

BBC BASIC

Your Notebook contains a powerful BASIC interpreter that can be used to write your own programs. BASIC is the most popular programming language for beginners to learn. In fact, the name BASIC is short for Beginners All-purpose Symbolic Instruction Code because it is designed with beginners in mind. Symbolic Instruction Code is just a technical way of saying "programming language". The version of BASIC built into the Notebook is compatible with BBC BASIC - the version that is taught in most schools and that is used on many other computers.

When you are using the Notebook you can switch to using BASIC at any time by holding down the **[Function]** key and pressing **[B]**. To leave BASIC when you have finished you type the command ***QUIT**. Programs are not saved automatically so you MUST use the **SAVE** command before leaving BASIC to preserve any program you have been working on.

You may like to set the 'Preserve context during power off' switch in the System Settings menu to 'Yes' so that if you switch off while using BASIC your program is still available when you next switch on.

When you are using BASIC you must ensure that Caps Lock is switched on at all times because BASIC expects all its commands to be entered in upper case. If you type in a command and just see the message "Mistake" then it may well be that you have mistakenly used lower case. To help you, each time you switch to BASIC Caps Lock will be turned on (if it wasn't already). When you leave BASIC the setting of Caps Lock will be returned to its original state.

When BASIC is started the screen will clear and you will see the message:

```
BBC BASIC (NC200) Version 3.12
(C) Copyright R.T.Russell 1992
>
```

The ">" symbol is the BASIC "prompt" and it shows that BASIC is ready for you to type in a command. There are two ways in which BASIC can be used. You can just type individual commands at the prompt and the result of them will be shown immediately.

188

You might wonder why the lines have been numbered 10, 20, 30 rather than just 1, 2 and 3. The reason for this is that if you later choose to add a line between 10 and 20 you could pick a number such as 15 that would be stored between them. If the lines were numbered 1, 2, 3 there would be no room to add a line between 1 and 2. You cannot use 1.5 as a line number. You can only use whole numbers. Try typing in:

```
15 PRINT "The result of 3 added to 4 is ";
```

and then type **LIST** to see how that new line has been slotted in between 10 and 20. The semi-colon on the end of line 15 is a special command to BASIC that means that the next thing it prints should appear on the same line as the preceding text.

So far, all that you have done is to enter the lines of a program. To actually see what happens when the program runs you must give the immediate command **RUN**. You will see the program print the following:

```
Start
The result of 3 added to 4 is      7
End
```

When you have finished writing a program you can store it permanently by using the **SAVE** command. You should type the word **SAVE** followed by the name of a file in which you want to store your BASIC program. The file name must be given in quotation marks. To save your first program you could type:

```
SAVE "FIRST"
```

To clear the memory so that you can start entering a completely new program you use the command **NEW**. Just type:

```
NEW
```

You must be careful with this command. If you type **NEW** without first saving anything you want to keep you will lose everything that you have entered. If you type the command **LIST** you will see that your first program has disappeared. BBC BASIC has a special command, **OLD**, which you can use immediately after a **NEW** command to recover what has just been erased. You must use **OLD** straight away, as soon as you enter a new program line the old program is lost forever.

190

Immediate commands

Try typing:

```
PRINT 3 + 4
```

When you press **[Enter]** the result, 7, is displayed immediately. This is known as immediate mode. Now, try typing the following (you must end every command by typing **[Enter]**).

```
PRINT "Hello World"
```

(To type "Hello World" you must switch Caps Lock off temporarily by pressing the **[Caps Lock]** key, but remember to switch it back on before typing further BASIC commands)

PRINT is a command which does just what the name suggests and tells BASIC that when the line is executed it should print whatever follows the command on the line. If you just want some text printed you must put it in quotation marks. Anything that isn't in quotation marks BASIC assumes are further instructions. (As it did with 3 + 4, which if understood were instruction to tell it to add 3 and 4).

Writing programs - a short tutorial

Besides immediate mode the other way in which BASIC is used is for you to type in the lines of a program and these will be stored in the Notebook's memory. This is known as program mode. It is only when you give the special command, **RUN**, that the commands you have entered are actually acted upon. BASIC knows to store a command rather than act on it immediately if it starts with a number. Each line you type in must have a different number and the lines will be stored in number order. So, for example, if you were to type:

```
10 PRINT "Start"
20 PRINT "End"
30 PRINT 3 + 4
```

The lines would actually be stored in the order 10, 20, 30. You can see this by typing the immediate mode command **LIST**. This will show you the program that is being held in the Notebook's memory.

189

To check that the little program that you just saved can be recovered, type the command:

```
LOAD "FIRST"
```

This will load the program back into BASIC's program memory. You can now type **LIST** and you will see that the program has been recovered. However, we no longer want to keep a copy of that program in BASIC's memory so type the **NEW** command to clear BASIC's memory then a new program can be entered. Now that BASIC is ready for a new program to be entered type in the following small program. Remember that BASIC keywords (such as **PRINT**, **IF**, **THEN**, **GOTO**) must be in upper case. If you forget you will see the error "Mistake in line x" when you **RUN** the program.

```
10 NUM = 1
20 PRINT NUM
30 NUM = NUM + 1
40 IF NUM < 8 THEN GOTO 20
```

Type **RUN** and you should see that the program prints out the numbers 1 to 7.

Line 10 sets a variable called **NUM** to a starting value of 1. A **variable** is the name you give to an item that will store either a number or a piece of text. In this case the item is storing the number 1 for us and we have chosen to call it **NUM**.

Line 20 uses the **PRINT** command to show the contents of the variable **NUM** on the screen. Because **NUM** is not enclosed in quotation marks, BASIC knows that it must look up the value stored in a variable called **NUM** instead of just printing the word **NUM** on the screen.

Line 30 adds one onto the current value stored in **NUM**. In effect the line is saying "Set the variable called **NUM** to be equal to the value currently store in **NUM** with one added to it".

Line 40 checks to see if **NUM** is less than 8 (the left angle bracket is a special symbol used by BASIC to mean "less than"). If **NUM** still has a value less than 8 then BASIC goes on to execute the command **GOTO 20** which means, go back to line 20 and carry on running the program from there. If **NUM** is 8 or more then the part of the line after **THEN** is ignored and it goes on to execute the next line in

191

sequence. Because there are no more lines after 40, the program stops running.

This simple 4 line program does not do very much but it has shown many of the concepts involved in programming. There is the use of variables to store information (NUM), there is the printing of results, there is the manipulation of variables to change the value that is held and there is the redirection of program flow depending on whether a certain condition is met. It is this last feature that sets a computer apart from a simple calculator. Calculators (however complex) can only perform arithmetic. It is only once the flow of calculation can be changed as the result of a previous operation that a calculating device can be considered a computer.

Anyone familiar with BASIC may have already realised that there is a much neater way of achieving exactly the same effect as that program we have just entered. Type NEW to start a new program and then enter the following 3 lines:

```
10 FOR num = 1 TO 7
20 PRINT num
30 NEXT num
```

To save you having to type in the numbers 10, 20, 30 you could type the immediate mode command **AUTO** and BASIC will generate the number in sequence for you starting at 10 and going up in steps of 10. When it shows "40" just press the **ESC** key to get back to the BASIC prompt ">".

Notice that this time we have put NUM in lower case "num". This just makes the program easier to read. We could just as easily have used NUM but the important thing to know is that BASIC treats variable names as case sensitive which means that it treats NUM and num (and Num and nUm and nuM and so on) as different variables. You must always make sure that variable names match correctly. It may be easiest to always use lower case for variable names and upper case for BASIC keywords. This makes your programs easier to read.

The words **FOR** and **NEXT** are 2 commands in BASIC that are always used together. The **FOR** command starts a variable at a certain value and sets an upper limit for it. Then every command in between the **FOR** and the **NEXT** command is executed and one is added to the variable. If it has not reached the limit a jump is made back to the instruction after the **FOR** command. The command

192

had VAT added to it at the given rate. Like before, lines 10 and 20 use the **INPUT** command to get the user of the program to input some values. The wording of the question is enclosed in quotation marks and this is followed by a semicolon (;) and then the name of the variable in which the value should be stored. If you don't use a semicolon then no question mark is printed.

Line 30 creates a new variable called total which is the result of adding the amount multiplied by the VAT rate to the original amount. The part of the calculation in parentheses is calculated first before the final addition is performed. The asterisk (*) is the symbol BASIC understands to mean "multiply" and the slash (/) is the symbol that means "divide by".

Line 40 prints the original amount stored in the variable amount, followed by the message (in quotation marks) followed by the value of the variable total.

This program has one or two shortcomings. Firstly, it was not really necessary to create the intermediate variable called total to hold the result of the calculation. Instead, line 30 could be deleted and line 40 changed to read

```
40 PRINT amount "With VAT added is " amount + (amount * vtrate / 100)
```

To delete line 30 just type the number 30 on its own and press **DEL**. This is the standard way to remove a single line from a program. If you run the program again it will work exactly as before even though it has been simplified.

The next problem that we could overcome is the fact that each time the program is RUN it just allows one set of numbers to be entered and then stops. What we could do with is having the program loop back to the start each time it gets to the end. This is easy to do. Just add line 50:

```
50 GOTO 10
```

When you type RUN it will ask you for the VAT rate and then the amount and then display the result. It will then go back round and ask for the VAT rate again. This isn't really what we wanted. It should only be necessary to enter the VAT rate once each time the program is run. What's more, there doesn't appear to be any way to stop the program running.

194

NEXT is followed by the same name of the variable as used in the **FOR** command so that any particular **NEXT** command knows which **FOR** command it should jump back to.

So far our little programs have had fixed numbers built into the program (the first program could only show the result of 3 + 4, the second and third would just print the numbers 1 to 7 on the screen). Normally you will want to make your programs more versatile so that each time they are run they ask for some information and then modify the operation of the program according to the information entered. You do this with the **INPUT** command.

Type LIST to see the current three line program and then type:

```
5 INPUT "Start";start
7 INPUT "Finish";finish
10 FOR num = start TO finish
```

Now type LIST and you will see that not only have two new lines been added to the start of the program but line 10 has been replaced by a new version. If you type in a new line with the same number as an existing line then that existing line will be replaced by the new version.

The sequence of numbers in our program is now 5, 7, 10, 20, 30. This is bit untidy. Type the command **RENUMBER** and then LIST. You will see that the program has been renumbered with the line numbers going up in steps of 10.

When you RUN the modified program it will stop and ask for a Start value. Type the number 10 and press **ENTER**. When it asks for "Finish" type 14. It will then print the numbers 10 to 14. If you run it again and enter different start and finish values it will print a different set of numbers each time.

The following is another example of a program that asks for you to input numbers when it is run and then processes the numbers to show a result.

```
10 INPUT "VAT rate as a percentage (0..100) "; vtrate
20 INPUT "Amount "; amount
30 total = amount + (amount * vtrate / 100)
40 PRINT amount " With VAT added is " total
```

When you RUN this program it will ask you to input the VAT Rate and then an amount. It will print out what that amount is when it has

193

To stop the program press the **ESC** key. If you press the **ESC** key when a program is running it will stop running and the BASIC prompt will appear after a message which says "Escape at line x". This tells you which line BASIC was executing when you stopped it.

The easy answer to not being asked to input the VAT rate every time is to make the destination of the final jump to be line 20. So type the command:

```
EDIT 50
```

The current contents of line 50 will be displayed. Use the **←** and **→** arrow keys to move the cursor to the end of the line and then delete 10 and replace it with 20. When you have finished editing the line press **ENTER**.

If you ever **EDIT** a line and then realise that you would like to keep the old version just press **ESC** and your changes will be ignored. You will find when editing lines that many of the quick methods you may have learnt about in the word processor can be used to move about the line. For example, the left and right arrow keys **←** and **→** pressed with **SPACE** will move a word at a time and when pressed with **ESC** will jump to the start or end of the line.

In our program there is still a problem that the only way to stop the program is by pressing the **ESC** key. As the program keeps waiting to ask for an amount to be input we could arrange for the program to stop running completely if the value 0 were input. Add the following line:


```
25 IF amount = 0 THEN STOP
```

Including this line means that if the value 0 is input when the program is asking for "amount" the program will STOP. **STOP** is a command to BASIC that does exactly what the name suggests and stops a program running returning the BASIC prompt.

The above has given you an idea of the first steps in learning BASIC but unfortunately there isn't room in this manual for a complete tutorial. What we suggest is that you get one of the many hundreds of books available on programming in BASIC and you will find that most of what they say applies equally well to the BBC BASIC in your Notebook. If possible, get a book that is specifically written with BBC BASIC in mind.

195

Example BASIC programs

The following are a few simple programs that give a small taste of what is possible in BASIC on the Notebook. Don't worry too much if you don't understand all the commands used. You will probably find them fun to use even if you do not understand exactly how they work. One thing to watch is that you must type them exactly as shown including all spaces and punctuation symbols. Don't forget to press  at the end of each line that you type.

PROGRAM 1: Reaction time tester

This program uses the 1/100th second clock that BASIC gives access to in order to time your reactions:

```
10 PRINT "Get ready..."
20 D = RND(300)+200
30 TIME = 0
40 REPEAT
50 UNTIL TIME > D
60 PRINT "Press a key..."
70 T = TIME
80 X = GET
90 T = TIME - T
100 PRINT "You took" T/100 " seconds."
```


Line 20 picks a number at random between 0 and 300, adds 200 to it to make it between 200 and 500 and then sets the clock to zero in line 30.

Lines 40 and 50 then delay for that number of 1/100ths of a second. (between 2 and 5 seconds)

Line 60 prints the message you must react to. The current time is remembered in variable T and then after the GET statement in line 80 has got a key press a calculation is made to see how much time has elapsed.


This value is printed in line 100 (divided by 100 because it is a measurement in 1/100ths not whole seconds so must be converted).

Type NEW to clear any existing program. Type in the lines above, then type RUN to use the program. You may like to use SAVE to save a copy for later use.

semitones not $1/4$ semitones. Finally it is added onto Cnote to make the  key, (note=1) note C. The duration of note is set to $4/20$ ths of a second ($1/5$ th) so each key press will just make a short beep.

The last line of the program just directs control back round to line 20 so it will always wait for keys to be pressed (until the space bar is used to stop the program).

Well that's the program but what do you do with it? Well, RUN it and press some keys on the middle line of the keyboard. You should hear musical notes. The keys have been picked to try and imitate the black and white notes on a piano keyboard:

keys 

C# Eb F# Ab Bb C# Eb F# Ab

notes C D E F G A B C D E F G

So try tapping in the following tunes (spacing gives an idea of rhythm):

Patriotic:






GG H T GH JJ K J HG HG T G
GHJK LLL L KJ KKK K JH
JKJHG J K L ; K J H GE

Seafaring:

KJK AA GFDG KK ;LK L SS LKJ L ##


Scottish and lyrical:

W TTY UU Y TTE WW
W TTY U O P O
O P P O U U O J U Y T E
W W T U O P O U Y T

While you are using the program press the    or  keys next to the  key and see what happens. When you have finished press the space bar to stop the program running.

PROGRAM 2 : A musical organ program

```
10 k$ = "AWSDFGTGYHJKLP;']#+CHRS(13) : Cnote=96
20 a$ = GET$
30 IF a$ = " " THEN STOP
40 IF a$>"1" AND a$<="4" THEN Cnote=48*(ASC(a$)-ASC("0")) : GOTO 20
45 IF a$>"a" AND a$<="z" THEN a$=CHRS(ASC(a$)-32)
50 note = INSTR(k$, a$)
60 IF note = 0 THEN GOTO 20
70 SOUND 1, 0, note*4 + Cnote, 4
80 GOTO 20
```


Line 10 defines a string called k\$ which holds all the keys that the program recognises. The  key is a special case as it returns the value 13 that cannot be typed into a string - hence the "+ CHRS(13)" at the end of the string. The variable Cnote stores the pitch value for the musical note C. By varying this between 48, 96, 144 and 192 it is possible to play 4 octaves.

Line 20 is the one which reads the keys pressed on the keyboard and puts them into a\$.

Line 30 gives us a neat way to stop the program (without having to hit Stop). It tests to see if the character typed was the space bar and if so the program stops.

Line 40 checks to see if the character type was between "1" and "4". If it was it sets Cnote equal to 48 * the number 1, 2, 3 or 4.

Line 45 converts any lower case letters that have been typed into upper case. This relies on the fact that the character numbers of all the upper case letters are exactly 32 less than the lower case letters. ASC converts a string to its character number and CHRS converts a character number back to a string.

Assuming the key pressed wasn't 1, 2, 3, 4 or space, Line 50 then looks up the character that has been typed in k\$ and sets the variable called note to be equal to the position number (so A=1, W=2, S=3, E=4 and so on up to =21).

Line 60 then checks to see if note is zero (which means the key wasn't found in k\$). If this is the case a jump is made back to line 20 to read another key.

The program will get to line 70 if the variable note contains a valid note number. This is multiplied by 4 because sounds go up in

PROGRAM 3 : Scientific graph plotting

This program asks for a mathematical function to be typed in then plots a curve of it on the screen.

```
10 pi4 = PI * 4 : st = pi4 / 480 : xscale = 480 / pi4
20 INPUT "Function of x to plot (eg SIN(x)) : " func$
30 INPUT "Y scaling value (try 63 to start) : " yscale
40 CLG
50 PRINT TAB(0,0);func$
60 FOR x=0 TO pi4 STEP st
70 y = INT(EVAL(func$)*yscale + 63)
80 PLOT 69, x * xscale, y
90 NEXT x
```

Line 10 defines some constants to be used in the program (you can have several statements on one line separated by colons ':'). The reason for making the variable pi4 is that we use the value of PI (3.14159) multiplied by 4 on several occasions so it is quicker to just calculate it the once. The variable st is used as the STEP value in a FOR..NEXT loop and xscale is used to make sure that the plotted graph will exactly fill the full width of the screen.

In line 20 The user is asked to input a function to plot. BBC BASIC is very powerful in that one can type in a function of x and later have it evaluated even though it is effectively just a string of characters.

Line 30 lets the user set a y scaling value. The reason for this is that some graphs have a much larger amplitude (top to bottom height) than others. Lower values input for y scaling will reduce the height of the graph until it will fit on the screen.

Line 40 clears the graphics screen ready to plot the graph and line 50 just prints the function that has been input as a title. The TAB(0,0) makes sure that the text appears in the first column of the first line on screen

Line 60 sets up a FOR..NEXT loop that will step x from 0 to PI*4 (this is a measure of angle in radians and is the equivalent of 720 degrees or twice round a circle). So that 480 dots are printed across the screen the previously calculated STEP value is used to increase x in a very small increment each time.

Line 70 is the business end of the program that takes the function of x that has been input and evaluates it with EVAL. The resultant y value is multiplied up by the y scaling factor so that numbers between 0 and 1 (such as you get from SIN(x)) will make a

noticeable displacement on the display (+/- 64 pixels from the centre line). The resulting value is added to 63 to position it about the middle of the screen and only the integer part is taken (using the INT function) because co-ordinates with decimal fractions would not make much sense to the PLOT command.

Each dot of the curve to be plotted is individually set by line 80. This uses the ubiquitous PLOT command which has myriad uses. It just so happens that "69" is the one that means plot a point at an absolute (x,y) position. A list of all the possible PLOT commands is given later in the manual.

Line 90 just completes the FOR..NEXT loop so that all 480 dots across the screen are used.

When you RUN this program start off with an easy function. Enter "SIN(x)" for the function (the word SIN must be in upper case but x must be in lower case). When asked for the scaling value enter 63.

Having tried that, run the program again but this time enter the function as:

```
SQR (ABS (SIN (x)))
```

again use a scale value of 63. The reason for the ABS function in the above is to prevent the SQR (square root) function being given a negative value which will just cause the program to stop with an error.

For a final run of the program try the very interesting (and complex) function:

```
SIN (x) * COS (x) - SIN (x) * SQR (x)
```

For the scaling value enter 16. This curve starts off with a small amplitude but gets greater and greater as the plot continues.

Try making up your own functions using combinations of the various mathematical functions in BASIC which are listed in the brief summary of all BASIC commands that follows later.

produces one quadrant to a circle. It is far easier to use the formulae that:

```
x = r sin(theta)
```

and

```
y = r cos(theta)
```

Then write a program that varies theta between 0 and 360 degrees and calculate the (x,y) points on a circle. This is roughly what the program does. However, it is recognised that a clock face only has 60 distinct points so there is no point in calculating 360 points. So the program just counts theta up from 0 to 59 and then uses the values of COS and SIN of theta multiplied by 6. As there is a lot of sine and cosine calculations to be done, the program does all the calculations at the very start and stores the fixed results in two arrays of variables. In addition to these a second set of arrays hold the same results but shifted across the screen by 240 points and up by 63. (To the centre of the screen).

The arrays are chosen to be integer arrays (that's what the "%" after each name means) this makes them quicker to access and more compact.

While the initial calculations are being made and the 60 points on the clock face are being drawn, a check is made to see if theta is divisible by 5 (using the MOD function). When it is a named procedure (Draw_Hour_Blob) is called to make a bigger mark to distinguish the twelve hour points on the clock face.

Once the face has been drawn the main loop of the program is entered. Every second this reads the setting of the system date and time (in TIMES) and breaks out the hour, minutes and seconds settings into separate strings. The strings are then converted to numbers using the VAL function. For each of the three quantities a call is made to the Draw_Hand named procedure. When this is called two parameters are supplied. One is the setting of hours, minutes or seconds. And the other value passed is the length of the pointer (*1.0 would be as long as the radius of the clock face, *0.4 is 2/5th the length of the radius and is used for the hour hand). Once the strings have been converted to numbers the hour value is multiplied by 5 to make it in the range 0.55. One twelfth of the minutes setting is added so that the hour hand will take one of five distinct positions between one hour and the next.

PROGRAM 4 : An analogue clock

```
10 DIM sn%(60), cs%(60), snoff%(60), csoff%(60)
20 CLG
30 FOR theta=0 TO 59
40 sn%(theta) = INT(60 * SIN(RAD(theta * 6)))
50 snoff%(theta) = sn%(theta) + 240
60 cs%(theta) = INT(60 * COS(RAD(theta * 6)))
70 csoff%(theta) = cs%(theta) + 63
80 PLOT 69, snoff%(theta), csoff%(theta)
90 IF (theta MOD 5)=0 THEN PROC_Draw_Hour_Blob(theta)
100 NEXT theta
110 hour$=MID$(TIMES$, 17, 2)
120 min$=MID$(TIMES$, 20, 2)
130 sec$=MID$(TIMES$, 23, 2)
140 PRINT TAB(60, 4);MID$(TIMES$, 17, 8)
150 hour=VAL(hour$) : min=VAL(min$) : sec=VAL(sec$)
153 IF hour<12 THEN hour=hour-12
155 hour=hour*5 + (min/12)
160 PROC_Draw_Hand(hour, 0.4)
170 PROC_Draw_Hand(min, 0.8)
180 PROC_Draw_Hand(sec, 0.9)
190 IF INKEY(1)=-1 THEN GOTO 110
200 STOP
300 :
400 DEF PROC_Draw_Hand(time, length)
410 newtime = time
420 IF time=0 THEN oldtime=(time-1) ELSE oldtime=(time+59)
430 MOVE 240,63
440 PLOT 7,sn%(oldtime) * length + 240, cs%(oldtime) * length + 63
450 MOVE 240,63
460 DRAW sn%(newtime) * length + 240, cs%(newtime) * length + 63
470 ENDPROC
480 :
500 DEF PROC_Draw_Hour_Blob(angle)
510 MOVE snoff%(angle), csoff%(angle)
520 DRAW sn%(angle) * 1.1 + 240, cs%(angle) * 1.1 + 63
530 ENDPROC
```

It's a pretty mammoth program but we hope you think it is worth the effort of typing it in!

The program draws a circular clock face and shows the current time that the Notebook's clock is set to as a set of pointers on the clock face. The time is shown as text digits alongside.

There are several ways to draw a circle on a computer. One method is to use the formula for a circle $r^2=x^2+y^2$ and rearrange this to give $y=\sqrt{r^2-x^2}$. r - the radius is a fixed quantity (60 in our case) so you just vary x between 0 and r and calculate the corresponding y values and plot the points. However, this only

After the calls to draw the hands a check is made to see if a key is pressed. If one is then the program stops, otherwise it jumps back to read the new time and re-draw the pointers. That forms the main loop of the program. Line 300 with a single colon is just a neat way of spacing the lines of the main program from the procedure definitions that follow.

Finally there are the two procedures. The one to draw hands uses two time settings, the current hour, minute or second and the previous hour minute or second. It draws a blank line (PLOT 7) at the old location and then draws in the new pointer. When calculating the previous minute a special case of the hour/min/sec being 0 is made. In this case the previous hour/min/sec would be negative so 59 is added on.

The final procedure is the one that draws the extended legends on the "hours". It does this by drawing lines out from the circle edge to points that are 1.1*the radius further out.

We hope you enjoy these simple programs and may be inspired to delve deeper into the world of BASIC programming. The range of things you could attempt is endless. How about writing a drawing package? Or a cardfile program? Or a spreadsheet? Or a game? Or a terminal program to use the serial port (hint: treat it like a file called "COM:;") ?

Making programs run automatically

If you write a program in BASIC, each time you want to run it it would seem that you have to switch to BASIC by pressing **Function** and **⏏**, give a LOAD "filename" command and finally type RUN to start the program running. If you want to make it easy for others to use your program then give it the special name "AUTO". If there is a file of this name in the Notebook's memory then when you press **Function** **⏏** the program will be immediately loaded and start running. It is possible to have a second file called NOTEPAD.RUN which is also auto-run. This will be loaded and run after the program called AUTO.

If such a program contains an OSCLI("ESC OFF") command then you might get into the situation that you could never get back to BASIC's immediate mode. This is not a problem. Switch to the wordprocessor's List Stored Documents screen and use **⏏** **⏏** to rename the file to something other than AUTO or NOTEPAD.RUN.

BASIC memory usage

BBC BASIC allocates as much RAM as possible for its use on entry, up to a maximum of about 40 Kbytes for user programs, variables and stack. This figure cannot be increased by the addition of a PCMCIA memory card but such a card would allow several programs to be stored and the CHAIN command could be used to switch from one to the next.

If less than 40K of memory is available the value of PAGE will be raised accordingly. At least 2 Kbytes of memory will be left free for new files. Note that you must not raise HIMEM above its initial value or lower PAGE below its initial value; any attempt to do so will most probably crash the machine.

It may also be possible to crash the machine by injudicious use of BBC BASIC's equivalent of the POKE command (?n = x). Also, almost any attempt to use the assembler built into BBC BASIC will crash the machine. If you do any of these things then the only way to correct matters may be to perform a hard reset by holding down **Function** **⏏** and **⏏** when switching the machine on. This will lose all data and documents stored in the machine's memory. It would, therefore, be advisable to avoid using any of these features unless you are absolutely sure you know the effect they may have.

Z80 assembler

Unlike the Acorn computers that are based on the 6502 processor, the Notebook is based on the Z80 processor. Consequently, the assembler built into the BASIC recognises Z80 mnemonics rather than the 6502 variety. Assembly language programming is such an advanced subject that there is no way it can be covered in this manual. We would warn people not familiar with machine code programming to avoid attempting to use the assembler feature as it will almost certainly lead to a machine crash and subsequent loss of data.

Differences between BBC BASIC on the Notebook and other computers

Not every feature of BBC BASIC is supported by the version in the Notebook. Obviously the sound facilities are very limited and the screen is only 480 pixels wide by 128 pixels deep, also, it can only show two "colours". This does mean that some of the standard BBC BASIC commands are limited on the Notebook.

The following pages comprise a list of all the keywords recognised in BBC BASIC, this is not a complete reference but may be useful to those who already know a version of BASIC.

Following this is a description of operating system specific features of BBC BASIC on the Notebook.

Those language elements which are machine specific, particularly hardware-dependent features, are indicated accordingly. In most cases their operation has been made as compatible as possible with the original Acorn versions, within the constraints of the NC200 design and its operating system: