(No pictures are available in preliminary doc)

TCS GENIE 16

Preliminary User Guide

(C) Copyright 1984.

Advance Technology.
8A Hornsey Street.
London N7 8HR.
U.K.

Copyright

Copyright (C) 1984 by Advance Technology (UK) Ltd. All rights reserved. No part of this documentation may be reproduced, transmitted, stored in a retrieval system, or translated into any language or computer language, in any form or by any means, electronic, mechanical, magnetic, optical, chemical, manual or otherwise without the prior written permission of Advance Technology (UK) Ltd., 8A Hornsey Street, London, N7 8HR, U.K.

Notice

Information in this manual is subject to change without notice and does not represent a commitment on the part of Advance Technology (UK) Ltd.

Limited warranty

Advance Technology (UK) Ltd shall have no liability or responsiblity to purchaser or any other person or entity with respect to any liability, loss or damage caused directly or indirectly by this product, including but not limited to any interruption of service, loss of business or anticipatory profits or consequential damages resulting from the use of this product.

The above is a limited warranty and the only warranty made by Advance Technology (UK) Ltd. Any and all warranties for merchantability and/or fitness for a particular purpose are expressly excluded.

To report errors in this documentation or suggestions for improvement, please complete and return the reader comment form at the back of this manual. Thank you for your help.

Advance 86, is a trademark of Advance Technology (UK) Ltd.

MSDOS, XENIX, GW-BASIC are trademarks of Microsoft Corporation.

Perfect Writer, Perfect Speller,
Perfect Calc, Perfect Filer are trademarks of Perfect Software Inc.

Supercalc, Supercalc 3,
Superwriter, Super Spellguard are trademarks of Sorcim Corporation.

IBM, IBM PC
are trademarks of International Business Machines Corporation.

Document no. UG86-DRAFT.

Introduction

Thank you for purchasing the Advance. We hope you will derive much pleasure in learning about and using your powerful new computer.

This is the Guide to Operations, which is intended to enable you to get the machine up and running, and to be happy with its basic workings. Hence it takes nothing for granted, and assumes that you are an absolute beginner. Those with some familiarity with microcomputers will need only the brief reference sections. There is enough in this guide to allow the beginner to read and use the wide literature on learning to program, on Basic and other languages, and to employ software packages. Full information on programming the Advance 86 is available in two companion manuals "Advance 86 Programmers Reference Manual" and "Advance Basic".

CHAPTER TWO THE HARDWARE

		Page
2.1	The System Unit	2-1
2.2	The Keyboard	2-2
2.3	Connections and Cables	2-3
2.4	Converting the 86a to the 86b	2-4
2.5	Connections on the 86b	2-4
2.6	Additional Requirements	2-4
2.7	The 86b unit and the Disk Drives	2-5
2.8	Diskettes	2-6
2.9	Inserting and removing Diskettes	2-7

THE HARDWARE

2.1 The System Unit

The Advance 86 consists of the basic system unit, which with the keyboard is supplied as the 86a, and an expansion package which clips into the top of the 86a, turning it into the 86b. .The Advance system unit is the centre piece of your system. Inside there is the 8086 16 bit microprocessor, RAM (random access memory) and ROM (read only memory), and the necessary electronics to allow the Advance to communicate with you. This can be via a television or monitor display, a keyboard, or loudspeaker. Other connectors allow cassette recorder, parallel printer, joysticks, light pen, and extension cards to be plugged in.

This manual introduces both versions of the 86. It follows that there will be some sections more relevant to a particular version, which can be skipped by someone working with the other.

These will be indicated clearly.

The 86b

Place the machine on a table or desk, front first, as shown above. For familiarisation, and so you can easily connect the necessary cables, have it at the front of the desk, so that you have easy access to the back as well.

2.2 The Keyboard

Press down gently on the front of the smoked perspex cover, and open it. Inside is the keyboard, with its connecting cable. Take it out, and put it on the desk to one side of the machine. Close the cover. The keyboard is how you communicate with your Advance. It is like a typewriter, but with additional functions. Like some typewriter keys, its keys have more than one function - they do more than one thing when used together with another key.

The keyboard is connected to the system unit by a coiled cable and a 5-pin Din plug. It connects, as shown above, to the socket at the right-hand-side bottom of the system unit, just below the red on-off switch. (If you are not used to DIN plugs, they are located by aligning the large raised ridge in the plug with the large hole at the bottom of the socket, and then easing the pins in.)

Place the connected keyboard alongside the computer. At the back of the keyboard are two short legs, which are released by a button on each side, allowing the keyboard to be tilted when standing on a desk. When you get going, you can use the keyboard on your knee, on a separate table, or wherever you prefer.

If you are replacing the keyboard in the space in the systems unit, disconnect the cable, and make sure the legs are put back into the storage position, that is, flat. Put the keyboard in front first, with the back and the cable going in last.

2.3 Connections and Cables

The rear of the system unit looks like this.

From left to right:

Mains Out provides power for a peripheral device, eg for a television or monitor. It uses an integral 3-pin plug.

Mains In supplies power for the system from the mains. The power cable is supplied, and should be correctly connected to a mains plug, with a 3-amp fuse. (Brown to live, blue to neutral, green and yellow to earth.)

T.V. provides a signal to an ordinary domestic television, colour or black and white. This should be carried by a cable with a phono plug at the computer end, and a coaxial (aerial) socket at the other for your T.V. The Advance will work with any TV and cannot damage it during normal use.

Comp.Sync. provides on outlet for a signal to a Composite Monitor, colour or black and white.

RGB provides an outlet for a signal to an RGB (red-green-blue) monitor.

Printer is a Centronics standard output for a parallel printer. It takes a DB25 plug.

Joystick allows the connection of two joysticks. It requires a DB15 connector.

Cassette is a 5-pin DIN socket which allows the connection of an ordinary cassette recorder for the storage of programs and other information on tape.

In the case of all these connections, you will require appropriate leads and plugs. Your Advance supplier or other computer or electronic shop, will be able to help you.

2.4 Converting the 86a to the 86b

This is a straightforward task, and the instructions are included with the 86b system expansion unit.

2.5 Connections on the 86b

86b users have a set of slots and connectors on the back of their machine, looking like this:

The DB25 socket provides an RS232 port which can be used to interface to a serial printer or a Modem. The expansion slots can be used in a multitude of ways, e.g. for further memory , additional serial or parallel ports or a real-time clock. Some of tne potential uses are listed in the Options and Possibilities section below.

2.6 Additional Requirements

In addition to the package, you will need a domestic television or monitor (composite or RGB) and an appropriate cable to connect it to the Advance. The display (TV or monitor) is where the computer usually indicates to you what is going on. It registers what you have typed in ,allows you to change it, takes it away and works on it (in ways it has been told to, earlier, by you or other people) and brings it back to the screen for your inspection.

86a owners will also need a cassette recorder (an ordinary domestic one is fine, as is one of the similar machines marketed specifically for computer use). 86b owners can, of course, also use a cassette recorder for storage, or as a source of programs, in addition to diskettes.

Any printer that uses a standard Centronics output can be connected to the parallel printer port. Follow the manufacturer's instructions for the printer, connecting it through a DB25 connector. (See appendix.)

2.7 The 86b unit and the Disc Drives.

(86a owners can skip this section.)

86b users should remove any packing material and cardboard protectors from the disc units. These are two slots in the front of the computer, each protected by a latch which rotates from quarter-past to half past the hour. This latch only operates when there is a disk in the machine, and is otherwise locked in the open position. Don't try to force it closed if there is no disk in the unit.

The latch stops inadvertent removal of disks during use, and raises and lowers the reading head of the disk read-write machinery. This is the equivalent of the recording head on a tape recorder. When the machine is switched off it does no harm to operate the lever with a disk inserted in the machine. With power on, the red warning light (just above the disk slot, on the right) indicates when the disk is in use. When the red light is on, you should not operate the lever or attempt to remove or insert a disk. If the machine is reading from or writing to the disks, the red light will go on and off, switching from disk to disk. Don't try and change disks while this is going on. It's o.k. to operate the lever and change disks at any other time. Unlike the stylus on a record player you don't have to park the head yourself and you needn't worry about its position.

If you are going to leave disks unused in the machine overnight, it is best to move the latch to the upright position, but there's no need to panic if you forget.

2.8 DISKETTES

With your Advance 86b, you get a DOS System diskette, together with a package of additional diskettes containing applications software. Blank diskettes are available from your Advance supplier and other computer or electronic shops. They are standard 5-1/4" floppy diskettes (called "disks" or "floppies" for short). For the Advance 86 you should buy 5 1/4", double-sided double density, 40 track soft sectored disks.

A diskette is a flexible plastic disk coated with a magnetic recording medium like that on a cassette recorder. This disk is protected by a black plastic cover which is internally coated with lubricant to keep the disk spinning freely. The surface of the disk is exposed in three places, at the centre, to its right hand-side, and in the long head slot, to allow recording and playback ("writing" and "reading"). Like a tape, the diskette can be used any number of times. Writing to it erases the existing information and replaces it by the new.

The information stored on diskettes is very dense, and it is most important to keep it in perfect condition. Don't touch the recording surface. Keep your diskettes in the outer envelope in which they are supplied, and store them vertically in boxes or purpose-made storage units. Keep them away from heat sources. Their medium is magnetic, so that it can be disrupted by other magnetic sources in their neighbourhood.

Diskettes have a permanent label attached, normally with the name something like "Dual head, double side, soft sector, double density". In the pack of diskettes you will find further labels of your own to attach. If you are going to write on either label, when it is attached to the disk, do not use ball-pen or pencil or anything requiring any degree of pressure. Use a felt-tip pen,

or, best, only write on a label before you attach it.

On the right-hand side of a diskette, there can be a small cut-out square. On other diskettes, this is covered by a tab. When it is so covered, the diskette is write-protected. The computer will not record on it, and consequently what is recorded cannot be destroyed by being recorded over. If you are sure you want to write over the contents, you can remove the tab.

Packs of diskettes you purchase will have small foil stickers to be placed over the notch, so that you can write-protect your own disks.

As a matter of interest only, information on diskettes is written on to the tracks on the diskette, which are arranged in concentric circles. They are 40 in number, numbered from the outside inwards. Each track is divided into sectors. The head moves over the tracks, looking for specified sectors, or empty ones to write to. The unit of measurement is usually the "byte", which is the space to hold a single character. On the Advance, each sector holds 512 of these; the normal double-sided diskette holds 348,640. The disk drives cannot immediately put information on to wholly blank disks. They have to be organised into the sector arrangement for the machine, a process called "formatting", which you will learn how to do in the DOS section,Ch.5.

2.9 Inserting and removing Diskettes.

Using a blank diskette, practice inserting and removing disks from the disk units. They are inserted into the slot with the label upwards, and towards you. (See diagram above.) The end with the head slot (the oval cutout) goes in first. If you have your right thumb over the manufacturer's label, that's right. Gently push the disk in, until you feel resistance. The whole disk will be inside the slot and the lever will close with ease. Removing the disk is similarly straightforward.

CHAPTER THREE POWERING UP

	Page
3.1 Powering up and system tests	3-1
3.2 Using a cassette recorder with the Advance 86	3-2
3.3 Files	3-4
3.4 Tapes	3-4
3.5 Saving to Cassette	3-4

3.1 Powering up and System tests.

Check that:

1. The mains lead from the Advance to the mains power supply is correctly wired and runs to the MAINS IN socket.

2. A television, or video monitor is connected to the correct socket on the Advance.

3. (86b owners) There is no diskette in either disk drive.

4. 86a owners (and 86b owners who wish to use cassette storage). The cassette recorder is connected to the mains, and to the cassette socket on the Advance.

Switch on by pressing the red switch on the right hand side of the system unit. Switch on your TV or monitor.

The Advance will perform a self-test which takes a few seconds (On the 86b the disc drives will whirr, and the red light on the front of the drive will come on). After a pause, there will should be a single beep from the loudspeaker, indicating everything is OK and the computer has passed its power-up tests. If nothing appears on your TV or monitor screen, turn up the brightness and contrast controls until you get the initial message. If you start with them turned right up, you can adjust downward later.

If your screen displays an error message on the top line, or you hear a sequence of multiple beeps from the loudspeaker there is a hardware fault. Contact your Advance supplier for help. (There is further information on such messages in Appendix E)

After the power up your screen should display a message like this:

```
ADVANCE TECHNOLOGY   BASIC VER. 1.0  1982
(C) Copyright Microsoft 1982

xxxxx  Bytes free
Ok

--

1LIST        2RUN<        3LOAD"     4SAVE"     5CONT<
```

The machine is now using the BASIC interpreter built into the memory (ROM). You can continue with this BASIC if you wish, and it is the one normally used with the 86a and cassette storage.

The "--" is a representation of a single flashing line on the screen, the CURSOR. This indicates where the next character you type will appear - and it is under your control. It moves as your text moves, as you enter and change information, or you can make it move yourself. Try a "space". On the numeric keypad to the right of the keyboard, there are four keys with arrows pointing in different directions. Try these. If you hold them down, the cursor moves further in the indicated direction.

86b owners will probably now prefer to familiarise themselves with the Disc Operating System. Chapter 5 shows you how to load and begin using the Advance Disk Operating system.

3.2 Using a cassette recorder with the the Advance 86.

Cassette Recorders differ in the facilities they offer. The Advance provides a motor control signal via the 5-pin DIN cassette plug. Motor control is when the computer starts and stops the motor on the cassette recorder, and uses the facility provided on most recorders for control via the switch on an external microphone. If your recorder has this facility, you will need, for example, a lead with a 5-pin DIN socket, and a 3-pin + jack plug at the other. (Pin connections shown in Appendix F.) Motor control is not necessary to load and save programs with your Advance. You simply start and stop the cassette recorder at the appropriate time yourself.

In what follows, I will say "start", "stop" and "rewind" the recorder as if you have no motor control. If you have, "start" and "stop" will happen automatically.

Your screen should be as above, with the BASIC prompt, Ok ,which tells you BASIC is ready to go (see Ch.6 if you are puzzled) and the cursor. Cassette control is done with the Basic language, so you now have to learn some simple BASIC commands.

The command LOAD, when you type it into the computer, and press ENTER, either of the keys labelled " " loads a program from cassette into computer memory. (The ENTER key is like the carriage return on a typewriter, with the extra function of passing what is on the screen to the computer itself. It is thus essential that after each instruction to the computer you ENTER it. We needn't keep repeating this. To make it go, press ENTER.)

Place the WELCOME cassette in your recorder. Make sure it is rewound to the beginning of Side 1. Adjust the volume control to about three quarters maximum .Type LOAD "WELCOME,R (The key on the left, F3, types LOAD" automatically, and all you do is type WELCOME,R if you prefer to do it this way.) Press the ENTER key. Press "play" on the recorder, and let the tape run. After a short while (not more than 45 secs.) there should be some sound, and

the screen should display the message

Cassette control o.k.

Volume control correctly set.

If you are not working with motor control, stop the tape recorder. As you will have worked out, the information to display this message and make the noise was stored on the tape. In fact, it was repeated six times, to help you find it.

If the tape played on for more than a minute, with nothing happening, press the keys marked CTRL and Break (the last is in the top right hand corner) and stop the recorder. CTRL and Break stops the program in the middle (it tells you where, but you don't need this information at the moment), and gets you back to BASIC Ok and cursor. Rewind the tape, increase the volume control, and try again. Once you find a working setting on the recorder, leave it there.

Rewind the tape, and load the program again, without the ,R at the end. This bit of the instruction tells the computer to run the program when it is loaded. If you leave it off, the program will load but not run. Instead, you will get the message

WELCOME .B Found.

Don't do anything for a moment, because the machine tells you this when it has found the program, but not loaded it. When it is loaded, it will come back with Ok and the cursor. Then, if you type Run the same simple message will appear.

As remarked, the first WELCOME program is recorded six times on the tape. There are other programs on the tape, each recorded twice. These are simply demonstration programs, and the list is shown with the second WELCOME program.

To load one of these, simply repeat the steps above, changing the filename. Remember to include the inverted commas. Go back to the beginning of the tape, and let it run. When it comes to the WELCOME program, the machine will flash the name at you, and tell you that it is skipping that one, since it wasn't requested. The message will be like "WELCOME.B skipped."It will load when it gets to the requested filename.

If you want to know what is on a tape, type a filename that you know (or suspect) is not used, and the machine will search for

it, flashing up in succession all the ones that are used.

3.3 FILES

The term "file" is used very much as in an office. The contents of files we have been using up to now have been programs, although they can contain other things. Just as a file has a label, we have been using FILE NAMES to get the contents of the FILE off the tape. The usage is a natural one. When you come to make your own files, for filenames:

do not use more than eight characters, which can be letters and/or numbers;

do not include colons in the name;

leave no spaces within them;

to avoid confusion, do not use any BASIC keywords, i.e. those in the Basic Quick Reference Section, Ch.6., or DOS keywords, listed in Ch.5. The computer will probably not become confused, but you might!

Filenames can be extended by a full stop and three characters. Thus "Chap.6" is valid, as is "REFER.UK1". Make them useful reminders.

3.4 TAPES

Information stored on cassette tape is subject to the same general conditions as music or speech in audio usage. That is, it can be deleted by recording over, and the tape is available for indefinite re-use. If you try playing a cassette with computer information, you get a series of high-pitched tones. This can be useful when reading from or writing to a tape, since you can hear that the information is where it is.

The same conditions for care apply as for sound tapes. Keep the cassette heads and tape paths clean. Tauten slack tapes when using, and store them in boxes.

You will find it most convenient to use short high-quality tapes specified for computer usage (although others work o.k.). Make a note of where programs are on the tapes by using the tape counters, to permit rapid relocation. Label your tapes with the filenames.

3.5 SAVING TO CASSETTE

Obviously, you will now want to know how to save your own files, that is, to store information (write) to the tape. This information will be as important as LOADing when you start writing your own programs. For the present, and as a demonstration, let's SAVE "HILO", which we will come back to, in the section on BASIC. Type in, exactly, the program on p.6-2 .

After each line of text, press ENTER ( ), which tells the computer to store that line.

When you have it typed in,type RUN. Play a game or two, until you are familiar with it. Side Two of your WELCOME cassette is empty, and you can save HILO on there. Type

SAVE "HILO"

and press ENTER. Start the motor on your recorder. Stop when you get the O.K. and the cursor. Wind the tape a little, and do it a second time. To convince yourself, switch the computer off. You have erased HILO from the computer's RAM. Power up again, and load HILO from tape. You can now load and save your own files. To find something useful to put in them, go to the section on Basic.

CHAPTER FOUR THE ADVANCE KEYBOARD

		Page
4.1	Introducing the Keyboard	4-1
4.2	Function Keys	4-5
4.3	<Num-Lock>	4-6
4.4	Using Multiple Keys	4-9
4.5	DOS and the Keyboard	4-10
4.6	Typing In	4-12

4.1 Introducing The Keyboard

You communicate with your Advance through the keyboard. At first glance the keyboard may appear very complicated - after all there are 84 keys on it (if we include the space bar).

However, if we look at the keyboard in detail and describe the keys and their use you should soon be able to find your way around it. You will have already noticed that the keyboard is divided into three sections and we shall look at the light grey keys in the centre section first.

When we refer to specific keys in this chapter and in the rest of the manual, we shall use the convention of putting the key symbol within two angle brackets, for example <Ctrl>.
The table below explains the abbreviations used.

Table 4.1. Key abbreviations and references.

<Ctrl>	Control
<Esc>	Escape
<Caps Lock>	Capitals lock
<Alt>	Alternate
<Num Lock>	Number lock
<Del>	Delete
<Pg Up>	Page Up
<Pg Dn>	Page Down
<Ins>	Insert
<PrtSc>	Print Screen
< ↑ >	Up arrow. Key 8 on number pad.
< ↓ >	Down arrow. Key 2 on number pad
< ← >	Left arrow. Key 4 on number pad
< → >	Right arrow. Key 6 on number pad.

The keys shaded above are similar to keys on a typewriter. On the Advance keyboard these are all coloured light grey.

On the top row of keys there are the numerals and additional characters obtained by using the shift key <⇧>.

On the second row down, the key on the extreme right is a backward slash <\>. There is a forward slash key </> on the bottom row and you will find that it is very important to use the correct slash key when you are entering commands to the computer.

On the third row, the two extreme righthand keys <'> and <'> provide single scare quotes. Unlike many typewriters, it is not necessary to use the shift key to obtain these characters.

The wide, horizontal bar key at the bottom is, of course, the space bar.

If you have not used a microcomputer keyboard before, many of these keys will be new to you. They are used to help you enter information, to give commands to the system and to write and run programs.

Taking the keys on the left first -

<Esc> (Escape)

This removes the line that the cursor is on so that you may correct it. The line is still present in the computer memory.

<⟵⇥>

This is a tab key and it works in the same way as a typewriter tab key. With most programs the tabs are set to move eight characters at a time.

<Caps Lock>

If you press the key, all the letters will appear in block capitals (uppercase). To return to lowercase letters, press the key again. (This is known as a toggle key).

<Ctrl> (Control).

This key is never used on its own, but with another key. For example, <Ctrl> + <Home> used together will clear the screen, leaving the cursor at the top right hand corner.

This is a shift key. As on a typewriter keyboard, there is a shift key at the left and at the right of the keyboard.

<Alt> (Alternate)

This key is used with the alpha keys (letter keys) when you are entering BASIC keywords. It is also used as an alternative control key in some applications.

(backspace)

This key 'backspaces' the cursor to the left and removes one character each time you press the key.

There are two keys with this symbol. They are both <ENTER> keys and they both work exactly the same way so you may use whichever one is easiest for you. <ENTER> must be pressed whenever you have finished typing in a command.

4.2 Function Keys

These keys are called function keys. They perform different functions depending on the software application program you are using. For example, in BASIC the function keys allow single key entry of common keywords. When you are using BASICA, you will see at the bottom of the screen a line similar to the following-

```
1LIST 2RUN   3LOAD"  4SAVE"  5CONT   6"LPT1 7TRON  8TROFF 9KEY
```

The number on each label corresponds to the number on the function key. For example, 5CONT means if you press <F5>, the machine will type the keyword CONT for you. In Basic you can assign the function keys yourself, using a KEY statement (see chapter six).

When you are using other applications, for example, Perfect Writer, the function keys will have other functions. If this sound complicated, don't worry. You will find that most programs that use the function keys help you by displaying a reminder of how the function keys operate within that particular program.

4.3 <Num Lock>

The key at the top left is <NUM LOCK> which stands for 'number lock'. This is another 'toggle' key. Pressing it once activates the number pad, pressing it again cancels it.

When <NUM LOCK> has been pressed -

1. The numeric keys 1 to 9 are operative.
2. <Del> is the decimal point key.
3. <Ins> is the zero key.
4. <-> is the minus key.
5. <+> is the plus key.

When <NUM LOCK> is off

1. < 7 > Home — moves the cursor to the top of the screen.
2. < ↑ 8 > — moves the cursor up one line at a time.
3. < 9 > PgUp — When you are running some software packages such as word processing, moves the cursor up one page at a time.
4. < ← 4 > — moves the cursor one character to the left.

5. < → 6 > moves the cursor one character to the right.

6. < 1 > moves the cursor to the end of the current line.
End

7. < ↓ 2 > moves the cursor down one line.

8. < 3 > moves the cursor down one page when running, for
PgDn example, a word processing package.

9. < 0 > A toggle key. Pressing <Ins> allows you to insert
Ins characters. Pressing again cancels insert mode.

10. <Del> Deletes the character where the cursor is placed.

These keys, like the function keys, can change their use according to the particular application you are running so you should make a point of checking this in the relevant manual.

There are two more keys -

<PrtSc> (Print Screen)

This key prints an * or if used with the shift key, causes all data on the screen to be printed on the printer.

<Scroll> Lock
Break This key in only used with <Ctrl> (see next page)

4.4 Using Multiple Keys

Sometimes you will need to press down more than one key at once, just like pressing the shift key on a typewriter to type capital letters. The most usual combinations you will use are

1. <Ctrl> + <Scroll Lock/Break> or <Ctrl> + <Break>

   This stops the program you are running.

2. <Ctrl> + <Num Lock>

   This stops the program but you may press any key to continue.

3. <Ctrl> + < →| > and <Ctrl> + < |← >

   Tabs the cursor to the next word on the line in the direction of the arrow.

4. <Ctrl> + <Home>

   Clears the screen. The cursor will move to the top left hand corner of the screen.

5. <Ctrl> + <Alt> + <Del>

   This causes what is called a 'system reset'. Hold <Ctrl> and <Alt> down together, then, keeping them down, press <Del>.

4.5 DOS and the Keyboard

These keys are used when you are using DOS. Their functions are explained more fully in Chapter 5.

<F1> Redisplays, one character at a time, a previously entered line.

<F2> Entering F2 followed by a character causes the screen to redisplay a line already entered upto that character.

<F3> The whole of a previously entered line is shown.

<F4> Entering F4 followed by a character causes the screen to pass over all the characters of a line you have already entered upto that character i.e. it is the opposite of F2

<F5> Saves the line currently displayed.

The <Ctrl> key also has special functions under DOS.

1. <Ctrl> + <Num Lock>

   Stops the system operation. Pressing any key causes the system operation to continue.

2. <Ctrl> + <Prtsc>

   This 'echoes' and prints on the printer all screen display. To cancel, press the same keys.

3. <Ctrl> + <Scroll Lock/Break>

   Stops the program you are running.

4.6 Typing In

Typing-in using the Advance keyboard is very easy but everyone makes the occasional 'typo' (typing error) To correct a mistake use the <⟵> (backspace) key. When you use this key you will see that the character to the left of the cursor disappears and the cursor moves one space to the left. You can then retype the word(s) correctly.

When you have typed in your instruction, you must remember to press the <ENTER> key <↵>. The computer will not receive your 'command' until you have pressed this key.

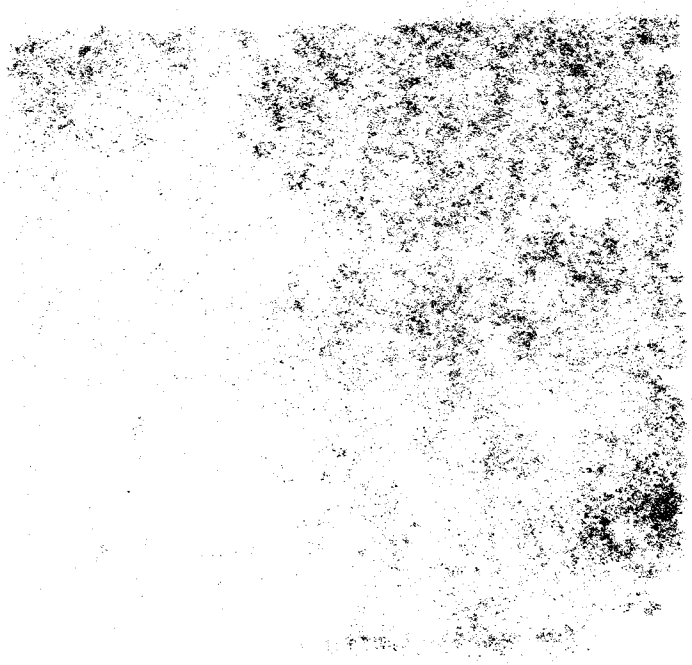
Summary In this chapter we have

1. described the keyboard, use of control and function keys and how the number pad functions.

2. explained how to correct typing errors and enter instructions.

CHAPTER FIVE ADVANCE DOS

		Page
5.1	Introduction	5-1
5.2	Booting the DOS	5-2
5.3	Entering the date	5-3
5.4	Entering the time	5-4
5.5	Changing from Drive A to Drive B	5-5
5.6	Formatting a diskette	5-6
5.7	System and non-system diskettes	5-7
	Formatting a system diskette	5-8
5.8	Copying diskettes	5-9
5.9	Introducing Files	5-11
	Filenames	5=11
	File specification	5-12
	Names that cannot be used for filenames	5-12
5.10	Wildcards	5-14
5.11	Listing filenames	5-16
5.12	Introducing file directories	5-18
5.13	Paths to sub-directories and files	5-20
5.14	DOS commands and path specifications	5-22
5.15	Introducing DOS commands	5-27
5.16	Batch processing	5-30
	The Autoexec.Bat file	5-32
	Creating a .Bat file with replaceable parameters	5-33
5.17	Input and Output	5-36
	Redirecting Output	5-36
	Filters	5-37
	Piping commands	5-38
5.18	DOS commands	5-40
	Command formats	5-40
	Batch processing commands	5-88
5.19	DOS Editing and function keys	5-95
	Control character functions	5-100
5.20	The Line Editor (EDLIN)	5-101
	Special DOS Editing Keys	5-103
	EDLIN command options	5-115
	EDLIN Editing commands	5-117
	EDLIN Error messages	5-141
5.21	File Comparison Utility (FC)	5-144
	Using FC	5-145
	FC Options	5-148
	How file differences are reported	5-147
	How to redirect FC output	5-148
	Examples of using FC	5-148
	FC Error messages	5-152



CHAPTER FIVE

5.1 Introducing the Advance Disk Operating System (DOS)

What is DOS? DOS is your 'silent partner' when you are using the Advance 86. It is a sophisticated piece of software that provides the interface between the computer hardware and both you (the user) and other software.

Advance DOS is a custom implementation of the standard disk operating system for 16-bit Microcomputers - Microsoft MS-DOS. The Advance implementation of MS-DOS is designed to be compatible with the IBM Personal Computer DOS, giving the Advance user access to the wide range of applications software which has been designed for the IBM and compatible machines.

Advance DOS allows you to load and execute applications programs, format and copy floppy disks, create and edit files and control whatever hardware devices you have attached to the Advance.

If some of the above terms are new to you, don't worry - as we go along we will introduce and define their usage.

How do you use DOS?

One of the diskettes supplied with your Advance will be labelled 'DOS'. On this floppy disk is the collection of computer programs that make up the Advance Disk Operating System.

To use DOS you will need to know how to load and start DOS and the names and use of DOS commands. In the first part of this chapter we shall introduce you to the most commonly used commands. Part 5.18 of this chapter defines the operation of all the commands and programs supplied with Advance DOS.

You need only master a small subset of the Advance DOS commands before you can do productive work. DOS is a toolbox; individual programs make up each tool. As with all toolboxes, some tools are so specialised you use them only rarely. Others you will use almost every time you switch on your computer. It is these that we will introduce now.

A note on notation.

First, let's make our notation clear. DOS will accept both capital (A-Z) and lower case letters (a-z) but for ease, we have printed what needs to be typed in in capital letters in bold print. For example

(type) **DISKCOPY**

5.2 Booting the DOS

To use DOS you must first load it from the DOS diskette into the computer memory. For historical reasons loading DOS is often known as 'booting' DOS.

If your computer is not switched on

1. Put the DOS diskette into Disk Drive A and close the drive latch. (Remember to insert the diskette carefully as we described in chapter 2).

2. Switch on the monitor or television, the printer, if you have one, and then the computer. There will be a few seconds pause, then the short beep of the system check, then the red Drive 'in use' light will come on. This indicates that the computer is transferring the DOS information from the disk to the computer memory. You will hear the Disk Drive clicking.

3.When DOS is loaded, the Disk Drive will stop clicking and the red light will go out. On the screen you will see a message similar to the following

```
Current date is Sun  01-01-1984
Enter new date: _
```

If your computer is switched on

1. Insert the DOS diskette into Drive A and close the Drive latch. (Remember to insert the diskette carefully, as we described in chapter 2).

2. Press and hold down the <Ctrl> +<ALT> keys, and press <Del>. This is called a system reset and you use these three keys whenever you wish to restart DOS.

The red light will come on, on Disk Drive A, indicating that the information from the DOS diskette is being transferred into the computer memory. You will also hear the Disk Drive clicking.

3. When DOS is loaded the Disk Drive will stop clicking and the red light will go out. On your screen you will see a message similar to the following

```
'Current date is  Sun 01-01-1984
 Enter new date _'
```

5.3 Entering the date

To enter the date use the number keys across the top of the keyboard.

1. Type one or two numbers for the month.
2. Type a dash - or a slash /
3. Type one or two numbers for the day
4. Type a dash - or a slash /
5. Type the numbers of the year. You can used four numbers for example 1984 or the last two, for example 84.
6. Press the <ENTER> key.

Try this now.

The current date should now be on the screen.

If you have made a mistake in typing in the date, DOS will return an error message

'Invalid date
Enter new date'

Check the following -

1. The month, day and year must be separated by either a dash or a slash, NOT a space.

2. Only numbers can be used, NOT letters.

3. Did you make a typing error? If you entered an impossible number for the day of the month, for example, 34, DOS will return the error message.

DOS does not insist that you enter the date. If you do not want to just press the <ENTER> key after the message

'enter new date _'

5.4 Entering the time.

After you have entered the date you will then see a message similar to the following

```
'Current time is 9: 30 : 42.21
 Enter new time:'
```

The display gives the hours, minutes , and seconds.

To enter the time use the number keys across the top of the keyboard.

1. Type the number of hours, between 0 and 23.

2. Type a colon (remember to use the shift key).

3. Type the number of minutes.

4. Type a colon

5. Type the number of seconds.

6. Press the <ENTER> key.

Try this.

The current time should now be displayed on the screen.

If DOS returns an error message

```
'Invalid time
 Enter new time _'
```

check the following -

1. Did you put colons between the hours, minutes and seconds? Semi-colons, slashes etc. are not accepted.

2. Only a period (fullstop) between seconds and hundreds of seconds will work.

If DOS gives you the error message try again.

If you do not wish to enter the time press the <ENTER> key after

```
Enter new time: _
```

Note:
If you do not wish to enter either the date or the time then press the <ENTER> key twice after 'Enter new date'. However we strongly recommend that you do enter the date and time correctly.

5.5 Changing from Drive A to Drive B

After you have either entered the date and time, or pressed the <ENTER> key twice, the following message is displayed

```
The Advance 86 Personal Computer DOS
Version 2.11 (C)Copyright Advance Technology (UK) Ltd 1983

A>_
```

The 'A>' on the last line is called the prompt. There are different prompts for when you are running different programs but whenever you see this prompt it means that the computer is expecting you to enter a DOS command.

The A indicates that you are using the disk in Drive A.

The Drive that is in use is often known as the 'default or 'current'' drive.

You may wish to use programs on a disk that is in Drive B. To change from Drive A to Drive B type

B:

Try it. (Remember to use the shift key (⇧) to type the colon).

The prompt should now be

B> -

Now change back.

type A:

The prompt is now

A>

5.6 Formatting a Diskette

Before you can use a new blank diskette it has to be prepared or 'formatted'. The DOS command to do this is

FORMAT

FORMAT checks the diskette for defective areas and builds a directory ready to hold the files that will be written to it.

To format a diskette-

1. Load DOS diskette in Drive A.

2. When the A> prompt is displayed, type the command

FORMAT B:

3. You will see the message

'Insert new diskette for drive B:
and strike any key when ready.'

4. Insert your new diskette in Drive B and close the latch. Press a key.

5. You will see the message

'formatting ....'

Formatting takes a few seconds and you will hear the Disk Drive clicking.

6. When formatting is completed you will see a message similar to the following

'Formatting...Format complete

362496 bytes total disk space
362496 bytes available on disk

Format another (Y/N)?'

Note: If FORMAT gives you an error message (see the further information on FORMAT in Part 5.18 of this chapter) we recommend that you abandon this diskette and re-run FORMAT on a new diskette.

7. Type Y for yes, or N for no.

5.7 System and non-system diskettes.

Your Advance loads DOS from the floppy disk in three sections or files as follows -

COMMAND.COM	Processes the commands you enter and then runs the appropriate program.
IO.SYS	Controls the hardware of the machine.
MSDOS.SYS	DOS itself.

IO.SYS and MSDOS.SYS are normally 'hidden', that is, they will not appear in the directory .

Disks that have these three files stored on them are called 'system disks'. You can copy these programs from the DOS diskette onto a previously formatted diskette. System disks can be used in place of the Advance DOS diskette whenever you switch your computer on or restart DOS.

If you try to load DOS from a non-system disk you will get an error message similar to

'Non-system disk or disk error
Replace and strike any key when ready'

Take the non-system disk out of the drive, insert a system disk and press a key. DOS will be loaded normally.

Formatting a System Disk

FORMAT can be used to format and copy the system files onto a new diskette at the same time. The procedure is the same as the one above but this time the command is

FORMAT B: /S

The 'S' stands for system. When formatting is completed the message this time will be similar to the following

```
'Formatting....Format complete
 System transferred

 362496  bytes total disk space
  40960  bytes used by system
 321536  bytes available on disk'

 Format another (Y/N)?
```

The message tells you how much space the system files have taken up and how much space there is still available on the diskette.

Again, if FORMAT gives an error message, we recommend that you abandon this diskette and re-run FORMAT on a new diskette.

5.8 Copying diskettes

One very good habit to develop is copying your diskettes so that you have always another copy of your programs available. Diskettes are easily damaged and if this happens, you will probably lose the programs that are stored on it. (For how to take care of diskettes, see chapter 2.) Copying a diskette is known as 'backing-up'.

The DOS command that enables you to do this is

DISKCOPY

Since it is so very important we suggest that the first diskette you should copy or 'back up' is the DOS disk itself.

Read through the following steps carefully before you start.

1. Using Drive A and the DOS diskette, load DOS. Make sure the A> prompt is displayed.

2. Insert the diskette (previously formatted, see 5.7) onto which you are going to copy DOS in Drive B.

3. Type

DISKCOPY A: B:

and press the <ENTER> key.

If DOS returns an error message, check the following -

1.There has to be a space between A: and B:
2. Colons only after letters A and B.

4. This message is now on the screen

'Insert source diskette in drive A:
Insert formatted target diskette in drive B:

Strike any key when ready.'

You have already inserted the source diskette (the DOS diskette) in Drive A and the new diskette (the target diskette) in Drive B. Press any key to tell DOS you are ready.

5. The following message will appear on the screen.

'Copying .... '

While DOS files are being copied from the diskette in Drive A to the diskette in Drive B, the red 'in use' light will go on on Drive A and then the one on Drive B.

6. When the copying is complete the message is

```
'Copy complete
 Copy another?  (Y/N)'
 _
```

7. Type N

This tells the computer that you do not wish to make another copy.

The DOS prompt is returned

```
A>_
```

8. Remove the diskette in Drive B and label it, using a felt tip pen.

Summary. You should now be able to

1. To load DOS
2. To enter the date
3. To enter the time
4. To change from Drive A to Drive B
5. To back up a diskette.

5.9 Introducing files.

File is the name given to a collection of related information. You may, for example, enter into the computer, details of current stock held and this information would be kept in one file. Another file might contain information about personnel and another details of accounts.

Filenames.

When you want information from a particular file you must give DOS the name of the file.

A name of a file has two parts.

1. Filename. This can be one to eight characters long, made up of

	*	letters	For example STOCK
and/or	*	numbers 0 to 9	For example STOCK84
and/or	*	any of the following special characters	$ # & @

! % ( ) _ { } < > ' - / ^ ~ | '

2. An extension. This is optional but it is useful for distinguishing between different sorts of files. You can use the extension TXT to indicate a text file. Some extensions are assigned by the programs, for example, BASICA expects .BAS

An extension
* starts with a period (full stop).
* consists of one, two or tnree characters.
* follows immediately after the file name.

For example STOCK84.TXT

Remember that if the name of your file does consist of both a filename and an extension you must always give DOS both parts.

It's a good idea to use names that indicate the content of the files. AAAAAAAA is an acceptable filename but not particularly informative!

If your filename is not accepted DOS will return an error message. Check the following -

1. There should be no spaces between the characters.

2. You cannot have more than eight characters in a file name or more than tnree characters in an extension.

3. Commas are not acceptable.

4. If the name of a file has an extension, the extension begins with a period and comes after the file name.

File specification.

In addition to knowing the name of your file, you can tell DOS which Disk Drive to use. To specify the Drive you type the letter A or B followed by a colon.

The filename follows immediately.

A:STOCK84.TXT

Notice that there are no spaces between the three parts of the file specification.

Whenever the drive specification is the same as the drive you are using currently, there is no need to specify the drive. So if you are using a diskette in Drive A and want to access another file on that diskette, there is no need to specify the drive in your file specification.

If you do not specify a Drive DOS assumes the file can be found on the current Drive. If your file is in fact on the diskette in Drive B DOS will give you the following error message.

'File not found'

Check that you have remembered to give the drive specification. Unless the file is on the current drive the drive specification must be given.

Names that cannot be used for filenames

There are some names you cannot use because they are 'reserved' to refer to devices, for example, the printer or keyboard. These 'illegal names' are

CON:	The Console device. Normally the Advance keyboard and screen, unless the CTTY command has been used.
AUX: or COM1:	Serial port No 1.
COM2:	Serial port No 2.
PRN: or LPT1:	Parallel printer port No 1.
LPT2:	Parallel printer port No 2.
LPT3:	Parallel printer port No 3.

NUL: This is a dummy device name.

Points to remember about device names

* If you use a device name make sure the device does exist and is connected on your system.

* You may use the reserved device name instead of a filename.

* DOS will ignore any drive specifier or filename extension entered with these device names.

*Typing a colon after the reserved device name is optional.

5.10 Wild Cards

This is the name given to two special characters that can be used in filenames and extensions. They enable you to pick out a group of files and can save you considerable time.

The ? wild card.

A ? in a file name or extension means that any character can occupy that place. So DOS will select all files that have matching names except for the character in the ? position. For example, suppose the following files are on the diskette

STOCK84.TXT

STOCK83.TXT

STOCK82.TXT

PERSON84.TXT

PERSON83.TXT

If you type

DIR STOCK8?.TXT

all the STOCK files will be listed on the screen. (DIR is the DOS command which you use when you want to list one or more files. You will find more information about DIR in parts 5.11 and 5.18).

You can used more than one ? in your specification and it may take the position of a character in either the filename, the extension or both.

The * wildcard.

A * in a filename or extension means that any character can occupy that position OR any other position in that filename or extension. Only one * is used, unlike the ?

For example if your diskette contains the following files

LIST1.EXE

LIST2.EXE

LIST3.EXE

LIST4.EXE

LISTINGS

```
DIR  LIST*.*
```

would display all five filenames.

It is possible to use both ? and * together.

The command

```
DIR  *.*
```

refers to all the files on your diskette.

5.11 Listing filenames.

To list all the files on the DOS diskette.

1. Switch on your computer and load DOS

2 Type DIR and press the <ENTER> key.

Your list on the screen should be similar to the following -

```
 Volume in drive A is ADVANCE DOS
 Directory of A:

COMMAND   COM    17984    1-01-80   12:59a
FORMAT    COM     6463    1-01-80   12:01a
MORE      COM      282    1-01-80   12:15a
PRINT     COM     4506    1-01-80   12:30a
CHKDSK    COM     6468    1-01-80   12:48a
RECOVER   COM     2308    1-01-80    1:01a
SYS       COM     1454    1-01-80    1:22a
EDLIN     COM     8080   10-19-83    7:51p
DISKCOPY  COM     1409   10-19-83    7:51p
SORT      EXE     1664    1-01-80    1:35a
FIND      EXE     6400    1-01-80    1:40a
FC        EXE     2585   10-19-83    7:51p
SET40     COM       16    1-01-80   12.02a
SET80     COM       16    1-01-80   12.03a
EXE2BIN   EXE     1649   10-19-83    7:51p
DEBUG     COM    12223   10-19-83    7:52p
BASICA    COM    59392    1-01-80    5:30a
LINK      EXE    42330   10-19-83    7.51p
      18 File(s)      154624 bytes free
```

To list all the filenames on another diskette.

1. Load DOS in Drive A.

2. Insert your diskette in Drive B

3. Type DIR B:

4. Press the <ENTER> key.

Remember that you have to tell DOS where to find the diskette. If you do not, DOS will look for your file on the current drive.

To list one filename on a diskette

1. Load DOS in Drive A

2. Insert the diskette into Drive B

3. Type DIR B:<filename>

4.Press the <ENTER> key.

The full filename, its size, and the date and time it was last modified are shown.

Summary. In parts 5.9 to 5.11 you have learned

1. How to name files.

2. How to use wildcards.

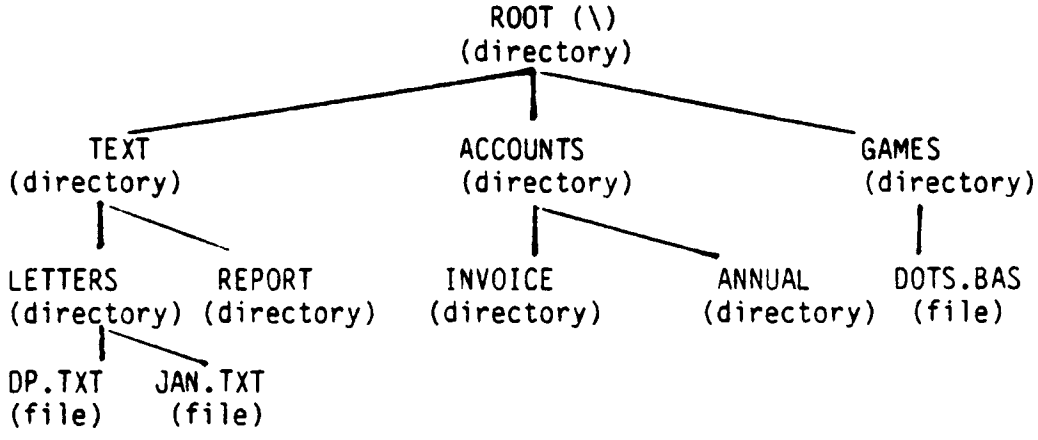
3. How to list filenames.

5.12 Introducing File Directories.

As we mentioned above the names of your files are kept in a directory on each diskette. This initial directory is known as the root or system directory and is the directory that is automatically created whenever you format a new diskette.

When you have a large number of files on your diskette, or when you wish to classify your files, you will probably find it more convenient to divide your files into groups. For example, you may wish to group all your text files together in one directory, all your accounting files in another and all your games files in another. To do this you create sub-directories, in this case one for text files, one for the accounts files and a third for the games files. These sub-directories can in turn contain the names of other sub-directories so we could divide the texts files into letters and reports and the accounts files into invoices received and annual reports. This way of organising your files is known as an hierarchical directory structure. (Don't panic! It is not nearly as complicated as it sounds.)

The structure for our example is as below -



The backslash (\) after ROOT is a special symbol for the root directory.

You may find it helpful to think of this structure as an upside down 'tree' structure with the root directory at the first level, the sub-directories as branches and the files within those sub-directories as the leaves.

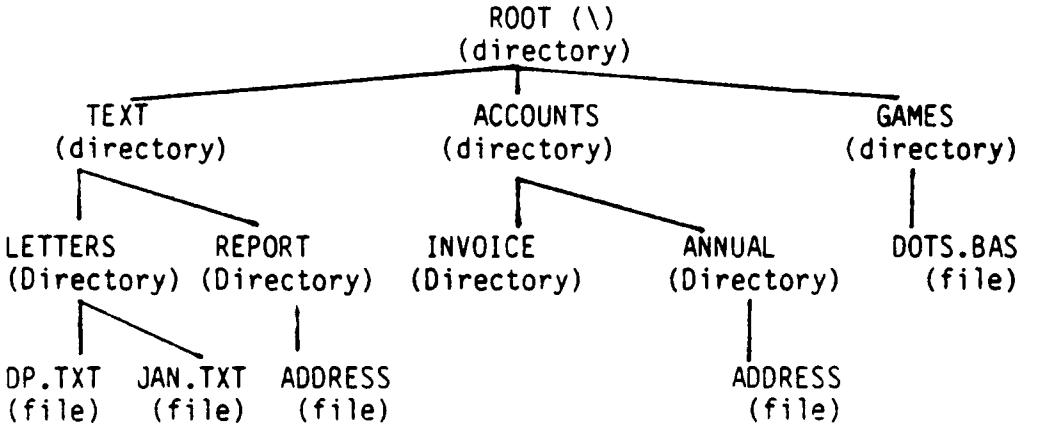
In the example above we have the root directory at the first level. At the second level there are three sub-directories, TEXT, ACCOUNTS and GAMES. TEXT contains two further sub-directories named LETTERS and REPORT. ACCOUNTS has two further sub-directories, INVOICE and ANNUAL. LETTERS has two files, DP.TXT and JAN.TXT. GAMES has one file, DOTS.BAS

Sub-directories, unlike the root directory, are actually files and this means that they are not restricted in size. Hence they can hold any number of filenames but for practical reasons we suggest you do not keep more than 30 - 40 files in any one directory. The only limitation will be the amount of available space on the diskette. So your tree structure can grow as you add more sub-directories and more files within those sub-directories.

Naming a sub-directory.

The rules for naming a sub-directory are the same as those for a file. (see 5.9)

It is possible to use a file name that is also in another sub-directory. For example, a file named ADDRESS could be created within the REPORT sub-directory and the same filename ADDRESS could be used for a file in the ANNUAL sub-directory. Provided that the files are defined in separate sub-directories, there is no problem.



The Current Directory.

The current directory,like the current drive, is the one that DOS will assume you are using unless you tell it otherwise. DOS will search your current directory if you enter a filename without telling it which sub-directory it is in and will create your new file within your current directory unless you take special action.

To check what your current directory is or to change your current directory you use the CHDIR (CD) command. (CHange DIRectory).Note that when DOS is first started it will use the root directory as the current directory until you use CHDIR.
For more details about CHDIR see 5.14 and 5.18.

5.13 Paths to sub-directories and files.

When you are using hierarchical directories DOS needs to know the path or route to the sub-directory or file you want to access. This is done by giving DOS the pathname. This is a sequence of sub-directory names ending with the file name. Each name must be separated by a backslash (\).

The path you specify can start from the root directory or the current directory. Putting a backslash (\) at the beginning of the pathname instructs DOS to start at the root directory. For example

```
\TEXT\LETTERS
```

directs DOS from the root directory, through the TEXT sub-directory to LETTERS.

Without this initial slash, DOS will start its search in the current directory and search downwards. If your current directory is TEXT, and you wish to use a file in REPORT, the pathname would be

```
REPORT\<filename>
```

If the file you wish to access is in your current directory you do not need to specify a path because DOS automatically searches the current directory.

There are two special shorthand notations that DOS can use.

. This indicates the name of the current directory. DOS creates this automatically when a subdirectory is made.

. . This indicates the parent directory of the current sub-directory. In the example above, TEXT is the parent of LETTERS

This particular special notation is useful when you are specifying a path to DOS because the double period is a quick way of telling DOS to move back up one level. For example, if the current directory is LETTERS and you wish to access ADDRESS you could use

```
\TEXT\REPORT\ADDRESS
```

or

```
..\REPORT\ADDRESS
```

The second instruction tells DOS to back up one level to the parent of the current directory and to continue from there.

This useful shorthand form can be used more than once in a path and it can be used with DOS commands such as DIR

```
DIR ..
```

lists the files in the parent directory of the current directory.

```
DIR..\..
```

lists the files in the parent of the parent directory!

A final point to remember about path specifications is that the longest path you can specify is limited to a maximum of 63 characters.

If necessary, the drive is specified at the beginning of the path specification. For example

B:\ACCOUNTS\ANNUAL\<filename>

5.14 DOS commands and path specifications.

There are two types of DOS commands:

External commands.

External commands reside on the diskette as a program file. They must, therefore be read from the diskette before they will execute. When you give DOS an external command, DOS will immediately search the current directory to find that command. If the external command is not in the current directory, you must tell DOS where to look for it by using the command

PATH

For example, if your current directory is REPORT, and all the external commands are in TEXT, the command

PATH \TEXT

instructs DOS to search both in your current directory and TEXT.

5.18 gives more details about the command PATH.

Internal commands

Internal commands are built in to DOS so they will execute immediately. They are the most commonly used commands. Some internal commands can use paths. and this gives them greater flexibility.

DIR \TEXT\LETTERS

lists all the files in the directory LETTERS.

DEL \TEXT\LETTERS

tells DOS to erase all the files in LETTERS.

If you try to delete the files in a sub-directory a message will appear on the screen.

'Are you sure (Y/N)?'

This gives time for second thoughts but if you wish to go ahead type Y (for yes). If you type N DOS will not delete the files in the subdirectory.

The command TYPE displays the contents of a file on a screen.

TYPE TEXT\REPORTS\<Filename>

Notice that this pathname has to have the filename specified at the end.

Displaying the current sub-directory.

To find out the name of your current sub-directory type

CHDIR

For example, if your current sub-directory is \TEXT\REPORT

and you type CHDIR

DOS will return the message

A:\TEXT\REPORT

This gives you your current drive and your current directory. To see what is in your current directory you use the command

DIR

The display will resemble the following

```
            Volume in drive A has no label
            Directory of A:\text\report

.                  <DIR>            1-01-80      12:00a
..                 <DIR>            1-01-80      12:00a
ADDRESS               17984         1-01-80      12:59a
COLOUR    BAS          1408         1-01-80      12:08a
        4 File(s)        296960  bytes free
```

There was no volume label assigned when the diskette was formatted.

REPORT has two files, ADDRESS and COLOUR.BAS.

Notice that files and directories are listed together. This means you cannot use the same name for both a sub-directory and a file.

'.' indicates the current directory \TEXT\REPORT

'..' is the shortnand for the parent directory \TEXT

Creating a sub-directory.

To create a sub-directory in your current directory use the command

MKDIR

or

MD

For example, to create a new sub-directory named MONREP type

MKDIR MONREP

To create a sub-directory in another part of the tree structure the command is

MKDIR <pathname> <name of new sub-directory>

For example

MKDIR \ACCOUNTS\MONTHFIG

creates a new sub-directory, MONTHFIG, under ACCOUNTS.

Changing the current directory.

It is very easy to change from your current sub-directory to another. The command is

CHDIR <pathname>

For example

CHDIR \TEXT\REPORT

changes the current directory from \REPORT to \TEXT

The command

CHDIR ..

will always put you in the parent directory of the current sub-directory.

Deleting a Sub-directory.

To delete a sub-directory the command is

RMDIR

For example

RMDIR LETTERS

would remove this from the current directory. The command will not work, unless the sub-directory is empty (except for the . and .. entries). This ensures that you do not delete files and sub-directories by accident.

To remove directories other than the current one, the command is RMDIR <pathname>

For example, to remove the REPORT directory

RMDIR \TEXT\REPORT

Once more, you must ensure that the sub-directory is empty except for the . and .. entries.

To remove all the files in a directory, the command is

DEL <pathname>

For example, to delete all the files in TEXT\REPORT, type

DEL \TEXT\REPORT

DOS returns the message

'Are you sure' (Y/N)

The . and .. entries cannot be deleted. DOS creates these as part of the hierarchical directory structure.

Summary. In Parts 5.12 to 5.14 we have talked about

1. Hierarchical file directories.
2. File names and path names.
3. Path specifications and the following DOS commands

   PATH

   DIR

   TYPE

   DEL

4. Displaying the current directory or changing the current directory using

   CHDIR

5. Creating a new directory using

   MKDIR

6. Removing a directory using

   RMDIR

7. Removing files using

   DEL

Part 5.18 of this chapter contains more information about these and additional DOS commands.

5.15 Introducing DOS commands

'Command' is the term used for an instruction you give to the system. DOS commands are used to ask the system to perform the following tasks.

* Copy and format diskettes

* Copy, compare, display and delete files

* Copy DOS system files to another diskette

* Load and execute systems programs such as EDLIN

* Load and execute applications programs such as SuperCalc 3 or Perfect Writer.

* Load and execute your own programs

* List filenames and directories

* Enter date, time and comments

* Set various printer and screen options

Internal and External Commands.

Internal commands are the most commonly used commands. When you use an internal command it will execute immediately. The following internal commands are described in Part 5.18.

BREAK	DEL (ERASE)	MKDIR (MD)	SET
CHDIR (CD)	DIR	PATH	SHIFT
CLS	ECHO	PAUSE	TIME
COPY	EXIT	PROMPT	TYPE
CTTY	FOR	REM	VER
DATE	GOTO	REN (RENAME)	VERIFY
IF	RMDIR	VOL	

Commands in brackets are synonyms and can also be used.

External commands are stored as program files on diskette. DOS has to read them from the diskette before it can execute tnem. If the relevant file is not on the diskette in your current drive, DOS will not be able to find it, and will return the error message -

'Bad command or file name'

All filenames with the following extensions .COM .EXE .BAT are considered as external commands. For example, FORMAT.COM and FIND.EXE are external commands.

Because all external DOS commands reside on the diskette, you can create external commands and add them to your system. Progams created with most programming languages will be .EXE (that is, executable) files.

Note that when you are entering the name of an external command the file extension should not be entered.

The following external commands are described in 5.18.

CHKDSK	MORE
DISKCOPY	PRINT
FIND	RECOVER
FORMAT	SET40
EXE2BIN	SET80
MODE	SORT
	SYS

Command options

You may add further information to a DOS command by using a command option. If you do not include an option, DOS assumes a 'default' value.

For example if you just type

DIR

DOS lists all the entries on the current drive.

If you type

DIR B:DOTS.BAS

only information about the file DOTS.BAS will be listed.

The default values for individual commands are given in the descriptions in part 5.18.

Points to Remember about all DOS Commands

* Commands are usually followed by one or more options.

* Commands and options may be entered in uppercase, lowercase, or a combination of both.

* Commands and options must be separated by 'delimiters', such as a space or a comma (,). You could also use a semicolon, (;), an equals sign, (=), or the tab key. You may use different delimiters within one command but you will probably find the space or the comma (,)the easiest to use.

* File specifications (drive:filename.ext) already contain delimiters (the colon and the period) so do not separate the three parts by additional delimiters.

* Files do not have to have an extension when you create them but if a filename does have an extension, you must include this when you are giving the filename to DOS.

* Most commands which are followed by a filename will also accept a path (directory) specification in front of the filename. If you are not creating directories you will not need path specifications.

* Commands can be aborted while they are running by pressing

<Ctrl> + <Break>

* DOS will only execute your command after you have pressed the <ENTER> key.

* Wildcards (? and *) and device names (for example PRN or CON) cannot be used as commands but can be used in command options.

* When commands produce a large amount of output on the screen, the display will automatically scroll to the next screen. To stop the display use

<Ctrl> + <Num Lock>

To continue, press any numeric or letter key.

* DOS editing and control keys can be used while entering DOS commands (see 5.19).

* Disk drives are referred to as source drives and target drives. The source drive is the drive from which you are transferring information. The target drive is the drive to which you will be transferring the information.

* The usual prompt for you to enter a command is the current drive letter plus >, for example,'A>'

* When the command has been completed, the system prompt will appear on the screen. If no error message appears on your screen the task has been successfully completed.

5.16 Batch processing

If you find yourself typing in the same sequence of commands over and over again to perform an often used task, you will find it more convenient to create a Batch file. This performs the entire sequence simply on receiving the name of the batch file. 'Batches' of your commands in such files are followed as if they were typed in at the keyboard.

Batch filenames

Each Batch file must be named with the extension .BAT. However, only the filename is entered to run the batch file. You do not enter its extension.

Creating batch files

There are two ways to create a batch file.

1. Use the Line Editor (EDLIN). Information on how to use EDLIN is given in 5.20.

2. Use the **COPY** command directly from the keyboard.

Batch Commands

There are two DOS commands available especially for use in batch files.

REM and PAUSE

REM allows you to include comments within your batch file.

PAUSE stops the system processing and allows you either to continue or to abort the batch process.

REM and **PAUSE** are described more fully in 5.18

An example of a batch file

One useful batch file would be one which could be used whenever you want to format and check a new diskette. Such a batch file would look like this;

```
1. REM        This file formats and checks new diskettes.

2. REM        The name of the file is NEWDISK.BAT

3. PAUSE      (Insert new diskette in Drive B:)

4. FORMAT B:

5. DIR B:

6. CHKDSK B:
```

To execute this file you would simply type in the filename

NEWDISK

You do not type in the extension.

The result is the same as if you had typed in each of the lines at the keyboard as individual commands.
The three steps are

1. Write a program	NEWDISK
2. Give a filename extension of .BAT to your filename and save on your directory	Directory: NEWDISK.BAT
3. Type NEWDISK as a command to DOS	Execute NEWDISK batch process.

Points to remember about batch processing

* Only the filename should be entered. Do not enter the extension.

* Only the commands in the file named <filename>.BAT will be executed.

* If you press <Ctrl>+ <Break> while the Batch program is running (that is, while you are in 'batch mode'),this message appears on the screen.

'Terminate batch job (Y/N)?'

If you press Y the batch processing will stop and the system prompt (>) will appear.

If you press N only the current command ends and the batch processing will continue.

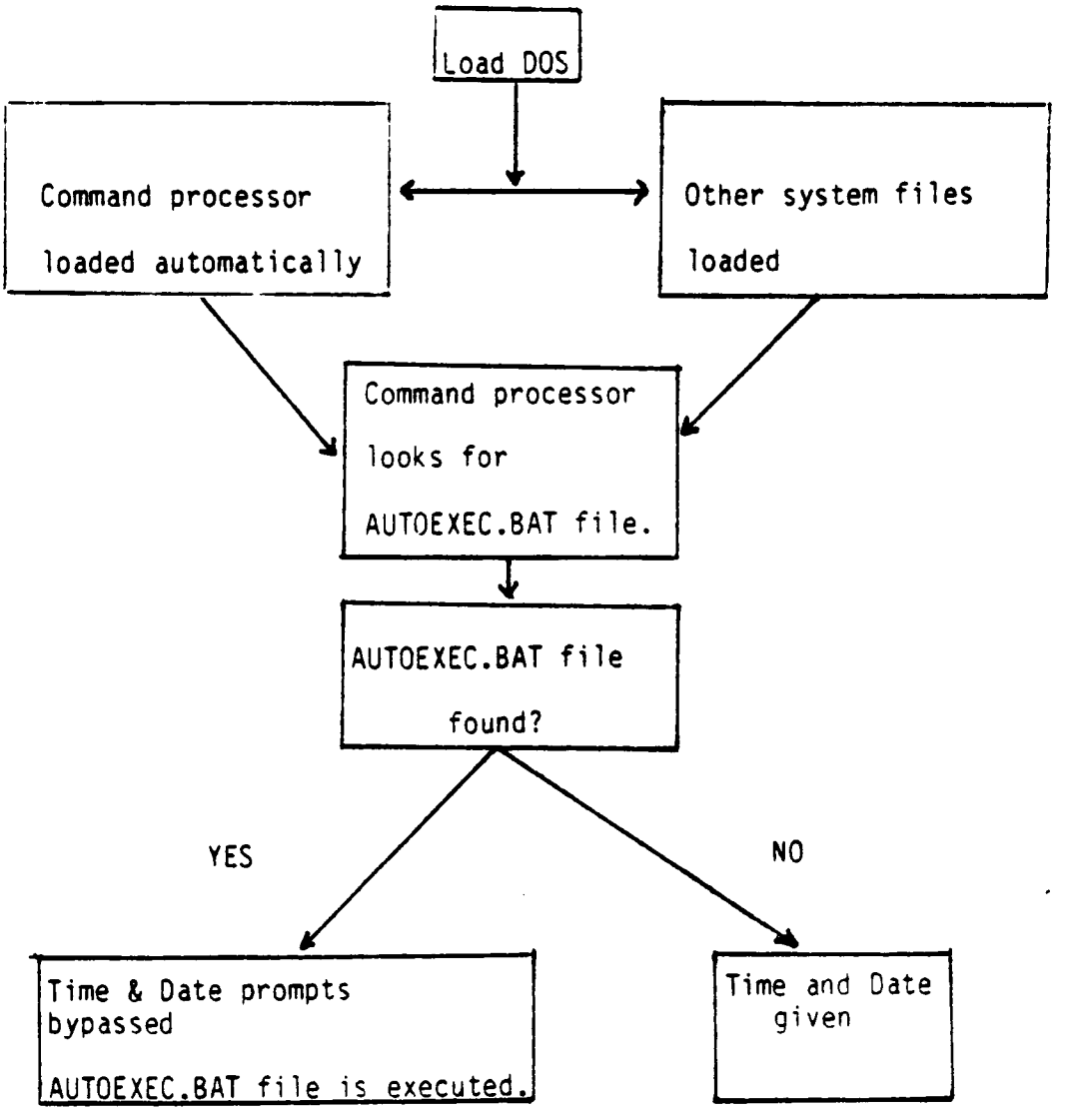
* If you remove the diskette containing a batch file being executed, DOS prompts you to insert it again before the next command can be read.

* The last command in a batch file may be the name of another batch file. This allows you to call one batch file from another when the first is finished.

The AUTOEXEC.BAT File.

AUTOEXEC stands for Automatic Program Execution. An AUTOEXEC.BAT file allows you to execute programs automatically when you start DOS. This is useful when you want to run a specific package (for example SuperCalc 3) under DOS and when you want DOS to execute a batch program automatically each time you start the system. Using an AUTOEXEC.BAT file avoids having to load two separate disks to perform either of these tasks.

When you start or restart DOS, the command processor searches the DOS diskette for the file AUTOEXEC.BAT. If DOS finds the file AUTOEXEC.BAT, the file will be immediately executed by the command processor. The date and time prompts are bypassed. If DOS does not find the AUTOEXEC.BAT file, then the date and time prompts will be given.The diagram below shows how DOS uses the AUTOEXEC.BAT file.



Creating an AUTOEXEC.BAT file using COPY

Suppose you want to load BASIC automatically and run a program called MENU each time you start DOS. You could create the following AUTOEXEC.BAT file. Do not forget to press the <ENTER> key at the end of each statement.

1. Type COPY CON: AUTOEXEC.BAT

This statement tells DOS to copy the information from the keyboard into the AUTOEXEC.BAT file.

2. Type BASICA MENU

This statement will go into the AUTOEXEC.BAT file. It tells DOS to load BASIC and run the MENU program whenever DOS is started.

3. Press <F6>, then the <ENTER> key to put the command BASICA MENU into the AUTOEXEC.BAT file.

4. The MENU program will now run automatically whenever you start DOS.

To run your own BASIC program, type the name of your program in place of MENU in the second line of the example above.

Points to remember about creating an AUTOEXEC.BAT file

* The AUTOEXEC.BAT file must be created in the root directory of your diskette.

* You can enter any DOS command or series of commands in the AUTOEXEC.BAT file.

* If you use an AUTOEXEC.BAT file, DOS will not prompt you for date and time unless you include the DATE and TIME commands in the AUTOEXEC.BAT file. It is a good idea to do this since DOS uses this information to keep your directory upto date.

Creating a .BAT file with Replaceable Parameters

It may be that you want to create an application program and run it with different sets of information (data). These data may be stored in various DOS files. A .BAT file with replaceable, ('dummy') parameters will help you to do this.

A parameter is an option which you can include in your command statement amd it gives additional information to the system. With DOS you can create a batch file which has dummy parameters. These are replaced by values supplied when the batch file is being executed.

The dummy parameters are named %0, %1 and so on upto %9.

For example, when you type the following command line

COPY CON MYFILE.BAT

the next lines you type are copied from the keyboard to a file named MYFILE.BAT on the current drive.

A>COPY CON MYFILE.BAT <ENTER>

COPY %1.MAC %2.MAC <ENTER>

TYPE %2.PRN <ENTER>

TYPE %0.BAT <ENTER>

Now press <F6> and then the <ENTER> key.

DOS responds with the message

1 File(s) copied

A> _

The file MYFILE.BAT, which consists of three commands, now resides on the diskette in the current drive.

The dummy parameters %1 and %2 are replaced sequentially by the parameters you supply when you execute the file. The dummy parameter %0 is always replaced by the drive designator, if specified, and the filename of the batch file (e.g. MYFILE).

Points to remember about creating a .BAT file with replaceable parameters

* Upto 10 dummy parameters (%0 to %9) can be specified. If you want to specify more than 10, refer to the description of the command SHIFT in 5.18.

* If you use a percent sign (%) as part of a filename within a batch file you must type it twice.

For example

To specify the file DNE%.EXE you must type is as DNE%%.EXE in the batch file.

Executing A .BAT file.

To execute the batch file MYFILE.BAT and to specify the parameters that will replace the dummy ones, you must enter the batch filename (but not its extension), followed by the parameters you want DOS to substitute for %1, %2, and so on.

You will remember that MYFILE.BAT contains the following three lines.

```
COPY %1.MAC %2.MAC

TYPE %2.PRN

TYPE %0.BAT
```

To execute MYFILE, type

```
MYFILE A:PROG1 B:PROG2
```

MYFILE is substituted for %0,

A:PROG1 for %1

B:PROG2 for %2.

The result is the same as if you had typed in each of the following commands, with their parameters.

```
COPY A:PROG1MAC B:PROG2.MAC

TYPE B:PROG2.PRN

TYPE MYFILEBAT
```

The following table illustrates how DOS replaces each of the above parameters.

BATCH FILENAME	PARAMETER1(%0) MYFILE	PARAMETER2 %1) (PROG1)	PARAMETER3 (%2) (PROG2)
MYFILE	MYFILE.BAT	PROG1.MAC	PROG2.MAC
			PROG2.PRN

A point to remember about executing a .BAT file

Remember that the dummy parameter %0 is always replaced by the drive designator (if specified) and the filename of the batch file.

5.17 Input and Output

DOS always assumes that input comes from the keyboard and output goes to the screen. However you can redirect both input and output. Input, for example, can come from a file, rather than the keyboard, and output can be sent to a file or a printer, instead of to the screen. Moreover you can create 'pipes' that allow output from one command to become the input to another.

Redirecting output

Most commands produce output that is sent to your screen. To send this information to a file you use a greater-than sign (>) in your command.

For example

```
DIR
```

sends a list of the directory on the current drive to the screen. But the same command, DIR, plus a filename

for example

```
DIR >LIST1.TXT
```

sends the output to the file, LIST1.TXT.

If the file LIST1.TXT does not exist, DOS creates it. If it already exists, DOS will overwrite what is in the file with the new information.

Another example.

```
DIR >PRN
```

This command sends the output to the printer.

If you want to add your directory, or file, to another file, two greater than signs (>>) can be used to tell DOS to add the output of the command (for example, a directory listing) to the end of the specified file.

For example

```
DIR >>MYFILE
```

would add your directory listing to the file MYFILE. If MYFILE does not exist, DOS creates it.

In some circumstances you will want input to come, not from the keyboard, but from another file. This is made easy in DOS by using a less-than sign (<) in the command.

For example

SORT <NAMES >LIST1

sorts the file NAMES alpabetically and sends the sorted output to a file called LIST1. The input is from the file NAMES, not the keyboard.

A point to remember about redirecting input.

When you use this way of providing input to a program you have to be sure all the program's input is in the file. If the program attempts to obtain more input after the end of the file is reached, DOS cannot supply the input and processing will cease. You can return to the DOS prompt (>) by entering Ctrl-Break.

Filters

A filter is a program or command that reads your input, transforms or modifies it in some way, and then 'outputs' or 'writes' it , perhaps to your screen, or perhaps to your printer. Hence the data is said to have been 'filtered' by the program.

For example

SORT

is a filter that reads input, sorts it and then outputs the results to the screen or printer.

Filters can be put together in many different ways so often you can use a few filters to replace a large number of specific commands.

There are three filters on the DOS diskette.

FIND	This searches for a constant string (word) of text in a file.
MORE	Takes standard keyboard output and displays it, one screen at a time. Then pauses with the message - MORE -
SORT	Sorts text data.

Piping commands

If you want to give more than one command to the system at a time, you can 'pipe' commands to DOS. You may want to do this, for example, if you need to have the output of one program sent as the input to another program.

Piping is done by separating commands with the 'pipe separator', (|).

For example

```
DIR | SORT
```

will produce an alpabetically sorted list of your directory. The '|' pipes the output generated by DIR to be the input of SORT.

To send the result, the sorted directory, to a file, called perhaps READLST.FIL you would type

```
DIR | SORT >READLST.FIL
```

DOS will create the new file, READLST.FIL on your current drive.

To specify a drive other than the current one, type

```
DIR | SORT >B:READLST.FIL
```

In this case the sorted data would be sent to a file called READLST.FIL on drive B.

A pipeline can consist of more than two commands.

For example

```
DIR | SORT | MORE
```

will sort your directory and show it to you one screen at a time, putting -MORE- at the bottom of the screen when there is more output to be seen.

Summary In 5.15 to 5.17 we have looked at

1. Internal and External DOS commands.
2. Command Options.
3. Points to remember about all DOS commands.
4. Batch processing.
5. Creating and executing an AUTOEXEC.BAT file.
6. Creating and executing a .BAT file with replaceable parameters.

7. Redirecting input and output.

8. Using filters and piping.

9. New DOS commands, as listed below.

REM	Adds comment line for batch files.
PAUSE	Suspends the execution of a batch file.
MORE	Displays a screen full of data at a time, then pauses with the message -MORE- when there is more output to be seen.
SORT	Sorts text data.

You will find more information about these commands in the next section (5.18) of this chapter.

5.18 DOS COMMANDS

Command Formats

The following notation indicates how you should enter DOS commands:

* The words in this section shown in capital letters and bold print are called keywords. You can enter these keywords in any combination of uppercase and lowercase letters. DOS converts all keywords to uppercase.

* You supply the text for any items enclosed in angle brackets (< >). For example, you should enter the name of your file when <filename> is shown in the format.

* Items in square brackets ([ ]) are optional. If you wish to include optional information, do not include the square brackets, only the information within the brackets.

* An ellipsis (...) indicates that you may repeat an item as many times as you want.

* You must include all punctuation where shown (with the exception of square brackets), such as commas, equal signs, question marks, colons, or slashes.

* 'd' in the format description refers to the disk drive specification.

* Items separated by '|' mean that you can enter one of the separated items.

For example **ON | OFF**

means that you can either enter ON or OFF, but not both.

* 'Filename' refers to any valid name for a file, including a filename extension.

BREAK (Control Break) Command.

TYPE: Internal

PURPOSE Allows you to tell DOS to check for a control break whenever a program asks DOS to perform any functions, (for example, disk operations).

FORMAT **BREAK [ON|OFF]**

COMMENTS

If you are running an application program that uses Ctrl-break function keys, you will want to turn off the DOS Ctrl-break function so that when you press <Ctrl-Break> you affect your program and not the operating system. Specify **BREAK OFF** to turn off Ctrl-Break and **BREAK ON** when you have finished running your application program and are using DOS.

Entering **BREAK** with no parameters causes DOS to display the current position (on or off) of Ctrl- Break checking.

CHDIR (CD) command.

TYPE Internal

PURPOSE Change the DOS current directory of the specified or current drive, or to display the current directory path of a drive.

FORMAT CHDIR [pathname] or CD [pathname]

COMMENTS

If you do not specify a drive, DOS assumes the current drive.

Examples.

CD

This displays your current directory.

CD \

This changes the current directory of the current drive to its root directory.

CD..

This commands puts you in the parent directory of your current directory.

If your current directory is \TEXT\REPORT and you want to change your path to another directory such as \TEXT\REPORT\ADDRESS, type

CD \TEXT\REPORT\ADDRESS

DOS will put you in the new directory.

CD B:\ACCOUNTS\ANNFIG

This changes drive B's current directory to the path

'root ------- ACCOUNTS -------- ANNFIG'

CHKDSK Command (Check Disk)

TYPE External

PURPOSE Scans the directories and the File Allocation Table on the specified disk drive; reports on the status of the disk and memory.

FORMAT **CHKDSK [d:] <filespec> [/F] [/V]**

COMMENTS

If you specify a filename, **CHKDSK** displays the number of non-contiguous areas occupied by the file or files.

CHKDSK only looks in the current directory for files.

CHKDSK should be run occasionally on each disk to check for errors in the directory. If any errors are found, CHKDSK will display error messages, if any, and then a status report.

The following is an example of a status report.

```
Volume ADVANCE DOS created Jan 30,1984 12:32a

 362496  bytes total disk space
  22528  bytes in 3 hidden files
 194560  bytes in 22 user files
 145408  bytes available on disk

 131072   bytes total memory
 105312   bytes free
```

CHKDSK will not correct the errors found in your directory unless you specify the '/F' (fix) option.

Typing /V causes CHKDSK to display messages while it is running.

You can redirect the output from CHKDSK to a file. Simply type:

```
CHKDSK A: >filename
```

The errors will be sent to the filename specified. Do not use the /F option if you redirect CHKDSK output.

The following errors will be corrected automatically if you specify the /F option:

* Invalid drive specification
* Invalid parameter
* Invalid sub-directory entry
* Cannot CHDIR to <filename>
  Tree past this point not processed
* First cluster number is invalid
  entry truncated
* Allocation error, size adjusted
* Has invalid cluster, file truncated
* Disk error reading FAT
* Disk error writing FAT
* <filename> contains
  non-contiguous blocks
* All specified file(s) are contiguous

You must correct the following errors returned by CHKDSK, even if you specified the /F option

'Incorrect DOS version'

You cannot run CHKDSK on versions of DOS that are not 2.11 or higher.

'Insufficient memory
Processing cannot continue'

There is not enough memory in your machine to process CHKDSK for this disk. You must obtain more memory to run CHKDSK.

'Errors found, F parameter not specified
Corrections will not be written to disk'

You must specify the /F option if you want the errors corrected by CHKDSK.

'Invalid current directory
Processing cannot continue'

Restart the system and re-run CHKDSK.

'Cannot CHDIR to root
Processing cannot continue'

The disk you are checking is bad. Try restarting DOS and RECOVER the disk.

'<filename> is cross linked on cluster'

Make a copy of the file you want to keep, and then delete both files that are cross linked.

'X lost clusters found in y chains
Convert lost chains to files (Y/N)?'

If you respond Y to this prompt, CHKDSK will create a directory entry and a file for you to resolve this problem (files created by CHKDSK are named FILEnnnnnnnn).

CHKDSK will then display:

'X bytes disk space freed'

If you respond N to this prompt and have not specified the /F option, CHKDSK frees the clusters and displays:

'X bytes disk space would be freed'

'Probable non-DOS disk
Continue (Y/N)?'

The disk you are using is a non-DOS disk. You must indicate whether or not you want CHKDSK to continue processing.

'Insufficient room in root directory
Erase files in root and repeat CHKDSK'

CHKDSK cannot process until you delete files in the root directory.

'Unrecoverable error in directory
Convert directory to file (Y/N)?'

If you respond Y to this prompt, CHKDSK will convert the bad directory into a file. You can then fix the directory yourself or delete it.

CLS command. (Clear Screen).

TYPE Internal

PURPOSE Clears the display screen.

FORMAT CLS

COMMENTS

The CLS command causes DOS to send the ANSI escape sequence ESC[2J (which clears your screen) to your keyboard. This commmand will only work if you have the ANSI.SYS device driver loaded.

COPY command.

TYPE Internal

PURPOSE Copies one or more files to another diskette. If you prefer, you can give the copies different names. This command can also copy files on the same diskette, but if you do this, you must give the copies different names (unless different directories are are specified).

FORMAT **COPY <filespec> [filespec] [pathname] [pathname] [/V]**

COMMENTS

If you do not give the second filespec option, the copy will be on the current drive and will have the same name as the original file (first filespec option). If the first filespec is on the current drive and the second filespec is not specified, the COPY will be aborted.

DOS will display the error message:

'File cannot be copied onto itself
0 File(s) copied'

The second option may take three forms:

1. If the second option is a drive designation (d:) only, the original file is copied with the original filename to the designated drive.

2. If the second option is a filename only, the original file is copied to a file on the default drive with the filename specified.

3. If the second option is a full filespec, the original file is copied to a file on the default drive with the filename specified.

The /V option causes DOS to verify that the sectors written on the destination disk are recorded properly. Although there are rarely recording errors when you run COPY, you can verify that critical data has been correctly recorded. This

option causes the COPY command to run more slowly because DOS must check each entry recorded on the disk.

The COPY command also allows file concatenation (joining) while copying. Concatenation is accomplished by simply listing any number of files as options to COPY, separated by +.

For example

```
COPY  A.XYZ + B.TXT + B:C.TXT BIGFILE.CRP
```

This command concatenates files named A.XYZ,B.TXT, and B:C.TXT and places them in the file on the default drive called BIGFILE.CRP.

To combine several files using wild cards into one file, you could type:

```
COPY *.LST COMBIN.PRN
```

This command would take all files with a filename extension of .LST and combine them into a file named COMBIN.PRN.

In the following example, for each file found matching *.LST, that file is combined with the corresponding .REF file. The result is a file with the same filename but with the extension .PRN. Thus, FILE1.LST will be combined with FILE1.REF to form FILE1.PRN; then XYZ.LST with XYZ.REF to form XYZ.PRN; and so on.

```
COPY *.LST + *.REF *.PRN
```

The following COPY command combines all files matching *.LST, then all files matching *.REF, into one file named COMBIN.PRN:

```
COPY *.LST + *.REF COMBIN.PRN
```

Do not enter a concatenation COPY command where one of the source filenames has the same extension as the destination.

For example the following command is an error if ALL.LST already exists:

```
COPY *.LST ALL.LST
```

The error would not be detected, however, until ALL.LST is appended. At this point it could have already been destroyed.

COPY compares the filename of the input file with the filename of the destination. If they are the same, that one input file is skipped, and the error message

"Content of destination lost before copy"

is printed. Further concatenation proceeds normally. This allows "summing" files, as in this example:

```
COPY ALL.LST + *.LST
```

This command appends all *.LST files, except ALL.LST itself, to ALL.LST. This command will not produce an error message and is the correct way to append files using the COPY command.

CTTY command

TYPE Internal

PURPOSE Allows you to change the device from which you issue commands (TTY represents the keyboard).

FORMAT CTTY <device>

COMMENTS

The <device> is the device from which you are giving commands to DOS. This command is useful if you want to change the device on which you are working. The command

CTTY AUX

moves all command I/O (input/output) from the current device (the keyboard) to the AUX port.

The command

CTTY CON

moves I/O back to the original device (here, the keyboard).

DATE command.

TYPE Internal

PURPOSE Allows you to enter or change the date known to the system. This date will be recorded in the directory for any files you create or alter.

You can change the date from your terminal or from a batch file. (DOS does not display a prompt for the date if you use an AUTOEXEC.BAT file, so you may want to include a DATE command in that file.)

FORMAT DATE [<mm>-<dd>-<yy>]

COMMENTS

If you type DATE, DATE will respond with the message:

```
Current date is <mm>-<dd>-<yy>
Enter new date:_
```

Press <ENTER> if you do not want to change the date shown.

You can also type a particular date after the DATE command, as in:

```
DATE 3-9-81
```

In this case, you do not have to answer the 'Enter new date:'prompt.

The new date must be entered using numerals only; letters are not permitted. The allowed options are:

```
<dd> = 1-31
<mm> = 1-12
<yy> = 80-99 or 1980-2099
```

The date, month, and year entries may be separated by hyphens (-) or slashes (/). DOS is programmed to change months and years correctly, whether the month has 31, 30, 29, or 28 days. DOS handles leap years, too!

If the options or separators are not valid, DATE displays the message:

```
Invalid date
Enter new date:_
```

DATE then waits for you to enter a valid date.

You may change the date either from the keyboard or from a batch file. If you use an AUTOEXEC.BAT file when you start the system, it does not prompt you for the date. However, if you wish, you can include a DATE command in the AUTOEXEC.BAT file.

DEL command (Delete) (ERASE)

TYPE Internal

PURPOSE Deletes all files with the designated filespec.

FORMAT **DEL [filespec][pathname]**

COMMENTS

If the filespec is *.*, the prompt 'Are you sure? appears.

If you do wish to erase all the files on the diskette, type Y or y. All files will be deleted as requested. Otherwise type N or n and press the enter key.

Although you can use the wildcard characters ? and * in the filename and in the extension, you should do this with care to avoid erasing multiple files accidentally with the single command.

You can also type **ERASE** for the **DELETE** command.

Examples.

DEL [d:] *.*

deletes all files in the current directory.

DEL [d:] [pathname]

deletes all files in the specified directory.

A>DEL A:DPFILE.TXT

deletes the file DPFILE.TXT from the current directory of drive A.

DIR command (Directory)

TYPE Internal

PURPOSE Lists the files in the directory.

FORMAT **DIR [filespec][pathname][/P][/W]**

COMMENTS

If you just type DIR, all directory entries on the current drive are listed.

If only the drive specification is given DIR d:, all entries on the disk in the specified drive are listed.

If only a filename is entered with no extension DIR **[filename]**, then all files with the designated filename on the disk in the current drive are listed.

If you designate a file specification for example

DIR d:filename.ext

all files with the filename specified on the diskette in the drive specified are listed.

In all cases, files are listed with their size in bytes and with the time and date of their last modification.

The wild card characters ? and * (question mark and asterisk) may be used in the filename option.

It is useful to know that the following DIR commands are equivalent:

COMMAND	EQUIVALENT
DIR	DIR *.*
DIR FILENAME	DIR FILENAME.*
DIR .EXT	DIR *.EXT

Two options may be specified with DIR. The '/P' option selects 'Page Mode'. With '/P', display of the directory pauses after the screen is filled. To resume display of output, press any key.

The '/W ' option selects 'Wide Display.' With '/W,'only filenames are displayed, without other file information. Filenames are displayed five per line.

DISKCOPY command (Copy diskette)

TYPE External

PURPOSE Copies the contents of the disk in the source drive to the disk in the target drive.

FORMAT **DISKCOPY [d:] [d:]**

COMMENTS

The first option you specify is the source drive. The second option is the target drive.

The disk in the target drive must be formatted prior to using DISKCOPY.

You can specify the same drives or you may specify different drives. If the drives designated are the same, a single-drive copy operation is performed. You are prompted to insert the disks at the appropriate times.

DISKCOPY waits for you to press any key before continuing.
After copying, DISKCOPY prompts:

```
'Copy complete
 Copy another (Y/N)?'
```

If you press Y, the next copy is performed on the same drives that you originally specified, after you have been prompted to insert the proper disks.

To end the COPY, press N.

Remember

1. If you omit both options, a single-drive copy operation will be performed on the default drive.

2. If you omit the second option, the default drive will be used as the destination drive.

3. Both disks must have the same number of physical sectors and those sectors must be the same size.

4. Disks that have had a lot of file creation and deletion activity become fragmented, because disk space is not allocated sequentially. The first free sector found is the next sector allocated, regardless of its location on the disk.

   A fragmented disk can cause poor performance due to delays involved in finding, reading, or writing a file. If this is the case, you must use the COPY command, instead of DISKCOPY, to copy your disk and eliminate the fragmentation.

   For example:

   ```
   COPY A:*.* B:
   ```

   copies all files from the disk in drive A: to the disk in drive B:.

5. DISKCOPY automatically determines the number of sides to copy, based on the source drive and disk.

6. If disk errors are encountered during a DISKCOPY, DOS displays:

   ```
   DISK error while reading drive A
   Abort, Ignore, Retry?
   ```

   Refer to Appendix A, Disk Errors, for information on this error message.

EXE2BIN command

TYPE External

PURPOSE Converts .EXE (executable) files to binary format.

FORMAT **EXE2BIN <filespec> [d:][<filename>[<.ext>]]**

COMMENTS

This command is useful only if you want to convert .EXE files to binary format. The file named by filespec is the input file. If no extension is specified, it defaults to .EXE. The input file is converted to .COM file format (memory image of the program) and placed in the output file.

If you do not specify a drive, the drive of the input file will be used.

If you do not specify an output filename, the input filename will be used.

If you do not specify a filename extension in the output filename, the new file will be given an extension of .BIN.

The input file must be in valid .EXE format produced by the linker. The resident, or actual code and data part of the file must be less than 64K. There must be no STACK segment.

Two kinds of conversions are possible, depending on whether the initial CS:IP (Code Segment:Instruction Pointer)is specified in the .EXE file:

1. If CS:IP is not specified in the .EXE file, a pure binary conversion is assumed. If segment fixups are necessary (i.e., the program contains instructions requiring segment relocation), you will be prompted for the fixup value. This value is the absolute segment at which the program is to be loaded. The resulting program will be usable only when loaded at the absolute memory address specified by a user application. The command processor will not be capable of properly loading the program.

2. If CS:IP is specified as 0000:100H, it is assumed that the file is to be run as a .COM file with the location pointer set at 100H by the assembler statement ORG; the first 100H bytes of the file are deleted. No segment fixups are allowed, as .COM files must be segment relocatable; that is, they must assume the entry conditions explained in the Programmers Reference Manual. Once the conversion is complete, you may rename the resulting file with a .COM extension. Then the command processor will be able to load and execute the program in the same way as the .COM programs supplied on your DOS disk.

If CS:IP does not meet either of these criteria, or if it meets the .COM file criterion but has segment fixups, the following message will be displayed:

'File cannot be converted'

This message is also displayed if the file is not a valid executable file.

If EXE2BIN finds an error, one or more of the following error messages will be displayed:

'File not found'

The file is not on the disk specified.

'Insufficient memory'

There is not enough memory to run EXE2BIN.

'File creation error'

EXE2BIN cannot create the output file. Run CHKDSK to determine if the directory is full, or if some other condition caused the error.

'Insufficient disk space'

There is not enough disk space to create a new file.

'Fixups needed - base segment (hex):'

The source (.EXE) file contained information indicating that a load segment is required for the file. Specify the absolute segment address at which the finished module is to be located.

'File cannot be converted'

The input file is not in the correct format.

'WARNING -Read error on EXE file.'

Amount read less than size in header. This is a warning message only.

EXIT command

TYPE Internal

PURPOSE Exits the program COMMAND.COM (the command processor) and returns to a previous level, if one exists.

FORMAT EXIT

COMMENTS

This command can be used when you are running certain applications program and want to start the DOS command processor, then return to your program. For example, to look at a directory on drive B: while running an application program, you must start the command processor by typing

COMMAND

in response to the current drive prompt:

A>COMMAND

You can now type the DIR command and DOS will display the directory for the default disk. When you type EXIT, you return to the previous level (your application program).

FIND command

TYPE External

PURPOSE Searches for a specific string of text in a file or files.

FORMAT **FIND [/V /C /N] <string> [<filename...>]**

COMMENTS

FIND is a filter that takes as options a string and a series of filenames. It will display all the lines that contain a specified string from the files specified in the command line.

If no files are specified, FIND will take the input on the screen and display all lines that contain the specified string.

Options for FIND are:

/V causes FIND to display all lines not containing the specified string.

/C causes FIND to print only the count of lines that contained a match in each of the files.

/N causes each line to be preceded by its relative line number in the file.

The string should be enclosed in quotes.

For Example:

FIND "Fool's Paradise" BOOK1.TXT BOOK2.TXT

displays all lines from BOOK1.TXT and BOOK2.TXT (in that order) that contain the string "Fool's Paradise."

The command

DIR B: | FIND /V "DAT"

causes DOS to display all names of the files on the disk in drive B: which do not contain the string DAT.

Type double quotes around a string that already has quotes in it.

When an error is detected, FIND responds with one of the following error messages:

'Incorrect DOS version'

FIND will only run on versions of DOS that are 2.11 or higher.

'FIND: Invalid number of parameters'

You did not specify a string when issuing the FIND command.

'FIND: Syntax error'

You typed an illegal string when issuing the FIND command.

'FIND: File not found <filename>'

The filename you have specified does not exist or FIND cannot find it.

'FIND: Read error in <filename>'

An error occurred when FIND tried to read the file specified in the command.

'FIND: Invalid parameter <option-name>'

You specified an option that does not exist.

FORMAT command

TYPE External

PURPOSE Formats the disk in the specified drive to accept DOS files.

FORMAT **FORMAT [d]:[/1][/8][/0][/V][/S]**

COMMENTS

This command initializes the directory and file allocation tables. If no drive is specified, the diskette in the current drive is formatted.

Remember the following options must be specified in the order in which they appear above.

The /1 option creates a single sided disk.

The /8 option creates a disk with 8 sectors per track.

The /0 option causes FORMAT to produce an IBM personal Computer DOS version 1.X compatible disk. The /0 option causes FORMAT to reconfigure the directory with an 0E5 hex byte at the start of each entry so that the disk may be used with 1.X versions of IBM PC DOS, as well as MS-DOS 1.25/2.00 and IBM PC DOS 2.00. This option should only be given when needed because it takes a fair amount of time for FORMAT to perform the conversion, and it noticably decreases 1.25 and 2.00 performance on disks with few directory entries.

The /V option causes FORMAT to prompt for a volume label after the disk is formatted.

If the /S option is specified, it must be the last option typed; then FORMAT copies operating system files from the disk in the default drive to the newly formatted disk. The files are copied in the following order:

IO.SYS
MSDOS.SYS
COMMAND.COM

MKDIR command. (MD) (Make Directory)

TYPE Internal

PURPOSE Makes a new directory.

FORMAT **MKDIR <pathname>**

COMMENTS

This command is used to create a hierarchical directory structure. When you are in your root directory, you can create subdirectories by using the MKDIR command.

For example

MKDIR \REPORTS

will create a subdirectory \REPORTS in your root directory. To create a directory named CHAPMAN under \REPORTS, type:

MKDIR \REPORTS\CHAPMAN

MODE

MORE command

TYPE External

PURPOSE Sends output to the screen 25 lines at a time.

FORMAT MORE

COMMENTS

MORE is a filter that reads from standard input and displays one screen of information at a time. The MORE command then pauses and displays the '--MORE--' message at the bottom of your screen.

Pressing the <ENTER> key will display another screen of information. This process continues until all the input data has been read.

The MORE command is useful for viewing a long file one screen at a time.

For example if you type

```
TYPE MYFILE.TXT | MORE
```

DOS will display the file MYFILE.TXT (on the current drive) one screen at a time.

PATH command

TYPE Internal

PURPOSE Sets a command path.

FORMAT PATH [<pathname>[;<pathname>]...]

COMMENTS

This command allows you to tell DOS which directories should be searched for external commands after DOS searches your current directory. The default value is no path.

For example, to tell DOS to search your \TEXT\REPORT\CHAPMAN directory for external commands, type:

PATH \TEXT\REPORT\CHAPMAN

DOS will now search the \TEXT\REPORT\CHAPMAN directory for external commands until you set another path or shut down DOS.

You can tell DOS to search more than one path by specifying several pathnames separated by semicolons.

For example,

PATH \TEXT\REPORT\CHAPMAN;\TEXT\REPORT\COHEN

tells DOS to search the directories specified by the above pathnames to find external commands. DOS searches the pathnames in the order specified in the PATH command.

The command PATH with no options will print the current path.

If you specify PATH ;

DOS will set the NUL path, meaning that only the current directory will be searched for external commands.

PRINT command

TYPE External

PURPOSE Prints a text file on a printer while you are processing other DOS commands (usually called 'background printing' or 'print spooling').

FORMAT **PRINT [[filespec][/T][/C][/S]]...**

COMMENTS

You will use the PRINT command only if you have a printer attached to your computer. The following options are provided with this command:

/T TERMINATE: this option deletes all files in the print queue (those waiting to be printed). A message to this effect will be printed.

/C CANCEL: This option turns on cancel mode. The preceding filespec and all following filespecs will be suspended in the print queue until you type a /P option.

/P PRINT: This option turns on print mode. The preceding filespec and all following filespecs will be added to the print queue until you issue a /C option.

PRINT with no options displays the contents of the print queue on your screen without affecting the queue.

For example

PRINT /T

empties the print queue.

PRINT /T *.ASM

empties the print queue and queues all .ASM files on the current drive.

PRINT A:TEMP1.TST/C A:TEMP2.TST A:TEMP3.TST

removes the three files indicated from the print queue.

PRINT TEMP1.TST /C TEMP2.TST /P TEMP3.TST

removes TEMP1.TST from the queue, and adds TEMP2.TST and TEMP3.TST to the queue.

If an error is detected, PRINT will display one of the following error messages:

'Name of list device [PRN:]'

This prompt appears when PRINT is run the first time. Any current device may be specified and that device then becomes the PRINT output device. As indicated in the brackets, simply pressing <ENTER> results in the device PRN being used.

'List output is not assigned to a device'

This message will be displayed if the "Name of list device" specified to the above prompt is invalid. Subsequent attempts will return the same message until a valid device is specified.

'PRINT queue is full'

There is room for 10 files in the queue. If you attempt to put more than 10 files in the queue, this message will appear on the console.

'PRINT queue is empty'

There are no files in the print queue.

'No files match d:XXXXXXXX.XXX'

A filespec was given for files to add to the queue, but no files match a specification.

NOTE: if there are no files in the queue to match the cancelled filespec, no error message will appear.

'Drive not ready'

If this message occurs when PRINT attempts a disk access, PRINT will keep trying until the drive is ready. Any other error causes the current file to be cancelled. An error message would be output on your printer in such a case.

'All files cancelled'

If the /T (TERMINATE) option is issued, the message "All files cancelled by operator" will be output on your printer. If the current file being printed is cancelled by a /C, the message "File cancelled by operator" will be printed.

PROMPT command

TYPE Internal

PURPOSE Changes the DOS command prompt.

FORMAT **PROMPT [<prompt-text>]**

COMMENTS

This command allows you to change the DOS system prompt (for example, A>). If no text is typed, the prompt will be set to the default prompt, which is the default drive designation.

You can set the prompt to a special prompt, such as the current time, by using the characters indicated below.

The following characters can be used in the prompt command to specify special prompts. They must all be preceded by a dollar sign ($)in the prompt command:

Specify This Character	To Get This Prompt:
$	The '$' character
t	The current time
d	The current date
p	The current directory of the default drive
v	The version number
n	The default drive
g	The '>' character
l	The '<' character
b	The '|' character
_	A CR LF sequence
s	A space (leading only)
h	A backspace
e	ASCII code X'1B' (escape)

For example

PROMPT $n

Sets the normal DOS prompt (>).

PROMPT Time = t_Date = $d

Sets a two-line prompt which prints:

'Time = (current time)
Date = (current date)'

If your Advance has the ANSI SYS driver loaded, then you can use escape sequences in your prompts.

For example:

PROMPT $e[7m$n:$e[m

Sets the prompts in inverse video mode and returns to video mode for other text.

RECOVER command

TYPE External

PURPOSE Recovers a file or an entire disk containing bad sectors.

FORMAT **RECOVER <filename | d:>**

COMMENTS

If a sector on a diskette is bad, you can recover either the file containing that sector (without the bad sector) or the entire diskette (if the bad sector was in the directory).

To recover a particular file, type:

RECOVER <filename>

This will cause DOS to read the file sector by sector and to skip the bad sector(s). When DOS finds the bad sector(s), the sector(s) are marked and DOS will no longer allocate your data to that sector.

To recover a disk, type:

RECOVER <d:>

where d: is the letter of the drive containing the diskette to be recovered.

If there is not enough room in the root directory, RECOVER will print a message and store information about the extra files in the File Allocation Table. You can run RECOVER again to regain these files when there is more room in the root directory.

REM command (Remark)

TYPE Internal

PURPOSE Displays remarks which are on the same line as the REM command in a batch file during execution of that batch file.

FORMAT REM [comment]

COMMENTS

The only separators allowed in the comment are the space, tab, and comma.

For example

```
1: REM  This file checks new disks
2: REM  It is named NEWDISK.BAT
3: PAUSE  Insert new disk in drive B:
4: FORMAT B:/S
5: DIR B:
6: CHKDSK B:
```

REN command (RENAME)

TYPE Internal

PURPOSE Changes the name of the first option (filespec) to the second option (filename).

FORMAT **REN <filespec> <filename>**

COMMENTS

The first option (filespec) must be given a drive designation if the diskette resides in a drive other than the current drive. Any drive designation for the second option (filename) is ignored. The file will remain on the disk where it currently resides.

The wild card characters may be used in either option. All files matching the first filespec are renamed. If wild card characters appear in the second filename, corresponding character positions will not be changed.

For example

the following command changes the names of all files with the .LST extension to similar names with the .PRN extension:

REN *.LST *.PRN

In the next example, REN renames the file ABODE on drive B: to ADOBE:

REN B:ABODE ?D?B?

The file remains on drive B:.

An attempt to rename a filespec to a name already present in the directory will result in the error message "File not found."

RMDIR command (RD) (Remove Directory)

TYPE Internal

PURPOSE Removes a directory from a hierarchical directory structure.

FORMAT **RMDIR <pathname>**

COMMENTS

This command removes a directory that is empty except for the . and .. shorthand symbols.

For example type:

RMDIR \TEXT\REPORT\CHAPMAN

The directory has been deleted from the directory structure.

SET command

TYPE Internal

PURPOSE Sets one string value equivalent to another string for use in later programs.

FORMAT SET [<string=string>]

COMMENTS

This command is meaningful only if you want to set values that will be used by programs you have written. An application program can check all values that have been set with the SET command by issuing SET with no options.

For example,

SET TTY=VT52

sets the TTY value to VT52 until you change it with another SET command.

The SET command can also be used in batch processing. In this way, you can define your replaceable parameters with names instead of numbers. If your batch file contains the statement "LINK %FILE%", you can set the name that DOS will use for that variable with the SET command.

The command SET FILE=DOMORE replaces the %FILE% parameter with the filename DOMORE. Therefore, you do not need to edit each batch file to change the replaceable parameter names. Note that when you use text (instead of numbers) as replaceable parameters, the name must be ended by a percent sign.

SET40 command

TYPE External

PURPOSE Sets display width to 40 columns

FORMAT SET40

SET80 command

TYPE External

PURPOSE Sets display width to 80 columns

FORMAT SET80

SORT command

TYPE External

PURPOSE SORT reads input from your terminal, sorts the data, then writes it to your terminal screen or files.

FORMAT SORT [/R] [/+n]

COMMENTS

SORT can be used, for example, to alphabetize a file by a certain column. There are two options.

/R reverse the sort; that is, sort from Z to A.

/+n sort starting with column n where n is some number. If you do not specify this option, SORT will begin sorting from column 1.

For example

The following command will read the file UNSORT.TXT, reverse the sort, and then write the output to a file named SORT.TXT:

```
SORT /R <UNSORT.TXT >SORT.TXT
```

The following command will pipe the output of the directory command to the SORT filter. The SORT filter will sort the directory listing starting with column 14 (this is the column in the directory listing that contains the file size), then send the output to the console. Thus, the result of this command is a directory sorted by file size:

```
DIR | SORT /+14
```

The command

```
DIR | SORT /+14 | MORE
```

will do the same thing as the command in the previous example, except that the MORE filter will give you a chance to read the sorted directory one screen at a time.

SYS (System) command

TYPE External

PURPOSE Transfers the DOS system files from the diskette in the current drive to the disk in the drive specified by d:.

FORMAT SYS <d>:

COMMENTS

SYS is normally used to update the system or to place the system on a formatted disk which contains no files. An entry for d: is required.

If IO.SYS and MSDOS.SYS are on the target disk, they must take up the same amount of space on the disk as the new system will need.

The target diskette must be completely blank or only have the system files IO.SYS and MSDOS.SYS on it.

The transferred files are copied in the following order:

IO.SYS
MSDOS.SYS

IO.SYS and DOS.SYS are both hidden files that do not appear when the DIR command is executed. COMMAND.COM (the command processor) is not transferred. You must use the COPY command to transfer COMMAND.COM.

If SYS detects an error, one of the following messages will be displayed:

'No room for system on destination disk'

There is not enough room on the destination disk for the IO.SYS and MSDOS.SYS files.

Incompatible system size'

The system files IO.SYS and MSDOS.SYS do not take up the same amount of space on the destination disk as the new system will need.

TIME command

TYPE Internal

PURPOSE Displays and sets the time.

FORMAT **TIME** [<hh>[:<mm>]]

COMMENTS

If the TIME command is entered without any arguments, the following message is displayed:

```
Current time is <hh>:<mm>:<ss>.<cc>
Enter new time:_
```

Press the <ENTER> key if you do not want to change the time shown.

A new time may be given as an option to the TIME command as in:

```
TIME 8:20
```

The new time must be entered using numerals only; letters are not allowed.

The allowed options are:

```
<hh> = 00-24
<mm> = 00-59
```

The hour and minute entries must be separated by colons. You do not have to type the <ss>(seconds) or <cc> (hundredths of seconds) options.

DOS uses the time entered as the new time if the options and separators are valid. If the options or separators are not valid, DOS displays the message:

```
'Invalid time
 Enter new time:_'
```

DOS then waits for you to type a valid time.

TYPE command

TYPE Internal

PURPOSE Displays the contents of the file on the screen.

FORMAT **TYPE <filespec>**

COMMENTS

Use this command to examine the contents of a text file without modifying it. (Use DIR to find the name of a file and EDLIN to alter the contents of a file. EDLIN is discussed in 5.20). The only formatting performed by TYPE is that tabs are expanded to spaces consistent with tab stops every eighth column. Note that a display of binary files (.BIN, .COM or .EXE) causes control characters to be sent to your computer, including bells, form feeds, and escape sequences.

VER command (version)

TYPE Internal

PURPOSE Prints DOS version number.

FORMAT VER

COMMENTS

If you want to know what version of DOS you are using, type

VER <ENTER>

The version number will be displayed on your screen.

VERIFY command

TYPE Internal

PURPOSE Turns the verify option on or off when writing to disk.

FORMAT VERIFY [ON|OFF]

COMMENTS

This command has the same purpose as the /V option in the COPY command. If you want to verify that all files are written correctly to disk, you can use the VERIFY command to tell DOS to verify that your files are intact (no bad sectors, for example). DOS will perform a VERIFY each time you write data to a disk.

You will receive an error message only if DOS was unable to successfully write your data to disk.

VERIFY ON remains in effect until you change it in a program (by a SET VERIFY system call), or until you issue a VERIFY OFF command to DOS.

If you want to know what the current setting of VERIFY is, type

VERIFY <ENTER>

VOL command (Volume)

TYPE Internal

PURPOSE Displays disk volume label, if it exists.

FORMAT **VOL [d:]**

COMMENTS

This command prints the volume label of the disk in drive d:. If no drive is specified, DOS prints the volume label of the disk in the current drive.

If the disk does not have a volume label, VOL displays:

'Volume in drive x has no label'

BATCH PROCESSING COMMANDS

The following commands are called batch processing commands. They can add flexibility and power to your batch programs. The commands discussed are ECHO, FOR, GOTO, IF, and SHIFT.

If you are not writing batch programs, you do not need to read this section.

ECHO command

TYPE Internal

PURPOSE Turns batch echo feature on and off.

FORMAT **ECHO [ON |OFF| message]**

COMMENTS

Normally, commands in a batch file are displayed ("echoed") on the screen when they are seen by the command processor.

ECHO OFF

turns off this feature.

ECHO ON

turns the echo back on.

If ON or OFF are not specified, the current setting is displayed.

ECHO <message> will display the message on the console device.

FOR command

TYPE Internal

PURPOSE Command extension used in batch and interactive file processing.

FORMAT **FOR %%<c> IN <set> DO <command>**

- for batch processing

FOR %<c> IN <set> DO <command>

- for interactive processing

COMMENTS

<c> can be any character except 0,1,2,3,..,9 to avoid confusion with the %0-%9 batch parameters.

<set> is (£<item>*£)

The %%<c> variable is set sequentially to each member of <set>, and then <command> is evaluated. If a member of <set> is an expression involving * and/or ?, then the variable is set to each matching pattern from disk. In this case, only one such <item> may be in the set, and any <item> besides the first is ignored.

Remember the words IN,FOR, and DO must be in upper case.

For example

```
FOR %%f IN ( *.ASM ) DO MASM %%f;

FOR %%f IN (FOO BAR BLECH) DO REM %%f
```

The '%%' is needed so that after batch parameter (%0-%9) processing is done, there is one '%' left. If only '%f' were there, the batch parameter processor would see the '%', look at 'f', decide that '%f' was an error (bad parameter reference) and throw out the '%f', so that the command FOR would never see it. If the FOR is not in a batch file, then only one '%' should be used.

GOTO command

TYPE Internal

PURPOSE Command extension used in batch file processing.

FORMAT **GOTO <label>**

COMMENTS

GOTO causes commands to be taken from the batchfile beginning with the line after the <label> definition. If no label has been defined, the current batch file will terminate.

For example

```
:foo
REM looping...
GOTO foo
```

will produce an infinite sequence of messages:

'REM looping...'

Starting a line in a batch file with ':' causes the line to be ignored by batch processing.

The characters following GOTO define a label, but this procedure may also be used to put in comment lines.

IF command

TYPE Internal

PURPOSE Command extension used in batch file processing.

FORMAT IF <condition> <command>

COMMENTS

The parameter <condition> is one of the following:

ERRORLEVEL <number>

True if and only if the previous program executed by COMMAND had an exit code of <number> or higher.

<string1> == <string2>

True if and only if <string1> and <string2> are identical after parameter substitution. Strings may not have embedded separators.

EXIST <filename>

True if and only if <filename> exists.

NOT <condition>

True if and only if <condition> is false.

The IF statement allows conditional execution of commands. When the <condition> is true,then the <command> is executed. Otherwise, the <command> is ignored.

Note: The words ERRORLEVEL, EXIST and NOT must be in uppercase.

For example

```
IF NOT EXIST \TMP\FOO ECHO Can't find file

IF NOT ERRORLEVEL 3 LINK $1,,;
```

PAUSE command

TYPE Internal

PURPOSE Suspends execution of the batch file.

FORMAT PAUSE [comment]

COMMENTS

During tne execution of a batch file, you may need to change diskettes or perform some other action. PAUSE suspends execution until you press any key, except <Ctrl-Break>.

When the command processor encounters PAUSE, it prints:

'Strike a key when ready . . .'

If you press <Ctrl-Break>, another prompt will be displayed:

'Abort batch job (Y/N)?'

If you type Y in response to this prompt, execution of the remainder of the batch command file will be aborted and control will be returned to the operating system command level.

Therefore, PAUSE can be used to break a batch file into pieces, allowing you to end the batch command file at an intermediate point.

The comment is optional and may be entered on the same line as PAUSE. You may also want to prompt the user of the batch file with some meaningful message when the batch file pauses. For example, you may want to change disks in one of the drives. An optional prompt message may be given in such cases. The comment prompt will be displayed before the "Strike a key" message.

SHIFT command

TYPE Internal

PURPOSE Allows access to more than 10 replaceable parameters in batch file processing.

FORMAT SHIFT

COMMENTS

Usually, command files are limited to handling 10 parameters, %0 through %9. To allow access to more than ten parameters, use SHIFT to change the command line parameters.

For example:

```
if      %0 = "foo"
        %1 = "bar"
        %2 = "name"
        %3...%9 are empty
```

then a SHIFT will result in the following:

```
        %0 = "bar"
        %1 = "name"
        %2...%9 are empty
```

If there are more than 10 parameters given on a command line, those that appear after the 10th (%9) will be shifted one at a time into %9 by successive shifts.

5.19 DOS EDITING AND FUNCTION KEYS

DOS editing keys are used to make corrections to commands and input lines as they are being entered. The DOS editing keys are used to edit within a line. The line editor program (EDLIN), discussed in the next part of this chapter, operates on complete lines within a file.

It is important to remember that some program packages,such as a word processing package, have special editing conventions and the DOS editing keys may not work with them in the way described below. You can also set up special editing rules yourself when using the BASIC Program Editor in BASIC.

The special DOS editing keys deserve particular emphasis because they depart from the way in which most operating systems handle command input. Any line that you enter from the keyboard is kept in an input 'buffer' when you press <ENTER>. The line is then made available for processing to your program. Since the line remains in the input buffer you can use it as a 'template' for editing purpose. The special DOS editing keys operate on that copy of the line.

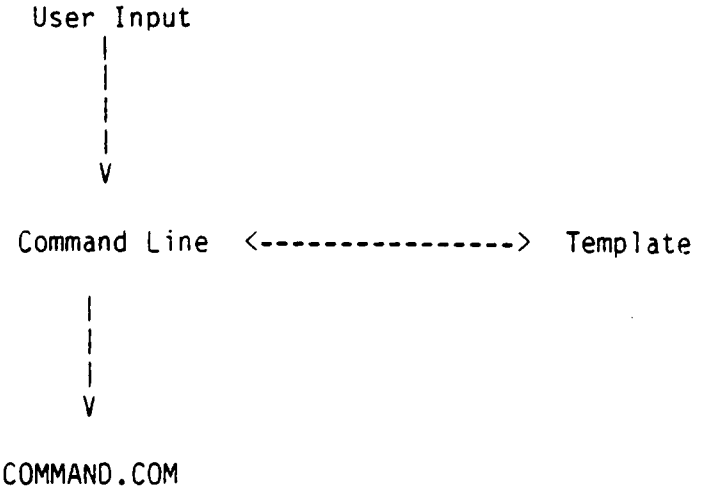
By using the template and the special editing keys, you can take advantage of the following DOS features:

1. A command line can be instantly repeated by pressing two keys.

2. If you make a mistake in the command line, you can edit it and retry without having to retype the entire command line.

3. A command line that is similar to a preceding command line can be edited and executed with a minimum of typing by pressing a special editing key.

The relationship between the command line and the template is shown in the diagram below.



As seen in the diagram, you type a command to DOS on the command line. When you press the <ENTER> key, the command is automatically sent to the command processor (COMMAND.COM) for execution. At the same time, a copy of this command is sent to the template. You can now recall the command or modify it with DOS special editing keys.

The table below contains a complete list of the special editing keys. You will find a full description of these keys and their use in editing your text files in 5.20.

Table 5.1 Special Editing Functions

Key	Editing Function
<F1> or <→>	Copies one character from the template to the command line
<F2>	Copies characters up to the character specified in the template and puts these characters on the command line
<F3>	Copies all remaining characters in the template to the command line
<Del>	Skips over (does not copy) a character in the template
<F4>	Skips over (does not copy) the characters in the template up to the character specified .
<Esc>	Voids the current input; leaves the template unchanged
<Ins>	Enters/exits insert mode
<F5>	Re-edits line with new template
<F6>	Puts a CONTROL-Z (1AH) 'end-of-file character in the new template

For example

If you type the following command

DIR PROG.COM

DOS displays information about the file PROG.COM on your screen. The command line is also saved in the template. To the command, just press two keys: <F3> and <ENTER>.

The repeated command is displayed on the screen as you type, as shown below:

<F3>DIR PROG.COM<ENTER>

Notice that pressing the <F3> key causes the contents of the template to be copied to the command line; pressing <ENTER> causes the command line to be sent to the command processor for execution.

If you want to display information about a file named PROG.ASM, you can use the contents of the template and type:

<F2>C

Typing <F2>C copies all characters from the template to the command line, up to but not including C. DOS displays:

DIR PROG._

Note that the underline is your cursor. Now type:

.ASM

The result is:

DIR PROG.ASM_

The command line DIR PROG.ASM is now in the template and ready to be sent to the command processor for execution. To do this, press <ENTER>.

Now assume that you want to execute the following command:

TYPE PROG.ASM

To do this, type:

TYPE<Ins> <F3><RETURN>

Notice that when you are typing, the characters are entered directly into the command line and overwrite corresponding characters in the template. This automatic replacement feature is turned off when you press the insert key. Thus,the characters "TYPE" replace the characters "DIR " in the template. To insert a space between "TYPE" and "PROG.ASM", you pressed <INS> and then the space bar. Finally, to copy the rest of the template to the command line, you press <F3> and then <ENTER>.

The command TYPE PROG.ASM has been processed by DOS, and the template becomes TYPE PROG.ASM.

If you had misspelled TYPE as BYTE, a command error would have occurred. Still, instead of throwing away the whole command, you could save the misspelled line before you press <ENTER> by creating a new template with the <F5> key:

BYTE PROG.ASM<F5>

You could then edit this erroneous command by typing:

T<F1>P<F3>

The <F1> key copies a single character from the template to the command line. The resulting command line is then the command that you want:

TYPE PROG.ASM

As an alternative, you can use the same template containing

BYTE PROG.ASM

and then use the <Del> and <Ins> keys to achieve the same result:

<Del><Del><F1><Ins>YP<F3>

To illustrate how the command line is affected as you type, examine the keys typed on the left; their effect on the command line is shown on the right:

<Del>	_	Skips over 1st template character
<Del>	_	Skips over 2nd template character
<F1 >	T	Copies 3rd template character
<Ins> YP	TYP	Inserts two characters
<F3>	TYPE PROG.ASM	Copies rest of template

Notice that <Del> does not affect the command line. It affects the template by deleting the first character.

Similarly, <F4> deletes characters in the template, upto but not including a given character.

These special editing keys can add to your effectiveness at the keyboard.

The next section describes control character functions that can also help when you are typing commands.

CONTROL CHARACTER FUNCTIONS

A control character function is a function that affects the command line. You have already learned about <Ctrl-Break> and <Ctrl-Num Lock>. Other control character functions are described below.

Remember that when you type a control character, such as <Ctrl-Break>, you must hold down the control key and then press the Break key.

Table 5.2 Control Character Functions

Control Character	Function
<Ctrl-Break>	Aborts current command.
<Ctrl-H> or < ← >	Removes last character from command line, and erases character from terminal screen.
<Ctrl-Enter>	Inserts physical end-of-line, but does not empty command line. Use the <LINE FEED> key to extend the current logical line beyond the physical limits of one terminal screen.
<Ctrl-P> or <Ctrl-PrtSc>	Toggles terminal output to printer.
<↑-PrtSc>	Screen print
<Ctrl-Num Lock>	Suspends output display on terminal screen. Press any key to resume.
<Esc>	Cancels the current line; empties the command line; and then outputs a back slash (\), carriage return, and line feed. The template used by the special editing commands is not affected.

5.20 THE LINE EDITOR (EDLIN)

EDLIN is the line editor program which you can use to create, change, and display files, whether they are source program or text files.

You can use EDLIN to:

Create new source files and save them.

Update existing files and save both the updated and original files.

Delete, edit, insert, and display lines.

Search for, delete, or replace text within one or more lines.

The text in files created or edited by EDLIN is divided into lines, each up to 253 characters long. Line numbers are generated and displayed by EDLIN during the editing process, but are not actually present in the saved file.

When you insert lines, all line numbers following the inserted text advance automatically by the number of lines being inserted. When you delete lines in a file, all line numbers following the deleted text decrease automatically by the number of lines deleted. As a result, lines are always numbered consecutively in your file.

Starting EDLIN

To start EDLIN, type:

EDLIN <filename>

If you are creating a new file, the <filename> should be the name of the file you wish to create. If EDLIN does not find this file on a drive, EDLIN will create a new file with the name you specify. The following message and prompt will be displayed:

```
New file
*_
```

Notice that the prompt for EDLIN is an asterisk (*).

You can now type lines of text into your new file. To begin entering text, you must enter an I (Insert) command to insert lines. The I command is discussed later in this chapter.

If you want to edit an existing file, <filename> should be the name of the file you want to edit. When EDLIN finds the file you specify on the designated or default drive, the file will be loaded into memory. If the entire file can be loaded, EDLIN will display the following message on your screen:

```
End of input file
*
```

You can then edit the file using EDLIN editing commands.

If the file is too large to be loaded into memory, EDLIN will load lines until memory is 3/4 full, then display the * prompt. You can then edit the portion of the file that is in memory.

To edit the remainder of the file, you must save some of the edited lines on disk to free memory; then EDLIN can load the unedited lines from disk into memory. Refer to the Write and Append commands in this chapter for the procedure.

When you complete the editing session, you can save the original and the updated (new) files by using the End command. The End command is discussed in this chapter in the section "EDLIN Commands".

The original file is renamed with an extension of .BAK, and the new file has the filename and extension you specify in the EDLIN command. The original .BAK file will not be erased until the end of the editing session, or until disk space is needed by the editor (EDLIN).

Do not try to edit a file with a filename extension of .BAK because EDLIN assumes that any .BAK file is a backup file. If you do have to edit such a file, rename the file with another extension (using the DOS RENAME command discussed in 5.18), then start EDLIN and specify the new <filename>.

SPECIAL DOS EDITING KEYS

The special editing keys and template discussed in part 5.19 of this chapter can be used to edit your text files. The table below summarises the use of these keys. Detailed descriptions follow the table.

Table 5.3 Special Editing Keys

Function	Key	Description
Copy one character	<F1> or →	Copies one character from the template to the new line.
Copy up to character	<F2>	Copies all characters from the template to the new line, up to the character specified.
Copy template	<F3>	Copies all remaining characters in the template to the screen.
Skip one character	<Del>	Does not copy (skips over) a character.
Skip up to character	<F4>	Does not copy (skips over) the characters in tne template, up to the character specified.
Quit input	<Esc>	Voids the current input; leaves the template unchanged.
Insert mode	<Ins>	Enters/exits insert mode.
New template	<F5>	Makes the new line the new template.

EDLIN KEY <F1>

PURPOSE Copies one character from the template to the command line.

COMMENTS

Pressing the <F1> key copies one character from the template to the command line. When the <F1> key is pressed, one character is inserted in the command line and insert mode is automatically turned off.

For example:

Assume that the screen shows:

```
1:*This is a sample file.
1:*_
```

At the beginning of the editing session, the cursor (indicated by the underline) is placed at the beginning of the line. Pressing the <F1> key copies the first character (T) to the second of the two lines displayed:

```
     1:*This is a sample file
<F1> 1:*T_
```

Each time the <F1> key is pressed, one more character appears:

```
<F1> 1:*Th_
<F1> 1:*Thi_
<F1> 1:*This_
```

EDLIN KEY <F2>

PURPOSE Copies multiple characters up to a given character.

COMMENTS

Pressing the <F2> key copies all characters up to a given character from the template to the command line. The given character is the next character typed after <F2>; it is not copied or displayed on the screen. Pressing the <F2> key causes the cursor to move to the single character that is specified in the command. If the template does not contain the specified character, nothing is copied.

Pressing <F2> also automatically turns off insert mode.

Example:

Assume that the screen shows:

```
1:*This is a sample file.
1:*_
```

At the beginning of the editing session, the cursor (indicated by the underline) is positioned at the beginning of the line. Pressing the <F2> key copies all characters up to the character specified immediately after the <F2> key.

```
          1:*This is a sample file
<F2>p     1:*This is a sam_
```

EDLIN KEY <F3>

PURPOSE Copies template to command line.

COMMENTS

Pressing the <F3> key copies all remaining characters from the template to the command line. Regardless of the cursor position at the time the <F3> key is pressed, the rest of the line appears, and the cursor is positioned after the last character on the line.

Example:

Assume that the screen shows:

```
1:*This is a sample file.
1:*_
```

At the beginning of the editing session, the cursor (indicated by the underline) is positioned at the beginning of the line. Pressing the <F3> key copies all characters from the template (shown in the upper line displayed) to the line with the cursor (the lower line displayed):

```
     1:*This is a sample file   (template)
<F3> 1:*This is a sample file._ (command line)
```

Also, insert mode is automatically turned off.

EDLIN KEY <Del>

PURPOSE Skips over one character in the template.

COMMENTS

Pressing the <Del> key skips over one character in the template. Each time you press the <Del> key, one character is not copied from the template. The action of the <Del> key is similar to the <F1> key, except that <Del> skips a character in the template rather than copying it to the command line.

Example:

Assume that the screen shows:

```
1:*This is a sample file.
1:*_
```

At the beginning of the editing session, the cursor (indicated by the underline) is positioned at the beginning of the line. Pressing the <Del> key skips over the first character (T).

```
1:*This is a sample file
<Del> 1:*_
```

The cursor position does not change and only the template is affected. To see how much of the line has been skipped over, press the <F3> key, which moves the cursor beyond the last character of the line.

```
      1:*This is a sample file.
<Del> 1:*
 <F3> 1:*his is a sample file._
```

EDLIN KEY <F4>

PURPOSE Skips multiple characters in the template up to the specified character.

COMMENTS

Pressing the <F4> key skips over all characters up to a given character in the template. This character is not copied and is not shown on the screen. If the template does not contain the specified character, nothing is skipped over. The action of the <F4> key is similar to the <F2> key, except that <F4> skips over characters in the template rather than copying them to the command line.

Example:

Assume that the screen shows:

```
1:*This is a sample file.
1:*_
```

At the beginning of the editing session, the cursor (indicated by the underline) is positioned at the beginning of the line. Pressing the <F4> key skips over all the characters in the template up to the character pressed after the <F4> key:

```
     1:*This is a sample file
<F4>p 1:*_
```

The cursor position does not change. To see how much of the line has been skipped over, press the <F3> key to copy the template. This moves the cursor beyond the last character of the line:

```
      1:*This is a sample file:
<F4>p 1:*
<F3>  1:*ple file._
```

EDLIN KEY <Esc>

PURPOSE Quits input and empties the command line.

COMMENTS

Pressing the <Esc> key empties the command line, but it leaves the template unchanged. <Esc> also prints a back slash (\), carriage return, and line feed, and turns insert mode off. The cursor (indicated by the underline) is positioned at the beginning of the line. Pressing the <F3> key copies the template to the command line and the command line appears as it was before <Esc> was pressed.

For example: Assume that the screen shows:

```
1:*This is a sample file.
1:*_
```

At the beginning of the editing session, the cursor (indicated by the underline) is positioned at the beginning of the line. Assume that you want to replace the line with "Sample File:"

```
1:*This is a sample file.
1:*Sample File_
```

To cancel the line you just entered (Sample File), and to keep "This is a sample file.", press <Esc>. Notice that a backslash appears on the Sample File line to tell you it has been cancelled.

```
      1:*This is a sample file.
<Esc> 1:*Sample File\
      1:_
```

Press <Enter> to keep the original line, or to perform any other editing functions. If <F3> is pressed, the original template is copied to the command line:

```
<F3> 1: This is a sample file._
```

EDLIN KEY <Ins>

PURPOSE Enters/exits insert mode.

COMMENTS

Pressing the <Ins> key causes EDLIN to enter and exit insert mode. The current cursor position in the template is not changed. The cursor does move as each character is inserted. However, when you have finished inserting characters, the cursor will be positioned at the same character as it was before the insertion began. Thus, characters are inserted in front of the character to which the cursor points.

Example:

Assume that the screen shows:

```
1:*This is a sample file.
1:*_
```

At the beginning of the editing session, the cursor (indicated by the underline) is positioned at the beginning of the line.

Assume that you press the <F2> and f keys:

```
      1:*This is a sample file
<F2>f 1:*This is a sample _
```

Now press the <Ins> key and insert the characters "edit" and a space:

```
          1:*This is a sample file.
<F2>f     1:*This is a sample _
<Ins>edit 1:*This is a sample edit _
```

If you now press the <F3> key, the rest of the template is copied to the line:

```
      1:*This is a sample edit _
<F3> 1:*This is a sample edit file._
```

If you pressed the <Enter> key, the remainder of the template would be truncated, and the command line would end at the end of the insert:

```
<Ins>edit <Enter> 1:*This is a sample edit _
```

To exit insert mode, simply press the <Ins> key again.

EDLIN KEY <F5>

PURPOSE Creates a new template.

COMMENTS

Pressing the <F5> key copies the current command line to the template. The contents of the old template are deleted. Pressing <F5> outputs an @ ("at sign" character), a carriage return, and a line feed. The command line is also emptied and insert mode is turned off.

NOTE:

<F5> performs the same function as the <Esc> key, except that the template is changed and an @ ("at sign"character) is printed instead of a \ (backslash).

For example:

Assume that the screen shows:

```
1:*This is a sample file.
1:*_
```

At the beginning of the editing session, the cursor (indicated by the underline) is positioned at the beginning of the line.

Assume that you enter <F2>m, <Ins>lary,<Ins> tax, and then <F3>:

```
                1:*This is a sample file.
 <F2>m          1:*This is a sa
<Ins>lary       1:*This is a salary_
<Ins> tax       1:*This is a salary tax_
  <F3>          1:*This is a salary tax file._
```

At this point, assume that you want this line to be the new template, so you press the <F5> key:

```
<F5>1:*This is a salary tax file.@
```

The @ indicates that this new line is now the new template. Additional editing can be done using the new template.

COMMAND INFORMATION

EDLIN commands perform editing functions on lines of text. Points to remember;

1. Pathnames are acceptable as options to commands. For example, typing EDLIN \TEXTS.84\REPORT.AUG\CHAPMAN \SALES will allow you to edit the SALES file in the subdirectory CHAPMAN.

2. You can reference line numbers relative to the current line (the line with the asterisk). Use a minus sign with a number to indicate lines before the current line. Use a plus sign with a number to indicate lines after the current line.

   Example:

   -10,+10L

   This command lists 10 lines before the current line, the current line, and 10 lines after the current line.

3. Multiple commands may be issued on one command line. When you issue a command to edit a single line using a line number (<line>), a semicolon must separate commands on the line. Otherwise, one command may follow another without any special separators. In the case of a Search or Replace command, the <string> may be ended by a <F5> instead of an <ENTER>.

   For example:

   The following command line edits line 15 and then displays lines 10 through 20 on the screen.

   15;-5,+5L

   The command line in the next example searches for "This string" and then displays 5 lines before and 5 lines after the line containing the matched string. If the search fails, then the displayed lines are those line numbers relative to the current line.

   SThis string<F6>-5,+5L

4. You can type EDLIN commands with or without a space between the line number and command. For example, to delete line 6, the command 6D is the same as 6 D.

5. It is possible to insert a control character (such as Ctrl-Break) into text by using the quote character Ctrl-V before it while in insert mode. Ctrl-V tells DOS to recognize the next capital letter typed as a control character. It is also possible to use a control character in any of the string arguments of Search or Replace by using the special quote character. For example:

   S<Ctrl-V>Z
   will find the first occurrence
   of CONTROL-Z in a file

   R<CONTROL-V>Z<CONTROL-Z>foo
   will replace all occurrences
   of CONTROL-Z in a file by foo

   S<CONTROL-V>C<CONTROL-Z>bar
   will replace all occurrences
   of CONTROL-C by bar

It is possible to insert CONTROL-V into the text by typing CONTROL-V-V.

6. The <F6> key ordinarily tells EDLIN, "This is the end of the file." If you have CONTROL-Z characters elsewhere in your file, you must tell EDLIN that these other control characters do not mean end-of-file. Use the /B option to tell EDLIN to ignore any CONTROL-Z characters in the file and to show you the entire file.

SUMMARY of EDLIN COMMANDS.

The EDLIN commands are summarized in the following table. They are also described in further detail following the description of command options.

EDLIN Commands

Command	Purpose
<line>	Edits line no.
A	Appends lines
C	Copies lines
D	Deletes lines
E	Ends editing
I	Inserts lines
L	Lists text
M	Moves lines
P	Pages text
Q	Quits editing
R	Replaces lines
S	Searches text
T	Transfers text
W	Writes lines

EDLIN COMMAND OPTIONS

Some EDLIN commands accept one or more options. The effect of a command option varies, depending on with which command it is used. The following list describes each option.

<line> <line> indicates a line number that you type. Line numbers must be separated by a comma or a space from other line numbers, other options, and from the command.

<line> may be specified one of three ways:

Number Any number less than 65534. If a number larger than the largest existing line number is specified, then <line> means the line after the last line number.

Period (.) If a period is specified for <line>, then <line> means the current line number. The current line is the last line edited, and is not necessarily the last line displayed. The current line is marked on your screen by an asterisk (*) between the line number and the first character.

Pound (£) The pound sign indicates the line after the last line number. If you specify £ for <line>, this has the same effect as specifying a number larger than the last line number.

<ENTER> A carriage return entered without any of the <line> specifiers listed above directs EDLIN to use a default value appropriate to the command.

? The question mark option directs EDLIN to ask you if the correct string has been found. The question mark is used only with the Replace and Search commands. Before continuing, EDLIN waits for either a Y or <ENTER> for a yes response, or for any other key for a no response.

<string> <string> represents text to be found, to be

replaced, or to replace other text. The <string> option is used only with the Search and Replace commands. Each <string> must be ended by a <CONTROL-Z> or a <RETURN> (see the Replace command for details). No spaces should be left between strings or between a string and its command letter, unless you want those spaces to be part of the string.

EDLIN EDITING COMMANDS

The following pages describe EDLIN editing commands.

EDLIN (A)ppend

PURPOSE Adds the specified number of lines from disk to the file being edited in memory. The lines are added at the end of lines that are currently in memory.

FORMAT [<n>]A

COMMENTS

This command is meaningful only if the file being edited is too large to fit into memory. As many lines as possible are read into memory for editing when you start EDLIN.

To edit the remainder of the file that will not fit into memory, lines that have already been edited must be written to disk. Then you can load unedited lines from disk into memory with the Append command. Refer to the Write command in this chapter for information on how to write edited lines to disk.

NOTE:

1. If you do not specify the number of lines to append, lines will be appended to memory until available memory is 3/4 full. No action will be taken if available memory is already 3/4 full.

2. The message "End of input file" is displayed when the Append command has read tne last line of the file into memory.

EDLIN (C)opy

PURPOSE Copies a range of lines to a specified line number. The lines can be copied as many times as you want by using the <count> option.

FORMAT [<line>],[<line>],<line>,[<count>]C

COMMENTS

If you do not specify a number in <count>, EDLIN copies the lines one time. If the first or the second <line> are omitted, the default is the current line. The file is renumbered automatically after the copy.

The line numbers must not overlap or you will get an "Entry error" message. The following, for example,

3,20,15C

would result in an error message.

For example:

Assume that the following file exists and is ready to edit:

```
1: This is a sample file
2: used to show copying lines.
3: See what happens when you use
4: the Copy command
5: (the C command)
6: to copy text in your file.
```

You can copy this entire block of text by issuing the following command:

1,6,7C

The result is:

```
1: This is a sample file
2: used to show copying lines.
3: See what happens when you use
4: the Copy command
5: (the C command)
6: to copy text in your file.
7: This is a sample file
8: used to show copying lines.
9: See what happens when you use
10: the Copy command
11: (the C command)
12: to copy text in your file.
```

If you want to place the text within other text, the third

<line> option should specify the line before which you want the copied text to appear. For example, assume that you want to copy lines and insert them within the following file:

```
 1: This is a sample file
 2: used to show copying lines.
 3: See what happens when you use
 4: the Copy command
 5: (the C command)
 6: to copy text in your file.
 7: You can also use COPY
 8: to copy lines of text
 9: to the middle of your file.
10: End of sample file.
```

he command 3,6,9C results in the following file:

```
 1: This is a sample file
 2: used to show copying lines.
 3: See what happens when you use
 4: the Copy command
 5: (the C command)
 6: to copy text in your file.
 7: You can also use COPY
 8: to copy lines of text
 9: to the middle of your file.
10: See what happens when you use
11: the Copy command
12: (the C command)
13: to copy text in your file.
14: End of sample file.
```

EDLIN (D)elete

PURPOSE Deletes a specified range of lines in a file.

FORMAT [<line>][,<line>]D

COMMENTS

If the first <line> is omitted, that option will default to the current line (the line with the asterisk next to the line number). If the second <line> is omitted, then just the first <line> will be deleted. When lines have been deleted, the line immediately after the deleted section becomes the current line and has the same line number as the first deleted <line> had before the deletion occurred.

For example;

Assume that the following file exists and is ready to edit:

```
 1: This is a sample file
 2: used to show dynamic line numbers.
 3: See what happens when you use
 4: Delete and Insert
  .
  .
  .
25: (the D and I commands)
26: to edit the text
27:*in your file.
```

To delete multiple lines, type <line>,<line>D:

```
5,24D
```

The result is:

```
1: This is a sample file
2: used to show dynamic line numbers.
3: See what happens when you use
4: Delete and Insert
5: (the D and I commands)
6: to edit text
7:*in your file.
```

To delete a single line, type:

```
6D
```

The result is:

```
1: This is a sample file
2: used to show dynamic line numbers.
3: See what happens when you use
4: Delete and Insert
5: (the D and I commands)
6:*in your file.
```

Next, delete a range of lines from the following file:

```
1: This is a sample file
2: used to show dynamic line numbers.
3:*See what happens when you use
4: Delete and Insert
5: (the D and I commands)
6: to edit text
7: in your file.
```

To delete a range of lines beginning with the current line, type:

```
,6D
```

The result is:

```
1: This is a sample file
2: used to show dynamic line numbers.
3:*in your file.
```

Notice that the lines are automatically renumbered.

EDLIN <line> Edit

PURPOSE Edits line of text.

FORMAT [<line>]

COMMENTS

When a line number is typed, EDLIN displays the line number and text; then, on the line below, EDLIN reprints the line number. The line is now ready for editing. You may use any of the EDLIN editing commands to edit the line. The existing text of the line serves as the template until the <ENTER> key is pressed.

If no line number is typed (that is, if only the <ENTER> key is pressed), the line after the current line (marked with an asterisk (*))is edited. If no changes to the current line are needed and the cursor is at the beginning or end of the line, press the <ENTER> key to accept the line as is.

WARNING

If the <ENTER> key is pressed while the cursor is in the middle of the line, the remainder of the line is deleted.

For example:

Assume that the following file exists and is ready to edit:

```
1: This is a sample file.
2: used to show
3: the editing of line
4:*four.
```

To edit line 4, type:

```
4
```

The contents of the line are displayed with a cursor below the line:

```
4:* four.
4:*_
```

Now, using the <F3> special editing key, type:

```
<Ins>number 4: number_
<F3><ENTER> 4: number four.
               5:*_
```

EDLIN (E)nd

PURPOSE Ends the editing session.

FORMAT E

COMMENTS

This command saves the edited file on disk, renames the original input file <filename>.BAK, and then exits EDLIN. If the file was created during the editing session, no .BAK file is created.

The E command takes no options. Therefore, you cannot tell EDLIN on which drive to save the file. The drive you want to save the file on must be selected when the editing session is started. If the drive is not selected when EDLIN is started, the file will be saved on the disk in the default drive. It will still be possible to COPY the file to a different drive using the DOS COPY command.

You must be sure that the disk contains enough free space for the entire file. If the disk does not contain enough free space, the write will be aborted and the edited file lost, although part of the file might be written out to the disk.

For example:

E<ENTER>

After execution of the E command, the DOS default drive prompt (for example, A>) is displayed.

EDLIN (I)nsert

PURPOSE Inserts text immediately before the specified <line>.

FORMAT [<line>]I

COMMENTS

If you are creating a new file, the I command must be given before text can be typed (inserted). Text begins with line number 1. Successive line numbers appear automatically each time <ENTER> is pressed.

EDLIN remains in insert mode until <Ctrl-Break>is typed. When the insert is completed and insert mode has been exited, the line immediately following the inserted lines becomes the current line. All line numbers following the inserted section are incremented by the number of lines inserted.

If <line> is not specified, the default will be the current line number and the lines will be inserted immediately before the current line.If <line> is any number larger than the last line number, or if a pound sign (£) is specified as <line>, the inserted lines will be appended to the end of the file. In this case,the last line inserted will become the current line.

For example:

Assume that the following file exists and is ready to edit:

```
1: This is a sample file
2: used to show dynamic line numbers.
3: See what happens when you use
4: Delete and Insert
5: (the D and I commands)
6: to edit text
7:*in your file.
```

To insert text before a specific line that is not the current line, type <line>I:

```
7I
```

The result is:

```
7:_
```

Now, type the new text for line 7:

```
7: and renumber lines
```

Then to end the insertion, press <F6> on the next line:

```
8: <F6>
```

Now type L to list the file. The result is:

```
1: This is a sample file
2: used to show dynamic line numbers.
3: See what happens when you use
4: Delete and Insert
5: (the D and I commands)
6: to edit text
7. and renumber lines
8:*in your file.
```

To insert lines immediately before the current line, type:

```
I
```

The result is:

```
8: _
```

Now, insert the following text and terminate with a <F6> on the next line:

```
8: so they are consecutive
9: <F6>
```

Now to list the file and see the result, type L:

The result is:

```
1: This is a sample file
2: used to show dynamic line numbers.
3: See what happens when you use
4: Delete and Insert
5: (the D and I commands)
6: to edit text
7: and renumber lines
8: so they are consecutive
9:*in your file.
```

To append new lines to the end of the file, type:

```
10I
```

This produces the following:

```
10: _
```

Now, type the following new lines:

```
10: The insert command can place new lines
11: in the file; there's no problem
12: because the line numbers are dynamic;
13: they'll go all the way to 65533.
```

End the insertion by pressing <F6> on line 14. The new lines will appear at the end of all previous lines in the file. Now type the List command, L:

The result is:

```
 1: This is a sample file
 2: used to show dynamic line numbers.
 3: See what happens when you use
 4: Delete and Insert
 5: (the D and I commands)
 6: to edit text
 7: and renumber lines
 8: so they are consecutive
 9: in your file.
10: The insert command can place new lines
11: in the file; there's no problem
12: because the line numbers are dynamic;
13: they'll go all the way to 65533.
```

EDLIN (L)ist

PURPOSE Lists a range of lines, including the two lines specified.

FORMAT [<line>][,<line>]L

COMMENTS

Default values are provided if either one or both of the options are omitted. If you omit the first option, as in:

,<line>L

the display will start 11 lines before the current line and end with the specified <line>. The beginning comma is required to indicate the omitted first option.

NOTE: If the specified <line> is more than 11 lines before the current line, the display will be the same as if you omitted both options.

If you omit the second option, as in

<line>L

23 lines will be displayed, starting with the specified <line>.

If you omit both parameters, as in

L

23 lines will be displayed--the 11 lines before the current line, the current line, and the 11 lines after the current line. If there are less than 11 lines before the current line, more than 11 lines after the current line will be displayed to make a total of 23 lines.

For example:

Assume that the following file exists and is ready to edit:

```
 1: This is a sample file
 2: used to show dynamic line numbers.
 3: See what happens when you use
 4: Delete and Insert
 5: (the D and I commands)
  .
  .
  .
15:*The current line contains an asterisk.
  .
```

```
    .
    .
26: to edit text
27: in your file.
```

To list a range of lines without reference to the current line, type <line>,<line>L: 2,5L

The result is:

```
2: used to show dynamic line numbers.
3: See what happens when you use
4: Delete and Insert
5: (the D and I commands)
```

To list a range of lines beginning with the current line, type ,<line> L:

```
,26L
```

The result is:

```
15:*The current line contains an asterisk.
    .
    .
    .
26: to edit text
```

To list a range of 23 lines centered around the current line,

type only L

The result is:

```
 4: Delete and Insert
 5: (the D and I commands)
    .
    .
    .
13: The current line is listed in the middle.
14: The current line remains unchanged.
15:*The current line contains an asterisk.
    .
    .
    .
26: to edit text.
```

EDLIN (M)ove

PURPOSE Moves a range of text to the line specified.

FORMAT [<line>],[<line>],<line>M

COMMENTS

Use the Move command to move a block of text (from the first <line> to the second <line>) to another location in the file. The lines are renumbered according to the direction of the move. For example,

,+25,100M

moves the text from the current line plus 25 lines to line 100. If the line numbers overlap, EDLIN will display an "Entry error" message.

To move lines 20-30 to line 100, type:

20,30,100M

EDLIN (P)age

PURPOSE Pages through a file 23 lines at a time.

FORMAT [<line>][,<line>]P

COMMENTS

If the first <line> is omitted, that number will default to the current line plus one. If the second <line> is omitted, 23 lines will be listed. The new current line becomes the last line displayed and is marked with an asterisk.

EDLIN (Q)uit

PURPOSE Quits the editing session, does not save any editing changes, and exits to the DOS operating system.

FORMAT Q

COMMENTS

EDLIN prompts you to make sure you don't want to save the changes.

Type Y if you want to quit the editing session.

No editing changes are saved and no .BAK file is created. Refer to the End command in this chapter for information about the .BAK file.

Type N or any other character except Y if you want to continue the editing session.

NOTE:

When started, EDLIN erases any previous copy of the file with an extension of .BAK to make room to save the new copy. If you reply Y to the Abort edit (Y/N)? message, your previous backup copy will no longer exist.

For example:

```
Example: Q
         Abort edit (Y/N)?Y<RETURN>
       A>_
```

EDLIN (R)eplace

PURPOSE Replaces all occurrences of a string of text in the specified range with a different string of text or blanks.

FORMAT [<line>][,<line>][?]R<string1><F6><string2>

COMMENTS

As each occurrence of <string1> is found, it is replaced by <string2>. Each line in which a replacement occurs will be displayed. If a line contains two or more replacements of <string1> with <string2>, then the line will be displayed once for each occurrence. When all occurrences of <string1> in the specified range are replaced by <string2>, the R command terminates and the asterisk prompt reappears.

If a second string is to be given as a replacement, then <string1> must be separated from <string2> with a <F6>. <String2> must also be ended with a <F6><ENTER> combination or with a simple <ENTER>.

If <string1> is omitted, then Replace will take the old <string1> as its value. If there is no old <string1>, i.e., this is the first replace done, then the replacement process will be terminated immediately. If <string2> is omitted, then <string1> may be ended with an <ENTER>. If the first <line> is omitted in the range argument (as in ,<line>) then the first <line> will default to the line after the current line. If the second <line> is omitted (as in <line> or <line>,), the second <line> will default to £. Therefore, this is the same as <line>,£. Remember that £ indicates the line after the last line of the file.

If <string1> is ended with <F6> and there is no <string2>, <string2> will be taken as an empty string and will become the new replace string.

For example,

R<string2><F6><ENTER>

will delete occurrences of <string1>, but

R<string1><return> and
R<ENTER>

will replace <string1> by the old <string2> and the old <string1> with the old <string2>, respectively. Note that "old" here refers to a previous string specified either in a Search or a Replace command.

If the question mark (?) option is given, the Replace command will stop at each line with a string that matches <string1>, display the line with <string2> in place, and then display the prompt O.K.?. If you press Y or the <ENTER> key, then <string2> will replace <string1>, and the next occurrence of <string1> will be found. Again, the O.K.? prompt will be displayed. This process will continue until the end of the range or until the end of the file. After the last occurrence of <string1> is found, EDLIN displays the asterisk prompt.

If you press any key besides Y or <ENTER> after the O.K.? prompt, the <string1> will be left as it was in the line, and Replace will go to the next occurrence of <string1>. If <string1> occurs more than once in a line, each occurrence of <string1> will be replaced individually, and the O.K.? prompt will be displayed after each replacement. In this way, only the desired <string1> will be replaced, and you can prevent unwanted substitutions.

For example:

Assume that the following file exists and is ready for editing:

```
 1: This is a sample file
 2: used to show dynamic line numbers.
 3: See what happens when you use
 4: Delete and Insert
 5: (the D and I commands)
 6: to edit text
 7: in your file.
 8: The insert command can place new lines
 9: in the file; there's no problem
10: because the line numbers are dynamic;
11: they'll go all the way to 65533.
```

To replace all occurrences of <string1> with <string2> in a specified range, type: 2,12 Rand<F6>or<ENTER>

The result is:

```
 4: Delete or Insert
 5: (the D or I commors)
 8: The insert commor can place new lines
```

Note that in the above replacement, some unwanted substitutions have occurred. To avoid these and to confirm each replacement, the same original file can be used with a slightly different command.

In the next example, to replace only certain occurrences of the first <string> with the second <string>, type:

```
2? Rand<F6>or<ENTER>
```

The result is:

```
4: Delete or Insert
O.K.? Y
5: (The D or I commands)
O.K.? Y
5: (The D or I commors)
O.K.? N
8: The insert commor can place new lines
O.K.? N
*_
```

Now, type the List command (L) to see the result of all these changes:

```
   .
   .
4: Delete or Insert
5: (The D or I commands)
   .
8: The insert command can place new lines
   .
   .
```

EDLIN (S)earch

PURPOSE Searches the specified range of lines for a specified string of text.

FORMAT [<line>][,<line>][?]S<string><ENTER>

COMMENTS

The <string> must be ended with an <ENTER>. The first line that matches <string> is displayed and becomes the current line. If the question mark option is not specified, the Search command will terminate when a match is found. If no line contains a match for <string>, the message "Not found" will be displayed.

If the question mark option (?) is included in the command, EDLIN will display the first line with a matching string; it will then prompt you with the message O.K.? If you press either the Y or <ENTER> key, the line will become the current line and the search will terminate. If you press any other key, the search will continue until another match is found, or until all lines have been searched (and the Not found message is displayed).

If the first <line> is omitted (as in ,<line>S<string>), the first <line> will default to the line after the current line. If the second <line> is omitted (as in <line> S<string> or <line>, S<string>), the second <line> will default to £ (line after last line of file), which is the same as <line>,£ S<string>. If <string> is omitted, Search will take the old string if there is one. (Note that "old" here refers to a string specified in a previous Search or Replace command.) If there is not an old string (i.e., no previous search or replace has been done), the command will terminate immediately.

For example:

Assume that the following file exists and is ready for editing:

```
   1: This is a sample file
   2: used to show dynamic line numbers.
   3: See what happens when you use
   4: Delete and Insert
   5: (the D and I commands)
   6: to edit text
   7: in your file.
   8: The insert command can place new lines
   9: in the file; there's no problem
  10: because the line numbers are dynamic;
  11:*they'll go all the way to 65533.
```

To search for the first occurrence of the string "and", type 2,12 Sand<ENTER>

The following line is displayed:

```
4: Delete and Insert
```

To get the "and" in line 5, modify the search command by typing:

```
<Del><F3>,12 Sand<ENTER>
```

The search then continues from the line after the current line (line 4), since no first line was given. The result is:

```
5: (the D and I commands)
```

To search through several occurrences of a string until the correct string is found, type:

```
1, ? Sand
```

The result is:

```
4: Delete and Insert
O.K.?_
```

If you press any key (except Y or <ENTER>), the search continues, so type N here:

```
O.K.? N
```

Continue:

```
5: (the D and I commands)
O.K.?_
```

Now press Y to terminate the search:

```
O.K.? Y
*_
```

To search for string XYZ without the verification (O.K.?), type: SXYZ

EDLIN will report a match and will continue to search for the same string when you issue the S command:

```
S
```

EDLIN reports another match.

```
S
```

EDLIN reports the string is not found.

Note that <string> defaults to any string specified by a previous Replace or Search command.

EDLIN (T)ransfer

PURPOSE Inserts (merges) the contents of <filename> into the file currently being edited at <line>. If <line> is omitted, then the current line will be used.

FORMAT [<line>]T<filename>

COMMENTS

This command is useful if you want to put the contents of a file into another file or into the text you are typing. The transferred text is inserted at the line number specified by <line> and the lines are renumbered.

EDLIN (W)rite

PURPOSE Writes a specified number of lines to disk from the lines that are being edited in memory. Lines are written to disk beginning with line number 1.

FORMAT [<n>]W

COMMENTS

This command is meaningful only if the file you are editing is too large to fit into memory. When you start EDLIN, EDLIN reads lines into memory until memory is 3/4 full.

To edit the remainder of your file, you must write edited lines in memory to disk. Then you can load additional unedited lines from disk into memory by using the Append command.

NOTE:

If you do not specify the number of lines, lines will be written until memory is 3/4 full. No action will be taken if available memory is already more than 3/4 full. All lines are renumbered, so that the first remaining line becomes line number 1.

EDLIN ERROR MESSAGES

When EDLIN finds an error, one of the following error messages is displayed:

'Cannot edit .BAK file--rename file'

Cause: You attempted to edit a file with a filename extension of .BAK. .BAK files cannot be edited because this extension is reserved for backup copies.

Cure: If you need the .BAK file for editing purposes, you must either RENAME the file with a different extension; or COPY the .BAK file and give it a different filename extension.

'No room in directory for file'

Cause: When you attempted to create a new file, either the file directory was full or you specified an illegal disk drive or an illegal filename.

Cure: Check the command line that started EDLIN for illegal filename and illegal disk drive entries. If the command is no longer on the screen and if you have not yet typed a new command, the EDLIN start command can be recovered by pressing the <F3> key.

If this command line contains no illegal entries, run the CHKDSK program for the specified disk drive. If the status report shows that the disk directory is full, remove the disk. Insert and format a new disk.

'Entry Error'

Cause: The last command typed contained a syntax error.

Cure: Retype the command with the correct syntax and press <ENTER>.

'Line too long'

Cause: During a Replace command, the string given as the replacement caused the line to expand beyond the limit of 253 characters. EDLIN aborted the Replace command.

Cure: Divide the long line into two lines, then try

the Replace command twice.

'Disk Full--file write not completed'

Cause: You gave the End command, but the disk did not contain enough free space for the whole file. EDLIN aborted the E command and returned you to the operating system. Some of the file may have been written to the disk.

Cure: Only a portion (if any) of the file has been saved. You should probably delete that portion of the file and restart the editing session. The file will not be available after this error. Always be sure that the disk has sufficient free space for the file to be written to disk before you begin your editing session.

'Incorrect DOS version'

Cause: You attempted to run EDLIN under a version of DOS that was not 2.11 or higher.

Cure: You must make sure that the version of DOS that you are using is 2.11 or higher.

'Invalid drive name or file'

Cause: You have not specified a valid drive or filename when starting EDLIN.

Cure: Specify the correct drive or filename.

'Filename must be specified'

Cause: You did not specify a filename when you started EDLIN.

Cure: Specify a filename.

'Invalid Parameter'

Cause: You specified an option other than /B when starting EDLIN.

Cure: Specify the /B option when you start EDLIN.

'Insufficient memory'

Cause: There is not enough memory to run EDLIN.

Cure: If you have multiple programs loaded you must free some memory by writing files to disk or by deleting files.

'File not found'

Cause: The filename specified during a Transfer command was not found.

Cure: Specify a valid filename when issuing a Transfer command.

'Must specify destination number'

Cause: A destination line number was not specified for a Copy or Move command.

Cure: Reissue the command with a destination line number.

'Not enough room to merge the entire file'

Cause: There was not enough room in memory to hold the file during a Transfer command.

Cure: You must free some memory by writing some files to disk or by deleting some files before you can transfer this file.

'File creation error'

Cause: The EDLIN temporary file cannot be created.

Cure: Check to make sure that the directory has enough space to create the temporary file. Also, make sure that the file does not have the same name as a subdirectory in the directory where the file to be edited is located.

5.21 FILE COMPARISON UTILITY (FC)

The File Comparison Utility (FC) compares the contents of two files. The differences between the two files can be output to the screen, the printer or to a third file. You may use FC to compare text files or binary files (files output by the MACRO assembler, the LINK Linker utility, or by a high-level language compiler).

FC makes the comparisons in one of two ways: on a line-by-line or a byte-by-byte basis. The line-by-line comparison isolates blocks of lines that are different between the two files and prints those blocks of lines. The byte-by-byte comparison displays the bytes that are between the two files.

Limitations On text file comparisons

FC uses a large amount of memory as buffer (storage) space to hold the text files. If the text files are too large for the available memory, FC will compare what can be loaded into the buffer space. If no lines match in the parts of the files in the buffer space, FC will display only the message:

*** Files are different ***

For binary files larger than available memory, FC compares both files completely, overlaying the portion in memory with the next portion from disk. All differences are output in the same manner as those files that fit completely in memory.

FILE SPECIFICATIONS

All file specifications use the following format

[d:]<filename>[<.ext>]

d: is the letter designating a disk drive. If the drive designation is omitted, FC defaults to the current drive.

filename is a one- to eight-character name of the file.

USING FC

The format of FC is as follows:

```
FC [/ /B /W /C] <filename1> <filename2>
```

FC matches the first file (filename1) against the second (filename2) and reports any differences between them. Both filenames can be pathnames. For example,

```
FC B:\TEXT\REPORT\FILE1.TXT \REPORT\FILE2.TXT
```

FC takes FILE1.TXT in the \TEXT\REPORT directory of disk drive B: and compares it with FILE2.TXT in the \REPORT directory. Since no drive is specified for filename2, FC assumes that the \REPORT directory is on the disk in the current drive.

FC OPTIONS

There are four options that you can use with the File Comparison Utility:

/B Forces a binary comparison of both files. The two files are compared byte-to-byte, with no attempt to re-synchronize after a mismatch. The mismatches are printed as follows:

```
--ADDRS----F1----F2-
xxxxxxxx   yy    zz
```

(where xxxxxxxx is the relative address of the pair of bytes from the beginning of the file). Addresses start at 00000000; yy and zz are the mismatched bytes from file1 and file2, respectively. If one of the files contains less data than the other, then a message is printed out. For example, if file1 ends before file2, then FC displays:

```
***Data left in F2***
```

/ £ stands for a number from 1 to 9. This option specifies the number of lines required to match for the files to be considered as matching again after a difference has been found. If this option is not specified, it defaults to 3. This switch is used only in text comparisons.

/W Causes FC to compress whites (tabs and spaces) during the comparison. Thus, multiple contiguous whites in any line will be considered as a single white space. Note that although FC compresses whites, it does not ignore them. The two exceptions are beginning and ending whites in a line, which are ignored. For example (note that an underscore represents a white)

___More__data_to_be_found____

will match with

More_data_to_be_found

and with

_____More_____data_to_be____found_____

but will not match with

____Moredata_to_be_found

This option is used only in text comparisons.

/C Causes the matching process to ignore the case of letters. All letters in the files are considered uppercase letters. For example,

Much_MORE_data_IS_NOT_FOUND

will match

much_more_data_is_not_found

If both the /W and /C options are specified, then FC will compress whites and ignore case. For example,

___DATA_was_found____

will match:

data_was_found

This switch is used only in source comparisons.

HOW FILE DIFFERENCES ARE REPORTED

FC reports the differences between the two files you specify by displaying the first filename, followed by the lines that differ between the files, followed by the first line to match in both files.

FC then displays the name of the second file followed by the lines that are different, followed by the first line that matches.

The default for the number of lines to match between the files is 3. (If you want to change this default, specify the number of lines with the / option). For example,

```
        ...
        ...

----------<filename1>
<difference>
<1st line to match file2 in file1>

----------<filename2>
<difference>
<1st line to match file1 in file2>

-----------------------------------------------

        ...
        ...
```

FC will continue to list each difference.

If there are too many differences (involving too many lines), the program will simply report that the files are different and stop.

If no matches are found after the first difference is found, FC will display:

```
*** Files are different ***
```

and will return to the Advance DOS default drive prompt (A>).

HOW TO REDIRECT FC OUTPUT TO A FILE

The differences and matches between the two files you specify will be displayed on your screen unless you redirect the output to a file. This is done in the same way as DOS command redirection (refer to 5.17)

To compare File1 and File2, and then send the FC output to DIFFER.TXT, type:

```
FC FILE1 FILE2 >DIFFER.TXT
```

The differences and matches between File1 and File2 will be put into DIFFER.TXT on the default drive.

EXAMPLES OF USING FC

Example 1:

Assume these two ASCII files are on disk:

ALPHA.TXT	BETA.TXT
FILE A	FILE B
A	A
B	B
C	C
D	G
E	H
F	I
G	J
H	1
I	2
M	P
N	Q
O	R
P	S
Q	T
R	U
S	V
T	4
U	5
V	W
W	X
X	Y
Y	Z
Z	

To compare the two files and display the differences on the screen, type:

```
FC ALPHA.TXT  BETA.TXT
```

FC compares ALPHA.TXT with BETA.TXT and displays the differences on the screen. All other defaults remain intact. (The defaults are: do not use tabs, spaces, or comments for matches, and do a text file comparison on the two files.)

The output will appear as follows on the screen (the Notes do not appear):

```
----------ALPHA.TXT
D                       NOTE: ALPHA file
E                       contains defg,
F                       BETA contains g.
G

----------BETA.TXT
G

--------------------------------

----------ALPHA.TXT
M                       NOTE: ALPHA file
N                       contains mno where
O                       BETA contains j12.
P

----------BETA.TXT
J
1
2
P

--------------------------------

----------ALPHA.TXT
W                       NOTE: ALPHA file
                        contains w where
----------BETA.TXT      BETA contains 45w.
4
5
W
```

Example 2:

You can print the differences on the printer using the same two source files. In this example, four successive lines must be the same to constitute a match.

Type:

```
FC /4 ALPHA.TXT BETA.TXT >PRN
```

The following output will appear on the printer:

```
----------ALPHA.TXT
D
E
F
G
H
I
M
N           NOTE: P is the 1st of
O           a string of 4 matches.
P

----------BETA.TXT
G
H
I
J
1
2
P

------------------------------

----------ALPHA.TXT
W
                      NOTE: W is the 1st of a
----------BETA.TXT     string of 4 matches.
4
5
W
```

Example 3:

This example forces a binary comparison and then displays the differences on the screen using the same two text files as were used in the previous examples.

Type:

```
FC /B ALPHA.ASM BETA.ASM
```

The /B switch in this example forces binary comparison. This switch and any others must be typed before the filenames in the FC command line. The following display should appear:

```
--ADDRS----F1---F2--
 00000009   44   47
 0000000C   45   48
 0000000F   46   49
 00000012   47   4A
 00000015   48   31
 00000018   49   32
 0000001B   4D   50
 0000001E   4E   51
 00000021   4F   52
 00000024   50   53
 00000027   51   54
 0000002A   52   55
 0000002D   53   56
 00000030   54   34
 00000033   55   35
 00000036   56   57
 00000039   57   58
 0000003C   58   59
 0000003F   59   5A
 00000042   5A   1A
```

FC ERROR MESSAGES

When FC detects an error, one or more of the following error messages will be displayed:

'Incorrect DOS version'

You are running FC under a version of Advance DOS that is not 2.11 or higher.

'Invalid parameter:<option>'

One of the switches that you have specified is invalid.

'File not found:<filename>'

FC could not find the filename you specified.

'Read error in:<filename>'

FC could not read the entire file.

'Invalid number of parameters'

You have specified the wrong number of options on the FC command line.

CHAPTER SIX ADVANCE 86 BASIC

		Page
6.1	For the Absolute Beginner in Basic	6-1
6.2	Versions of Advance Basic	6-3
6.3	Loading Basic	6-4
6.4	The Basic Prompt	6-4
6.5	Basic programs	6-5
6.6	Variable names and types	6-5
6.7	Arrays	6-7
6.8	Arithmetric operations	6-8
6.9	The Basic Screen Editor	6-11
6.10	Free memory	6-14
6.11	Format of SAVEd programs	6-14
6.12	The character set	6-15
6.13	Reading the keyboard	6-15
6.14	Random Number Generator	6-17
6.15	Soft keys and key trapping	6-17
6.16	Sound	6-21
6.17	The Advance display	6-24
6.18	The Text screen	6-25
6.19	The virtual screen	6-27
6.20	The Graphics screen	6-28
6.21	Advance Basic Quick Reference Section	6-38
6.22	Basic Error messages	6.69

This chapter introduces the Basic programming language supplied with your Advance 86.

6.1 For the absolute beginner in Basic

This chapter is not intended as a tutorial for newcomers - it's a guide for those who already know Basic. If you want to learn Basic, you should refer to the separate Advance Basic manual or indeed any of the many Basic tutorials available on the market. If you do buy an 'independent' Basic tutorial, you'll find that there are commands and features that are specific to the Advance. This chapter will serve as a good introduction to them.

However, if you want to begin in Basic, and do not have immediate access to a tutorial you can at least make a start. This little section will explain the absolute minimum about Basic, using a short program called HILO. 86a users already have this program; 86b users can type it in from page (For how to get Basic running, see the sections immediately following this which explain how to load Basic).

Basic is a programming language, which means that there is a way of getting computers to work out problems by using a special set of instructions; the set of instructions is the programming language called Basic. The Beginners All-purpose Symbolic Instruction Code (i.e. Basic) is not just one language; there are several variants of Basic in use on computers.

Each computer tends to have its own version of Basic, but there are sufficient similarities to make it sensible to talk of Basic as one language.

When we examine Basic there are certain key functions: input (actually entering information into the computer); output (displaying information); calculation and comparison. We have to study, also, the ways in which the sequence of instructions in the program can be varied depending upon circumstance. All these aspects can be illustrated by a simple program.

Suppose our problem is to design a simple number guessing game. One simple game is for the person who is not guessing to choose a number between,say, 1 and 100, and not to tell the guesser. The guesser then guesses and is told whether the guess is higher or lower than the chosen number. Depending on the number originally chosen, and on the guesses, it is possible to see how even such a simple problem can produce many different sequences of numbers: one problem is to cope with all the possible sequences; for example, it is no game if the number chosen is the same every time.

First of all, then, we have to work out in outline the way in which the game can be played, so that all eventualities are covered by one problem-solving method. In programming, the way in which we solve the problem is called the algorithm. For this case

we must first choose a secret number (step 1); then set the number of guesses made to 1 (step 2, not really necesary, but helpful); next the guesser is asked for a number (step 3); we then find whether the guess is correct, and tell the guesser, and end, or if the guess is higher or lower, tell the guesser (step 4); we increase the number of guesses by 1, and return to step 3.

In this algorithm note that we have input (what number is guessed), output (we tell the guesser how their guess compares), calculation (what number is chosen in the first place), comparison (how does the guess compare in size to the chosen number?) and control (what happens depends upon the guess and the original number chosen).

The algorithm is developed as a Basic program in the listing below.

```
10  REM  High/Low - A Number Guessing Game
20  REM
30  REM  This version by Boris Allan
40  REM
50  PRINT "HILO - Choose a number between 0 and 100,  and try  to
match the computer's number": REM Heading for program.
55  REM
60  LET computernumber = INT(RND*(100+1))
65  REM
70  LET guesses = 1
75  REM
80  INPUT "What is your guess";n
85  REM
90  IF  n > computernumber THEN PRINT "Your number  is  too
      high": REM  First check
95  REM
100 IF  n < computernumber THEN PRINT "Your number  is  too
    low": REM Second check
105 REM
110 IF n = computernumber THEN GOTO 140: REM last check
115 REM
120 LET guesses=guesses+1:REM Increment to the number of guesses
125 REM
130 GOTO 80: REM return to the input line
125 REM
140 PRINT "Success in";guesses: REM Final line
145 REM
150 STOP
```

We can now see how a Basic program is developed. All lines in this program start with a number: the number orders the sequence of instructions and allows jumps to be made from one line to another. The program starts with line 10, and the program line can extend over two lines on the printer: the computer considers this to be only one real line -- the physical lines are unimportant here. Line 10 follows the 10 by the word REM (short for REMark). This means that the rest of the line is not meant for the computer, but meant for the human reader -- to help

understand what is happening in the program. Lines 20, 30 and 40, are further information for the reader - who wrote the program, plus some spaces to clarify the reading (as are all the lines ending in 5).

The first line which is actually an instruction to do something is line 50: PRINT out on the screen a program heading. Line 60 calculates something called 'computernumber' and is a number chosen at random by the computer (i.e. the RND command). This is the number the user will have to guess. Before we get the user to do anything we have to set (line 70) the number of guesses equal to 1.

When we come to line 80, the user is asked "What is your guess?" and types in his answer. If (line 90) the user's guess is greater than the computer number THEN the computer tells the user that "Your number is too high": if the number is too low, the user (line 100) is so told. If the guess is neither too high nor too low (line 110) then a jump is made in the program to line 140, and line 140 informs the user of success and how many guesses were made.

If there has not been a jump to 140, then the number of guesses is increased by 1 (line 120) and a jump is made from line 130 back to line 60. From line 60 the process is repeated.

The program is one way of implementing the algorithm but there are other possible ways. As users are human, really to be fool - proof we need to have some way of checking that the user types in proper numbers. When we say the numbers from 1 to 100, we do not include fractional numbers and really need some way of weeding out fractional numbers.

A rather more interesting problem is that some people, when asked "how many?" actually type in FIFTY FOUR and not 54. However, if the program is to be used by people then we have to expect perfectly reasonable (although in program terms, wrong) behaviour which is not as we expect. We would also have to try to ensure that strange input does not lead to ridiculous answers. But this will do for our example.

If any of the detail of the instructions still puzzles you, such as the purposes of INT(RND*(100+1) look up the Basic commands, here, INT and RND, in the list below, which will enable you to figure out what they do. (You know that RND generates a random number: you will find under the entry that this is between 0 and 1, which would not make for a good game, so you multiply this fraction by 101, but it then needs rounding to an integer - and INT does that.)

Having got so far, you should be able to make some sense of what follows in this chapter. You should progress, skimming, to the Screen Editor section, and then try the programs under SOUND, typing them in and seeing how they work.

Basic is compatible with the Basic used on the IBM Personal Computer and many other 16-bit microcomputers. What's more, because it is derived from the standard old 'Microsoft Basic', it's easy to convert programs from other machines to run on the Advance. If you learn on the Advance, you'll also have no trouble using the Basics on other machines.

Programmers familiar with IBM Personal Computer Basic should note that while Advance Cassette Basic is the equivalent of IBM Cassette Basic, Advance Disk Basic is the equivalent of IBM Advanced Basic (BASICA).

6.3 Loading Basic

On an 86a, just switch on! Basic is stored in ROM and is available instantly. On an 86b, Basic has to be loaded from disk - it's in a file called BASICA.COM. To load it, at the DOS prompt (A>) type BASICA and press <Enter>. If you are unsure about how to do this, refer to the DOS chapter, page 5-17. Your 86b will stay in Basic until you execute the SYSTEM command.

Cassette Basic is still available on the 86b although there will be few times when you need it. To run Cassette Basic, switch on the 86b with no disk in drive A or, if it's already switched on, remove any disks and reset the machine by pressing <Ctrl>, <Alt> and <Del> at the same time. The 86b will look for a disk in drive A. As it can't find one, it will start Cassette Basic.

Be warned! Any work you do in Cassette Basic can only be SAVEd to a cassette recorder. There is no way to change to Disk Basic without clearing the machine's memory. This situation is not likely to cause problems since you can use Disk Basic for everything that Cassette Basic does including LOADing and SAVEing onto cassettes.

6.4 The Basic prompt

Basic prompts with the message "Ok" and the flashing cursor on the line below, indicating that it is ready and waiting for you to type a command. Across the bottom of the screen are a series of white boxes. These are labels for the ten function keys (<F1> to <F10>) and show you what will be typed whenever you press one of the keys. For example, pressing <F1> types the word "LIST" and <F2> enters the command "RUN". You can turn this line off, get a full list of the settings and set your own values using the KEY command.

Basic runs in two 'modes'. You can type in individual commands and have Basic respond immediately - this is sometimes called "Immediate" or "Direct" mode and is used primarily for developing and testing programs. Alternatively, Basic might actually be running a program stored in its memory. All you'll see is whatever the programmer chose to make the program display.

You will return to immediate mode when the program ends (Basic hits a STOP or END command or simply runs out of program!) or when an error occurs or when you interrupt it.

The Break Key

To stop a running Basic program, particularly if it has got stuck in an infinite loop, hold down <Ctrl> and press <Scroll Lock>. You should see the Basic "Ok" message. Notice that the front of this key is actually marked <Break> as a reminder!

6.5 Basic programs

Program lines can be up to 240 characters in length. Each line starts with a line number between 0 and 65529. More than one Basic statement may be entered per line by separating them with full colons (:).

All the standard Basic editing techniques apply. Use NEW before entering a new program. New lines are added simply by typing them - they can be entered in any order - Basic automatically sorts them into numerical sequence. Existing lines can be modified by re-typing a new version of the line. A line can be deleted by typing just its line number and pressing <Enter>. Alternatively, the DELETE command allows a whole block of lines to be deleted. In addition, Advance Basic has a powerful screen editor (described below) which allows efficient and easy editing of programs.

6.6 Variable names and types

Advance Basic is very flexible about variable names. However, there are some rules. Variable names must:

* Start with a letter

* Be in upper case. You can type variables in any way you want but Basic will convert them to all upper case. So "john" is the same name as "JOHN"

* Not be a reserved word. Basic words such as LIST, RUN, PRINT and so on can't be used as variable names. However, a name can contain a reserved word. So LET=10 won't work but LETTER=10 will.

* Have 40 significant characters. Names can be any length but Basic only uses the first forty characters to distinguish between them. This is hardly likely to prove a restriction in practice!

Basic supports four data types and conversion between them. The types can either be specified as part of the variable name (or by DEF INT, DEF DBL, DEF SNG and DEF STR) or by implying the type in a constant. The types are:

Single precision.

This is the most common type - numbers can be up to seven digits, including decimal places. Constants in a program with less than 8 digits will be assumed to be single precision - you can force numbers to be single precision by following them with an exclamation mark (!). Variables with no special suffix on their name are single precision. Mathematical functions such as SIN, COS and TAN return single-precision values.

Integer

Integers are whole numbers in the range -32768 to +32767 and are indicated by adding a percent sign (%) to a variable name or constant. For example, LET A%=10/3:PRINT A% gives a result of 3.

Double precision

Double precision numbers can be up to sixteen digits long and are indicated by adding a hash sign (#) to a variable name or constant.

Strings

Strings hold sequences of characters - that is letters, digits, punctuation symbols and so on. The characters are stored by converting each one into a code given by the ASCII code. This convention is used on many microcomputers and a full listing appears in appendix D at the back of the manual. A string literal is any text enclosed in double quotation marks, such as:

```
"Hello World"
"**??!!££"
"123"
```

A string variable is indicated with a dollar sign, for example LET NAME$="JOHN". Strings may be concatenated (joined together) using the + sign. For example, LET NAME$="SIR "+NAME$. A string can be up to 255 characters long.

Scientific notation

Most of the time, Basic uses numbers that you will recognise and understand. However, very large or very small numbers are given in a special form - similar to the way a calculator behaves. Scientific notation divides the number into two parts: a mantissa - which represents the actual number reduced to a value between 0 and 10 and an exponent which gives the power of ten to which the mantissa must be raised to get the actual number. That sounds more complicated than it is.

For example, 6.23E07 represents 62300000. 6.23 is the mantissa and 7 is the exponent. When translating from scientific notation, notice that all you do is move the decimal point the number of places specified by the exponent. If the exponent is positive, the point moves to the right, if it's negative, the point moves to the left.

4.356E-09 therefore represents 0.000000004356 which is a really small number. Basic will use scientific notation for both single and double precision numbers. In the former case, an "E" proceeds the exponent whereas double precision scientific numbers have a "D" before the exponent.

6.7 Arrays

All the variables discussed so far have been simple variables - one name is associated with each value. Basic supports arrays - groups of related variables all stored under the same name.

An array has one name and many separate values, each called elements. The names and types are just like simple variables. Each element is referenced by a number (or 'subscript') in brackets after the name. So in an array NAME$, the elements would be NAME$(1), NAME$(2), NAME$(3) ... and so on. The maximum number of elements available is declared with the DIM statement.

Individual elements can be treated just like simple variables. But their real advantage is that, because a subscript can be any numeric expression, a short routine can process all the elements of an array in turn. It might print them out, sort them or search through them for a particular value.

The simpest form of array is often called a list. It has just one 'dimension' for each element. For example, DIM NAME$(20) creates a list of elements:

```
     NAME$(1)
     NAME$(2)
     NAME$(3) ...
up to
     NAME$(20)
```

However, an array can have two dimensions. DIM A(3,4) creates a table or matrix of elements like this:

```
A(1,1)   A(2,1)   A(3,1)
A(1,2)   A(2,2)   A(3,2)
A(1,3)   A(2,3)   A(3,3)
A(1,4)   A(2,4)   A(3,4)
```

dimension is "rows" and which is "columns". Arrays can have three, four or more dimensions although these are less frequently useful. The absolute limits are 255 dimensions, each with 32767 elements - however, you'll find that memory limits restrict you before you get anywhere near this sort of size.

Once an array has been DIMed, it's size is fixed. If you might need to add items to it later on in a program, declare the array bigger than you need it in the first place. Large arrays don't actually use up a lot of memory until you fill them with information. However, if you don't mind losing the information in an array, you can scrap it and then DIM it to another size using the ERASE command.

Basic arrays have two surprise features. Until you declare an array with DIM, Basic assumes that all arrays are 11 element lists. The second thing is that, by default, arrays start with element 0 not element 1. So the default array size is actually from A(0),A(1),A(2) ... A(10) and DIM X(100), for example, creates a list of 101 elements, from X(0) to X(100).

You can, of course, simply ignore the zeroth element. Alternatively, you can make sure that your program actually uses it consistently all the way through. If you get confused, the OPTION BASE command can set the first element to be 1 rather than 0.

Users familiar with Basic on mainframe or minicomputers may have used MAT commands associated with manipulating two-dimensional arrays (matrices). Advance Basic, like most microcomputer Basics, does not provide these facilities.

6.8 Arithmetic operations

Basic provides a number of operators for performing calculations and making expressions. These include standard arithmetic such as add, subtract, divide and multiply although the symbols used may be unfamiliar at first.

Basic evaluates expressions in the same way as a mathematician or good calculator - in a previously determined and standard order. For example, the result of 4*10+2 is 42 not 48 - that is, the multiply (*) is performed before the add (+). The order of operations is called their precedence. This list gives all the operators in order of precedence.

1. Basic Functions — Commands such as ABS and SQR are always evaluated first.

2. Brackets — Any section of an expression enclosed in brackets is evaluated before the rest of the expression. You can use brackets wherever you want to be sure of the order of calculation. For example, 4*(10+2) really is 48 not 42.

3. ^ Exponentiation — The up arrow symbol is used to raise numbers to powers. For example, 2^3 is two cubed and 4^(1/3) is the cube root of four. For small powers, such as ^2 and ^3, you will often find multiplication quicker.

4. - Negation — For example, -4 is negative four

5. *,/ — Multiply and Divide

6. \ Integer divide — This is the same as ordinary division except that the result is always a whole number (integer). For example, 10\3 evaluates to 3 not 3.33333 recurring. Integer division only works for numbes in the range ±32767

7. MOD — Modulo arithmetic

   This returns the remainder after one number is divided by another. For example, 25 MOD 6 is 1.

8. +,- — Addition and Subtraction

9. Relational operators

   These are used to compare two values. The result is either 0 (the specified condition is false) or -1 (it's true). These can be used with care in arithmetic expressions although they usually occur within IF THEN statements.

   The operators are:

=	equals
<> or ><	not equals
<	less-than
>	greater-than
<= or =<	less-than-or-equal
>= or =>	greater-than-or-equal

10. Logical operators

These are used to perform logical and boolean operations on numeric values. They occur most commonly within IF THEN statements (for example IF X<0 OR X>10 THEN ...) However, if used within numeric expressions they return bit-wise logical values.

The operators return a true (-1) or false (0) depending on the true (non-zero) or false (0) value of the two operands. The operators, and the results they return, appear below:

	! X Y	X Y	X Y	X Y
	! F F	F T	T F	T T
X AND Y	! F	F	F	T
X OR Y	! F	T	T	T
X XOR Y	! F	T	T	F
X EQV Y	! T	F	F	T
X IMP Y	! T	T	F	T

Note: F represents false (0) and T represents True. Any non-zero value is treated as truth in evaluating an expression but a result of 'truth' is always given as -1.

There is one other operator NOT, which requires a single operand and merely inverts false to truth and vice-versa.

6.9 The Basic Screen Editor

Advance Basic has a very powerful screen editor to provide a simple and quick way of making corrections to Basic programs. The basic idea is to move the flashing cursor freely around the screen, making insertions and corrections to program lines previously listed on the display. It's a very powerful system and it is well worth learning to use it.

The screen editor uses the cursor pad on the right of the keyboard. To use it, the <Num Lock> key should be off. To check this, press the 6 key on the cursor pad. It should move the cursor one space to the right and not type a 6. If it types a 6, press <Num Lock> once. The cursor pad now produces commands for the screen editor rather than numbers.

The following keys are used with the screen editor:

<Up arrow>	(8)	Move up one line
<Down arrow>	(2)	Move down one line
<Left arrow>	(4)	Move left one character
<Right arrow>	(6)	Move right one character
<Home>	(7)	Move to top of screen
<End>	(1)	Move to end of current line
<Esc>		Clear current line
<Backspace>		Delete character to left of cursor
<Ins>		Start inserting characters
<Del>		Delete character to right of cursor
<Enter>		Accept current line

To correct a program line, LIST the line on the screen. It may already be there - because it appears in a previous listing or you've only just entered it. If Basic finds an error while running a program it will automatically LIST the offending line for you. If the line isn't on the screen, use the EDIT command. This LISTs the line and puts the cursor ready at its start.

Move to the point where you want to make changes by using the arrow keys and <Home> and <End>. Type new characters over the old ones or use the <Ins>, <Del> and <Backspace> keys to make changes. When the line is correct, press the <Enter> key.

Using <Ins> When you press <Ins>, the cursor will change shape to show that you are now inserting characters. As you type, all the text to the right of the cursor will move along to make space for the new information. This is very useful for inserting missing spaces and so on as well as adding whole new commands to long lines.

The editor will continue to insert until you press <Ins> again or use one of the other screen editor keys.

Using <Del> The <Del> key deletes characters to the right of the cursor. As each character is deleted, the remaining text closes up the gap. The cursor stays still.

Using <Backspace> <Backspace> works just like it always does - it moves the cursor to the left and rubs out one character as it goes. Any text to the right of the cursor moves to the left to fill the gap.

Your changes are only 'accepted' when you press <Enter>. Sometimes, it will be easier to make lots of changes to lots of lines and then go back through them pressing <Enter> once on each line. Also remember that the screen editor doesn't mind program lines that extend over several actual screen lines. <Ins>, <Del> and so on all work no matter how long the Basic line is.

Once you start using the screen editor a lot, you'll often find that you are typing a new command on a line that already has text on it. To avoid errors, you should get into the habit of pressing <Escape> before starting a new command on a 'busy' screen. This will clear the line you are on and so make sure that your new command doesn't get mixed up with any old information already on the screen.

Alternatively, many people always use the <Down arrow> to drive the cursor to the bottom of the screen before starting new commands. Don't let the screen get too messy - it'll will become easy to make mistakes. After a lot of correcting, clear the screen (CLS) and LIST the program again.

Tricks with the screen editor

There are lots of useful 'tricks' when using the screen editor. As you get used to it, you'll discover lots more for yourself. Some of the popular ones are:

Forget changes:	If you make a mess of correcting a line, press <Escape> and then try EDITing it again.
Repeat a command:	Just put the cursor on the same line as a command that is already on the screen and hit <Enter>.
Copy a line:	To copy a line, just move the cursor to its start, alter its line number and press <Enter>. You'll get two copies of tne line - one with the old and one with the new number. If you actually want to move a line, copy it and then delete the original by entering its line number.
Recover a deleted line:	If you delete a line by mistake and you have a copy of it on the screen, you can re-enter it into the program by placing the cursor at its start and pressing <Enter>. This can be very useful but don't rely on it as a technique - it's best not to delete lines by mistake in the first place!

6.10 Free memory

To discover how much memory is remaining at any time, enter PRINT FRE(0). You'll notice that at most, the Advance returns a figure of around 62K despite your machine having an internal memory of at least 128K.

This limit is imposed by the architecture of the Advance's 8086 microprocessor which divides memory up into 64K segments. Persuading Basic to ignore these divisions would result in a significant loss of performance. However, you are unlikely (and possibly unwise) to need more memory space for Basic programs.

Should you need to, you can access the remainder of the Advance's Ram by switching segments (the DEF SEG command) and using Basic's interfaces to machine language - PEEK, POKE, CALL and USR.

The Advance microcomputer does offer a tremendous memory capacity but one that is generally impractical for Basic. Other langauges allow access to the full Ram - such as machine code, and C. However, the main reason for providing so much memory is to allow professional applications programs such as word processors and databases the space they need to run.

6.11 Format of SAVEd programs

Advance Basic normally SAVEs programs in its own internal compressed format. This saves disk space and speeds up loading. However, it makes the files pretty incomprehensible to anything but Basic. It is possible to save a Basic program as a pure ASCII text file using:

```
SAVE "FILENAME",A
```

This produces a file which can easily be edited by another DOS program, such as the EDLIN line editor. It is also essential that routines that are intended to be MERGEd into other programs are saved in this way.

Many other microcomputers use similar Basics to the Advance. But the internal format would normally prevent you moving Basic programs across from these machines. By using the SAVE,A option on the source micro, it should be possible to read in a Basic program from another machine - if it uses the MS-DOS disk format, you can probably read its disks directly on the Advance. Otherwise, you would need to transfer the ASCII file of the Basic program over a serial line linking both computers.

Basic automatically recognises ASCII files when it is loading programs. So you just LOAD an ASCII file in exactly the same way as an internal format file.

There's a second option on the SAVE comand. SAVE "FILENAME",P. This saves the program in an encrypted binary format so that it can no longer be listed or edited. Use this option to protect sensitive data and/or programs. However, be warned! There is no way to remove the encryption. Protect a program and you protect it for good. So only ever use this option when you have a number of normal copies of the complete program safely locked away!

6.12 The character set

The Advance has a full character set consisting of 256 different symbols arranged according to the ASCII code common to most computers. A full list is given in appendix D. All the characters are accessible with PRINT CHR$(code). As you can see, the range of symbols lets you create displays for almost any work - including mathematical and business formulae and even simple games. The range of block and box characters make it easy to produce neat rules and boxes to present tidy displays. Should this array of characters not satisfy you, it is possible to define your own character shapes using a technique described in the Advance Basic manual.

6.13 Reading the keyboard

Advance Basic provides all the usual Basic commands for reading information from the user, including INPUT and LINE INPUT. In addition, it has a number of other useful commands for reading just individual keystrokes.

INPUT$

INPUT$ reads a certain number of characters from a data file or from the keyboard. It blindly accepts all characters including codes that it would normally be impossible to enter such as <Enter> and <Back space>. The only way to interrupt an INPUT$ is with the Break key. The characters that are typed do not normally appear on the screen. For example:

```
10 CLS:PRINT "Type a 3 letter password"
20 IF INPUT$(3)<>"BYE" THEN 10
30 PRINT "BYE"
```

INPUT$ should be used whenever the program must halt and wait for at least one character to be entered.

INKEY$

This function reads back a single keypress from the keyboard. The function does not wait for a key to be pressed. If no key has been pressed, then the function returns a null string (""). INKEY$ is used for applications which don't actually halt when they scan the keyboard such as simulations and games.

The Advance keyboard does not always generate a single key code for a single key press. The Soft keys <F1> to <F10> will return tneir contents character-by-character if they are enabled. If they are not, they return a special extended key code consisting of two characters - an ASCII null (CHR$(0)) and then a special scan code for the key that was pressed.

The extended key code mechanism allows the user to generate a whole set of extra functions from the keyboard without distorting the normal ASCII codes. The only problem with this approach is that occasionally INKEY$ will return a two character string, starting with a 00 and followed with an extended code. Programs that rely on INKEY$ therefore should always check the length of returned strings and take appropriate action.

The complete set of extended codes is:

Extended code	Key pressed
3	NUL
15	<Shift><Tab>
16-25	<Alt> and Q,W,E,R,T,Y,U,I,O,P
30-38	<Alt> and A,S,D,F,G,H,J,K,L
44-50	<Alt> and Z,X,C,V,B,N,M
59-68	<F1> to <F10> if not being used as soft keys
71	<Home>
72	<Up arrow>
73	<PgUp>
75	<Left arrow>
77	<Right arrow>
79	<End>
80	<Down arrow>
81	<PgDn>
82	<Ins>
83	<Del>
84-93	<Shift> and <F1> to <F10>
94-103	<Ctrl> and <F1> to <F10>
104-113	<Alt> and <F1> to <F10>
114	<Ctrl> and <PrtSc>
115	<Ctrl> and <Left arrow>
116	<Ctrl> and <Right arrow>
117	<Ctrl> and <End>
118	<Ctrl> and <PgDn>
119	<Ctrl> and <Home>
120-131	<Alt> and 1,2,3,4,5,6,7,8,9,0,-,=
132	<Ctrl> and <PgUp>

6.14 Random number generator

Basic provides a function to generate pseudo-random numbers. The computer, like most microcomputers, can't provide for true-randomness but can provide a close enough imitation suitable for games, simulations and similar applications.

The Advance random number function is RND(n) which returns a random number between 0 and 1. If 'n' is greater than 0, RND returns the next random number in sequence from its random number table. If 'n' is 0, RND returns the last random number generated. This can be very useful in debugging!

A common formula to convert RND(1) into a number between 1 and X is often used as a defined function like this:

```
DEF FN R(X)=INT(RND(1)*X)+1
```

The general form of this can be remembered as:

INT(RND(1)*number of numbers to choose from) + lowest possible number to choose.

6.15 Soft keys and key trapping

The Advance's ten function keys can be used in two different ways. Both Cassette and Disk Basic let you use them as 'soft keys', setting them up so that they type a string of characters at the touch of a button. Disk Basic also lets the keys work as interrupts, telling Basic to tempoarily halt a program and execute another section of program instead.

Using the keys as Soft keys is easy. The command KEY ON displays a row of key labels at the bottom of the screen. In 40 column mode, the first five labels will be visible. In 80 column mode, all ten keys are labelled. KEY OFF will switch the label line off again. Each of the ten keys can store up to 15 characters, including special codes for keys such as <Enter> and so on. However, the labels will only show the first six characters of the key setting. To get a full list, use the command KEY LIST.

To set the contents of a key, use a command of the form KEY number, string. For example, to set <F10> to clear the screen, enter KEY 10,"CLS"+CHR$(13). The CHR$(13) is necessary to add an <Enter> to the end of the string. 13 is the character code for <Enter>. If you now press <F10>, the characters C, L, S and <Enter> are automatically typed and the screen will clear.

To disable a soft key, assign it a null string with KEY number,"". The disabled key will produce a two-character sequence, the first being a CHR$(0) and the second being a code for the key's position on the keyboard.

It's usual to use the soft keys to generate useful commands during program development. When you first start Basic, the keys are set to ten useful commands such as LIST, RUN, LOAD" and so on. You may find your own working habits prefer other settings.

You can use the keys in your own programs by setting appropriate strings to generate commands for your program although this can be a long-winded way of using them. If you do, notice that the INKEY$ command will read each character out of a soft key just as if you were typing them in turn on the keyboard.

Key Trapping

The second method of using the keys is only available in Disk Basic and turns them into special devices which interrupt the Basic intepreter. Basic handles the keys in the same way as several other devices such as a light pen or a joystick "Fire' button. The support provided is called "Event trapping".

To use key trapping, you should first set up a series of ON KEY(n) GOSUB statements to tell Basic where to branch to should one of the specified keys be pressed. The key numbers are:

1 to 10	Function keys <F1> to <F10>
11	Up arrow
12	Left arrow
13	Right arrow
14	Down arrow

For example, if you set ON KEY(1) GOSUB 1000, Basic would jump to a subroutine at line 1000 every time it 'trapped' the user pressing <F1>. Once the ON KEY(n) GOSUBs are set, trapping can be turned on with KEY(n) ON. This is different from the KEY ON command for displaying Soft key labels. With KEY(n) ON, Basic keeps an eye open for the user pressing the particular key specified. If he does, Basic will stop whatever it is doing and jump to the specified subroutine.

Key trapping can be turned off again with KEY(n) OFF though note that this doesn't affect the ON KEY(n) GOSUB setting - it just tells Basic not to bother looking for that particular key. A variation on this is KEY(n) STOP. This stops Basic actually interrupting when the key in question is pressed. But Basic takes note that it has been pressed and jumps to the subroutine the moment that trapping is turned back on again with KEY(n) ON.

Each key that is being trapped must have its own separate 'handling' routine that carries out the appropriate action for that key. Once Basic has jumped to a trap handling subroutine there is no way for commands such as INKEYS and INPUT to read which key it was. At the end of the subroutine, you can return to where you left off with a normal RETURN command or, if you're careful, you can go back to a particular line with a RETURN line number command. This can be dangerous as the trap may have caused Basic to leap out of FOR loops or subroutines without properly finsihing them.

If you think about it, you can see that there would be obvious problems if a key was trapped again during its own trap handling routine. To prevent this, whenever a trap occurs, Basic does an automatic KEY(n) STOP. When you return to the normal program, it does a KEY(n) ON again.

Let's try a simple key trapping demo:

```
   10 ON KEY(1) GOSUB 1000
   20 I=1
   30 PRINT I
   40 I=I+1
   50 GOTO 30
 1000 REM Handle F1 trap
 1010 PRINT:PRINT"You pressed F1!"
 1020 RETURN
```

RUN this as it is to prove that normally, Basic ignores any keys that are pressed while it is busy. Now add a line 15 KEY(1) ON and RUN it again. This time, pressing <F1> interrupts the program and executes the subroutine. A slightly more daring trap handler is one that alters variables outside of its own confines. This one works quite nicely:

```
   10 ON KEY(11) GOSUB 1000 ' up arrow
   20 ON KEY(14) GOSUB 1020 ' down arrow
   30 KEY(11) ON:KEY(12) ON
   40 I=1:C=0
   50 PRINT I
   60 I=I+C
   70 GOTO 50
 1000 REM count upwards
 1010 C=1:RETURN
 1020 REM count downwards
 1030 C=-1:RETURN
```

When this is running, the up and down arrows (make sure you're in cursor pad mode) alter the direction of the counting. Obviously, handlers should be very careful about altering information outside their own scope. If you have any parts of the program that really shouldn't be trapped, make sure they are surrounded by KEY(n) OFFs.

Controlled interrupts like tnis are traditionally only available in machine languaqe. Advance Basic implements this powerful facility in an easy form. However, it does need to be used with care if your programs are to work reliably. Some of the more obvious uses for the system are panic buttons - to get people out of sections of program they don't want to be in or to present help with that particular operation. You could even have a function key open a litle window with a calculator or watch in it!

6.16 Sound

The Advance has a built in speaker which can be used not only for warning beeps but also for special noises and music. Advance Basic provides a number of commands to help here.

BEEP

BEEP is available in both versions of the Basic and makes a single beep sound on the speaker. This is the equivalent of the Bell on old fashioned mechanical computer terminals and you'll find that printing a Bell character (CHR$(7)) actually rings the bell! Basic itself frequently uses the Bell to tell you that something has gone wrong. If you use BEEP in your own programs, many users will recognise it is a warning sound. Other than that, all it's good for is a temporary warning command to insert in programs as you debug them.

SOUND

SOUND is a much more useful command. It is followed by the frequency of the note to play and the number of 'clock ticks' to play it for. There's 18.2 ticks in one second and the duration can vary from 0 to 65535. The program continues to execute even while a sound is playing but will halt at the next SOUND command and wait for the first one to finish. Use a frequency of 32767 for a pause in the sound.

RNDSOUND

```
10 SOUND INT(RND(1)*1000)+130,RND(1)
20 GOTO 10
```

SOUND1

```
10 REM SOUND DEMO1
20 FOR F=523 TO 261 STEP -10
30 SOUND F,F/1000
40 NEXT F
50 GOTO 20
```

PLAY

The final sound command is PLAY. This is only available in Disk Basic. PLAY provides a very powerful technique for producing tunes and sequences of sounds. It is followed by a string which contains one-letter codes for various actions. As an example,

```
PLAY "CDEFGABC"
```

plays a scale. The complete "music language" is:

A-G — Plays the specified note in the current octave. Each note can be followed by a £ or + (for a sharp note) or a - (for a flat note). The sharps and flats only work for a normal scale - if you put B£, Basic won't figure out that you mean a C!

Each note can also be followed by a number to say how long it plays for. The number can vary from 0 to 64 and the actual note played is 1/number - that is, a crotchet is followed by 1, a quaver by 2, a semi-quaver by 4 and so on.

On — Selects an Octave from 0 to 6. Basic starts off in octave 4, an octave higher than middle C. Each octave starts with C and ends at B.

Nn — This is an alternative way of selecting from the notes available - particularly useful if you only want to change octaves for just a single note. n can be from 0 to 84.

Ln — L sets the length of the following notes. This works just as lengths for individual notes but it sets the length for all the notes following.

Pn — Pauses for 1/n. The length of the pause is set in the same way as for notes (above).

a dotted note.

Tn — This sets the overall tempo of the music by setting the number of semi-quavers in a minute to a value from 32 to 255.

MF — Selects foreground mode. A new sound will cause the program to stop and wait until an old sound has finished playing. Basic starts off in the foreground mode.

MB — Selects background mode. Music (from either SOUND or PLAY) is automatically queued up so that a Basic program can carry on executing while music is being played. Up to 32 notes can be queued at once.

MN Selects 'normal' music

ML Selects legato music

MS Selects staccato music

X variable; Executes the named string as if it were a PLAY command. This allows particular sequences to be put into separate strings and then combined into a large PLAY command. This makes it easy to repeat phrases and so on.

Putting PLAY strings together.

You can combine as many commands as you like in a PLAY string in any order you like. If you can't get the string long enough, use separate strings and then use a final PLAY command which "Xs" them together. For clarity commands can be separated with a space. Where 'n' occurs above, it can either be a constant (such as "G4A2" and so on) or an equals sign followed by a variable name and a semi-colon (such as "G=Y;A=Y;"). PLAY commands are designed primarily for tunes but, as the examples below should show, you can use them for special effects:

```
10 REM PLAY DEMO
30 T$="C16E8F4"
40 FOR OCTAVE=1 TO 5
50 PLAY "O=OCTAVE;XT$;"
60 NEXT OCTAVE
70 PLAY "O3"
80 FOR T=32 TO 252 STEP 30
90 PLAY "T=T;XT$;"
100 NEXT T
```

```
10 REM NOISE DEMO WITH PLAY
20 PLAY "T255L64"
30 N$="BAGFEDC"
40 FOR I=1 TO 10
50 PLAY N$
60 NEXT I
70 FOR I=1 TO 500:NEXT I
80 FOR O=1 TO 6
90 PLAY "O=O;XN;"
100 NEXT O
```

6.17 The Advance display

The Advance has a very powerful range of display options selected using the WIDTH and SCREEN commands. These are:

TEXT screens

40x25 characters, 16 colours + flashing, 8 screens
80x25 characters, 16 colours + flashing, 4 screens

Medium resolution graphics

320x200 graphics, 40x25 text, 4 colours, 2 palettes

High resolution graphics

640x200 graphics, 80x25 text, Black-and-White

In addition, you can set the border colour of the screen. Users familiar with IBM should note that the Advance effectively has only a Colour/Graphics adapter and should ignore the complications introduced by the IBM monochrome adapter.

The SCREEN command can be used to switch between different display modes and, within a particular mode, COLOR sets the various colour options. CLS will clear the current screen.

The SCREEN command looks like this:

SCREEN mode,burst,apage,vpage

'mode' selects the type of display. 0 is either text mode, 1 is medium resolution graphics and 2 is high resolution graphics. 'burst' turns the colour signal on or off. You may need this if you are working with a monochrome screen. In a text mode, a value of 0 turns colour off and a value of 1 turns colour on. In medium resolution graphics, 'burst' should be 0 for colour and 1 for no colour! High resolution graphics are in black and white so the setting of 'burst' is irrelevant.

'apage' and 'vpage' work only in text mode. The Advance has enough memory to keep several copies of a screen display available. You could put a help screen on one and enter data on another. In 40 column mode, there are 8 such screens (or 'pages') and in 80 column mode, there's 4 of them. 'apage' selects which screen is active - that is which one any changes such as PRINTs, CLSs and so on will affect. This need not necessarily be the screen you can see. 'vpage' selects which screen is actually visible. By manipulating these parameters, it's possible to create and update screens while the user is doing something else and switch instantly between them.

Fortunately, you don't always need to work out all the parameters in a SCREEN command. You can miss some off or skip them by just putting extra commas in. Here are some examples:

SCREEN 0,0,0 Selects a black and white text screen with screen 0 active.

SCREEN ,,1,1 Moves onto and displays text screen number 1.

SCREEN 2 Selects High resolution mode

SCREEN 1,0 Selects Medium resolution graphics and makes sure that the colour is on

6.18 The Text screen

The text screen is the easiest to deal with. Basic starts off in text mode but you should always make sure with a SCREEN 0 ... command. The two different screen widths are set with either WIDTH 40 or WIDTH 80. Executing these will also clear the screen and set the border to black.

Selecting text colours

In text mode, the COLOR command works like this:

COLOR foreground, background, border

'foreground' is the colour that characters (text) will appear in. This can be from 0 to 31 as shown below, the colours from 16 to 31 being flashing versions of the colours from 0 to 15.

'background' sets the backdrop colour of the screen. This can only be taken from colours 0 to 7.

'border' selects the colour for the screen border. You can pick any colour from 0 to 15.

The text colours are:

0 Black	8 Gray
1 Blue	9 Light blue
2 Green	10 Light green
3 Cyan	11 Light cyan
4 Red	12 Light red
5 Magenta	13 Light magenta
6 Brown	14 Light brown
7 White	15 Bright white

So a simple example of a COLOR command is COLOR 4,7,7 which sets red text on a white screen with a white border. This program shows you (most of) the possible combinations:

```
10 REM 16 COLOURS
20 SCREEN 0,1:WIDTH 40:KEY OFF:CLS
30 FOR F=0 TO 15:FOR B=0 TO 7
40 COLOR F,B:PRINT " 86 ";
50 NEXT:NEXT:NEXT
60  COLOR 20,15:LOCATE 20:PRINT "And there's 16 flashing  colours
":COLOR 15,0:PRINT
```

Controlling the cursor

Advance Basic provides two ways of controlling the cursor. The Basic command LOCATE will move it to a particular position on the screen. In addition, printing certain special control codes will move it. The useful codes are:

09 Tab the cursor to the next column
10 Line feed (Moves the cursor to the start of the next line)
11 Home the cursor in the top left
12 Form feed (Clears the screen)
13 Move the cursor to the start of the next line
28 Move the cursor right
29 Move the cursor left
30 Move the cursor up
31 Move the cursor down

These codes can easily be used, for example PRINT STRING$(4,CHR$(29)) will move the cursor four places to the left.

LOCATE row,column,cursor

Alternatively, LOCATE row,column moves the cursor to any point on the screen. The functions POS(0) and CSRLIN return the currrent column number and row of the cursor repectively. The top left of the screen is, in both cases, 1,1.

Normally, there's 24 lines of screen to play with because the 25th status line is being used for soft key labels. You can use the 25th line yourself by executing KEY OFF and then LOCATE 25,1 followed by PRINT. Information on the 25th line doesn't get scrolled off the screen like information on lines 1-24. So it makes an ideal place for status or help information in your own programs.

The 'cursor' parameter of LOCATE can be used to turn the flashing cursor on and off within programs. Normally, when a program is running, the cursor is not visible. A value of 1 for 'cursor' turns it on (for example, LOCATE ,,1). Setting 'cursor' to 0 switches it off. This facility is useful in providing a flashing cursor when commands such as INKEY$ are being used.

6.19 Virtual Screens

As discussed above, the Advance has space for 8 40x25 text images and 4 80x25 text images. Basic has a built in facility where you can swap these copies of the screeen around, making it easy to provide instant information and tidy displays. By default, the Advance writes all its information onto to Screen 0 and shows you Screen 0.

To view an alternate screen, use SCREEN ,,,n where n is the number of the screen (0 to 7 in 40 column mode or 0 to 3 in 80 columns). Unless you tell it otherwise, Basic will still be writing new information on Screen 0. To pick which screen is actually being used, use SCREEN ,,n. This system allows you to create new screens without the user ever seeing them and also to have a set of ready-made screens instantly available.

The Advance however has only one cursor. If you are trying to write information onto a number of screens, you will have to remember the cursor position as you switch between them. In this case, the code to switch screens would look more like this:-

```
100 REM go to screen 2
110 CX1=POS(0):CY1=CSRLIN:REM remember position
120 SCREEN ,,2,2
130 LOCATE CY2,CX2: REM restore cursor
      :
      :
200 REM go to screen 1
210 CX2=POS(0):CY2=CSRLIN:REM remember position
220 SCREEN ,,1,1
230 LOCATE CY1,CX1: REM restore cursor

    and so on.
```

The following example program uses the virtual screen facility to provide an 'electronic jotter' where the user can jot down up to seven screenfuls of messages, reminders and so on. A more developed version of this could prove quite a handy office tool!

```
10 REM Virtual screen demo
20 REM
30 REM A 7-page electronic jotter
40 REM
50 REM Set everything up
60 SCREEN 0,1,0,0:WIDTH 40:KEY OFF
70 FOR S=0 TO 7
80 SCREEN ,,S,0:CX(S)=1:CY(S)=1
90 CLS:LOCATE 25,1:PRINT"PAGE ";S;": press F1 last F2 next F3 end"
100 NEXT S
110 FOR I=1 TO 3:KEY I,CHR$(I):NEXT I
120 SCREEN ,,0,0:S=0:LOCATE 1,1
130 REM Loop to get & print keypresses
140 LOCATE ,,1:A$=INKEY$:IF A$="" THEN 140 ELSE LOCATE ,,0:A=ASC(A$)
150 IF A=0 THEN GOSUB 210 ' cope with arrow keys
```

```
160 IF A=1 THEN PO=-1:GOSUB 280:GOTO 140
170 IF A=2 THEN PO=+1:GOSUB 280:GOTO 140
180 IF A=3 THEN SCREEN ,,0,0:CLS:END
190 IF A=8 THEN PRINT CHR$(29);" ";CHR$(29);:GOTO 140 ' do a backspace
200 PRINT CHR$(A);:GOTO 140
210 REM Convert arrow codes into cursor codes
220 A=ASC(MID$(A$,2))
230 IF A=72 THEN A=30:RETURN
240 IF A=75 THEN A=29:RETURN
250 IF A=77 THEN A=28:RETURN
260 IF A=80 THEN A=31:RETURN
270 A=0:RETURN ' trap any other codes
280 REM Change the screen that's showing
290 CX(S)=POS(0):CY(S)=CSRLIN:S=S+PO
300 IF S<0 THEN S=7
310 IF S>7 THEN S=0
320 SCREEN,,S,S:LOCATE CY(S),CX(S)
330 RETURN
```

6.20 The Graphics Screen

The Advance must be in a graphics mode before it can produce finely detailed drawings and diagrams. Do this with a SCREEN 1 SCREEN 2 for high resolution, black-and-white, graphics. The WIDTH command works in the graphics modes - in medium resolution mode, WIDTH 80 will switch to high resolution mode and in high resolution mode, WIDTH 40 acts just like a SCREEN 1.

Selecting the Palette

This only applies to medium resolution graphics - High resolution graphics are in black-and-white and using the COLOR command produces an error. Medium resolution graphics provide a choice of one of two sets of four colours. The sets are called palettes and are:

Colour number	Palette 0	Palette 1
0	Background	Background
1	Green	Cyan
2	Red	Magenta
3	Brown	White

As you can see, colour 0 is always the same as the current background colour. To set the colour, you use the COLOR command like this:

```
COLOR background,palette
```

'background' can be selected from any of the sixteen colours described in the text screen section (a number from 0 to 15). If 'palette' is an even number, palette 0 will be selected. If it's odd, palette one will be selected. When the palette is changed all the colours currently on the screen will instantly change to their new equivalents. This can be useful for special effects. Once the palette is set, the other graphics commands (such as PSET, PRESET, LINE and so on can choose from any of the four colours in the palette). Text that appears on the graphics screen will normally appear in colour 3

A quick tour of the graphics commands

Put the system into medium resolution graphics with SCREEN 1,0. The graphics screen is divided up into a grid of dots called pixels (short for picture elements). There's 320 across (numbered from 0 to 319) and 200 down (from 0 to 199). When you use graphics, you can set the colour of each particular pixel - setting a row of them produces a straight line for example. In order to specify which pixels we want to alter (basically where we want to draw), we use a co-ordinate system.

Starting in the top left of the screen, we say how far along the pixel is and then how far down it is. The very top-most left-most pixel is pixel (0,0). (160,100) is in the centre of the screen and so on. The bottom right-hand corner is (319,199).

PSET (x,y),colour

This is the simplest of graphics commands - it sets the colour of a particular pixel on the screen. 'x' and 'y' are its co-ordinates and 'colour' is a number from 0 to 3 specifying which colour you want it to be. If you miss 'colour'off, PSET will use colour 3. To see this in action, clear the screen (CLS) and enter PSET (160,100),3. This should produce a tiny dot in the centre of the screen. This program produces a few more:

```
10 REM random dots
20 SCREEN 1,0:COLOR 0,0
30 PSET (INT(RND(1)*320), INT(RND(1)*200)), INT(RND(1)*3+1
40 GOTO 30
```

Use the break key to stop this and try changing the palette with COLOR ,1 and COLOR ,0. PSET has a near twin called PRESET. This does exactly the same thing only if you miss the colour off, it will set the specified pixel to the background colour, effectively "turning it off".

LINE

It would take ages to draw things with PSET! First of all, we need a fast way of drAwing lines. Clear the screen and try this:

```
LINE (0,0)-(319,0)
LINE -(0,199)
LINE (160,0)-(160,199)
```

And perhaps the odd box here and there:

```
LINE (10,10)-(260,170),2,B
LINE (130,30)-(190,180),,B
```

Even filled in boxes:

```
LINE (70,120)-(300,150),1,BF
LINE (180,10)-(310,190),2,BF
```

The LINE command looks very formidable:

```
LINE (x1,y1)-(x2,y2),colour,BF
```

'x1' and 'y1' are the starting co-ordinates of the line. You can leave these out if you want to, for example LINE -(10,20) draws a line from the last point referenced on the screen to 10,20. 'x2' and 'y2' are, of course, where you want the line to go to. 'colour' as you might guess is the colour that the line will be drawn in. If you leave it out, Basic will use colour 3.

Next 'B' stands for Box. If you put a B here, Basic will take (x1,y1) and (x2,y2) to be the corners of a rectangle and draw a box around it. Use BF here and it will fill the rectangle with colour. As usual, you can leave off any parameters you don't want to set and fill in any missing ones with dummy commas. So LINE -(50,60),,BF draws and fills a box with one corner at the last point we drew to on the screen and the opposite corner at (50,60) in colour 3.

One final trick. The second co-ordinate can be given relative to the first. This method doesn't give an actual (or absolute) position on the screen but tells Basic to move so many pixels down and so many along - negative numbers will move up and left. To do this, use a command like LINE - STEP(10,10). Here's a little example:

```
10 SCREEN 1,0:CLS
20 PSET 0,199
30 FOR I=1 TO 10
40 LINE - STEP(30,-20),,BF
50 NEXT I
```

CIRCLE

This is only available in Disk Basic

Circle provides a quick way to draw circles, arcs and ellipses. The command is of the form:

CIRCLE (x,y),r,colour,start,end,aspect

'x' and 'y' are the co-ordinates of the centre and 'r' is the radius of the circle. As usual, 'colour' can be set to a value 0 to 3 or skipped with a dummy comma. The last three values control how much of a circle is drawn - both how complete it is and how round it is.

'start' and 'end' tell CIRCLE where to start and stop drawing. Be default, you'll get a full circle. To set the limits, use values in the range ±2*PI (PI=3.141593). The circle is drawn starting at 3-o'clock (0) and progresses anti-clockwise round to 2*PI. 12-o'clock is at PI/2, 6 is at 3*PI/2 and 9-o'clock is equivalent to PI. A negative angle will join the arc to the centre point.

'aspect' controls how squashed (or elliptical) the circle is. If the 'aspect' is 1, CIRCLE will produce a visually correct circle. Less than 1 and it is squashed downwards. More than one and the circle is squashed sideways. The following programs demonstrate some of these features:

```
10 REM Smile
20 SCREEN 1,0:CLS
30 CIRCLE (15,15),15
40 CIRCLE (10,12),4:CIRCLE(20,12),4
50 CIRCLE (15,13),10,3,3.927,5.498
60 LOCATE 10,1
```

```
10 SCREEN 1,0:CLS
20 FOR A=0 TO 2 STEP .15
30 CIRCLE (160,100),95,,,,A
40 NEXT A
```

PAINT

Tnis is only available in Disk Basic

PAINT can be used to fill any shape on the screen with a particular colour. You have to make sure that the shape really is fully enclosed and the more complex a shape is the more memory you Advance will need while it is trying to fill it. If you have problems with memory, reserve some more stack space with the CLEAR command.

PAINT is used like this:

PAINT (x,y),paint colour,edge colour

'x' and 'y' are the co-ordinates of any point within the shape to be filled in. 'paint colour' is the colour that you want the shape filled with. Occasionally you might like to specify an 'edge colour' as well - this tells PAINT where to stop PAINTing so that you achieve effects such as a green area with a red edge around it.

You should be able to try PAINT out for yourself, filling areas created by LINE and CIRCLE. Some examples:

```
10 REM PAINT demo
20 SCREEN 1,0:KEY OFF:CLS
25 F=1
30 DEF FN R(X)=INT(RND(1)*X)+1
40 LINE (0,0)-(319,199),,B
100 X=30+FNR(260):Y=30+FNR(130):R=FNR(50):A=FNR(10)
105 IF A<5 THEN  A=A/5
110 CIRCLE (X,Y),R,,,,A
120 PAINT (X,Y),FNR(4)-1,3
125 IF RND(1)>.9799999 THEN COLOR 0,F:F=ABS(F-1)
130 GOTO 100
```

```
10 SCREEN 1,0:CLS
20 FOR A=0 TO 2 STEP .15
30 CIRCLE (160,100),95,,,,A
40 NEXT A
50 PI=3.141593
60 R=INT(RND(1)*100:A=RND(1)*2*PI
70 PAINT 160+COS(A)*R, 100+SIN(A)*R, INT(RND(1)*3)+1
80 GOTO 60
```

DRAW

This is only available in Disk Basic

DRAW provides a method for defining and drawing any shapes you want to. It works a bit like the PLAY command in that the shape is defined by a series of one-letter commands in a string. For example, DRAW "M160,100 U5R5D5L5" will produce a square in the centre of the screen. There's a whole mini-language that can be used with DRAW.

Un	Move up n units
Dn	Move down n units
Ln	Move left n units
Rn	Move right n units
En	Move diagonally up and right
Fn	Move diagonally down and right
Gn	Move diagonally down and left
Hn	Move diagonally up and left

Mx,y Move to positon (x,y) on the screen. If you want to move relative to your current position, put + and - signs in front of x and y as appropriate. You must have either a + or a - sign in front of x for this to work. With a relative move, x and y specify how many units right (or left) and down (or up) to move rather than the actual position to go to.

B This is used in front of any of the movement commands. The move is carried out but no points are drawn.

N This is used in front of any of the movement commands. It means move but return to the starting position afterwards.

An Sets angle to n. n can be from 0 to 3 and says whether the drawing is rotated a quarter, half or three quarter turn.

Cn Sets the colour to a value from 0 to 3 (medium resolution) or 0 to 1 (high resolution).

Sn Sets scale factor n from 1 to 255. This affects the size of the units used in the moving commands and thus the overall size of the drawing.

Xvariable; Executes the named string as if it were a DRAW command in itself. This allows drawings to be built up as separate strings and then combined together to form complex and large patterns.

Just like PLAY, n can either be a number or an equals sign followed the name of a numeric variable and a semi-colon. So you can use DRAW "U10" as well as DRAW "U=V;". This neat example uses the S command to draw a series of growing octogons, with interesting results:

```
10 REM PYRA
20 SCREEN 1,0:KEY OFF:CLS
40 DRAW "bm200,10"
50 OT$="FDGLHUER"
60 FOR S=10 to 255 STEP 5
70 DRAW "S=S;XOT$;"
80 NEXT S
```

```
10 REM DRAW tester
20 SCREEN 1,0:KEY OFF:CLS
30 ON ERROR GOTO 150
40 PRINT "DRAW tester"
50 PRINT:PRINT"This program lets you try out commands in the DRAW
              language one at a time."
60 PRINT:PRINT"You can also enter CLS to clear the
              screen and END to stop the program."
70 PRINT:PRINT"Press SPACE to start ";
80 IF INPUT$(1)<>" " THEN 80
90 CLS
100 LOCATE 1,1:PRINT SPACE$(40);
110 LOCATE 1,1:LINE INPUT "";A$
120 IF A$="CLS" OR A$="cls" THEN CLS:DRAW "BM160,100":GOTO 100
130 IF A$="END" OR A$="end" THEN END
140 DRAW A$:GOTO 100
150 BEEP:RESUME 100
```

GET and PUT

These commands are only available in Disk Basic

GET and PUT are a powerful pair of commands that allow images to be 'picked up' off the graphics screen and copied and moved around it at high speed. The basic idea is that GET captures an area of the graphics screen in an ordinary Basic array. This area can then be PUT back onto the screen in different ways and places.

To capture an area of the screen, you use:

```
GET (x1,y1)-(x2,y2),array
```

where (x1,y1) is the top left corner and (x2,y2) is the bottom right corner of a rectangle surrounding the area you wish to capture. 'array' is name of the array you want the image copied to. This can be any numeric type though if you use an integer array, you may be able to work out your own ways of manipulating the image once it is captured in an array.

There's a problem in working out how big the array needs to be. Generally, if you make the array have as many elements as there are pixels in the area you are trying to capture, it will work but you will be wasting memory space. The actual number of bytes needed to store the picture is:

y*INT((2*x+7)/8)+4 for medium resolution graphics

y*INT((x+7)/8)+4 for high resolution graphics

where 'x' and 'y' are the number of pixels across and down you are trying to capture. If you use an integer array, each element holds two bytes. You can therefore halve the result of the above calculation to get the number of elements needed. For example, to GET a 10x20 block of pixels off a medium resolution screen, we could use an integer array of (20*INT((2*10)+7)/8)+4)/2 elements. All we need is to DIM PIC%(32).

Once you've got the array size right (and you should find your Advance has enough memory to let you be lazy and use a massively oversized array!), GET and PUT are very easy. Use GET to grab the image. Replace it on the screen with:

PUT (x,y),array,mode

(x,y) are the co-ordinates where you want to place the drawing. 'array' is the name of the array that has got the image in it. 'mode' sets exactly how the stored image will be drawn onto the screen. Some of the words used here will already be familiar to you. 'mode' can be:

PSET
PRESET
XOR
OR
AND

and will automatically be XOR if you don't specify anything else. PSET is our old friend for setting pixels on. In this case, it just means that the image is copied as it was back onto the screen.

PRESET will cause every colour to be inverted. What was colour 0 will become colour 3. What was colour 1 will become colour 2. And vice versa. Try it and you'll find that PRESET produces a negative image. By GETing an image and then immediately PUTing it back with PRESET, you'll cause it to go negative on the screen!

OR will superimpose the image on any image that is already on the screen while AND works like a mask ... the image will only appear where there is already a drawing below.

The most useful 'mode' is XOR. XOR superimposes the image onto an existing image except it inverts the existing image wherever the two cross! This gives a rather odd property - you can do two XORed PUTs onto a complex background and you'll leave the background totally unchanged. This is very useful for animation. You XOR the object onto the screen, work out its new position, XOR it onto its old position again and go back to the first step!

Even so, the only way to see PUT and GET working is to try them:

```
10 REM Get and PUT demo
20 DIM A%(125):SCREEN 1,0:CLS
50 REM Capture a face
60 CIRCLE (15,15),15
70 CIRCLE (10,12),4:CIRCLE(20,12),4
80 CIRCLE (15,13),10,3,3.927,5.498
90 GET (0,0)-(30,30),A%
100 REM Copy it all over
110 CLS:FOR I=1 TO 500:NEXT I
120 FOR Y=0 TO 150 STEP 30
125 FOR X=0 TO 270 STEP 30
130 PUT (X,Y),A%,PSET
140 NEXT X,Y
150 REM pause and do it again
160 CLS:FOR I=1 TO 500:NEXT I
170 FOR Y=0 TO 150 STEP 30
175 FOR X=0 TO 270 STEP 30
180 PUT (X,Y),A%,PRESET
190 NEXT X,Y
200 GOTO 110
```

```
10 REM Movement with GET & PUT
20 DIM A%(125):SCREEN 1,0:CLS
30 DEF FN R(X)=5*INT(RND(1)*3-1)
40 REM Capture a face
50 CIRCLE (15,15),15
60 CIRCLE (10,12),4:CIRCLE(20,12),4
70 CIRCLE (15,13),10,3,3.927,5.498
80 GET (0,0)-(30,30),A%
90 REM Set up a background display
100 CLS
110 FOR I=5 TO 15:LOCATE I,8:PRINT"Animation with GET & PUT":NEXT I
120 REM Move the face around
130 X=160:Y=100 ' start in centre
140 PUT (X,Y),A%,XOR
150 NEWX=X+FNR(5):NEWY=Y+FNR(Y)
160 IF NEWX<0 THEN NEWX=0
170 IF NEWY<0 THEN NEWY=0
180 IF NEWX>290 THEN NEWX=290
190 IF NEWY>160 THEN NEWY=160
200 PUT (X,Y),A%,XOR:X=NEWX:Y=NEWY
210 GOTO 140
```

```
10 REM MORE GET & PUT
20 DIM A%(125):SCREEN 1,0:CLS
30 DEF FN R(X)=5*INT(RND(1)*3-1)
40 REM CAPTURE A CIRCLE
50 CIRCLE (15,15)
60 GET (0,0)-(30,30),A%
70 REM TRY DIFFERENT STEP VALUES AND
80 REM DIFFERENT 'MODES' WITH PUT
90 CLS
100 FOR X=0 TO 2*3.141593 STEP .05
110 PUT (X*40,70+SIN(X)*70),A%,R
120 NEXT X
```

6.21 Advance Basic Quick Reference Section

This is a list of Basic commands, statements and functions available on the Advance Personal Computer.

The form of the entries is as follows: alphabetic order; term underlined with an indication of whether it is command, statement or function; syntax of term; statement of purpose; remarks (if any); then C for "available with cassette Basic" and/or D for "available with disc Basic".

A command is an instruction that returns control to the operating system after the instruction has been performed. A statement is an instruction that is entered as part of a program source line.A function converts a value into some other value according to a fixed formula. These functions are built-in to the BASIC, and may be called from any program without further definition. Arguments to functions are always enclosed in parentheses. In syntax as set out, arguments abbreviated as follows: X,Y represent any numeric expression; I,J represent any integer expression; X$,Y$ represent string expressions.

General notes:

"filespec" : string expression of form device:filename, enclosed in inverted commas. Device = Casl (cassette),A:,B:,disc drives.

(...) indicates optional repetition.<...> indicates description of possible substitution that will be further defined, unless wholly obvious, in the remarks that follow. [.....] indicate that what is enclosed by the brackets is optional.

For further explanations, a much fuller treatment, and examples, see the Advance Basic Manual.

ABS Function. ABS(X)
Returns absolute value of expression X.
C,D.

ASC Function. ASC(X$)
Returns ASCII code for first character of X$. C,D.

ATN Function. ATN(X).
Returns the arctangent of X, where X is in radians. C,D.

AUTO Command. AUTO [<line number>][,<increment>].
AUTO begins numbering at <line number>, increments each subsequent line number by <increment>. Default on both = 10, if <line number> omitted, begins at 0. If AUTO generates a number already in use, an asterisk indicates this. Press ENTER to keep existing line, and generate next line number. CTRL-Break to stop AUTO. C,D.

BEEP Statement. BEEP.
Sounds speaker at 800hz for 1/4 sec.C,D.

BLOAD Command. BLOAD <filespec> [,<offset>].
Loads specified memory image file into memory from disc or cassette. If using cassette Basic, CAS1 is assumed. <offset> is a numeric expression in range 0 - 65535, the address at which loading is to start in segment declared by last DEF SEG statement. If offset omitted, offset specified at BSAVE used. BLOAD does not check address range: do not load over Basic or DOS. C,D.

BSAVE Command. BSAVE <filespec>,<offset>,<length>.
Saves contents of specified area of memory as a disc or cassette file. <offset> as for BLOAD; saves from this address in segment declared by last DEF SEG statement. <length> is numeric expression in range 1 - 65535 = length in bytes of memory image file to be saved. C,D.

CALL Statement. CALL <variable name> [(<argument list>)]
Calls a machine language subroutine, or compiler routine written in another high level language."variable name" contains an address that is the starting point in memory of the subroutine. Called as an offset into current segment of memory, as defined by last DEF SEG statement. <argument list> specifies variables to be passed as arguments to external subroutine.

CDBL Function. CDBL(X). Converts X to a double precision number.

CHAIN Statement. CHAIN [MERGE] <filespec>[,[<line number exp>] [,ALL][,DELETE <range>]]
Calls another program, passing control to it, can pass variables. <filespec> specifies program to be called; <line number> is starting point in chained program - if omitted, execution begins at first line. Not affected by RENUM command. "ALL" specifies that every variable in current program is passed to called

program. If ALL omitted, current program must contain COMMON statement listing variables to be passed. "MERGE" option allows subroutine to be brought into Basic program as an overlay: current program and called program are merged. Called program must be ASCII file to be merged. After overlay used, desirable to delete it using DELETE option, so that new overlay can be brought in. D.

CHR$ Function. CHRS$ (I)
Replaces an ASCII code I by its character equivalent. C.D.

xCINT Function. CINT(X)
Converts X to an integer. C,D.

CIRCLE Statement. CIRCLE (<xcentre>,<ycentre>),<radius>[,<color>[,<start>,<end>,[,<aspect>]]]
Draws an ellipse with specified centre and radius. <xcentre.ycentre> coordinates x,y for centres of circles; <radius> is radius;<color> specifies colour of ellipse, see COLOR statement. 0= background colour, 1-3 foreground colours (in medium resolution.) <start,end> are angles in radians. Specify where ellipse begins and ends. <aspect> is the aspect ratio, ratio of x radius to y radius. D.

CLEAR Statement. CLEAR [,[<exp1>][,<exp2>]]
<exp1> is a memory location that, if specified, sets highest location available for use by Advance Basic. <exp2> sets aside stack space for Basic. CLEAR closes all files, clears all COMMON variables, resets numeric variables and arrays to zero, resets stack and string space, resets all string variables and arrays to null, releases all disc buffers, deletes all DEF FN statements. C,D.

CLOSE Statement. CLOSE [[#]<file number>[,[#]<file number...>]].
Concludes I/O to a file. <file number> is the number used when file was opened. Association between file and file number terminates with CLOSE. CLOSE for sequential output writes final buffer of output. C,D.

CLS Statement. CLS . Erases screen. C,D.

COLOR Statement.
Varies with mode. (1) Text mode.
COLOR [<foreground>] [,[<background>][,<border>]
Sets colours for screen.
<foreground> is number in range 0-31, 0-15 set colour,16-31, same colours, but flashing. For list of colours, see p.6-21. <background> is number in range 0-7, sets background colour. <border> is number in range 1-15, sets colour for border around screen.
(2) Graphics Mode.
COLOR [<background>[,<palette>]
Sets screen colour for medium resolution graphics. <background> is number 0-15, standing for colour range, see p.6-21. Colour of background. <palette> number selects foreground colours, see p.6-

24. Any parameters outside specified ranges, returns "Illegal function call" error.Foreground colour may be same as background colour, making displayed characters invisible. Any parameter can be omitted, previous value retained.

COM Statement. COM (n) ON
COM (n) OFF
COM (n) STOP

Enables or disables event trapping of communications activity on the specified channel. "n" is the number of the communications adapter, 1 or 2. This statement allows communications event trapping by ON COM statement. With COM (n) ON, and specification of non-zero line in ON COM statement, Basic checks between every statement to see if there has been communication activity - if so, ON COM is executed. D.

COMMON Statement. COMMON <list of variables>. Passes variables to a chained program. Used in conjunction with CHAIN statement. Recommended tnat it is used at beginning of program. Same variable cannot occur in more than one COMMON statement.Array variables specified by appending "()"to variable chain. D.

CONT Command. CONT
Continues program execution after BREAK, STOP or END, from where break occurred. Invalid if program has been edited during break. C,D.

COS Function. COS(X)
Returns cosine of X. where X in radians. C,D.

CSNG Function. CSNG(X)
Converts X to single precision number. C,D.

CSRLIN Function. x = CSRLIN
Stores current line position of cursor in numeric variable. Value returned in range 1 - 24. C,D.

CVI,CVS,CVD Functions. CVI (<2-byte string>)
CVS (<4-byte string>)
CVD (<8-byte string>).

Converts string values to numeric values. Numeric values read in from random access disc file must be converted from strings back into numbers. I converts 2-byte string to integer, S 4-byte string to single precision number, D 8-byte string to double precision number. D.

DATA Statement. DATA <list of constants>
Stores numeric and string constants accessed by program's READ statement(s). Nonexecutable; placed anywhere in program; can contain as many constants as will fit on a line (separated by commas). READ statements access DATA statements in order. Constants must not be expressions. Must agree with type (numeric or string) in READ statement. C,D.

DATE$ Statement. DATE$ = <string expression>
Sets current date. <string expression> should return string in one of following forms:mm-dd-yy
mm-dd-yyyy
mm/dd/yy
mm/dd/yyyy
D.

DATE$$ Function. x$ = DATE$
Retrieves current date,in form mm-dd-yyyy. D.

DEF FN Statement. DEF FN name [(<parameter list>)] = <function definition>
Defines and names a function that is written by the user. "name" must be legal variable name, becomes name of function. <parameter list> = variable names in function definition that are replaced when function is called; items in list separated by commas. <function definition> = expression that performs operation of function. Variable names appearing in expression only define function - do not affect program variables with same name. If variable name used in function definition appears in parameter list, value of parameter supplied when function is called - if not, current value of variable used.

Variables in parameter list represent one-to-one argument variables or variables given in function call.

May define either numeric or string functions - right type must be specified, otherwise type mismatch. C,D.

DEF INT/SNG/DBL/STR Statements. DEFINTrange(s) of letters
DEFSNGrange(s) of letters
DEFDBLrange(s) of letters
DEFSTRrange(s) of letters
Declares variable types as integer, single precision, double precision or string. Any variable names beginning with the letter(s) specified in the range considered the type of variable specified. C,D.

DEF SEG Statement. DEF SEG [= address]
Assigns current segment address to be referenced by a subsequent BLOAD, BSAVE, CALL,CALLS or POKE statement or by USR or PEEK function. <address> is numeric expression returning unsigned integer in range 0-65535. Saved for use as segment required by subsequent statements, functions. If omitted, data segment used as default. C,D.

DEF USR Statement. DEF USR[<Digit>] = <integer expression>
Specifies starting address of assembly language subroutine. <Digit> is 1-9, corresponds to number of USR routine whose address is being specified. If omitted, DEF USR0 assumed. Value of <integer expression> is starting address of USR routine. Any number can be used, allowing access to any number of subroutines. C,D

DELETE Command. DELETE [<line number>] [-<line number>]
Deletes program lines. C,D.

DIM Statement. DIM <list of subscripted variables>
Specifies maximum values for array variable subscripts and allocates storage accordingly. If array variable name used without DIM statement, maximum value of array's subscript(s) is assumed to be 10. DIM statement sets all elements of specified arrays, to initial value of zero. C,D.

DRAW Statement. DRAW <string expression>
Draws lines. <string expression> is a subcommand specifying direction and distance from current graphics position - last one defined with LINE or PSET, or, by default, the centre of the screen. Subcommands: U [<n>] - Move up scale factor*n points (and so on for following subcommands) D, down; L, left; R, right; diagonally up and right; F, diagonally up and left; G, diagonally down and left; H, diagonally down and right.

M x,y Move absolute and relative. If x is preceded by + or -, x and y (which will itself have a + or - specification numbered in pixels) are added to the current graphics position and connected by a line. Otherwise, line drawn to point x,y from current cursor position.

Following prefix commands may precede any of above movement commands: B, move but do not plot; N, move, but return to original position; A n, set angle n. "n" may range from 0-3, 0=0 degrees, 1=90 degrees, 2=180 degrees, 3=270 degrees; C n, set colour n, may range from 0-3; S n, set scale factor, may range from 1-255.

X<string expression>. Execute substring; allows execution of second substring from a string. D.

EDIT Command. EDIT <line number>
Enters edit mode at the specified line. C,D

END Statement. END
Terminates program execution, closes all files, returns to command level. C,D

EOF Function. EOF (<file number>)
Tests for the end-of-file condition. Returns -1 (true) if end of sequential file has been reached. C,D

ERASE STATEMENT ERASE <list of array variables>...
Eliminates arrays from memory. C,D

ERR&ERL Variables. var=ERR var=ERL
Variable ERR contains error code for last error, Variable ERL line number of line in which error was detected. Used in IF...THEN statements directing program flow in error handling routine. C,D

ERROR ERROR <integer expression>
Simulates occurrence of Basic error, or allows error codes to be defined by user. <integer expression> between 0 and 255. If it is same as error code already in use by Basic, error statement will simulate that error. If defining own error code, select from highest integer expressions; user defined error can be conveniently handled in error handling routine. Otherwise, Basic will return "Unprintable Error". C,D

EXP Function. EXP(X)
Calculates exponential function, ie e (base of natural logarithms) to the power of X. X must be <= 88.02969. C,D

FIELD Statement. FIELD [#] <file number>,<field width >AS<string variable>....
Allocates space for variables in a random file buffer. Before GET or PUT executed, must be FIELD statement to format random file buffer. <file number> is number under which file was opened. <field width> is number of characters to be allocated to <string variable>. Total number of bytes allocated in FIELD statement must not exceed record length specified when file opened - otherwise "Field overflow" error occurs.

Any number of FIELD statements may be executed for the same file. All FIELD statements that have been executed will remain in effect at the same time. D

FILES Statement. FILES [<filespec>]
Prints names of files residing on specified disk. <filespec> includes file name and optional device designation. If omitted, all files on currently selected disk listed. ? or * can be used as wild cards, first matching any single character, second one or more characters starting at position. D.

FIX Function. FIX(X)
Returns truncated integer part of X. Unlike INT, FIX does not return next lower number of negative X. C,D.

FOR...NEXT Statement. FOR <variable>=x TO y [STEP z]

.

.

.

NEXT [<variable>][,<variable>...]

where x,y,z are numeric expressions.

Allows a series of instructions to be performed in a loop a given number of times. <variable> used as a counter, from first, x, to last, y. Program lines following FOR are executed until NEXT statement, then counter adjusted by amount specified by STEP. Check performed to see if value of counter is now greater than final value, y. If no, Basic branches back to the statement after FOR statement, process repeated. If greater, goes to statement after NEXT. If STEP omitted, defaults to 1. C,D

FRE Function. FRE(X) FRE(X$)
X and X$ are dummy arguments. With number, returns number of bytes in memory not being used by Basic. With string, forces housecleaning, tidying garbage before returning free bytes. C,D

GET Statement. With FILES. GET [#] <file number> [,<record number>]
Reads record from random disk file into random buffer. <file number> is number under which file was opened. <record number>, up to 32767, is number of record to be read. If omitted, next record (after last GET) read into the buffer. After GET, INPUT£ and LINE INPUT£ allow reading characters from random file buffer. After GET statement has been executed use INPUT£ and LINE INPUT£ to read characters from the random file buffer.

With Graphics. GET (x1,y1)-(x2,y2),<array name> used with PUT (x1,y1),<array name>[,action verb]
where (x1,y1)-(x2,y2) is a rectangle on the screen.

<Array name> name assigned to place that will hold image. Array can be any type except string. Must be dimensioned large enough to hold whole of image.In PUT, <x1,y1> gives co-ordinate of top left-hand corner of image.(If image transferred is too large, error will be returned). <action verb> is one of:PSET, PRESET, AND, OR, XOR. PSET transfers data on to screen verbatim.PRESET same but negative image produced. AND is used to transfer image if an image already exists on screen. OR is used to superimpose image on to an existing image. XOR causes points on screen to be inverted where a point exists in the array image. Used for animation, see p.6-30.C,D

GOSUB & RETURN Statements. GOSUB<line number>

.
.
.
.
RETURN [<line number>]

Branches to, returns from, subroutine. <line number> is first line of subroutine. Subroutine can be called any number of times from within program. May also be called from within another subroutine. Nesting limited only by available memory. RETURN makes Basic branch back to statement following most recent GOSUB statement. <line number> option causes return to specific line.C,D

GOTO Statement. GOTO<line number>
Branches unconditionally out of normal program sequence to specified line number. C,D

HEX$ Function. HEX$(X)
Returns string that represents hexadecimal value of decimal argument. X is rounded to integer before HEX(X) is evaluated. C,D.

IF THEN,ELSE,GOTO Statements.

IF <expression> THEN <statement(s)> ELSE <statement(s)>

IF <expression> GOTO <line number> ELSE <statement(s)>

Makes a decision regarding program flow based on result returned by an expression. <statement(s)> = statement or sequence of statements (with : between), or line number to branch to. If result of <expression> is not zero, THEN or GOTO clause is executed. If zero, THEN,GOTO clauses ignored, ELSE executed. Execution continues with next executable statement. Comma allowed before THEN.

IF...THEN...ELSE statements can be nested. Nesting limited only by length of line. C,D

INKEY$ Function. INKEY$
Returns a 1-character string containing character read from terminal, or null string if no character pending at terminal. C,D

INP Function. INP(I)
Returns byte read from port I. I must be in range 0-65535. Complementary to OUT. C,D

INPUT Statement.
INPUT [;] [<"prompt string">;]<list of variables>
Allows input from keyboard during program execution. When INPUT statement encountered, program execution pauses, ? printed to indicate program waiting for data. If <prompt string> included, string printed before ? To suppress ? use comma instead of semi-colon after <prompt string>. For D, [;] and enter by user does not produce carriage return on screen.

Data entered is assigned to variable(s) given in <variable list>. Number supplied, separated by commas, must be same as list, same type. Otherwise, error message will be returned. C,D.

INPUT# Statement. INPUT#<file number>,<variable list>
Reads data from sequential device or file, assigns them to program variables. <file number> is number used when file was OPENed for input. <variable list> contains variable names to be assigned to items in file - type must match. Whatever source used, data items in file should appear just as if data were being typed for INPUT. With numeric values, leading spaces, carriage returns, linefeed, ignored. Similarly, if Basic scanning file for string item, will ignore leading spaces, carriage return, linefeed - first character encountered assumed to be string start. If string begins with ", second" marks end of string - so no " within "". C,D

INPUT$ Function. INPUT$ (X[,[#]Y])
Returns string of X characters, read from file number Y. If file number not specified, input from keyboard, with no screen echo. All control characters passed through except CTRL-break, used to interrupt execution of INPUT$ function. C,D.

INSTR Function. INSTR([I,]X$,Y$)
Searches for first occurrence of Y$ in X$, returns position at which match is found. Optional I sets position for search to start. If X$ null,Y$ not in X$, or I greater than length of X$, INSTR returns 0. If Y$ null, returns I or 1. C,D

INT Function. INT(X)
Returns largest integer <=X. C,D.

KEY Statement. KEYn,x$
KEY LIST
KEY ON
KEY OFF

Assigns soft key values to function keys and displays values. n is number (1-10) of function key. x$ is text assigned to specified key. KEY statement allows function keys to be designated for soft key functions. Each F key can be assigned 15 byte string that when key is pressed will be input to Basic.

Initially soft keys assigned following values: F1 LIST <space>, F2 RUN <enter>, F3 LOAD",F4 SAVE",F5 CONT <enter>, F6 ,"LPT1:", <enter>, F7 TRON <enter>, F8 TROFF<enter>, F9 KEY<space>, F10 EDIT<space>

KEY ON displays soft key values on 25th line of screen, first six characters of each key only. For all 10, use Mode 80. KEY OFF erases soft key display, making line available for programming. KEY LIST displays all 10 values onscreen, all 15 characters displayed.

Assigning null string to soft key disables function key as soft key. If value entered for n not in range 1-10, "Illegal function call" error returned.

When softkey called, INKEY$ function returns one character of softkey string. C,D.

KEY(N) Statement. KEY(n)ON
KEY(n)OFF
KEY(n)STOP

Enables or disables event trapping of soft key or cursor direction key activity for specifed function key.

(n) is number of key, as F-1-10, then 11-14 for cursor up,left, right,down. While trapping enabled, if non-zero line number specified in ON KEY statement (q.v.),Basic checks between every statement to see if specified key was used. If so, ON KEY statement executed. KEY(n)OFF disables event trap, event not remembered. KEY(n)STOP disables event trap, but if event occurs, remebered and ON KEY executed as soon as trapping enabled. D

KILL Statement. KILL<filespec>
Deletes file from disk. D.

LEFT$ Function. LEFT$(X$,I)
Returns string comprising the leftmost I characters of X$. I must be in range 1-255. If I greater than number of characters in X$ (LEN (X$)), entire string returned. If I = 0, null string returned. C,D.

LEN Function. LEN(X$)
Returns number of characters in X$. Nonprinting characters and blanks are counted. C,D

LET Statement. [LET] <variable>=<expression>
Assigns value of expression to variable. LET is optional. C,D

LINE Statement. LINE[(x1,y1)]-x2,y2 [,[<color>][,b[f]]]
Draws a line or a box on the screen.

(x1,y1) is optional co-ordinate for starting point of line. (x2,y2) is ending point. <color> is colour of line. Can be used

with ,b or, bf. ,b draws box with co-ordinates as opposite corners, ,bf filled box.

When out of range co-ordinates given, closest legal value assigned. Co-ordinate form STEP (xoffset,yoffset can be used in place of absolute co-ordinate. If,for example, most recent point referenced was (0,0) the statement LINE STEP (10,5) would specify a point at offset 10 from x and 5 from y. If STEP option used for second co-ordinate on LINE statement, it is relative to first co-ordinate in statement. C,D.

LINE INPUT Statement. LINE INPUT[;][<"prompt string">;],<string variable>
Inputs entire line (up to 254 characters)to a string variable, without use of delimiters. <"prompt string"> is string literal printed at terminal before input is accepted. No ? unless part of string. All input from end of prompt string to carriage return assigned to <string variable>. If linefeed/carriage return, in that order, encountered, both characters echoed, carriage return ignored, linefeed goes in <string variable>, data input continues. C,D.

LINE INPUT# Statement. LINE INPUT <file number>,<string variable>
Reads entire line (up to 254 characters) without delimiters, from sequential data file to string variable. <file number> is number under which file was OPENed. <string variable> is variable to which line will be assigned. LINE INPUT# reads all characters in sequential file up to carriage return, skips over carriage return/linefeed sequence. Next LINE INPUT# reads all characters up to next carriage return. If linefeed/carriage return sequence encountered, preserved. C,D.

LIST Command. LIST[<line number>]
LIST[<linenumber>][-[<line number>]][<filespec>]
Lists all or part of program currently in memory at terminal. If Line number omitted, whole program listed. If second syntax used, and only 1st line specified, that and all higher numbers specified. If only 2nd line specified, all lines from beginning through that line listed. If both, range listed. C,D

LLIST Command. LLIST [<line number>[-[<line number>]]]
Lists all or part of program currently in memory at line printer, assuming 132-character-wide printer. Options as for LIST, syntax 2. C,D.

LOAD Command. LOAD <filespec>[,R]
Loads program from disk or cassette into memory. <filespec> must include filename used when file was saved. R option runs program after in has been loaded. LOAD closes all open files, deletes all variable lines and program lines currently in memory. However, if R option, all open data files kept open - may be used to chain segments or programs. Information may be passed between programs using data files. C,D.

LOC function. LOC(<file number>)

With random disk files, LOC returns record number of last record read or written. With sequential files, LOC returns number of records read from, or written to, file since it was opened. Basic reads from any file opened for sequential input, so LOC retuns 1 before input. For communications file, LOC(X) used to determine if characters in input queue remain to be read. D.

LOC Statement. LOCATE [row][,[col][,[cursor][,[start][stop]]]]
Moves cursor to specified position. Optional parameters turn blinking cursor on/of, define vertical start and stop lines, size of cursor.

<row> is vertical line number on screen, expressed as number 1 - 25. <col> is column number on screen (1-40 or 1-80.) <cursor> is Boolean value indicating whether cursor should be visible or not. 0=off,1=on. <start> is cursor starting line, <stop> is cursor stop line, range 0-31. Any omitted parameter means previous value is assumed. Values outside ranges result in "Illegal function call error". C,D.

LOF Function. LOF(<file number>)
Returns length of file in bytes. <file number> is that used when file opened. D.

LOG Function. LOG(X)
Returns natural logarithm of X. X must be greater than zero. C,D.

LPOS Function. LPOS(X)
Returns current position of line printer head within line printer buffer. For Disk Basic, X is number indicating which line printer is being tested, 0/1,2 or 3. C,D.

LPRINT & LPRINT USING Statements. LPRINT [<list of expressions<]
LPRINT<string expression>;<list of expressions>

Prints data at the line printer.<list of expressions> list of numeric or string expressions to be printed. Expressions punctuated by commas, semicolons. <string expression> gives format to be used, using special formatting characters.

Print Positions

Position of each printed item is determined by the punctuation used to separate the items in the list. Basic divides line into print zones of 14 spaces. Comma causes next value to be printed at beginning of next zone; semi-colon causes it to be printed immediately after last value; spaces have semicolon effect.

If there is a comma, or semicolon, at the end of the list of expressions, next LPRINT statement begins printing on the same line. If none of these, and carriage return inserted, Basic goes to next line. If printed line longer than width, Basic continues print on next line.

Printed numbers are always followed by a space. Positive preceded by a space, negative by -. Numbers are printed in different formats depending on whether they are single or double precision, and possibility of representing them accurately in unscaled format - otherwise scaled.

Special Formatting Characters and String Fields

With LPRINT USING, one of 3 formatting characters can be used to format string field: "!" specifies only first character printed; "\N SPACES\ specifies that 2+n cnaracters from string to be printed. If string longer than field, extra characters ignored. If field longer than string, string left-justified in field and padded with spaces on right. "&" specifies variable length string field: when so specified, string output without modifications.

Special characters for numeric fields: # . + - ** $$ **$

, ^^^^ %

represents each digit position - number sign. Always filled - if fewer digits, filled with spaces.

"." (Decimal point) may be inserted at any position in the field. If the format string specifies that a digit is to precede the decimal point, it will always be printed. Numbers are rounded as necessary.

+ at beginning or end of format string causes number to be printed with sign (+/-),before or after.

- at end of format field causes negative numbers to be printed with trailing minus sign.

** at beginning of format string causes leading spaces in mumeric field to appear with trailing minus sign, ** also specifies positions for two more digits.

$$ causes dollar sign to be printed to immediate left of formatted number, specifies two more digit positions, one is $ sign. No exponential format with $$. Negative numbers cannot be used unless - sign trails to right.

$ combines effects of above two symbols. Leading spaces asterisk filled, and a $ sign is printed before number.$ specifies three more digit positions, one of which is $. No exponentials with **$. When negative numbers printed, - appears immediately to left of $.

","(Comma) to left of decimal point in formatting string causes comma to be printed to left of every 3rd. digit to left of decimal point. Comma at end of format string printed as part of string. Comma specifies another digit position. Comma has no effect if used with exponential (^^^^) format.

^^^^ (carets or up-arrows) may be placed after digit position characters to specify exponential format. Allow space for E+xx to be printed. Any decimal point position may be specified. Significant digits left-justified, exponent adjusted. Unless leading + or trailing +/- specified, one digit position will be used to left of decimal point to print space or - .

Underscore _ in format string causes next character to be output as literal character: may itself be underscored by placing "_ _" in the format string.

% printed if number is larger than specified field. If rounding causes the number to exceed the field ,% printed in front of rounded number.

In the Advance implementation of GW-BASIC, LPRINT assumes 80-character wide printer. To avoid skipped line after exactly 80 characters, print ; at end of line. C,D.

LSET & RSET Statements.
LSET<string variable>=<string expression>
RSET <string variable>=<string expression>
Moves data from memory to random file buffer in preparation for PUT statement. <string variable> defined in FIELD statement, <string expression> data to go in field. If <string expression> requires fewer bytes than were fielded to <string variable>, LSET left-justifies string in field, RSET right justifies. Spaces pad out.If string too long, characters dropped from right. Numeric values must be converted to strings before LSET or RSET. See MKI$,MKS$,MKD$. D.

MERGE Command. MERGE<filespec>
Merges specified file into program currently in memory. <filespec> must include filename used when saved, must be in ASCII format. If any lines in data file have same line numbers as program in memory, lines from file replace those in memory. After MERGE, Basic returns to command level. C,D.

MID$ Statement. MID$(<string exp1>,n[,m])=<string exp2>
Replaces a portion of one string with another string. n,m are integer expressions, and <string exp1/2> are string expressions. Characters in <string exp1> are replaced by characters in <string exp2>. Optional "m" refers to number of characters used to replace, if omitted, all of <string exp2> used. Replacement never goes beyond orginal length of <string exp1>. C,D.

MID$ Function. MID$(X$,n[,m])
Returns a string of length m characters from X$, beginning with nth character. n,m must be 1-255. If m omitted, or if there are fewer than m characters to right of nth character, all rightmost characters beginning with the nth character are returned. If n < no. of characters in X$ (LEN(X$)), null string returned. C,D.

MKI$,MKS$,MKD$ Functions. MKI$(<integer expression>

MKS$(<single precision expression>)
MKD$(<double precision expression>)

Converts numeric values to string values. MKI$ converts integer to 2-byte string, MKS$ single precision number to 4-byte string,MKD$ converts double precision number to 8-byte string. D.

MOTOR MOTOR(state)
Turns cassette motor on or off. <state> is Boolean value (0 or 1...) indicating off or on. If state omitted, MOTOR switches the motor to the opposite state than current one. C,D.

NAME Statement. NAME <old filename> AS <new filename>
Changes name of disk file. <old filename> must exist and <new filename>must not exist, else error. File may not be renamed with new drive designation - otherwise "Rename Across Disks" error. File with new name in same space as before. D.

NEW Command. NEW
Deletes program currently in memory and clears all variables, closes all files, turns tracing off. C,D.

OCT$ Function. OCT$(X)
Returns a string that represents octal value of digital argument. X rounded to integer before OCT$(X) evaluated. C,D.

ON COM Statment. ON COM(n) GOSUB <line number>
Specifies first line number of subroutine to be performed when activity occurs on communications channel. n is number of communications channel. <line number> is number of first line of subroutine. If zero, communications event trap disabled. See COM ON/OFF/STOP for further control of event trapping on communications activity.

Event trapping only takes place when Basic is executing a program; disabled when error trap occurs. When event trap occurs, automatic COM STOP activated to avoid recursive trapping. RETURN from trapping subroutine sets COM ON automatically, unless explicit COM OFF performed inside subroutine.

ON ERROR GOTO Statement. ON ERROR GOTO <line number>
Enables error handling and specifies first line of error handling routine. If line number 0, error handling disabled. Subsequent errors will print error and stop execution. If line number does not exist, "Undefined line" error. If error occurs during execution of error handling routine, that error message printed and execution terminates. Error trapping does not occur within error handling routine. C,D

ON...GOSUB & ON...GOTO Statements.

ON<expression>GOTO<list of line numbers>
ON<expression>GOSUB<list of line numbers>

Branches to one of several specified line numbers, depending on

value returned when an expression evaluated.

Value of <expression> determines which line number in list will be used for branching, eg if value is 3, 3rd line number will be destination. (If value is noninteger, fractional portion rounded.) In ON...GOSUB, each line number must be first line number of subroutine. If value of <expression> is zero or greater than number of items in list (but <=255) Basic continues with next executable statement. If value negative, or >255, "Illegal function call"error.

ON KEY Statement. ON KEY(n) GOSUB <line number>
Specifies first line number of subroutine to be performed when specified function or cursor direction key is pressed. <line number> is first line number of subroutine. n is as KEY and KEY(n) statements,q.v.

<line number> of zero disables event trap.

ON KEY statement only executed if a KEY (n)ON statement has been executed to enable event trapping. If enabled, and if <line number> in ON KEY statement is not zero, Basic checks between statements to see if specified function or cursor direction key has been pressed. If so GOSUB is performed to specified line. If KEY(n)OFF statement has been executed for specified key, GOSUB is not performed and is not remembered.

If a KEY STOP statement has been executed for specified key, GOSUB not performed, but will be as soon as a Key(n) ON statement is executed. When an event trap occurs (i.e.GOSUB executed)an automatic KEY(n) STOP is executed so that recursive traps cannot take place. The RETURN from trapping routine will automatically perform a KEY(n) ON statement unless an explicit Key(n) OFF was performed inside the subroutine.

RETURN <line number> form of RETURN statement may be used to return to a specific line number from the trapping subroutine. Take care, since other GOSUBS WHILES and FORS may be active, resulting in possibility of "For without NEXT error" etc.

Event trapping does not take place when no program being executed, event trapping disabled when error trap occurs.

When a key is trapped, that occurrence of the key is destroyed. Cannot test for key using INPUT or INKEY$. If different functions to be assigned to particular keys, set up a different subroutine for each key.

ON PEN Statement. ON PEN GOSUB <line number>
Specifies first line number of subroutine to be performed when lightpen is activated. <line number> is first line. <line number> of zero disables pen event trap.

ONPEN statement only executed if a PEN ON statement has been executed to enable event trapping. If enabled, and if <line number> in ON PEN statement not zero, Basic checks between statements to see if lightpen has been activated.If so a GOSUB is performed to specified line. If PEN OFF statement has been executed for specified key, GOSUB is not performed and is not remembered.

If a PEN STOP statement has been executed GOSUB is not performed, but will be as soon as a PEN ON statement is executed. When an event trap occurs (i.e.GOSUB executed)an automatic PEN STOP is executed so that recursive traps cannot take place. The RETURN

from trapping routine will automatically perform a PEN ON statement unless an explicit PEN OFF was performed inside the subroutine.

RETURN<line number> form of RETURN statement may be used to return to a specific line number from the trapping subroutine. Take care, since other GOSUBS WHILES and FORS may be active, resulting in possibility of "For without NEXT error" etc.

Event trapping does not take place when no program being executed, event trapping disabled when error trap occurs. D.

ON STRIG(n) Statement. ON STRIG(n) GOSUB <line number>

Specifies first line number of subroutine to be performed when joystick trigger is pressed. n is number of joystick trigger. <line number> is number of first line of subroutine.

<line number> of zero disables event trap.

ON STRIG statement is only executed if a STRIG ON statement has been executed to enable event trapping. If enabled, and if <line number> in ON STRIG statement is not zero, Basic checks between statements to see if specified function or cursor direction key has been pressed. If so a GOSUB is performed to specified line. If STRIG OFF statement has been executed for specified key, GOSUB is not performed and is not remembered.

If aSTRIG STOP statement has been executed for specified key, GOSUB not performed, but will be as soon as a STRIG ON statement is executed. When an event trap occurs (i.e.GOSUB executed)an automatic STRIG STOP is executed so that recursive traps cannot take place. The RETURN from trapping routine will automatically perform a STRIG ON statement unless an explicit STRIG OFF was performed inside the subroutine.

RETURN <line number> form of RETURN statement may be used to return to a specific line number from the trapping subroutine. Take care, since other GOSUBS WHILES and FORS may be active, resulting in possibility of "FOR without NEXT error" etc.

Event trapping does not take place when no program being executed, event trapping disabled when error trap occurs. D.

OPEN Statement.

OPEN <mode>,[#]<file number>,<filespec>[,<record length>]

OPEN <filespec> [FOR <mode>] AS [#] <file number> [LEN = <record length>]

Allows I/O to a file or device.

<mode> is string expression whose first character is one of following: O - specifies sequential output mode. I - specifies

sequential input mode. R - specifies random input-output mode, and this is default value. A -specifies sequential output mode and sets the file pointer at the end of the file and record number as last record of the file. PRINT or WRITE statement will then extend (append) the file. Disk only.

<file number> is integer expression whose value is between 1 and 15. Number is then associated with file for as long as it is open, used to refer other disk I/O statements to the file.

<record length> is integer expression that sets record length for random files - do not use with sequential ones.

Disk file must be opened before any disk I/O operation can be performed on that file. OPEN allocates a buffer for I/O to the file and determines the mode of access that will be used with the buffer.

A file can be opened for sequential input or random access on more than one file number at a time. For output, however, only one file number at a time.

Notes on Cassette Files: if the cassette is the addressed device, and no fileneame is given, the next file on the cassette will be read. On the tape, only one file can be open at a time; other open files could be printer, keyboard and screen. C,D.

OPEN COM Statement.

OPEN "COMn: [<speed>] [,[<parity>] [,[<data>] [,[<stop>] [,RS] [,CS[n]] [,DS[n]] [,CD[n]] [,BIN] [,ASC] [,LF]]]] AS [#]<device number>

Opens and initialises a communications channel for input/output.

Comn: name of device to be opened. <speed> is baud rate, in bits per second. <parity> designates parity of device to be opened. Valid entries are, e.g. N (none),E (even), or O(odd).<data> designates number of bits per byte. Valid entries are 5,6,7,or8.<stop> designates stop bit. Valid entries 1 or 2. RS suppresses RTS (request to send). CS(n) controls CTS (clear to send). n specifies time in milliseconds before error returned, if omitted or 0 line status is not checked. (Default 1 sec.) DS(n) controls DSR (data set ready). "Device Timeout Error" will occur if DSR not detected. CD(n) controls CD (carrier detect). <device number> is number of device tc be opened.

<speed><parity><data><stop>options must be listed in this order - others in any order, after these.

LF specifies that linefeed is to be sent after carriage return. Allows communication files to be printed on a serial line printer.Nb INPUT and LINE INPUT , when used to read from a COM file that was opened with the LF option, stop when they see a

carriage return, ignoring the linefeed.

LF option superseded by BIN option. BIN opens device in binary mode, selected unless ASC is specified. In BIN mode, tabs not expanded to spaces, carriage return not forced at end-of-line, Control Z not treated as end of file. When channel closed, Control Z will not be sent over the RS232 line.

In ASC mode, tabs expanded, carriage returns are forced at the end-of-line, Control Z is treated as end-of-file. When channel is closed, Control Z sent over RS232 line. D.

OPTION BASE Statement. OPTION BASE n
Declares minumum value for array subscripts. n is 1 or 0. Default base 0. If used, must be coded before defining or using arrays. C,D

OUT Statement. OUTI,J
Sends a byte to a machine output port. I is port number. J is data to be transmitted. C,D

PAINT Statement. PAINT (<xstart>,<ystart>)[,<paint color>[,<border color>]]
Fills graphics figure with colour specified.

<xstart,ystart> are co-ordinates where painting is to begin. Should be within area, not at border. <paint color> is number of colour. If not specified, foreground colour used. <border color> is colour of border, when encountered, painting stops. If not specified, <paint colour> used. D.

PEEK Function. PEEK(I)
Returns byte read from indicated memory location (I).

Returned value integer in range 0 - 255. I is offset from current segment, defined by last DEF SEG statement. C,D.

PEN Statement/Function.
PEN ON
PEN OFF
PEN STOP
x=PEN(n)

PEN ON enables lightpen read function and event trapping. PEN OFF disables. PEN STOP (D only) disables functions and trapping, but remembers event for trapping when enabled. Function reads light pen co-ordinates. (n) is numeric expression range 1-9, returning values as follows: 0/-1 - was pen down since last poll? -1,if down, 0 if not. 1 - returns x co-ordinate where pen last pressed. 2 - y co-ordinate same. 3 - current pen switch value, -1 if down, 0 if up. 4 returns last known valid x co-ordinate.5, y, same. 6 - character row position where pen last pressed. 7 - column position, same. 8 - last known character row where pen was positioned. 9 - last known valid character column where pen positioned.

Pen initially off, PEN ON executed before any pen read function calls. Call to pen function with PEN OFF gives "Illegal function call" error. See ON PEN for event trapping. Lightpen may return inaccurate values in border areas of screen. C,D.

PLAY Statement. PLAY <string expression> . Plays music as specified by string expression. Like DRAW, PLAY embeds a macro language, here defining sequential notes, into single statement. Turns music into character string, andvice versa.

Subcommands for <string expression> are :

A-G [# or +,-] - plays note in range A-G, + sharp, - flat.

L(n) - length of notes, 1 equals whole note, up to 64 being fractions of notes (eg 4 is quarter note). Length may also follow note if change of length for one note only - eg A 16 equivalent to L16A.

MF sets music (PLAY) statement and SOUND to run in foreground, i.e.,each subsequent note or sound will not start until previous note or sound has finished; this is default mode.

MB - Music (PLAY statement) and SOUND set to run in background, i.e., each note or sound placed in buffer allowing program to continue executing while note or sound plays in background.

MN - sets "music normal" so that each note plays 7/8 of time determined by length. ML- sets "music legato" so that each note will play full period set by length.

N<n> Plays note n. n may range from 0 - 84 (7 octaves); n = 0 means a rest.O<n> - sets current octave; 7 octaves numbered 0-6.

P<n> specifies pause, ranging 1-64; option corresponds to L<n>.

T<n>sets "tempo", number of L4s in 1sec, ranging from 32 - 255, default 120. Period after note causes note to play 3/2 times L multiplied by T (Tempo), may be multiple, scaled accordingly; works with P, same.

X <string> - executes substring; may be executed by appending character form of substring address to "X". D

POINT Function. <xcoordinate>,<ycoordinate>
Reads colour value of pixel from screen. <Coordinates> are coordinates of pixel to be referenced. If specified point is out of range, value -1 returned. C,D.

POKE Statement. POKE I,J
Writes a byte into memory location. I is address, in integer range 0 to 65535. Offset from current segment set by last DEF SEG statement. J is data byte. C,D.

POS Function. POS(I)
Returns current horizontal (column) position of cursor. (I)is dummy argument. Leftmost position is 1. C,D.

PRESET Statement.
PRESET (<x coordinate>,<y coordinate>)[,<color>]
Draws specified point on screen, if <color> not specified, background colour selected. <coordinates> specify pixel to be set. <color> is number assigned to color used for specified point. If out-of-range coordinate given, no action taken, error message returned. Co-ordinates can be shown as absolutes, or STEP option can be used to reference a point relative to most recent point used.

STEP <xoffset>,<yoffset>. C,D.

PRINT Statement. PRINT [<list of expressions>]
Outputs data on the screen. Expressions in list may be numeric or string expressions in quotation marks. If <list of expressions> omitted, blank line is printed. If included, values of expression printed at terminal.

Further specifications of print positions are identical to LPRINT; please refer to these. C,D.

PRINT USING Statement. PRINT USING <string exp.>;<list of expressions>.
Prints strings or numbers using a specified format. <list of expressions> comprised of string expressions or numeric expressions that are to be printed, separated by semicolons. <string exp> is a string literal (or variable) composed of special formatting characters, determining field and format of printed strings and numbers.

For specification of fields and formats, see details under LPRINT USING Statement, which are identical. C,D.

PRINT# and PRINT USING# Statements.
PRINT#<file number>,[USING<string exp>;]<list of expressions>

Writes data to a sequential file. <file number> is number used when file was opened for output. <string exp> formatting characters as for LPRINT USING. <list of expressions> expressions that will be written to file.

PRINT# does not compress data. Image of data written to file, just as would be displayed on terminal with PRINT. In list of expressions, numeric expressions should be delimited by semicolons. If commas are used as delimiters, extra blanks inserted between print fields will also be written to file. String expressions must be separated by semicolons in the list. Use explicit delimiters to format string expressions correctly in the file. If strings themselves contain commas, semicolons, significant leading blanks, carriage returns or linefeeds, write them to file surrounded by explicit quotation marks with

CHR$(34). Print statement may also be used with the USING option to control format of file.

See also WRITE Statement. C,D.

PUT Statement. PUT [#]<file number>[,<record number>].
Writes record from random buffer to random access file. <file number> is number under which file was opened. <record number> , 1-32767, is record number for record - if omitted, next available number, after last PUT, is used. PRINT ,PRINT USING and WRITE may be used to put characters in the random file buffer before executing a PUT statement. With WRITE , buffer padded with spaces up to carriage return.

PUT statement with graphics modes - syntax PUT (x1,y1),array name> [,<action verb>].
For further specification see GET statement. D.

RANDOMIZE Statement. RANDOMIZE [<expression>]
Reseeds random number generator. <expression> is integer (-32768 to 32767) used as seed, if omitted, query asking for this returned. If random number generator not reseeded, RND function returns same sequence each time program is run. To change sequence of random numbers every time program run, place a RANDOMIZE statement at the beginning of program, and change argument with each run. C,D

READ Statement. READ <list of variables>
Reads values from a DATA statement and assigns them to variables. (See DATA statement.) READ statement must always be used with DATA statement. READ assigns variables to DATA values on one-to-one basis. READ variables may be numeric or string, and values read must agree with variable types specifed, or "Syntax error" will result.

A single read statement may access one or more DATA statements (accessed in order), or several READ statements may access the same DATA statement. If number of variables in <list of variables> exceeds number of elements in DATA statement(s), "Out of data" error message printed. If number of variables specifed is fewer than number of elements in DATA statement(s), subsequent READ statements will begin reading data at first unread element. If there are no subsequent READ statements, extra data is ignored. To reread DATA statements from start, use RESTORE statement. C,D.

REM Statement. REM <remark>
Allows explanatory remarks to be inserted into program. REM statements are not executed, but are output exactly as entered when program is listed. REM statements may be branched into from a GOTO or GOSUB statement. Execution will continue with first executable statement after the REM statement. Remarks may be added to end of line by preceding remark with single quotation mark instead of :REM. C,D.

RENUM Command.
RENUM [[<new number>][,[<old number>][,<increment>]]] Renumbers program lines. <new number> is first line number to be used in new sequence, default 10. <old number> is line in current program where renumbering is to begin. Default is first line in program. <increment> is increment to be used in new sequence, default 10.

RENUM also changes all line number references following GOTO,GOSUB,THEN, ON....GOTO, ON...GOSUB, and ERL statements to reflect new line numbers. If nonexistent line number appears after one of these statements, error message "Undefined line number xxxxx in yyyyy" is printed. Incorrect reference (xxxxx) not changed by RENUM, but yyyyy may be changed. RENUM cannot be used to change order of program lines or to create line numbers greater than 65529, else "Illegal function call"error. C,D.

RESET Command. RESET
Closes all files on all drives. D.

RESTORE Statement. RESTORE [<line number>]
Allows DATA statements to be reread from a specified line. After RESTORE statement executed, next READ statement accesses first item in first DATA statement in the program. If <line number> specified, next READ statement accesses first item in specified DATA statement. C,D

RESUME Statement. RESUME [0]
RESUME NEXT
RESUME <line number>

Continues program execution after error recovery procedure has been performed. Any format can be used, depending on where execution is to be resumed. RESUME [0] resumes at statement that caused error. RESUME NEXT resumes at statement following error. RESUME <line number> resumes at specified line. RESUME statement that is not in an error handling routine causes a "RESUME without error" message to be printed. C,D

RETURN Statement. See GOSUB....RETURN Statements.

RIGHT$ Function. RIGHT$(X$,I)
Returns rightmost I characters of string X$. If I equal to number of characters in X$ (LEN(X$)), returns X$. IF I = 0, the null string (length zero) is returned. C,D

RND Function. RND [(X)]
Returns a random number between 0 and 1. The same sequence of random numbers is run each time the program is run unless the random number generator is reseeded (See RANDOMIZE statement.)

RUN Statement/Command. RUN [<line number>]
RUN filespec [,R]

<line number> is line number where execution is to begin. Otherwise, execution begins at lowest line number. <filespec> loads a file from disk into memory and runs it. <filespec> must include filename used when file was saved. Except with R option which keeps files open, RUN closes all open files and deletes current contents of memory before loading designated program. C,D.

SAVE Command. SAVE <filespec> [{,A or,P}]
Saves a program file on disk or cassette. <filespec> is string expression conforming to general rules for file names. "A"option saves file in ASCII format. P is protected by saving in encoded binary format - when later LOADed or RUN cannot be LISTed or edited. C,D.

SCREEN Function. X = SCREEN (<row>,<column>[,z])
Reads a character or colour at specified screen location, returns ASCII code. <row> is numeric expression 1 - 25. <column> is numeric expression 1-40 or 80 depending on screen width. x is valid numeric expression returning Boolean result (i.e. 0 is false, 1+..true.) If optional parameter z is given and is true, color returned instead.

SCREEN Statement.
SCREEN [<mode>][,[<burst>][,[<apage>][,vpage>]]]
Sets specifications for display screen. <mode> is numeric expression value 0,1,2. 0 = text mode, 1 medium resolution graphics, 2 High resolution. <burst> is numeric expression retuning Boolean result (0 is false,1+...true.)

Works differently according to mode: - in text mode, false gives black and white, true gives colour; MR mode, opposite.

<apage> integer expression, width 40, 0-7, width 80, 0-3. Selects page to be written to, text mode only.<vpage> same for visual page, can be different from active page. Default is <apage>. For further explanation and examples, see p.6 - 24.

On execution, screen cleared, black and white display, new mode stored. In text mode,<apage> and <vpage> permit alternation of pages on the screen, without loss. Values outside range will result in"Illegal function call"error. C,D.

SGN Function. SGN(X)
Indicates value of X, relative to zero: if X>0, returns 1, X=0, returns 0, X<0, returns -1. C,D.

SIN Function. SIN(X)
Returns sine of X, where X in radians. C,D.

SOUND Statement. SOUND <freq>,<duration>
Generates sound through the speaker. <freq> is desired frequency in hertz, numeric expression in range 37 to 32767. <duration> is duration in clock ticks (tick = 18.2 per sec.); numeric expression in range 0 to 65535. If duration is zero, any current SOUND statement running will be turned off. If no SOUND statement currently running, SOUND statement with duration of zero will have no effect. C,D.

SPACE$ Function. SPACE$(X)
Returns a string of spaces of length X. X must be integer in range 0 to 255. C,D

SPC Function. SPC(I)
Used with PRINT , LPRINT statements, skips I spaces, in range 0 to 255. ";" assumed to follow SPC(I) command. C,D

SQR Function. SQR(X)
Returns square root of X, where X<=0.C,D

STICK Function. x = STICK(n)
Returns x and y coordinates of two joysticks. n is numeric expression returning integer in range 0 to 3: 0, x coordinate for joystick A; 1, y co-ordinate; 2, x coordinate of joystick B; 3, y coordinate. C,D

STOP Statement. STOP
Terminates program execution and returns to command level. STOP statements may be used anywhere in a program to terminate execution. When encountered, "Break in line nnnnn" printed. Basic returns to command level after STOP. Execution resumed by issuing CONT command. C,D.

STR$ Function. STR$(X)
Returns a string representation of the value of X. C,D

STRIG Statement/Function.

```
STRIG ON
STRIG OFF
STRIG STOP
x = STRIG(n)
```

STRIG ON enables event trapping of joystick function, OFF disables. STOP disables, but event remembered , trapped as soon as trapping enabled. Function returns status of specified joystick trigger. For further remarks see ON STRIG statement.

In function, values returned for n can be 0, returns -1 if trigger A was pressed since last STRIG(0) statement; returns 0 if not. 1, returns -1 if trigger A is currently down, 0 if not.2,returns -1 if trigger B was passed since last STRIG(2) statement, 0 if not. 3, returns -1 if trigger B is currently down,0 if not.

When joystick event trap occurs, that occurrence of event is destroyed. Therefore, the x=STRIG(n) function will always return false inside a subroutine, unless event has been repeated since the trap. So, if you wish to perform different procedures for various joysticks, must set up different routine for each joystick, rather than including all procedures in single subroutine. C,D.

STRING$ Function. STRING$(I,J)
STRING$(I,X$)

Returns a string of length I whose characters all have ASCII code J or first character of X$. C,D

SWAP SWAP <variable>,<variable>
Exchanges values of two variables. Any type variable may be swapped, but two variables must be of same type or "Type Mismatch" error results. C,D

SYSTEM Command. SYSTEM
Closes all open files and returns control to operating system. When SYSTEM command is executed, a "warm start" is performed, DOS reloaded. D.

TAB Function. TAB(I)
Moves print position to I. If current print position already beyond space I, TAB goes to that position on the next line. Space 1 is leftmost position, and the rightmost position is the width minus one. I must be in range 1 - 255. TAB may only be used in PRINT and LPRINT statements. C,D

TAN Function. TAN(X)
Returns tangent of X. X should be given in radians. C,D.

TIME$ Function. TIME$
Retrieves current time. TIME$ function returns 8-character string, hh,mm,ss (hours, minutes seconds: 24hr.clock.) D

TRON/TROFF Statements/ Commands. TRON
TROFF

Trace execution of program statements. As debugging aid, TRON (in direct or indirect mode) enables a trace flag that prints each line number of program as it is executed. Numbers appear enclosed in square brackets. Disabled by TROFF. C,D

USR Function. USR [<digit>][(<argument>)]
Calls assembly language subroutine. <digit> specifies which USR routine is being called, set by DEF USR statement, q.v. If <digit> omitted, USR0 assumed. <argument> is value passed to subroutine. May be any numeric or string expression.

If segment other than default segment (data segment DS) is to be used, a DEF SEG statement must be executed prior to a USR

function call. Address given in DEF SEG statement determines segment address of subroutine. For each USR function, corresponding DEF USR statement must be executed to define USR call offset. This offset, and currently active DEF SEG segment address determine starting address of subroutine. Type of variable receiving function call must be consistent with argument passed. C,D.

VAL Function. VAL(X$)
Returns numerical value of string X$. Strips leading blanks, tabs, and linefeeds from argument string. X$ is string expression. C,D.

VARPTR Function,(1). VARPTR(<variable name>)
(2). VARPTR(#<file number>)

(1) Returns address of first byte of data identified with <variable name>. Value must be assigned to <variable name> prior to execution of VARPTR, otherwise, "Illegal function call" error. Any type variable name may be used. For string variables, address of addresss of first byte of string descriptor is returned. Address returned will be integer in range 0 to 65535. All simple variables should be assigned before calling VARPTR for an array, because addresses of arrays change whenever a new simple variable is assigned. C,D

(2) For sequential files, returns starting address of the disk I/O buffer assigned to <file number>. For random files, returns the address of the FIELD buffer assigned to <file number>. D

VARPTR$ VARPTR$(<variable name>)
Returns character form of memory address of variable. Primarily used with DRAW and PLAY in programs that will be compiled. Value must be assigned to <variable name> prior to execution of VARPTR$, or "Illegal function call" occurs. Any type variable can be used. Returns 3-byte string in form: byte 0 = type, byte 1 = low byte of address, byte 3 = high byte of address. Because array addresses change whenever a new variable is assigned, always assign all simple variables before calling VARPTR$ for an array element. D

WAIT Statement. WAIT <port number>,I,[,J]
Suspends program execution while monitoring status of machine input port. I and J are integer expressions, <port number> is specified machine input port. WAIT statement suspends execution until specified port develops specified bit pattern. Data read at the port is exclusive ORed with J and then ANDed with I. If result is zero, Basic loops back and reads data at the port again. If result is nonzero, execution continues with next statement. If J omitted, assumed to be 0. Possible to enter infinite loop with WAIT statement, in which case machine will have to be reset. To avoid, set WAIT with specified value at <port number> during some point in the program execution. C,D

WHILE....WEND Statements. WHILE<expression>

.
.
.

[<loop statements>]

.
.
.

WEND

Executes a series of statements in a loop as long as a given expression is true. If expression is not zero (i.e. true) <loop statements> are executed until the WEND statement is encountered. Basic then returns to <expression> and checks, if still true, repeats; if not true, resumes with statement following WEND statement. WHILE...WEND statements can be nested to any level. Each WEND will match the most recent WHILE. Unmatched WHILE statement causes a "WHILE without WEND" error, and v.v. C,D

WIDTH Statement (1). WIDTH [LPRINT]<size>
(2). WIDTH <file number>,<size>
(3). WIDTH <device>,<size>

Sets printed line width in number of characters for screen or line printer. <size> is numeric expression in range 0 to 255, specifies width of printed line. If 255, line width is "infinite", i.e.no carriage return is inserted. POSition of cursor or print head returns to zero after 255. <file number> is numeric expression in range 1-15, number of file that is open. <device> is string expression indicating device to be used.

(1) If tne LPRINT option is omitted, line width is set at screen (40 or 80). If LPRINT included, line width set at line printer. (2) If file is open to line printer, width immediately changed to specified size, file remains open. (3) Width assignment stored, but current setting not changed. Subsequent OPEN <device> FOR OUTPUT AS n will use specified value for width while file is open.

WRITE Statement. WRITE [<list of expressions>]
Outputs data to screen. If <list of expressions> omitted, blank line output. Included, values of expressions output to screen. May be numeric and/or string expressions, separated by commas. When output, each item separated by comma, strings delimited by quotation marks. After last item in list printed, Basic inserts carriage return, linefeed. WRITE outputs numeric values using same format as PRINT statement. C,D

WRITE# Statement. WRITE# <file number>,<list of expressions>
Writes data to a sequential file. <file number> is number under which file was OPENed. <list of expressions> string or numeric expressions separated by commas. WRITE inserts commas between items as they are written to file, and delimits strings with quotation marks. Carriage return/linefeed sequence is inserted after last item in list is written to file. C,D.

6.22 Basic Error Messages

We all make mistakes. Basic provides a complete set of error messages to tell you what's gone wrong and where. This list gives the error message, the error number (for use with ON ERROR and ERR) and some helpful suggestions for each possible error.

NEXT without FOR (1)

A NEXT statement doesn't have a corresponding FOR statement. Tne usual cause is tnat you have given a different variable name in the NEXT tnan in the last FOR statement. Remember that FOR-NEXT loops can be nested but they can't cross over.

Syntax error (2)

Tnere's something Basic can't understand. A command used in the wrong way or some characters typed in the wrong order. Usually, you'll find Syntax errors are typos - missing brackets or commas for example. Or it might be that the type of data given in a DATA statement doesn't match the READ statement that is reading it.

RETURN without GOSUB (3)

You can't RETURN from a subroutine if you've not done a GOSUB to get there. The usual cause for this is letting the main code of a program `fall' into a subroutine rather than ending it with an END statement.

Out of DATA (4)

You've asked the program to READ more data items than you've given in DATA lines. It's usually a matter of counting the number of elements on the DATA lines. You may find that a missing comma has caused two items to be treated as one.

Illegal function call (5)

The parameter given to a function is out of range. This error covers a variety of sins - from trying to delete non-existent lines to asking for a square root of a negative number. Look at the functions on the line and see if you are providing them with reasonable values. Try each one out as an immediate command to see where the problem is.

Overflow (6)

A number is too big for Basic to handle. This applies if an integer value exceeds 32767.

Out of memory (7)

Your program is too large or you are using too many nested FOR loops and subroutines. This error also applies to too many variables, too complex expressions or complex PAINT commands. Remember Basic needs space to think - don't fill the machine up.

Undefined line number (8)

You've tried to GOTO or GOSUB to a line which doesn't exist.

Subscript out of range (9)

You've tried to use an array with a subscript that was too large or had the wrong number of dimensions. Check the value used in the line against the values given in the DIM statement used to set up the array.

Duplicate definition (10)

You've tried to define the same array twice. Either you've used DIM a second time or tried to DIM the array after you've already used it as a default list of eleven elements. You'll also get the error if you use OPTION BASE after your DIM statements.

Division by zero (11)

You can't divide by zero and neither can your Advance. However, after issuing this message, your Advance will continue with the program, using its own `machine infinity' as a result of the calculation. If this could prove dangerous, use ON ERROR to trap for error number 11.

Illegal direct (12)

The command you entered can only be used as part of a program. It is not acceptable in immediate mode.

Type mismatch (13)

You've mixed up variables of different types. Normally this doesn't matter but it will cause an error if you try to put a string value in a numeric variable or vice-versa.

Out of string space (14)

There's not enough memory to store all the strings you are currently using. You should find it hard to generate this message accidentally but if you do, make sure that you don't have unused strings hanging around in memory. You can re-use variables throughout a program - always try to minimise the number of variables in use.

String too long (15)

Strings can have up to 255 characters in them. If you need more, figure out a way of working with a number of separate shorter strings.

String formula too complex (16)

An string expression is too long or too complex to evaluate. Try breaking the calculation down into a series of smaller steps.

Can't continue (17)

The CONT command cannot continue a program that either stopped due to an error, has been changed after being stopped or simply doesn't exist. You may be able to restart the program using GOTO as an immediate command. RUN isn't very effective in this situation as it clears all current variables before starting execution.

Undefined user function (18)

You must define a function with DEF FN before you use FN to call it.

No RESUME (19)

Your program has ended during an ON ERROR trapping routine. You should end the error handling routine with RESUME.

RESUME without error (20)

RESUME has been executed without any error being trapped. Usually, you'll find that this is because your program accidently `falls' into the error trapping routine from the main program.

Missing operand (22)

You've missed an operand out of an expression - for example used + or * without any value following it.

Line buffer overflow (23)

You've tried to enter a line that is too long. If it's a program line, separate it into a number of shorter lines. If it's got long strings in it, make sure that they are stored in a variable rather than written out as constants.

Device timeout (24)

Basic has failed to receive a response from some device it is communicating with. With your 86a, this could be a problem with the cassette recorder or the printer. It is not a problem with your program. Try and fix whatever is wrong and retry the operation.

Device fault (25)

A device communicating with Basic has developed a fault. On your 86a, this could only be a printer fault. Check that the printer is on-line (`selected'), has paper and is connected correctly.

FOR without NEXT (26)

A program has ended with a FOR loop left unfinished. You should add a suitable NEXT statement or remove the FOR if it is not needed.

Out of Paper (27)

The printer has run out of paper. Reload it and check it is back `on-line' then continue with the program.

WHILE without WEND (29)

A program has ended with a WHILE still active. You should either remove the WHILE if you don't need it, or add a suitable WEND statement.

WEND without WHILE (30)

A WEND was executed before a corresponding WHILE. Make sure that each WHILE and WEND forms a pair in the correct order.

FIELD overflow (50)

A FIELD statement is demanding more bytes for each random record in a file than you've set aside when you opened the file. Make sure that your OPEN and FIELD statements don't have different ideas about how long the records are.

Internal error (51)

You should never see this message. It indicates an internal fault within Advance Basic. Check that your machine is running properly and if the fault continues to occur, contact your supplier.

Bad file number (52)

A file number (or file or device name) is invalid for some reason the name is too long and so on.

File not found (53)

A command has refered to a file that doesn't exist. The usual cause of this is a mis-spelt file name or failure to specify the right drive that tne file is on.

Bad file mode (54)

You're using commands that don't relate to the type of files involved, such as: GET and PUT with a sequential file or MERGE with a tokenised file and so on. Check that you have the file OPENed as you intended.

File already open (55)

You can't OPEN a file twice. You'll also get this error if you try to KILL a file while it is open - you must close it first.

Device I/O Error (57)

An input/output device, such as an RS232 port, has detected a fault in data transmission. This error is fatal to Basic.

File already exists (58)

You've used NAME to try to rename a file to a name that already exists. Pick a different name.

Disk Full (61)

There's no space left on the disk in use. This error closes all the files on the disk so you can often substitute another disk and try the operation again. If this occurs when you are running an application program, you may need to use a file delete option to erase some unwanted files before trying again. Try and avoid the situation by keeping a close eye on disk free space before you start running programs.

Input past end (62)

You've tried to read past the end of a file (or you've got the file open for output or append!). To trap this, your reading routines should use the EOF function.

Bad record number (63)

GET and PUT allow random record numbers from 0 to 32767 only.

Bad file name (64)

A file name is invalid for some reason. Check the section on naming files (page 5-11) for further information.

Direct statement in file (66)

An ASCII program file contains lines that are not valid Basic lines (that is, they don't start with a line number). If the file contains a handful of these duff lines, you should be able to remove them using a text editor such as the EDLIN program supplied with your Advance.

Too many files (67)

There are too many files on the disk in use to create another one - either KILL some files or switch disks.

Device unavailable (68)

You've tried to use a device that either doesn't exist or is currently disabled. Check that the device is correctly installed and that you've done nothing to disable it.

Communications buffer overflow (69)

You've tried to read more data from a communications port while the data buffer is still full. This error should be trapped with ON ERROR to prevent data being lost. If you can't process data faster than you are receiving it, you need to run the communications at a slower rate or implement a more sophisticated handshaking system.

Disk Write Protect (70)

The disk you are trying to write to is write-protected. To continue, remove the write-protect tab from the disk and try again. But think first there may be a good reason why the disk is protected.

Disk not ready (71)

The Advance can't read the disk at all - probably because you've got no disk in the drive, have put one in sideways or left the door open. Check the drive and try again!

Disk media error (72)

A fault has developed with the disk in use. This might be a hardware fault but is more likely to mean that the disk in question has become damaged or worn. You should try and copy any valuable files off the disk immediately. If your files are damaged, there are commercial packages available that can read from corrupt disks.

It is often possible to reformat a duff disk but if you continue to get errors, it's probably best to throw it away. You do have backup copies don't you?

Unprintable error (invalid number)

An error has occured for which there is no error message. You can get this by using an undefined error number with the ERROR statement.

CHAPTER SEVEN OPTIONS and POSSIBILITIES.

		Page
7.1	The Printer	7-1
7.2	Plotters	7-2
7.3	Monitors	7-2
7.4	Communications Adaptors	7-2
7.5	Additional Memory	7-2
7.6	Hard Disks	7-3
7.7	Games Control	7-3
7.8	Expansion Boards	7-3
7.9	Additional Software	7-3

OPTIONS AND POSSIBILITIES

The Chapters on DOS and BASIC have outlined the fundamentals of communication with the Advance, and shown its specific features. In this Chapter, for the beginner, we briefly outline some of the possibilities for use of the Computer, its applications, and its potentials for extension.

The major usage on computers of this power is of Applications Programs, packages of software designed around particular tasks or areas of jobs. The best known and most obvious of these are Word Processors,Spread Sheets, Financial Packages, and Business Analysis, but these are only a sample of what is available.Your Advance comes with a library of software, including a powerful word processing package and a sophisticated Spread Sheet. Precise instructions in using these packages are included with them in the shape of Manuals which you will now find approachable and easy to work with. Remember to follow immediately the advice in these manuals (and elsewhere) and back up the discs! (See both the DOS Ch.5.8., and the instructions in the Manuals, if you are unsure how to do this.) These extremely effective programs and applications will run with no knowledge of programming, by merely following the simple procedures outlined.

When you have begun to use these, your enthusiasm for new uses will rapidly grow, and you will start thinking about still further applications and possibilities. In this section, we will run through a few of the most obvious extensions.

7.1 The Printer.

It may very well be that you have already acquired a Printer. Your Word Processing package will not be particularly useful without one, and after the main system components, including upgrading to disc drives and the 868, a printer would be most people's first choice. Prices on printers now put them well within the reach of modest budgets.

They work in two main ways: dot matrix and daisy wheel (each of which vary immensely in technical specification.) It is possible to connect a good quality electric typewriter, normally via a SERIAL INTERFACE, but output would be very slow (if of good quality.) A serial interface card transmits data through a limited numer of channels (probably two) in a stream, ie with the data ordered sequentially. This is contrasted with a PARALLEL INTERFACE, which transmits data in chunks of 8-bits at a time. A printer may work either way. The Advance has a parallel port on the back of the A unit, and a serial port on the B.

Daisy wheel printers produce very high quality print, often described as letter-quality. The daisy wheels exchange, like similar devices or golf-balls on a typewriter, to produce different typefaces. Dot matrix printers are normally regarded as producing less quality, but it is more than adequate for all

normal purposes, and they tend to be rather faster and rather cheaper. They also will produce varying fonts, like italic or 'heavier' print described as "emphasised", or bold, normally by software programs that can be simply selected. The whole area is one of rapid technical advance, and in selecting your printer you should attend to price, the quality you really require, and tne speed you desire. For a relatively small outlay, you can have a printer that will produce a thoroughly acceptable product very swiftly, and which will photocopy and reduce to produce manuals, books etc with a minimum of trouble. Any standard printer will connect easily to the Advance.

7.2 PLOTTERS

A plotter is an automated draughtsman's workboard powered by your computer. The plotter uses pens with coloured inks under program control. This means, in effect, that you can have hard copy of what you can produce electronically on the screen. The Advance's built in ports for serial and parallel transmission mean that all common plotters can be plugged straight in, without any special adapter.

7.3 MONITORS

The Advance will output to an ordinary black-and-white or colour T.V. These devices are not, however, designed to give the best resolution or highest colour quality. If you want these, you should investigate the purchase of a video monitor. The two most common types are composite monitor (or comp.sync.) and R.G.B. (Red,Green,Blue.) A composite monitor sums the inputs, and drives the video guns with that summed signal. An R.G.B. monitor supplies the signals separately to the guns, producing the highest of quality. Again, you need to calculate your precise preferences and needs against price. Either kind of monitor will interface immediately to your Advance, via the connections described on Ps. 4 & 5.

7.4 COMMUNICATIONS ADAPTERS

A communications adapter allows you to connect your Advance to other computers. With a modem, electronic or via an acoustic coupler, the connection can be via a 'phone line. By these means you could send or receive electronic mail; exchange information with other computer users; connection to Prestel and similar systems; use of networks with other microcomputer systems.

7.5 ADDITIONAL MEMORY

The 868 has 128k of memory board. For anyone who finds this a limitation, your supplier will fit a furtner 128k on to the system board. Memory up to 640k can be installed in the bus structure.

7.6 HARD DISKS

The Disk units supplied with the 86 use floppy disks. A hard disk is a larger, non-flexible disk housed permanently in a drive unit. Hard disks are fast, reliable, and of high capacity - forty times more than a floppy. They are suitable for large-memory application, with the standard disk drives as backups and for transfer.

7.7 GAMES CONTROL

Two analog-type joysticks will plug directly into the adapter built into the Advance. These are enjoyable tools for arcade style games.

7.8 EXPANSION BOARDS

The Advance 86 BUS structure is hardware compatible with that of the IBM PC, and most cards designed for that and similar systems can be installed without problem. See the Advance Technical Reference Guide for more information.

7.9 ADDITIONAL SOFTWARE

This represents (obviously) endless possibilities.Nonetheless, a great deal of care is required to make the most advantageous acquisitions. Utilise your Advance supplier. S/he will have the best information on what will be the best for the potential of your machine. Obviously, with I.B.M.compatibility, the range of software available will be unmatched. Even so, check carefully that it will run on your system with its present configuration. Where possible, see it demonstrated on an Advance. Make sure you have a specific guarantee of replacement in case of unforeseen trouble. Check out an evaluation copy; test it to its limits. Try unusual options, enter impossible data. Certainly, if you have a specific application in mind, try it out at the Dealer's.

DISK ERRORS

If a disk or device error occurs at any time during a or program, DOS returns an error message in the following format:

```
<yyy> ERROR WHILE <I/O action> ON DRIVE x
Abort,Ignore,Retry:_
```

In this message,<yyy> may be one of the following:

```
WRITE PROTECT
BAD UNIT
NOT READY
BAD COMMAND
DATA
BAD CALL FORMAT
SEEK
NON-DOS DISK
SECTOR NOT FOUND
NO PAPER
WRITE FAULT
READ FAULT
DISK
```

The <I/O-action> may be either of the following:

```
READING
WRITING
```

The drive <x> indicates the drive in which the error has occurred.

DOS waits for you to enter one of the following responses:

A Abort. Terminate the program requesting the disk read or write.

I Ignore. Ignore the bad sector and pretend the error did not occur.

R Retry. Repeat the operation. This response is to be used when the operator has corrected the error (such as with NOT READY or WRITE PROTECT errors).

Usually, you will want to attempt recovery by entering responses in this order:

R (to try again)
A (to terminate program and try a new disk)

One other error message might be related to faulty disk read or write:

FILE ALLOCATION TABLE BAD FOR DRIVE x

This message means that the copy in memory of one of the allocation tables has pointers to nonexistent blocks. Possibly the disk was incorrectly formatted or not formatted before use. If this error persists, the disk is currently unusable and must be formatted prior to use.

HOW TO CONFIGURE YOUR SYSTEM

In many cases, there are installation-specific settings for DOS that need to be configured at system startup.

The DOS configuration file (CONFIG.SYS) allows you to configure your system with a minimum of effort. With this file, you can add device drivers to your system at startup. The configuration file is simply an ASCII file that has certain commands for DOS startup (boot). The boot process is as follows:

1. The disk boot sector is read. This contains enough code to read DOS code and the installation's BIOS (machine-dependent code).

2. The DOS code and BIOS are read.

3. A variety of BIOS initializations are done.

4. A system initialization routine reads the configuration file (CONFIG.SYS), if it exists, to perform device installation and other user options. Its final task is to execute the command interpreter, which finishes the MS-DOS boot process.

CHANGING THE CONFIG.SYS FILE

If there is not a CONFIG.SYS file on the DOS disk, you can use the DOS editor, EDLIN, to create a file; then save it on the DOS disk in your root directory.

The following is a list of commands for the configuration file CONFIG.SYS:

BUFFERS = <number>
This is the number of sector buffers that will comprise the system list. It is installation-dependent. If not set, 10 is a reasonable number.

FILES = <number>
This is the number of open files that the XENIX system calls can access. It is installation-dependent. If not set, 10 is a reasonable number.

DEVICE = <filename>
This installs the device driver in <filename> into the system list. (See below.)

BREAK = <ON or OFF>
If ON is specified (the default is OFF), a check for CONTROL-C as input will be made every time the system is called. ON improves the ability to abort programs over previous versions of the MS-DOS.

SHELL = <filename>
This begins execution of the shell (top-level command processor) from <filename>.

A typical configuration file might look like this:

```
Buffers = 10
Files = 10
Device = \BIN\NETWORK.SYS
Break = ON
Shell = A:\BIN\COMMAND.COM A:\BIN /P
```

Note here that the Buffers and Files parameters are set to 10. The system initialization routine will search for the filename \BIN\NETWORK.SYS to find the device that is being added to the system. This file is usually supplied on disk with your device. Make sure that you save the device file in the pathname that you specify in the Device parameter.

This configuration file also sets the DOS command EXEC to the COMMAND.COM file located on disk A: in the \BIN directory. The A:\BIN tells COMMAND.COM where to look for itself when it needs to re-read from disk. The /P tells COMMAND.COM that it is the first program running on the system so that it can process the DOS EXIT command.

Advance at a glance

CPU Type:	16-bit 8086 running at a clock speed of 4.77 Mhz.
RAM Memory:	128K bytes main memory with parity checking. Additional 128K bytes can be fitted to system board. Additional expansion to 640K bytes via bus structure.
	Separate 16K bytes of video memory.
ROM Memory:	64K bytes.
ROM Contents:	Self-test, diagnostic routines. Advance Cassette Basic interpreter. ROM BIOS.
Keyboard:	Separate detached keyboard. Fully programmable. Typematic, with auto-repeat. 84 Keys. 10 Programmable function keys. Numeric/cursor control pad.
Character set:	256 characters. Software definable in graphics modes.
Display:	UHF Television via integral modulator. RGB monitor. Composite video monitor, colour or black and white.
Text display:	80 x 25 or 40 x 25. 8 video pages with 40 x 25 text. 4 video pages with 80 x 25 text.
Graphics display:	640 x 200, bit mapped display with 2 colours. 320 x 200, bit mapped display with 4 colours.
	Additional modes possible with custom programming.
Sound:	Built-in loudspeaker under software control.
Cassette interface:	Audio cassette interface, with motor control.
Diskette interface:	Twin 5 1/4" minifloppy drives. 360K capacity on each. Read/write/format wide variety of popular, soft sectored, disk formats.
DOS:	Advance 86 Personal Computer DOS 2.11.
Parallel printer:	Centronics standard parallel printer interface.
Serial interface:	RS232 serial interface.

Game ports:	Provision for two analogue joysticks, plus four trigger inputs.
Expansion:	Provision for 8087 floating point co-processor. 3 expansion slots to IBM PC bus structure. 2 16-bit expansion slots.

0	<NUL>	32		64	@	96	`	128	Ç	160	á	192	└	224	α
1	☺	33	!	65	A	97	a	129	ü	161	í	193	┴	225	ß
2	☻	34	"	66	B	98	b	130	é	162	ó	194	┬	226	Γ
3	♥	35	#	67	C	99	c	131	â	163	ú	195	├	227	π
4	♦	36	$	68	D	100	d	132	ä	164	ñ	196	─	228	Σ
5	♣	37	%	69	E	101	e	133	à	165	Ñ	197	┼	229	σ
6	♠	38	&	70	F	102	f	134	å	166	ª	198	╞	230	µ
7	<BEL>	39	'	71	G	103	g	135	ç	167	º	199	╟	231	τ
8	◘	40	(	72	H	104	h	136	ê	168	¿	200	╚	232	Φ
9	<TAB>	41	)	73	I	105	i	137	ë	169	⌐	201	╔	233	Θ
10	<LF>	42	*	74	J	106	j	138	è	170	¬	202	╩	234	Ω
11	<HOM>	43	+	75	K	107	k	139	ï	171	½	203	╦	235	δ
12	<FF>	44	,	76	L	108	l	140	î	172	¼	204	╠	236	∞
13	<CR>	45	-	77	M	109	m	141	ì	173	¡	205	═	237	ø
14	♫	46	.	78	N	110	n	142	Ä	174	«	206	╬	238	ε
15	☼	47	/	79	O	111	o	143	Å	175	»	207	╧	239	∩
16	►	48	0	80	P	112	p	144	É	176	░	208	╨	240	≡
17	◄	49	1	81	Q	113	q	145	æ	177	▒	209	╤	241	±
18	↕	50	2	82	R	114	r	146	Æ	178	▓	210	╥	242	≥
19	‼	51	3	83	S	115	s	147	ô	179	│	211	╙	243	≤
20	¶	52	4	84	T	116	t	148	ö	180	┤	212	╘	244	⌠
21	§	53	5	85	U	117	u	149	ò	181	╡	213	╒	245	⌡
22	▬	54	6	86	V	118	v	150	û	182	╢	214	╓	246	÷
23	↨	55	7	87	W	119	w	151	ù	183	╖	215	╫	247	≈
24	↑	56	8	88	X	120	x	152	ÿ	184	╕	216	╪	248	°
25	↓	57	9	89	Y	121	y	153	Ö	185	╣	217	┘	249	∙
26	→	58	:	90	Z	122	z	154	Ü	186	║	218	┌	250	·
27	←	59	;	91	[	123	{	155	¢	187	╗	219	▉	251	√
28	<CUR>	60	<	92	\	124	|	156	£	188	╝	220	▄	252	n
29	<CUL>	61	=	93	]	125	}	157	¥	189	╜	221	▌	253	²
30	<CUU>	62	>	94	^	126	~	158	₧	190	╛	222	▐	254	■
31	<CUD>	63	?	95	_	127		159	ƒ	191	┐	223	▀	255	<BLK>

Anschlußbelegung des Genie 16:

Parallelport:

1	Strobe**
2	DATA 0
.	.
.	.
9	DATA 7
10	Acknowledge
11	Busy
12	Paper Out
13	Printer selected
14	Printer: CR nach LF
15	Printer ERROR**
16	Printer Initialize**
17	Select In**
18	0 V
25	0 V

** = negiertes Signal

Serialport:

2	TX Data
3	RX Data
4	RTS
5	Clear to Send
6	Data set Ready
7	0 V
8	Carrier detect
9	+ XM ITCL REL
11	- XM ITCL Data
18	+ Receive Clock Data
20	DTR
22	RI
25	- Receive Clock Return

SELF TESTS

When you first switch your computer on, it goes through a complete set of self-test routines, designed to check that everything is working O.K. If any of these tests fail the Advance will attempt to give an indication of what is wrong on the loudspeaker or video display.

Errors signalled by the loudspeaker.

Error	Loudspeaker notes	
	Long	Short
Processor failure	1	0
ROM checksum failure	1	2
RAM checksum failure (including video ram)	2	1
PIC Interrupt controller failure	0	2
Time failure	0	3
Video logic failure	2	2

Errors signalled on the video screen

Error	Message
KEY E	Keyboard error, keyboard reset did not complete
KEY STUCK	A key is stuck on the keyboard, or the keyboard interface circuitry is malfunctioning.
CAS E	Cassette interface circuitry is faulty.
DSK E	Disk interface circuitry is faulty.

PIN CONNECTIONS

Keyboard connector J1.

Pin	
1	Keyboard clock
2	Keyboard data
3	Reserved
4	Ov
5	+5v

Cassette interface J2

Pin	
1	Motor control output 1
2	Ov
3	Motor control output 2
4	Data in - from audio output of cassette recorder
5	Data out - to AUX input of cassette recorder

Parallel printer port J5

Pin	
1	Strobe signal (active low)
2-9	Printer data. Pin 2 is data bit 0, pin 9 is data bit 7
10	Acknowledge (active low)
11	Busy (active high)
12	Paper end (active high)
13	Printer selected (active high)
14	Print LF after CR (active low)
15	Printer error (active low)
16	Initialise printer (active low)
17	select in (active low)
18	Ov
25	Ov

Advance 86 Personal Computer User guide.

Reader Comment form.

Please use this form to report errors, or suggest changes or improvements that you feel would improve the quality of this documentation.

Thank you for your comments.

Please return this form to:

Advance Technology (UK) Ltd.
Attn: Technical Publications.
8A Hornsey Street.
London. N7 8HR.
U.K.