GOTO 2800
2860 IF C%(C3%,C4%)=Q% GOTO 2820 ELSE 2870
2870 IF VR% > 0 THEN RETURN
2880 GOTO 2820
2890 DATA 1,0,30,1,20,10,10,20,1,30,1,1,1,1,3
2900 DATA 3,3,3,1,1,0,1,20,3,5,5,5,5,3,20,-1,1,10,3,5
2910 DATA 0,0,5,3,10,-1,0,10,3,5,0,0,5,3,10,-1
2920 DATA -1,20,3,5,5,5,5,3,20,0,-1,1,1,3,3,3,1,1,1,-
    1,30,1,20,10,10,20,1,30
2930 DATA 263,100,263,120,270,130,255,130,255,130,255,1
    40,255,140,270,140
2940 DATA 270,140,270,150,270,150,255,150,255,160,270,1
    60,270,160,270,180
2950 DATA 270,180,255,180,270,170,255,170,270,190,270,2
    10,270,200,255,200
2960 DATA 255,200,255,210,255,220,270,220,270,220,270,2
    30,270,230,255,230
2970 DATA 255,230,255,240,255,240,270,240,255,250,270,2
    50,270,250,270,260
2980 DATA 270,260,255,260,255,250,255,270,270,280,270,3
    00,270,300,255,300
2990 DATA 255,310,255,330,255,330,270,330,270,330,270,3
    10,270,310,255,310
3000 DATA 255,320,270,320

```

continued on the next page

```

3010 DATA 310,355,310,375,350,355,335,355,335,355,335,3
65,335,365,350,365
3020 DATA 350,365,350,375,350,375,335,375,365,355,380,3
55,380,355,380,375
3030 DATA 380,375,365,375,380,365,365,365,410,355,410,3
75,410,365,395,365
3040 DATA 395,365,395,375,425,355,440,355,440,355,440,3
65,440,365,425,365
3050 DATA 425,365,425,375,425,375,440,375,455,375,455,3
55,455,355,470,355
3060 DATA 470,355,470,365,470,365,455,365,485,375,500,3
75,500,375,500,355
3070 DATA 515,375,515,355,515,355,530,355,530,355,530,3
75,530,375,515,375
3080 DATA 515,365,530,365

```

Raffles

Break into His Lordship's house and steal the booty. Lots of obstacles to trip you up, and you must beware of the dog! For one player against the computer. Keyboard or Joystick.

```

10 'RAFFLES by DAVID RADISIC
20 'copyright (c) AMSOFT 1985
30 '
40 MODE 0:INK 0,0:BORDER 0:INK 1,26:INK 2,15:INK 3,25
50 INK 4,14:INK 5,24,12:INK 6,0:INK 7,0:INK 8,0:PAPER #
  1,7
60 delay=200
70 DIM objx(5,20),objy(5,20),gemx(5,20),gemy(5,20)
80 GOSUB 380
90 GOSUB 720
100 pause=200:GOSUB 340
110 IF gems=0 THEN GOSUB 970
120 PEN 4
130 FOR i=10 TO 12
140 LOCATE 15,i:PRINT"SWAG";
150 NEXT
160 PAPER 0:CLS#2:PAPER 8
170 GOSUB 1170
180 GOSUB 1230

```

continued on the next page

```

190 GOSUB 1370
200 GOSUB 1510
210 IF rm=0 THEN GOSUB 1900
220 IF dead=0 THEN 160
230 pause=100:GOSUB 340
240 PAPER 0:CLS:PEN 1
250 LOCATE 4,3:PRINT"Do you want";
260 LOCATE 4,5:PRINT"Another game";
270 PEN 5:LOCATE 9,7:PRINT"Y/N";
280 i$=UPPER$(INKEY$):IF i$<>"Y" AND i$<>"N" THEN 280
290 IF i$="N" THEN MODE 2:PEN 1:STOP
300 RUN
310 IF dog=1 THEN RETURN
320 dog=1:dogx=minx(rm):dogy=miny(rm)
330 RETURN
340 FOR loop=1 TO pause
350 FRAME
360 NEXT
370 RETURN
380 rm=1:xp=6:yp=4:man$=CHR$(224):dog=0:stolen=0
390 SYMBOL 240,8,8,8,8,8,8,8,8
400 SYMBOL 241,0,0,0,0,255,0,0,0
410 SYMBOL 242,0,0,0,0,15,8,8,8
420 SYMBOL 243,0,0,0,0,248,8,8,8
430 SYMBOL 244,8,8,8,8,248,0,0,0
440 SYMBOL 245,8,8,8,8,15,0,0,0
450 SYMBOL 246,8,12,13,14,12,12,8,8
460 SYMBOL 247,8,12,12,14,13,12,8,8
470 SYMBOL 248,8,24,88,56,24,24,8,8
480 SYMBOL 249,8,24,24,56,88,24,8,8
490 SYMBOL 250,0,0,255,129,129,129,255,0
500 SYMBOL 251,28,20,20,20,20,20,20,28
510 SYMBOL 252,0,0,255,255,255,255,255,0
520 SYMBOL 253,28,28,28,28,28,28,28,28
530 SYMBOL 255,195,165,60,126,90,60,36,24
540 ENT 1,12,-4,1
550 ENT -2,=1000,60,=3000,40
560 ENV 1,10,1,5,2,-4,1,2,-1,20
570 windw$(1)=STRING$(2,250):windw$(2)=CHR$(251)+CHR$(8)
    +CHR$(10)+CHR$(251)+CHR$(8)+CHR$(10)+CHR$(251)
580 door$(1)=STRING$(2,252):door$(2)=CHR$(253)+CHR$(8)+
    CHR$(10)+CHR$(253)+CHR$(8)+CHR$(10)+CHR$(253)
590 switch$(1,0)=CHR$(246):switch$(1,1)=CHR$(247)
600 switch$(2,0)=CHR$(248):switch$(2,1)=CHR$(249)

```

continued on the next page

```

610 gem$=CHR$(144):obj$=CHR$(233):dog$=CHR$(255)
620 hit$=CHR$(246)+CHR$(248)+CHR$(247)+CHR$(249)+CHR$(2
52)+CHR$(253)+CHR$(250)+CHR$(251)+gem$+obj$+dog$
630 RESTORE 3010
640 FOR i=1 TO 5
650 READ minx(i),miny(i),maxx(i),maxy(i)
660 READ dir(i,1),dir(i,2),dir(i,3),dir(i,4)
670 NEXT
680 WINDOW #1,minx(rm)-1,maxx(rm)+1,miny(rm)-1,maxy(rm)
+1
690 WINDOW #2,1,14,1,25
700 CLS #1:PAPER #0,8
710 RETURN
720 ORIGIN 50,50
730 INK 6,24,12
740 RESTORE 3060
750 GOSUB 1280
760 LOCATE 3,20
770 PEN 5:PRINT">";
780 PEN 1:PRINT"Escape Routes";
790 PEN 5:PRINT"<";:PEN 1
800 LOCATE 10,2:PRINT"IN";
810 pause=300:GOSUB 340
820 CLS:LOCATE 1,3:INK 6,0
830 PEN 1:PRINT man$;" You The Thief":PRINT
840 PEN 2:PRINT LEFT$(door$(1),1);LEFT$(door$(2),1);" D
oors":PRINT
850 PEN 3:PRINT switch$(1,0);switch$(2,0);" LightSwitch
(OFF)"
860 PEN 3:PRINT switch$(1,1);switch$(2,1);" LightSwitch
(ON)":PRINT
870 PEN 4:PRINT LEFT$(windw$(1),1);LEFT$(windw$(2),1);"
Windows":PRINT
880 PEN 5:PRINT gem$;" Precious Gems":PRINT
890 PAPER 1:PEN 0:PRINT obj$;" Obstructions ":PEN 1:PA
PER 0:PRINT
900 PEN 1:PRINT dog$;" The Dog"
910 PEN 5:PRINT:PRINT:PRINT
920 PRINT" Use Joystick":PRINT" Or Cursor keys"
930 dummy=REMAIN(1)
940 AFTER delay*4,1 GOSUB 340
950 RETURN
960 '
970 'Generate Gems/obstacles

```

continued on the next page

```

980 '
990 FOR room=1 TO 5
1000 gemr=INT(RND*8)+2:objr=INT(RND*10)+5
1010 minx=minx(room):miny=miny(room):maxx=maxx(room):ma
    xy=maxy(room)
1020 FOR i=1 TO gemr
1030 x=INT(RND*(maxx-minx+1))+minx
1040 y=INT(RND*(maxy-miny+1))+miny
1050 gemx(room,i)=x:gemy(room,i)=y
1060 gems=gems+1
1070 NEXT i
1080 FOR i=1 TO objr
1090 x=INT(RND*(maxx-minx+1))+minx
1100 y=INT(RND*(maxy-miny+1))+miny
1110 objx(room,i)=x:objy(room,i)=y
1120 NEXT i
1130 gems(room)=gemr:obj(room)=objr
1140 NEXT room
1150 CLS
1160 RETURN
1170 ON rm GOTO 1180,1190,1200,1210,1220
1180 RESTORE 2670:RETURN
1190 RESTORE 2740:RETURN
1200 RESTORE 2810:RETURN
1210 RESTORE 2880:RETURN
1220 RESTORE 2960:RETURN
1230 PAPER 0:READ rm$:PAPER 8
1240 WINDOW #1,minx(rm)-1,maxx(rm)+1,miny(rm)-1,maxy(rm
    )+1:CLS #1
1250 PEN 1:LOCATE 1,1:PRINT SPACES$(19);
1260 LOCATE 1,1:PRINT "Room :";rm$;
1270 IF lights(rm) THEN INK 7,10:INK 8,10 ELSE INK 7,0:
    INK 8,0
1280 READ a$:IF a$="END" THEN RETURN
1290 IF a$="D" THEN 2180
1300 IF a$="W" THEN 2260
1310 IF a$="L" THEN GRAPHICS PEN 1:GOTO 2340
1320 IF a$="S" THEN 2420
1330 IF a$="F" THEN GRAPHICS PEN 6:GOTO 2340
1340 PRINT "***ERROR ***";
1350 STOP
1360 '
1370 'Display gems/objects
1380 '

```

continued on the next page

```

1390 PEN 6
1400 FOR i=1 TO obj(rm)
1410 LOCATE objx(rm,i),objy(rm,i)
1420 PRINT obj$;
1430 NEXT
1440 PEN 5
1450 FOR i=1 TO gems(rm)
1460 LOCATE gemx(rm,i),gemy(rm,i)
1470 PRINT gem$;
1480 NEXT
1490 PEN 1:LOCATE xp,yp:PRINT man$;
1500 RETURN
1510 xf=0:yf=0:PEN 1
1520 IF INKEY(0)<>-1 OR INKEY(72)<>-1 THEN yf=-1
1530 IF INKEY(2)<>-1 OR INKEY(73)<>-1 THEN yf=1
1540 IF INKEY(8)<>-1 OR INKEY(74)<>-1 THEN xf=-1
1550 IF INKEY(1)<>-1 OR INKEY(75)<>-1 THEN xf=1
1560 IF xf=0 AND yf=0 THEN 1630
1570 LOCATE xp+xf,yp+yf:ht$=COPYCHR$(#0)
1580 IF ASC(ht$)>239 AND ASC(ht$)<246 THEN 1510
1590 IF ht$<>" " THEN 1660
1600 LOCATE xp,yp:PRINT " ";
1610 PAPER 0:LOCATE 15,5:PRINT " ";:PAPER 8
1620 xp=xp+xf:yp=yp+yf
1630 LOCATE xp,yp:PRINT man$;
1640 IF dog>0 THEN dog=dog MOD 2+1:IF dog=2 THEN 2550
1650 GOTO 1510
1660 hit=INSTR(hit$,ht$):char=ASC(MID$(hit$,hit,1))
1670 ON hit GOTO 1690,1690,1690,1690,1750,1750,1850,190
0,1970,2090,2650
1680 GOTO 1600
1690 IF hit>2 AND hit<5 THEN char=char-1
1700 IF hit<3 THEN char=char+1
1710 PEN 3:LOCATE xp+xf,yp+yf:PRINT CHR$(char);
1720 lights(rm)=lights(rm) XOR 1
1730 IF lights(rm) THEN INK 7,10:INK 8,10 ELSE INK 7,0:
INK 8,0
1740 GOTO 1510
1750 IF xf<>0 AND yf<>0 THEN 1630
1760 IF xf<0 THEN dir=4 ELSE IF xf>0 THEN dir=3
1770 IF yf<0 THEN dir=1 ELSE IF yf>0 THEN dir=2
1780 IF dir(rm,dir)=-1 THEN 1630 ELSE rm=dir(rm,dir)
1790 IF dog>0 THEN GOSUB 310
1800 IF dir=1 THEN xp=6:yp=maxy(rm)

```

continued on the next page

```

1810 IF dir=2 THEN xp=6:yp=miny(rm)
1820 IF dir=3 THEN xp=minx(rm):yp=13
1830 IF dir=4 THEN xp=maxx(rm):yp=13
1840 RETURN
1850 IF xp>5 AND xp<8 THEN 1880
1860 IF xp<6 THEN dir=4 ELSE dir=3
1870 GOTO 1780
1880 IF yp>13 THEN dir=2 ELSE dir=1
1890 GOTO 1780
1900 PAPER 0:CLS:PEN 1
1910 LOCATE 3,3:PRINT"You have escaped";
1920 LOCATE 9,5:PRINT"with";
1930 IF gems=stolen THEN LOCATE 8,7:PRINT"All"; ELSE LO
    CATE 9,7
1940 PRINT USING " ##";stolen;
1950 PEN 5:LOCATE 9,9:PRINT"Gems";
1960 dead=1:RETURN
1970 LOCATE xp,yp:PRINT" ";xp=xp+xf:yp=yp+yf
1980 i=0
1990 i=i+1
2000 IF i>gems(rm) THEN 1510
2010 IF gemx(rm,i)<>xp OR gemy(rm,i)<>yp THEN 1990
2020 IF i=gems(rm) THEN 2050
2030 gemx(rm,i)=gemx(rm,gems(rm))
2040 gemy(rm,i)=gemy(rm,gems(rm))
2050 gems(rm)=gems(rm)-1:stolen=stolen+1
2060 MOVE 400,150+(stolen*2),1,1:DRAW 520,150+(stolen*2
    ),1,1
2070 SOUND 129,248,10,12,0,1
2080 GOTO 1980
2090 noise=INT(RND*15)
2100 SOUND 1,3000,10,noise,0,0,10
2110 PAPER 0:LOCATE 15,5:PRINT"Crash ";:PAPER 8
2120 IF noise<10 OR delay=50 THEN 1630
2130 delay=delay-50
2140 dummy=REMAIN(1)
2150 AFTER delay*4,1 GOSUB 310
2160 GOTO 1630
2170 '
2180 'Draw doors
2190 '
2200 READ no,dr$
2210 IF dr$="V" THEN dr=2 ELSE dr=1
2220 PEN 2

```

continued on the next page

```

2230 pic$=door$(dr):GOSUB 2500
2240 GOTO 1280
2250 '
2260 'Draw windows
2270 '
2280 READ no,wi$
2290 IF wi$="V" THEN wi=2 ELSE wi=1
2300 PEN 4
2310 pic$=window$(wi):GOSUB 2500
2320 GOTO 1280
2330 '
2340 ' Draw lines
2350 '
2360 READ x1,y1,x2,y2
2370 MOVE x1,y1,,0
2380 DRAW x1,y2,,0:DRAW x2,y2,,0
2390 DRAW x2,y1,,0:DRAW x1,y1,,0
2400 GOTO 1280
2410 '
2420 'Draw switches
2430 '
2440 READ no,sw$
2450 IF sw$="L" THEN sw=1 ELSE sw=2
2460 PEN 3
2470 pic$=switch$(sw,0):GOSUB 2500
2480 GOTO 1280
2490 '
2500 'Print char
2510 '
2520 READ x,y:LOCATE x,y:PRINT pic$;
2530 no=no-1:IF no>0 THEN 2520
2540 RETURN
2550 PEN 1:LOCATE dogx,dogy:PRINT " ";
2560 man$=CHR$(225)
2570 IF (dogx=xp AND dogy=yp) OR (dogx=xp+xf AND dogy=y
    p+yf) THEN 2650
2580 IF dogx<xp THEN dogx=dogx+1
2590 IF dogx>xp THEN dogx=dogx-1
2600 IF dogy<yp THEN dogy=dogy+1
2610 IF dogy>yp THEN dogy=dogy-1
2620 LOCATE dogx,dogy:PRINT dog$;
2630 SOUND 1,0,RND*40,10,1,2,31
2640 GOTO 1510
2650 PRINT"SNAP";

```

continued on the next page

```

2660 dead=1:RETURN
2670 DATA Hallway
2680 DATA L,64,308,226,4
2690 DATA D,2,H,6,3,6,22
2700 DATA D,2,V,4,12,9,11
2710 DATA S,1,L,4,11
2720 DATA S,1,R,9,14
2730 DATA END
2740 DATA Lounge
2750 DATA L,2,308,258,4
2760 DATA D,1,V,10,12
2770 DATA W,1,H,6,3
2780 DATA W,1,V,2,12
2790 DATA S,2,R,10,11,10,15
2800 DATA END
2810 DATA Dining room
2820 DATA L,2,308,258,4
2830 DATA W,1,V,10,12
2840 DATA W,1,H,6,3
2850 DATA D,1,V,2,12
2860 DATA S,2,L,2,11,2,15
2870 DATA END
2880 DATA Kitchen
2890 DATA L,2,276,384,4
2900 DATA D,2,H,6,5,6,22
2910 DATA W,1,H,10,22
2920 DATA W,1,V,14,13
2930 DATA D,1,V,2,13
2940 DATA S,1,L,2,16
2950 DATA END
2960 DATA Pantry
2970 DATA L,2,276,256,4
2980 DATA D,1,V,10,12
2990 DATA S,1,R,10,11
3000 DATA END
3010 DATA 5,4,8,21,0,4,3,2
3020 DATA 3,4,9,21,-1,-1,1,-1
3030 DATA 3,4,9,21,-1,-1,-1,1
3040 DATA 3,6,13,21,1,0,-1,5
3050 DATA 3,6,9,21,-1,-1,4,-1
3060 DATA L,64,308,480,100
3070 DATA F,250,98,294,102
3080 DATA F,250,306,294,310
3090 DATA F,390,94,430,106

```

continued on the next page

```
3100 DATA F,390,302,430,314
3110 DATA F,474,240,488,270
3120 DATA F,474,124,488,154
3130 DATA F,58,240,72,270
3140 DATA L,226,308,322,180
3150 DATA L,160,180,480,100
3160 DATA L,64,180,160,100
3170 DATA END
```

If you've enjoyed these games, you may like to join the AMSTRAD COMPUTER USERS' CLUB. As well as many other benefits and privileges, you get a free monthly magazine which includes program listings for games and 'utilities', special features, free competitions, and up to the minute information -from the source!

Appendix 4

Index

(Note that all references given in this index correspond to chapter and page number, e.g. 1.42 refers to chapter 1 page 42.)

A

!A	1.42	1.72	5.7
Aborting system disc functions			1.76
ABS			3.3
AFTER	3.4		8.29
Amplifier (external)	1.9		7.39
AMSDOS	1.72	5.1	5.25
AMSDOS commands			5.7
AMSDOS error messages	5.14	7.31	7.32
AMSDOS filenames		5.2	5.3
AND	3.4		8.18
AND (LOGO)			6.11
Animation			8.53
.APV			6.29
Arithmetic operations	1.33	7.27	7.28
Arithmetic operations (LOGO)			6.10
Arrays	2.3	3.18	3.25 7.28
ASC			3.5
ASCII	6.7	7.8	7.21
ASCII characters		7.8	7.9 8.15
ASCII files	1.43	3.71	5.11 7.29
ASM			5.27
ATN			3.5
AUTO		2.10	3.5

B

!B	1.42	1.72	5.8
Backup master disc	1.74	1.75	4.1
BASIC	1.22	3.1	7.32 8.7
BASIC disc			4.2
BF			6.7
BIN\$			3.6
Binary files	1.43		3.71

Binary numbers	8.9
Bit	8.9
BK	6.18
BL	6.7
BOOTGEN	5.27
BORDER	1.47 3.6 7.6
BREAK	3.6
BRIGHTNESS control	1.4
BUTTONP	6.23
BYE	6.6 6.26
Byte	8.9

C

CALL	3.7
CAPS LOCK key	1.16
Cassette operation	1.7 1.73 4.7 5.22
CAT	1.41 3.7
CAT (cassette)	4.7
CATCH	6.28
CHAIN	3.7
CHAIN (cassette)	4.9
CHAIN MERGE	3.8
CHAIN MERGE (cassette)	4.9 7.39
Changing discs	5.2 7.31
CHAR	6.7
Characters	1.53 7.9 7.43 8.14
Checking discs	1.76 5.22
Checksum	8.32
CHKDISC	1.76 5.22
CHR\$	1.53 3.8 8.16
CINT	3.8
Circles	1.58
CLEAN	6.15
CLEAR	3.9
CLEARINPUT	3.9
CLG	3.9
CLOAD	5.22
CLOSEIN	2.10 3.10
CLOSEIN (cassette)	4.10
CLOSEOUT	2.9 3.10

CLOSEOUT (cassette)	4.11
CLR key	1.17
CLS	1.22 3.10 7.4
CO	6.26
Colours	1.46
CON	5.20
Configuration sector	5.17
Configuring a disc	4.5
Connecting a mains plug	1.1
Connecting up the computer	1.2
Connecting peripherals	1.7 7.38 7.39 7.40 7.41
Console control codes (CP/M)	5.17
Console commands	5.18
CONT	3.10 7.29
.CONTENTS	6.28
CONTRAST control	1.4
Control characters	7.3 8.51
Control codes	7.1 7.3 8.51
Copy cursor editing	1.28
COPYCHR\$	3.11
COPYDISC	1.75 5.22
Copying discs	1.74 5.22
Copying files	1.42 5.10 5.21
COPY key	1.28
COS	3.11
COS (LOGO)	6.10
COUNT	6.8
CP/M	5.16
!CPM	1.39 5.8
CP/M filenames	5.18
CP/M system disc	1.39 1.74 4.2
CP/M system tracks	5.17
CREAL	3.11
CS	6.16
CSAVE	5.22
CT	6.14
Cube root	1.35
CURSOR	3.12 7.3
Cursor keys	1.15

D

DATA	3.12	7.27	8.31
Data only format	5.25	7.75	
DC sockets	1.2	1.3	
DDT		5.27	
DEC\$		3.13	
.DEF		6.30	
DEF FN	3.13	7.29	
DEFINT		3.14	
DEFREAL		3.14	
DEFSTR		3.15	
DEG		3.15	
DEL key		1.15	
DELETE		3.16	
.DEPOSIT		6.29	
DERR	3.16	7.31	7.32
DI		3.17	8.30
DIM	2.3	3.18	7.28
DIR (CP/M)	1.40	5.19	
DIR (LOGO)		6.22	
DIR		5.8	
Direct console commands		5.18	
Discs	1.11	1.38	4.1
DISC		1.73	5.8
DISCCHK		1.76	5.22
DISCCOPY		1.74	5.22
Disc directory		5.1	7.31
Disc drive (additional)	1.8	1.75	7.40
DISC DRIVE 2 socket		1.9	7.40
Disc file commands (LOGO)		6.22	
DISC.IN		1.73	5.8
Disc organisation		7.44	
DISC.OUT		1.73	5.8
DOT		6.16	
Dotted lines		3.43	8.49
DRAW	1.56	3.19	
DRAWR		3.19	
DRIVE		5.9	
Dr. LOGO		6.1	
DUMP		5.27	

E

ED	5.27
EDIT	1.27 3.20
Editing	1.27
Editing (LOGO)	6.5 6.14
EI	3.20 8.30
EJECT button	1.14
ELSE	1.29 3.20
EMPTY	6.8
END	3.21
END (LOGO)	6.13
ENT	1.69 3.21 8.41
ENT (LOGO)	6.25
ENTER key	1.15 1.18
ENV	1.67 3.23 8.38
ENV (LOGO)	6.25
Envelope planner	7.37
EOF	3.25 4.10 5.7 7.29
ER	6.21
ERA	5.19
ERA	5.9
ERASE	3.25
ERL	3.25
ERN	6.21
ERR	3.26 7.32
ERRACT	6.29
ERROR	3.26
ERROR (LOGO)	6.28
Error messages	7.27
Error messages (AMSDOS)	5.14 7.31 7.32
Error numbers	7.27
ESC key	1.17
EVERY	3.27 8.29
EX1 and EX2 programs	5.5
.EXAMINE	6.29
Exception handling commands (LOGO)	6.28
EXP	3.27
Expansion characters	3.37 7.22
EXPANSION socket	1.10 7.40
Exponentiation	1.35 1.37
External commands	5.7 7.30

F

FALSE	6.29
FD	6.18
FENCE	6.16
FILECOPY	4.2
File copying tables	5.12 5.13
Filenames	1.41 5.2 5.3 5.18
Filetypes	5.2
FILL	1.60 3.27 8.48
FIRST	6.8
FIX	3.28
Flashing colours	1.50
Flow of control commands (LOGO)	6.26
Flush sound channels	3.73 8.38
FN	3.28
FOR	1.30 3.28 7.27 7.30 8.16
FORMAT	1.40 5.25
Format (print)	3.61 8.22
Format (disc)	1.38 1.40 7.44
FPUT	6.8
FRAME	1.55 3.29
FRE	3.29
FS	6.16

G

GLIST	6.21
GO	6.26
GOSUB	1.31 3.30 7.27
GOTO	1.24 3.30
Graphic screen commands (LOGO)	6.15
Graphics	1.53 8.47
GRAPHICS PAPER	3.30 8.50
GRAPHICS PEN	3.31 8.48
GPROP	6.22

H

Hardware	7.45 7.46 8.56
Headers	5.3 5.12 5.13
Headphones	1.9

HEX\$	3.31
Hexadecimal numbers	8.11
HIMEM	3.32
Hold sound channels	3.73 8.38
HT	6.18

I

IBM format	5.25 7.45
IF	1.28 3.32
IF (LOGO)	6.26
Indicator lamp (disc)	1.14
INK	1.48 3.33 7.6
INKEY	3.33 7.43
INKEY\$	2.12 3.34 7.43
Ink modes	7.5 8.52
INP	3.34
INPUT	1.25 2.2 3.35 7.28
INPUT (cassette)	4.10
Inserting discs	1.11
INSTR	2.5 3.36
INT	3.36
INT (LOGO)	6.10
Interrupts	7.7 8.29
I/O	7.38 7.46
ITEM	6.9

J

JOY	3.37 7.43
Joystick commands (LOGO)	6.23
Joysticks	1.7 7.21 7.23 7.43
JOYSTICK socket	1.7 7.38

K

KEY	3.37 7.22
KEY DEF	3.38 7.22 7.43
Keyboard	1.15 7.21 7.22 7.23 7.43
KEYP	6.23
Keywords	1.22 3.1 7.32

L

LABEL	6.27
LEFT\$	3.39
LEN	2.8 3.39
LET	3.39
LINE INPUT	3.40
LINE INPUT (cassette)	4.10
LIST	1.23 3.40
LIST (LOGO)	6.9
List processing commands (LOGO)	6.7
LOAD	1.42 3.41
LOAD (cassette)	4.9
LOAD (LOGO)	6.23
Loading software	1.20
Loading the Welcome program	1.21
LOCAL	6.13
LOCATE	1.53 4.41 7.6
LOG	4.42
LOG10	3.42
Logging in a disc	5.17
Logic	8.18
Logical operations (LOGO)	6.11
LOGO	6.1
LST	5.20
LT	6.18
LOWER\$	3.42

M

Machine code	7.7
Mains plug connections	1.1
MAKE	6.13
MASK	3.43 8.49
MAX	3.43
MEMORY	3.44 7.27
Memory map	7.45
Menu	2.5 3.51 3.52
MERGE	3.44
MERGE (cassette)	4.9
MID\$	3.44
MIN	3.45

Mixed calculations	1.36
MOD	1.34 3.46
MODE	1.45 3.46 7.3
Modulator/power supply (MP2)	1.3 1.5 1.65
Monitor	1.2
MONITOR socket	1.2 1.3 7.39
MOVCPM	5.26
MOVE	1.57 3.47
MOVER	3.47
Musical notes	7.24
Music planner	7.37

N

NEW	3.48
NEXT	1.30 3.48 7.27 7.30 8.16
NODES	6.6 6.21
NOT	3.48 8.20
NOT (LOGO)	6.11

O

ON BREAK CONT	3.49
ON BREAK GOSUB	3.49
ON BREAK STOP	3.50
ON ERROR GOTO	3.50 7.29 7.32
ON GOSUB	2.10 3.51
ON GOTO	3.52
ON indicator	1.4 1.5
ON SQ GOSUB	3.52 7.7 8.44
OP	6.27
OPENIN	2.10 3.53 7.29 7.30
OPENIN (cassette)	4.10
OPENOUT	2.9 3.53 7.30
OPENOUT (cassette)	4.11
Operators	1.33 8.18
OR	3.54 8.19
OR (LOGO)	6.12
ORIGIN	1.59 3.54
OUT	3.55

P

PADDLE	6.24
PAL	6.16
PAPER	1.47 3.55 7.4
PAUSE	6.28
PD	6.18
PE	6.19
PEEK	3.56
PEN	1.47 3.56 7.15
Peripheral management	5.20
Peripherals	1.7
PI	3.57
PIP	5.20
Planners	7.34 7.35 7.36 7.37
PLIST	6.22
PLOT	1.56 3.57
PLOTR	3.58
PO	6.13
POKE	3.58
POS	3.59 4.10
POTS	6.13
POWERswitch	1.4 1.5
PPROP	6.22
PR	6.14
Primitives	6.7
.PRM	6.30
PRINT	1.22 3.59 8.22
Print formatting	3.61 8.22
PRINTSPC	3.60 8.23
PRINTTAB	3.60 8.23
PRINT USING	3.61 8.23
PRINTERsocket	1.8 7.41
Printers	1.8 7.41 7.42 7.43
Procedures	6.3 6.13
Property list commands (LOGO)	6.21
Protected files	1.43 3.71
PU	6.19
PUN	5.20
PX	6.19

Q

Queue (sound) 3.52 3.77 7.7 8.44

R

RAD 3.64
RANDOM 6.10
RANDOMIZE 3.64
Random numbers 3.69
RC 6.24
RDR 5.20
READ 3.64 7.27 7.28 8.31
Read errors 4.8
Read/Only files 5.11 5.23 7.31
RECYCLE 6.6 6.21
REDEFP 6.29
RELEASE 3.65 8.43
RELEASE (LOGO) 6.25
REM 1.30 2.2 3.65
REMAIN 3.66 8.30
REMPROP 6.22
REN 5.19
| REN 5.9
Rendezvous sound channels 3.73 8.37
RENUM 3.66
REPEAT 6.27
Resetting the computer 1.20 1.21
RESTORE 3.67 8.33
RESUME 3.67 7.29
RESUME NEXT 3.68
RETURN 1.31 3.68 7.27
RIGHT\$ 3.69
RL 6.24
RND 3.69
RQ 6.25
ROINTIME.DEM program 1.20
ROUND 3.70
RT 6.19
RUN 1.23 1.42 3.70
RUN (cassette) 4.9
RUN (LOGO) 6.27
Running the Welcome program 1.21

S

SAVE	1.41	1.43	3.71
SAVE (cassette)			4.13
SAVE (CP/M)			5.18
SAVE (LOGO)			6.23
Saving to cassette			4.12
Saving variables	2.9	3.90	5.5
Screen dump	1.44	3.71	5.5
SE			6.9
SETH			6.19
SETPAL			6.16
SETPC			6.20
SETPOS			6.20
SETSPLIT			6.15
Setting up			1.1
SETUP			5.24
SF			6.17
SGN			3.72
SHIFT keys			1.15
SHOW			6.15
Sideways ROMS			7.46
SIN			3.72
SIN (LOGO)			6.10
Software			1.20
SOUND	1.65	3.73	7.24 8.35
SOUND (LOGO)			6.25
Sound commands (LOGO)			6.25
Sound envelope planner			7.37
SPACE\$			3.75
SPC			3.75 8.23
Speakers (external)			1.9
Speech synthesiser			1.10
SPEED INK			3.75
SPEED KEY			3.76
SPEED WRITE			3.76 4.13
Sprites			8.54
SQ			3.77 8.45
SQR			1.34 3.77
Square root			1.34 3.77
SS			6.17
ST			6.20

STAT	5.23
STEP	3.78
Stereo	1.9 1.65 7.39
STEREO socket	1.9 1.65 7.39
STOP	3.78
STOP (LOGO)	6.27
STR\$	3.78
STRING\$	3.79
String variables	1.25 7.28
SUBMIT	5.27
SWAP	3.79
Switching on	1.4 1.5
SYMBOL	3.79 7.5 8.21
SYMBOL AFTER	3.81
Syntax error	1.18 7.27
SYSGEN	5.26
System disc	1.39 1.74 4.2
System format	5.25 7.44
System management	5.23
System primitives (LOGO)	6.28
System properties (LOGO)	6.29
System tracks (CP/M)	5.17
System variables (LOGO)	6.29

T

TAB	3.82 8.23
TAG	3.82 8.50
TAGOFF	3.83 8.50
TAN	3.83
!TAPE	1.73 5.9
!TAPE.IN	1.73 5.10
!TAPE.OUT	1.73 5.10
TAPE socket	1.7 7.39
TEST	3.84
TESTR	3.84
Text screen commands (LOGO)	6.14
Text/window planners	7.34 7.35 7.36
TF	6.20
THEN	1.28 3.84
THROW	6.28

TIME	3.85
TO	3.85
TO (LOGO)	6.14
Tone envelope	1.69 3.21 8.41
TOPLEVEL	6.29
Transient commands	5.20
Transparent writing	7.5 8.51
TROFF	3.86
TRON	3.86
TRUE	6.29
TS	6.15
Turnkey BASIC disc	4.3
Turnkey CP/M disc	4.4
Turtle graphics commands (LOGO)	6.18
TV receiver	1.3 1.5
TYPE (CP/M)	5.19
TYPE (LOGO)	6.15

U

UNT	3.86
UPPER\$	3.86
User defined characters	3.79 8.21
User defined keys	3.37 7.21 7.22 7.23
!USER	5.10
USING	3.87 8.23

V

VAL	3.87
Variables	1.25 7.32 8.15
Variables (LOGO)	6.4 6.13
Variables (saving)	2.9 3.90 5.5
Vendor format	5.25 7.44
Vertical hold control	1.4
Vibrato	8.42
VOLUME control	1.9 1.65
Volume envelope	1.67 3.23 8.38
VPOS	3.87

W

WAIT	3.88
WAIT (LOGO)	6.27
Wake-up message	1.4 1.5
Welcome program	1.21
WEND	3.88 7.30
WHILE	3.88 7.27 7.30
WIDTH	3.89
Wild cards	5.4
WINDOW	2.11 3.89 7.5 8.26
WINDOW (LOGO)	6.17
Window planners	7.34 7.35 7.36
WINDOW SWAP	3.90 8.27
WORD	6.9
WORDP	6.9
Workspace management commands (LOGO)	6.21
WRAP	6.17
WRITE	2.9 3.90 8.23
Write protection	1.12

X

XOR	3.91 8.20
XPOS	3.92
XSUB	5.27

Y

YPOS	3.92
------------	------

Z

ZONE	3.92 8.22
------------	-----------