

+-----+
| BASIC'S ASSISTANT |
+-----+

Written by Charles Peter White 1987-1993

Converted to PDF by Thomas Antoni 2000
thomas@antonis.de <http://www.antonis.de>

NOTICE

This book is copyright. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electrical, mechanical, photocopying, recording or otherwise without the prior written permission of Charles P. White.

Every effort has been made to ensure complete and accurate information concerning the material presented in this book. However, the Author, can neither guarantee nor be held legally responsible for any mistakes in printing or faulty instructions contained in this book. The Author will appreciate receiving any information on this subject, including any errors or misprints. The programs presented in this book are for demonstration and testing purposes only. The Author will take no responsibility nor can be held legally responsible for damages caused by improper handling or use of the demonstration programs.

This is the First Edition and First Publication.
First released on Shareware Diskette in 1987/88.
Registration required at 10 uk pounds sterling.
Used as a teaching aide: 70 uk pounds sterling.

Preface

I hope that this book assists you to understand the concepts of Basic Programming a little better. It may be of assistance, if you have already used Basic on another Computer and wish to move over to IBM standards. With this in mind, this book may help to covert some routines from a non standard Basic.

This book is for the Beginner and in no way implies a complete coverage of the Basic language.

Special thanks to Mr. David Bransfield for giving up his time to do all the prove reading for me.
Thankyou David. 1987/88/89/90/91

Reference material used: Using Basic on the IBM PC
By : Angela and Michael Trombetta

A special thankyou to Angela and Michael Trombetta for a well written book.

Charles Peter White
November 1987,1988,1989,1990,1991,1992 and 1993

This book is dedicated to my wife Pauline, and
daughters, Paule and Christine.

```
+-----+  
| BASIC'S ASSISTANT |  
+-----+
```

Written by Charles Peter White 1987-1993
26, Oakdene, Stourport-On-Severn,
Worcestershire, DY13 9NF, ENGLAND.
(C) Copyright. All Rights Reserved 1987-1993

```
+-----+  
| BasicA/GWBasic/P-Basic/Quick Basic/Turbo/Power. |  
+-----+
```

TurboBasic(r) Borland International Inc.
PowerBasic(r) Spectra Publishing.

Microsoft(r), and MS-DOS are registered
trademarks of Microsoft Corporation.

IBM(r) is registered trademark of
International Business Machines Limited.

Introduction

This book was written for those of you who really want to program but have found it difficult because of the technical way books have previously been written. You can type in the routines (which are covered with notes on how things work), and see how they operate and what results they produce. I have placed a great deal of emphasis on routines that really work, giving good screen layouts, proper keyboard entry systems (with error checking for wrong key entry) and a host of other things.

The Basic's Assistant is a quick tutorial for the Beginner and the Intermediate programmer. Basic is referred to as Microsoft's BasicA, TurboBasic, PowerBasic, P-Basic, Basic, GW-Basic and Quick Basic 3.0 to 4.5. These programming languages are generally used for IBM machines and close compatibles, some differences may be found in some machines. MS/PC-DOS is also required to execute the routines in this book. It is suggested that you have the manual [as supplied with your computer or obtained from a library] for one of the topics you wish to learn or further your knowledge. BASIC terms will be used in the form of GW-Basic, as this has all the BasicA commands and a few other subtle differences, while Quick Basic will be in both GW-Basic and Quick Basic 3.0 and above [if possible]. This book is divided into four main sections, 1) Lets Get going, 2) Reference. This will list out the commands and their meanings, giving you a quick and easy reference, 3) Errors. If you have typed something wrong or the computer states that an error has

occurred, this will show what has happened, and finally, 4) USEFUL TIPS. Some tips on how to look at keys, what type of monitor you are using and other functions.

For Amstrad users, use P-Basic/Turbo/Power/QB.

BASICS and their use

Although Basic is a High Level language and Assembler is a low level one, it should be understood that Basic is very powerful and can do almost anything that assembler can. Only a few points are against Basic as a programming tool.

A). Basic is slow because it requires an interpreter to run it, thus slowing the speed down, and only allowing 64,000 bytes to be used 'in full' as the main program.

B). Programming is simple and requires little thought, but the finished program can be 'all over the place' without the need for proper structuring and therefore leaving the programmer at a loss when he/she comes to correct or add routines.

C). To produce clever effects with scrolling screens, special windows and nice pull-down windows, it takes a lot of thought and a great deal of programming and memory.

D). The larger the program the smaller the data files become.

HOWEVER.....

.....

+---+
| 4 |
+---+

There are many other types of special utilities on the market that will enhance your Basic programs and produce very fast and compact routines that will work efficiently and use memory as machine code does. These Utilities are called Compilers, Basic aids and Basic enhancements. They need not cost the Earth, and will provide the user 'That's you', with most of what you require, with the desired effects. It is best to remember that if you require these special programs/aids, then it is in your interest to make sure that they are compatible with your system and do not require special knowledge to operate them. Most helpful routines should be written in assembler with special code that you can use within your basic programs. Having to adapt them is really a waste of time and money. Always find someone who knows the product and can show you how it works with 'YOU IN MIND'. Always ask questions until you understand, NEVER pretend that you understand, you end up paying for a product that could be the wrong thing for you. You can get some very good products from Shareware at a very small production cost, which in most cases are complete with documentation. The Authors may ask for a small fee, which is in everyone's interest to pay. This encourages the Authors to produce better and better products.

The routine TXT2COM is also my early Shareware version. Please remember, that this routine can only be used for private and personal use only.

If you use a Bulletin Board to get your software, please check for full documentation and requirements for that product. If anything is missing, report it.

There are many companies that deal with Shareware and can be found in your computer magazines, and BBS Systems. The production cost will vary from one company to another but is usually between £1.50 to £3.50 per disk. It is in your interest to seek them and get the product lists from them. If you use them, REGISTER them. There is always something of interest. You can get products from a vast array of companies other than Shareware.

I have found that top quality products with support cost around 20-80 pounds. This includes registration fees. If you have written something yourself, then why not place your program/s with them for distributing. Remember to place a Copyright notice with your full name and address on all your programs and documentation. Any restrictions on use of your software must be placed within the documentation.


```

+-----+
| Index |
+-----+

```

Introduction	2
Alphabetical Index	216
Functional Index	220
Protect you Copyright Material	230
ANSI Colour Codes	231
ASCII Codes	234
Extended Key Codes	238
LETS GET GOING:	
Manipulating Strings	10
STRINGS.BAS	10
DATE.BAS	12
INDELOT.BAS	13
LOADER.BAS	21
Read and Data commands:	24
TXT2COM.BAS	25
Peek and Poke Grey keys:	30
PK-BRD.BAS	30
Function keys:	
FUNKEY1.BAS	33
FUNKEY2.BAS	35

Using the printer:

PRINTER.BAS	38
Reading and Saving data files:	47
ON ERROR and RESUME	49
APPEND	50
WRAP1.BAS	52
Modem/Network Communications Index::	62

REFERENCE

The direct command from DOS	118
The editor and its commands	120
Similar dos type commands in basic	127
Shortcut keys for the editor	129
Editing keys	130
File handling	132
Error trapping	142
Variables and constants	144
Command structures	147
Data handling	150
Mathematics	153
Screen output	156
Graphics	163

Getting input from the keyboard	173
String handling	175
Printer output	179
Techniques in using the interrupts	181
Machine code	185
Sound commands	187
Memory accessing	189
I/O communications	192
ERRORS	

Error messages	196
USEFUL TIPS	205

Special keys	206
Monitor type	206
Memory (RAM)	206
DOS keyboard fonts	206
Screen locations & DEMO program	207
Dis/Enable Keyboard	210
Check for various devices .BAS	210
COMMAND\$ Line Switching	212


```
+-----+
| 10 |
+-----+
```

```
+-----+
|Let's Get Going|
+-----+
```

After you type in a program, SAVE it before testing. Load in your Basic interpreter by following its manual.

Manipulating strings:

First we shall have a look at string handling. How to examine a string by splitting it up and printing the results.

STRINGS.BAS:

```
1 A$="ABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMN
OPQRSTUVWXYZ1234567890123456789012"
```

We first define a string (A\$) with some characters.

```
2 A=LEN(A$)
```

We now set a variable (A) to the total numerical LENGTH of A\$. In this case the length given to (A) will be 72 (characters).

```
3 CLS
```

Now clear the screen.

```
4 FOR I=1 TO A
```

We set a loop to start at 1 and end with the value of (A).

```
5 LOCATE 1,1:PRINT LEFT$(A$,I)
```

Set the cursor to the top of the screen and print the left expression of the string we set up in A\$ to the total value of characters as

```
+-----+
|  11  |
+-----+
```

determined by the value in 'I'. So, if 'I' =1 then the first character will be printed. If 'I' is 2, both first and second characters will be printed, and so on.

```
6 LOCATE 3,1:PRINT MID$(A$,I,1)
```

Set the cursor to the third line from the top and print only one character according to the value of 'I'. The (A\$ is the string, I is the position in that string we want, 1 is the amount of characters we want). Change 1 to 2 etc; and run the routine to see what you get. Then swap around I and 1. This will give you a very good idea of how the MID\$ function works. You could also change LOCATE 3,1 to LOCATE 3,I this will move the character along under the top display.

```
7 R$=RIGHT$(A$,I):LOCATE 5,1:PRINT R$
```

There we set R\$ as the right most character of the string A\$. As 'I' starts with 1 in the loop, the character will be the last, but as 'I' increases its value you will get the last two and so on. Here we are printing backwards, from the last to first.

```
8 S$=RIGHT$(A$,A-I):LOCATE 7,1:PRINT S$;" "
```

The string S\$ is set to the right most character of string a\$, but with a formula of A-I. 'A' is set as the total length of A\$, while 'I' starts with one in the loop. Therefore, as 'I' increases its value it is subtracted from 'A' which gives the result of the characters we want to see. If I=2 then the result will be a total of 70 (A=72, I=2, 72-2=70).

What we are doing here is to look at a chunk starting with the complete string (A\$) and

```
+-----+
|  12  |
+-----+
```

moving along one letter at a time to the right, Still looking and displaying the balance of A\$.

```
9 FOR T=1 TO 500:NEXT
```

This is a delay loop so that you can see what is happening.

```
10 NEXT
```

Continues the FOR I loop. When 'I' has reached the same value as 'A' then the loop finishes. That concludes this routine.

DATE.BAS

Here's a quick routine you can use to include in any program to give the correct dates form the year 100.

```
10 CLS:INPUT "DAY, MONTH, YEAR(SEPERATE WITH
COMMAS)";DD,MM,YY
20 IF YY<100 THEN YY=YY+1900
30 IF MM>2 THEN 50
40 MM=MM+12:YY=YY-1
50 N=DD+2*MM+INT(.6*(MM+1))+YY+INT(YY/4)-INT(YY
/100)+INT(YY/400)+2:N=INT((N/7-INT(N/7))*7+.5)
60 FOR A=0 TO 6:READ N$(A):NEXT:PRINT N$(N):END
70 DATA "SATURDAT","SUNDAY","MONDAY","TUESDAY"
,"WEDNESDAY","THURSDAY","FRIDAY"
```

You could modify it to read the system time first, then compare the date placed by the user and give the necessary responce.

Let us continue with a routine that will allow you to use the INKEY\$, RIGHT\$, MID\$, CHR\$() commands, read the keyboard, select special keys or not, insert/over type text. Move

HOME/END of a line and a host of other nice facilities. It is a short program to type in.

This routine is called INDELOT-
[insert/delete/over type].

INDELOT.BAS: FUNCTION KEYS

Control/O Over types old characters with
 new ones.

F1 Clear line F2 Display original text

F5 Save new text in another string

F9 Re-display the newly saved text

Home Cursor to the beginning of text line

End Cursor to the end of text line

Del Delete character under cursor [move the
rest of the text on the right of the cursor,
one place to the left].

<-Del Delete a character one position to the
 left of the cursor.

By default, this routine is in Insert mode. As you type, any characters to the right of cursor will move over to accommodate the typed character. For test reasons, there is only one line of text which is displayed at the top left of the screen. This enables you to see how text editors work, while not being too complicated to program. Type in the following LINE numbers. Pressing ENTER or RETURN when finished. Do not type in the explanation paragraphs which proceed each line. Your finished program will throw up errors.

+-----+
| 14 |
+-----+

```
1 A$="This is a test program":B=LEN(A$):PT=B:  
G$=A$
```

A\$ is the string which we want to print. B is the variable LENGTH of the whole string of A\$. In this case twenty two characters. PT is a switch variable we use later which shows the current length of the string. G\$ is made the same as A\$. As it is needed for the RESTORED original text key.

```
2 CLS:PRINT A$:LOCATE 1,1,1:S=1
```

CLS clears the screen. PRINT A\$ prints the text. LOCATE 1,1,1 places the cursor to the top left of the screen and switches on the cursor (the last 1 in the LOCATE command). S is to count the position of the cursor on that line.

```
10 R$="":R$=INKEY$:B$=RIGHT$(R$,1):L=LEN(R$)
```

We use R\$ for key input as in R\$=INKEY\$. Special keys that INKEY\$ can not get on its own, have to be looked at another way. Normal keys have a single value while others have two. You have to get the right expression of the double figure for those keys. B\$ is a check string of the RIGHT\$(R\$,1). 1 being the far right. L is the variable LENGTH of the R\$ (key pressed).

```
25 IF S<=B AND (L=2 AND B$="M") THEN  
S=S+1:PY=1:GOSUB 150:GOSUB 100:GOTO 10
```

If the cursor position (S) is less than or equals to the total length of the whole string (B) and the right cursor key was pressed, then ADD 1 to cursor position (IN S) ready to move it to the right. PY is a switch. If set to 1 it will do

something, If 0 then the routine or command will be avoided. GOSUB 150 and GOSUB 100 are temporary jumps to a routine with a 'RETURN' statement. GOTO 10 is a direct command to GOTO a specified line.

```

30 IF S>1 AND (L=2 AND B$="K") THEN
S=S-1:PY=1:GOSUB 150:GOSUB 100:GOTO 10

```

If the cursor position (S) is greater than one (the first character) and the key pressed was cursor left, then decrease cursor position by

one. Set switch on (PY=1) and do the two subroutines, returning to next command which is GOTO 10 (key input).

```

32 IF A$<>" AND (L=2 AND B$="S") THEN
B$=LEFT$(A$,S-1):GOSUB 150:A$=B$+C$:GOSUB 120

```

If the string (A\$) is not equal to empty and the key pressed was DEL then set B\$ as the LEFT part of the string from the cursor position. Temporary jump to line 150 and RETURN. Having deleted a character, set the new length of A\$ by adding B\$ with C\$ (which is the right side characters of the cursor.)

```

34 IF A$<>" AND (S>1 AND R$=CHR$(8))
THEN B$=LEFT$(A$,S-2):S=S-1:C$=RIGHT$(A$,
,B-S):A$=B$+C$:GOSUB 120

```

If the string (A\$) is not equal to empty and the cursor position is greater than one, (the first character), and the key pressed (R\$) was <-DEL then set B\$ to the left-1 of the cursor position.

Subtract cursor position by 1. Set C\$ for the characters on the right position of the cursor. Add B\$ with C\$ to make total of A\$. Temporary jump to line 120.

```
36 IF L=2 AND B$="O" THEN S=B+1:C$="":GOSUB 120
```

If an extended key was pressed and it was END then cursor position equals the total length of a\$ + 1. The cursor will be displayed one character after the last displayed. Set the string value used for the right part of the text to empty. Temporary jump to line 100

```
38 IF L=2 AND B$="G" THEN S=1:GOSUB 120
```

If the extended key pressed was HOME then set cursor position to beginning of line (position 1). Temporary jump to line 120

```
40 IF XO=0 AND R$=CHR$(15) THEN LOCATE 1,77:
PRINT"O/T":XO=1:OT=1:GOSUB 130
```

XO is a switch which indicates if a key for over-type has been pressed. 0=on 1=off. CHR\$(15)= CTRL/O. Place the cursor on the first line at position 77 and print O/T for Over-Type on. Set the XO switch off. OT states that over-type is on, this is used again when the same combination of keys is pressed to switch off over-type mode. Temporary jump to line 130

```
42 IF XO=1 AND R$=CHR$(15) THEN LOCATE 1,77:
PRINT"  ":XO=0:OT=0:GOSUB 130
```

If the XO switch is ON and CTRL/O was pressed then print three spaces on the top line where O/T was displayed. Set both XO and OT switches OFF ready for use.

```
44 IF L=2 AND B$=";" THEN H$=G$:A$="":S=1:
LOCATE1,1:PRINT SPACE$(B):B=0:LOCATE 1,S
```

If F1 was pressed then save the string into H\$ and empty the display string A\$. Place the

cursor at the top line and print spaces to the length of the string as in B. Set B switch to 0, as there is no string characters being displayed. Place the cursor back to the beginning of the line.

```

46 IF L=2 AND B$("<" THEN A$=H$:GOSUB 72:S=1:
GOSUB 120

```

If F2 was pressed then make the current string (A\$) from (H\$).
Temporary jump to routine line 72. Set the cursor position to the beginning of the line.
Temporary jump to 120.

```

48 IF L=2 AND B$="C" THEN A$=I$:GOSUB 72:S=1:
GOSUB 120

```

If F9 key was pressed, place saved text into current display (A\$).
Temporary jump to sub-routine 72. Set cursor position to beginning of line. Temporary jump to sub-routine 120.

```

50 IF L=2 AND B$="?" THEN I$=A$:GOSUB 72:
GOSUB 120

```

If F5 was pressed then make I\$ as a saved string of text from the currently displayed text. Temporary jump to sub-routine 72 and then to 120.

```

67 IF (S<=B AND OT=1) AND (R$>=CHR$(32) AND
R$<=CHR$(126)) THEN GOSUB 155

```

At this point we are looking at two specific conditions before we can do a procedure.

A) If the cursor position (S) is less than or equals to the length of the string (B) and OVER/TYPE (OT) has been selected and

B) If R\$ equals to character 32 (space) and R\$ is less than or equals to character 126 (~). Characters from 32 to 126 are only allowed.

If these two conditions are met then temporary jump to sub-routine in line 155. Both conditions must be met or we avoid this line.

```
69 IF (S+C<73 AND OT=0) AND (R$>=CHR$(32 AND
R$<=CHR$(126)) THEN A=1:GOSUB 100
```

Once again there are two sets of conditions that we are looking for.

A) If the cursor position plus any characters to the right are less than 73 (total characters per line) and Over/Type is off, and

B) If R\$ (key input) equals to character 32 (space) and R\$ is less than or equal to character 126 (~) then temporary jump to sub-routine 100. Once again, all these condition must be met or this line is avoided.

```
70 A=0:GOTO 10
```

The 'A' is a switch: See if you can work out what it is for.

```
72 LOCATE 1,1:PRINT SPACE$(B):RETURN
```

Locate the cursor to the top of the screen. Print a line of spaces according to the value of B and RETURN to the line that called this sub-routine.

```
100 IF S=1 AND A=1 THEN C$=RIGHT$(A$,B+1):
GOTO110
```

If the cursor position is on the first position of the line and the switch 'A' is 1 then set C\$

to WHAT ?

```
105 B$=LEFT$(A$,S-1):C=LEN(B$):C$=RIGHT$(A$,B-S+1)
```

```
110 IF A=1 AND S=1 THEN A$=R$+C$:GOTO 120
```

Now you should be getting the hang of this.
Try to work out what the rest of the program
does.

```
115 IF A=1 THEN A$=B$+R$+C$
```

```
120 IF PY=1 THEN PY=0:GOTO 122
```

```
121 LOCATE 1,1:PRINT A$ " "
```

```
122 B=LEN$(A$):C=LEN(C$)
```

```
125 IF A=1 THEN S=S+1
```

```
130 LOCATE 1,S:R$="":RETURN
```

```
150 IF S<=B THEN C$=RIGHT$(A$,B-S):PT=LEN(C$)+1
```

```
151 RETURN
```

```
155 B$=LEFT$(A$,S):C=LEN(B$):C$=RIGHT$(A$,B-S)
```

```
160 A=1:IF S=1 THEN A$=R$+C$:GOTO 120
```

```
162 A$=LEFT$(A$,S-1)+R$+C$:GOTO 120
```

Now you can see how to get input from the key
board and separate it into two different types
of selections. For standard keys and for the
special keys like F1 etc. How to control
keyboard input from allowing any other keys
other than the ones you wanted.

You have looked at the right part of a string
by using the RIGHT\$ and the left with LEFT\$.

To find the length of a string, the total number of characters/numbers etc, you have used LEN.

If you like this routine, why not try to see if you can make it smaller by altering it. See if you can get other special keys.

In this example we have not used the IF..THEN..ELSE conditions, but by a very slight alteration to the program they can be demonstrated.

Under line 34 we shall insert another line (35). Type in:-

```
35 If L=2 then 36 ELSE 40
```

Then remove both L=2 AND at the beginning of lines 36/37. They should both start with IF B\$=. Does that work ?

The next routine we will look at uses the FILES command with a combination of what was used before. Also used is then DIM statement, DEF SEG, KEY OFF and PEEK.

This routine will display the current directory contents on the screen and allow you to move the cursor around each item as if to select them for processing on.

```

LOADER.BAS:          FUNCTION KEYS
-----

```

The cursor keys move you through the displayed directory.

Some of the lines are very long. Just keep typing and then press RETURN, ENTER, or <+.

```

1 CLS: CLEAR: KEY OFF: CLT=PEEK(&H10): IF CLT=48
  THEN DEF SEG=&HB000 ELSE DEF SEG=&HB800

```

First clear the screen. Clear any variables/strings from memory. Turn OFF any pre-defined function keys. Now we look to see what type of monitor we are using. If that value is 48 then we are using a MONO monitor otherwise it's a colour one.

```

2 LOCATE 1,30: PRINT "DEMO FOR LOADING FILES":
  C=3:D=1: FILES

```

We use C for the Vertical position and D for the Horizontal position of the cursor. Now read the directory.

```

3 A=320: IF PEEK(336)=46 THEN A=480: C=4

```

'A' is set to the actual screen position we want for when we look at it with PEEK. If it is a 'full stop' then we move down one line (160 bytes). Remember that every even number holds the character value, while every odd number holds the colour value. 'C' is then reset to 4. One line further down.

```

4 FOR I=1 TO 12: B=PEEK(A): C$=C$+CHR$(B):
  IF PEEK(A)=32 THEN 6

```

Look at the 12 characters and place them into C\$. If one of them is a 32 (space) then goto line 6

+-----+
| 22 |
+-----+

```
5 B$=B$+CHR$(B)
```

The B\$ is made to hold the full character set as displayed on the screen. With the spaces. B\$ will be used to replace the inverted display when the cursor keys are used.

```
6 A=A+2:NEXT
```

As 'A' is the character position (even numbers), we scan the next location to the right. NEXT continues the FOR loop until all 12 have been scanned.

```
7 LOCATE C,D:COLOR 0,7:PRINT C$:LOCATE 23,1:
COLOR 7,0:PRINT "b$="      ":LOCATE 23,5:PRINT B$
```

```
8 S$=INKEY$:LL=LEN(S$):T$=RIGHT$(S$,1):
IF LL<>2 THEN 8
```

```
9 IF T$="M" AND PEEK(A+12)<>32 THEN GOSUB 15:
A=A+12:D=D+18:GOTO 4
```

```
10 IF T$="K" AND PEEK(A-38)<>32 THEN GOSUB 15:
A=A-60:D=D-18:GOTO 4
```

```
11 IF T$="P" AND (PEEK(A+136)<>32 AND
PEEK(A+136)<>101) THEN GOSUB15:A=A+136:
C=C+1:GOTO 4
```

```
12 IF T$="H" AND (PEEK(A-182)<>58) AND
(PEEK(A-184)<>32) THEN GOSUB15:A=A-184:
C=C-1:GOTO 4
```

```
14 GOTO 8
```

```
15 LOCATE C,D:COLOR 7,0:
PRINT C$:C$="":B$="":RETURN
```

INDEL0T is designed to be used with a short

directory. You can specify the type of files wanted by using various formats. ie;

```
FILES ".DOC"
FILES ".TXT"
```

See your computer manual for more details.

Let us now look at the READ and DATA commands.

type in the following:-

```
1 READ A$:PRINT A$:END
```

```
2 DATA "THIS IS A TEST"
```

Only one DATA Statement to read which is easy,

but what about more than one?

```
1 READ A$:IF A$=";" THEN END
```

We read the string A\$. If we find a semi-colon then end.

```
2 PRINT A$
```

```
3 GOTO 1
```

```
4 DATA "PETER","PAUL","MARY",";"
```

Using the READ/DATA statements are good, but they can be slow to execute, especially if the data statements are very long.

READ and DATA Commands

These data statements are in the form of strings.

Now we look at variables (numbers).

```
1 READ A:IF A=-1 THEN END
```

The -1 is set to determine the end of the data.

```
2 PRINT A
```

```
3 goto 1
```

```
4 DATA 23, 33, 123, 17, -1
```

You could do a loop, a FOR/NEXT statement. This is alright but requires a little more memory space. A combination of strings and variables in the data statement require some little thought. If you want to, you can read in by doing the following:-

```
1 READ A$,A:IF A$="," OR A=-1 THEN END
```

```
2 PRINT A$,A:GOTO 1
```

```
3 DATA "PETER",123,"PAUL",34,"MARY",36,";",-1
```

TEXT2COM.BAS:

The next routine will make all ASCII text files (like BAS, TXT, C, ASM,DOC, and so on) into COM files. Files that will load and run automatically, giving you Paging UP/DOWN, HOME (first page), END (last page), and to PRINT the document onto the printer.

The total length of a document to be converted must be less than 65,500 bytes long. The machine code (M/C) is in the form of DATA statements. This is read and placed into a file that you can specify or leave the original name by pressing RETURN.

Once the data has been SAVED the original text is added to that code and the file is then closed. The original ASCII text file will not be touched in any way. The now newly created COM file can be loaded to view the text. You can dispense with the original ASCII file if you want to.

The new statements used are OPEN filename FOR INPUT (OUTPUT) AS #1, INPUT#1, CLOSE, DEFINT, OPEN FOR RANDOM AS #1, FIELD #1, LSET, PUT#1, SHELL, READ, DEF F, and DATA.

```

1 CLS:CLOSE:CLEAR:LOCATE 1,7:PRINT"TEXT2COM:
MAKE ASCII FILES AUTOMATICALLY RUN AND DISPLAY.
Vs 1.00":LOCATE 2,25:PRINT"(c) C.P.WHITE
1987-1989":CG$=SPACE$(79)

```

```

11 DEFINT A-Z:DEFFNEXT$(S$)=
LEFT$(S$,INSTR(S$+".",".")-1) + ".COM":
ON ERROR GOTO 281

```

```

21 INPUT "Enter your source text file:
";INFILE$

```

```

31 PY=1:OPEN INFILE$ FOR INPUT AS #1

```

```

41 INPUT#1,A$

51 CLOSE:PY=0

61 OUTFILE$=FNEXT$(INFILE$):PRINT"Enter your
program name: ["; outfile$;"]";

71 INPUT" ",TEMP$:IF LEN(TEMP$)<>0 THEN
OUTFILE$=FNEXT$(TEMP$)

81 PY=0:OPEN OUTFILE$ FOR INPUT AS #1

91 INPUT#1,A$:IF A$<>" " THEN LOCATE
5,1:BEEP:PRINT CG$:BEEP:LOCATE 5,1:PRINT"FILE
EXISTS: CHANGE NAME![";OUTFILE$;"]";:CLOSE:
GOTO 71

101 CLOSE

111 OPEN"XXX.PGM" FOR RANDOM AS #1 LEN=1:
FIELD #1,1 AS A$

121 READ B$:IF B$="," THEN 151

131 BYTE=VAL("&H"+B$):LSET A$=CHR$(BYTE)

141 PUT #1:GOTO 21

151 CLOSE #1

161 OPEN"XX.PGM" FOR RANDOM AS #1 LEN=1:
FIELD #1,1,A$

Open a temporary file.

171 B$="1A":BYTE=VAL("&H"+B$):LSETA$=CHR$(BYTE):
PUT #1:CLOSE

Place the value of two bytes (1A) to the file
and close it.

```

```

181 JG$="COPY /B XXX.PGM + " + INFILE$ + " +
XX.PGM " + OUTFILE$

```

Now set the string JG\$ to use the DOS copy command. Copy the files xxx.pgm + infile\$ + xx.pgm into a file OUTFILE\$.

```

191 SHELL JG$

```

This now uses the DOS command to copy.

```

201 SHELL "DEL XXX.PGM"

```

DOS command to delete.

```

211 SHELL "DEL XX.PGM"

```

```

221 CLS:LOCATE 12,35:PRINT"JOB COMPLETED":CLOSE

```

```

231 LOCATE 14,30:PRINT"ANOTHER CONVERSION
(Y/N)"

```

```

241 G$=INKEY$

```

```

251 IF G$="Y" OR G$="y" THEN RUN

```

```

261 IF G$="N" OR G$="n" THEN END

```

```

271 GOTO 241

```

```

281 IF PY=0 AND ERR=53 THEN RESUME 101

```

If switch PY=0 and the error had a value of 53

```

291 IF PY=1 AND ERR=53 THEN LOCATE 12,24:PRINT
INFILE$;" NOT PRESENT..TRY AGAIN..":FOR U=1 TO
15:BEEP:NEXT:RUN

```

If the switch PY is 1 and the error had a value of 53, then the file that you stated was to be converted was not present.

```

301 LOCATE 12,30:PRINT"DEVICE ERROR...TRY
AGAIN":FOR U=1 TO 15: BEEP:NEXT:RUN

```

This line prints the error message and re-runs the program.

Locate the cursor at vertical tab 12 and horizontal position 30.

Print the error message. Do a loop for 15

counts while beeping and then re-run.

The next lines hold the data statements for the machine code that will be placed onto the disk/ette.

```

6000 DATA EB,38,90,9,9,45,6E,64,20,20,20
,48,6F,6D,65,20,20,20,50,67,55,70,20,20,
20,50,67,44,6E,20,20,20,45,53,43,20,20,20,
50,3D,50,72,69,6E,74,20,44,6F,63,24,0,0,0,
0,0,0,0,0,BE,0,0,BF,0,0,83,3E,34,1,1,74,3,
E8,0,1,52,51,53,B9,0,0

```

```

6005 DATA 8D,1E,BC,2,8A,17,80,FA,1A,74,6C,83,
6,38,1,1,83,3E,34,1,1,74,4,B4,2,CD,21,43,83,
3E,38,1,50,74,5,80,FA,A,75,DC,83,6,36,1,1,
41,89,E,32,1,83,3E,38,1,50,74,1,47,C7,6,38,1,
0,0,83,F9,17,72,BF,B9,0,0,83,3E,34,1,1,74,B5,B4

```

```

6010 DATA 0,CD,16,3C,5A,7C,2,2C,20,3C,1B,74,
77,3C,50,74,46,80,FC,47,74,72,80,FC,51,74,7E,
80,FC,49,74,61,80,FC,4F,74,5,EB,D8,EB,61,
90,80,FA,1A,74,D0,C7,6,34,1,1,0,EB,8,90,83,
3E,36,1,18,76,E6,59,BE,0,0,C7,6,36,1,0,0,BF,
0,0,C7,6,32,1

```

```

6015 DATA 0,0,E9,4B,FF,EB,CE,8B,F3,52,B4,2,
BA,0,0,CD,17,F6,C4,29,75,13,53,8D,1E,BC,2,
8A,17,80,FA,1A,74,7,B4,5,CD,21,43,EB,F2,5B,5A,
8B,DE,8A,17,EB,A4,EB,31,90,B4,4C,CD,21,EB,AE,
83,3E,34,1,1,74,23,C7,6,34,1,0,0,EB,8C,80,FA,
1A,74,89,C7,6

```

```
6020 DATA 32,1,0,0,E8,3,0,E9,A,FF,B4,0,B0,2,  
CD,10,E8,4A,0,C3,83,3E,36,1,17,76,9A,51,56,  
8B,F7,8B,E,32,1,83,3E,34,1,1,74,3,83,C1,17,C7  
,6,34,1,0,0,2B,F1,83,EB,1,8A,17,83,FF,1,74,  
A9,80,FA,A,74,2,EB,EF,3B,FE,74,3,4F,EB,E8,5E,  
59,C7
```

```
6025 DATA 6,32,1,0,0,B9,0,0,83,C3,1,EB,9C,53,  
52,B7,0,B6,18,E8,F,0,BA,3,1,B4,9,CD,21,B6,0,  
E8,3,0,5A,5B,C3,B2,0,B4,2,CD,10,C3,""
```

There is one thing to be said about this

program. The display of the text on to the screen is in teletext mode (printing one character at a time). It may be a bit slow, but it does the job. Printing is done continuously without any form of paging.

Therefore, any listing done will print continuously unless the text file that you converted already had page feeds written into it. Lines with more than 80 characters will wrap around the screen onto the next line.

If you are producing your own documentation files, it may be a good idea to ensure that they are not more than 78 characters per line. This is because you will have to work out the paging.

PEEK AND POKE GRAY KEYS: PK-BRD.BAS:

The next program listing is for PEEK and POKE. Looking at the keyboard and getting NUM/LOCK and CAPS/LOCK. The program is written in a round-about-way. You should be able to sort it by just having a few DEF SEG statements. You can see the values of other special keys in the section called: 'Useful Tips'.

```
1 CLS:DEF SEG=(&H40)
```

Clear the screen and set the computer to look at address &H40.

```
2 A=PEEK(&H17)
```

This the location that the keys like NUM/LOCK are looked at.

```
3 IF A<>32 OR A<>64 OR A<>96 THEN DEF
SEG:LOCATE 1,60:PRINT "    NORMAL    "
```

Make the condition so that the two keys we are looking at (plus a combination of both) are only allowed. If not, then print at the top of the screen "NORMAL", to show nothing was pressed.

```
4 IF A=32 THEN DEF SEG:LOCATE1,60:
PRINT"NUM/LOCK    "
```

If 'A' is NUM/LOCK then end the DEF SEG statement and print the key that was pressed.

```
5 IF A=64 THEN DEF SEG:LOCATE 1,60:
PRINT"          CAPS"
```

```
6 IF A=96 THEN DEF SEG:LOCATE1,60:
PRINT"NUM/LOCK + CAPS"
```

+-----+
| 31 |
+-----+

```
11 LOCATE 9,26:PRINT"Select mode to switch  
on":PRINT
```

We now show a selection menu.

```
12 LOCATE 11,26:PRINT"1.  NUM/LOCK"
```

```
13 LOCATE 12,26:PRINT"2.  CAPS"
```

```
14 LOCATE 13,26:PRINT"3.  NUM/LOCK + CAPS"
```

```
15 LOCATE 14,26:PRINT"4.  nothing (normal)"
```

```
16 LOCATE 15,26:PRINT"5.  KEYBOARD (press a key  
first)"
```

```
17 LOCATE 16,26:PRINT"0.  QUIT"
```

Now to input from the keyboard.

```
18 G$=INKEY$:IF G$="1" THEN DEF SEG=(&H40):POKE  
&H17,32:GOTO 2
```

We POKE the value of 32 in address &H17. Switch
on NUM/LOCK.

```
19 IF G$="2" THEN DEF SEG=(&H40):POKE  
&H17,64:GOTO 2
```

We POKE 64 for CAPS/LOCK

```
20 IF G$="3" THEN DEF SEG=(&H40):POKE  
&H17,96:GOTO 2
```

We POKE 96 for the combination of both keys.

```
21 IF G$="4" THEN DEF SEG=($H40):POKE  
&H17,0:GOTO 2
```

We set the both keys OFF, back to normal.

```
22 IF G$="5" THEN 1
```

Now we want to have a look at what keys were pressed. To make this work, you must press either NUM/LOCK or CAPS/LOCK or both and then press 5. This way we are now PEEKing (to look for) a key.

```
23 IF G$="0" THEN END
```

```
24 GOTO 18
```

The next routine is for your use. It will display all the values of the special keys (when you press the key) at the top of the screen. Press the ALT key to properly exit. You may find that the values are somewhat different, depending on the machine you are running on. Change the above program if necessary.

```
1 CLS:DEF SEG=(&H40)
```

```
2 A=PEEK(&H17)
```

```
3 DEF SEG:LOCATE 1,1:PRINT "      "
```

```
4 LOCATE 1,1:PRINT A
```

```
6 GOTO 1
```

Function keys:

FUNKEY1.BAS:

The next section deals with the function keys. Here, there is a listing that shows you how to program them to prevent certain sequences of keys from being pressed. ALT/CTRL/BREAK which will automatically reboot the computer, and CTRL/BREAK which will halt a program (allowing people to see how your program works). Another listing which sets up the bottom line to display the type of function and the routine itself to perform them. This is used extensively on many programs, giving them a professional touch.

The first program demonstrates the protection against pressing unwanted keys. This may not work on earlier versions of Basic, as keys 15 to 19 are used for the 101 keyboard. You may have to upgrade your Basic Interpreter. This is not the case with QB3 to 4.5

```
1 KEY 15,CHR$(&H8)+CHR$(&H1D):KEY
16,CHR$(&H4)+CHR$(&H38)
```

Key 15 is set for ALT/CTRL and Key 17 for CTRL/ALT

```
2 KEY 17,CHR$(&H4)+CHR$(&H55):KEY
18,CHR$(&H8)+CHR$(&H53)
```

Key 17 is set for CTRL/.DEL and Key 18 for ALT/.DEL

```
3 KEY 19,CHR$(&H4)+CHR$(&H46)
```

Key 19 is set for CTRL/BREAK

```
4 FOR I=15 TO 19:KEY(I) ON:ON KEY(I) GOSUB  
6000:NEXT
```

Do a loop from 15 to 19 switching on the

control keys. Then if an error has occurred (if a sequence of keys was pressed as stated in the function keys 15 to 19), jump to an Error message section.

```
10 CLS:LOCATE 10,25:PRINT"DEMO to prevent  
ALT/CTRL/BREAK sequences"
```

Clear the screen. Set cursor to vertical position 10 along line 25 and print our message.

```
11 LOCATE 12,25:PRINT"PRESS ESC TO END"
```

```
12 G$=INKEY$:IF G$=CHR$(27) THEN 14
```

Get a key and see if is Chr\$(27) for the ESC key.

```
13 GOTO 12
```

```
14 FOR I=15 TO 19:KEY(I) OFF:NEXT
```

Switch off the predefined keys, otherwise we will not be able to stop a program by using CTRL/BREAK or re-boot the system with ALT/CTRL/.DEL. You could disable them completely by adding to the KEY(I) OFF with :KEY(I)". This will clear the predefined statements as in lines 1 to 3.

```
15 END
```

```
6000 RETURN
```

This RETURN is part of the ON KEY(I) statement

in line 4. If an ERROR had occurred (when any sequence of keys was pressed, as defined in lines 1 to 3), the routine will jump here and then return to the program. You could have an error message printed here. For example:-

```
6000 LOCATE 23,1:PRINT"YOU CAN NOT STOP THIS
PROGRAM":FOR I=1 TO 10:BEEP:NEXT:LOCATE
23,1:PRINT"                                ":
RETURN
```

PLEASE NOTE !!

It is important that you always REM out this routine while you are working on your program. It is possible that, while testing other parts of a program, you may have forgotten to set a key correctly or even an escape key to escape. This will cause you to lose your program, as you will have to reboot the system by turning OFF and ON again. For the final version, just before saving, remove the lines 12 to 14 as these are only for your benefit. This routine is of more use for Compilers.

FUNKEY2.BAS:

To program the function keys 1 to 10, you can add sequences of control codes to perform things like a SHELL or FILES and so on. The following program will set up the bottom line to display the commands and to activate those commands when a function key is pressed. The maximum length of the displayed character at the bottom for each function, is 6.

Let's take a look at the following:-

```
1 CLS:SCREEN 0,0,0:WIDTH 80:CLEAR:FOR I=1 TO
10:KEY I,"":NEXT
```

First clear the screen. Set the screen to the default text screen 0. Set the width to 80 columns. Clear memory space just incase there is anything lurking around. Do a loop from 1 to 10 clearing all the function keys of any previous defined functions.

```
2 KEY 1,"LIST "
```

Function key, when pressed, will write list to the screen. You can decide which lines to list or just press RETURN, ENTER or <+.

```
3 KEY 2,"RUN"+CHR$(13)
```

Function key 2 has the command RUN followed by a RETURN. This will (when pressed) make the current program automatically execute without you having to type in RUN followed by return.

```
4 KEY 3,"LOAD"+CHR$(34)
```

Displays LOAD command with an inverted comer (") ready for you to just type in your program name followed by RETURN, ENTER or <+.

```
5 KEY 4,"SAVE"+CHR$(34)
```

As above but with the SAVE command.

```
6 KEY 5,"CONT"+CHR$(13)
```

Continue to execute a program if BREAK was pressed.

```
7 KEY 6,""+CHR$(34)+"LTP1 "
```

```
8 KEY 7,"CLS":FILES"+CHR$(13)
```

This will clear the screen and display the

+-----+
| 37 |
+-----+

current directory.

9 KEY 8,"FILES "

10 KEY 9,"KEY "

11 KEY 10,"SCREEN 0,0,0"+CHR\$(13)

12 KEY ON

Switches on the display at the bottom of the
screen.

Using the printer

Your printer manual should cover all the aspects of your printer.

It is worth remembering that PRINT prints to the screen, PRINT # file number, will to a drive, while LPRINT will print to the printer.

The TAB() command in Basics shifts the character/s to be printed to the numeric expression given in the brackets. However, the characters printed before are not removed, unlike printing on the screen. You should use the command given in your manual rather than any other.

Use your printer manual to understand how the printer works and then try some little routines.

As an example, you could change the PRINT statement in these listings to LPRINT and see what happens. Following your printer manual, add other commands to change the script and tabulations.

PRINTER.BAS:

This next program will allow you to select the type of font you want, and then after typing some text and pressing RETURN, will print to the printer.

Type in only the line numbers and their text. Remember not to type the comments.

```
1 CLS:CLEAR:ON ERROR GOTO 67
```

Clear the screen. If an error occurs, then goto our error routine. The next lines set up strings for printer font types. Have a look at

your own printer manual and compare the CHR\$(
statements to what they are.

```

2 SMALL$ = CHR$(15):UNSMALL$ =
CHR$(18):EXPAND.SMALL$ = CHR$(14) + CHR$(15)

3 CANX.EXP.COMP$ = CHR$(18) + CHR$(20):EXPAND$
= CHR$(14)

4 EXPAND.END$ = CHR$(20):DARK$ = CHR$(27) +
CHR$(69)

5 ITALIC$ = CHR$(27) + CHR$(52)

6 ITALIC.EXP.CON$ = CHR$(14) + CHR$(15) +
CHR$(27) + CHR$(52)

7 ITALIC.EXP$ = CHR$(27) + CHR$(52) +
CHR$(14):END.ITALIC$ = CHR$(27) + CHR$(53)

8 START.ULINE$ = CHR$(27) + CHR$(45) +
CHR$(1):END.ULINE$ = CHR$(27) + CHR$(45) +
CHR$(0)

9 PERM.EXPAN$ = CHR$(27) + CHR$(87) +
CHR$(1):END.PERMEXP$ = CHR$(27) + CHR$(87) +
CHR$(0)

10 START.DSTRIKE$ = CHR$(27) +
CHR$(71):END.DSTRIKE$ = CHR$(27) + CHR$(72)

11 TINYLINE$ = CHR$(27) + CHR$(83) +
CHR$(0):END.TINY$ = CHR$(27) + CHR$(84):ESC$ =
CHR$(27):CLEANUP$ = CHR$(27) + CHR$(64)

13 WIDTH "LPT1:",80

```

We set up the printer width to 80 columns using
printer device port LPT1.

+-----+
| 40 |
+-----+

```
14 LPRINT CLEANUP$;
```

Set the printer.

```
15 CLS:DIM A$(60)
```

Clear the screen. Dimension the A\$ to not more than 60 items per line. A\$(1) could have "Dear Sir". A\$(2) could have "HI !" and so on.

Now set out on the screen the selection menu.

```
16 LOCATE 2,25:PRINT "SELECT PRINT SIZE AND  
STYLE"
```

```
17 LOCATE 4,20:PRINT "    PRINT SIZE  
SELECTION"
```

```
18 LOCATE 5,20:PRINT "    =====  
====="
```

```
19 LOCATE 7,20:PRINT "10 PITCH ITALICS  
1"
```

```
20 LOCATE 9,20:PRINT "10 PITCH EMPHASIZED  
2"
```

```
21 LOCATE 11,20:PRINT "10 PITCH TINYPRINT  
3"
```

```
22 LOCATE 13,20:PRINT "CONDENSED EXPANDED  
4"
```

```
23 LOCATE 15,20:PRINT "ITALICS EXPANDED  
5"
```

```
24 LOCATE 17,20:PRINT "EXPANDED DOUBLE-STRIKE  
6"
```

```
+-----+
|  41  |
+-----+
```

```
25 G$=INKEY$:IF G$<"1" OR G$>"6" THEN 25
```

Using the INKEY\$ function, we set G\$ as out
input key. Check that only selections from 1
to 6 are valid. If not, repeat input.

```
26 G=VAL(G$)
```

As we want a numeric expression (ie; 1, 2, 3
etc) to work on, we look at the VALue of G\$ and
place it in 'G'.

```
27 LOCATE 19,25:INPUT "DESIRED LEFT MARGIN ",M!
```

Set the cursor on line 19, character position
29 along the screen and request an INPUT for a
margin on the left side. If you want to start
typing with a margin of 5, then type 5 and
press RETURN.

```
28 ON G GOSUB 31,34,39,42,46,50
```

Here we use the ON variable GOSUB statement.
Depending on the value of 'G', the routine will
execute a subroutine. If 'G=3', then third
expression on the GOSUB will be executed (38).
Having executed a routine, the GOSUB command
always returns to the next statement after that
command. In this case it's a new line number
(as below). The subroutines are to setup
strings for the type of print fonts required.

```
29 GOTO 52
```

Now we move on to set up the screen display and
get the input to be printed on the printer.

```
31 WIDTH "LPT1:",80
```

Set the width of printing to 80 columns.

```
32 LPRINT CLEANUP$ + ITALIC$ + DARK$
```

Printout in 10 pitched ITALICS form.

```
33 RETURN
```

This a subroutine, so RETURN to the caller.

```
35 WIDTH "LPT1:",80
```

Set the column width to 80.

```
36 LPRINT CLEANUP$ + DARK$
```

Printout in 10 pitched EMPHASIZED form.

```
37 RETURN
```

This is a subroutine, so RETURN to the caller.

```
39 LPRINT CLEANUP$ + TINYLINE$
```

Printout in 10 pitch TINYPRINT.

```
40 RETURN
```

Return to caller.

```
42 WIDTH "LPT1:",80
```

Set column width to 80.

```
43 LPRINT CLEANUP$ + CHR$(27) CHR$(87) +
CHR$(1) + EXPAND.SMALL$ + DARK$
```

Printout in CONDENSED and EXPANDED form.

```
44 RETURN
```

Return to caller.

```
46 WIDTH "LPT1:",80
```

Set to 80 columns.

```
47 LPRINT CLEANUP$ + CHR$(27) CHR$(87) +
CHR$(1) + ITALIC.EXP.CON$ + DARK$
```

Printout in ITALICS + EXPANDED + CONDENSED +
EMPHASIZED form.

```
48 RETURN
```

Return to caller.

```
50 LPRINT CLEANUP$ + CHR$(27) CHR$(87) +
CHR$(1) + START.DSTRIKE$
```

Printout in EXPANDED and DOUBLE STRIKE form.

```
51 RETURN
```

Return to caller.

Below we have the screen layout and the input
routine for the text to be printed.

```
52 CLS:LOCATE 25,20
```

Clear the screen and place the cursor in line
25, character position 20.

```
55 PRINT "WHEN FINISHED TYPING, USE
CTRL/BREAK";
```

```
57 LOCATE 1,10
```

Place cursor in line 1, character position 10
and print the three

following lines:-

```

58 PRINT"1234567890123456789012345678901234
5678901234567890123456789012345678901234567
89"

```

```

59 PRINT "0-----1-----2-----3-----
-----4-----5-----6-----7-----
---8";

```

```

60 PRINT

```

```

61 FOR I = 1 TO 55:PRINT TAB( M! - 2)I;

```

Now we setup a loop to run for 55 lines. We print a TAB(Margin-2) with the line count 'I'. Remember you were asked what margin you required.

```

62 LINE INPUT A$(I)

```

Now we use the LINE INPUT command. This allows you to type in along a line as specified by locate or the last print command. As the loop starts with 1, the first line is A\$(1).

Once the text has been entered and ENTER pressed we do the following:-

```

63 LPRINT TAB( M!);A$(I):NEXT

```

Print to the printer at the margin given, our text in A\$(I).

Then the NEXT statement tells the FOR to increment by 1 and repeat the process until 55 times has been reached. Then we process the next line.

```
64 CLS:LOCATE 10,20:PRINT "END OF THE PAGE"
```

Clear the screen. Place the cursor on line 10 at character position 20 and print the text.

```
65 FOR I = 1 TO 6000:NEXT:RUN
```

This is a delay loop to 6000. Then the program re-runs.

Line 67, 68 and 69 are for ERROR trapping. This is called upon when an error occurred while trying to print to the printer. This is set in line 1. The Statement is ON ERROR GOTO 67.

```
67 IF ERR = 24 THEN 70 ELSE 68
```

If the ERR code was 24 (Device timeout) then goto line 70 which tells us there has been an error, otherwise goto to line 68.

```
68 IF ERR = 25 THEN 70 ELSE 69
```

If the ERR code was 25 (Device fault) then goto line 70 and so on.

```
69 ON ERROR GOTO 0
```

As we have not protected for all errors, the ON ERROR GOTO 0, allows the routine to continue at the point from where the error occurred. In

this case, the routine will wait until you sort things out and press a key.

```
70 BEEP:BEEP:LOCATE 25,1:PRINT SPC(79):LOCATE 25,20
```

Sound twice a warning note. Put the cursor on line 25 and print 79 spaces to clear that line.

Then re-locate the cursor on the same line but a character position 20 and print the text below.

```
71 PRINT "PRINTER ERROR - RESET and PRESS A  
KEY"
```

```
72 IF INKEY$ = "" THEN 72
```

Check if a key has been pressed, if not, keep looking until one has.

```
73 RESUME
```

When using error trapping, a RESUME must be used. Therefore, we resume to where the fault occurred in a line number.

```
74 END
```

The end of the program.

Reading and Saving data files

To read a file you start with the OPEN
"drive:\path\filename" FOR INPUT AS #1. The
drive and path are optional.
Then to INPUT #1,A\$ or the string/variable or
combination you saved in the first place.

Let us look at the small routine below.

```
1 A$="This is a test line that we want to save"
```

The A\$ has been assigned some text that we want
to deal with.

```
2 OPEN "TEST.FIL" FOR OUTPUT AS #1
```

We open a file name "TEST.FIL" to save to.

```
3 PRINT #1,A$:CLOSE
```

We print the string A\$ to that file and then
close it.

To read the file back again you have to use the
following:-

```
OPEN "TEST.FIL" FOR INPUT AS #1
```

Now to read in what we have just saved.

```
4 OPEN "TEST.FIL" FOR INPUT AS #1
```

```
5 INPUT #1,A$:PRINT A$:CLOSE
```

What happens when we have a file that has a
total number of accounts which could change
from time to time. How do we load it without
having to know how many records there are in
that file?

There are many ways to do this. A simple suggestion would be to place names in a string array. Remember that if you exceed 9 in the brackets then you will have to use the DIM A\$() expression.

```
1 A$(1)="peter":A$(2)="paul":A$(3)="MARY":B=3
```

Let us assume that we have typed in three names and each time we have typed in a name, the 'B' variable has increased by one.

The result is 3.

You could have line one say INPUT A\$(1):INPUT A\$(2):INPUT A\$(3) and type in your own names.

```
2 OPEN "TEST2.FIL" FOR OUTPUT AS #1
```

```
3 PRINT #1,B:FOR I=1 TO B:PRINT
#1,A$(I):NEXT:CLOSE
```

First we save the total amount of records in that file. Then we do a loop to that value (as in B). Print each record and close.

Now to READ that file and display the contents.

```
5 OPEN "TEST2.FIL" FOR INPUT AS #1
```

```
6 INPUT #1,B:FOR I=1 TO B:INPUT A$(I):PRINT
A$(I):NEXT
```

```
7 CLOSE
```

What happens if there are no records or the actual file is not present ? Then we have to use:-

ON ERROR and RESUME

For this we should use the ON ERROR GOSUB statement at the beginning of the routine. Place an error routine at the end of the program which will tell it what to do.

```
1 ON ERROR GOSUB 6000
```

If an error occurs then gosub to the error handling line 6000.

```
2 OPEN "TEST3.FIL" FOR INPUT AS #1
```

The TEST3.FIL in this case does not exist, as we have not written a program to create it.

```
3 INPUT #1,B:FOR I=1 TO B:INPUT #1,A$(I):PRINT
  A$(I):NEXT
```

```
4 CLOSE
```

```
5 END
```

```
6000 RESUME 4
```

As the file does not exist, an error occurred. We told the program to direct the error to line 6000. In this case we RESUMED to close the file and end. We could easily have checked for a specified error code and then printed a message.

For example:-

```
6000 IF ERR = 53 THEN LOCATE 23,1:PRINT"FILE
NOT FOUND"
```

You could add other commands to the line like 'press a key' etc and then RESUME 4

Remember that a RESUME must return to the

routine that caused the error and not to any other part of the program. Serious problems will occur.

For the error codes see error messages on page 196.

The ON ERROR can be used to debug a program. To see what error had occurred just place in line 6000 PRINT ERR:END and then look up the codes. On the other hand, if you also want to see where the error occurred in a line then do the following:-

```
6000 PRINT ERL;" CODE: ";ERR:END
```

APPEND

To add items to a file, we use the APPEND command.

```
1 INPUT " ENTER TEXT ";A$
2 OPEN "TEST.TXT" FOR APPEND AS #1
3 WRITE #1,A$
4 CLOSE:GOTO 1
```

When finished, use CTRL-Break to stop this when entering a name.

Run this routine and type the following:-

```
THIS IS LINE ONE (press ENTER)
THIS IS LINE TWO (press ENTER)
```

Once you have finished, have a look at the TEST.TXT file and see the result. Type SHELL (without a line number) and you will be placed in DOS. NB!! Quick Basic users will have to select from the FILE menu:
D or S depending on the version they have.

Type TYPE TEST.TXT and press ENTER.

You should see the two lines you typed in. Now

type EXIT (press ENTER), and you will be returned to your Basic Interpreter. Run the routine again. Now type THIS IS LINE THREE (press ENTER)

Stop the routine with CTRL-Break and SHELL to DOS.

Look again at the TEST.TXT file. The new line that you have typed in will be APPENDED to the last line.

If you use this routine for displaying documentation, Please place in that file that you used this routine and from where it came from. This routine is Copyright, All rights reserved. You can get the updated version from Shareware under the name FL2COM45. This includes a TSR version with remove.

WRAP1.BAS

Wrap1 converts a Documentation file that has over 78 characters per line and places larger lines onto the next line, indenting it automatically to the same start as the previous line. The maximum line length is 230 characters while Graphic BOXES (over 78 characters) will have to be manually changed BEFORE running this routine.

```
1 CLS : CLOSE : CLEAR : LOCATE 1,1:
PRINT "Part of the REM series: WRAP 1.0(C)All
Rights Reserved. C.P. White 1987-90"
```

First clear the screen. Close any files or devices that were left open. Clear out memory, removes any variables/strings that were assigned etc. Place the cursor on the top line (left) starting a position one. Now print the copyright notice.

```
2 PRINT:PRINT" Convert ASCII files to less than 78
Characters per line.":PRINT
```

Print two lines some text and then print a blank line.

```
3 ON ERROR GOTO 11
```

If an ERROR occurs during processing, GOTO line 11 which handles this.

```
4 INPUT " Name + Ext (ENTER=Quit) :", INFILE$:
IF INFILE$="" THEN 10
```

Use the standard INPUT command to prompt for an entry, and print some text on the same line. The string used here is INFILE\$ for the result of the input. If the result is nothing then goto line ten.

```
5 PY = 1: OPEN INFILE$ FOR INPUT AS #1
```

The variable PY is a switch to indicate what part of the program we are using. If an ERROR occurs, the error routine will be able to deal with it. We OPEN the file that was entered in line 4, as an INPUT file with a file number of one.

```
6 INPUT #1, A$
```

Read in from that file to insure that it really is there. If the file does not exist, an ERROR will occur. As soon as this happens, the routine will jump (GOTO) the routine that deals with this (starting from line 11).

```
7 CLOSE : PY = 0:OLDFILE$=INFILE$
```

If all is well and no error occurs, then we know that the file exists. So, the file is closed. The variable PY is switched OFF as we no longer need it for error checking. Now, we know that INFILE\$ contains the name of the file we want to convert. As the original file is not to be touched, we assign it to a string called OLDFILE\$. This can be printed onto the screen later as a display.

```
8 LOCATE ,,1:PRINT:PRINT " Convert Document (Y/N)?
:";
```

We switch the cursor ON by using the third parameter of the LOCATE without altering the cursor position. A blank line is printed and then some text. The semicolon on the end will place the next PRINTED text on the same line.

```
9 GOSUB 44:PRINT RW$:LOCATE ,,0:IF RW$="YES" THEN 14
```

The routine GOSUB's line 44 which has the entry for Y or N. The response is printed (remember, printed after the - semicolon in line 8). The cursor is

switched OFF. If the input from the keyboard was Y (YES for subroutine), then goto line 14 for another prompt and input by user. If this was not the case.

```

10 PRINT:PRINT" No action taken !!    Terminate
Requested.":END

```

Print a blank line and then print some text and finish.

```

11 IF PY = 1 AND ERR = 53 THEN LOCATE 12, 24:
PRINT INFILE$; " NOT PRESENT..TRY AGAIN..": FOR U =
1 TO 15: BEEP: NEXT: RUN

```

Lines 11 to 13 handles ERRORS that we want to protect from. Line 11 compares the PY switch. If it is 1 then it is for the 'FILE NOT PRESENT' error. The ERR=53 denotes the error type we are looking for. So, if both conditions are TRUE, then place the cursor on line 12 at position 24. Print the File Name followed by our comment. Do a loop 15 times and BLEEP each time the loop is incremented by one. The NEXT is required to do the loop. Once completed, RUN the program again.

```

12 IF FLICK=1 AND ERR=>52 THEN RESUME 39

```

The FLICK switch is used to check for an error while converting the file. Once again, if both conditions are TRUE then the routine will RESUME to line 39.

```

13 LOCATE 12, 30: PRINT "DEVICE ERROR...TRY AGAIN":
FOR U = 1 TO 15: BEEP: NEXT: RUN

```

Line 13 will cover any error that we forgot to cover. It will place the cursor on line 12 and place it in position 30. Print our message, bleep 15 times and re-run the program.

+-----+
| 55 |
+-----+

```
14 LOCATE ,,1:PRINT:PRINT" Remove Empty lines (Y/N) :  
";GOSUB 44:PRINT RW$:LOCATE ,,0
```

Give a prompt to the User, print response.

```
15 IF RW$="YES" THEN NOSPACES=1
```

If the response is Y (YES), we switch on a variable called NOSPACES to 1. In another words, this tells us that a part of the program is to used to remove blank lines form the incomming document.

```
16 FLICK=1
```

This variable is used to switch on the error checking for this part of the routine. If an error occurs, then the error handling routine (or lines) will know what to do.

```
17 PRINT:COLOR 23:PRINT".....Checking and  
converting !!!" :COLOR 7,0
```

Print a blank line, set the cursor to flash.
Print the text and reset the cursor to normal.

```
18 OPEN INFILE$ FOR INPUT AS #6:ARF=LEN(INFILE$):  
FILE$=LEFT$(INFILE$,ARF-4)+".NEW"
```

The documentation file is opened as file 6.
A varaible ARF is set the the total length of that file name (with its extension). We now want to change the extension to .NEW as we will not be touching the orignal file. FILE\$ is set by looking at the complete name and subtracting four. The 4 is the extension plus the full-stop.

```
19 OPEN FILE$ FOR OUTPUT AS #7:INFILE$=FILE$
```

The new named file is opend to output to the disk as file 7. The orignal file name is now made into this new name.

```
20 LINE INPUT#6,A$
```

Load in a LINE from the file including carriage returns and line feeds. LINE INPUT# will do this for you. It is loaded into the string A\$.

```
21 IF VAL(A$)>=32 THEN 24
```

If the value of A\$ is greater than or equals to 32 (space) then we jump (goto) to line 24

```
22 IF NOSPACES=1 AND LEN(A$)=4 THEN 20
```

If a request to remove spaces is active then check to see that the length is 4 (should be carriage return plus line- feed). If both conditions are TRUE then jump (goto) to 20

```
23 IF NOSPACES=1 AND LEN(A$)=0 THEN 20
```

Once again, if request for no spaces is active check for A\$ being nothing. If both conditions are TRUE then 20

```
24 IF LEN(A$)<=78 THEN GOSUB 36:GOTO 20
```

If the length of A\$ is below or equals to 78 then gosub the routine to save it. Now load in another line.

```
25 A=LEN(A$):B$=RIGHT$(A$,A-78):B=LEN(B$):
A$=LEFT$(A$,A-B):IF B>78 THEN E$=RIGHT$(B$,B-78):
E=LEN(E$):B$=LEFT$(B$,B-E)
```

Set variable A to length of A\$. Make B\$ (string) hold all the characters from the 79th position in A\$. Set variable B to the length of B\$: Now set A\$ to hold 78 characters. If the length of B is greater than 78 then we have to make a new variable and string to hold the result in. So E\$ holds anything over 78 characters in B\$.

```
26 FOR I=1 TO LEN(A$):IF MID$(A$,I,1)<>" " THEN 29
```

We now do a loop to check the first line (A\$) to see what position the first character is in. This routine will auto indent any other lines we may have.

```
27 ADD$=ADD$+" "
```

This String will be placed in front of a line before it is saved. Therefor automatically indenting it.

```
28 NEXT
```

This is need to complete the LOOP.

```
29 IF RIGHT$(A$,1)=" " OR LEFT$(B$,1)=" " AND
LEFT$(E$,1)=" " THEN C$=B$:B$="":GOTO 33
```

Now we check each string (A\$, B\$ and E\$) to ensure that the word on the end of A\$ is complete. That the word in front of B\$ is complete and that E\$ is Complete. If the conditions are TRUE, we miss the next lines. If, on the other hand, the conditions are NOT TRUE, then the next lines apply.

```
30 C=C+1:IF MID$(A$,79-C,1)=" " THEN C$=RIGHT$(A$,
C-1)+B$:A$=LEFT$(A$,78-C):GOTO 33
```

The variable C is incremented by. If MID\$ of A\$ equals a space, then from that position we take the word and place at the front of C\$ adding B\$ to it. A\$ now becomes shorter, as a word was removed.

```
31 IF C=LEN(A$) THEN 33
```

Here, we check that C does not become greater that the length of A\$, or we would get an error.

```
32 GOTO 30
```

We goto 30 to make C increment. Faster than a loop.

```
33 B$="":D$=""
```

Just clear these strings.

```
34 IF E$<>"" THEN PRINT#7,A$:A$=ADD$+C$+E$:E$="":
C=0:D=0:C$="":ADD$="":GOTO 25
```

First we check to see if we have a third line. If we do, then we save A\$. Now make A\$ have the indent plus the last two lines. It now is over 78 characters. We clear the strings and variable we have used and goto the routine that will sort A\$ out.

```
35 GOSUB 36:C=0:D=0:GOTO 20
```

If line 34's condition was not TRUE then we goto a subroutine. Once that has done its work, it returns back and we clear out variables C and D and then back to load in another line.

```
36 PRINT#7,A$
```

Save line one (A\$)

```
37 IF C$<>"" THEN C$=ADD$+C$:PRINT#7,C$:C$="":A$="":
ADD$=""
```

If line two is not empty make C\$ have the indent at the beginning of itself. Save the text. Clear the variables used.

```
38 RETURN
```

As this is a Subroutine, we RETURN to the calling line.

```
39 CLOSE#6:CLOSE#7
```

When no more information can be loaded from the original named file, an error will occur. The error routine will come to this line so that we can carry on with the main program. Both files will be closed.

```
40 LOCATE 11,1:PRINT" New File name is :";
INFILE$. This is converted !!!"
```

Place cursor to line 11 at position 1 and print the text followed by the new name file.

```
41 PRINT:PRINT" Original :";OLDFILE$;": Not changed !"
```

Print a blank line and then print the text/oldfile name and the rest of the text.

```
42 LOCATE ,,1:PRINT:PRINT" Convert Another ASCII file
? (Y/N)";: GOSUB 44:LOCATE ,,0:IF RW$="YES" THEN RUN
```

Switch on the cursor without altering the cursor position. Print a blank line then print some text with a semi-colon. Get response from the keyboard. Switch off cursor. If response was Y (YES), RUN the program again. If not.....

```
43 PRINT:PRINT:PRINT" Finished...Many thanks...":
PRINT:PRINT" REM stands for: Realtime Employment of
Memory.":END
```

Print two blank lines, print some text. Print blank line, print some text and then END.

```
44 RW$=INKEY$
```

This is the subroutine for Y and N responses. We set RW\$ to get a key.

+-----+
| 60 |
+-----+

45 IF RW\$="n" OR RW\$="N" THEN RW\$="NO":RETURN

If response is either upper/lower case N then we
make RW\$ say NO and RETURN to the caller line.

46 IF RW\$="y" OR RW\$="Y" THEN RW\$="YES":RETURN

If response is either upper/lower cas Y then we
make RW\$ say YES and RETURN to the caller.

47 GOTO 44

Continue to get response.

End of topic.

+-----+
 | Communications |
 +-----+

Alphabetical Index	216
Functional Index	220
Protect your Copyright Material	230
ANSI Colour Codes	231
ASCII Character Codes	234
Opening the Channel	64
Monitoring the Channel	70
Input from Caller	75
Check for Special Restricted Characters	78
LOCATE(ing) Screen Characters	80
Dialing out	82
Receiving/Sending Text:	85
MO-4.BAS	87
COMMAND Line Switches [Compilers]	95,212
TIME Delay [For Action]	97
Changing Baud Rates	99
Bulletin Board Files	101
Caller Message System	102
Make a Password Program	110
Quick Basic Compiler Vs 3.0 FIX	117

Opening a Channel

Although the word "Communications" puts fear into most peoples harts, believe me, it is not that difficult to do. I will not go into any technical jargon, all I will do, is to let you see what is happening as you go along with me.

You already know how to open files to your disk/ette. It is almost as easy, using the same syntax [in general]. For those of you who use Microsoft Quick Basic Compiler 3.0, remember to link the GWCOM.OBJ code to the final .EXE file. Without this, these routines will not operate correctly in you finished program. Standard Basic Users need not worry about this problem.

First we need to open a channel to the Modem. We tell it that we need to use it as a device to communicate with [and through] the telephone line.

Which port have you plugged the Modem into ? Is it the Serial Port [25 pin] at the back of the computer, or is it inside the computer [plugged into one of the slots] ? If outside, it is classified as COM [Communications Port] device 1, while inside the machine it is COM 2 or Higher.

Please note..The Serial port actually has two Com Port addresses. They are COM1 and COM3. The internal fitting card will have addresses COM2, COM4 and so on.

OPEN....We start to OPEN the device we want.
"COM2:..The Port we need that contains that Device.
If your Modem is outside the Computer,
change to COM1

,1200...The Baud Rate at which your Modem will run.
[Please read your Modem Manual].
,N.....No parity.
,8.....We are using 8 bits per character.
,1.....One stop bit.
,RS.....Suppress Request to Send signal.
,CS.....Wait for Clear To Send signal.
,DS.....Wait for Data Set Ready signal.

We also want to tell the device that we want to INPUT
from it, as well as OUTPUTing [Reading and Writing]
with a file [device] number.

The command for this is 'RANDOM AS #' and we will use
the file number as 1.

So, the first command will be:

```
10 CLS:OPEN "COM2:1200,N,8,1,RS,CS,DS" FOR RANDOM AS  
#1
```

You should use a string expression to assign what
Port you are using, therefore TYPE IN ON ONE LINE

```
10 CLS:A$="COM2:":OPEN A$+"1200,N,8,1,RS,CS,DS" FOR  
RANDOM AS #1
```

Please see your Modem manual for the correct required
speed.

If you run this now, you will have an Error of the
following:

" Device Time out "

That is to be expected, as we have not covered the
program for various checks.

What we now want to do, is to monitor the signal and
see if:

a). The Carrier has a signal. If not, continue to
monitor.

- b). If Carrier has a signal, we need to have a reaction.
- c). Then, either hang up and quit or hang up and reset the Modem to re-answer.

Now we need to set various parameters to look at, so that we can determine what action to take.

```

20 IF A$="COM1" THEN LSB=&H3F8:MSB=&H3F9:LCR=&H3FB:
MCR=&H3FC:LSR=&H3FD:MSR=&H3FE ELSE LSB=&H2F8:MSB=
&H2F9:LCR=&H2FB:MCR=&H2FC:LSR=&H2FD:MSR=&H2FE

```

Notice here we use the IF THEN ELSE statements.
 IF COM Port 1 THEN these variables are worth this
 ELSE they are worth that.

To read the Port we must use the INP instruction.
 Therefore:

```

30 IF (INP(MSR) AND &H40)=0 THEN 30

```

Here we look at the Port to check to see if the value is 0. 0 = No signal received at present.

As soon as the value is changed, we have a signal and the program will continue to execute the next line. We now have to decide what to do. Let us Print on the screen the message RINGING.

```

40 LOCATE 12,38:COLOR 16,7,0:PRINT " RINGING ":
COLOR 7,0,0

```

Locate the cursor on line 12 along the line at Position 38. Switch on flashing and "Inverted". Print RINGING and reset the colour to normal.

```

70 FOR I=1 TO 5000:NEXT

```

We will have a delay. This is so that we can see and here whats going on.

100 CLOSE#1:GOTO 10

The Modem is switched off and the the cycle is restarted. This only demonstrates the ability of the modem knowing when the telephone is ringing. What we have not done in this example is to answer the telephone and then hang up, resetting the modem to receive.

We will do that now.

Add these lines after line 70

71 LOCATE 12,38:PRINT"ANSWERING"

72 A\$="ATQOX1V1A"+CHR\$(13)+CHR\$(10)

A\$ is set to the command to answer the telephone line.

73 PRINT#1,A\$

Tell the modem to answer the line.

74 FOR I=1 TO 5000:NEXT

A delay loop just for our visual pleasure.

75 LOCATE 12,38:PRINT"HANGING UP"

76 PRINT#1,"ATZ"+CHR\$(13)+CHR\$(10)

The Modem is instructed to Hang up Receiver.

77 FOR I=1 to 5000:NEXT

Another delay for our pleasure.

78 CLOSE#1:GOTO 10

The Modem is closed and we restart the routine again.

Save this under the title of "MO-1.BAS"

The delay loops are just to let you see what is happening, otherwise this routine will wizz along without you seeing anything.
To test it out, run this routine and pick up your telephone receiver a couple of times. It depends on the type of apparatus you have.
It may take a couple of "Hanging ups" to enable it to work. Alternately, you could ask someone to telephone you. If you do this, Please RUN the routine when the telephone rings.

This little routine has almost done everything for you.

I did say almost.

What is left ? The truth is, one should always monitor all signals using a Timer. This is needed to check on the state of the Port. You need to see if it is ready to transmit or receive, otherwise problems will occur. What is also needed, is to see if a computer is on the other side of the line. If you do not do this, you would be sending a signal down the line..regardless.

To do this, we either wait for a signal from the other computer and see it matches our speed, or we could send a "ARE YOU THERE" signal and wait for the response within a given time scale.

The next routine will answer the line, see if there is another machine on the other side. It will hang up and reset itself. If the line has been answered, it will print the message "ITS ALIVE" and hang up.

This routine now uses the TIME\$ to set various time limits for watching actions taken. Various checks are made to ensure that all is well.

If you are using Basic, change the loops from 9000 to around 3000. 9000 is set for Compilers.

I am not going into to much detail as I belive that it is far better that you get up and running as fast as possible. Most routines can be used as they stand or be altered between each other to get the result you desire. You will have to renumber where necessary.

Monitoring the channel:

```
1 CLS:CLEAR:CLOSE#1:DEFINT A-Z
```

Clear the screen and any preset data.
State we want to use Integer variables.

```
2 DEF SEG
```

We reset to normal.

```
7 DEF FNTI!=CSNG(FIX((VAL(MID$(TIME$,1,2))*60*60)+
(VAL(MID$(TIME$,4,2))*60)+(VAL(MID$(TIME$,7,2))*1)))
```

FNTI! is set using the clock. We need a single-precision figure to work on, so CSNG command is used. Also required is an Integer portion of the expression of ((VAL(MID\$(TIME\$,1,2))*60*60), so FIX is used.

You can add PRINT FNTI! to see the result.
This is necessary for us to set our sync times if the modem is to answer the line correctly.

```
9 IF INP(MSR)<128 THEN OUT MCR,&H4:GOSUB 6020
:OUT MCR,&H0
```

First check to see if the signal address is less than 128 and then it must be reset to 0

```
10 A$="COM2:":OPEN A$+"1200,N,8,1,RS,CS,DS"
FOR RANDOM AS #1
```

The COM Port is 2. Change this if you are using COM1. The Baud rate here is 1200. Once again, change this if your setting is different.

```
20 IF A$="COM1" THEN LSB=&H3F8:MSB=&H3F9:LCR=&H3FB
MCR=&H3FC:LSR=&H3FD:MSR=&H3FE ELSE LSB=&H2F8:MSB=
&H2F9:LCR=&H2FB:MCR=&H2FC:LSR=&H2FD:MSR=&H2FE
```

+-----+
| 71 |
+-----+

25 IF (INP(MSR) AND &H40)=0 THEN 25

If no signal (0) then NOT RINGING. Keep monitoring the line.

30 LOCATE 12,38:COLOR 16,7,0:PRINT " RINGING "
:COLOR 7,0,0

Got a signal, so we print 'RINGING'.

35 FOR I=1 TO 9000:NEXT

A delay for our benefit.

40 GOSUB 6040

Just to check and keep us in time.

45 LOCATE 12,38:PRINT"ANSWERING"

Print ANSWERING for our benefit.

46 A\$="ATQOX1V1A"+CHR\$(13)+CHR\$(10)

Lift RECEIVER COMMAND

47 PRINT#1,A\$

Do it !

50 TCC!=FNTI!+30

We need another variable to set to a delay to see if anyone is out there.

60 IF INP(MSR)<128 AND FNTI!<TCC! THEN 60 ELSE
IF INP(MSR)<128 THEN 76 ELSE GOSUB 6000

We check for anyone out there, if not we hang up
ELSE there is and we get into time delay.

+-----+
| 72 |
+-----+

61 IF LOC(1)<>0 THEN DF\$=DF\$+INPUT\$(LOC(1),1) ELSE 65

We check to see if our device is not in error and we get the response from the other side into string DF\$.

63 IF INSTR(DF\$,"CONNECT") THEN 120

If the response is CONNECT then we goto the line that will send a message.

65 IF FNTI!>=TCC! THEN 76 ELSE 60

As long as we are less than our requested sync timer we continue or we keep checking until something happens. This could be:

- a). The device at the other end is not responding.
- b). We just do not have a signal within the time.
- c). We are just out of sync at present.

76 LOCATE 12,38:PRINT"HANGING UP"

77 FOR I=1 TO 9000:NEXT

A delay for our visual pleasure.

78 PRINT#1,"ATZ"+CHR\$(13)+CHR\$(10)

Tell the Modem to HANG UP.

100 CLOSE

110 RUN

120 A\$="Thank you for calling....Hanging up now !":
PRINT#1,A\$:PRINT A\$:GOTO 76

A\$ is setup with the message that is needed to be sent. It is printed onto the callers screen as well

as our screen. Then the Modem is hung up and reset.

The next section [below],/is where we check for timing. You can always use this routine for any type of Modem communications package that you write.

```
6000 DE!=FNTI!+1
```

We add one to the timer [sync] compare variable.
This is needed to compare against the timer [FNTI!].

```
6010 GOTO 6030
```

```
6020 DE!=FNTI!+3
```

We add three to the timer [sync] compare variable
This will be used at a later stage in the book.

```
6030 IF FNTI!<DE! AND DE!<2400! THEN 6030 ELSE RETURN
```

If the timer is less than the compare variable
and the compare variable is less than 2400 then
we keep going until we are in sync. You should
actually change 2400! to 86400!, as this [although
slower] is the correct setting for ensuring
a correct adjustment for delay [sync] times.

```
6040 WHILE (INP(MSR) AND &H40)
```

Why use this line ?. Print it to the screen an see.

```
6050 WEND
```

```
6100 RETURN
```

Back to our caller routine.

So far you have managed to:

- a). Monitor the line.
- b). Set a time sequence to manage the call.

+-----+
| 74 |
+-----+

- c). Send a response to the caller.
- d). Hang up the line correctly.

Save this as MO-2.BAS

INPUT from the Caller:

What you now want to do is to get a response from the caller in the form of some text.

The caller must enter His or Her name followed by a Carriage Return. In this example there will be no checks for invalid characters, or for any ERRORS that may occur from within the program.

LINE INPUT# 1,B\$ will do the trick. This will keep getting characters until a Carriage Return or 255 characters have been entered. Dimension the string as normal, for those extra long lines that may be entered.

The normal practise for this procedure, is to check for those invalid characters. The ones that you do not want to get through. You can scan the string and shift each character to the left eliminating them. The final resulting string will be as sent, but safe.

Another method is to assign a string with LINE INPUT.

```
120 B$=""
```

When entering the input routine, always clear out the string variable before continuing.

```
122 TCC!=FNTI!+180
```

Set time sync for entering characters.

```
124 LOCATE ,,1
```

Switch the cursor on.

```
126 IF EOF(1) THEN 76
```

Always check if caller has terminated output.

```
128 B$=INPUT$(LOC(1),3)
```

Get the response within three minutes.

```
130 GOSUB 6000
```

Start time sync.

```
132 PRINT#1,B$;
```

Print the character to callers console along the line.

```
134 GOTO 126
```

Back to check if the caller has hung up.

You can use another method, like the following:

```
120 B$=""
```

```
122 IF (INP(MSR) AND &H40)=0 THEN 76
```

Check to see if line is dead. If so, restart.

```
124 IF EOF(1) THEN 128
```

Check if End Of File.

```
126 IF LOC(1)<>0 THEN B$=INPUT$(1,#1)
```

If the file is ok, set B\$ to input characters.

```
128 R$=INKEY$
```

```
130 IF R$<>"" THEN B$=R$
```

```
132 IF B$="" THEN 242
```

If nothing has been entered we check for time.

+-----+
| 77 |
+-----+

134 TCC!=FNTI!+180

Set the timer sync.

200 C\$=C\$+B\$

The string c\$ is made up by each character entered by the caller.

240 PRINT#1,C\$;

Print the string C\$ to the callers console.

242 IF FNTI!<TCC! THEN 120

Check for the time delay. If the caller is within the limit, then carry on screening for characters.

244 GOTO 76

Caller was out of time, so we hang up and reset the Modem correctly and rerun the routine.

You will have to experiment a bit with this routine. It really depends on what you require the result to be.

In this example, there has been no checking for characters which require special attention. The next section will deal with that.

Check for Special Restricted characters. -----

Like all programs, testing for restricted characters, or a combination of characters, is normal. No one wants errors to occur while the user is using their program.

Here, you use the same techniques as you would do in any normal program. The resulting String or Variable is checked for those offending characters. In some cases, it is necessary to assist the displayed characters [or line] by moving the cursor to the correct position and altering the String to the correct format.

For Example:

To check for the DEL key and alter the final C\$ [String].

```
136 IF B$=CHR$(8) AND LEN(C$)=0 THEN 242
```

So, IF the input is the DEL key and the final String length of C\$ is Zero THEN the routine will goto the line that compares the elapsed time condition.

```
138 C=LEN(C$):IF B$=CHR$(8) THEN C$=LEFT$(C$,C-1)
```

Set variable C to the length of C\$. IF the input key is the DELeTe key THEN C\$ is subtracted by one character.

What is needed in this routine is to position the cursor on the console.

To continue with the train of thought. You must also look for the Carriage Return and Line feed characters. If you fail to do this, problems may occur when you print to the callers console. As M\$ is assigned with those two characters, you could find extra lines

being printed when they are no required.

```
140 IF B$=CHR$(10) THEN 242
```

```
142 IF B$=CHR$(13) THEN 240
```

When you are altering the String in any way, you must LOCATE the position of the input line on the callers console. If you fail to do this, the display will not be correct. As with any Editor that you use, you will see that the cursor remains on the correct line even if you delete a character, overtype or insert.

Save this routine as MO-3.BAS

Locating screen characters.

On the outset, when you are setting your program out, always assume that you are going to use the full LINE INPUT command. There are instances where you may require a Single letter response, but generally a large String [or combination of strings] will be used.

At the beginning of your program, where you setup the String/Variable tables, set these components.

EXAMPLE:

```
4  CURSOR_ONE$=CHR$(8)+CHR$(32)+CHR$(8)
```

```
5  CURSOR_TWO$=CHR$(29)+CHR$(32)+CHR$(29)
```

Within the routine MO-2.BAS, we must have a small routine that controls the character position.

This routine must also check for the POSITION of the cursor at all times.

```
500 IF POS(0) >1 THEN PRINT CURSOR_ONE$:PRINT#1,
CURSOR_TWO$
```

```
502 RETURN
```

Providing the position of the cursor is greater than 1, THEN we print our cursor line control strings. However, if it is 1 or less, it is ignored.

Now change line 138 to look like this:

```
138 IF B$=CHR$(8) THEN C$=LEFT$(C$,C-1):GOSUB 500:
GOTO 242
```

Whenever you manipulate the string using MID\$, RIGHT\$

```
+-----+  
|  81  |  
+-----+
```

and LEFT\$, always set your cursor position before printing to the Callers console.

You can also have the response from the Caller printed onto your screen. This is done quite simply by placing another string character with the value of the Callers string and PRINT that to your screen. You can use the LOCATE command or the same control commands as used for the CALLER, or just PRINT string name. One thing must be said. Always develop your program without the worry of proper screen layout. This can be done last. Getting the program to work correctly is of prime importance. With Communications it is best to develop a system whereby you can always refer to any section on the program and KNOW what is what.

Dialing out: -----

All that is required is to instruct the Modem to Dial a set of digits. To activate this mode, one will use the ATDP command. AT is for Attention, D for Dial and P is for pulse. If this does not operate on your line, use the command ATDT [T for Tone].

So, to dial out you can either have a small routine that allows you to enter the digits or you could include them with the command string.

Always set the parameters at the beginning of your program with CLS:CLEAR:CLOSE at the very beginning. This assists in preventing any problems later on.

You could set all the Modem commands in the following way:

```
M$=CHR$(13)+CHR$(10):MD$(1)="ATQOX1V1A"+M$:MD$(2)="ATDP":MD$(3)="ATDT":MD$(4)="ATZ" and so on.
```

```

M$ is set for Carriage Return plus Line Feed.
MD$(1) is set for Answer line.
MD$(2) is set for Dial Pulse.
MD$(3) is set for Dial Tone.
MD$(4) is set for Hang up.

```

Lets say the number that you want to dial is 123456 and you want to dial using the Pulse generator.

The instruction given will look like:

```
PRINT#1,"ATDP0123456"
```

Once the instruction is given, you still need to monitor the line.

```
25 IF (INP(MSR) AND &H40)=0) THEN 25
```

What about a). If the line is engaged.
 b). No such number.
 c). Not answered.

By using the subroutine in line 6000, you can give a time limit to the action that is taking place. Therefore, you would monitor the line with GOSUB 6000 instruction to check for time limit. Then continue to monitor. You do not have to do this, but its a good idea, just for safety.

Once the line has been answered at the other end, you need not worry for the possible problem of

"Answered But Not a Computer"

as line 63 in the routine MO-2.BAS is only looking for the answering computers "CONNECT" signal. If this is not received, it automatically disconnects and reruns itself.

You can also say, if the line was answered, but within a given time a "CONNECT" signal was not received, then it must be A Human. Print "HUMAN...Please Answer !" to your console, BEEP until a key has been pressed to indicate that the hand receiver has been lifted. Then reset and run or wait for a further key press to rerun.

If you use this method, it is best to get the Modem to reset itself [ATZ] but not to rerun until you have finished the call. The reason for this is. If the program reruns before you have finished your call, it could think it is answering the line when you hang up the receiver. On most telephones, on replacing the receiver, a CLICK is produced, which could trigger off the program in thinking it has an incoming call.

We have now dealt with dialing a number and checking for a human caller. But what has not been covered here, is what do we do when a computer is on the other side of the line. With the next section, I will try

to show you how to respond to some aspects of getting
transmitted text over the line.

Receiving/Sending Text:

When your program is the "CALLER", it needs to be able to receive information from the other computer. Remember, the responding Computer will always send some text to the console of the CALLING Computer. This would take the form of a). Welcome message and various information lines. b). Password and selection menus.

There are a few points that must be taken into consideration. First of all, is the responding device going to wait for ASCII text to be sent to it ? And two, will it send some ASCII text and expect a response via the keyboard from the CALLING Computer ? Expect all situations ! Therefore, within any program you must be able to Receive & Send Data down the line.

LINE INPUT# is quite straight forward, and is used in the normal way to receive any ASCII data. But you must also know when it is time for keyboard entry at the CALLING machines end.

You could examine the incoming data for the ASCII character 1A [->], but this is not always the case. Some text editors do not place 1A onto the end of a ASCII file. Or you could, once again use a TIMER variable to compare the elapse of a predetermined time to indicate TIME TO USE MY KEYBOARD. The best method is to incorporate both your keyboard and incoming data as part of one routine with WHILE EOF() command. In this way, you have control from within your program.

It is important that you have keyboard input at any time, as you may want to Quit from a procedure, answer a question posed from the receiving computer and so on. Also required, is to position the cursor on your screen as the text is being displayed, as well as having the ability to display correctly as you type.

Using the EOF(file number you opened the Modem with),
you Input from the modem. If there is no incoming
signal, then you GOTO or GOSUB to the normal keyboard
input routine which will print [using your subroutine
to LOCATE the cursor on the current line] to the
other Computers console or input request routine.
Remember to always monitor the telephone line, just
incase the carrier goes dead. On the next page is
a sample routine that should assist you to understand
a little bit more.

```

MO-4.BAS          [ Will CALL and TYPE to Carrier.]
-----

```

```

1 CLS:CLEAR:CLOSE:DEFINT A-Z:M$=CHR$(13)+CHR$(10)

```

Clear the screen and any preset data.
 State we want to use Integer variables.
 Set M\$ to Carriage Return and line feed.

```

2 DEF SEG

```

We reset to normal any preset Define Segment commands.

```

3 DEF FNTI!=CSNG(FIX((VAL(MID$(TIME$,1,2))*60*60)+
  (VAL(MID$(TIME$,4,2))*60)+(VAL(MID$(TIME$,7,2))*1)))

```

FNTI! is set using the clock. We need a single -
 precision figure to work on, so CSNG command is used.
 Also required is an integer portion of the expression
 of ((VAL(MID\$(TIME\$,1,2))*60*60), so FIX is used.
 You can add PRINT FNTI! to see what the result is.
 This is necessary for us to set our timer sync
 if the modem is to answer the line correctly.

```

4 IF INP(MSR)<128 THEN OUT MCR,&H4:GOSUB 6020:OUT MCR,
  &H0
5 CLOSE

```

First check to see if the signal address
 is less than 128 and then it must be reset to 0
 and close all devices.

```

10 A$="COM2:":OPEN A$+"1200,N,8,1,RS,CS,DS" FOR RANDOM
  AS #1

```

The COM Port is 2. Change this if you are
 using COM1. The Baud rate here is 1200. Once
 again, change this if your setting is different.

```

20 IF A$="COM1" THEN LSB=&H3F8:MSB=&H3F9:LCR=&H3FB
MCR=&H3FC:LSR=&H3FD:MSR=&H3FE ELSE LSB=&H2F8:MSB=&H2F9
LCR=&H2FB:MCR=&H2FC:LSR=&H2FD:MSR=&H2FE

```

If we are using COM1 Port then we want to use these addresses ELSE it will be COM2 and we use those.

```

22 GOSUB 6040:GOSUB 6020:
PRINT#1,"ATZ~M~~~AT S7=45 S0=0 V1 X1^M~"
23 GOSUB 6040

```

Set the timing. Initialize the Modem.
This command may be different. Please see your Modem manual in SETUP section. If the Modem does not dial, then remove 22/23 and rerun again.

```

24 PRINT#1,"ATDP1234567890"

```

The computer is instructed to Dial this number.
Change this number for your local BBS.

```

25 IF (INP(MSR) AND &H40)=0 THEN 25

```

If no signal (0) then NOT RINGING. Keep monitoring the line.

```

30 LOCATE 12,38:COLOR 16,7,0:PRINT " RINGING ":
COLOR 7,0,0

```

Got a signal, so we print 'RINGING'

```

35 FOR I=1 TO 9000:NEXT

```

A delay for our benefit.

```

40 GOSUB 6040

```

Just to check the timer sync.

```
41 LOCATE 12,38:PRINT"ANSWERING"
```

Print ANSWERING for our benefit.

```
42 A$="ATQOX1V1A"+CHR$(13)+CHR$(10)
```

Lift RECEIVER COMMAND

```
43 PRINT#1,A$
```

Do it !

```
50 TCC!=FNTI!+30
```

We need another variable to set a delay to see if anyone is out there. This is set for Dialing out. If you require to monitor the line for in coming calls, change to suit for your own requirements.

```
60 IF INP(MSR)<128 AND FNTI!<TCC! THEN 60 ELSE IF
INP(MSR)<128 THEN 115 ELSE GOSUB 6000
```

We check for anyone out there, if not we hang up ELSE there is and we get into time sync.

```
61 IF LOC(1)<>0 THEN DF$=DF$+INPUT$(LOC(1),1) ELSE 65
```

We check to see if our device is not in Error and we get the response from the other side into string DF\$.

```
63 IF INSTR(DF$,"CONNECT") THEN 120
```

If the response is CONNECT then we should send a message here or get a response from the other Computer.

```
65 IF FNTI!>=TCC! THEN 76 ELSE 60
```

As long as we are less than our requested sync timer we continue or we keep checking

```

until something happens.  This could be:
    a).  The device at the other end is not
          responding.
    b).  We just do not have a signal within
          the time.
    c).  We are just out of time at present.

76 LOCATE 12,38:PRINT"HANGING UP"

77 FOR I=1 TO 9000:NEXT

A delay for our visual pleasure.

78 PRINT#1,"ATZ"+CHR$(13)+CHR$(10)

Tell our Modem to HANG UP.

100 CLOSE

110 END

115 LOCATE 12,38:PRINT"  ENGAGED  ":GOTO 100

As the line was engaged we say so and
Finish the routine.  Back to DOS.

120 WHILE EOF(1)

While other Computers is sendind, OK, otherwise..

122 A$="":A$=INKEY$

Get a key from this Computers keyboard.

124 IF A$<>" THEN print#1,a$;:goto 138

If we touched a key then it will be sent
to the other Computer and your Computer screen.

126 WEND

```

```
128 A$=INPUT$(1,1)
```

Input a single character at a time from
the other Computer [Modem line].

```
130 IF A$=CHR$(13) THEN 138
```

If it is Carriage Return then we want
to avoid printing extra lines on the screen.
This is used, if you are calling a bulletin
board.

```
135 PRINT A$;
```

Print the character on your screen.
This is for your visual support [always.]

```
138 GOTO 120
```

We continue to monitor the Input.

Below is the timer routine, that supports
you call.

```
6000 DE!=FNTI!+1
```

We add one to the timer [sync] compare variable.
This is needed to compare against the timer [FNTI!].

```
6010 GOTO 6030
```

```
6020 DE!=FNTI!+3
```

We add three to the timer [sync] compare variable.
This will be used at a later stage.

```
6030 IF FNTI!<DE! AND DE!<2400! THEN 6030 ELSE RETURN
```

If the timer is less than the compare variable
and the compare variable is less than 2400 then
we keep going until we are in the same. You could

actually change 2400! to 86400!, as this [although slower] is the correct setting for ensuring a correct adjustment for sync time.

```
6040 WHILE (INP(MSR) AND &H40)
```

As we are still at this memory location.

```
6050 WEND
```

```
6100 RETURN
```

Back to our caller routine.

It must be said, that some Modems [of the latest kind] will automatically initialize themselves when powered up. Therefore lines 22/23 could be omitted, but I do suggest that you always do the initial routine first. I may not work on some Modems.

Line 24, which sets up the command string for the dialling of a number, could be changed to include you own routine to allow the User to type in His or Her own number. This could be a simple F\$=INKEY\$ type routine, or a standard INPUT command. Once again, use the single key [INKEY\$] to build a final string. This will help you to not only check for unwanted keys, but to stop any ERRORS that can occur by using the straight INPUT command.

Do not forget to assign two strings for LOCATING the cursor and a subroutine to do this.

As an example:

```
4 CURSOR_ONE$=CHR$(8)+CHR$(32)+CHR$(8)
```

```
5 CURSOR_TWO$=CHR$(29)+CHR$(32)+CHR$(29)
```

These two would be setup in the earlier stages of the program with the DIM statements etc;

The next two lines are a subroutine for your program.

```
500 IF POS(0) >1 THEN PRINT CURSOR_ONE$:PRINT#1,
CURSOR_TWO$

502 RETURN
```

Providing the position of the cursor is greater than 1, THEN we print our cursor line control strings. However, if it is 1 or less, it is ignored. These lines would locate the characters on the line.

The next line is an example for you to work on. If you build a string using A\$ as the initial input and C\$ as resulting string, and back DEL was pressed.

```
124 IF A$=CHR$(8) THEN C$=LEFT$(C$,C-1):GOSUB 500:
PRINT#1,C$:GOTO 138
```

In this example, C\$ would be one less each time the back DEL key was pressed. Have you noticed anything, yes, no protection if we reach the first character position. See what you can do for this. I would ensure that providing the cursor position is greater than 0 then allow the routine to delete the character. Does that help ?

Remember that it is always a good practise to have a look at various keys and then you can prevent things getting through.

What is missing here, is a line to monitor the Carrier [telephone line] to see if the other end has hung up on you. When this occurs, you should give a message to the screen, wait a moment, then either issue ATZ and a CLOSE command or just CLOSE and rerun the routine. If you decide to have other routines within the program, you could go and do those.

The next thing you can use a GOSUB in line 135 to assist to in the location of the printed characters on the screen, as well as assisting with the layout when printing over the Modem.

Once again, it depends on if you are going to send a message first [As with a program that acts like an Answer Machine], or as a Caller program, which monitors for characters from the other side as well as allowing keystrokes being entered from this side of the program. It is a choice that you have to make on the outset of your design.

The next two routines that are included in this book are for demonstration proposes. A bit of fun, just to let you have a breather.

COMMAND LINE SWITCHING

Another demo program is given on page 212.

Save this as SWITCH-1.BAS

There is a small routine for those of you who like to use COMPILERS. This routine was the one I added into S.A.M Sysop's Answer Machine for Master Gossage, Which I did a complete revised version on his behalf. This routine just delivers some messages according to the switch you entered.

```
0 IF$=COMMAND$:A=LEN(IF$):IF IF$="" THEN 570
```

The command line in DOS is examined for any extra characters. 'A' is set for the length. If no extra characters were found, then the program goes to a display screen. 204 in this case just has a REM statement and then 205 which ENDS the program.

```
4 FOR I=1 TO A
```

```
5 C=ASC(MID$(IF$,I,1))
```

```
6 REM**SORT INTO UPPERCASE
```

```
24 IF C=44 OR C=46 OR C=47 OR C=32 THEN 64
```

```
44 C=ASC(MID$(IF$,I,1)) AND &HDF
```

```
64 F$=F$+CHR$(C)
```

```
65 NEXT
```

```
66 IF$=F$
```

```
84 IF IF$<>"/R AND IF$<>"/E" THEN 144
```

```
104 IF IF$="/R" THEN 564
```

```
124 IF IF$="/E" THEN 564

144 PRINT:PRINT"WRONG SYNTAX !   USE THE FOLLOWING:"

145 PRINT:PRINT"SWITCH-1          TO RUN NORMALLY."

148 PRINT"SWITCH /E          GOTO EDIT SETUP."

149 PRINT"SWITCH /R          TO RUN."

150 END

564 CLS:IF IF$="/E" THEN PRINT" YOU SELECTED /E":END

568 IF IF$="/R" THEN PRINT"YOU SELECTED /R":END

570 PRINT"PROGRAM TO RUN NORMALLY":END
```

You can alter the lines to suit your program.

To get this routine to operate correctly, Compile it as DOS-1.EXE and then run it with one of the switches.

As an example:

```
DOS-1 /R
DOS-1 /E
DOS-1 /
DOS-1
```

Each switch will give you a responding message.

I always recommend that where possible, the use of Switches should be made available for the more advanced User.

TIME DELAY FOR ACTION -----

This could be used as part of the Command line switching. If you selected /R then goto the start of this routine. Alternately, you could use this as a fail safe method, just incase a part of a routine was selected incorrectly.

This Routine [is once again, an addition I placed in S.A.M]. The delay is for 10 seconds for the routine to run, but there is also a keyboard entry system during that time which will allow for actions to take part before that time elapses [Not included.]

Save this as DELAY.BAS

```
7165 DDD$=TIME$
```

```
7166 TIME$="00:00:00"
```

```
7167 LOCATE 8,23
```

```
7168 PRINT"A DEMONSTRATION OF A TIME DELAY AND
SELECTION"
```

```
7169 LOCATE 10,13
```

```
7170 PRINT"(E)....GOTO EDITOR    (Q)...QUIT    (R)...
RUN"
```

```
7184 G$=INKEY$
```

```
7185 IF G$="Q" OR G$="q" THEN TIME$=DDD$:END
```

```
7204 IF G$="E" OR G$="e" THEN TIME$=DDD$:
PRINT"YOU SELECTED EDITOR":END
```

```
7224 LOCATE 12,23
```

```
7225 PRINT"AUTORUN IN 10 SECONDS....";RIGHT$(TIME$,2)
```

```
7226 FFF$=RIGHT$(TIME$,2)
```

```
7227 IF FFF$="10" THEN TIME$=DDD$+FFF$:PRINT"TIME
DONE": END
```

```
7244 IF G$="R" OR G$="r" THEN TIME$=DDD$:PRINT"RUN":
END
```

```
7264 GOTO 7184
```

I've reset the TIMER to 00 in this example and assigned the original time to DDD\$. However, this is not a great way to do things. I suggest that you use a Variable to setup the control. Using switches gives your programs a professional touch while allowing the User various methods of operating your programs.

Changing Baud Rates

[300 to 2400]

It is always nice to change the baud rate from within the current program. There is quite a lot of factors to be taken into consideration. Firstly, if the change is to be done from within the program, are you looking at a configuration file from disk. This would be to verify the callers Name, Password and baud rate. Secondly, if you require to change the baud rate directly from the program and from looking at the signal that was received, then it becomes more complicated.

BE VERY CAREFUL AND TYPE IN EXACTLY WHAT IS LISTED BELOW. THE 'OUT' COMMAND [IF USED INCORRECTLY] CAN CAUSE PROGRAM CRASHES, OR EVEN DAMAGE.

In line 63 of the program MO-4.BAS, the line would look like this.

```

63 IF INSTR(DF$,"CONNECT") THEN RATE=VAL(MID$(DF$
,INSTR(DF$,"CONNECT")+8,4)):GOTO 600

```

Add the following starting with line 600

```

600 GOSUB 800

605 IF RATE=0 OR RATE=300 THEN RATED=&H180:PERSEC=-1:
GOSUB 700 ELSE IF RATE=1200 OR RATE=2400 THEN PERSEC=
-2-(RATE/1200):RATED=48*(PERSEC+5):GOSUB 700 ELSE 61

610 GOTO 120

```

Now add:

```

699 REM TO CHANGE TO CALLERS BAUD RATE

700 A1=INP(LCR):A2=INP(MSB):OUT MSB,0

```

```
+-----+  
| 100 |  
+-----+
```

```
705 OUT LCR,A1 OR 128  
  
710 IF RATED=384 THEN OUT LSB,&H80:OUT MSB,&H1:GOTO  
    730  
  
715 IF RATED=256 THEN OUT LSB,&H0:OUT MSB,&H1:GOTO  
    730  
  
720 IF RATED=96 THEN OUT LSB,&H60:OUT MSB,&H0:GOTO  
    730  
  
725 IF RATED=48 THEN OUT LSB,&H30:OUT MSB,&H0  
  
730 OUT LCR,A1:OUT MSB,A2:RETURN  
  
799 REM CHECK RESPONSE TIME IN 3 MINUTES  
  
800 WHILE NOT EOF(1)  
  
805 DF$=INPUT$(LOC(1),3)  
  
810 WEND  
  
815 RETURN
```

If your Modem has a single baud rate, it would still be advisable to add this routine to your program. After all, if you write programs for other people, they may have the facility in their Modems to have various Baud rates.

It is important to state, that it takes a little practice to actually get the rate to change. Try to printout [on the screen] the RATE and the RATED variables and see what is the actual values are.

This part is only a guide line for future references. As I said, you have to know exactly what addresses are to be looked at and how to alter them. This is not within the scope of this book.

Bulletin Board Files

If your modem has the facility to change its Baud rate, then you could ask the Caller for His or Her Name with a Password. Then look at a special table file which has all the information of the users.

As they enter their name and password, you could check to validity. If either Name or Password was incorrect then you could open a file, using the APPEND command, and save the details of all users that failed to complete their entries. You could make things very difficult for unauthorized entry, by allowing the Names and Passwords [or sentence] to be in mixed cases. This would allow ALPHA and Numeric and even special characters. The combinations of failure increase with every extra possible entry.

I like using multiple entry systems with long lines. for example, as a password:

PRogRaMmer ExtROdiNAiRE

Imagine the problem of trying to crack this beauty. Thats not all, add Numbers, special characters and you have a pretty good bit of protection. What could be added, is, while the Caller is typing, a check for each character could be done. This now adds to the complexity of the Caller trying to crack the access code. There again, If someone is using a routine to crack the codes, then don't check each letter except for unwanted control characters. With BBS systems, often people with security clearance to access the DOS and rummage around someones computer, have quite an easy entry. However, with a little routine with these options [with the checking of those control codes etc;] you have the almost crack free system. There is nothing in this world that is 100% perfect, only degrees of near perfection.

+-----+
| 102 |
+-----+

Caller Message System

This routine is to be added to MO-2.BAS and then saved as MO-5.BAS.
The line numbers may have to be changed to suit your application program.

What this does is allow the Caller to place a message, Edit, List, Continue and Abort.

This routine would start in line 120. You could place a gosub 7000 instead of what it has.

If you want to see what is going on then place a PRINT with the string or variable expression after each PRINT# statement. You can just remove the #1, from the PRINT statement to display onto the screen. Remember to replace them when you want to use the routine correctly. Change the input routine to match an input from your keyboard.

First add these three parameters to the beginning of the program.

```
DIM LI$(90)
```

```
CURSOR_ONE$=CHR$(8)+CHR$(32)+CHR$(8)
```

```
CURSOR_TWO$=CHR$(29)+CHR$(32)+CHR$(29)
```

Now you can enter the following:

```
7000 PRINT#1,"E)dit, L)ist, C)ontinue, A)bort."+M$;
```

Print to the Modem the selection.

```
7010 GOSUB 7100
```

Get a response from the callers keyboard.

+-----+
| 103 |
+-----+

7020 C\$=LEFT\$(B\$,1):IF C\$="E" OR C\$="e" THEN 8010

Set C\$ to the first character. If the Caller wants to EDIT then we goto that routine.

7025 IF C\$="L" OR C\$="l" THEN 8100

If the Caller wants to LIST then we goto that routine.

7030 IF C\$="C" OR C\$="c" THEN 8200

If the Caller wants to CONTINUE with His or Her message then we goto that routine.

7030 IF C\$="A" OR C\$="a" THEN 76

If the Caller wants to ABORT, then we HANG UP.

7050 A\$="SYNTAX ERROR....Try again !"+M\$

If none of the above keys was selected the there was a keyboard entry error. A\$ is assigned the text.

7055 GOSUB 6040:PRINT#1,A\$

We gosub to the time delay routine and the print the error message that we set in A\$.

7060 GOTO 7000

The we restart the display.

Line 7100 monitors the input response via the Modem line.

7100 B\$="":TCC!=FNTI!+180:LOCATE,,1

B\$ is cleared out. The time factor is increased, and the cursor is switched on.

7105 IF EOF(1) THEN 7125

We check to see if we have an input character.

7110 A\$=INPUT\$(LOC(1),3)

We set the input file#1 to a time limit of 3 minutes.

7115 GOSUB 6000

Jumps to the subroutine timer.

7120 GOTO 7105

Back we go again to get input.

7125 A\$="":IF (INP(MSR) AND &H80)=0 THEN 76

Watch out for the Carrier to be dropped on us. If it is, we also hang up.

7130 IF EOF(1) THEN 7140

Has anything been entered.

7135 IF LOC(1)<>0 THEN A\$=INPUT\$(1,#1)

7140 Y\$=INKEY\$

7145 IF Y\$<>" THEN A\$=Y\$

7150 IF A\$="" THEN 7200

7155 TCC!=FNTI!+180

Increase the timer.

7160 IF A\$=CHR\$(8) AND LEN(B\$)=0 THEN 7200

If the back DElete was used and no characters were assigned to B\$, then we jump to check time delay.

+-----+
| 105 |
+-----+

```
7163 B=LEN(B$):IF A$=CHR$(8) THEN B$=LEFT$(B$,B-1):  
GOSUB 8000:GOTO 7200
```

If the back DElete key was used then B\$ is subtracted one character from the right.

```
7165 IF A$=CHR$(10) THEN 7200
```

```
7170 IF A$=CHR$(13) THEN 7210
```

The Caller has finished entering a line.

```
7175 IF (ASC(A$)>96) AND (ASC(A$)<123 THEN A$=CHR$(  
ASC(A$)-32)
```

Set a limit on characters that are used.

```
7180 IF LEN(B$)>78 THEN PRINT#1,CHR$(7);:GOTO 7200
```

The line length is limited to no more than 78.

```
7185 B$=B$+A$
```

```
7190 IF UC=-1 THEN A$="."
```

```
7195 PRINT#1,A$;
```

```
7200 IF FNTI!<TCC! THEN 7125
```

We still check to see if any characters have been entered within a time limit.

```
7205 GOTO 76
```

If a character has not been entered within that time limit, then we hang up.

```
7210 PRINT#1,M$;
```

A Carriage Return and a Line Feed is printed to the Callers console.

7215 LOCATE,,0

The cursor is switched OFF.

7220 RETURN

Return to the calling routine.

The routine below is a subroutine for controlling the cursor position.

8000 IF POS(0)>1 THEN PRINT CURSOR_ONE\$;:PRINT#1,
CURSOR_TWO\$;

8005 RETURN

Now for the Editing facility.

8010 PRINT#1,M\$+"Enter line number to Edit or Q
to Quit : "+M\$;

Print to the Callers console the above message.

8020 GOSUB 7100

Monitor the entry.

8025 CHECK\$=LEFT\$(B\$,1): IF CHECK\$="q" OR
CHECK\$="Q" THEN 7000

If the Caller selected Q for Quit, we then
goto the beginning of the display prompt.

8030 B=VAL(B\$)

Set B variable to the value of B\$.

+-----+
| 107 |
+-----+

8035 A\$=STR\$(B)+" >" +LI\$(B)

We assign A\$ for the displaying of the line number
with the text.

8040 GOSUB 6040:PRINT#1,A\$

First do the time delay then print the line etc;

8045 PRINT#1,B;">" ;

Print the line number to be edited without the text.

8050 GOSUB 7100

Monitor the Callers keyboard response.

8055 LI\$(B)=B\$

Assign the current string to the dimensioned string.

8060 GOTO 8010

Back to the ENTER LINE NUMBER OR ... message.

8100 COUNTER=1

Set the counter variable to 1.

8105 IF LI\$(COUNTER)="*** END ***" THEN 8135

If the current dimensioned dimensioned string is
END then jump to line 8135

8110 FM\$=STR\$(L)

8115 A\$=FM\$+" : "+LI\$(COUNTER)

8120 GOSUB 6040:PRINT#1,A\$+M\$;

8125 COUNTER=COUNTER+1

+-----+
| 108 |
+-----+

Increment the counter by one.

8130 GOTO 8105

8135 A\$=M\$

8140 GOSUB 6040:PRINT#1,A\$+M\$;

8145 GOTO 7000

Below is the routine to Continue.

8200 A\$="Press ENTER on blank line to end entry !"+M\$

8205 GOSUB 6040:PRINT#1,A\$+M\$;

8210 COUNTER=1

We set a variable as COUNTER to use.

8215 IF COUNTER=20 THEN LI\$(COUNTER)="*** END ***":
GOTO 7000

If the counter has reached 20 then it is the end of
the possible entry for the Caller. The routine will
restart the display. You could place a subroutine
to save the text using the APPEND command.

8220 PRINT#1,COUNTER;">"

Line number is printed ready for the text entry.

8225 GOSUB 7100

Scan for key entry.

8230 IF B\$="" THEN LI\$(COUNTER)="*** END ***":
GOTO 7100

If nothing has been entered the the last [or
current string is assigned.

+-----+
| 109 |
+-----+

8235 LI\$(COUNTER)=B\$

The entry of B\$ is assigned to the current string.

8240 COUNTER=COUNTER+1:GOTO 8215

The counter is incremented by one and then the routine continues to monitor both the line count and the key entry.

I have omitted a few PRINT# statements for you to add. By making this routine work on the screen, you will see when the text is not printed onto the screen. PLEASE SAVE THIS as MO-5.BAS.

With additions, you could have a very nice routine to get the Caller to operate.

You must always remember to cover your program with REM statements, so that you can come back to it at any time and know exactly What does What and where the branches are. There really is no point writing very long Code when the thread of it can be lost.

+-----+
| 110 |
+-----+

Make a Password Program

The following routine contains all the necessary information to build a really good entry system. It has been written [although in a fairly long form], to look up a Password and Entry file and compare it with the incoming information. It will then Shell to DOS [as one would if the Caller has the security clearance] and APPEND the information of the Caller to a file, with His or Her full details. You will have to convert the routine to work on the Modem, as this demonstration only uses your screen and keyboard.

You must first make up the three files that this program will use in one form or another.

Use your ASCII text editor or the COPY CON method.

The first will be called OK.DAT and look like this:

```
....NAME          Passwords
-----
Peter Wallis*This is a test
```

The next will be called BAD.DAT and look like this:

```
....Name      Town      Sec Call *Logged System*  Tel N0.  Baud
-----
```

NOTE ! Sec is the Security level of the Caller.

The third will be called INFO.DAT and look like this:
Note..[The 'REM statements MUST be removed.]

```
Peter Wallis  'REM Full name of caller
0
Leaf Town     'REM town
1000          'REM Security level
```

```
+-----+  
| 111 |  
+-----+
```

```
000  
A  
Let Me In      'REM Standard Password  
0  
0  
08:23  
19:29 05/05/90  
A  
0  
999  
0  
0  
0000000000  
02/05/90 19:29  
A  
A  
02/01/90  
0  
0  
0  
0  
0  
0  
A  
COM2  'REM change this if COM1  
07/24/52  
1200  'REM Baud Rate of Callers Modem  
A  
A  
02/06/90 00:29  
0  
0
```

REMOVE all REMs and save it now please.

This last file is similar to a file created by the WILDCAT BBS Sysops Program. If you are using this type of program, then this just might be the thing you are looking for.

```
+-----+
| 112 |
+-----+
```

Here is the main program for you. Remember to keep typing and then press ENTER. Each line will be a continuous line of commands.

```
1 CLEAR:COLOR 7,0,0:CLS:DIM A$(27),BB$(36):
ON ERROR GOTO 251:OPEN "i",9,"BAD.DAT":INPUT#9,JK:
OPEN "i",10,"OK.dat":INPUT#10,A$:
OPEN "i",11,"INFO.DAT":INPUT#11,DGF$:
OPEN "I",12,"INFO.DAT":FOR TE=1 TO 33:
INPUT#12,BB$(TE):NEXT

11 CLOSE#9:CLOSE#10:CLOSE#11:CLOSE#12:
IF SWITCH=1 THEN 311

21 TIMER ON:PRINT"+-----+

---+":PRINT"|C.P.White (C) 1989 (FOR MODEM BOOK) |"
31 PRINT"|";:COLOR 0,7:PRINT "DROP TO DOS REQUEST";:
COLOR 23,0,0:PRINT" 2 Minutes only!";:COLOR 7,0:PRINT
"|":COLOR 7,0:PRINT "+-----+
-----+"

41 PRINT:PRINT "** * * Quit = ENTER three times. * * *":
PRINT

51 PRINT"Penalties may be encountered if you":PRINT:
PRINT"are not a registered SYSOP user!!":PRINT:
ON TIMER (120) GOSUB 271

61 IF SWITCH=2 THEN 291

71 RS$(1)="Name or Names ":RS$(2)="Your password ":
RS$(3)=RS$(2):FOR KL=1 TO 3:IF KL>1 THEN PRINT
CHR$(10);

81 PRINT RS$(KL);:LINE INPUT;A$(KL):NEXT:
IF A$(1)="" AND A$(2)="" AND A$(3)="" THEN PRINT:
PRINT:PRINT"                                Returning without
penalties ":KK$=" Aborted ?":GOTO 291
```

+-----+
| 113 |
+-----+

```
91 OPEN "i",11,"OK.DAT":INPUT#11,XC$:INPUT#11,XXC$

101 WHILE NOT EOF(11)

111 POI=POI+1:INPUT#11,DSA$:FOR UYT=1 TO LEN(DSA$):
AZZ$=MID$(DSA$,UYT,1):IF AZZ$="*" THEN 161

121 AZS$=AZS$+AZZ$

131 NEXT:IF AZS$=A$(3) THEN 181

141 DSA$="":GOTO 101

145 WEND

151 CLOSE#11:GOTO 181

161 IF A$(1)=BB$(1) AND A$(1)=AZS$ THEN AZS$="":
GOTO 131

171 AZS$="":GOTO 131

181 SRT=VAL(BB$(4)):IF BB$(1)<>A$(1) AND SRT<1000
THEN KK$=" security risk":GOTO 281

191 IF BB$(1)=A$(1) AND SRT<1000 THEN KK$=" no
clearance":GOTO 281

201 IF BB$(1)<>A$(1) THEN KK$=" bad name":
GOTO 281

211 IF BB$(7)<>A$(2) THEN KK$=" bad BBS pass":
GOTO 281

221 IF AZS$=A$(3) THEN KK$=" ok 1":GOTO 291

231 IF AZS$<>A$(3) THEN KK$=" bad DOS pass":GOTO 281

241 KK$=" ?????????? ":GOTO 291
```

+-----+
| 114 |
+-----+

```
251 IF ERR=53 OR ERR=64 OR ERR=52 OR ERR=53 THEN
PRINT:PRINT:PRINT"SORRY...Sysop withdrawn this
facility for now.": FOR I=1 TO 1000:NEXT
```

```
261 IF SWITCH<2 THEN SWITCH=1:RESUME 11
```

```
271 PRINT:PRINT"                                **** OUT
OF TIME ****":SWITCH=2:KK$="    time out":RETURN 61
```

```
281 PRINT:PRINT"    You have either entered incorrect
ly or CLEARENCE has NOT been given !"
```

```
291 CLOSE#11:BJ$=BB$(1)+" "+BB$(3)+" "+BB$(4)+" "+
BB$(10)+" "+BB$(11)+" "+BB$(17)+" "+BB$(31):BJ$=BJ$+
KK$:OPEN "BAD.DAT" FOR APPEND AS #9:PRINT#9,BJ$:
CLOSE#9:IF KK$=" ok          1" THEN 301 ELSE GOTO 311
```

```
301 TIMER OFF:COLOR 7,0:PRINT:PRINT "Type EXIT when
ready to return to CALER Program !":SHELL
```

```
311 TIMER OFF
```

Now save this routine as PASSWORD.BAS

When changing this program to work on the Modem, remember to change the LINE INPUT to check for the Delete key. Also to create a string that holds the created text, this will be reduced by one when the Delete key is used. You must LOCATE the cursor and use the POS command.

Running this routine will produce three entries. The first will be the actual Name [in the correct format] that you log onto a BBS system. The second, will be the password for using the BBS board. You can remove this, if you do not require. The third is the special password to SHELL to DOS. All these must be typed in exactly, Uppercase and Lowercase letters. Where the special password is concerned, it may have any combination of any characters and so on.

There is a two minute countdown for the entries to be completed. As this routine only uses the straight forward INPUT routine, you will have to alter it slightly to monitor the TIMER between each character entry. You can add some extra conditions to stop unwanted characters getting through.

First make sure that you have Saved the routine then RUN it.

Type in Peter Wallis and press Enter, then type Let Me In. So far all is well. Now just type in anything. The file BAD.DAT will be APPENDED to, as the last password was not correct. Re run the routine and type in everything correctly, with the third line as "This is a test". The routine will now SHELL to DOS. You can use TYPE BAD.DAT to see the file of callers. Once finished, type EXIT and press ENTER. In this demo, the routine will return to itself and then just exit. You could use a preset Batch file [if you Compile the routine with QB 3.0 etc;] to control the action of this program. This could have the name of this program first, then the name of the calling [or main program] that allowed this routine to operate.

This routine will also check the security level of the caller. If it is less than 1000, even if all the information entered was correct, the Callers details will be APPENDED to the BAD.DAT file. This was done, just incase the Caller was demoted for one reason or another.

A small bonus, is that if any of the configuration files were removed, the Caller will not be given access to the routine and returned to the Parent program. Once again this was done for a good reason. If the Sysop decided to withdraw this facility, for any reason then He or She just has to rename one of the files.

+-----+
| 116 |
+-----+

I decided to add this routine as a normal procedure to get you thinking on how to get it to operate correctly over the Modem. The methods are available within the other routines. Have a go at it. You'll be suprised what you have learned so far.

Quick Basic Compiler Version 3.0 FIX

If you are using Quick Basic Compiler Vs 3.0, you may like to know that there is a problem with the Carrier being dropped when exiting from a Compiled program. To eliminate this, the following Patch can be used.

```

+-----+
| NEVER EVER ALTER THE ORIGINAL PROGRAMS !!!!|
+-----+

```

Copy both BCOM30.OBJ and BRUN30.EXE to a seperate diskette with DEBUG. Rename BRUN30.EXE to BR.X Now type in the following from the DOS prompt:

```

DEBUG BCOM30.OBJ
-a DC0B
xxxx:DC0B  Mov AL,1
xxxx:DC0D
W
Q

```

Now type in the following:

```

DEBUG BR.X
-a 6969
xxxx:6969  Mov AL,1
xxxx:696B
W
Q

```

Rename BR.X to BRUN30.EXE and you are ready to use these when compiling.

A different Fix to the problem is with the GWCOM.OBJ file. Use a Disk Editor and Replace the word COMCLOSE with spaces. When you produce your EXE file, LINK yourFile GWCOM,,; do not worry about all the error mesages thrown up. Your program will work perfectly it all aspects.

REFERENCE: BASIC and its commands.

The direct command from DOS:

```

BASIC[prog][<input>][<output>][ /c:buffer]
[/d][ /f:files][ /m:[address][,blocks]]
[/s:buffer][I]

```

prog

Execute a program after interpreter has loaded. If not stated then a READY prompt will be displayed.

<in File or device to read from.

>out File or device to direct output to.

/c:buffer Size of data buffer when
communications adaptor used.
Default 256 with maximum of 32767
bytes.

/d Mathematical function. ATN, EXP,
LOG, SIN, SQR and TAN. All using
double-precision mathematics.

/f:files

Total amount of files accessed at one time.
Default 3 with maximum of 15.

/m:address,blocks

Normally 64k. Can protect a portion of memory
at highest level above Interpreter.

/s:buffer

Random Access file default length is 128,
maximum 32767.

/I Allocates a Dynamic space to support
file operations.

+-----+
| 120 |
+-----+

The BASIC Editor and its commands

AUTO [stln/.][,inc]

stln Start first line number. Default is 0.

. Current line.

inc Incrementation between lines required.

CLEAR [data][,stack]

data

Reserve area for variables. Machine Code can be entered by decreasing the default [65535].

stack Reserve memory on the stack.
 Default 512 bytes.

CLEAR on its own will clear all variables from memory.

CONTD Continues a program that has been stopped by using the CTRL/BREAK key or by an END or STOP command within a program. If a program has halted due to a fault in the program, then it can not be restarted.

DELETE [stln/.][-[endln/.]

stln Start line to delete.

. Current line number.

- Deletes both start and end lines stated and all between. If not stated then lines from the start to the end will be deleted. If 'endln' is not stated after the minus then all lines from the beginning of the program to the stln will be deleted.

endln End line to delete to.

EDIT [ln/.]

ln Line number currently changed.

. Abbreviation for current line.

FILES ["flspc"]

flspc

Files on directory. Example: ".DOC" displays all files with the extension of .DOC If FILES is used on its own then all files will be displayed on current directory.

FRE(0) or FRE("")

FRE(0) Displays free memory.

FRE("")

Does the same thing by removing unused string space variables.
This does a garbage collection.

GOTO ln

ln

States which line number to execute a program from.

KEY ON/OFF/LIST

ON 25th line displays current settings.

OFF Switches function keys off.

LIST Displays to screen the current settings.

KEY num, strn

num

From 1 to 10 to assign function keys to specific tasks.

strn

Maximum programmable characters is 15 per function key, each can contain control characters. If "" (double quotes) are expressed then the num is disabled.

LIST [from/.][-[to/.]][,dev/filspc]

from First line number.

. Abbreviation for current line number.

- Separation between from and to line.

to Last line number.

dev Program to be listed to a curtain device.

filspc File to be listed.

LLIST [from/.][-[.to]

from Program to be listed from first line to a printer.

. Abbreviation for current line number.

- Separation between from and to line.

to Last line required to be listed to a printer.

LOAD "[drv:\path\file"][,R]

drv: Drive from where the program is to be loaded.

\path Name of directory.

\file A Basic program to be loaded.

,R Run the program after it has been loaded.

If programs are on the current drive, then drv, /path ,R can be omitted. ,R is optional.

MERGE "[drv:\path\file.BAS]"

drv Merge a program from a drive.

\path Name of directory.

\file Program to be merged.

.BAS Can be omitted as BAS extension is assumed.

If programs are on the current drive, the drv and \path can be omitted.
 The file must be in ASCII format [saved with the ,A command (see SAVE)] and must have higher line numbers than the current file in memory.
 If the file being merged has a start line number lower or equal to the one in memory then it replaces those lines.

NEW

Removes a program from memory, initialises all variables.

REM or :REM txt/'txt

txt

Can be any character. Used as remark statements which are not executed by the Basic Interpreter, but does take up memory with the exception of Quick Basic Compiler which when compiled does not include these statements as part of the finished .EXE file.

' Quick Basic Compiler will use this (if required) on its own instead of the REM statement.

RENUM [newln/.][,fromln/.][,inc]

newln Start renumbering to this new line number.

fromln

Starts renumbering from this current program line number.

inc Set incrementation (step) between line numbers.

Can be omitted. Default is step 1.

. Abbreviation for current line number.

RUN ["[drv:\path\][stln/."] [,R]

drv: Drive from where program is to loaded.

\path Directory name.

\stln

The start of a line from which the program will be executed.

. The current line within memory.

,R This indicates all open files are to remain open.

RUN or RUN stln/.

Are used to run a program from within memory. All other commands are used for loading programs and running them with various parameters.

stln Start of line number to execute program.

SAVE "drv:\path\nme"[,A/ ,P]

drv:

Drive on which program is to be saved. Can be omitted.

\path Directory name. Can be omitted.

\nme

Name of program to be saved. The extension can be named. If a file name saved is the same as a file on the disk/ette then it will be overwritten.

,A Save a file in ASCII format.

,P

Save a file to disk in protected format, which can not be listed or edited.

STOP

This command will stop a program at a certain point while it is running.

```
200  FOR I = 1 TO 100 : NEXT : STOP
```

```
210  BEEP
```

When the loop has finished the program will stop with a statement of:

Break in line 200

You can continue the program from the next line by using CONT command providing a error has not occurred.

SYSTEM

This will return you to DOS, leaving the Basic Interpreter completely and not saving the current program in memory.

TRON/TROFF

A trace facility which can be turned ON and OFF which displays the current line being executed.

Similar DOS type commands in Basic:

CHDIR, MKDIR and RMDIR

CHDIR path Change path

MKDIR dirname Make a new sub-directory

RMDIR dirname Remove sub-directory

Abbreviations can not be used as in DOS.

ENVIRON entry = assignment

DOS stores information about the environment.
This can be accessed from Basic.

entry PATHs, COMSPEC or parameter
 specified by SET.

assignment A new parameter to be set to
 entry.

```
ie;  ENVIRON "PATH = B:\PETER" or
      ENVIRON "PROMPT = $p$_$n:"
```

ENVIRON\$ ("entry") or (num)

The information stored in DOS about the
environment can be accessed.

entry A legal variable like PROMPT, PATH
 etc. The return assignment is
 displayed.

```
PRINT ENVIRON$ ("PATH")
```

The readout may be... B:\MASM

+-----+
| 128 |
+-----+

IOCTL # filename, control_string

A control string is sent to a device driver which has already been OPENed. The MS-DOS drivers will not except IOCTL strings.

filename File number used when file OPENed

Control_string The number containing data
 to be evaluated by the driver

IOCTL\$ ie, A\$ = IOCTL\$(# filename)

The answer from the device driver given to a control string sent by IOCTL\$.

filename File number used when OPENed.

SHELL filespec (optional. Files or DOS)

Filespec A program to be executed which
 must be enclosed with "". Used
at DOS level. ie, SHELL"DIR"
COMMAND.COM must be present on the disk.
if the 'filespec' is found, then on returning
to the program, the next line will be run.

TIMER ie, A = Timer

The 'A;' will return the time elapsed from midnight (0:00 hours) in seconds/hundreds. Independent of set time. Some versions will give aresult from the time the computer was turned on.

Shortcut KEYS for the Editor

These may differ depending on the version of
Basic.

Using the ALT key with another:

A = AUTO B = BSAVE C = COLOR D = DELETE

F = FOR G = GOTO

H = HEX\$ I = INPUT J = unused K = KEY

L = LOCATE M = MERGE

Editing keys

Enters the current line in memory ENTER or CTRL/M

All Input/Output is redirected to the printer.
CTRL/P or CTRL/PrtSc

Clear screen and place cursor top left.
CTRL/Home or CTRL/L

Clear rest of program line from cursor.
CTRL/End or CTRL/E

Clear line, place cursor at start of line.
ESC

Clear rest of screen from cursor line.
CTRL/Z or CTRL/PgDn

Delete character under cursor.
Del

Delete character left of cursor.
Backspace or CTRL/H

Delete program line to next colon or space.
PgDn

Insert blank line within the program line.
CTRL/ENTER or CTRL/J

Move the cursor right to the next tab position.
Tab or CTRL/I

Put cursor at end of line.
End or CTRL/N

Put cursor top left of screen only.
Home or CTRL/K

Move cursor to start of next word.
CTRL/->

Move left to start of previous word.
CTRL/<- or CTRL/B

Stop screen listing. Press again to undo.
CTRL/+ or Num Lock

End edit mode and cause any changes to be lost.
When running a program, the control keys will
return Basic into the direct mode:
Command is CTRL/C or CTRL/Break.

Print the current screen contents to printer.
If using graphics mode, use the GRAPHICS.COM
file supplied with some PC's.
Command is Shift/PrtSc.

Scroll the listing up. Cursor must be on a
line number to operate correctly. Command is
CTRL/Y or CTRL/cursor down.

Scroll the listing down. Cursor as above.
CTRL/X or CTRL/cursor up.

Toggle between overwrite and insert.
Ins or CTRL/R

Show function key operations on 25 line.
CTRL/T

File Handling

CHAIN [merge] file[stln][,all][,delete
fromln-toln]

Programs requiring CHAIN must be saved in ASCII
format with the ,A.

merge

Used in earlier versions of Basic and still
maintained depending on the version of Basic
you have. Same effect as the CHAIN command.
The program is executed at the stln (start
line) given.

file ie; drive:\path\

stln Starting line which will execute
when CHAINED.

all

Preserves all existing variables. If not used,
then the variables will be erased from memory.
See COMMON for specified variable protection.

delete fromln-toln Deletes line numbers
as stated:

fromln From first line number.

- Separator.

toln To last line number.

CLOSE [# filenum,...]

The CLOSE command on its own will close all
open files.

filnum

File number which has been opened. When closed, an error message will occur if attempts are made to read or save to it unless it has been re-opened.

COMMON var,...
var

The variable/s and or arrays [separated with comers] to be preserved whilst using CHAIN command.

A DIM command must follow a COMMON command and the variable/s and or arrays must be assigned a value.

CVD, CVI & CVS

Converts data compressed with MKD\$, MKI\$ and or MKS\$ to numeric value.

CVD

Returns a double-precision result.

CVI Integer result.

CVS single-precision.

ie: A = CVD [fieldvar]

fieldvar .. A named field variable which was assigned a string by MKD\$, MKI\$ and or MKS\$.

EOF# [filenum]

filenum Number of file when opened.

Test if end of file reached when reading a file.

+-----+
| 134 |
+-----+

FIELD# filename,length AS stringvar,...

filename The file number used when opened.

length AS stringvar

The length of individual fields of a buffer.
This is for buffer variable/s used to
read/write to random access files.

LSET and RSET must be used before they are
written to a file using the PUT# command.

GET# filename[,recnum]

filename The file number used when a file
 is opened.

recnum

Number to read of an open record. Values from
1 to 16,777,215.

INPUT# filename,var,...

filename File number used.

var

Variable/s stored in a data file. Arrays can
be used.

INPUT\$ IE, INPUT\$[numchars[,#filename]]

numchars

Number of string characters read from a file.
Value 1 to 255.

filename Number used when file opened.

KILL "drv:\path\filespec"

"drv: Drive required. Optional.

\path Sub-Directory required. Optional.

\filespec" Deletes a file name.

LINE INPUT# filenum,stringvar

line input#

Read characters from a file until 255
characters have been loaded or until carriage
return/line feed has been encountered.

filenum File number of opened file.

stringvar Data of which a string variable is
stored.

LOC ie; A = LOC [filenum]

filenum Current file number used when
opened.

Sequential files return a value in 128-byte
blocks for either Read or Write.

LOF ie; A = LOF [filenum]

filenum Current file number used when
opened.

Size of previously opened file in multiples of
128 bytes.

LSET & RSET

LSET

sets the string to left justified, while RSET set it right justified also used to format normal string variables.

LSET FIELD-var = stringvar

FIELD-var

The named String variable as defined in FIELD statement.

string-var Name of string to be assigned to
FIELD-var.

MKD\$, MKI\$ & MKS\$

Numerical expression converted to a string for a Random-access file/s ONLY.

A\$ = MKD\$ [expression]
expression

Constant, numerical or result of a function or calculation.

MKD\$

A numeric expression in double-precision, converted to 8 byte string.

MKI\$ Integer expression converted into
2-byte string.

MKS\$

Single-precision numerical expression which is converted into a 4-byte string.

NAME old-file TO new-file.

old-file

Optional drive & path. Filename of file to be renamed with extension.

new-file

New filename with extension in either variable or string constant form. Constants must be enclosed in quotes. Strings are not allowed. Error will be given if new-file already exists and will not be over-written.

OPEN "drv:\path\file" FOR mode AS # filenum [
LEN = recordlen]

"drv: Drive required. Optional.

\path\ Sub-directory name required.
 Optional.

If drv and or \path\ are omitted then you will operate from the current drive and path you are in.

file" File name.

mode ie; INPUT Open data file to
 read from.

OUTPUT Open data file to write to.

APPEND Open in OUTPUT mode but place file
 pointer at end so it can be extended
 later on.

If no file exists then one is created.

filenum

The file number that is required by you to perform a function.

[LEN = recordlen]

LEN

If LEN is used instead of mode then the file will be opened as a random access file.

= recordlen

Record length set the total length of the file in bytes. Range from 1 to 32767.

OPEN "mode", # filenum,"drv:\path\filenme"
[,recordlen]

As used and supported in earlier versions of GW-Basic.

mode:

I Open data file to read with pointer in front of file.

O Open data file to write with pointer as above.

If file exists then it over-writes that file.

A Same as APPEND.

R File is opened for random-access operations.

filenum File number required by you to read or write to.

"drv: Drive required. Optional.

\path\ Sub-directory name required.
Optional.

filenme" File name required.

,recordlen

Record length must be stated in mode,R. Length
range is from 1 to 32767.

PRINT # filenum, exp,....[;/,]

To write a data file.

filenum A number given by you to write.

exp An expression of constants,
variables or a result of

WRITE# Filenum [SPC(num)] [expression][,]

filenum The number to assign to an OPENed file.

SPC(number) Write a requested number of spaces.

expression Can be variables, constants of resulting
calculations. Can take any form.

, Tabs line with eight blank spaces before
the next item is written.

Note ! You can not use the TAB command !

Functions and Calculations.

```

;      Expresses a constant flow of data.

```

```

,      Expresses add eight spaces between
      expressions.

```

```

PRINT# filenum USING "mask"; expres

```

This formats data output to an open file.

```

filenum      Number of file required ( after
              opened ).

```

```

"mask"       See Screen Output for full details.

```

```

expres       The data to be written. Constants,
              variables calcs or functions.

```

```

PUT # filenum [,recnum]

```

A file record is written to disk from the defined FIELD and as many characters as predefined by the OPEN command.

```

filenum      Number of file required ( after
              opened ).

```

```

recnum       The record file number to be
              written.

```

```

WRITE # filenum [SPC(num)][exp.....][,]

```

```

filenum      Number of file required ( after
              opened ).

```

```

SPC (num)     NUMBER of spaces as specified in
              num.

```

```

exp          Type of data to be written.
              Constants, variables etc

```

```
+-----+  
| 141 |  
+-----+
```

, Tab output of eight spaces before
 next item.

Error Trapping -----

```

ERDEV          ie;  A = ERDEV

```

Interrupted by MS-DOS error set via INTERRUPT
24 giving low byte. High byte gives bit pattern
of device driver that caused error.

```

ERDEV$         ie;  A$ = ERDEV$

```

```

PRN            Error from printer

```

```

A:B:C          Error from Disk.

```

```

OTHER          Error on driver.

```

This actually gives the error name of the
device.

```

ERL            ie;  A = ERL

```

Reports the line number that produced the
error.

```

ERR            ie;  A = ERR

```

Gives the Code Number for the error.

```

ERROR num

```

```

num            Number that you want to use to
                simulate an error.

```

```

ON ERROR GOTO linenum

```

```

linenum        The line number that the error
                trapping routine begins.  If a 0
                is used then error trapping is turned off and
                the normal error messages via DOS is displayed.

```

+-----+
| 143 |
+-----+

RESUME [linenum/NEXT/0]

linenum The line number from which the
 program is to continue.

NEXT Continue the program from the next
 line down after the error occurred.

0 Similar to RESUME command but
 without the parameters and the line
 that produced that error is
 executed again.

Variables and Constants

DEF... Allowing several variables in data
format.

DEFINT from-to Data type as an interger.

DEFSNG from-to Single-precision.

DEFDBL from-to Double-precision.

DEFSTR from-to String.

from Global definition of first line.

to Global definition of last line

DIM varnme (subscripts) [,varnme (subscripts)]

Reserve memory for one/multi dimensioned arrays
with indexes greater than 10.

varnme The array name to be dimensioned.

Numeric arrays are defined by %,! # and are
appended to the name. Strings have \$.

subscripts One or more expressions to specify
the number of elements and
separated by comers. Maximum 255.

ERASE var [,var....]

To erase one or more predefined variable/s.

var Name of array/s to be removed.

LET var=val ie; LET A = 10
LET is optional.

var Assign a variable value to a variable.

val The numeric value of either a variable or numeric expression to be assigned to var.

OPTION BASE 0/1

Array indexes are set at lower limits with 0 or 1.

0 Starting indexes of 0

1 Starting indexes of 1

READ and DATA

READ var,[var....] ie; READ A, B

DATA ["txt"/txt],/val ie; DATA "PETER", 10

Read fixed data values defined from within the program. Can be made into single statements or multi.

var Read a variable from a predefined table in DATA.

"txt" Data to be read by a string
ie; READ A\$

txt Data to be read by a string
ie; READ, one,two

All data types can be mixed providing that they are READ with their corresponding string and variable.

RESTORE linenum

Reset the DATA pointer to READ from a specified line number.

linenum Line number stated. If this is omitted then the pointer is set to the very first DATA line number in the program.

SWAP var1, var2 ie; SWAP a\$,b\$
Both must be the same type.

Swaps round the contents of variables around.

Command structures

FOR....NEXT

FOR indexvar=startval TO endval[STEP
stepwidth]statement(s):NEXT[indexvar]

A loop statement which executes until then NEXT
command has reached the end of the parameter of
endval.

indexvar The integer value to hold the
 value of the loop.

startval The start integer value of the
 loop.

endval The maximum range wanted.

stepwidth The increment in either positive
 or minus values.

positive values are normal expressions ie; 1,2
etc.. which step through forward. Minus -1,
-2 etc, step backwards.

statement Supported by GW-BASIC to include
 any command or function.

NEXT [indexvar] Index variable to specify
 the maximum lines for the
 loop to execute.

EXAMPLE:

100 FOR I = 1 TO 100 : NEXT I (or)

100 FOR I = 1 TO 100 : PRINT "PETER" : NEXT I

The [indexval] need not be specified.

GOSUB....RETURN ie; GOSUB [linenum]
 RETURN[linenum]

Calls and executes a subroutine and returns to the command line or line number in the program as specified in RETURN.

GOTO linenum

Branches unconditionally to a line number.

linenum Line number that the program is to jump to.

IF...THEN...ELSE

IF condition THEN statement1 [ELSE statement2]

Executes program statements (or branches) based on results of a condition or conditions set.

statement1 and statement2

Single or multi statements of commands or functions separated with colons. Total length not to exceed one line.

condition Any comparison or condition.

ON x GOTO/GOSUB

ON index GOTO line number/(s) or
ON index GOSUB line(s).

This tells the routine to branch conditionally to subroutines or sub-programs via the amount set in the index.

EXAMPLE:

100 ON A GOTO 100 or

100 ON A GOSUB 10,20,30 Single or multi
 indexes.

index

The integer value of a function, variable or
calculation

line number/s

Line/s separated with commas to branch on value
of index.

WHILE....WEND

WHILE condition WEND

As long as a condition is true the statement is
executed.

condition Logical expression for comparison.

EXAMPLE:

WHILE A = 10 THEN do this

WEND If <> 10 then ignore this expression.

+-----+
| 150 |
+-----+

Data handling -----

= IF this = (equals to) this THEN do this.

< IF this is < (less than) this THEN do this.

<= IF this is <= (less than or equals to) this THEN do this.

> IF this is > (greater than) this THEN do this.

>= IF this is >= (greater than or equals to) this THEN do this

<> IF this is <> (no equal) to this THEN do this.

EXAMPLE:

100 IF A = 10 THEN GOTO 300

200 GOTO 400

300 PRINT "A = 10" : END

400 REM the rest of the program

AND ie; IF condition1 AND condition2
THEN do this.

The statement is executed if both expressions
are true. Same as

IF A = 1 AND B = 10 THEN do this.

This is a command

... THEN GOTO/GOSUB/RESTORE/RETURN

NOT ie; IF NOT condition THEN do this

condition Use =, <, >, <>, <=, >=

OR ie; IF condition OR condition THEN
do this.

ASC (expression) Returns a decimal value of
an ASCII char.

Only first character of a string constant,
variable or function is converted.

CDBL (expression)

This converts a numeric expression to a
double-precision value.

CHR\$(number)

Convert numeric value to ASCII character.

CINT(expression)

Integer value of a numeric value.

CSNG(expression)

Single-precision value of a numeric expression.

DEF FN varname(parm,...) = function
FN varname(pram,...)

Defines and calls a user defined routine.

varname A numerical function that is called
with FN

parm A parameter of one or more specified
variables.

function Defines how the variables are used.

+-----+
| 152 |
+-----+

HEX\$ (num)

Converts a decimal value to a hexadecimal one.

num A number from the range of -32768 to
 + 65535.

OCT\$ (num)

Converts a decimal value to a string of octal
digits.

num A constant, numeric value or
 calculation, or the result of a
 numerical function in a 16 bit
 value.

RANDOMIZE [NUM] and RND [NUM]

Sets a numerical basis for random numbers and
generates a random number. NUM can be from
-32768 to + 32767.

STR\$ (express) ie; A\$ = STR\$(32)

Convert a number to a string.

Can be a numerical result of a function, a
constant etc.

VAL (stringexpress) ie; A = VAL (A\$)

Converts a string (which has numerical
contents) to a numeric value.

```
Addition (+)      result = expression +
                    expression
```

Subtraction (-) result = expression -
expression

```
Multiplication (*)    result = expression *
                      expression
```

```
Division (/)      result = expression /
                  expression
```

A back slash (\) is used for integer.

```
Exponentiation (^)    result = expression ^
                      expression
```

ABS Absolute value.
ie; A = ABS(expression)

AND ie; A = B AND C Result if both are
 set or one is set.

ATN **ie; A = ATN(expression)**

Angle of arctangent calculated and return radians in single precision. If Basic/D is used when calling from DOS then double precision is used.

COS **ie; A = COS(expression)**

Cosine of an expression is calculated and returns the radians in single precision. If Basic/D is used, then as above.

If B and C are not the same, the bit is reset.

```
EXP      ie; A = EXP(expression)
```

The constant e is raised to the power of the expression. The result is in single precision. If /D then double precision.

FIX ie; A = FIX(expression)

This returns the integer portion of the expression. Functions after the decimal point are truncated.

```
IMP      ie; A = B IMP C
```

If the combination of the two values B and C are the same, or if B is reset and C is set then the resulting bit is set.

INT ie; A = INT(expression)

The expression is returned with the greatest integer value.

LOG ie; A = LOG(expression)

The expression is returned with the natural logarithm.

MOD (modulo) ie, A = expression MOD
 expression

Gives the remainder of an integer division.

+-----+
| 155 |
+-----+

NOT ie; A = NOT Y

Sets each bit in Y if either set/reset
depending on the action taken before calling.

XOR ie; A = B XOR C

If both B and C are equal then bit is set.
The resulting bit is set only if either B or C
is set.

SGN ie; A = SGN(expression)

Returns the sign of the expression. +1 for
positive, -1 for negative and 0 for equal to
zero.

SIN ie; A = SIN(expression)

The sine of the expression is calculated in a
single precision radians. If /D is used the
double precision.

SQR ie; A = SQR(expression)

Square root of the expression in single
precision. If /D then as above.

TAN ie; A = TAN(expression)

Returns the tangent of the expression in single
precision. If /D then Double precision.

+-----+
| 156 |
+-----+

Screen output -----

CLS Clear screen and place cursor to top
 left of the screen.

COLOR [text col],[background]

Set up screen's text and background colours.

NB!!! Colour Cards:

CGA card only allows selections between 0 and 7
while EGA cards allow 0 to 15.

Composite:

The colours are in shades of gray and some
combinations will not be suitable to read.

text col A value between 0 and 15 to set
 character colours.

background A value between 0 and 7 to set
 background colour.

Monochrome Monitors -----

```

Normal          COLOR 7,0
Double Intensity COLOR 15,0
Underline       COLOR 1,0
Reverse Video   COLOR 0,7
Flashing        COLOR 31,0

```

Colour -----

```

0 Black   1 Blue   2 Green  3 Cyan
4 Red     5 Magenta
6 Brown   7 Light Gray 8 Dark Gray
9 Light Blue
10 Light Green 11 Light Cyan 12 Light Red
13 Light Magenta 14 Yellow 15 White

```

You can add 16 to the text colour value which causes it to flash.

```
CSRLIN    ie; A = CSRLIN
```

Gives the current cursor position.

```
LOCATE [line][,column][,1 (on/off)
2][,fromln][,toln]
```

Sets the cursor position on the screen, turns it on or off and sets the cursor shape.

line The Vertical position between 1 and 25.

column The horizontal position on a line between 1 and 80 or 1 and 40 depending on the screen type.

on/off The cursor can be switched on or off. 1=on and 2=off.

fromln, toln The cursor can have a number of shapes depending on the monitor and card you are using. The top line is 0 while the bottom line can be up to 31. For Composite and colour it is 0 to 7, while Monochrome is 0 to 31.

POS(0) ie; A = POS(0)

Shows where the cursor is located by column.

SCREEN [mode][,colour][,page][,display]

To set text mode and colour and selects display pages.

mode In which operating mode the screen is to operate.

0 text mode 80 x 25 or 40 x 25

1 graphics mode 320 x 200 pixels

2 graphics mode 640 x 200 pixels

x graphics mode 640 x 400 pixels for version 2 and mono cards.

colour Colour remains the same if set to 1. If set to 0 then text colour is light gray and background is black.

page There are 4 pages available (0 to 3) on 80 x 25.

There are 8 pages for the 40 x 25 monitors.

display To actually display a page. Same selection as above.

SCREEN ie; A = SCREEN(line,column[0/1])

The ASCII value (code or attribute) is returned from a position on the screen.

line, column The position of the character of the screen.

Values of 1 to 80 or 1 to 40 depending on monitor type.

0/1 0 for the ASCII code or 1 for the colour code.

VIEW PRINT [fromln] [TO] [toln]

This command defines the size of a text window.

fromln , toln Only full lines can be displayed. The range of line numbers is 1 to 25.

WIDTH numchars

To state a number of characters to be displayed on output.

numchars States either 40 or 80 column. Use 40 or 80.

This command will clear the screen before setting the width.

+-----+
| 160 |
+-----+

```
PRINT [TAB(column)][SPC(num)]  
[expression....][;][,]
```

This displays to screen.

column A value of between 1 to 80 or 1
 to 40 depending on the width of
your screen. A horizontal position on one
line.

num A number of spaces specified.

expression The type of data to be printed.

;
 Suppress the carriage return and
 print next statement directly after
 the one just printed on the same
 line.

,
 Print next character 8 places along
 on the same line.

PRINT USING mask; expression

A user formatted data display.

mask A string variable or constant to
 set formatting the characters ON
 the screen.

expression Data type to be printed.
 Can be constants, variables
 etc.

Numerical data MASK type:

"#####" Each hash(#) represents a digit.

If a sign is required then an extra hash is
added. Numbers are rounded off before being
printed.

"#####.##" Decimal places.

"#####.##" A plus before or after will display the sign either

"#####.##+" before or after the result.

"#####.##-" A minus sign will display the negative value only.

Positive values are without signs.

"#####.##^" An Exponential expression is displayed.

"*#####.##" If the value is less than the digits required then the asterisks are printed in front of the displayed value.

"\$#####.##" The dollar sign will place a dollar in front of the displayed value. Can not be used with ^ and if preceding asterisks then only the dollar will be printed.

"#####,.##" Comma which has been incorporated will display in groups of three digits separated with commas.

String data MASK types:

"\ \ " Number of characters to be printed depending on the amount of spaces in between the back slashes(\).

"&" Print the string as unformatted.

"!" Print only the first letter of
 a string.

"_" (underline) Everything preceeding the
 underline is not formatted.

This is allowed for those characters which are
not normally included in the mask rules.

"#####.## text" The text is printed relative
 to the print.

;
 Used to separate an expression
 from a mask and visa-versa.

WRITE [SPC(num)]["expression....."][,]

Prints output to screen.

SPC(num) Print a specified number of spaces
 on current line.

expression The type of data to be printed.

' Print 8 characters along on the
 same line.

The TAB and semicolon (;) can not be used with
WRITE.

Graphics

CIRCLE[STEP] (out_x,out,y),radius[,pen_col]
[,arc_str,arc_end] [,axes_ratio]

To draw an ellipse or a circle on a graphic screen.

out_x,out_y The absolute coordinates of a circle or ellipse used with out the STEP specifications. STEP specifies the position relative to the last point on the screen. Y range from 0 to 199 and X range from 0 319 or 639 depending on the type of screen you have.

radius The radius of a ellipse or circle.

arc_str,arc_end The starting and ending points. Values are angles in radians. Values from -2 pi to +2 pi.

COLOR [foreground][,background][,text_colour]

Sets a high-resolution graphics screen mode (640x200 points).

foreground Both text and graphic colours. Range 0 to 15.

background Default value is set to black. High-resolution is for monochrome and some conversion is necessary:

0 Black 1 White 2 Black 3 White

+-----+
| 164 |
+-----+

COLOR

[background][palette][,graf_back][,graf_fore]
[,text_col]

Set medium-resolution graphics mode (320x200 points).

background Set colour of entire screen.
 0 to 15.

palette One of two palettes to be
 set:

0 colour 0 = background

0 colour 1 = green

1 colour 0 = background

graf_back,graf_fore Standard values of
 drawing colours used in
graphic commands depending on palette last
used.

text_col Based on the palette used,
 specifies the text colour.
The value of '0' cannot be used.

DRAW string Draws a graphic as set out by
 a string.

string Can be a variable, constant,
 or a result of a string
expression. The current position at which DRAW
will begin, either the centre or the last
position used. PSET must be set before the
DRAW command can be issued. The string can be
set by using the following:

U num To draw a number of points up.

D num Down.

```

L num      Left.

R num      Right.

E num      Diagonally to the right.

H num      Diagonally to upper left.

F num      Diagonally to lower right.

G num      Diagonally to lower left.

```

The variable within the string must have equals sign (=) before it and a semi-colon after it.

```

B dir      Move the pen one point without
            drawing.

```

The dir uses U,D,L,R,E,F,G or H.

```

N dir      Move the pen one point in any given
            direction.

```

```

M,X,Y      If X,Y are not preceded with a plus
            or minus sign, then the line is
            drawn from the current point.

```

```

C          Set the drawing colour.

```

```

P,F,B      Fill a position enclosed by border
            color B and fill it with color F.
P does not have any parameters and must be
used.

```

```

A angle    Set the angle of movement.

```

ie; 0= 0 degrees, 1 = 90,

2 = 180, 3 = 270 and so on.

```

TA angle   Angle of movement. Range -360 to
+360.

```


+-----+
| 166 |
+-----+

S factor To return a value from the formula
factor/4 that should

be multiplied with all the numerical parameters
for movement to follow. This allows objects to
be drawn to scale with one another.
Range from 0 to 255.

X string_var; To store often repeated
procedures.

GET [STEP] (X1,Y1)-[STEP] (X2,Y2), array

Stores graphics in an array so that they can be
stored onto a diskette or moved onto the
screen.

STEP (X1,Y1),STEP(X2,Y2)

X1,Y1 are values of the lower left corner and
X2,Y2 are for the upper right corner to be
stored.

Y range up to 199 while X can be up to 319 or
639 depending on the resolution selected.

array Must be previously dimensioned with
DIM of which the size is to be
stored. Size can be calculated by the
following:

$A = 4 + \text{INT}((\text{pointX} * B + 7) / 8) * \text{pointY}$

A = 1 for 640x200 resolution.

A = 2 for 320x200.

pointX = width of graphic in pixels.

pointY = height of graphic in pixels.

+-----+
| 167 |
+-----+

```
LINE [STEP] [(fromX,fromY)]-[STEP](toX,toY)
[draw_col][B,[F]] [,raster]
```

To draw rectangles and lines.

fromX,fromY The starting coordinates of
 the lines be drawn.

If the STEP command is not stated then the
absolute value is taken as the starting point.

STEP fromX,fromY puts the point
 relative to the last address
position on the screen. X range 0 to 39. Y

range 0 to 639 depending on the resolution of
the screen. For a rectangle, this is the
bottom left.

toX,toY The ending point of a line.
 Y range 0 to 199 and X range
from 0 to 319 or 639 depending on resolution.
For a rectangle, this is the upper right
corner.

draw_col The colour of the line to be
 drawn. Range 0-3.

B To draw a rectangle.

F If F follows a B then the
 rectangle is filled in with
the colour specified in COLOR command.

raster Normally a solid line is
 shown, but the appearance can
be changed with a 16 bit mask.

LINE This converts the coordinates
 outside the defined SCREEN area
to prevent the program terminating in a error.

PAINT[STEP] (X,Y) [,mode] [,border_col]

The enclosed area is filled with a given colour or pattern.

(X,Y) The absolute coordinates of a point. With the STEP command; last relative position on the screen. Y range 0-199 and X range 0-319 or 639 depending on the screen resolution.

mode Colour or pattern an area. If COLOR has a value of from 0 to 3 then that colour is used. If not, you can make a 8x64 bit mask (64 bytes) as a pattern. The pattern is specified with CHR\$.

border_col The surrounding area of the region to be filled.

Colour is set by border and ranges from 0 to 3.

PMAP ie; A = MAP(coordinate),mode

Assigns a variable with the changed coordinate assigned with WINDOW.

coordinate X or Y converted coordinate according to mode.

mode=0 X coordinate is set with WINDOW and converted to an absolute coordinate value.

mode=1 Y (as above).

mode=2 An absolute X coordinate which is converted to an axes set with WINDOW.

mode=3 Y (as above).

+-----+
| 169 |
+-----+

POINT ie; A = POINT (X,Y)

Assigns a variable with the point colour at a given coordinates.

(X,Y) The coordinates being tested.

The returned colour value is between 0 and 3.

A	0	Coordinate of X
	1	Coordinate of Y
	2	World coordinates of X
	3	World coordinates of Y

PRESET [STEP] (X,Y) [,draw_col]

A specified point is cleared from the screen.

(X,Y) The absolute coordinates or if specified with STEP, will be the position of the last relative point.

Range of Y, from 0 to 199 and X between 0-319 or 639 depending on the resolution of the screen.

draw_col The point colour. If omitted, the point will be in background colour. Range 0-3

PSET [STEP] (X,Y) [,draw_col]

Set a point at a specified coordinates.

STEP Relative position of the last position on the screen.

Without STEP, gives the absolute position.

```
+-----+
| 170 |
+-----+
```

(X,Y) Y range between 0-199, X range
 0-319 or 639.

draw_col To specify the actual point
 colour. 0 to 3.

PUT (X,Y),array [,mode]

With the GET command, will display a graphic
stored in an array.

(X,Y) Upper left point.

array The array that contains the
 graphic.

mode States how the stored graphic
 is to be displayed.

Mode commands -----

PSET	Output exactly as read.
PRESENT	Invert the graphic (in reverse).
XOR	The point is inverted one way or the other, depending on how it is displayed.
AND	The point is ANDed.
OR	The point is ORed.

See mathematics for more descriptions.

```
SCREEN [mode][,colour][,output_page]
[,display_page]
```

Set the graphics mode and resolution.

Mode commands:

- 0 Text.
- 1 Medium resolution.
- 2 High resolution.
- 100 Special modes for other cards.

colour When the SCREEN is called and color=1, then the colour is retained. But, if color=0 the default colours are set.

Default colours: Black background and white foreground.

output_page,display_page

In graphics mode there is only one page.
Parameters are ignored.

VIEW [SCREEN](X1,Y1)-(X2,Y2)[background_col]
[,border_col]

The size and colour of the window to be
displayed.

(X1,Y1)-(X2,Y2) Defines the lower left and
upper right corners of the window.

background_col Background colour of the
window. 0-3.

boarder_col Border colour of the out side
of the window.

WINDOW [SCREEN](X1,Y1)-(X2,Y2)

This command defines the coordinate system to
be used for the current window, and is normally
in world coordinates.

World coordinates start a 0,0 for the centre of
the screen. The X value above the centre are
positive and below are negative. The Y values
to the right of the screen are positive and to
the left are negative.

(X1,Y1) The upper left corner.

(X2,Y2) the lower right corner.

Getting input from the keyboard

```

INKEY$      ie;  A$ = INKEY$
             NOTE!  same as commodore GET$.

```

Inputs an ASCII character from the keyboard in a two byte form. The first byte is 0 while the second contains the ASCII value.

To get special functions keys you will have to determine the LEN of that string, that being two, and then work on the right\$. The cursor is set in the OFF state. Can be switched on by using the third parameter of LOCATE. 1 = ON 0 = OFF.

```

INPUT [;]["mess"];/,var,....

```

```

;           Line feed will be suppressed if
             added.

```

```

"mess"      A text message that prompts for an
             answer.

```

```

,           Suppresses the quotation mark.

```

```

var         Variable/s used to store values.
             ie; A  or a$ etc..

```

```

INPUTS$     ie;      A$ = INPUT$ (numchars)

```

Characters specified to be read from the keyboard.

```

numchars    Number of characters to be read from
             the keyboard.

```

```

LINE INPUT [;]["mess"];/,stringvar

```

One line read from the keyboard.

+-----+
| 174 |
+-----+

; Line feed is suppressed.
"mess" Text prompt.
, Suppress the question mark.
stringvar A single string variable input.

String handling

+ ie; A\$ = string1 + string2.....
Concatenates strings.

string Variable, constant or the result of
 a function.

Total length of characters is 255.

DATE\$ and TIME\$

A\$ = DATE\$ Variable set to current date.

DATE\$ = A\$ Date changed to value of A\$.

ie; DATE\$ = "mm-dd-yy" month, date, year.

Same procedure for TIME\$.

ie; TIME\$ = "hrs:min:sec[hsec]"

DEF FN stringnme(parm...) = function

FN stringnme(parm)

DEF FN is used to define a function of a string
while FN on its own is used to call that
function.

stringnme Name of a string variable when
 called.

parm One or more string\$.

function Parameter operation performed.

The DEF FN function must be defined first, then
called with FN.

FRE("")

Clears up the string storage area for garbage collection.

INSTR ([frompos],string,searchchar)

To search and return the position of a certain character from within a string.

frompos Numerical expression requesting the start position for the search to begin from within the STRING. If omitted, search begins from the first character.

string The string or element of a string array to be searched.

searchchar A character string of constant, variable or result.

LEFT\$ ie; A\$ = LEFT\$(string,numchars)

Returns the result of the leftmost part of a string.

string String or elements of an array to be dissected.

numchars Number of characters to be returned.

LEN ie; A\$ = LEN(string)

Returns the length of the whole string or array.

MID\$
ie; A\$ = MID\$(string,firstchar[,numchar])

Returns a number of characters beginning at a given position.

string A string or array to be dissected.

firstchar Number of characters to be returned.

MID\$ (deststring,firstchar[,numchars]) =
srcstring

Replace one section of a string with another.

deststring String or array to be replaced.

firstchar The destination character string
 where the source string is to be
 inserted.

numchar Number of destination characters to
 be replaced.

srcstring Start to replace source characters
 starting at the firstchar.

RIGHT\$ ie; A\$ = RIGHT\$(string,numchars)

Returns the rightmost number of characters of a
string.

string String or array to be dissected.

numchars The rightmost number of characters
 that are returned.

SPACE\$ ie; A\$ = SPACE\$(number)

Returns a string with the number of spaces as
specified.

number Specified number of spaces.

+-----+
| 178 |
+-----+

STRING\$ ie; A\$ = STRING\$(number,char)

Returns a specified number of single characters
in a string.

number Specified number of characters to be
repeated.

char A constant, variable or result of a
string function to be repeated.

Printer output -----

LCOPY (num)

This command is not included in all Basics. It prints the screen to the printer.

num Depending on Basic version, usually 0.

Some PC's have a program called GRAPHICS.COM which will allow a graphic screen to be printed when you press Shift-PrtSc. This must be called first. LCOPY only works in text mode.

LPOS ie; A = LPOS(printernum)

'A' is returned with the position of the buffer pointer.

printernum The interface number, 1,2 or 3.
Default is 1.

LPRINT[TAB(column)][SPC(num)][expression....]
[;][,]

This command will print your data to the printer.

column The printer tab position
(horizontally) in numerics and
must be set first, if required.

num The number of spaces.

expression Data to be printed which can be
constants, variables etc. Data
can be separated with a ';' (semicolon) to suppress line
feeds, thus allowing multiple expressions on
one line.

+-----+
| 180 |
+-----+

Data with a ',' (comma) forces a tab of 8 character positions.

The printer can be opened as a file for output.

```
ie; OPEN "LTPx:" FOR OUTPUT AS #1 : PRINT  
#1,A$
```

The device number (x) can be 1,2 or 3 and the file must be closed when you have finished with it.

LPRINT USING "mask";expression

Formats the output to the printer by using the "USING (mask)" expression. See Screen Output for 'mask' details.

expression The type of data to be printed.

WIDTH LPRINT numchars or
WIDTH "LPT printernum;",charnum

Used to set the number of characters to be printed.

numchars The number of characters to be
 printed before a linefeed given a
range from 0 to 255. The default is set to 80.

printernum Printer interface number,
 1,2 or 3.

Techniques in using the Interrupts

```
KEY keynum,CHR$(toggle/shift)+CHR$(scancode)
```

This command defines a key, or combination of keys to perform certain tasks by using the ON KEY statement.

keynum Specify a values from 15 to 20.

toggle/shift: Depends on the keyboard driver of your machine.

CTRL 04 (decimal) or &H04 (hexadecimal)

ALT 08 &H08

NUMLOCK 32 &H20

SHIFT 64 &H40

scancode This very much depends on the driver used.

When the key is depressed, a value (not ASCII) is given. It is those values you are looking at. See your computer manual.

If KEY (keynum) STOP is used, the routine is not executed for that 'keynumber'. However, the flag is set for the next KEY() ON and if encountered, that routine is executed.

KEY (keynum) STOP command is used to prevent an infinite loop.

Use RETURN at the end of the routine.

If you are using the ON ERROR GOTO command, error handling will suspend the key interrupt. The command RESUME will restore it.

To stop the key interrupt from re-enabling, use KEY() OFF for the KEY(keynumber) routine required.

ON COM channel GOSUB linenum COM channel
ON/OFF/STOP

If a byte is read during the interrupt, the serial interface will execute a routine.

channel A number of 1 to 4 depending
 on interfaces.

linenum The line number of the routine
 to be executed.

COM channel ON Switch on the interrupt
 checking.

COM channel OFF Switch off the interrupt
 checking.

COM channel STOP Stop checking until next
 channel ON command.

ON KEY(keynum) GOSUB linenum KEY(x) ON/OFF/STOP

This function reads the desired interrupt key and executes the routine given by the line number.

keynum

F1 to F10 1 to 10

Cursor up 11

Cursor left 12

Cursor right 13

Cursor down 14

linenum The line number of the routine
 that is to be executed when the
 key is depressed.

KEY(keynum) ON Switch on the interrupt for
 checking key/s.

KEY(keynum) OFF Switch off interrupt.

KEY(keynum) STOP Stops the interrupt checking
 until the next

 command of KEY(keynum) ON is found.

ON PLAY (num_notes) GOSUB linenum PLAY
ON/OFF/STOP

Checks the number of notes when then PLAY
buffer is checked. If the number of notes is
less or equal (<=) to num_notes, then the
appropriate routine is executed. Can be used
to play music in the background.

num_notes Total number of notes still
 remaining in the buffer which is
 to be executed by the routine
 required.

linenum The line number at which the
 routine starts.

PLAY ON Switches the interrupt on for
 checking.

PLAY OFF Switches the interrupt off.

PLAY STOP Switches the interrupt off until
 the next PLAY ON.

ON TIMER (seconds) GOSUB linenum TIMER
ON/OFF/STOP

Depending on the time specified and the result,
the interrupt is read and the routine starting
on (line number) is executed.

seconds A value of 1 to 86400 for which an
 interval is given so that a
 routine can be executed at a line
 number specified.

linenum The line number at which a routine
 is to be executed.

TIMER ON Switch on the interrupt for
 checking.

TIMER OFF Switch off the interrupt.

TIMER STOP Stops the interrupt checking
 until then next command of
 TIMER ON is found.

Machine Code

CALL [S] offset_var[,data_var]

This calls a machine code program which has been POKED into memory or BLOADED from a drive.

[S] CALLS is also known as CALL which places the current segment onto the stack in low byte/high byte form. Compilers are required to pass the segmented address.

offset_var The address at which the routine begins. This is a integer value which must be set with DEF SEG before called.

data_var Data or values to be processed are passed to the routine by variables. A 2 byte offset pointer of the contents of the variable is placed on the stack in low/high order and is processed from there.

DEF USR and USR ie; DEF USR number = offset

Called with USR number = [data_var]

DEF USR sets the offset variable while USR calls the machine code routine which has been POKED or BLOADED into memory.

number A value between 0 and 9 of which one or more routines is numbered. They are called by USR (required routine).

offset The offset address at which the routine starts. The segment must be defined with DEF SEG at the start.

+-----+
| 186 |
+-----+

data_var Variables whose contents are to be
 processed. The value is loaded into
the CPU registers:

AL contains: &H02 integer, &H03 string,
 &H04 single precision
 &H08 double precision.

BX contains: The offset pointer to the FAC
where the numerical value is stored.

DX contains:
A 3 byte area that contains the length of the
string in the first byte, and the offset
address of the string in the next 2 bytes.
These are in the form of a low/high.

Sound commands -----

BEEP	A single note is executed.
PLAY string	A set of notes is played depending on the string.
string	This can contain the following:
MF(mode foreground)	Notes to be in the foreground.
MB(mode background)	A maximum of 32 notes for background.
MN(mode normal)	Normal length notes. 7/8 of length as specified in the string by L.
ML(mode legato)	Smooth tones as specified in the string by L.
MS(mode staccato)	Short notes. 3/4 of length, specified in the string by L.
Ox(octave)	Set the octave. Range from 0 to 6.
>	Raise tone by an octave. Maximum of 6.
<	Lower tone by an octave. Minimum of 0.
A to G[-][+/#]	Musical tones from A to G. The + or # sign rises the tone by half (sharp). The - sign lowers it (flat).

N note_num One of 84 notes from within
 each seven octaves.

The range is 0 to 84. The 0 produces a reset.

P note_length This produces a rest between
 notes and can be range from 1
to 64. Each note length is 4 giving a quarter.

L note_length The length of notes. Range of
 1 to 64. If L with a
 parameter follows a note then
 it only applies to that note.

T tempo The number of quarter notes
 per minute. 32 to 255

. The full stop causes the tone
 to run one and a half times
 faster than the values set in
 L and T.

X string_var; Executes a designed string.
 ie; "T = Tempo;"

SOUND frequency, duration

frequency The pitch to be created.
 Range, 37 to 32767.

duration Length of tone measured in clock
 ticks. 0 to 65535.

Memory accessing

BLOAD filespec, offset

Loads a file into a given memory area.

filespec The [drive\path\] filename of the
file to be loaded.

offset The address as specified with the
DEF SEG command used at the
beginning of the program or
routine. Data is read until an
EOF character (&H1A) is read.

BSAVE filespec, offset, num_bytes

Saves a given area in memory to the diskette.

filespec The [drive\path\] file name to be
loaded.

offset The address to be saved as set by
DEF SEG.

num_bytes The number of bytes to be saved
from the offset.

DEF SEG = segment_address

Used for BSAVE, BLOAD, PEEK, POKE, VARPTR and
VARPTR\$.

segment_address The segment referred to
commands and functions
with the possible ranges
from 0 to 65535. If this is
not specified then the data
segment (DS) currently
occupied by Basic is set as

+-----+
| 190 |
+-----+

the default. Basic does not
check to see if the selected
segment is occupied.

PEEK ie; A = PEEK(offset)

This returns an 8 bit value of the specified
address (offset) in the current segment.

offset The address in the range of 0 to
65535.

POKE ie; A = POKE offset,value

This command will write a value into memory.

offset The memory address within the
segment. 0 to 65535

value A numeric value to be written to the
location. 0 to 255

VARPTR ie; A = VARPTR(variable) or
A = VARPTR(# file_num)

Determines the offset of a variable in a data
segment (DS).

variable A variable can be any type
including arrays, but must
be assigned first before this
function is called.

file_num The file number that has been
opened.

This returns the address of the first byte of
file control block (FBC) for sequential files.
Random files return the first byte of the FIELD
buffer.

+-----+
| 191 |
+-----+

VARPTR\$ ie; A\$ = VARPTR\$(variable)

The variable type and offset is determined by the variable in the data segment (DS). The first byte of the string contains the information of the type of variable:

CHR\$(2) is an integer

CHR\$(3) is a string

CHR\$(4) is single precision

CHR\$(8) is double precision

The second and third bytes contain the address of the first byte of the variable (low byte/high byte):

address = low byte + 256 * high byte

variable A variable can be any type including arrays, but must be assigned first before this function is called.

It is important to note, that garbage collection will change the addresses of the variables.

I/O Communications -----

GET# file_num,num_chars

Gets a specified number of characters from the serial interface into the communications buffer.

file_num The file number used in the OPEN command.

num_chars The number of characters to be read. Cannot be longer than the record length.

INP ie; A = INP (port) . Port value
 0 to 65535

Reads the current value of a control or data port.

LOC and LOF
 ie; A = LOC(filename) A = LOF(filename)

Return the number of characters in the transfer buffer that has been read.

filename The file number used in the OPEN command.

OPEN COM

OPEN "COM channel:[baud][parity][,word
 with[,stop bits][,RS]
 [,CS[time]][,DS[time]][,CD[time]],[BIN] [,ASC]
 [,LF]" [FOR mode] AS #[filename] [LEN=length]

This opens and initialises the serial port for communications.

channel The interface number. Normally 1.

baud rate To set the number of bits per
 second for data transfer.

Possible values are 75, 110, 150, 300, 600,
120, 1800, 2400, 4800, 9600

parity Data transfer parity. Normally
 set to N (no parity).

These abbreviations can be used:

N no parity

E even parity

U odd parity

S space

M mark

word width The number of bits per character.
 Default is 7.

stop bits The number of bits sent after
 each character.

Unless stated, this is normally set for 2 bits
if the baud rate is less or equal (<=) to
110.

RS This command suppresses the
 Request to Send (RTS)
 command during transmission.

CStime Waits for a Clear To Send (CTS)
 signal from the device.

DStime	Waits for the Data Set Ready (DSR) signal from the device.
CDtime	Waits for the Carrier Detect (CD) signal from the device.
BIN	The data received is to be treated as binary.
CR and LF and EOF are not interpreted, therefore not used.	
ASC	Received data as ASCII.
LF	A line feed command is to be sent after carriage return.
mode	Sets transfer mode: INPUT=receive OUTPUT=send
The FOR mode should be stated otherwise the data will be sent in at random, therefore allowing simultaneous input and output.	
file_num	The file number for input/output. 1 to 15 max.
length	The length of data to transfer. Default is 255 for INPUT and 128 bytes for OUTPUT.
OUT port,value	Writes a value to a control or data port.
port	A value of between 0 and 65535 for data to write.
value	A single value from 0 to 255.

It is suggested that you read up on I/O communications from other sources.

```
+-----+
| 195 |
+-----+
```

PUT# filename,numchars

Write a number of characters from the buffer to the interface.

Same parameters as GET. Data is written into the buffer using the commands PRINT#, WRITE# and PRINT# USING.

WAIT port,mask1,mask2

Stop a program until a given bit pattern is read from the port. This can create an

infinite loop which CTRL/C or CTRL/Break can not get out of.

The value read from the port is combined with the masks:

ie; read value XOR mask1 AND mask2

If the result is zero the port is read until the result is not equal to (<>) zero.

port A value between 0 and 65535.

mask1,mask2 Values used to make up a formula.

WIDTH "COM channel: ";num_chars

To set the number of characters sent before a CR is sent.

channel The serial port being addressed. Normally one is only installed, so the default is 1. There can be 1 to 4.

num_chars The number of characters to be sent.

ERROR MESSAGES

01 NEXT without FOR

As the program has ran it encountered a NEXT command that did not have FOR statement to execute it. ie; FOR I = 1 TO 100 : NEXT

02 Syntax error

A keyword that was entered has a typing error, was an illegal command not supported in Basic, simply forgot commas, semicolons or there are too many characters in a line.

03 RETURN without GOSUB

A RETURN command was encountered before first executing a GOSUB command. ie;

100 FOR I = 1 TO 100

110 RETURN

120 NEXT I

04 Out of data

The command READ attempted to read more data than was available in the DATA statements. ie;
100 FOR I = 1 TO 3 : READ A : NEXT

Rest of program here.

500 data 255, 13

05 Illegal function call

- * GET# or PUT# for a random access file is 0
- * Using a USR command without defining the DEF USR
- * A mathematical value was illegal
- * An illegal parameter was passed to a function
- * An array was indexed with a negative number

06 Overflow

The calculation performed was too large.

07 Out of memory

The program was too large to fit into memory, or that there is not enough memory for variables. Also, too many nested FOR/NEXTs WHILE/WENDs, and GOSUBS/RETURNS. Use FRE("") to perform garbage collection.

08 Undefined line number

A line number that is being referred to does not exist.

09 Subscript out of range

An array being addressed has not been defined with a DIM statement or that the array is larger than that defined DIM statement. Also if lower than the boundary set by OPTION BASE.

10 Duplicate definition

An array has already been dimensioned using the DIM() statement. To re-dimension, use ERASE first. OPTION BASE must be used at the very beginning of the program before the DIM statements.

11 Division by zero

An attempt was made to divide a value by 0.

12 Illegal direct

A program command was used in direct mode.

13 Type mismatch

Variables in a mathematical operation do not match.

14 Out of string space

An overflow has occurred because too many string operations have been used. Use garbage collection with FRE("").

15 String too long

A string (ie; A\$) or a combination of strings can not exceed 255 characters.

16 String formula too complex

A string operation has proved to complex for the Basic interpreter and therefore needs to be split into smaller operations.

17 Can't continue

If a program has stopped with the STOP command or has been changed in some way.

18 Undefined user function

DEF FN or DEF USR has not been defined before using a FN or USR function.

19 No RESUME

The error handling routine did not contain a RESUME command.

20 RESUME without error

Within the program line a RESUME was encountered and an error had not been encountered.

22 Missing operand

An operand within an operation is missing.

23 Line buffer overflow

A line number length was longer than 255 characters. You will have to split it up into 2 lines

24 Device timeout

An answer was not received from a device. Check that it is switched on, or connected. The error can be trapped with the ON ERROR GOTO command and use IF ERR = 24 THEN RESUME

25 Device fault

The device called for an operation does not exist.

26 FOR without NEXT

The NEXT command is missing from a FOR loop/s.

27 Out of paper

The printer has sent an "out of paper" error code.

29 WHILE without WEND

The WHILE operation has a missing WEND statement.

30 WEND without WHILE

The WEND statement has a missing WHILE operation.

50 FIELD overflow

A field was made larger than the actual file length as specified in the OPEN command.

51 Internal error

The Basics Interpreter has made an error within its self.

52 Bad file number

A file that you try to access has not been opened, or is out of range (1 - 15). It is possible that you have already opened the maximum range of files.

53 File not found

The file that you require can not be found. In most cases it is possible that you have omitted the drive\path.

54 Bad file mode

You tried to operate a device with the wrong command.

55 File already open

A file that you required to OPEN is already open. You should CLOSE # file_number after all operations.

57 Device I/O error

An error has occurred while an operation was being performed on and device. If it is the hard disk, then there could be a problem. If on the printer, the paper may have run out, or the machine has been switched off. If the floppy disk, then it may be that the diskette has not been formatted, or that it is write protected, or for some other reason.

58 File already exists

In changing the name of your file, you have tried to name it with the same name.

61 Disk full

There is no more space to store programs.

62 Input past end

While reading a file, you have gone past the end. Use the EOF operation to check for the end of file.

63 Bad record number

The requested record number is 0 or greater than 16,777,215

64 Bad file name

The file name used has either too many characters or illegal characters.

66 Direct statement in file

Basics require line numbers except most compilers. While the file was being loaded, it found a line number missing. To add line numbers use an ASCII editor (if saved with the ,A) or EDLIN.

67 Too many files

The directory of the disk has too many files. You will have to use another diskette.

68 Device unavailable

The device being accessed with an OPEN is nonexistent.

69 Communication buffer overflow

The buffer for the COM interface has exceeded its capacity.
You can set it using the Basic/C: buffer

70 Disk write protected

The diskette has a write protect tab on it. Depending on the operation, you may leave it on or take it off.

72 Disk media error

* The diskette has something physically wrong with it.

* The Read/Write head cannot be positioned correctly.

* Defective sectors have been found.

* The Read/Write head is dirty.

Use CHKDSK.COM to check out the diskette and also try to copy the files to another diskette.

74 Rename across disks

You cannot specify other drives while trying to rename a file.

+-----+
| 204 |
+-----+

75 Path/file access error

A given file or path contains an error.

76 Path not found

The path requested for a file is not correct

USEFUL TIPS

Special Keys::

Using PEEK and POKE.

Address &H40 with offset &H17.

ALT = 8 CTRL = 4 BREAK = 16

NUM/LOCK = 32 CAPS/LOCK = 64

INS = 128 Left SHIFT = 2 Right SHIFT = 1

Combinations of keys are:-

Left SHIFT + Right SHIFT = 3

CTRL + Left SHIFT = 6

CTRL + ALT = 12

CTRL + Right SHIFT = 5

ALT + Left SHIFT = 10

ALT + Right SHIFT = 9

INS + NUM/LOCK = 160

(INS first then NUM/LOCK = on, reverse = off)

NUM/LOCK + CAPS/LOCK = 96

NUM/LOCK + CAPS/LOCK + BREAK = 112

These are some of the combinations that can be had.

+-----+
| 206 |
+-----+

Monitor type -----

```
DEF SEG=0:A=PEEK(&H410) AND &H40:DEF SEG
```

'A' holds the value for the monitor type being used.

1 = 40 x 25 Colour

32 = 80 x 25 colour

48 = Mono

64 = both colour and mono.

As with the programs in this book, it is simpler to say:-

```
IF A = 48 THEN DEF SEG=&HB000 ELSE DEF  
SEG=&HB800
```

Memory (RAM):

```
DEF SEG=0:A%=PEEK(&H413) + (256*  
PEEK(&H414)):DEF SEG
```

A% now holds the amount of memory. You can now display the amount of actual memory by PRINT A% wherever you want on the screen.

DOS keyboard fonts:

If you have used the English keyboard set and want to return to the American keyboard, then press CTRL/ALT/F1 together.
To get back to the English keyboard press CTRL/ALT/F2 together.

Screen locations -----

For characters, the locations are EVEN numbers.
0,2,4,6 etc; while the colour for each
character is held in the odd numbers.
1,3,5,7 etc;. Therefore, a character in
position 0 will have it's colour in position 1.
A character in position 2 will have the colour
in 3 and so on. Remember to find out what
monitor is being used first and set a variable
to that value (as in the section 'monitor
type'.)

Type in this routine and see what I mean.
I am only going to do this for 80 coloum machines.

```
1 CLS:KEY OFF:DEF SEG=0:AA=PEEK(&H410) AND &H40
```

Clear the screen. Set to look at Monitor type.
Set the variable 'AA' to look at the monitoe type.
Now 'AA' holds a value.

```
2 IF AA=48 THEN DEF SEG=&HB000 ELSE DEF SEG
=&HB800
```

If the variable AA is 48 then it is a Mono screen
ELSE it is a colour one.

```
3 A=160:B=240:FOR I=0 TO 80 STEP 2
```

The variable 'A' is set for the last character
position of the screen. 'B' is set to last
character position in the screen line two. We do
a loop from 0 to 80 stepping over every ODD number.
Remember, ODDs are for colour attributes. Here we
want to poke onto the screen a character first.

```
4 POKE I,219:POKE 3840+1,219
```

We poke an Inverted space character (ASCII 219) onto

```
+-----+  
| 208 |  
+-----+
```

the screen every even number. We also Poke the same character onto the bottom line of the screen.

```
5 IF B<3770 THEN POKE A,219:POKE B,219:A=A+160:  
B=B+160
```

Providing we have not gone over a 40 character wide limit on the line we are poking onto, we also poke the same character vertically down the screen.

```
6 NEXT:A=161:B=241
```

The loop is repeated by the NEXT command. We then set the variable 'A' to 161, the colour attribute for character position 160. The variable 'B' is also set for attribute. We now have Poked a box of 40 characters wide and 23 characters deep with inverted spaces.

```
7 FOR J=1 TO 81 STEP 2
```

Now what is needed, is to get the box to flash colours onto the screen around that predefined box.

We set a loop to execute the colour attribute of each character within that box, stepping twice.

```
8 IF AA<>48 THEN S=RND(1)*15:GOTO 11
```

If we have a colour screen then we want to Randomise 15 whole numbers to get colours. As it is colour, we avoid the next two lines. These are for mono.

```
9 S=RND(1)*2:IF S=1 THEN S=0
```

We Randomise 2 whole numbers. If S=1 then we want the colour BLACK.

```
10 IF S=2 THEN S=7
```

The variable is set to WHITE if not BLACK

```
+-----+  
| 209 |  
+-----+
```

```
11 POKE J,S:POKE 3840+J,S
```

We poke the attribute colour onto the screen, both top and bottom of the box.

```
12 POKE A,S:POKE B,S:A=A+160:B=B+160
```

We also poke the colour attributes down both the sides of the box. 'A' is increased by one line as is 'B'.

```
14 IF B>=3920 THEN A=161:B=241
```

If 'B' is greater than or equals to 3920 the 'A' is made to ODD numbers, so to is the variable 'B'. This is for the colour attribute of the current character.

```
15 NEXT:GOTO 7
```

Once the loop has been completed, we go over the same ground to give us a great screen display.

+-----+
| 210 |
+-----+

Dis/Enable Keyboard

Disable keyboard (while doing a special
process)

DEF SEG=64:OUT 97,204

then Enable again with DEF SEG=64:OUT 97, 76

Check for various devices

6000 DEFINT A-Z:DEF SEG=&H40:KEY OFF:F\$=" no"

6001 MEMORY=PEEK(&H13)+PEEK(&H14)*256

6002 DISKS=(PEEK(&H10) AND &HC0)\&H40+1

6003 HARD=PEEK(&H75)

6004 PRINTERS=(PEEK(&H11)AND &HC0)\&H40

6005 SERIAL=(PEEK(&H11)AND &HE)\2

6006 GAMEPORT=(PEEK(&H11)AND &H10)\&H10

6007 IF(PEEK(&H10)AND &H30)=&H30 THEN MONO=-1

6008 DEF FNNUM\$(X)=MID\$(F\$+STR\$(X),-3*(X<>0)+1,4)

6009 CLS:COLOR 7,0

6010 PRINT "Floppy Drives: ";FNNUM\$(DISKS)

6011 IF HARD THEN PRINT "Hard Drives: ";FNNUM\$(HARD)
ELSE PRINT "No Hard Drives"

6012 IF PRINTERS THEN PRINT "Printer Ports: ";
FNNUM\$(PRINTERS) ELSE PRINT "No Printer Ports":
IF SERIAL THEN PRINT "You may have a Serial Printer"

+-----+
| 211 |
+-----+

6013 IF SERIAL THEN PRINT "Serial Ports: ";
FNNUM\$(SERIAL) ELSE PRINT "No serial Port"

6014 IF GAMEPORT THEN PRINT "A Games Port"
ELSE PRINT "No Games Port"

6015 IF MONO THEN PRINT "Monochrome "; ELSE PRINT
"Color/Graphics ";

6016 PRINT"Screen adaptor"

6017 COLOR 7,0:DEF SEG:END

USING THE COMMAND\$ STRING FOR QUICK BASIC COMPILERS -----

You can have a system, whereby the USER can make some selections from the DOS command line. This term is called SWITCHES. You see them quite often. You are normally given options to load a program in. ie; TEST /N/V and so on.

```
1 CLS:CLINE$=COMMAND$:TLENGTH=LEN(COMMAND$)
```

Clear the screen. Look at the COMMAND\$ line and set the TotalLENGTH variable to its total length.

```
2 MAXIMUM=(TLENGTH/2)+1
```

What we want is to be able to Dimension a string variable later on with the total switche commands used.

```
3 DIM ARG$(MAXIMUM)
```

We now dimesion the ARG\$ (argrement) string to the total of commands on the COMMAND line.

```
4 TRUE=-1:FALSE=0:I=1:NUMBER=0:INWORDS=TRUE
```

We now set up some variables to control the flow of the program

```
5 WHILE I<=TLENGTH
```

Using the WHILE/WEND commands, we state that, WHILE the variable 'I' is less than or equals to the TotalLength, we do the following:

```
6 CH$=MID$(CLINE$,I,1)
```

String CH\$ will be equals to MID\$ of the COMMAND line for each single character as long as:

+-----+
| 213 |
+-----+

7 IF CH\$<>" " THEN

If our CH\$ (Character Buildup String) is not equals to a SPACE, we do the following:

8 IF NOT INWORDS THEN INWORDS=TRUE

9 ARG\$(NUMBER)=ARG\$(NUMBER)+CH\$

The dimensioned array string is built up for each character that does not have a space after it. Otherwise;

10 ELSE IF INWORDS THEN

assuming that the variable INWORDS is true, we can;

11 NUMBER=NUMBER+1

increase the numbers of our ARGument\$.

12 INWORDS=FALSE

We know that a complete Switch command has been made, so we set out control variable to FALSE to tell us that we are ready for another.

14 END IF

We now END the multiple IF commands.

15 I=I+1

We increment the variable 'I' by 1, so that the WHILE command on line 5 will know what it is sposed to do.

16 WEND

We must use this command when using the WHILE command.


```
+-----+  
| 214 |  
+-----+
```

If you want to see what SWITCHES were used, add the following:

```
17 FOR i=0 to number:print arg$(i):next
```


ALPHABETICAL INDEX

Append	50
Command structures	147
Command Line Switches (Compilers)	95,212
Data handling	150
Dis/Enable Keyboard	210
DOS keyboard fonts	206
Editing keys	130
Error messages	196
Error trapping	142
File handling	132
Function keys	33
FUNKEY1.BAS	33
FUNKEY2.BAS	35
Getting input from the keyboard	173
Graphics	163
I/O communications	192
INDELLOT.BAS	13
Introduction	2
Let's Get Going	10

LOADER.BAS	21
Machine code	185
Manipulating Strings	10
Mathematics	153
Memory (RAM)	206
Memory accessing	189,206
Monitor type	206
ON ERROR and RESUME	49
Peek and Poke Grey keys	30
PK-BRD.BAS	30
Printer output	179-180
PRINTER.BAS	38
Read and Data commands	24
Reading and Saving data files	47
Screen locations	207-209
Screen output	156
Shortcut keys for the editor	129
Similar dos type commands in basic	127
Sound commands	187
Special keys	205
String handling	175

STRINGS.BAS	10
Techniques in using the interrupts	181
The direct command from DOS	118
The editor and its commands	120
TXT2COM.BAS	25
USEFUL TIPS	205
Using the printer	38
Variables and constants	144
WRAP1.BAS	52

FUNCTION INDEX

* Multiplication	153
+ Addition	153
+ Concatention	153
- Subtraction	153
/ Division	153
< Less than	150
<= Less than or Equal to	150
<> Not equal to	150
= Equal to	150
> Greater than	150
>= Greater than or Equal to	150
^ Exponentiation	153
ABS	153
AND	150,153
Arrays	144-146
deleting	144
dimensioning	144
ASC	151
ASCII characters	151
ATN	153
AUTO	120
BASIC	118
BASICA	118
BEEP	187
Binary files	189
BLOAD	189
BSAVE	189
CALL	185
CALL\$	185
CDBL	151
CHAIN	132
CHDIR	127
CHR\$	151
CINT	151
CIRCLE	163
CLEAR	120
Clearing Memory	124
CLOSE	132-133

```

+-----+
| 221 |
+-----+

```

CLS	156
COLOR	156-157,163-166
COMMON	133
Communications	62,192
buffer data (receive)	192
buffer data (send)	194
buffer functions	192
delay for port change	194
line length (COM)	193
opening channels	193-195
port value	192
writing to a port	194
Concatenation	175
Constants	144-146
CONT	120
Conversions	151
ASCII/decimal	151
decimal/ASCII	151
decimal/hex	152
decimal/octal	151
double-precision	151
number/string	151
single-precision	151
string/number	152
COS	153
CSNG	151
CSRLIN	157
Cursor:::	
column	157-158
location	158
position	158
CVD	133
CVI	133
CVS	133
Data:::	
comparisons	150-152
conversion	151-152
pointer	144
tables	144
DATE\$	175
DEF FN	151,175

DEF SEG	189,206
DEF USR	185
DEFDBL	144
DEFINT	144
DEFSGN	144
DEFSTR	144
DELETE	120
Deleting lines	120
DIM	144
Directory	127
change	127
creation	127
removal	127
Double-precision	151
DRAW	164
EDIT	121
Editing commands	121
auto-line numbers	120
renumber lines	124
ENVIRON	127
EVIRON\$	127
EOF	133
EQV	154
ERASE	144
ERDEV	142
ERDEV\$	142
ERL	142
ERR	142
ERROR	142
Error:::	
codes	142,196
device	142
generation	142
handling	142
numbers	142,196
trapping	142,142,196
Execution	
line number	120
program	124-125
Exiting GW-/BASIC	126
EXP	154

FIELD	134
FILES	121
append	50,137,138
closing	132
deleting	120
end-of-	133
formatting data	138
opening	137,138,180,192
position	134,136
random access	134,138,197
reading	134,145
renaming	132
size	138
writing	140,195
FIX	154
FN	151,175
FOR..NEXT	137,147
FRE("")	121
FRE(0)	121
Free memory	121
Function keys	121
Garbage collection	133,135,121
GET	134,166,192
GET#	134,192
Global data	144
GOSUB...RETURN	148
GOTO	121,148
Graphics:::	
circles	163
clearing points	169
coordinates	168
displaying	164
drawing	164-166
ellipses	163
graphic colours	163-164
lines	164-166,167
painting	168
point colour	169
rectangles	164-166
screen mode	168
setting points	169

storage	166
windows	172
BASIC/A/GW	118
Hardcopy	152
HEX\$	152
IF..THEN..ELSE	148
IMP	154
INKEY\$	173
INP	192
INPUT	173
INPUT#	134,135
INPUT\$	134-135,173
INSTR	176
INT	154
Integers	150
Interupts	181-184
communications	182
key definition	181
key reading	134,182
PLAY buffer	183
timer reading	112,128,184
IOCTL	128
IOCTL\$	128
Jumps:::	
conditional	121
unconditional	148
KEY	121-122
Keyboard input:::	
line programming	173
prompting	173
reading ASCII values	173-174
reading characters	145,173-174
KEY LIST	121-122
KEY OFF	121
KEY ON	121
KILL	135
LCOPY	179
LEFT\$	176
LEN	138,176
LET	144-145
LINE	135,167,173

LINE INPUT		135
LINE INPUT#		75,135
Line width		192
LIST		121-123
LLIST		122-123
LOAD		123
LOC		135,192
LOCATE		157-158
LOF		135,192
LOG		154
LPOS		179
LPRINT		179-180
LPRINT USING		180
LSET		136
Machine language:::		
calling		185
user-defined		185
Mathematics:::		
absolute values		153
addition	+	153
arctangent	ATN	153
cosine	COS	153
division	\ or /	153
equivalence	EQU	154
exclusive or	OR	151
exponent	EXP	154
exponentiation		153
implication	IMP	154
integer part	INT	154
logical and	AND	153
logical not	NOT	155
logical or	XOR	155
modular division	MOD	155
multiplication	*	153
natural logarithmn	LOG	154
sign	SGN	155
sine of angle	SIN	155
square root	SQR	155
subtraction	-	153
tangent of angle	TAN	155
truncation	FIX	154

Memory access	189-191
binary data loading	189
binary data saving	189
read memory	24,45,190,196
segment addresses	190
variable pointers	191
write to memory	190
MERGE (Merging programs)	124
MID\$	177-178
MKDIR	127
MKD\$	136
MKI\$	136
MKS\$	136
MOD	155
MS-DOS	127
drivers	128
environment table	127
programs	129
Music programming [Sound]	187-188
NAME	137
NEW	124
NOT	51,155
OCT\$	152
ON COM	182
ON ERROR GOTO	49,142
ON KEY	182-183
ON PLAY	182-183
ON TIMER	183
ON X GOTO	184
ON X GOSUB	184
OPEN	137,138,180,192-194
OPEN COM	192-194
OPTION BASE	145
OR	151
OUT	194
Overlays	132-139
PAINT	160
PEEK	190
PLAY	183
PMAP	168
POINT	169

POKE	190
POS	158
PRESET	169
PRINT	127,139,140,160
PRINT USING	160-162
PRINT#	140
PRINT# USING	140
Printer	179-180
buffer position	179
formatted data	180
line width	180
Program:::	
comment (REM,')	124
listing (printer)	122-123
listing (screen)	122
loops	147
PSET	169-170
PUT	179
PUT#	170,195
RANDOMIZE	154
Random numbers	154
READ	145
Reading memory	145
REM	124
RENUM	124
RESTORE	146
RESUME	49,143
RIGHT\$	177
RMDIR	127
RND	152
RSET	136
RUN	124-125
Running BASIC/A/GW	125
SAVE	125-126
SCREEN	158-159,171-172
Screen:::	
clearing	120
display	156
format	157-158
pages	156
Segment addresses	190-191

		+-----+
		228
		+-----+
SGN	155	
SHELL	128	
SIN	155	
Single-precision	153	
SOUND	188	
SPACE\$	177	
Spaces	162	
SQR	155	
Starting GW-/BASIC	124	
STOP	126	
STR\$	152	
String:::		
concatenation	175	
definition	175	
dissection	176	
functions	175	
searching	176	
STRING\$	178	
Subroutines	148	
SWAP	146	
SYSTEM	126	
System date	175	
System time	175	
TAN	155	
TIME\$	175	
TIMER	128,184	
Tone production	187-188	
Trace mode:::		
TROFF	126	
TRON	126	
User-defined functions	144,151-152,175	
USR	185	
VAL	152	
Variable:::		
content exchange	146	
initilization	144	
pointers	189-191	
VARPTR	190	
VARPTR\$	189-190	
VIEW	172	
VIEW PRINT	159	

+-----+
| 229 |
+-----+

WAIT	195
WHILE..WEND	149
WIDTH	180,195
WINDOW	159
WRITE	162
WRITE#	139
Writing to memory	190
XOR	155

How to protect your Copyright

After you have completed the program with its documentation and listings, send [via Registered Post] the documentation and listing to yourself and [optional] to a solicitor. Make sure the package is never tampered with, or that could make things very difficult to prove Copyright ownership in a court of law. There again, if you have the original listings of code it could be enough. I suggest that you do not leave things to chance.

Always place a full copyright notice within the first screen of the document file and [if possible] on the title page of the program.

EXAMPLE:

(C) Copyright Mr Book On Shelf. September 1987.
All rights reserved.

Place your name and FULL address, including Country. Add any conditions and restrictions for the use of the product or products.

Always state that the product must stay complete with documentation and all files as distributed. State: no copying, Sale, Point of Sale, Distribution, Passing around and so on. Decide what is relevant for you.

Although Copyright is De-facto in Great Britain, proving it is another matter. This is the reason as to why this information has been placed in this book.

+-----+
| 231 |
+-----+

ANSI Colour Graphics

Using ALT/27 [ESC] and left open square bracket [.

Foreground with Black Background

Black 0;30;40m Blue 0;34;40m Green 0;32;40m
Cyan 0;35;40m Red 0;36;40m Purple 0;31;40m
Yellow 0;33;40m White 0;37;40m

Highlighted

Blue 0;1;34;40m Green 0;1;32;40m Cyan 0;1;36;40m
Red 0;1;31;40m Purple 0;1;35;40m Yellow 0;1;33;40m
White 0;1;37;40m

Foreground Normal Flashing [47 is black background]

Black 0;5;30;47 Where 30 is, it is Foreground colour.
 Where 47 is, it is Background colour

Blue 34 Green 32 Cyan 33 Red 36 Purple 31
Yellow 33 White 37

Foreground Flashing Highlighted, 47= Black background

Gray 0;1;5;30;47 Where 30 is, it is foreground colour.
 Where 47 is, it is background colour.

Blue 34 Green 32 Cyan 36 Red 31 Purple 35
Yellow 33 White 37

Background Colours

Black=40 Blue=44 Green=42 Cyan=46 Red=41 Purple=45
Yellow 43 White 47

When printing a combination of characters, place a small 'm' in front of the very first character. If placing the cursor command just before the printed character, don't use 'm'. If you require to alter

```

+-----+
| 232 |
+-----+

```

the colour after the printed character/s, then issue ALT/27 [ESC] plus open square bracket with any colour code sequence.

EXAMPLE: ESC[0;37;40mThis is a testESC[0;33;40m

Command	ASCII	Dos Colour
Clear Home	ESC[2j	
Attributes OFF	ESC[0m	
Bold ON	ESC[1m	add 8
Underscore ON		
IBM MONO	ESC[4m	
Blink ON	ESC[5m	add 128
Reverse ON	ESC[7m	
Visable Cancel ON	ESC[8m	
Foreground		

Black	ESC[30m	0
Red	ESC[31m	4 [12=Bold]
Green	ESC[32m	2 [10=Bold]
Yellow	ESC[33m	6 [14=Bold]
Blue	ESC[34m	1 [9 =Bold]
Purple	ESC[35m	5 [13=Bold]
Cyan	ESC[36m	3 [11=Bold]
White	ESC[37m	7 [15=Bold]
Background		

Black	ESC[40m	
Red	ESC[41m	
Green	ESC[42m	
Yellow	ESC[43m	
Blue	ESC[44m	
Purple	ESC[45m	
Cyan	ESC[46m	
White	ESC[47m	

+-----+
| 233 |
+-----+

Cursor Movement

Cursor Right\left	ESC[1 to 79 C
Cursor Down\up	ESC[1 to 22 H
Cursor Home [top left]	ESC[H
Clear Screen and Cursor Home	ESC[2j
Cursor UP one	ESC[u

ASCII CODES

DEC	HEX	CHAR	NAME	Controle Codes
0	0		CTRL-@	NUL
1	1		CTRL-A	SOH
2	2		CTRL-B	STX
3	3		CTRL-C	ETX
4	4		CTRL-D	EOT
5	5		CTRL-E	ENQ
6	6		CTRL-F	ACK
7	7		CTRL-G	Bel
8	8		CTRL-H	BS
9	9		CTRL-I	HT
10	A		CTRL-J	LF
11	B		CTRL-K	VT
12	C		CTRL-L	FF
13	D		CTRL-M	CR
14	E		CTRL-N	SO
15	F		CTRL-O	SI
16	10		CTRL-P	DLE
17	11		CTRL-Q	DC1
18	12		CTRL-R	DC2
19	13		CTRL-S	DC3
20	14		CTRL-T	DC4
21	15		CTRL-U	NAK
22	16		CTRL-V	SYN
23	17		CTRL-W	ETB
24	18		CTRL-X	CAN
25	19		CTRL-Y	EM
26	1A		CTRL-Z	SUB
27	1B		CTRL-[ESC
28	1C		CTRL-\	FS
29	1D		CTRL-]	GS
30	1E		CTRL-^	RS
31	1F		CTRL--	US

+-----+
| 235 |
+-----+

DEC	HEX	CHAR	DEC	HEX	CHAR
32	20		71	47	G
33	21	!	72	48	H
34	22	"	73	49	I
35	23	#	74	4A	J
36	24	\$	75	4B	K
37	25	%	76	4C	L
38	26	&	77	4D	N
39	27	'	78	4E	M
40	28	(79	4F	O
41	29)	80	50	P
42	2A	*	81	51	Q
43	2B	+	82	52	R
44	2C	,	83	53	S
45	2D	-	84	54	T
46	2E	.	85	55	U
47	2F	/	86	56	V
48	30	0	87	57	W
49	31	1	88	58	X
50	32	2	89	59	Y
51	33	3	90	5A	Z
52	34	4	91	5B	[
53	35	5	92	5C	\
54	36	6	93	5D]
55	37	7	94	5E	^
56	38	8	95	5F	_
57	39	9	96	60	`
58	3A	:	97	61	a
59	3B	;	98	62	b
60	3C	<	99	63	c
61	3D	=	100	64	d
62	3E	>	101	65	e
63	3F	?	102	66	f
64	40	@	103	67	g
65	41	A	104	68	h
66	42	B	105	69	i
67	43	C	106	6A	j
68	44	D	107	6B	k
69	45	E	108	6C	l
70	46	F	109	6D	m

DEC	HEX	CHAR	DEC	HEX	CHAR
110	6E	n	149	95	—
111	6F	o	150	96	ù
112	70	p	151	97	ÿ
113	71	q	152	98	Ö
114	72	r	153	99	Ü
115	73	s	154	9A	ø
116	74	t	155	9B	£
117	75	u	156	9C	Ø
118	76	v	157	9D	×
119	77	w	158	9E	f
120	78	x	159	AF	á
121	79	y	160	A0	í
122	7A	z	161	A1	ó
123	7B	(162	A2	ú
124	7C	:	163	A3	ñ
125	7D	}	164	A4	Ñ
126	7E	~	165	A5	ª
127	7F		166	A6	º
128	80	Ç	167	A7	¿
129	81	ü	168	A8	®
130	82	é	169	A9	¬
131	83	â	170	AA	½
132	84	ä	171	AB	¼
133	85	à	172	AC	¡
134	86	å	173	AD	«
135	87	ç	174	AE	»
136	88	ê	175	AF	
137	89	ë	176	B0	
138	8A	è	177	B1	
139	8B	ï	178	B2	
140	8C	î	179	B3	
141	8D	ì	180	B4	Á
142	8E	Ä	181	B5	Â
143	8F	Å	182	B6	Ã
144	90	É	183	B7	©
145	91	æ	184	B8	
146	92	Æ	185	B9	
147	93	ô	186	BA	+
148	94	ö	187	BB	+

Continued

+-----+
| 237 |
+-----+

DEC	HEX	CHAR	DEC	HEX	CHAR
188	BC	+	222	DE	Î
189	BD	ç	223	DF	-
190	BE	¥	224	E0	Ó
191	BF	+	225	E1	ß
192	C0	+	226	E2	Ô
193	C1	-	227	E3	Õ
194	C2	-	228	E4	ö
195	C3	+	229	E5	Û
196	C4	-	230	E6	µ
197	C5	+	231	E7	þ
198	C6	ã	232	E8	ƒ
199	C7	Ã	233	E9	Ú
200	C8	+	234	EA	Û
201	C9	+	235	EB	Ü
202	CA	-	236	EC	Ý
203	CB	-	237	ED	Ÿ
204	CC		238	EE	-
205	CD	-	239	EF	˘
206	CE	+	240	F0	-
207	CF	¤	241	F1	±
208	D0	ö	242	F2	=
209	D1	Ð	243	F3	¾
210	D2	Ê	244	F4	¶
211	D3	Ë	245	F5	§
212	D4	È	246	F6	÷
213	D5	ì	247	F7	,
214	D6	Í	248	F8	°
215	D7	Î	249	F9	¨
216	D8	Ï	250	FA	•
217	D9	+	251	FB	1
218	DA	+	252	FC	3
219	DB		253	FD	2
220	DC	—	254	FE	
221	DD		255	FF	

EXTENDED Key Codes

SHIFT and	Dec	Hex	CTRL and	Dec	Hex
F1	84	54	F1	94	5E
F2	85	55	F2	95	5F
F3	86	56	F3	96	60
F4	87	57	F4	97	61
F5	88	58	F5	98	62
F6	89	59	F6	99	63
F7	90	5A	F7	100	64
F8	91	5B	F8	101	65
F9	92	5C	F9	102	66
F10	93	5D	F10	103	67
ALT and	Dec	Hex	F keys ONLY.	Dec	Hex
F1	104	68	F1	59	3B
F2	105	69	F2	60	3C
F3	106	6A	F3	61	3D
F4	107	6B	F4	62	3E
F5	108	6C	F5	63	3F
F6	109	6D	F6	64	40
F7	110	6E	F7	65	41
F8	111	6F	F8	66	42
F9	112	70	F9	67	43
F10	113	71	F10	68	44
CTRL and	Dec	Hex	ALT and	Dec	Hex
A	1	1	A	30	1E
B	2	2	B	48	30
C	3	3	C	46	2E
D	4	4	D	32	20
E	5	5	E	18	12
F	6	6	F	33	21
G	7	7	G	34	22
H	8	8	H	35	23
I	9	9	I	23	17
J	10	A	J	36	24

Continued

```
+-----+
| 239 |
+-----+
```

CTRL and	Dec	Hex	Alt and	Dec	Hex
K	11	B	K	37	25
L	12	C	L	38	26
M	13	D	M	50	32
N	14	E	N	49	31
O	15	F	O	24	18
P	16	10	P	25	19
Q	17	11	Q	16	10
R	18	12	R	19	13
S	19	13	S	31	1F
T	20	14	T	20	14
U	21	15	U	22	16
V	22	16	V	47	2F
W	23	17	W	17	11
X	24	18	X	45	2D
Y	25	19	Y	21	15
Z	25	1A	Z	44	2C

Standard keys:

Key	Dec	Hex
Home	71	47
End	79	4F

Special Keys.

```
DEFSEG=64
A=PEEK(64)
DEF SEG=0
```

A= VALUE OF KEY

		OFF	ON	Hex
PgUp	73	49		
PgDn	81	51		
Ins	82	52		
.Del	83	53		
<-Del	8	8		
Tab >	9	9		
Tab <	15	F		
Cur Up	72	48		
Cur Dn	80	50		
Cur Lft	75	4B		
Cur Rgt	77	4D		
CTRL LOCK	0	64	40	
NUM LOCK	0	32	20	
SCROLL LOCK	0	16	10	
INS	0	128	80	
CTRL	0	4	4	
ALT	0	8	8	
LEFT SHIFT	0	2	2	
RIGHT SHIFT	0	1	1	

Variable A contains value.