

Brief notes on C

Contents

Masking (Bitwise Operators)	2
GRAPHICS	2
RS232 communications using BIOSCOM	7
switch / case Construct	10
Functions	10
Arrays	11
POINTERS	12
Glossary	13

Masking (Bitwise Operators)

C unlike most other high level languages allows you to inspect individual bits of a piece of data. For this it uses the bitwise operators. On this module we have referred to this process as MASKING. Using the logic of the truth table individual bits can be tested.

The logic for this is that using the truth table formula for 'AND' a one plus a one always equals one any other combination is equal to zero.

GRAPHICS

Turbo C programs can either display information in a text mode (usually 80 character column by 25 rows) or graphics mode. The resolution of the graphics mode can vary from 640 pixels on the y axis and 480 on the x axis of the screen to 1024 x 768. The more pixels displayed the greater the detail of the image. In order to use this graphic capability in DOS based programs you will need to do three things:

1. Use the graphics.h header file in your program (this is already include in rack.h and therefore should not need to be 'included' separately).
2. Detect and set the appropriate graphics software driver for the screen resolution. Again this is done in rack.h and is called by you using the function call SetGraph();. However, if you are not using rack.h you must detect and set the graphics mode yourself, the code required for this should be given in a user function and is:

```
/* request auto detection */
int gdriver = DETECT, gmode, errorcode;

/* initialize graphics and local variables */
initgraph(&gdriver, &gmode, "");

/* read result of initialization */
errorcode = graphresult();
/* an error occurred */
if (errorcode != grOk)
{
    printf("Graphics error: %s\n",grapherrormsg(errorcode));
    printf("Press any key to halt:");
    getch();
    exit(1);
}
```

3. Include in the directory where your executable file is stored (the one with the .exe extension) the appropriate Borland graphics driver file, usually EGAVGA.BGI. Failure to do this will

result in the error message:

Graphics error: Device driver file not found (EGAVGA.BGI)

The following functions compose the Borland Graphics Interface and are available only for 16-bit DOS applications. Use them to create onscreen graphics with text. They are defined in graphics.h unless indicated otherwise.

A	G	I	S
arc	getarccoords	imagesize	sector
	getaspectratio	initgraph	setactivepage
B	getbkcolor	installuserdriver	setallpalette
bar	getcolor	installuserfont	setaspectratio
bar3d	getdefaultpalette		setbkcolor
	getdrivername	L	setcolor
C	getfillpattern	line	_setcursortype
circle	getfillsettings	linerel	(conio.h)
cleardevice	getgraphmode	lineto	setfillpattern
clearviewport	getimage		setfillstyle
closegraph	getlinesettings	M	setgraphbufsize
	getmaxcolor	moverel	setgraphmode
D	getmaxmode	moveto	setlinestyle
detectgraph	getmaxx		setpalette
drawpoly	getmaxy	O	setrgbpalette
	getmodename	outtext	settextjustify
E	getmoderange	outtextxy	settextstyle
ellipse	getpalette		setusercharsize
	getpalettesize	P	setviewport
F	getpixel	pieslice	setvisualpage
fillellipse	gettextsettings	putimage	setwritemode
fillpoly	getviewsettings	putpixel	
floodfill	getx		T
	gety	R	textheight
	graphdefaults	rectangle	textwidth
	grapherrormsg	registerbgidriver	
	_graphfreemem	registerbgifont	
	_graphgetmem	restorecrtmode	
	graphresult		

circle Syntax

```
#include <graphics.h>
void far circle(int x, int y, int radius);
```

Description: circle draws a circle in the current drawing color with its center at (x,y) and the radius given by radius. The linestyle parameter does not affect arcs, circles, ellipses, or pie slices. Only the thickness parameter is used. If your circles are not perfectly round, adjust the aspect ratio.

Return Value: None.

cleardevice Syntax

```
#include <graphics.h>
void far cleardevice(void);
```

Description: cleardevice erases (that is, fills with the current background color) the entire graphics screen and moves the CP (current position) to home (0,0).

Return Value: None.

drawpoly Syntax

```
#include <graphics.h>
void far drawpoly(int numpoints, int far *polypoints);
```

Description: drawpoly draws a polygon with numpoints points, using the current line style and color. *polypoints points to a sequence of (numpoints * 2) integers. Each pair of integers gives the x and y coordinates of a point on the polygon. In order to draw a closed figure with n vertices, you must pass n + 1 coordinates to drawpoly where the nth coordinate is equal to the 0th.

Return Value: None.

fillpoly Syntax

```
#include <graphics.h>
void far fillpoly(int numpoints, int far *polypoints);
```

Description: fillpoly draws the outline of a polygon with numpoints points in the current line style and color (just as drawpoly does), then fills the polygon using the current fill pattern and fill color. polypoints points to a sequence of (numpoints * 2) integers. Each pair of integers gives the x and y coordinates of a point on the polygon.

Return Value: None.

lineto Syntax

```
#include <graphics.h>
void far lineto(int x, int y);
```

Description: lineto draws a line from the CP to (x,y) , then moves the CP to (x,y) .

Return Value: None.

line Syntax

```
#include <graphics.h>
void far line(int x1, int y1, int x2, int y2);
```

Description: line draws a line in the current color, using the current line style and thickness between the two points specified, (x1,y1) and (x2,y2), without updating the current position (CP).

Return Value: None.

moveto Syntax

```
#include <graphics.h>
void far moveto(int x, int y);
```

Description: moveto moves the current position (CP) to viewport position (x,y).

Return Value: None.

outtext Syntax

```
#include <graphics.h>
void far outtext(char far *textstring);
```

Description: outtext displays a text string in the viewport, using the current font, direction, and size. outtext outputs textstring at the current position (CP). If the horizontal text justification is LEFT_TEXT and the text direction is HORIZ_DIR, the CP's x-coordinate is advanced by textwidth(textstring). Otherwise, the CP remains unchanged. To maintain code compatibility when using several fonts, use textwidth and textheight to determine the dimensions of the string.

If a string is printed with the default font using outtext, any part of the string that extends outside the current viewport is truncated. outtext is for use in graphics mode; it will not work in text mode.

Return Value: None.

outtextxy Syntax

```
#include <graphics.h>
void far outtextxy(int x, int y, char far *textstring);
```

Description: outtextxy displays a text string in the viewport at the given position (x, y), using the current justification settings and the current font, direction, and size. To maintain code compatibility when using several fonts, use textwidth and textheight to determine the dimensions of the string. If a string is printed with the default font using outtext or outtextxy, any part of the string that extends outside the current viewport is truncated. outtextxy is for use in graphics mode; it will not work in text mode.

Return Value: None.

setbkcolor Syntax

```
#include <graphics.h>
void far setbkcolor(int color);
```

Description: setbkcolor sets the background to the color specified by color. The argument color can be a name or a number as listed below: (These symbolic names defined in graphics.h.)

Number	Name	Number	Name
0	BLACK	8	DARKGRAY
1	BLUE	9	LIGHTBLUE
2	GREEN	10	LIGHTGREEN
3	CYAN	11	LIGHTCYAN
4	RED	12	LIGHTRED
5	MAGENTA	13	LIGHTMAGENTA
6	BROWN	14	YELLOW
7	LIGHTGRAY	15	WHITE

For example, if you want to set the background color to blue, you can call

```
setbkcolor(BLUE) /* or */ setbkcolor(1)
```

On CGA and EGA systems, setbkcolor changes the background color by changing the first entry in the palette. If you use an EGA or a VGA, and you change the palette colors with setpalette or setallpalette, the defined symbolic constants might not give you the correct color. This is because the parameter to setbkcolor indicates the entry number in the current palette rather than a specific color (unless the parameter passed is 0, which always sets the background color to black).

Return Value: None.

setcolor Syntax

```
#include <graphics.h>
void far setcolor(int color);
```

Description: setcolor sets the current drawing color to color, which can range from 0 to getmaxcolor. The current drawing color is the value to which pixels are set when lines, and so on are drawn. The drawing colors shown below are available for the CGA and EGA, respectively.

Palette number

0	CGA_LIGHTGREEN	CGA_LIGHTRED	CGA_YELLOW
1	CGA_LIGHTCYAN	CGA_LIGHTMAGENTA	CGA_WHITE
2	CGA_GREEN	CGA_RED	CGA_BROWN
3	CGA_CYAN	CGA_MAGENTA	CGA_LIGHTGRAY

Constant assigned to color number (pixel value): 1 2 3

Number	Name	Number	Name
0	BLACK	8	DARKGRAY
1	BLUE	9	LIGHTBLUE
2	GREEN	10	LIGHTGREEN
3	CYAN	11	LIGHTCYAN
4	RED	12	LIGHTRED
5	MAGENTA	13	LIGHTMAGENTA
6	BROWN	14	YELLOW
7	LIGHTGRAY	15	WHITE

You select a drawing color by passing either the color number itself or the equivalent symbolic name to `setcolor`. For example, in `CGAC0` mode, the palette contains four colors: the background color, light green, light red, and yellow. In this mode, either `setcolor(3)` or `setcolor(CGA_YELLOW)` selects a drawing color of yellow.

Return Value: None.

RS232 communications using BIOSCOM

bioscom Syntax

```
#include <bios.h>
int bioscom(int cmd, char abyte, int port);
```

Description: `bioscom` performs various RS-232 communications over the I/O port given in `port`. A port value of 0 corresponds to COM1, 1 corresponds to COM2, and so forth. The value of `cmd` can be one of the following:

Value	Meaning
0	Sets the communications parameters to the value in <code>abyte</code> .
1	Sends the character in <code>abyte</code> out over the communications line.
2	Receives a character from the communications line.
3	Returns the current status of the communications port.

`abyte` is a combination of the following bits (one value is selected from each of the groups):

0x02	7 data bits	0x00	110 baud
0x03	8 data bits	0x20	150 baud
		0x40	300 baud
0x00	1 stop bit	0x60	600 baud
0x04	2 stop bits	0x80	1200 baud
0x00	No parity	0xA0	2400 baud

0x08	Odd parity	0xC0	4800 baud
0x18	Even parity	0xE0	9600 baud

For example, a value of 0xEB (0xE0|0x08|0x00|0x03) for a byte sets the communications port to 9600 baud, odd parity, 1 stop bit, and 8 data bits. bioscom uses the BIOS 0x14 interrupt.

Return Value: For all values of cmd, bioscom returns a 16-bit integer, of which the upper 8 bits are status bits and the lower 8 bits vary, depending on the value of cmd. The upper bits of the return value are defined as follows:

- Bit 15 Time out
- Bit 14 Transmit shift register empty
- Bit 13 Transmit holding register empty
- Bit 12 Break detect
- Bit 11 Framing error
- Bit 10 Parity error
- Bit 9 Overrun error
- Bit 8 Data ready

If the abyte value could not be sent, bit 15 is set to 1. Otherwise, the remaining upper and lower bits are appropriately set. For example, if a framing error has occurred, bit 11 is set to 1.

With a cmd value of 2, the byte read is in the lower bits of the return value if there is no error. If an error occurs, at least one of the upper bits is set to 1. If no upper bits are set to 1, the byte was received without error.

With a cmd value of 0 or 3, the return value has the upper bits set as defined, and the lower bits are defined as follows:

- Bit 7 Received line signal detect
- Bit 6 Ring indicator
- Bit 5 Data set ready
- Bit 4 Clear to send
- Bit 3 Change in receive line signal detector
- Bit 2 Trailing edge ring detector
- Bit 1 Change in data set ready
- Bit 0 Change in clear to send

/* bioscom function information for the RS-232 communications link.

syntax -> int bioscom(int cmd, char abyte, int port);

Prototype in bios.h

cmd values are as follows:-

0 set communication parameter to abyte.

1 send abyte out.

2 receive a char (in low bits of return value).

3 return status.

port is 0 for COM1, 1 for COM2, etc.

Upper 8 bits of return value are status bits.

Lower 8 bits depends on the cmd specified.

*/

```
#include <stdio.h>
```

```
#include <bios.h>
```

```
#include <conio.h>
```

```
#define COMPORT 0
```

```
#define DATA_READY 0x100
```

```
#define SETTINGS ( 0x80 | 0x02 | 0x00 | 0x00)
```

```
void main(void)
```

```
{
```

```
int in, out, status, DONE = FALSE;
```

```
bioscom(0, SETTINGS, COMPORT);
```

```
cprintf("... BIOSCOM [ESC] to exit ...\n");
```

```
while (in != 0x1B)
```

```
{
```

```
status = bioscom(3, 0, COMPORT);
```

```
if (status & DATA_READY)
```

```
if ((out = bioscom(2, 0, COMPORT) & 0x7F) != 0)
```

```
putch(out);
```

```
if (kbhit())
```

```
{
```

```
if ((in = getch()) == '\x1B')
```

```
{
```

```
printf("\n\nexiting program;
```

```
printf("\n\npress any key to end");
```

```
getch();
```

```
}
```

```
bioscom(1, in, COMPORT);
```

```
}
```

}
}

switch / case Construct

By now you have discovered the various way of control the flow of your program, it doesn't have to be sequential. You have used various statements to control the flow such as 'while', 'do / while', 'for' and 'if'. You have learnt that the 'if' construct can come in various flavors, 'if', 'if/else' or 'if / else if' and that these can be nested (one put inside the other), indeed this is why you have 'if/else' or 'if/ else if'. These nested statements can sometimes be difficult to read and control can be ambiguous. To simplify the interpretation and provide a more efficient process we have introduced the 'switch/ case' construct.

The 'switch / case' construct is described on pages 35 & 36 of the module text book, "Learning to program in C". It shows the 'switch' statement evaluating a character variable which is then used to select (using the 'case' statement) a printf statement. Note that in the example referenced (pg 36, Learning to program in C), as a character is being evaluated, single quotes are required around the matching value. These quotes are NOT required if an integer variable were to be evaluated.

The SWITCH / CASE construct works by performing code that matches the value of a variable:

```
switch(x)
{
    case 1:do this
...;
        break;
    case 2:do this
...;
        break;
    case 3:do this
...;
        break;
    default:do this
..;
}
```

This function looks at the value of 'x' and depending upon its value will execute the code where the 'case' statement matches that value.

'break' is used after each section of code to allow the program to continue its normal flow without executing other sections within the 'switch' function.

The 'default' option can be used to identify any condition which does not match those stated.

The switch/ case constructs provides a more efficient method of selecting from multiple choice, a digital joystick input for example, and should be use in preference to nested or compound 'if' statements.

Functions

A function is a section of code that is used by its name. It can be used many times within a program and therefore save the need to repeat the same code. All 'C' programs consist of at least one function called *main()*. Functions are identified by having parenthesis following them. They allow data to be passed into them e.g. `UsersAge(age);`

or return data from them e.g.

```
PassorFail = CalculateMarks( );
```

Ideally the *main()* function should be short and the detail code reside in other functions.

```
void main(void)
{
    /* Short - maximum 1 page
       English like - self explanatory
       High level of abstraction    */
}
```

A function name consists of 3 parts:

```
    A      B      C
void FunctionName (void)
```

- A) Represents any data that is returned from the function as in the case of `getch()` which returns the value of the key pressed to a variable assigned.
- B) Is the function name, it should NOT have any spaces in it and should be meaningful.
- C) Represents any data that is sent to (or passed to) the function as in the case of `UsersAge(age)` where the value of 'age' is being sent to the function to be used by it.

Arrays

An array is a collection of variables given one name and selected by a subscript.

Example:

/ a program to record the reading from wind measurement equipment, placing the results in an array and the calculating the average.*/*

```
void main(void)
{
    int ws[3600], i, sum, mean;

    for (i=0; i<=3600; i++)
    {
        ws[i] = ReadWind();
        delay(1000);
    }
    for (i=0; i<3600; i++)
```

```

    {
        sum=sum+ws[i]; /*can also be expressed as sum+=ws[i]*/
    }
    mean=sum/3600;
} /*-----END -----*/

```

The above is an example of a single dimensional array, Arrays can also be two, three, four, five, etc dimensional. A two dimensional array forms rows and columns, similar to a table. This is achieved by having two subscripts, one for the rows and a second for the columns. (Tip: to remember the order ie rows then columns think of - Roman Catholic or Rugby Club or Rhubarb & Custard).

Example 1:

```
int Block-of-Flats [5] [10]; /* 5 rows by 10 columns*/
```

The location of the top left corner of the table is 0, 0.

Example 2:

```
char cOxo [3] [3];
printf("%c \n",cOxo [n] [n]);
```

POINTERS

Variables so far have been used to hold data, the use of variable names give access to contents of storage reserved for variable value. Where as Pointers are variables that hold addresses to gives access to the memory address of a variable.

When using pointers you must remember 3 things:

1. Pointers are variables;
2. They hold the memory address of other variables;
3. They must be of the same type as the variable to which they point.

They have three main uses:

1. To allow variable parameters passed to functions.
2. To create dynamic data structures; data structures built up from blocks of memory allocated from the heap at run-time.
3. To access data in arrays.

Pointer are declared like any type of declaration, but includes *

```
int *aptr; /* reserve storage for pointer to integer*/
char *string; /* reserve storage for pointer to character*/
```

Assigning addresses to pointers uses the Address Operator (&):

```
int a;
int *aptr;
```

```
aptr = &a; /* sets aptr to address of variable 'a'*/
```

To access the address stored in a pointer de-reference operator (*) is used:

```
char c1 = 'z', c2;  
char *cptr;
```

```
cptr = &c1;  
c2 = *cptr; /* 'c2' stores the value pointed to by 'cptr'*/
```

Glossary

Address

Reference to a memory location. In C pointers are used to hold addresses.

ANSI

American National Standards Institute

The United States government body responsible for approving US standards in many areas, including computers and communications. ANSI is a member of ISO.

Argument

A value passed to a function (see parameter).

Array

A collection of identically typed data items distinguished by their indices (or "subscripts"). A reference to an array element is written something like A[j] where A is the array name and j is the index. Arrays are appropriate for storing data which must be accessed in an unpredictable order, in contrast to lists which are best when accessed sequentially.

ASCII

American Standard Code for Information Interchange, a coding scheme which permits a standard letter-to-number encoding system for letters and numbers of American English. ASCII makes it practical for computers to work with text instead of only numbers.

Bit

Binary digit. In the binary numbering system, every digit is a 0 or a 1. Each digit is a bit. Computers store information by encoding it into sequences of bits. A very small piece of litter.

Block

A sequence of definitions, declarations and statements, enclosed within braces {}.

Boolean

The type of an expression with two possible values, "true" and "false". Also, a variable of Boolean type or a function with Boolean arguments or result. The most common Boolean functions are AND, OR and NOT.

Byte

A binary number 8 digits (8 bits) long. A byte is the smallest convenient entity manipulated in a computer. One character of text consumes one byte in the computer's storage.

C

A programming language designed by Dennis Ritchie at AT&T Bell Laboratories in 1972 for systems programming on the PDP-11 and immediately used to reimplement Unix.

It was called "C" because many features derived from an earlier compiler named "B". In fact, C was briefly named "NB". B was itself strongly influenced by BCPL. Before Bjarne Stroustrup settled the question by designing C++, there was a humorous debate over whether C's successor should be named "D" or "P" (following B and C in "BCPL").

Partly due to its distribution with Unix, C became immensely popular outside Bell Labs after about 1980 and is now the dominant language in systems and microcomputer applications programming. It has grown popular due to its simplicity, efficiency, and flexibility. C programs are often easily adapted to new environments. Ritchie's original C, known as K&R C after Kernighan and Ritchie's book, has been standardised (and simultaneously modified) as ANSI C.

Character Array.

A set of elements of type char. (Can be used to store a string).

Cohesion

A program exhibits cohesion when it has modules that perform only one self-contained set of operations, leaving unrelated tasks to other modules.

Compilation error

Error which occurs during the translation of source code into machine code.

Compiler

A program which converts source code into machine code.

Compound Condition

An IF statement in which there are two or more conditions being tested; each condition is separated by the word OR or AND.

Compound Statement

A sequence of simple statements.

Constant

An item that represents a value that cannot be changed. For Example: 123, 'x'

Constant (symbolic)

A symbol defined in a #define preprocessor directive to represent a constant value.

Construct

A structure of code to perform a particular programming task such as a loop or decision.

Coupling

The degree to which components depend on one another. There are two types of coupling, "tight" and "loose". Loose coupling is desirable for good software engineering but tight coupling may be necessary for maximum performance. Coupling is increased when the data exchanged between components becomes larger or more complex.

Data type

Definition of the data. int, char, float.

Declaration

A statement that 'declares' the existence a construct and associated attributes.

Variable declarations state the type of data to be stored and provides a user, data or identifier name which is attached to the storage location for reference by the programmer.

```
int a;  
char c;
```

Function prototype is the declaration of a function used within the program. It states the return type (if any), the name by which the function is called and the parameters (if any).

```
int WindSpeed(int mph, Time time);
```

Definition

A construct which specifies the name, parameters and return type of a function. For example a function definition would be:

```
long Sqr(int num){  
  
    num*=num;  
    return(num);  
}
```

Desk checking

A method of debugging programs by manually checking for typographic, keying and other errors prior to compiling. This method of debugging reduces computer time.

Echo

The display on the PC/terminal screen of characters typed in via the keyboard.

Escape sequence

Control codes comprising combinations of a backslash followed by letters or digits which represent non printing characters.

Executable program

Program which will run in the environment of the operating system or within an appropriate run time environment.

Executable (stand-alone) program

Program which will run within the environment of the operating system without additional utilities or support.

Expression

A sequence of operators and operands which may return a single value.

File

Data stored as an electronic file.

File descriptor

This is used in low level I/O (open/read/write/close functions) to identify a file. It is an integer number assigned to a file name by open and then used as a unique identifier by read/write and close.

Floating-point Number

Number having a decimal place or exponent.

Format specification

A string which controls how input or output shall be presented.

Hexadecimal

A numbering system using a base of 16 and the digits 0 to 8 and A,B,C,D,E,F.

Identifier

The names used to refer to stored data values such as constants, variables or functions.

ISO

International Organization for Standardization; A voluntary organisation founded in 1946, responsible for creating international standards in many areas, including computers and communications.

Its members are the national standards organisations of 89 countries, including the American National Standards Institute.

Integer

A number without a fractional part.

Iteration

Repeating a section of code more than once. Done by use of a loop. There are 3 loop constructs in C; for(.....), while(.....), and do{.....}while(.....).

Keyword

A word which has a predefined meaning to a 'C' compiler and therefore must not be used for any other purpose.

Library file

The file which contains compiled versions of commonly used functions which can be linked to an

object file to make an executable program.

Library function

A function whose code is contained in the external library file.

Literal

Characters, letters or strings which are to be taken literally and used as constants rather than identifiers.

Object Code

Code which is directly understandable by the machine (machine code).

Operand

An expression acted on by an operator. For example: $z = a + b$;

a and b are both operands of the + operator.

Parameter

A value received by a function.

pass by value/reference

All variables have two parts: a memory address and a value. When you pass by value you're passing that variable's value. When you pass by reference you're passing that variable's memory address.

pixel

Pixel stands for picture element, this represents a single dot on the screen that goes to make up a whole character or image. The number of pixels or dots per square inch on the screen is known as resolution.

Pointer

Variable containing an address.

Precedence (of operators)

The order in which operators are dealt with during the evaluation of an expression.

Preprocessor

A processor which manipulates the initial directives of the source file which contains instructions about how the source file shall be processed and compiled.

Preprocessor directive

Source file instruction about how the file shall be processed and compiled.

Program

A text file comprising code which can be compiled.

Radix

Or "base", "number base". In a positional representation of numbers, that integer by which the significance of one digit place must be multiplied to give the significance of the next higher digit place. Conventional decimal numbers are radix ten, binary numbers are radix two.

Resolution

see PIXEL.

Run time error

An error which occurs when a program is executed.

Reserved word

A word which has a predefined meaning to a 'C' compiler and therefore must not be used for any other purpose.

Selection

A method of 'selecting' a route through a program when a choice has been given. The main constructs that exist for selection in C are; if(.....), if(.....){.....}else{.....}, if(.....){.....}else if(.....){...} and switch(.....){case}.

Sequence

The execution of one command after another in a program.

Scope

The duration or life that a variable or data object exists in memory within a program. For example global variables retain their storage space until the end of the program even if they are not used. Where as, local variables, those declared within a function are only retained while that function is in use.

Source code

A text file comprising code which can be compiled.

Statement

A simple statement is an expression followed by a semicolon. (See compound statement and block).

String

A string in 'C' is an array of characters terminated by a Null character ('\0').

Syntax error

A mistake in the source code which prevents the compiler from converting it into object code.

Variable

An identifier (and storage) for a data type and for which the data value is allowed to change as the

program runs.

Appendix 1

ASCII Table

Dec	ASCII	Dec	ASCII	Dec	ASCII	Dec	ASCII
32	space	56	8	80	P	104	h
33	!	57	9	81	Q	105	i
34	“	58	:	82	R	106	j
35	#	59	;	83	S	107	k
36	\$	60	<	84	T	108	l
37	%	61	=	85	U	109	m
38	&	62	>	86	V	110	n
39	‘	63	?	87	W	111	o
40	(64		88	X	112	p
41)	65	A	89	Y	113	q
42	*	66	B	90	Z	114	r
43	+	67	C	91	[115	s
44	,	68	D	92	\	116	t
45	-	69	E	93]	117	u
46	.	70	F	94	^	118	v
47	/	71	G	95	_	119	w
48	0	72	H	96	`	120	x
49	1	73	I	97	a	121	y
50	2	74	J	98	b	122	z
51	3	75	K	99	c	123	{
52	4	76	L	100	d	124	
53	5	77	M	101	e	125	}
54	6	78	N	102	f	126	~
55	7	79	O	103	g	127	DEL

