THE IBM PC PROGRAMMER'S GUIDE TO C

3rd Edition

Matthew Probert

COPYRIGHT NOTICE

CONTENTS

BIOGRAPHICAL NOTES

Matthew Probert is a software consultant working for his own firm, Servile Software. He has been involved with micro-computer software design and programming since the age of eighteen and has been involved with the C programming language for the past ten years.

His educational background lies in the non-too distinguished honour of having a nervous break down during his last year of secondary school which resulted in a lack of formal qualifications. However, Matthew has made up for it by achieving a complete recovery and has been studying Psychology with particular attention to behaviourism and conversation ever since.

His chequered career spans back twelve years during which time he has trained people in the design and use of database management applications, been called upon to design and implement structured methodologies and has been a "good old fashioned" analyst programmer.

Matthew Probert is currently researching Artificial Intelligence with particular reference to the application of natural language processing, whereby a computer software package may decode written human language and respond to it in an intelligent manner. He is also monitoring the progress of facilitated communication amongst autistic and children with severe learning and challenging behaviour and hopes one day to be able to develope a computer based mechanism for true and reliable communication between autistic people and the rest of society.

Matthew Probert can be contacted via
Servile Software
5 Longcroft Close
Basingstoke
Hampshire
RG21 8XG
England

Telephone 01256 478576

PREFACE

In 1992, an English software house, Servile Software published a paper entitled "HOW TO C", which sought to introduce computer programmers to the C programming language. That paper was written by Matthew Probert. A follow up effort was "HOW TO CMORE", a document that was also published by Servile Software. Now those two documents have been amalgamated and thoroughly revamped to create this book. I have included loads of new source code that can be lifted directly out of the text.

All the program listings have been typed in to the Turbo C compiler, compiled and executed successfully before being imported into this document.

I hope you enjoy my work, and more I hope that you learn to program in C. It really is a great language, there can be no other language that gives the computer the opportunity to live up to the old saying;

"To err is human, to make a complete balls up requires a computer!"

Warning!

This document is the result of over ten years experience as a software engineer. This document contains professional source code that is not intended for beginers.

INTRODUCTION

The major distinguishing features of the C programming language are;

- block-structured flow-control constructs (typical of most high-level languages);
- freedom to manipulate basic machine objects (eg: bytes) and to refer to them using any particular object view desired (typical of assembly-languages);
- both high-level operations (eg: floating-point arithmetic) and low-level operations (which map closely onto machine-language instructions, thereby offering the means to code in an optimal, yet portable, manner).

This book sets out to describe the C programming language, as commonly found with compilers for the IBM PC, to enable a computer programmer with no previous knowledge of the C programming language to program in C using the IBM PC including the ROM facilities provided by the PC and facilities provided DOS.

It is assumed that the reader has access to a C compiler, and to the documentation that accompanies it regarding library functions.

The example programs were written with Borland's Turbo C, most of the non-standard facilities provided by Turbo C should be found in later releases of Microsoft C.

Differences Between the Various Versions of C

The original C (prior to the definitive book by K&R) defined the combination assignment operators (eg: +=, *=, etc.) backwards (ie: they were written =+, =*, etc.). This caused terrible confusion when a statement such as

```
x=-y;
```

was compiled - it could have meant

```
x = x - y;
```

or

```
x = (-y);
```

Ritchie soon spotted this ambiguity and changed the language to have these operators written in the now-familiar manner (+=, *=, etc.).

The major variations, however, are between K&R C and ANSI C. These can be summarized as follows:

- introduction of function prototypes in declarations; change of function definition preamble to match the style of prototypes;
- introduction of the ellipsis ("...") to show variable-length function argument lists;
- introduction of the keyword 'void' (for functions not returning a value) and the type 'void *' for generic pointer variables;
- addition of string-merging, token-pasting and stringizing functions in the preprocessor;
- addition of trigraph translation in the preprocessor;
- addition of the '#pragma' directive and formalization of the 'declared()' pseudofunction in the preprocessor;
- introduction of multi-byte strings and characters to support non-English languages;
- introduction of the 'signed' keyword (to complement the 'unsigned' keyword when used in integer declarations) and the unary plus ('+') operator.

C is a medium level language

The powerful facilities offered by C to allow manipulation of direct memory addresses and data, even down to the bit level, along with C's structured approach to programming cause C to be classified as a "medium level" programming language. It possesses fewer ready made facilities than a high level language, such as BASIC, but a higher level of structure than low level Assembler.

Key words

The original C language as described in; "The C programming language", by Kernighan and Ritchie, provided 27 key words. To those 27 the ANSI standards committee on C have added five more. This confusingly results in two standards for the C language. However, the ANSI standard is quickly taking over from the old K & R standard.

The 32 C key words are;

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

Some C compilers offer additional key words specific to the hardware environment that they operate on. You should be aware of your own C compilers additional key words. Most notably on the PC these are;

near far huge

Structure

C programs are written in a structured manner. A collection of code blocks are created that call each other to comprise the complete program. As a structured language C provides various looping and testing commands such as;

do-while, for, while, if

and the use of jumps, while provided for, are rarely used.

A C code block is contained within a pair of curly braces "{ }", and may be a complete procedure, in C terminology called a "function", or a subset of code within a function. For example the following is a code block. The statements within the curly braces are only executed upon satisfaction of the condition that "x < 10";

```
if (x < 10)
{
        a = 1;
        b = 0;
}
```

while this, is a complete function code block containing a sub code block as a do-while loop;

```
int GET_X()
{
        int x;
        do
        {
                printf("\nEnter a number between 0 and 10 ");
                scanf("%d",&x);
        }
        while(x < 0 || x > 10);
        return(x);
}
```

Notice how every statement line is terminated in a semicolon, unless that statement marks the start of a code block, in which case it is followed by a curly brace. C is a case sensitive but free flow language, spaces between commands are ignored, and therefore the semicolon delimiter is required to mark the end of the command line.

Having a freeflow structure the following commands are recognised as the same by the C compiler;

```
        x = 0;
        x          =0;
        x=0;
```

The general form of a C program is as follows;

```
        compiler preprocessor statements
        global data declarations




        return-type main(parameter list)
        {
                statements
        }

        return-type f1(parameter list)
        {
                statements
        }

        return-type f2(parameter list)
        {
                statements
        }
        .
        .
        .
        return-type fn(parameter list)
        {
                statements
        }
```

Comments

C allows comments to be included in the program. A comment line is defined by being enclosed within "/*" and "*/". Thus the following is a comment;

/* This is a legitimate C comment line */

Libraries

C programs are compiled and combined with library functions provided with the C compiler. These libraries are of generally standard functions, the functionality of which are defined in the ANSI standard of the C language, but are provided by the individual C compiler manufacturers to be machine dependant. Thus, the standard library function “printf()” provides the same facilities on a DEC VAX as on an IBM PC, although the actual machine language code in the library is quite different for each. The C programmer however, does not need to know about the internals of the libraries, only that each library function will behave in the same way on any computer.

DATA TYPES

There are four basic types of data in the C language; character, integer, floating point, and valueless that are referred to by the C key words;

“char”, “int”, “float” and “void” respectively.

To the basic data types may be added the type modifiers; signed, unsigned, long and short to produce further data types. By default data types are assumed signed, and the signed modifier is rarely used, unless to overide a compiler switch defaulting a data type to unsigned.

The size of each data type varies from one hardware platform to another, but the least range of values that can be held is described in the ANSI standard as follows;

Type	Size	Range
char	8	-127 to 127
unsigned char	8	0 to 255
int	16	-32767 to 32767
unsigned int	16	0 to 65535
long int	32	-2147483647 to 2147483647
unsigned long int	32	0 to 4294967295
float	32	Six digit precision
double	64	Ten digit precision
long double	80	Ten digit precision

In practice, this means that the data type ‘char’ is particularly suitable for storing flag type variables, such as status codes, which have a limited range of values. The ‘int’ data type can be used, but if the range of values does not exceed 127 (or 255 for an unsigned char), then each declared variable would be wasting storage space.

Which real number data type to use: ‘float’, ‘double’ or ‘long double’ is another tricky question. When numeric accuracy is required, for example in an accounting application, the instinct would be to use the ‘long double’, but this requires at least 10 bytes of storage space for each variable. And real numbers are not as precise as integers anyway, so perhaps one should use integer data types instead and work around the problem. The data type ‘float’ is worse than useless since its six digit precision is too inaccurate to be relied upon. Generally, then, you should use integer data types where ever possible, but if real numbers are required use at least a ‘double’.

Declaring a variable

All variables in a C program must be declared before they can be used. The general form of a variable definition is;

```
type name;
```

So, for example to declare a variable “x”, of data type “int” so that it may store a value in the range -32767 to 32767, you use the statement;

```
int x;
```

Character strings may be declared, which are in reality arrays of characters. They are declared as follows;

```
char name[number_of_elements];
```

So, to declare a string thirty characters long, and called ‘name’ you would use the declaration;

```
char name[30];
```

Arrays of other data types also may be declared in one, two or more dimensions in the same way. For example to declare a two dimensional array of integers;

```
int x[10][10];
```

The elements of this array are then accessed as;

```
x[0][0]
x[0][1]
x[n][n]
```

There are three levels of access to variable; local, module and global. A variable declared within a code block is only known to the statements within that code block. A variable declared outside any function code blocks but prefixed with the storage modifier “static” is known only to the statements within that source module. A variable declared outside any functions and not prefixed with the static storage type modifier may be accessed by any statement within any source module of the program.

For example;

```
int error;
static int a;

main()
{
        int x;
        int y;

}

funca()
{
        /* Test variable 'a' for equality with 0 */
        if (a == 0)
        {
                int b;
                for(b = 0; b < 20; b++)
                printf("\nHello World");
        }

    }
```

In this example the variable 'error' is accessible by all source code modules compiled together to form the finished program. The variable 'a' is accessible by statements in both functions 'main()' and 'funca()', but is invisible to any other source module. Variables 'x' and 'y' are only accessible by statements within function 'main()'. The variable 'b' is only accessible by statements within the code block following the 'if' statement.

If a second source module wished to access the variable 'error' it would need to declare 'error' as an 'extern' global variable thus;

```
    extern int error;

    funcb()
    {
    }
```

C will quite happily allow you, the programmer, to assign different data types to each other. For example, you may declare a variable to be of type 'char' in which case a single byte of data will be allocated to store the variable. To this variable you can attempt to allocate larger values, for example;

```
    main()
    {

        x = 5000;

    }
```

In this example the variable 'x' can only store a value between -127 and 128, so the figure 5000 will NOT be assigned to the variable 'x'. Rather the value 136 will be assigned!

Often you may wish to assign different data types to each other, and to prevent the compiler from warning you of a possible error you can use a cast to tell the compiler that you know what you're doing. A cast statement is a data type in parenthesis preceding a variable or expression;

```
    main()
    {
        float x;
        int y;

        x = 100 / 25;

        y = (int)x;
    }
```

In this example the (int) cast tells the compiler to convert the value of the floating point variable x to an integer before assigning it to the variable y.

Formal parameters

A C function may receive parameters from a calling function. These parameters are declared as variables within the parentheses of the function name, thus;

```
int MULT(int x, int y)
{
        /* Return parameter x multiplied by parameter y */
        return(x * y);
}

main()
{
        int a;
        int b;
        int c;

        a = 5;
        b = 7;
        c = MULT(a,b);

        printf("%d multiplied by %d equals %d\n",a,b,c);
}
```

Access modifiers

There are two access modifiers; 'const' and 'volatile'. A variable declared to be 'const' may not be changed by the program, whereas a variable declared as type as type 'volatile' may be changed by the program. In addition, declaring a variable to be volatile prevents the C compiler from allocating the variable to a register, and reduces the optimization carried out on the variable.

Storage class types

C provides four storage types; 'extern', 'static', 'auto' and 'register'.

The extern storage type is used to allow a source module within a C program to access a variable declared in another source module.

Static variables are only accessible within the code block that declared them, and additionally if the variable is local, rather than global, they retain their old value between subsequent calls to the code block.

Register variables are stored within CPU registers where ever possible, providing the fastest possible access to their values.

The auto type variable is only used with local variables, and declares the variable to retain its value locally only. Since this is the default for local variables the auto storage type is very rarely used.

OPERATORS

Operators are tokens that cause a computation to occur when applied to variables. C provides the following operators;

&	Address
*	Indirection
+	Unary plus
-	Unary minus
~	Bitwise compliment
!	Logical negation
++	As a prefix; preincrement
	As a suffix; postincrement
--	As a prefix; predecrement
	As a suffix; postdecrement
+	Addition
-	Subtraction
*	Multiply
/	Divide
%	Remainder
<<	Shift left
>>	Shift right
&	Bitwise AND
|	Bitwise OR
^	Bitwise XOR
&&	Logical AND
||	Logical OR
=	Assignment
*=	Assign product
/=	Assign quotient
%=	Assign remainder (modulus)
+=	Assign sum
-=	Assign difference
<<=	Assign left shift
>>=	Assign right shift
&=	Assign bitwise AND
|=	Assign bitwise OR
^=	Assign bitwise XOR
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
==	Equal to
!=	Not equal to
.	Direct component selector
->	Indirect component selector
a ? x:y	"If a is true then x else y"
[]	Define arrays
()	Parenthesis isolate conditions and expressions
...	Ellipsis are used in formal parameter lists of function prototypes to show a variable number of parameters or parameters of varying types.

To illustrate some more commonly used operators consider the following short program;

```
	main()
	{
		int a;
		int b;
		int c;
		a = 5;			/* Assign a value of 5 to variable 'a' */
		b = a / 2;		/* Assign the value of 'a' divided by two to
		variable 'b' */
		c = b * 2;		/* Assign the value of 'b' multiplied by two		to
	variable 'c' */

		if (a == c)		/* Test if 'a' holds the same value as 'c' */

			puts("Variable 'a' is an even number");
		else
			puts("Variable 'a' is an odd number");
	}
```

Normally when incrementing the value of a variable you would write something like;

```
		x = x + 1
```

C provides the incremental operator '++' as well so that you can write;

```
		x++
```

Similarly you can decrement the value of a variable using '--' as;

```
		x--
```

All the other mathematical operators may be used the same, so in a C program you can write in shorthand;

NORMAL	C
x = x + 1	x++
x = x - 1	x--
x = x * 2	x *= 2
x = x / y	x /= y
x = x % 5	x %= 5

and so on.

FUNCTIONS

Functions are the source code procedures that comprise a C program. They follow the general form;

```
return_type function_name(parameter_list)
{
        statements
}
```

The return_type specifies the data type that will be returned by the function; char, int, double, void &c.

The code within a C function is invisible to any other C function, and jumps may not be made from one function into the middle of another, although functions may call other functions. Also, functions cannot be defined within functions, only within source modules.

Parameters may be passed to a function either by value, or by reference. If a parameter is passed by value, then only a copy of the current value of the parameter is passed to the function. A parameter passed by reference however, is a pointer to the actual parameter that may then be changed by the function.

The following example passes two parameters by value to a function, funca(), which attempts to change the value of the variables passed to it. And then passes the same two parameters by reference to funcb() which also attempts to modify their values.

```
#include <stdio.h>

int funca(int x, int y)
{
        /* This function receives two parameters by value, x and y */

        x = x * 2;
        y = y * 2;

        printf("\nValue of x in funca() %d value of y in funca() %d",x,y);

        return(x);
}


int funcb(int *x, int *y)
{
        /* This function receives two parameters by reference, x and y */

        *x = *x * 2;
        *y = *y * 2;

        printf("\nValue of x in funcb() %d value of y in funcb() %d",*x,*y);

        return(*x);
}

main()
{
        int x;
        int y;
```

```
		int z;

		x = 5;
		y = 7;

		z = funca(x,y);
		z = funcb(&x,&y);

		printf("\nValue of x %d value of y %d value of z %d",x,y,z);
	}
```

Actually funcb() does not change the values of the parameters it receives. Rather it changes the contents of the memory addresses pointed to by the received parameters. While funca() receives the values of variables ‘x’ and ‘y’ from function main(), funcb() receives the memory addresses of the variables ‘x’ and ‘y’ from function main().

Passing an array to a function

The following program passes an array to a function, funca(), which initialises the array elements;

```
	#include <stdio.h>

	void funca(int x[])
	{
		int n;

		for(n = 0; n < 100; n++)
		x[n] = n;
	}

	main()
	{
		int array[100];
		int counter;

		funca(array);

		for(counter = 0; counter < 100; counter++)
			printf("\nValue of element %d is %d",counter,array[counter]);
	}
```

The parameter of funca() ‘int x[]’ is declared to be an array of any length. This works because the compiler passes the address of the start of the array parameter to the function, rather than the value of the individual elements. This does of course mean that the function can change the value of the array elements. To prevent a function from changing the values you can specify the parameter as type ‘const’;

```
	funca(const int x[])
	{
	}
```

This will then generate a compiler error at the line that attempts to write a value to the array. However, specifying a parameter to be const does not protect the parameter from indirect assignment as the following program illustrates;

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

```
int funca(const int x[])
{
        int *ptr;
        int n;

        /* This line gives a 'suspicious pointer conversion warning' */
        /* because x is a const pointer, and ptr is not */
        ptr = x;

        for(n = 0; n < 100; n++)
        {
                *ptr = n;
                ptr++;
        }
}

main()
{
        int array[100];
        int counter;

        funca(array);

        for(counter = 0; counter < 100; counter++)
                printf("\nValue of element %d is %d",counter,array[counter]);
}
```

Passing parameters to main()

C allows parameters to be passed from the operating system to the program when it starts executing through two parameters; argc and argv[], as follows;

```
#include <stdio.h>

main(int argc, char *argv[])
{
        int n;

        for(n = 0; n < argc; n++)
        printf("\nParameter %d equals %s",n,argv[n]);
}
```

Parameter argc holds the number of parameters passed to the program, and the array argv[] holds the addresses of each parameter passed. argv[0] is always the program name. This feature may be put to good use in applications that need to access system files. Consider the following scenario:

A simple database application stores its data in a single file called “data.dat”. The application needs to be created so that it may be stored in any directory on either a floppy diskette or a hard disk, and executed both from within the host directory and through a DOS search path. To work correctly the application must always know where to find the data file; “data.dat”. This is solved by assuming that the data file will be in the same directory as the executable module, a not unreasonable restriction to place upon the operator. The following code fragment then illustrates how an application may apply this algorithm into practice to be always able to locate a desired system file:

```
#include <string.h>

char system_file_name[160];

void main(int argc,char *argv[])
{
        char *data_file = "DATA.DAT";
        char *p;

        strcpy(system_file_name,argv[0]);
        p = strstr(system_file_name,".EXE");
        if (p == NULL)
        {
                /* The executable is a .COM file */
          p = strstr(system_file_name,".COM");
        }

        /* Now back track to the last '\' character in the file name */
        while(*(p - 1) != '\\')
                p--;

        strcpy(p,data_file);
}
```

In practice this code creates a string in system_file_name that is comprised of path\data.dat, so if for example the executable file is called "test.exe" and resides in the directory \borlandc, then system_file_name will be assigned with: \borlandc\data.dat

Returning from a function

The command 'return' is used to return immediately from a function. If the function was declared with a return data type, then return should be used with a parameter of the same data type.

Function prototypes

Prototypes for functions allow the C compiler to check that the type of data being passed to and from functions is correct. This is very important to prevent data overflowing its allocated storage space into other variables areas.

A function prototype is placed at the beginning of the program, after any preprocessor commands, such as #include <stdio.h>, and before the declaration of any functions.

THE C PREPROCESSOR

C allows for commands to the compiler to be included in the source code. These commands are then called preprocessor commands and are defined by the ANSI standard to be;

```
#if
#ifdef
#ifndef
#else
#elif
#include
#define
#undef
#line
#error
#pragma
```

All preprocessor commands start with a hash symbol, "#", and must be on a line on their own (although comments may follow).

#define

The #define command specifies an identifier and a string that the compiler will substitute every time it comes accross the identifier within that source code module. For example;

```
#define FALSE 0
#define TRUE !FALSE
```

The compiler will replace any subsequent occurence of ‘FALSE’ with ‘0’ and any subsequent occurence of ‘TRUE’ with ‘!0’. The substitution does NOT take place if the compiler finds that the identifier is enclosed by quotation marks, so

```
printf("TRUE");
```

would NOT be replaced, but

```
printf("%d",FALSE);
```

would be.

The #define command also can be used to define macros that may include parameters. The parameters are best enclosed in parenthesis to ensure that correct substitution occurs.

This example declares a macro ‘larger()’ that accepts two parameters and returns the larger of the two;

```
#include <stdio.h>

#define larger(a,b)       (a > b) ? (a) : (b)
```

```
int main()
{
        printf("\n%d is largest",larger(5,7));

}
```

#error

The #error command causes the compiler to stop compilation and to display the text following the #error command. For example;

```
#error REACHED MODULE B
```

will cause the compiler to stop compilation and display;

```
        REACHED MODULE B
```

#include

The #include command tells the compiler to read the contents of another source file. The name of the source file must be enclosed either by quotes or by angular brackets thus;

```
#include "module2.c"
#include <stdio.h>
```

Generally, if the file name is enclosed in angular brackets, then the compiler will search for the file in a directory defined in the compiler’s setup. Whereas if the file name is enclosed in quotes then the compiler will look for the file in the current directory.

#if, #else, #elif, #endif

The #if set of commands provide conditional compilation around the general form;

```
#if constant_expression
        statements
#else
        statements
#endif
```

#elif stands for '#else if' and follows the form;

```
#if expression
        statements
#elif expression
        statements
#endif
```

#ifdef, #ifndef

These two commands stand for '#if defined' and '#if not defined' respectively and follow the general form;

```
        #ifdef macro_name
                statements
        #else
                statements
        #endif

        #ifndef macro_name
                statements
        #else
                statements
        #endif
```

where 'macro_name' is an identifier declared by a #define statement.

#undef

Undefines a macro previously defined by #define.

#line

Changes the compiler declared global variables __LINE__ and __FILE__. The general form of #line is;

```
        #line number "filename"
```

where number is inserted into the variable '__LINE__' and 'filename' is assigned to '__FILE__'.

#pragma

This command is used to give compiler specific commands to the compiler. The compiler’s manual should give you full details of any valid options to go with the particular implementation of #pragma that it supports.

PROGRAM CONTROL STATEMENTS

As with any computer language, C includes statements that test the outcome of an expression. The outcome of the test is either TRUE or FALSE. The C language defines a value of TRUE as non-zero, and FALSE as zero.

Selection statements

The general purpose selection statement is "if" that follows the general form;

```
if (expression)
        statement
else
        statement
```

Where “statement” may be a single statement, or a code block enclosed in curly braces. The “else” is optional. If the result of the expression equates to TRUE, then the statement(s) following the if() will be evaluated. Otherwise the statement(s) following the else, if there is one, will be evaluated.

An alternative to the if....else combination is the ?: command that takes the form;

```
expression ? true_expression : false_expression
```

Where if the expression evaluates to TRUE, then the true_expression will be evaluated, otherwise the false_expression will be evaluated. Thus we get;

```
#include <stdio.h>

main()
{
        int x;

        x = 6;

        printf("\nx is an %s number", x % 2 == 0 ? "even" : "odd");
}
```

C also provides a multiple branch selection statement, switch, which successively tests a value of an expression against a list of values and branches program execution to the first match found. The general form of switch is;

```
switch (expression)
{
        case value1 :   statements
                        break;
        case value2 :   statements
                        break;
        .
        .
        .
        .
        case valuen :   statements
                        break;
        default :       statements
    }
```

The break statement is optional, but if omitted, program execution will continue down the list.

```
    #include <stdio.h>

    main()
    {
        int x;

        x = 6;

        switch(x)
        {
            case 0 : printf("\nx equals zero");
                     break;
            case 1 : printf("\nx equals one");
                     break;
            case 2 : printf("\nx equals two");
                     break;
            case 3 : printf("\nx equals three");
                     break;
            default : printf("\nx is larger than three");
        }
    }
```

Switch statements may be nested within one another. This is a particularly useful feature for confusing people who read your source code!

Iteration statements

C provides three looping or iteration statements; for, while, and do-while. The for loop has the general form;

```
    for(initialization;condition;increment)
```

and is useful for counters such as in this example that displays the entire ascii character set;

```
    #include <stdio.h>

    main()
    {
        int x;

        for(x = 32; x < 128; x++)
            printf("%d\t%c\t",x,x);
    }
```

An infinite for loop is also quite valid;

```
    for(;;)
    {
        statements
    }
```

Also, C allows empty statements. The following for loop removes leading spaces from a string;

```
for(; *str == ' '; str++)
        ;
```

Notice the lack of an initializer, and the empty statement following the loop.

The while loop is somewhat simpler than the for loop and follows the general form;

```
while (condition)
        statements
```

The statement following the condition, or statements enclosed in curly braces will be executed until the condition is FALSE. If the condition is false before the loop commences, the loop statements will not be executed. The do-while loop on the other hand is always executed at least once. It takes the general form;

```
do
{
        statements
}
while(condition);
```

Jump statements

The "return" statement is used to return from a function to the calling function. Depending upon the declared return data type of the function it may or may not return a value;

```
int MULT(int x, int y)
{
        return(x * y);
}
```

or;

```
void FUNCA()
{
        printf("\nHello World");
        return;
}
```

The "break" statement is used to break out of a loop or from a switch statement. In a loop it may be used to terminate the loop prematurely, as shown here;

```
#include <stdio.h>

main()
{
        int x;

        for(x = 0; x < 256; x++)
        {
                if (x == 100)
                        break;

                printf("%d\t",x);
                }
        }
```

In contrast to “break” is “continue”, which forces the next iteration of the loop to occur, effectively forcing program control back to the loop statement.

C provides a function for terminating the program prematurely, “exit()”. Exit() may be used with a return value to pass back to the calling program;

```
        exit(return_value);
```

More About ?:

A powerful, but often misunderstood feature of the C programming language is ?:. This is an operator that acts upon a boolean expression, and returns one of two values dependant upon the result of the expression;

```
        <boolean expression> ? <value for true> : <value for false>
```

It can be used almost anywhere, for example it was used in the binary search demonstration program;

```
        printf("\n%s\n",(result == 0) ? "Not found" : "Located okay");
```

Here it passes either “Not found” or “Located okay” to the printf() function dependant upon the outcome of the boolean expression ‘result == 0’. Alternatively it can be used for assigning values to a variable;

```
        x = (a  == 0) ? (b) : (c);
```

Which will assign the value of b to variable x if a is equal to zero, otherwise it will assign the value of c to variable x.

This example returns the name of the executing program, without any path description;

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

        char *progname(char *pathname)
        {
                /* Return name of running program */
                unsigned l;
                char *p;
                char *q;
                static char bnbuf[256];

                return pathname? p = strrchr (pathname, '\\'),
                                q = strrchr (pathname, '.'),
                                l = (q == NULL? strchr (pathname, '\0'): q)
                                  - (p == NULL? p = pathname: ++p),
                                strncpy (bnbuf, p, l),
                                bnbuf[l] = '\0',
                                strlwr (bnbuf)
                        : NULL;
        }
        void main(int argc, char *argv[])
        {
                printf("\n%s",progname(argv[0]));
        }
```

Continue

The continue keyword forces control to jump to the test statement of the innermost loop (while, do...while()). This can be useful for terminating a loop gracefuly as this program that reads strings from a file until there are no more illustrates;

```
        #include <stdio.h>

        void main()
        {
                FILE *fp;
                char *p;
                char buff[100];

                fp = fopen("data.txt","r");
                if (fp == NULL)
                {
                        fprintf(stderr,"Unable to open file data.txt");
                        exit(0);
                }

                do
                {
                        p = fgets(buff,100,fp);
                        if (p == NULL)
                                /* Force exit from loop */
                                continue;
                        puts(p);
                }
                while(p);
        }
```

With a for() loop however, continue passes control back to the third parameter!

INPUT AND OUTPUT

Input

Input to a C program may occur from the console, the standard input device (unless otherwise redirected this is the console), from a file or from a data port.

The general input command for reading data from the standard input stream 'stdin' is scanf(). Scanf() scans a series of input fields, one character at a time. Each field is then formatted according to the appropriate format specifier passed to the scanf() function as a parameter. This field is then stored at the ADDRESS passed to scanf() following the format specifiers list.

For example, the following program will read a single integer from the stream stdin;

```
main()
{
        int x;

        scanf("%d",&x);
}
```

Notice the address operator & prefixing the variable name 'x' in the scanf() parameter list. This is because scanf() stores values at ADDRESSES rather than assigning values to variables directly.

The format string is a character string that may contain three types of data:

whitespace characters (space, tab and newline), non-whitespace characters (all ascii characters EXCEPT %) and format specifiers.

Format specifiers have the general form;

```
%[*][width][h|l|L]type_character
```

After the % sign the format specifier is comprised of:

an optional assignment suppression character, *, which suppresses assignment of the next input field.

an optional width specifier, width, which declares the maximum number of characters to be read.

an optional argument type modifier, h or l or L, where:

h is a short integer

l is a long

L is a long double

the data type character that is one of;

d	Decimal integer
D	Decimal long integer
o	Octal integer
O	Octal long integer
i	Decimal, octal or hexadecimal integer
I	Decimal, octal or hexadecimal long integer
u	Decimal unsigned integer
U	Decimal unsigned long integer
x	Hexadecimal integer
X	Hexadecimal long integer
e	Floating point
f	Floating point
g	Floating point
s	Character string
c	Character
%	% is stored

An example using scanf();

```
#include <stdio.h>

main()
{
        char name[30];
        int age;

        printf("\nEnter your name and age ");
        scanf("%30s%d",name,&age);
        printf("\n%s %d",name,age);
}
```

Notice the include line, “#include <stdio.h>”, this is to tell the compiler to also read the file stdio.h that contains the function prototypes for scanf() and printf().

If you type in and run this sample program you will see that only one name can be entered, that is you can’t enter;

```
JOHN SMITH
```

because scanf() detects the whitespace between “JOHN” and “SMITH” and moves on to the next input field, which is age, and attempts to assign the value “SMITH” to the age field! The limitations of scanf() as an input function are obvious.

An alternative input function is gets() that reads a string of characters from the stream stdin until a newline character is detected. The newline character is replaced by a null (0 byte) in the target string. This function has the advantage of allowing whitespace to be read in. The following program is a modification to the earlier one, using gets() instead of scanf().

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

main()
{
        char data[80];
        char *p;
```

```
            char name[30];
            int age;

            printf("\nEnter your name and age ");
            /* Read in a string of data */
            gets(data);


            /* P is a pointer to the last character in the input string */
            p = &data[strlen(data) - 1];

            /* Remove any trailing spaces by replacing them with null bytes */
            while(*p == ' '){
                        *p = 0;
                        p--;
            }

            /* Locate last space in the string */
            p = strrchr(data,' ');

            /* Read age from string and convert to an integer */
            age = atoi(p);

            /* Terminate data string at start of age field */
            *p = 0;

            /* Copy data string to name variable */
            strcpy(name,data);

            /* Display results */
            printf("\nName is %s age is %d",name,age);
}
```

Output

The most common output function is printf() that is very similar to scanf() except that it writes formatted data out to the standard output stream stdout. Printf() takes a list of output data fields and applies format specifiers to each and outputs the result. The format specifiers are the same as for scanf() except that flags may be added to the format specifiers. These flags include;

- - Which left justifies the output padding to the right with spaces.
- + Which causes numbers to be prefixed by their sign

The width specifier is also slightly different for printf(). In its most useful form is the precision specifier;

```
width.precision
```

So, to print a floating point number to three decimal places you would use;

```
printf("%.3f",x);
```

And to pad a string with spaces to the right or truncate the string to twenty characters if it is longer, to form a fixed twenty character width;

```
printf("%-20.20s",data);
```

Special character constants may appear in the printf() parameter list. These are;

\n	Newline
\r	Carriage return
\t	Tab
\b	Sound the computer's bell
\f	Formfeed
\v	Vertical tab
\\	Backslash character
\'	Single quote
\"	Double quote
\?	Question mark
\O	Octal string
\x	Hexadecimal string

Thus, to display "Hello World", surrounded in quotation marks and followed by a newline you would use;

```
printf("\"Hello World\"\n");
```

The following program shows how a decimal integer may be displayed as a decimal, hexadecimal or octal integer. The 04 following the % in the printf() format tells the compiler to pad the displayed figure to a width of at least four digits padded with leading zeroes if required.

```
/* A simple decimal to hexadecimal and octal conversion program */

#include <stdio.h>

main()
{
        int x;

        do
        {
                printf("\nEnter a number, or 0 to end ");
                scanf("%d",&x);
                printf("%04d  %04X  %04o",x,x,x);
        }
        while(x != 0);

}
```

There are associated functions to printf() that you should be aware of. fprintf() has the prototype;

```
fprintf(FILE *fp,char *format[,argument,...]);
```

This variation on printf() simply sends the formatted output to the specified file stream.

sprintf() has the function prototype;

```
sprintf(char *s,char *format[,argument,...]);
```

and writes the formatted output to a string. You should take care using sprintf() that the target string has been declared long enough to hold the result of the sprintf() output. Otherwise other data will be overwritten in memory.

An example of using sprintf() to copy multiple data to a string;

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

```
int main()
{
        char buffer[50];

        sprintf(buffer,"7 * 5 == %d\n",7 * 5);

        puts(buffer);
}
```

An alternative to printf() for outputting a simple string to the stream stdout is puts(). This function sends a string to the stream stdout followed by a newline character. It is faster than printf(), but far less flexible. Instead of;

```
printf("Hello World\n");
```

You can use;

```
puts("Hello World");
```

Direct Console I/O

Data may be sent to, and read directly from the console (keyboard and screen) using the direct console I/O functions. These functions are prefixed 'c'. The direct console I/O equivalent of printf() is then cprintf(), and the equivalent of puts() is cputs(). Direct console I/O functions differ from standard I?o functions:

1. They do not make use of the predefined streams, and hence may not be redirected
2. They are not portable across operating systems (for example you cant use direct console I/O functions in a Windows programme).
3. They are faster than their standard I/O equivalents
4. They may not work with all video modes (especially VESA display modes).

POINTERS

A pointer is a variable that holds the memory address of an item of data. Therefore it points to another item. A pointer is declared like an ordinary variable, but its name is prefixed by ‘*’, thus;

```
char *p;
```

This declares the variable ‘p’ to be a pointer to a character variable. Pointers are very powerful, and similarly dangerous! If only because a pointer can be inadvertently set to point to the code segment of a program and then some value assigned to the address of the pointer!

The following program illustrates a simple, though fairly useless application of a pointer;

```
#include <stdio.h>

main()
{
        int a;
        int *x;

        /* x is a pointer to an integer data type */

        a = 100;
        x = &a;

        printf("\nVariable 'a' holds the value %d at memory address %p",a,x);
}
```

Here is a more useful example of a pointer illustrating how because the compiler knows the type of data pointed to by the pointer, when the pointer is incremented it is incremented the correct number of bytes for the data type. In this case two bytes;

```
#include <stdio.h>

main()
{
        int n;
        int a[25];
        int *x;

        /* x is a pointer to an integer data type */

        /* Assign x to point to array element zero */
        x = a;

        /* Assign values to the array */
        for(n = 0; n < 25; n++)
                a[n] = n;


        /* Now print out all array element values */
        for(n = 0; n < 25; n++ , x++)
                printf("\nElement %d holds %d",n,*x);
}
```

To read or assign a value to the address held by a pointer you use the indirection operator ‘*’. Thus in the above example, to print the value at the memory address pointed to by variable x I have used ‘*x’.

Pointers may be incremented and decremented and have other mathematics applied to them also. For example in the above program to move variable x along the array one element at a time we put the statement ‘x++’ in the for loop. We could move x along two elements by saying ‘x += 2’. Notice that this doesn’t mean “add 2 bytes to the value of x”, but rather it means “add 2 of the pointer’s data type size units to the value of x”.

Pointers are used extensively in dynamic memory allocation. When a program is running it is often necessary to temporarily allocate a block of data, say a table, in memory. C provides the function malloc() for this purpose that follows the general form;

```
any pointer type = malloc(number_of_bytes);
```

malloc() actually returns a void pointer type, which means it can be any type; integer, character, floating point or whatever. This example allocates a table in memory for 1000 integers;

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

main()
{
        int *x;
        int n;

        /* x is a pointer to an integer data type */

        /* Create a 1000 element table, sizeof() returns the compiler */
        /* specific number of bytes used to store an integer */

        x = malloc(1000 * sizeof(int));


        /* Check to see if the memory allocation succeeded */
        if (x == NULL)
        {
                printf("\nUnable to allocate a 1000 element integer table");
                exit(0);
        }

        /* Assign values to each table element */
        for(n = 0; n < 1000; n++)
        {
                *x = n;
                x++;
        }

        /* Return x to the start of the table */
        x -= 1000;

        /* Display the values in the table */
        for(n = 0; n < 1000; n++){
                printf("\nElement %d holds a value of %d",n,*x);
                x++;
        }
        /* Deallocate the block of memory now it's no longer required */
        free(x);
}
```

Pointers are also extensively used with character arrays, called strings. Since all C program strings are terminated by a zero byte we can count the letters in a string using a pointer;

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

main()
{
        char *p;
        char text[100];
        int len;

        /* Initialise variable 'text' with some writing */
        strcpy(text,"This is a string of data");

        /* Set variable p to the start of variable text */
        p = text;

        /* Initialise variable len to zero */
        len = 0;

        /* Count the characters in variable text */
        while(*p)
        {
                len++;
                p++;
        }

        /* Display the result */
        printf("\nThe string of data has %d characters in it",len);
}
```

The 8088/8086 group of CPUs, as used in the IBM PC, divide memory into 64K segments. To address all 1Mb of memory a 20 bit number is used comprised of an ‘offset’ to and a 64K ‘segment’. The IBM PC uses special registers called “segment registers” to record the segments of addresses.

This leads the C language on the IBM PC to have three new keywords; near, far and huge.

near pointers are 16 bits wide and access only data within the current segment.

far pointers are comprised of an offset and a segment address allowing them to access data any where in memory, but arithmetic with far pointers is fraught with danger! When a value is added or subtracted from a far pointer it is only actualy the segment part of the pointer that is affected, thus leading to the situation where a far pointer being incremented wraps around on its own offset 64K segment.

huge pointers are a variation on the far pointer that can be successfuly incremented and decremented through the entire 1Mb range since the compiler generates code to amend the offset as applicable.

It will come as no surprise that code using near pointers is executed faster than code using far pointers, which in turn is faster than code using huge pointers.

to give a literal address to a far pointer IBM PC C compilers provide a macro MK-FP() that has the prototype;

```
void far *MK_FP(unsigned segment, unsigned offset);
```

For example, to create a far pointer to the start of video memory on a colour IBM PC system you could use;

```
                    screen = MK_FP(0xB800,0);
```

Two coressponding macros provided are;

FP_SEG() and FP_OFF()

Which return the segment and offset respectively of a far pointer. The following example uses FP_SEG() and FP_OFF() to send segment and offset addresses to a DOS call to create a new directory path;

```
          #include <dos.h>

          int makedir(char *path)
          {
                    /* Make sub directory, returning non zero on success */

                    union REGS inreg,outreg;
                    struct SREGS segreg;

                    inreg.h.ah = 0x39;
                    segreg.ds = FP_SEG(path);
                    inreg.x.dx = FP_OFF(path);
                    intdosx(&inreg,&outreg,&segreg);

                    return(1 - outreg.x.cflag);
          }
```

Finally, the CPU's segment registers can be read with the function 'segread()'. This is a function unique to C compilers for the the 8086 family of processors. segread() has the function prototype:

```
          void segread(struct SREGS *segp);
```

It returns the current values of the segment registers in the SREGS type structure pointed to by the pointer 'segp'. For example:

```
          #include <dos.h>

          main()
          {
                    struct SREGS sregs;

                    segread(&sregs);
                    printf("\nCode segment is currently at %04X, Data segment is at %04X",sregs.cs,sregs.ds);
                    printf("\nExtra segment is at %04X, Stack segment is at %04X",sregs.es,sregs.ss);
          }
```

STRUCTURES

C provides the means to group together variables under one name providing a convenient means of keeping related information together and providing a structured approach to data.

The general form for a structure definition is;

```
        typedef struct
        {
                variable_type variable_name;
                variable_type variable_name;
        }
        structure_name;
```

When accessing data files, with a fixed record structure, the use of a structure variable becomes essential. The following example shows a record structure for a very simple name and address file. It declares a data structure called ‘data’, and comprised of six fields; ‘name’, ‘address’, ‘town’, ‘county’ , ‘post’ and ‘telephone’.

```
        typedef struct
        {
                char name[30];
                char address[30];
                char town[30];
                char county[30];
                char post[12];
                char telephone[15];
        }
        data;
```

The structure 'data' may then be used in the program as a variable data type for declaring variables;

```
        data record;
```

Thus declares a variable called 'record' to be of type 'data'.

The individual fields of the structure variable are accessed by the general form;

```
        structure_variable.field_name;
```

Thus, the name field of the structure variable record is accessed with;

```
        record.name
```

There is no limit to the number of fields that may comprise a structure, nor do the fields have to be of the same types, for example;

```
        typedef struct
        {
                char name[30];
                int age;
                char *notes;
        }
        dp;
```

Declares a structure 'dp' that is comprised of a character array field, an integer field and a character pointer field.

A structure variable may be passed as a parameter by passing the address of the variable as the parameter with the & operator thus;

```
        func(&my_struct)
```

The following is an example program that makes use of a structure to provide the basic access to the data in a simple name and address file;

```
/* A VERY simple address book written in ANSI C */

#include <stdio.h>
#include <stdlib.h>
#include <io.h>
#include <string.h>
#include <fcntl.h>
#include <sys\stat.h>

/* num_lines is the number of screen display lines */
#define num_lines       25


typedef struct
{
        char name[30];
        char address[30];
        char town[30];
        char county[30];
        char post[12];
        char telephone[15];
}
data;

data record;
int handle;


/* Function prototypes */

void ADD_REC(void);
void CLS(void);
void DISPDATA(void);
void FATAL(char *);
void GETDATA(void);
void MENU(void);
void OPENDATA(void);
int SEARCH(void);

void CLS()
{
        int n;

        for(n = 0; n < num_lines; n++)
```

```
                puts("");
}

void FATAL(char *error)
{
        printf("\nFATAL ERROR: %s",error);
        exit(0);
}

void OPENDATA()
{
        /* Check for existence of data file and if not create it */
        /* otherwise open it for reading/writing at end of file */

        handle = open("address.dat",O_RDWR|O_APPEND,S_IWRITE);

        if (handle == -1)
        {
                handle = open("address.dat",O_RDWR|O_CREAT,S_IWRITE);
                if (handle == -1)
                        FATAL("Unable to create data file");
        }
}

void GETDATA()
{
        /* Get address data from operator */

        CLS();

        printf("Name ");
        gets(record.name);
        printf("\nAddress ");
        gets(record.address);
        printf("\nTown ");
        gets(record.town);
        printf("\nCounty ");
        gets(record.county);
        printf("\nPost Code ");
        gets(record.post);
        printf("\nTelephone ");
        gets(record.telephone);
}

void DISPDATA()
{
        /* Display address data */
        char text[5];

        CLS();

        printf("Name %s",record.name);
        printf("\nAddress %s",record.address);
        printf("\nTown %s",record.town);
        printf("\nCounty %s",record.county);
        printf("\nPost Code %s",record.post);
        printf("\nTelephone %s\n\n",record.telephone);
        puts("Press RETURN to continue");
        gets(text);
}

void ADD_REC()
{
        /* Insert or append a new record to the data file */
        int result;

        result = write(handle,&record,sizeof(data));

        if (result == -1)
                FATAL("Unable to write to data file");
}

int SEARCH()
{
        char text[100];
        int result;

        printf("Enter data to search for ");
        gets(text);
        if (*text == 0)
                return(-1);

        /* Locate start of file */
        lseek(handle,0,SEEK_SET);

        do
        {
                /* Read record into memory */
                result = read(handle,&record,sizeof(data));
                if (result > 0)
                {
                        /* Scan record for matching data */
                        if (strstr(record.name,text) != NULL)
                                return(1);
                        if (strstr(record.address,text) != NULL)
                                return(1);
                        if (strstr(record.town,text) != NULL)
                                return(1);
                        if (strstr(record.county,text) != NULL)
                                return(1);
                        if (strstr(record.post,text) != NULL)
                                return(1);
                        if (strstr(record.telephone,text) != NULL)
                                return(1);
                }
        }
        while(result > 0);
        return(0);
}

void MENU()
{
        int option;
        char text[10];
```

```
	do
	{
		CLS();
		puts("\n\t\t\tSelect Option");
		puts("\n\n\t\t\t1 Add new record");
		puts("\n\n\t\t\t2 Search for data");
		puts("\n\n\t\t\t3 Exit");
		puts("\n\n\n\n\n");
		gets(text);
		option = atoi(text);

		switch(option)
		{
			case 1 : GETDATA();
				/* Go to end of file to append new record */
				lseek(handle,0,SEEK_END);
				ADD_REC();
				break;

			case 2 : if (SEARCH())
					DISPDATA();
				else
				{
					puts("NOT FOUND!");
					puts("Press RETURN to continue");
					gets(text);
				}
				break;

			case 3 : break;
		}
	}
	while(option != 3);
}

void main()
{
	CLS();
	OPENDATA();
	MENU();
}
```

Bit Fields

C allows the inclusion of variables with a size of less than eight bits to be included in structures. These variables are known as bit fields, and may be any declared size from one bit upwards.

The general form for declaring a bit field is;

```
        type name : number_of_bits;
```

For example, to declare a set of status flags, each occupying one bit;

```
typedef struct
{
        unsigned carry  : 1;
        unsigned zero   : 1;
        unsigned over   : 1;
        unsigned parity : 1;
}
df;

df flags;
```

The variable ‘flags’ then occupies only four bits in memory, and yet is comprised of four variables that may be accessed like any other structure field.

Uunions

Another facility provided by C for efficient use of available memory is the union structure. A union structure is a collection of variables that all share the same memory storage address. As such only one of the variables is ever accessible at a time.

The general form of a union definition is;

```
        union name
        {
                type variable_name;
                type variable_name;
                .
                .
                .
                type variable_name;
        };
```

Thus, to declare a union structure for two integer variables;

```
        union data
        {
                int vara;
                int varb;
        };
```

and to declare a variable of type 'data';

```
        data my_var;
```

The variable 'my_var' is then comprised of the two variables 'vara' and 'varb' that are accessed like with any form of structure, eg;

```
        my_var.vara = 5;
```

Assigns a value of 5 to the variable 'vara' of union 'my_var'.

Enumerations

An enumeration assigns ascending integer values to a list of symbols. An enumeration declaration takes the general form;

```
        enum name { enumeration list } variable_list;
```

Thus to define a symbol list of colours, you can say;

```
        enum COLOURS
        {
                BLACK,
                BLUE,
                GREEN,
                CYAN,
                RED,
                MAGENTA,
                BROWN,
                LIGHTGRAY,
                DARKGRAY,
                LIGHTBLUE,
                LIGHTGREEN,
                LIGHTCYAN,
                LIGHTRED,
                LIGHTMAGENTA,
                YELLOW,
                WHITE
        };
```

Then, the number zero may be referred to by the symbol BLACK, the number one by the symbol BLUE, the number two by the symbol GREEN and so on.

The following program illustrates the use of an enumeration list to symbolise integers;

```
        #include <stdio.h>

        enum COLOURS
        {
                BLACK,
                BLUE,
                GREEN,
                CYAN,
                RED,
                MAGENTA,
                BROWN,
		LIGHTGRAY,
		DARKGRAY,
		LIGHTBLUE,
		LIGHTGREEN,
		LIGHTCYAN,
		LIGHTRED,
		LIGHTMAGENTA,
		YELLOW,
		WHITE
	};


	void main()
	{
		int x;

		x = RED;

		printf("\nVariable 'x' holds %d",x);

	}
```

FILE I/O

C provides buffered file streams for file access. Some C platforms, such as Unix and DOS provide unbuffered file handles as well.

Buffered streams

Buffered streams are accessed through a variable of type 'file pointer'. The data type FILE is defined in the header file stdio.h. Thus to declare a file pointer;

```
#include <stdio.h>

FILE *ptr;
```

To open a stream C provides the function fopen(), which accepts two parameters, the name of the file to be opened, and the access mode for the file to be opened with. The access mode may be any one of;

Mode	Description
r	Open for reading
w	Create for writing, destroying any existing file
a	Open for append, creating a new file if it doesn't exist
r+	Open an existing file for reading and writing
w+	Create for reading and writing, destroying any existing file
a+	Open for append, creating a new file if it doesn't exist.

Optionaly either 'b' or 't' may be appended for binary or text mode. If neither is appended, then the file stream will be opened in the mode described by the global variable _fmode. Data read or written from file streams opened in text mode undergoes conversion, that is the characters CR and LF are converted to CR LF pairs on writing, and the CR LF pair is converted to a single LF on reading. File streams opened in binary mode do not undergo conversion.

If fopen() fails to open the file, it returns a value of NULL (defined in stdio.h) to the file pointer.

Thus, the following program will create a new file called "data.txt" and open it for reading and writing;

```
#include <stdio.h>

void main()
{
        FILE *fp;

        fp = fopen("data.txt","w+");

}
```

To close a stream C provides the function fclose(), which accepts the stream's file pointer as a parameter;

```
fclose(fp);
```

If an error occurs in closing the file stream, fclose() returns non zero.

There are four basic functions for receiving and sending data to and from streams; fgetc(), fputc(), fgets() and fputs().

fgetc() simply reads a single character from the specified input stream;

```
char fgetc(FILE *fp);
```

Its opposite number is fputc(), which simply writes a single character to the specified input stream;

```
char fputc(char c, FILE *fp);
```

fgets() reads a string from the input stream;

```
char *fgets(char s, int numbytes, FILE *fp);
```

It stops reading when either numbytes - 1 bytes have been read, or a newline character is read in. A null terminating byte is appended to the read string, s. If an error occurs, fgets() returns NULL.

fputs() writes a null terminated string to a stream;

```
int fputs(char *s, FILE *fp);
```

Excepting fgets(), which returns a NULL pointer if an error occurs, all the other functions described above return EOF (defined in stdio.h) if an error occurs during the operation.

The following program creates a copy of the file “data.dat” as “data.old” and illustrates the use of fopen(), fgetc(), fputc() and fclose();

```
#include <stdio.h>

int main()
{
        FILE *in;
        FILE *out;

        in = fopen("data.dat","r");

        if (in == NULL)
        {
                puts("\nUnable to open file data.dat for reading");
                return(0);
        }

        out = fopen("data.old","w+");

        if (out == NULL)
        {
                puts("\nUnable to create file data.old");
                return(0);
        }

        /* Loop reading and writing one byte at a time until end-of-file */
        while(!feof(in))
                fputc(fgetc(in),out);
        /* Close the file streams */
        fclose(in);
        fclose(out);

        return(0);
}
```

Example program using fputs() to copy text from stream stdin (usually typed in at the keyboard) to a new file called "data.txt".

```
#include <stdio.h>

int main()
{
        FILE *fp;
        char text[100];

        fp = fopen("data.txt","w+");

        do
        {
                gets(text);
                fputs(text,fp);
        }
        while(*text);

        fclose(fp);
}
```

Random access using streams

Random file access for streams is provided for by the fseek() function that has the following prototype;

```
int fseek(FILE *fp, long numbytes, int fromwhere);
```

fseek() repositions a file pointer associated with a stream previously opened by a call to fopen(). The file pointer is positioned ‘numbytes’ from the location ‘fromwhere’, which may be the file beginning, the current file pointer position, or the end of the file, symbolised by the constants SEEK_SET, SEEK_CUR and SEEK_END respectively. If a call to fseek() succeeds, a value of zero is returned.

Associated with fseek() is ftell(), which reports the current file pointer position of a stream, and has the following function prototype;

```
long int ftell(FILE *fp);
```

ftell() returns either the position of the file pointer, measured in bytes from the start of the file, or -1 upon an error occurring.

Handles

File handles are opened with the open() function that has the prototype;

```
int open(char *filename,int access[,unsigned mode]);
```

If open() is successful, the number of the file handle is returned. Otherwise open() returns -1.

The access integer is comprised from bitwise oring together of the symbolic constants declared in fcntl.h. These vary from compiler to compiler but may be;

O_APPEND	If set, the file pointer will be set to the end of the file prior to each write.
O_CREAT	If the file does not exist it is created.
O_TRUNC	Truncates the existing file to a length of zero bytes.
O_EXCL	Used with O_CREAT
O_BINARY	Opens the file in binary mode
O_TEXT	Opens file in text mode

The optional mode parameter is comprised by bitwise oring of the symbolic constants defined in stat.h. These vary from C compiler to C compiler but may be;

S_IWRITE	Permission to write
S_IREAD	Permission to read

Once a file handle has been assigned with open(), the file may be accessed with read() and write().

Read() has the function prototype;

```
int read(int handle, void *buf, unsigned num_bytes);
```

It attempts to read 'num_bytes' and returns the number of bytes actually read from the file handle 'handle', and stores these bytes in the memory block pointed to by 'buf'.

Write() is very similar to read() and has the same function prototype and return values, but writes 'num_bytes' from the memory block pointed to by 'buf'.

Files opened with open() are closed using close() that has the function prototype;

```
int close(int handle);
```

close() returns zero on success, and -1 if an error occurs trying to close the handle.

Random access is provided by lseek(), which is very similar to fseek(), except that it accepts an integer file handle as the first parameter rather than a stream FILE pointer.

This example uses file handles to read data from stdin (usually the keyboard) and copy the text to a new file called "data.txt".

```
#include <io.h>
#include <fcntl.h>
#include <sys\stat.h>

int main()
{
        int handle;
        char text[100];

        handle = open("data.txt",O_RDWR|O_CREAT|O_TRUNC,S_IWRITE);
```

```
		do
		{
			gets(text);
			write(handle,&text,strlen(text));
		}
		while(*text);

		close(handle);
	}
```

Advanced File I/O

The ANSI standard on C defines file IO as by way of file streams, and defines various functions for file access;

fopen() has the prototype;

```
	FILE *fopen(const char *name,const char *mode);
```

fopen() attempts to open a stream to a file name in a specified mode. If successful a FILE type pointer is returned to the file stream, if the call fails NULL is returned. The mode string can be one of the following;

Mode	Description
r	Open for reading only
w	Create for writing, overwriting any existing file with the same name.
a	Open for append (writing at end of file) or create the file if it does not exist.
r+	Open an existing file for reading and writing.
w+	Create a new file for reading and writing.
a+	Open for append with read and write access.

fclose() is used to close a file stream previously opened by a call to fopen(). It has the prototype;

```
	int fclose(FILE *fp);
```

When a call to fclose() is successful, all buffers to the stream are flushed and a value of zero is returned. If the call fails fclose() returns EOF.

Many host computers, and the IBM PC is no exception, use buffered file access, that is when writing to a file stream the data is stored in memory and only written to the stream when it exceeds a predefined number of bytes. A power failure occurring before the data has been written to the stream will result in the data never being written, so the function fflush() can be called to force all pending data to be written. fflush() has the prototype;

```
	int fflush(FILE *fp);
```

When a call to fflush() is successful, the buffers connected with the stream are flushed and a value of zero is returned. On failure fflush() returns EOF.

The location of the file pointer connected with a stream can be determined with the function ftell(). ftell() has the prototype;

```
	long int ftell(FILE *fp);
```

and returns the offset of the file pointer in bytes from the start of the file, or -1L if the call fails.

Similarly, you can move the file pointer to a new position with fseek(). fseek() has the prototype;

```
int fseek(FILE *fp, long offset, int from_what_place);
```

fseek() attempts to move the file pointer, 'fp' 'offset' bytes from the position 'from_what_place'. 'from_what_place' is predefined as one of;

SEEK_SET	The file's beginning
SEEK_CUR	The file pointer's current position
SEEK_END	End of file

The offset may be a positive value to move the file pointer on through the file, or negative to move backwards.

To move a file pointer quickly back to the start of a file, and clear any references to errors that have occurred C provides the function rewind() that has the prototype;

```
void rewind(FILE *fp);
```

rewind(fp) is similar to fseek(fp,0L,SEEK_SET) in that they both set the file pointer to the start of the file, but whereas fseek() clears the EOF error marker, rewind() clears all error indicators.

Errors occurring with file functions can be checked with the function ferror() that has the prototype;

```
int ferror(FILE *fp);
```

ferror() returns a nonzero value if an error has occurred on the specified stream. After checking ferror() and reporting any errors you should clear the error indicators, and this can be done by a call to clearerr() that has the prototype;

```
void clearerr(FILE *fp);
```

The condition of reaching end of file (EOF) can be tested for with the predefined macro feof() that has the prototype;

```
int feof(FILE *fp);
```

feof() returns a nonzero value if an end of file error indicator was detected on the specified file stream, and zero if the end of file has not yet been reached.

Reading data from a file stream can be achieved using several functions; A single character can be read with fgetc() that has the prototype;

```
int fgetc(FILE *fp);
```

fgetc() returns either the character read converted to an integer, or EOF if an error occurred.

Reading a string of data is achieved with fgets(). fgets() attempts to read a string terminated by a newline character and of no more than a specified number of bytes from the file stream. It has the prototype;

```
char *fgets(char s, int n, FILE *fp);
```

A successful call to fgets() results in a string being stored in ‘s’ that is either terminated by a newline character, or that is ‘n’ - 1 characters long, which ever came first. The newline character is retained by fgets(), and a null bytes is appended to the string. If the call fails a NULL pointer is returned.

Strings may be written to a stream using fputs() that has the prototype;

```
int fputs(const char *s,FILE *fp);
```

fputs() writes all the characters in the string ‘s’ to the stream ‘fp’ except the null terminating byte. On success, fputs() returns the last character written, on failure it returns EOF.

To write a single character to a stream use fputc() that has the prototype;

```
int fputc(int c,FILE *fp);
```

If successful, fputc() returns the character written, otherwise it returns EOF.

To read a large block of data, or a record from a stream you can use fread() that has the prototype;

```
size_t fread(void *ptr,size_t size, size_t n, FILE *fp);
```

fread() attempts to read 'n' items, each of length 'size' from the file stream 'fp' into the block of memory pointed to by 'ptr'. To check the success or failure status of fread() use ferror().

The sister function to fread() is fwrite() that has the prototype;

```
size_t fwrite(const void *ptr,size_t size, size_t n,FILE *fp);
```

that writes 'n' items each of length 'size' from the memory area pointed to by 'ptr' to the specified stream 'fp'.

Formatted input from a stream is achieved with fscanf() that has the prototype;

```
int fscanf(FILE *fp, const char *format[,address ...]);
```

fscanf() returns the number of fields successfully stored, and EOF on end of file. This short example shows how fscanf() is quite useful for reading numbers from a stream, but hopeless when it comes to strings!

```
#include <stdio.h>

void main()
{
        FILE *fp;
        int a;
        int b;
        int c;
        int d;
        int e;
        char text[100];

        fp = fopen("data.txt","w+");

        if(!fp)
        {
                perror("Unable to create file");
                exit(0);
        }

        fprintf(fp,"1 2 3 4 5 \"A line of numbers\"");

        fflush(fp);

        if (ferror(fp))
        {
```

```
                    fputs("Error flushing stream",stderr);
                    exit(1);
          }

          rewind(fp);
          if (ferror(fp))
          {
                    fputs("Error rewind stream",stderr);
                    exit(1);
          }

          fscanf(fp,"%d %d %d %d %d %s",&a,&b,&c,&d,&e,text);
          if (ferror(fp))
          {
                    fputs("Error reading from stream",stderr);
                    exit(1);
          }

          printf("\nfscanf() returned %d %d %d %d %d %s",a,b,c,d,e,text);
}
```

As you can see from the example, fprintf() can be used to write formatted data to a stream.

If you wish to store the position of a file pointer on a stream, and then later restore it to the same position you can use the functions fgetpos() and fsetpos(). fgetpos() reads the current location of the file pointer and has the prototype;

```
int fgetpos(FILE *fp, fpos_t *pos);
```

fsetpos() repositions the file pointer and has the prototype;

```
int fsetpos(FILE *fp, const fpos_t *fpos);
```

fpos_t is defined in stdio.h.

These functions are more convenient than doing an ftell() followed by an fseek().

An open stream can have a new file associated with it in place of the existing file by using the function freopen() that has the prototype;

```
FILE *freopen(const char *name,const char *mode,FILE *fp);
```

freopen() closes the existing stream and then attempts to reopens it with the specified file name. This is useful for redirecting the predefined streams stdin, stdout and stderr to a file or device.

For example; if you wish to redirect all output intended to stdout (usually the host computer's display device) to a printer you might use;

```
freopen("LPT1","w",stdout);
```

where LPT1 is the name of the printer device (on a PC host, LPT1 is the name of the parallel port).

Predefined I/O Streams

There are three predefined I/O streams; stdin, stdout, and stderr. The streams stdin and stdout default to the keyboard and display respectively, but can be redirected on some hardware platforms, such as the PC and under UNIX. The stream stderr defaults to the display and is not usually redirected by the operator. Thus it can be used for the display of error messages even when program output has been redirected, such as with;

```
fputs("Error message",stderr);
```

The functions printf() and puts(), output data to the stream stdout, and can therefore be redirected by the operator of the program. scanf() and gets() accept input from the stream stdin.

As an example of file i/o with the PC consider the following short program that does a hex dump of a specified file to the predefined stream stdout, which may be redirected to a file using;

```
dump filename.ext > target.ext
```

```
#include <stdio.h>
#include <fcntl.h>
#include <io.h>
#include <string.h>

main(int argc, char *argv[])
{
	unsigned counter;
	unsigned char v1[20];
	int f1;
	int x;
	int n;

	if (argc != 2)
	{
		fputs("\nERROR: Syntax is dump f1\n",stderr);
		return(1);
	}

	f1 = open(argv[1],O_RDONLY);

	if (f1 == -1)
	{
		fprintf(stderr,"\nERROR: Unable to open %s\n",argv[1]);
		return(1);
	}

	fprintf(stdout,"\nDUMP OF FILE %s\n\n",strupr(argv[1]));

	counter = 0;

	while(1)
	{
		/* Set buffer to zero bytes */
		memset(v1,0,20);

		/* Read buffer from file */
		x = _read(f1,&v1,16);

		/* x will be 0 on EOF or -1 on error */
		if (x < 1)
			break;

		/* Print file offset to stdout */
		fprintf(stdout,"%06d(%05x) ",counter,counter);

		counter += 16;
```

```
                        /* print hex values of buffer to stdout */
                        for(n = 0; n < 16; n++)
                                fprintf(stdout,"%02x ",v1[n]);

                        /* Print ascii values of buffer to stdout */
                        for(n = 0; n < 16; n++)
                        {
                                if ((v1[n] > 31) && (v1[n] < 128))
                                        fprintf(stdout,"%c",v1[n]);
                                else
                                        fputs(".",stdout);
                        }

                        /* Finish the line with a new line */
                        fputs("\n",stdout);
                }

                /* successful termination */
                return(0);
        }
```

STRINGS

The C language has one of the most powerful string handling capabilities of any general purpose computer language.

A string is a single dimension array of characters terminated by a zero byte.

Strings may be initialised in two ways. Either in the source code where they may be assigned a constant value, as in;

```
int main()
{
        char *p = "System 5";
        char name[] = "Test Program" ;
}
```

or at run time by the function strcpy() that has the function prototype;

```
char *strcpy(char *destination, char *source);
```

strcpy() copies the string pointed to by source into the location pointed to by destination as in the following example;

```
#include<stdio.h>

int main()
{
        char name[50];

        strcpy(name,"Servile Software");

        printf("\nName equals %s",name);
}
```

C also allows direct access to each individual byte of the string, so the following is quite permissible;

```
#include<stdio.h>

int main()
{
        char name[50];

        strcpy(name,"Servile Software");

        printf("\nName equals %s",name);

        /* Replace first byte with lower case 's' */
        name[0] = 's';

        printf("\nName equals %s",name);
}
```

The ANSI standard on the C programming language defines the following functions for use with strings;

char *strcat(char *dest, char *source)	Appends string source to the end of string destination.
char *strchr(char *s, int c)	Returns a pointer to the first occurence of character 'c' within s.

int strcmp(char *s1, char *s2)	Compares strings s1 and s2 returning < 0 if s1 is less than s2 == 0 if s1 and s2 are the same > 0 if s1 is greater than s2
int strcoll(char *s1, char *s2)	Compares strings s1 and s2 according to the collating sequence set by setlocale() returning < 0 if s1 is less than s2 == 0 if s1 and s2 are the same > 0 if s1 is greater than s2
char *strcpy(char *dest, char *src)	Copies string src into string dest.
unsigned strcspn(char *s1, char *s2)	Returns the length of string s1 that consists entirely of characters not in string s2.
unsigned strlen(char *s)	Returns the length of string s.
char *strncat(char *dest, char *src, unsigned len)	Copies at most 'len' characters from string src into string dest.
int strncmp(char *s1, char *s2, unsigned len)	Compares at most 'len' characters from string s1 with string s2 returning < 0 if s1 is less than s2 == 0 if s1 and s2 are the same > 0 if s1 is greater than s2
char *strncpy(char *dest, char *src, unsigned len)	Copies 'len' characters from string src into string dest, truncating or padding with zero bytes as required.
char *strpbrk(char *s1, char *s2)	Returns a pointer to the first character in string s1 that occurs in string s2.
char *strrchr(char *s, int c)	Returns a pointer to the last occurence of 'c' within string s.
unsigned strspn(char *s1, char *s2)	Returns the length of the initial segment of string s1 that consists entirely of characters in string s2.
char *strstr(char *s1, char *s2)	Returns a pointer to the first occurence of string s2 within string s1, or NULL if string s2 is not found in string s1.
char *strtok(char *s1, char *s2)	Returns a pointer to the token found in string s1 that is defined by delimiters in string s2. Returns NULLif no tokens are found.

The ANSI standard also defines various functions for converting strings into numbers and numbers into strings.

Some C compilers include functions to convert strings to upper and lower case, but these functions are not defined in the ANSI standard. However, the ANSI standard does define the functions; toupper() and tolower() that return an integer parameter converted to upper and lowercase respectively. By using these functions we can create our own ANSI compatible versions;

```
#include<stdio.h>

void strupr(char *source)
{
        char *p;
```

```
		p = source;
		while(*p)
		{
			*p = toupper(*p);
			p++;
		}
	}

	void strlwr(char *source)
	{
		char *p;

		p = source;
		while(*p)
		{
			*p = tolower(*p);
			p++;
		}
	}


	int main()
	{
		char name[50];

		strcpy(name,"Servile Software");

		printf("\nName equals %s",name);

		strupr(name);

		printf("\nName equals %s",name);

		strlwr(name);

		printf("\nName equals %s",name);
	}
```

C does not impose a maximum length that a string may be, unlike other computer languages. However, some CPUs impose restrictions on the maximum size a block of memory can be. For example, the 8088 family of CPUs, as used by the IBM PC, impose a limit of 64K bytes on a segment of memory.

An example program to reverse all the characters in a string.

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

	char *strrev(char *s)
	{
		/* Reverses the order of all characters in a string except the null */
		/* terminating byte */

		char *start;
		char *end;
		char tmp;

		/* Set pointer 'end' to last character in string */
		end = s + strlen(s) - 1;
```

```
          /* Preserve pointer to start of string */
          start = s;

          /* Swop characters */
          while(end >= s)
          {
                    tmp = *end;
                    *end = *s;
                    *s = tmp;
                    end--;
                    s++;
          }
          return(start);
}


main()
{
          char text[100];
          char *p;

          strcpy(text,"This is a string of data");

          p = strrev(text);

          printf("\n%s",p);
}
```

Strtok()

The function strtok() is a very powerful standard C feature for extracting substrings from within a single string. It is used where the substrings are separated by known delimiters, such as commas in the following example;

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

main()
{
          char data[50];
          char *p;

          strcpy(data,"RED,ORANGE,YELLOW,GREEN,BLUE,INDIGO,VIOLET");

          p = strtok(data,",");
          while(p)
          {
                    puts(p);
                    p = strtok(NULL,",");
          };
}
```

Or this program can be written with a for() loop thus;

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

```
main()
{
        char data[50];
        char *p;

        strcpy(data,"RED,ORANGE,YELLOW,GREEN,BLUE,INDIGO,VIOLET");

        for(strtok(data,","); p; p = strtok(NULL,","))
        {
                puts(p);
        };
}
```

They both compile to the same code but follow different programming styles.

Initially, you call strtok() with the name of the string variable to be parsed, and a second string that contains the known delimiters. Strtok() then returns a pointer to the start of the first substring and replaces the first token with a zero delimiter. Subsequent calls to strtok() can be made in a loop passing NULL as the string to be parsed, and strtok() will return the subsequent substrings.

Since strtok() can accept many delimiter characters in the second parameter string we can use it as the basis of a simple word counting program;

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

void main(int argc, char *argv[])
{
        FILE *fp;
        char buffer[256];
        char *p;
        long count;

        if (argc != 2)
        {
                fputs("\nERROR: Usage is wordcnt <file>\n",stderr);
                exit(0);
        }

        /* Open file for reading */
        fp = fopen(argv[1],"r");

        /* Check the open was okay */
        if (!fp)
        {
                fputs("\nERROR: Cannot open source file\n",stderr);
                exit(0);
        }

        /* Initialise word count */
        count = 0;

        do
        {
                /* Read a line of data from the file */
                fgets(buffer,255,fp);

                /* check for an error in the read or EOF */
                if (ferror(fp) || feof(fp))
                        continue;

                /* count words in received line */
                /* Words are defined as separated by the characters */
                /* \t(tab) \n(newline) , ; : . ! ? ( ) - and [space] */
                p = strtok(buffer,"\t\n,;:.!?()- ");
                while(p)
                {
                        count++;
                        p = strtok(NULL,"\t\n,;:.!?()- ");
                }
        }
        while(!ferror(fp) && !feof(fp));

        /* Finished reading. Was it due to an error? */
        if (ferror(fp))
        {
                fputs("\nERROR: Reading source file\n",stderr);
                fclose(fp);
                exit(0);
        }

        /* Reading finished due to EOF, quite valid so print count */
        printf("\nFile %s contains %ld words\n",argv[1],count);
        fclose(fp);
}
```

Converting Numbers To And From Strings

All C compilers provide a facility for converting numbers to strings. This being sprintf(). However, as happens sprintf() is a multi-purpose function that is therefore large and slow. The following function ITOS() accepts two parameters, the first being a signed integer and the second being a pointer to a character string. It then copies the integer into the memory pointed to by the character pointer. As with sprintf() ITOS() does not check that the target string is long enough to accept the result of the conversion. You should then ensure that the target string is long enough.

Example function for copying a signed integer into a string;

```
void ITOS(long x, char *ptr)
{
        /* Convert a signed decimal integer to a string */

        long pt[9] = { 100000000, 10000000, 1000000, 100000, 10000, 1000, 100, 10, 1 };
        int n;

        /* Check sign */
        if (x < 0)
        {
                *ptr++ = '-';
                /* Convert x to absolute */
                x = 0 - x;
        }
```

```
		for(n = 0; n < 9; n++)
		{
			if (x > pt[n])
			{
				*ptr++ = '0' + x / pt[n];
				x %= pt[n];
			}
		}
		return;
	}
```

To convert a string to a floating point number, C provides two functions; atof() and strtod(). atof() has the prototype;

```
	double atof(const char *s);
```

strtod has the prototype;

```
	double strtod(const char *s,char **endptr);
```

Both functions scan the string and convert it as far as they can, until they come across a character they don’t understand. The difference between the two functions is that if strtod() is passed a character pointer for parameter ‘endptr’, it sets that pointer to the first character in the string that terminated the conversion. Because of its better error reporting, by way of endptr, strtod() is often preferred to atof().

To convert a string to an integer use atoi() that has the prototype;

```
	int atoi(const char *s);
```

atoi() does not check for an overflow, and the results are undefined!

atol() is a similar function but returns a long. Alternatively, you can use strtol() and stroul() instead that have better error checking.

TEXT HANDLING

Human languages write information down as 'text'. This is comprised of words, figures and punctuation. The words being made up of upper case and lower case letters. Processing text with a computer is a commonly required task, and yet quite a difficult one.

The ANSI C definitions include string processing functions that are by their nature sensitive to case. That is the letter 'A' is seen as distinct from the letter 'a'. This is the first problem that must be overcome by the programmer. Fortunately both Borland's Turbo C compilers and Microsoft's C compilers include case insensitive forms of the string functions.

stricmp() for example is the case insensitive form of strcmp(), and strnicmp() is the case insensitive form of strncmp().

If you are concerned about writing portable code, then you must restrict yourself to the ANSI C functions, and write your own case insensitive functions using the tools provided.

Here is a simple implementation of a case insensitive version of strstr(). The function simply makes a copy of the parameter strings, converts those copies both to upper case and then does a standard strstr() on the copies. The offset of the target string within the source string will be the same for the copy as the original, and so it can be returned relative to the parameter string.

```
char *stristr(char *s1, char *s2)
{
        char c1[1000];
        char c2[1000];
        char *p;

        strcpy(c1,s1);
        strcpy(c2,s2);

        strupr(c1);
        strupr(c2);

        p = strstr(c1,c2);
        if (p)
                return s1 + (p - c1);
        return NULL;
}
```

This function scans a string, s1 looking for the word held in s2. The word must be a complete word, not simply a character pattern, for the function to return true. It makes use of the stristr() function described previously.

```
int word_in(char *s1,char *s2)
{
        /* return non-zero if s2 occurs as a word in s1 */
        char *p;
        char *q;
        int ok;

        ok = 0;
        q = s1;

        do
        {
                /* Locate character occurence s2 in s1 */
                p = stristr(q,s2);
                if (p)
                {
                    /* Found */
                    ok = 1;

                    if (p > s1)
                    {
                         /* Check previous character */
                         if (*(p - 1) >= 'A' && *(p - 1) <= 'z')
                              ok = 0;
                    }

                    /* Move p to end of character set */
                    p += strlen(s2);
                    if (*p)
                    {
                         /* Check character following */
                         if (*p >= 'A' && *p <= 'z')
                              ok = 0;
                    }
               }
               q = p;
          }
          while(p && !ok);
          return ok;
     }
```

Some more useful functions for dealing with text are truncstr() that truncates a string;

```
     void truncstr(char *p,int num)
     {
          /* Truncate string by losing last num characters */
          if (num < strlen(p))
               p[strlen(p) - num] = 0;
     }
```

trim() that removes trailing spaces from the end of a string;

```
     void trim(char *text)
     {
          /* remove trailing spaces */
          char *p;

          p = &text[strlen(text) - 1];
          while(*p == 32 && p >= text)
               *p-- = 0;
     }
```

strlench() that changes the length of a string by adding or deleting characters;

```
     void strlench(char *p,int num)
     {
          /* Change length of string by adding or deleting characters */

          if (num > 0)
               memmove(p + num,p,strlen(p) + 1);
          else
          {
               num = 0 - num;
```

```
                    memmove(p,p + num,strlen(p) + 1);
          }
     }
```

strins() that inserts a string into another string;

```
     void strins(char *p, char *q)
     {
          /* Insert string q into p */
          strlench(p,strlen(q));
          strncpy(p,q,strlen(q));
     }
```

strchg() that replaces all occurences of one sub-string with another within a target string;

```
     void strchg(char *data, char *s1, char *s2)
     {
          /* Replace all occurences of s1 with s2 */
          char *p;
          char changed;

          do
          {
                    changed = 0;
                    p = strstr(data,s1);
                    if (p)
                    {
                              /* Delete original string */
                              strlench(p,0 - strlen(s1));

                              /* Insert replacement string */
                              strins(p,s2);
                              changed = 1;
                    }
          }
          while(changed);
     }
```

TIME

C provides a function, time(), which reads the computer's system clock to return the system time as a number of seconds since midnight on January the first, 1970. However, this value can be converted to a useful string by the function ctime() as illustrated in the following example;

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

int main()
{
        /* Structure to hold time, as defined in time.h  */
        time_t t;

        /* Get system date and time from computer */
        t = time(NULL);
        printf("Today's date and time: %s\n",ctime(&t));
}
```

The string returned by ctime() is comprised of seven fields;

Day of the week,
Month of the year,
Date of the day of the month,
hour,
minutes,
seconds,
century of the year

terminated by a newline character and null terminating byte. Since the fields always occupy the same width, slicing operations can be carried out on the string with ease. The following program defines a structure 'time' and a function gettime() that extracts the hours, minutes and seconds of the current time and places them in the structure;

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

struct time
{
        int ti_min;             /* Minutes */
        int ti_hour;            /* Hours */
        int ti_sec;             /* Seconds */
};

void gettime(struct time *now)
{
        time_t t;
        char temp[26];
        char *ts;

        /* Get system date and time from computer */
        t = time(NULL);

        /* Translate dat and time into a string */
        strcpy(temp,ctime(&t));

        /* Copy out just time part of string */
```

```
                temp[19] = 0;
                ts = &temp[11];

                /* Scan time string and copy into time structure */
                sscanf(ts,"%2d:%2d:%2d",&now->ti_hour,&now->ti_min,&now->ti_sec);
        }

        int main()
        {
                struct time now;

                gettime(&now);

                printf("\nThe time is %02d:%02d:%02d",now.ti_hour,now.ti_min,now.ti_sec);

        }
```

The ANSI standard on C does actually provide a function ready made to convert the value returned by time() into a structure;

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

        int main()
        {
                time_t t;
                struct tm *tb;

                /* Get time into t */
                t = time(NULL);

                /* Convert time value t into structure pointed to by tb */
                tb = localtime(&t);

                printf("\nTime is %02d:%02d:%02d",tb->tm_hour,tb->tm_min,tb->tm_sec);
        }
```

The structure 'tm' is defined in time.h as;

```
        struct tm
        {
                int tm_sec;
                int tm_min;
                int tm_hour;
                int tm_mday;
                int tm_mon;
                int tm_year;
                int tm_wday;
                int tm_yday;
                int tm_isdst;
        };
```

Timers

Often a program must determine the date and time from the host computer’s non-volatile RAM. There are several time functions provided by the ANSI standard on C that allow a program to retrieve, from the host computer, the current date and time;

time() returns the number of seconds that have elapsed since midnight on January the 1st 1970. It has the prototype;

```
time_t time(time_t *timer);
```

time() fills in the time_t variable sent as a parameter and returns the same value. You can call time() with a NULL parameter and just collect the return value thus;

```
#include <time.h>

void main()
{
        time_t now;

        now = time(NULL);
}
```

asctime() converts a time block to a twenty six character string of the format;

```
Wed Oct 14 10:23:45 1992\n\0
```

asctime() has the prototype;

```
char *asctime(const struct tm *tblock);
```

ctime() converts a time value (as returned by time()) into a twenty six chracter string of the same format as asctime(). For example;

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

void main()
{
        time_t now;
        char date[30];

        now = time(NULL);
        strcpy(date,ctime(&now));
}
```

difftime() returns the difference, in seconds, between two values (as returned by time()). This can be useful for testing the elapsed time between two events, the time a function takes to execute, and for creating consistent delays that are irrelevant of the host computer.

An example delay program;

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


void DELAY(int period)
{
        time_t start;

        start = time(NULL);
        while(time(NULL) < start + period)
                ;
}
```

```
	void main()
	{
		printf("\nStarting delay now....(please wait 5 seconds)");

		DELAY(5);

		puts("\nOkay, I've finished!");
	}
```

gmtime() converts a local time value (as returned by time()) to the GMT time and stores it in a time block. This function depends upon the global variable timezone being set.

The time block is a predefined structure (declared in time.h) as follows;

```
	struct tm
	{
		int tm_sec;
		int tm_min;
		int tm_hour;
		int tm_mday;
		int tm_mon;
		int tm_year;
		int tm_wday;
		int tm_yday;
		int tm_isdst;
	};
```

tm_mday records the day of the month, ranging from 1 to 31; tm_wday is the day of the week with Sunday being represented by 0; the year is recorded less 1900; tm_isdst is a flag to show whether daylight saving time is in effect. The actual names of the structure and its elements may vary from compiler to compiler, but the structure should be the same in essence.

mktime() converts a time block to a calendar format. It follows the prototype;

```
				time_t mktime(struct tm *t);
```

The following example allows entry of a date, and uses mktime() to calculate the day of the week appropriate to that date. Only dates from the first of January 1970 are recognisable by the time functions.

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

	void main()
	{
		struct tm tsruct;
		int okay;
		char data[100];
		char *p;
		char *wday[] = {"Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday" ,
					"prior to 1970, thus not known" };
		do
		{
			okay = 0;
			printf("\nEnter a date as dd/mm/yy ");
			p = fgets(data,8,stdin);
			p = strtok(data,"/");
```

```
                        if (p != NULL)
                                tsruct.tm_mday = atoi(p);
                        else
                                continue;

                        p = strtok(NULL,"/");
                        if (p != NULL)
                                tsruct.tm_mon = atoi(p);
                        else
                                continue;

                        p = strtok(NULL,"/");

                        if (p != NULL)
                                tsruct.tm_year = atoi(p);
                        else
                                continue;
                        okay = 1;
                }
                while(!okay);

                tsruct.tm_hour = 0;
                tsruct.tm_min = 0;
                tsruct.tm_sec = 1;
                tsruct.tm_isdst = -1;

                /* Now get day of the week */
                if (mktime(&tsruct) == -1)
                tsruct.tm_wday = 7;

                printf("That was %s\n",wday[tsruct.tm_wday]);
        }
```

mktime() also makes the neccessary adjustments for values out of range, this can be utilised for discovering what the date will be in n number of days time thus;

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

        void main()
        {
                struct tm *tsruct;
                time_t today;

                today = time(NULL);
                tsruct = localtime(&today);

                tsruct->tm_mday += 10;
                mktime(tsruct);

                printf("In ten days it will be %02d/%02d/%2d\n", tsruct->tm_mday,tsruct->tm_mon + 1,tsruct-
        >tm_year);

        }
```

This program uses Julian Dates to decide any day of the week since the 1st of October 1582 when the Gregorian calendar was introduced.

```
char *WDAY(int day, int month, int year)
{
        /* Returns a pointer to a string representing the day of the week */

        static char *cday[] = { "Saturday","Sunday","Monday","Tuesday", "Wednesday","Thursday","Friday" };
        double yy;
        double yt;
        double j;
        int y1;
        int y4;
        int x;

        yy = year / 100;
        y1 = (int)(yy);
        yt = year / 400;
        y4 = (int)(yt);
        x = 0;

        if (month < 3)
        {
                year--;
                x = 12;
        }

        j = day + (int)(365.25*year);

        j += (int)(30.6001 * (month + 1 + x)) - y1 + y4;

        if (yy == y1 && yt != y4 && month < 3)
                j++;

        j = 1 + j - 7 * (int)(j/7);

        if (j > 6)
                j -= 7;

        return(cday[j]);
}
```

With time() and difftime() we can create a timer for testing the execution times of functions thus;

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

main()
{
        time_t now;
        time_t then;
        double elapsed;

        int n;

        now = time(NULL);
```

```
		/* This loop is adjustable for multiple passes */
		for(n = 0; n < 5000; n++)
			/* Call the function to test */
			func();

		then = time(NULL);

		elapsed = difftime(then,now);
		printf("\nElapsed seconds==%lf\n",elapsed);
	}
```

By way of time() and ctime() the current system date and time can be retrieved from the host computer thus;

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

	main()
	{
		time_t now;
		char *date;
		int n;

		/* Get system time */
		now = time(NULL);

		/* Convert system time to a string */
		date = ctime(&now);

		/*Display system time */
		printf("\nIt is %s",date);
	}
```

time_t is a type defined in time.h as the type of variable returned by time(). This type may vary from compiler to compiler, and therefore is represented by the type "time_t".

HEADER FILES

Function prototypes for library functions supplied with the C compiler, and standard macros are declared in header files.

The ANSI standard on the C programming language lists the following header files;

Header file	Description
assert.h	Defines the assert debugging macro
ctype.h	Character classification and conversion macros
errno.h	Constant mnemonics for error codes
float.h	Defines implementation specific macros for dealing with floating point mathematics
limits.h	Defines implementation specific limits on type values
locale.h	Country specific parameters
math.h	Prototypes for mathematics functions
setjmp.h	Defines typedef and functions for setjmp/longjmp
signal.h	Constants and declarations for use by signal() and raise()
stdarg.h	Macros for dealing with argument lists
stddef.h	Common data types and macros
stdio.h	Types and macros required for standard I/O
stdlib.h	Prototypes of commonly used functions and miscellany
string.h	String manipulation function prototypes
time.h	Structures for time conversion routines

DEBUGGING

The ANSI standard on C includes a macro function for debugging called assert(). This expands to an if() statement, which if it returns true terminates the program and outputs to the standard error stream a message comprised of:

```
Assertion failed: <test>, file <module>, line <line number>
Abnormal program termination
```

For example, the following program accidentally assigns a zero value to a pointer!

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

main()
{
        /* Demonstration of assert */

        int *ptr;
        int x;

        x = 0;

        /* Whoops! error in this line! */
        ptr = x;

        assert(ptr != NULL);
}
```

When run, this program terminates with the message:

```
Assertion failed: ptr != 0, file TEST.C, line 16
Abnormal program termination
```

When a program is running okay, the assert() functions can be removed from the compiled program by simply adding the line;

```
#define NDEBUG
```

before the #include <assert.h> line. Effectively the assert functions are commented out in the preprocessed source before compilation, this means that the assert expressions are not evaluated, and thus cannot cause any side effects.

FLOAT ERRORS

Floating point numbers are decimal fractions, decimal fractions do not accurately equate to normal fractions as not every number will divide precisely by ten. This creates the potential for rounding errors in calculations that use floating point numbers. The following program illustrates one such example of rounding error problems;

```
#include <stdio.h>

void main()
{
        float number;

        for(number = 1; number > 0.4; number -= 0.01)
                printf("\n%f",number);
}
```

At about 0.47 (depending upon the host computer and compiler) the program starts to store an inaccurate value for 'number'.

This problem can be minimised by using longer floating point numbers, doubles or long doubles that have larger storage space allocated to them. For really accurate work though, you should use integers and only convert to a floating point number for display. You also should notice that most C compilers default floating point numbers to ‘doubles’ and when using ‘float’ types have to convert the double down to a float!

ERROR HANDLING

When a system error occurs within a program, for example when an attempt to open a file fails, it is helpful to the program's user to display a message reporting the failure. Equally it is useful to the program's developer to know why the error occurred, or at least as much about it as possible. To this end the ANSI standard on C describes a function, perror(), which has the prototype;

```
        void perror(const char *s);
```

and is used to display an error message. The program's own prefixed error message is passed to perror() as the string parameter. This error message is displayed by perror() followed by the host's system error separated by a colon. The following example illustrates a use for perror();

```
        #include <stdio.h>

        void main()
        {
                FILE *fp;
                char fname[] = "none.xyz";

                fp = fopen(fname,"r");

                if(!fp)
                        perror(fname);
                return;
        }
```

If the fopen() operation fails, a message similar to;

```
        none.xyz: No such file or directory
```

is displayed.

You should note, perror() sends its output to the predefined stream 'stderr', which is usually the host computer's display unit.

perror() finds its message from the host computer via the global variable 'errno' that is set by most, but not all system functions.

Unpleasant errors might justify the use of abort(). abort() is a function that terminates the running program with a message;

```
        "Abnormal program termination"
```

and returns an exit code of 3 to the parent process or operating system.

Critical Error Handling With The IBM PC AND DOS

The IBM PC DOS operating system provides a user amendable critical error handling function. This function is usually discovered by attempting to write to a disk drive that does not have a disk in it, in which case the familiar;

```
Not ready error writing drive A
Abort Retry Ignore?
```

Message is displayed on the screen. Fine when it occurs from within a DOS program, not so fine from within your own program!

The following example program shows how to redirect the DOS critical error interrupt to your own function;

```
/* DOS critical error handler test */

#include <stdio.h>
#include <dos.h>

void interrupt new_int();
void interrupt (*old_int)();

char status;

main()
{
        FILE *fp;

        old_int = getvect(0x24);

        /* Set critical error handler to my function */
        setvect(0x24,new_int);

        /* Generate an error by not having a disc in drive A */
        fp = fopen("a:\\data.txt","w+");

        /* Display error status returned */
        printf("\nStatus ==  %d",status);

}

void interrupt new_int()
{
        /* set global error code */
        status = _DI;

        /* ignore error and return */
        _AL = 0;
}
```

When the DOS critical error interrupt is called, a status message is passed in the low byte of the DI register. This message is one of;

Code	Meaning
00	Write-protect error
01	Unknown unit
02	Drive not ready
03	Unknown command
04	Data error, bad CRC
05	Bad request structure length
06	Seek error
07	Unknown media type

08	Sector not found
09	Printer out of paper
0A	Write error
0B	Read error
0C	General failure

Your critical error interrupt handler can transfer this status message into a global variable, and then set the result message held in register AL to one of;

Code	Action
00	Ignore error
01	Retry
02	Terminate program
03	Fail (Available with DOS 3.3 and above)

If you choose to set AL to 02, terminate program, you should ensure ALL files are closed first since DOS will terminate the program abruptly, leaving files open and memory allocated, not a pretty state to be in!

The example program shown returns an ignore status from the critical error interrupt, and leaves the checking of any errors to the program itself. So, in this example after the call to fopen() we could check the return value in fp, and if it reveals an error (NULL in this case) we could then check the global variable status and act accordingly, perhaps displaying a polite message to the user to put a disk in the floppy drive and ensure that the door is closed.

The following is a practical function for checking whether a specified disc drive can be accessed. It should be used with the earlier critical error handler and global variable 'status'.

```
int DISCOK(int drive)
{
        /* Checks for whether a disc can be read */
        /* Returns false (zero) on error */
        /* Thus if(!DISCOK(drive)) */
        /*                  error();  */

        unsigned char buffer[25];

        /* Assume okay */
        status = 0;

        /* If already logged to disc, return okay */
        if ('A' + drive == diry[0])
                return(1);

        /* Attempt to read disc */
        memset(buffer,0,20);
        sprintf(buffer,"%c:$$$.$$$",'A'+drive);

        _open(buffer,O_RDONLY);

        /* Check critical error handler status */
        if (status == 0)
                return(1);

        /* Disc cannot be read */
        return(0);
```

```
            }
```

CAST

Casting tells the compiler what a data type is, and can be used to change a data type. For example, consider the following;

```
#include <stdio.h>

void main()
{
        int x;
        int y;

        x = 10;
        y = 3;

        printf("\n%lf",x / y);
}
```

The printf() function has been told to expect a double. However, the compiler sees the variables ‘x’ and ‘y’ as integers, and an error occurs! To make this example work we must tell the compiler that the result of the expression x / y is a double, this is done with a cast thus;

```
#include <stdio.h>

void main()
{
        int x;
        int y;

        x = 10;
        y = 3;

        printf("\n%lf",(double)(x / y));
}
```

Notice the data type ‘double’ is enclosed by parenthesis, and so is the expression to convert. But now, the compiler knows that the result of the expression is a double, but it still knows that the variables ‘x’ and ‘y’ are integers and so an integer division will be carried out. We have to cast the constants thus;

```
#include <stdio.h>

void main()
{
        int x;
        int y;

        x = 10;
        y = 3;

        printf("\n%lf",(double)(x) / (double)(y));
}
```

Because both of the constants are doubles, the compiler knows that the outcome of the expression will also be a double.

THE IMPORTANCE OF PROTOTYPING

Prototyping a function involves letting the compiler know in advance what type of values a function will receive and return. For example, lets look at strtok(). This has the prototype;

```
char *strtok(char *s1, const char *s2);
```

This prototype tells the compiler that strtok() will return a character pointer, the first received parameter will be a pointer to a character string, and that string can be changed by strtok(), and the last parameter will be a pointer to a character string that strtok() cannot change.

The compiler knows how much space to allocate for the return parameter, sizeof(char *), but without a prototype for the function the compiler will assume that the return value of strtok() is an integer, and will allocate space for a return type of int, that is sizeof(int). If an integer and a character pointer occupy the same number of bytes on the host computer no major problems will occur, but if a character pointer occupies more space than an integer, then the compiler wont have allocated enough space for the return value and the return from a call to strtok() will overwrite some other bit of memory. If, as so often happens the return value is returned via the stack, the results of confusing the compiler can be disastrous!

Thankfully most C compilers will warn the programmer if a call to a function has been made without a prototype, so that you can add the required function prototypes.

Consider the following example that will not compile on most modern C compilers due to the nasty error in it;

```
#include <stdio.h>

int FUNCA(int x, int y)
{
        return(MULT(x,y));
}

double MULT(double x, double y)
{
        return(x * y);
}


main()
{
        printf("\n%d",FUNCA(5,5));
}
```

When the compiler first encounters the function MULT() it is as a call from within FUNCA(). In the absence of any prototype for MULT() the compiler assumes that MULT() returns an integer. When the compiler finds the definition for function MULT() it sees that a return of type double has been declared. The compiler then reports an error in the compilation saying something like;

```
"Type mismatch in redclaration of function 'MULT'"
```

What the compiler is really trying to say is, prototype your functions before using them! If this example did compile, and was then run it probably would crash the computer’s stack and cause a system hang.

POINTERS TO FUNCTIONS

C allows a pointer to point to the address of a function, and this pointer to be called rather than specifying the function. This is used by interrupt changing functions and may be used for indexing functions rather than using switch statements. For example;

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

double (*fp[7])(double x);

void main()
{
        double x;
        int p;

        fp[0] = sin;
        fp[1] = cos;
        fp[2] = acos;
        fp[3] = asin;
        fp[4] = tan;
        fp[5] = atan;
        fp[6] = ceil;

        p = 4;

        x = fp[p](1.5);
        printf("\nResult %lf",x);
}
```

This example program defines an array of pointers to functions, (*fp[])() that are then called dependant upon the value in the indexing variable p. This program could also be written;

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

void main()
{
        double x;
        int p;

        p = 4;

        switch(p)
        {
                case 0 :  x = sin(1.5);
                          break;
                case 1 :  x = cos(1.5);
                          break;
                case 2 :  x = acos(1.5);
                          break;
                case 3 :  x = asin(1.5);
                          break;
                case 4 :  x = tan(1.5);
                          break;
                case 5 :  x = atan(1.5);
                          break;
```

```
                    case 6 :  x = ceil(1.5);
                              break;
          }
          puts("\nResult %lf",x);
}
```

The first example, using pointers to the functions, compiles into much smaller code and executes faster than the second example.

The table of pointers to functions is a useful facility when writing language interpreters, the program compares an entered instruction against a table of key words that results in an index variable being set and then the program simply needs to call the function pointer indexed by the variable, rather than wading through a lengthy switch() statement.

DANGEROUS PITFALLS

One of the most dangerous pitfalls can occur with the use of gets(). This function accepts input from the stream stdin until it receives a newline character, which it does not pass to the program. All the data it receives is stored in memory starting at the address of the specified string, and quite happily overflowing into other variables! This danger can be avoided by using fgets() that allows a maximum number of characters to be specified, so you can avoid overflow problems. Notice though that fgets() does retain the newline character scanf() is another function best avoided. It accepts input from stdin and stores the received data at the addresses provided to it. If those addresses are not really addresses where the data ends up is anybodys guess!

This example is okay, since scanf() has been told to store the data at the addresses occupied by the two variables ‘x’ and ‘y’.

```
void main()
{
        int x;
        int y;

        scanf("%d%d",&x,&y);
}
```

But in this example scanf() has been told to store the data at the addresses suggested by the current values of ‘x’ and ‘y’! An easy and common mistake to make, and yet one that can have very peculiar effects.

```
void main()
{
        int x;
        int y;

        scanf("%d%d",x,y);
}
```

The answer is, don’t use scanf(), use fgets() and parse your string manually using the standard library functions strtod(), strtol() and strtoul().

Here is the basis of a simple input string parser that returns the individual input fields from an entered string;

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

void main()
{
        char input[80];
        char *p;

        puts("\nEnter a string ");
        fgets(input,79,stdin);

        /* now parse string for input fields */
        puts("The fields entered are:");
        p = strtok(input,", ");
        while(p)
        {
                puts(p);
                p = strtok(NULL,", ");
        }
```

```
          }
```

SIZEOF

A preprocessor instruction, ‘sizeof’, returns the size of an item, be it a structure, pointer, string or whatever. However! take care when using ‘sizeof’. Consider the following program;

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

char string1[80]; char *text = "This is a string of data" ;

void main()
{
        /* Initialise string1 correctly */
        memset(string1,0,sizeof(string1));

        /* Copy some text into string1 ? */
        memcpy(string1,text,sizeof(text));

        /* Display string1 */
        printf("\nString 1 = %s\n",string1);
}
```

What it is meant to do is initialise all 80 elements of string1 to zeroes, which it does alright, and then copy the constant string ‘text’ into the variable ‘string1’. However, variable text is a pointer, and so the sizeof(text) instruction returns the size of the character pointer (perhaps two bytes) rather than the length of the string pointed to by the pointer! If the length of the string pointed to by ‘text’ happened to be the same as the size of a character pointer then no error would be noticed.

INTERRUPTS

The IBM PC BIOS and DOS contain functions that may be called by a program by way of the function's interrupt number. The address of the function assigned to each interrupt is recorded in a table in RAM called the interrupt vector table. By changing the address of an interrupt vector a program can effectively disable the original interrupt function and divert any calls to it to its own function. This was done by the critical error handler described in the section on error handling.

Borland's Turbo C provides two library functions for reading and changing an interrupt vector. These are: setvect() and getvect(). The corresponding Microsoft C library functions are: _dos_getvect() and _dos_setvect().

getvect() has the function prototype;

```
void interrupt(*getvect(int interrupt_no))();
```

setvect() has the prototype;

```
void setvect(int interrupt_no, void interrupt(*func)());
```

To read and save the address of an existing interrupt a program uses getvect() thus;

```
/* Declare variable to record old interrupt */
void interrupt(*old)(void);

main()
{
        /* get old interrupt vector */
        old = getvect(0x1C);
        .
        .
        .
}
```

Where 0x1C is the interrupt vector to be retrieved.

To then set the interrupt vector to a new address, our own function, we use setvect() thus;

```
void interrupt new(void)
{
        .
        .
        /* New interrupt function */
        .
        .
        .
}

main()
{
        .
        .
        .
        setvect(0x1C,new);
        .
        .
        .
        .
        }
```

There are two important points to note about interrupts;

First, if the interrupt is called by external events then before changing the vector you MUST disable the interrupt callers using disable() and then re-enable the interrupts after the vector has been changed using enable(). If a call is made to the interrupt while the vector is being changed ANYTHING could happen!

Secondly, before your program terminates and returns to DOS you must reset any changed interrupt vectors! The exception to this is the critical error handler interrupt vector that is restored automatically by DOS, so your program needn’t bother restoring it.

This example program hooks the PC clock timer interrupt to provide a background clock process while the rest of the program continues to run. If included with your own program that requires a constantly displayed clock on screen, you need only amend the display coordinates in the call to puttext(). Sincle the closk display code is called by a hardware issued interrupt, your program can start the clock and forget it until it terminates.

```
        /* Compile in LARGE memory model */

        #include <stdio.h>
        #include <dos.h>
        #include <time.h>
        #include <conio.h>
        #include <stdlib.h>

        enum { FALSE, TRUE };

        #define COLOUR          (BLUE << 4) | YELLOW

        #define BIOS_TIMER      0x1C

        static unsigned installed = FALSE;
        static void interrupt (*old_tick) (void);

        static void interrupt tick (void)
        {
                int i;
                struct tm *now;
                time_t this_time;
                char time_buf[9];
                static time_t last_time = 0L;
                static char video_buf[20] =
                {
                        ' ', COLOUR, '0', COLOUR, '0', COLOUR, ':', COLOUR, '0', COLOUR,
                        '0', COLOUR, ':', COLOUR, '0', COLOUR, '0', COLOUR, ' ', COLOUR
                };

                enable ();

                if (time (&this_time) != last_time)
                {
                        last_time = this_time;

                        now = localtime(&this_time);

                        sprintf(time_buf, "%02d:%02d.%02d",now->tm_hour,now->tm_min,now->tm_sec);

                        for (i = 0; i < 8; i++)
                        {
                                video_buf[(i + 1) << 1] = time_buf[i];
                        }

                        puttext (71, 1, 80, 1, video_buf);
                }

                old_tick ();
        }

        void stop_clock (void)
        {
                if (installed)
                {
                        setvect (BIOS_TIMER, old_tick);
                        installed = FALSE;
                }
        }

        void start_clock (void)
        {
                static unsigned first_time = TRUE;

                if (!installed)
                {
                        if (first_time)
                        {
                                atexit (stop_clock);
                                first_time = FALSE;
                        }

                        old_tick = getvect (BIOS_TIMER);
                        setvect (BIOS_TIMER, tick);
                        installed = TRUE;
                }
        }
```

SIGNAL

Interrupts raised by the host computer can be trapped and diverted in several ways. A simple method is to use signal().

Signal() takes two parameters in the form;

```
void (*signal (int sig, void (*func) (int))) (int);
```

The first parameter, 'sig' is the signal to be caught. These are often predefined by the header file 'signal.h'.

The second parameter is a pointer to a function to be called when the signal is raised. This can either be a user function, or a macro defined in the header file 'signal.h' to do some arbitrary task, such as ignore the signal for example.

On a PC platform, it is often useful to disable the 'ctrl-break' key combination that is used to terminate a running program by the user. The following PC signal() call replaces the predefined signal 'SIGINT', which equates to the ctrl-break interrupt request, with the predefined macro 'SIG-IGN', ignore the request;

```
signal(SIGINT,SIG_IGN);
```

This example catches floating point errors on a PC, and zero divisions!

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

void (*old_sig)();

void catch(int sig)
{
        printf("Catch was called with: %d\n",sig);
}


void main()
{
        int a;
        int b;

        old_sig = signal(SIGFPE,catch);

        a = 0;
        b = 10 / a;

        /* Restore original handler before exiting! */
        signal(SIGFPE,old_sig);
}
```

SORTING AND SEARCHING

The ANSI C standard defines qsort(), a function for sorting a table of data. The function follows the format;

```
        qsort(void *base,size_t elements,size_t width,int (*cmp)(void *, void *));
```

The following short program illustrates the use of qsort() with a character array.

```
        #include <string.h>

        main()
        {
                int n;
                char data[10][20];

                /* Initialise some arbirary data */

                strcpy(data[0],"RED");
                strcpy(data[1],"BLUE");
                strcpy(data[2],"GREEN");
                strcpy(data[3],"YELLOW");
                strcpy(data[4],"INDIGO");
                strcpy(data[5],"BROWN");
                strcpy(data[6],"BLACK");
                strcpy(data[7],"ORANGE");
                strcpy(data[8],"PINK");
                strcpy(data[9],"CYAN");

                /* Sort the data table */
                qsort(data[0],10,20,strcmp);

                /* Print the data table */
                for(n = 0; n < 10; n++)
                        puts(data[n]);
        }
```

Here is a program that implements the shell sort algorithm (this one is based on the routine in K & R), which sorts arrays of pointers based upon the data pointed to by the pointers;

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

        #define LINELEN         80
        #define MAXLINES        2000

        char *lines[MAXLINES];
        int lastone;

        void SHELL(void);

        void SHELL()
        {
                /* SHELL Sort Courtesy of K & R */
                int gap;
                int i;
                int j;
                char temp[LINELEN];

                for(gap = lastone / 2; gap > 0; gap /= 2)
                for(i = gap; i < lastone; i++)
                        for(j = i - gap; j >= 0 && strcmp(lines[j] , lines[j + gap]) >
                          0; j -= gap)
                        {
                                strcpy(temp,lines[j]);
                                strcpy(lines[j] , lines[j + gap]);
                                strcpy(lines[j + gap] , temp);

                        }
        }

        void main(int argc, char *argv[])
        {
                FILE *fp;
                char buff[100];
                int n;

                /* Check command line parameter has been given */
                if (argc != 2)
                {
                        printf("\nError: Usage is SERVSORT file");
                        exit(0);
                }

                /* Attempt to open file for updating */
                fp = fopen(argv[1],"r+");
                if (fp == NULL)
                {
                        printf("\nError: Unable to open %s",argv[1]);
                        exit(0);
                }

                /* Initialise element counter to zero */
                lastone = 0;

                /* Read file to be sorted */
                while((fgets(buff,100,fp)) != NULL)
                {
                        /* Allocate memory block*/
                        lines[lastone] = malloc(LINELEN);
                        if (lines[lastone] == NULL)
                        {
                                printf("\nError: Unable to allocate memory");
                                fclose(fp);
                                exit(0);
                        }
                        strcpy(lines[lastone],buff);
                        lastone++;

                        if (lastone > MAXLINES)
                        {
                                printf("\nError: Too many lines in source file");
                                exit(0);
                        }
                }
                /* Call sort function */
                SHELL();

                /* Close file */
                fclose(fp);

                /* Reopen file in create mode */
                fp = fopen(argv[1],"w+");

                /* Copy sorted data from memory to disk */
                for(n = 0; n < lastone; n++)
                        fputs(lines[n],fp);

                /* Close file finally */
                fclose(fp);

                /* Return to calling program */
                return(1);
        }
```

If we want to use qsort() with a table of pointers we have to be a bit more clever than usual.

This example uses the colours again, but this time they are stored in main memory and indexed by a table of pointers. Because we have a table of pointers to sort there are two differences between this program's qsort() and the previous one;

First we can't use strcmp() as the qsort() comparison function, secondly the width of the table being sorted is sizeof(char *), that is the size of a character pointer.

Notice the comparison function cmp() that receives two parameters, both are pointers to a pointer. qsort() sends to this function the values held in data[], which are in turn pointers to the data. So we need to use this indirection to locate the data, otherwise we would be comparing the addresses at which the data is held rather than the data itself!

```
#include <alloc.h>
#include <string.h>

/* Function prototype for comparison function */
int (cmp)(char **,char **);

int cmp(char **s1, char **s2)
{
        /* comparison function using pointers to pointers */
        return(strcmp(*s1,*s2));
}

main()
{
        int n;
        char *data[10];

        for(n = 0; n < 10; n++)
                data[n] = malloc(20);

        strcpy(data[0],"RED");
        strcpy(data[1],"BLUE");
```

```
                strcpy(data[2],"GREEN");
                strcpy(data[3],"YELLOW");
                strcpy(data[4],"INDIGO");
                strcpy(data[5],"BROWN");
                strcpy(data[6],"BLACK");
                strcpy(data[7],"ORANGE");
                strcpy(data[8],"PINK");
                strcpy(data[9],"CYAN");

                /* The data table is comprised of pointers */
                /* so the call to qsort() must reflect this */
                qsort(data,10,sizeof(char *),cmp);

                for(n = 0; n < 10; n++)
                        puts(data[n]);
        }
```

The quick sort is a fast sorting algorithm that works by subdividing the data table into two sub-tables and then subdividing the sub-tables. As it subdivides the table, so it compares the elements in the table and swaps them as required.

The following program implements the quick sort algorithm, which is usually already used by qsort();

```
        #include <string.h>

        #define MAXELE  2000

        char data[10][20];
        int lastone;

        void QKSORT()
        {
                /* Implementation of QUICKSORT algorithm */

                int i;
                int j;
                int l;
                int p;
                int r;
                int s;
                char temp[100];
                static int sl[MAXELE][2];

                /* sl[] is an index to the sub-table */

                l = 0;
                r = lastone;
                p = 0;

                do
                {
                        while(l < r)
                        {
                                i = l;
                                j = r;
                                s = -1;
                        while(i < j)
                        {
                                if (strcmp(data[i],data[j]) > 0)
                                {
                                        strcpy(temp,data[i]);
                                        strcpy(data[i],data[j]);
                                        strcpy(data[j],temp);
                                        s = 0 - s;
                                }

                                if (s == 1)
                                        i++;
                                else
                                        j--;
                        }

                        if (i + 1 < r)
                        {
                                p++;
                                sl[p][0] = i + 1;
                                sl[p][1] = r;
                        }
                        r = i - 1;
                }
                if (p != 0)
                {
                        l = sl[p][0];
                        r = sl[p][1];
                        p--;
                }
        }
        while(p > 0);
}

main()
{
        int n;

        /* Initialise arbitrary data */
        strcpy(data[0],"RED");
        strcpy(data[1],"BLUE");
        strcpy(data[2],"GREEN");
        strcpy(data[3],"YELLOW");
        strcpy(data[4],"INDIGO");
        strcpy(data[5],"BROWN");
        strcpy(data[6],"BLACK");
        strcpy(data[7],"ORANGE");
        strcpy(data[8],"PINK");
        strcpy(data[9],"CYAN");

        /* Set last element indicator */
        lastone = 9;

        /* Call quick sort function */
        QKSORT();

        /* Display sorted list */
        for(n = 0; n < 10; n++)
```

```
                        puts(data[n]);

        }
```

A table sorted into ascending order can be searched with bsearch(), this takes the format;

```
        bsearch(key,base,num_elements,width,int (*cmp)(void *, void *));
```

bsearch() returns a pointer to the first element in the table that matches the key, or zero if no match is found.

Or you can write your own binary search function thus;

```
        int BSRCH(char *key, void *data, int numele, int width)
        {
                /* A binary search function returning one if found */
                /* Zero if not found */

                int bp;
                int tp;
                int mp;
                int result;
                char *p;

                bp = 0;
                tp = numele;
                mp = (tp + bp) / 2;

                /* Locate element mp in table by assigning pointer to start */
                /* and incrementing it by width * mp */
                p = data;
                p += width * mp;

                while((result = strcmp(p,key)) != 0)
                {
                        if (mp >= tp)
                                /* Not found! */
                                return(0);
                        if (result < 0)
                                bp = mp + 1;
                        else
                                tp = mp - 1;

                        mp = (bp + tp) / 2;
                        p = data;
                        p += width * mp;
                }
                return(1);
        }

        void main()
        {
                int result;
                char data[10][20];

                /* Initialise some arbirary data */

                strcpy(data[0],"RED");
                strcpy(data[1],"BLUE");
                strcpy(data[2],"GREEN");
                strcpy(data[3],"YELLOW");
                strcpy(data[4],"INDIGO");
                strcpy(data[5],"BROWN");
                strcpy(data[6],"BLACK");
                strcpy(data[7],"ORANGE");
                strcpy(data[8],"PINK");
                strcpy(data[9],"CYAN");

                /* Sort the data table */
                qsort(data[0],10,20,strcmp);

                result = BSRCH("CYAN",data[0],10,20);

                printf("\n%s\n",(result == 0) ? "Not found" : "Located okay");
        }
```

There are other sorting algorithms as well. This program incorporates the QUICK SORT, BUBBLE SORT, FAST BUBBLE SORT, INSERTION SORT and SHELL SORT for comparing how each performs on a random 1000 item string list;

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

        char data[1000][4];
        char save[1000][4];

        int lastone;

        void INITDATA(void);
        void QKSORT(void);
        void SHELL(void);
        void BUBBLE(void);
        void FBUBBLE(void);
        void INSERTION(void);
        void MKDATA(void);

        void QKSORT()
        {
                /* Implementation of QUICKSORT algorithm */

                int i;
                int j;
                int l;
                int p;
                int r;
                int s;
                char temp[20];
                static int sl[1000][2];

                l = 0;
                r = lastone;
                p = 0;

                do
                {
                        while(l < r)
                {
                        i = l;
                        j = r;
                        s = -1;

                        while(i < j)
                        {
                                if (strcmp(data[i],data[j]) > 0)
                                {
                                        strcpy(temp,data[i]);
                                        strcpy(data[i],data[j]);
                                        strcpy(data[j],temp);
                                        s = 0 - s;
                                }

                                if (s == 1)
                                        i++;
                                else
                                        j--;
                        }

                        if (i + 1 < r)
                        {
                                p++;
                                sl[p][0] = i + 1;
                                sl[p][1] = r;
                        }
                        r = i - 1;
                }
                if (p != 0)
                {
                        l = sl[p][0];
                        r = sl[p][1];
                        p--;
                }
        }
        while(p > 0);
}

void SHELL()
{
        /* SHELL Sort Courtesy of K & R */

        int gap;
        int i;
        int j;
        char temp[20];

        for(gap = lastone / 2; gap > 0; gap /= 2)
        for(i = gap; i < lastone; i++)
                for(j = i - gap; j >= 0 && strcmp(data[j] , data[j + gap]) > 0;
                         j -= gap)
                {
                        strcpy(temp,data[j]);
                        strcpy(data[j] , data[j + gap]);
                        strcpy(data[j + gap] , temp);
                }
}
```

```
void BUBBLE()
{
        int a;
        int b;
        char temp[20];

        for(a = lastone; a >= 0; a--)
        {
                for(b = 0; b < a; b++)
                {
                        if(strcmp(data[b],data[b + 1]) > 0)
                        {
                                strcpy(temp,data[b]);
                                strcpy(data[b] , data[b + 1]);
                                strcpy(data[b + 1] , temp);
                        }
                }
        }
}

void FBUBBLE()
{
        /* bubble sort with swap flag*/

        int a;
        int b;
        int s;
        char temp[20];

        s = 1;

        for(a = lastone; a >= 0 && s == 1; a--)
        {
                s = 0;
                for(b = 0; b < a; b++)
                {
                        if(strcmp(data[b],data[b + 1]) > 0)
                        {
                                strcpy(temp,data[b]);
                                strcpy(data[b] , data[b + 1]);
                                strcpy(data[b + 1] , temp);
                                s = 1;
                        }
                }
        }
}

void INSERTION()
{
        int a;
        int b;
        char temp[20];

        for(a = 0; a < lastone; a++)
        {
                b = a;
                strcpy(temp,data[a + 1]);
```

```
                    while(b >= 0)
                    {
                              if (strcmp(temp,data[b]) < 0)
                              {
                                        strcpy(data[b+1],data[b]);
                                        b--;
                              }
                              else
                                        break;
                    }
                    strcpy(data[b+1],temp);
          }
}

void MKDATA()
{
          /* Initialise arbitrary data */
          /* Uses random(), which is not ANSI C */
          /* Returns a random number between 0 and n - 1 */

          int n;
          for(n = 0; n < 1000; n++)
                    sprintf(save[n],"%d",random(1000));
}

void INITDATA()
{
          int n;

          for(n = 0 ; n < 1000; n++)
                    strcpy(data[n],save[n]);
}

void main()
{
          MKDATA();

          /* Initialise arbitrary data */
          INITDATA();

          /* Set last element indicator */
          lastone = 999;

          /* Call quick sort function */
          QKSORT();


          /* Initialise arbitrary data */
          INITDATA();

          /* Set last element indicator */
          lastone = 1000;

          /* Call shell sort function */
          SHELL();

          /* Initialise arbitrary data */
          INITDATA();
```

```
                /* Set last element indicator */
                lastone = 999;

                /* Call bubble sort function */
                BUBBLE();

                /* Initialise arbitrary data */
                INITDATA();

                /* Set last element indicator */
                lastone = 999;

                /* Call bubble sort with swap flag function */
                FBUBBLE();

                /* Initialise arbitrary data */
                INITDATA();

                /* Set last element indicator */
                lastone = 999;

                /* Call insertion sort function */
                INSERTION();
        }
```

Here are the profiler results of the above test program run on 1000 and 5000 random items;

```
STRING SORT - 1000 RANDOM ITEMS

FBUBBLE         26.436 sec  41% |********************************************
BUBBLE          26.315 sec  41% |*******************************************
INSERTION       10.210 sec  15% |***************
SHELL           0.8050 sec   1%  |*
QKSORT          0.3252 sec  <1% |

STRING SORT - 5000 RANDOM ITEMS

FBUBBLE         563.70 sec  41% |********************************************
BUBBLE          558.01 sec  41% |********************************************
INSERTION       220.61 sec  16% |***************
SHELL           5.2531 sec  <1% |
QKSORT          0.8379 sec  <1% |
```

Here is the same test program amended for sorting tables of integers;

```
        /* Integer sort test program */

        #include <stdio.h>
        #include <stdlib.h>

        void INITDATA(void);
        void QKSORT(void);
        void SHELL(void);
        void BUBBLE(void);
        void FBUBBLE(void);
        void INSERTION(void);
        void MKDATA(void);
```

```
int data[1000];
int save[1000];

int lastone;

void QKSORT()
{
        /* Implementation of QUICKSORT algorithm */

        int i;
        int j;
        int l;
        int p;
        int r;
        int s;
        int temp;
        static int sl[1000][2];

        l = 0;
        r = lastone;
        p = 0;

        do
        {
                while(l < r)
                {
                        i = l;
                        j = r;
                        s = -1;

                        while(i < j)
                        {
                                if (data[i] > data[j])
                                {
                                        temp = data[i];
                                        data[i] = data[j];
                                        data[j] = temp;
                                        s = 0 - s;
                                }

                                if (s == 1)
                                        i++;
                                else
                                        j--;
                        }

                        if (i + 1 < r)
                        {
                                p++;
                                sl[p][0] = i + 1;
                                sl[p][1] = r;
                        }
                        r = i - 1;
                }
                if (p != 0)
                {
                        l = sl[p][0];
                        r = sl[p][1];
                        p--;
                }
        }
        while(p > 0);
}

void SHELL()
{
        /* SHELL Sort Courtesy of K & R */

        int gap;
        int i;
        int j;
        int temp;

        for(gap = lastone / 2; gap > 0; gap /= 2)
        for(i = gap; i < lastone; i++)
                for(j = i - gap; j >= 0 && data[j] > data[j + gap];
                         j -= gap)
                {
                        temp = data[j];
                        data[j] = data[j + gap];
                        data[j + gap] = temp;
                }
}

void BUBBLE()
{
        int a;
        int b;
        int temp;

        for(a = lastone; a >= 0; a--)
        {
                for(b = 0; b < a; b++)
                {
                        if(data[b] > data[b + 1])
                        {
                                temp = data[b];
                                data[b] = data[b + 1];
                                data[b + 1] = temp;
                        }
                }
        }
}

void FBUBBLE()
{
        /* bubble sort with swap flag */

        int a;
        int b;
        int s;
        int temp;

        s = 1;
```

```
		for(a = lastone; a >= 0 && s == 1; a--)
		{
			s = 0;
			for(b = 0; b < lastone - a; b++)
			{
				if(data[b] > data[b + 1])
				{
					temp = data[b];
					data[b] = data[b + 1];
					data[b + 1] = temp;
					s = 1;
				}
			}
		}
	}

	void INSERTION()
	{
		int a;
		int b;
		int temp;

		for(a = 0; a < lastone; a++)
		{
			b = a;
			temp = data[a + 1];
			while(b >= 0)
			{
				if (temp < data[b])
				{
					data[b+1] = data[b];
					b--;
				}
				else
					break;
			}
			data[b+1] = temp;
		}
	}

	void MKDATA()
	{
		int n;

		for(n = 0; n < 1000; n++)
			save[n] = random(1000);
	}

	void INITDATA()
	{
		int n;

		for(n = 0; n < 1000; n++)
		data[n] = save[n];
	}

	void main()
	{
```

```
                int n;

                /* Create 1000 random elements */
                MKDATA();

                /* Initialise arbitrary data */
                INITDATA();

                /* Set last element indicator */
                lastone = 999;

                /* Call quick sort function */
                QKSORT();

                /* Initialise arbitrary data */
                INITDATA();

                /* Set last element indicator */
                lastone = 1000;

                /* Call shell sort function */
                SHELL();

                /* Initialise arbitrary data */
                INITDATA();

                /* Set last element indicator */
                lastone = 999;

                /* Call bubble sort function */
                BUBBLE();

                /* Initialise arbitrary data */
                INITDATA();

                /* Set last element indicator */
                lastone = 999;

                /* Call bubble sort with swap flag function */
                FBUBBLE();

                /* Initialise arbitrary data */
                INITDATA();

                /* Set last element indicator */
                lastone = 999;

                /* Call insertion sort function */
                INSERTION();
        }
```

And here are the profiler results for this program;

```
INTEGER SORTS - 1000 RANDOM NUMBERS (0 - 999)

FBUBBLE              3.7197 sec  41% |*****************************************
BUBBLE               3.5981 sec  39% |***************************************
INSERTION            1.4258 sec  15% |***************
```

```
SHELL                   0.1207 sec   1%  |*
QKSORT                  0.0081 sec  <1% |

INTEGER SORTS - 5000 RANDOM NUMBERS (0 - 999)

FBUBBLE                 92.749 sec  42% |*********************************************
BUBBLE                  89.731 sec  41% |*********************************************
INSERTION               35.201 sec  16% |***************
SHELL                   0.9838 sec  <1% |
QKSORT                  0.0420 sec  <1% |

INTEGER SORTS - 5000 RANDOM NUMBERS (0 - 99)

FBUBBLE                 92.594 sec  42% |******************************************
BUBBLE                  89.595 sec  40% |*****************************************
INSERTION               35.026 sec  16% |***************
SHELL                   0.7563 sec  <1% |
QKSORT                  0.6018 sec  <1% |

INTEGER SORTS - 5000 RANDOM NUMBERS (0 - 9)

FBUBBLE                 89.003 sec  41% |********************************************
BUBBLE                  86.921 sec  40% |********************************************
INSERTION               31.544 sec  14% |***************
QKSORT                  6.0358 sec   2%  |**
SHELL                   0.5424 sec  <1% |

INTEGER SORTS - 5000 DESCENDING ORDERED NUMBERS

FBUBBLE                 122.99 sec  39% |*******************************************
BUBBLE                  117.22 sec  37% |*****************************************
INSERTION               70.595 sec  22% |**********************
SHELL                   0.6438 sec  <1% |
QKSORT                  0.0741 sec  <1% |

INTEGER SORTS - 5000 ORDERED NUMBERS

BUBBLE                  62.908 sec  99% |*******************************************
SHELL                   0.3971 sec  <1% |
INSERTION               0.0510 sec  <1% |
QKSORT                  0.0382 sec  <1% |
FBUBBLE                 0.0251 sec  <1% |

INTEGER SORTS - 10000 RANDOM NUMBERS (0 - 999)

FBUBBLE                 371.18 sec  42% |*****************************************
BUBBLE                  359.06 sec  41% |****************************************
INSERTION               140.88 sec  16% |**************
SHELL                   2.0423 sec  <1% |
QKSORT                  0.6183 sec  <1% |
```

Theory has it that the performance of a sorting algorithm is dependant upon;

a) the number of items to be sorted and
b) how unsorted the list is to start with.

With this in mind it is worth testing the various sorting routines described here to determine which one will best suit your particular application. If you examine the above profiler results you will see that:

1) With an already sorted list FBUBBLE() executes fastest

2) With a random list of small variations between the values SHELL() executes fastest

3) With a random list of large variations between the values QKSORT() executes the fastest

What the profiler does not highlight is that when the comparison aspect of a sort function takes a disproportionately long time to execute in relation to the rest of the sort function, then the bubble sort with a swap flag will execute faster than the bubble sort with out a swap flag.

When considering a sort routine take into consideration memory constraints and the type of data to be sorted as well as the relative performances of the sort functions. Generally, the faster a sort operates, the more memory it requires. Compare the simple bubble sort with the quick sort, and you will see that the quick sort requires far more memory than the bubble sort.

DYNAMIC MEMORY ALLOCATION

If a program needs a table of data, but the size of the table is variable, perhaps for a list of all file names in the current directory, it is inefficient to waste memory by declaring a data table of the maximum possible size. Rather it is better to dynamically allocate the table as required.

Turbo C allocates RAM as being available for dynamic allocation into an area called the “heap”. The size of the heap varies with memory model. The tiny memory model defaults to occupy 64K of RAM. The small memory model allocates upto 64K for the program/code and heap with a far heap being available within the remainder of conventional memory. The other memory models make all conventional memory available to the heap. This is significant when programming in the tiny memory model when you want to reduce the memory overhead of your program to a minimum. The way to do this is to reduce the heap to a minimum size. The smallest is 1 byte.

C provides a function malloc() which allocates a block of free memory of a specified size and returns a pointer to the start of the block; it also provides free() which deallocates a block of memory previously allocated by malloc(). Notice, however, that the IBM PC doesnot properly free blocks of memory, and contiuous use of malloc() and free() will fragmentise memory, eventually causing no memory to be available untill the program terminates.

This program searches a specified file for a specified string (with case sensitivity). It uses malloc() to allocate just enough memory for the file to be read into memory.

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

char *buffer;

void main(int argc, char *argv[])
{
        FILE *fp;
        long flen;

        /* Check number of parameters */
        if (argc != 3)
        {
                fputs("Usage is sgrep <text> <file spec>",stderr);
                exit(0);
        }

        /* Open stream fp to file */
        fp = fopen(argv[2],"r");
        if (!fp)
        {
                perror("Unable to open source file");
                exit(0);
        }

        /* Locate file end */
        if(fseek(fp,0L,SEEK_END))
        {
                fputs("Unable to determine file length",stderr);
                fclose(fp);
                exit(0);
        }

        /* Determine file length */
        flen = ftell(fp);
        /* Check for error */
        if (flen == -1L)
        {
                fputs("Unable to determine file length",stderr);
                fclose(fp);
                exit(0);
        }

        /* Set file pointer to start of file */
        rewind(fp);

        /* Allocate memory buffer */
        buffer = malloc(flen);

        if (!buffer)
        {
                fputs("Unable to allocate memory",stderr);
                fclose(fp);
                exit(0);
        }

        /* Read file into buffer */
        fread(buffer,flen,1,fp);

        /* Check for read error */
        if(ferror(fp))
        {
                fputs("Unable to read file",stderr);

                /* Deallocate memory block */
                free(buffer);

                fclose(fp);
                exit(0);
        }

        printf("%s %s in %s",argv[1],(strstr(buffer,argv[1])) ? "was found" : "was not found",argv[2]);

        /* Deallocate memory block before exiting */
        free(buffer);
        fclose(fp);
}
```

VARIABLE ARGUMENT LISTS

Some functions, such as printf(), accept a variable number and type of arguments. C provides a mechanism to write your own functions which can accept a variable argument list. This mechanism is the va_ family defined in the header file ‘stdarg.h’.

There are three macros which allow implementation of variable argument lists; va_start(), va_arg() and va_end() and a variable type va_list which defines an array which holds the information required by the macros.

va_start() takes two parameters, the first is the va_list variable and the second is the last fixed parameter sent to the function. va_start() must be called before attempting to access the variable argument list as it sets up pointers required by the other macros.

va_arg() returns the next variable from the argument list. It is called with two parameters, the first is the va_list variable and the second is the type of the argument to be extracted. So, if the next variable in the argument list is an integer, it can be extracted with;

```
<int> = va_arg(<va_list>,int);
```

va_end() is called after extracting all required variables from the argument list, and simply tidies up the internal stack if appropriate. va_end() accepts a single parameter, the va_list variable.

The following simple example program illustrates the basis for a printf() type implementation where the types of the arguments is not known, but can be determined from the fixed parameter. This example only caters for integer, string and character types, but could easily by extended to cater for other variable types as well by following the method illustrated;

```
#include <stdarg.h>

char *ITOS(long x, char *ptr)
{
        /* Convert a signed decimal integer to a string */

        long pt[9] = { 100000000, 10000000, 1000000, 100000, 10000, 1000, 100, 10, 1 };
        int n;

        /* Check sign */
        if (x < 0)
        {
                *ptr++ = '-';
                /* Convert x to absolute */
                x = 0 - x;
        }

        for(n = 0; n < 9; n++)
        {
                if (x > pt[n])
                {
                        *ptr++ = 48 + x / pt[n];
                        x %= pt[n];
                }
        }
        return(ptr);
}

void varfunc(char *format, ...)
```

```
	{
		va_list arg_ptr;
		char output[1000];
		char *ptr;
		int bytes;
		int x;
		char *y;
		char z;

		/* Initialise pointer to argument list */
		va_start(arg_ptr, format);

		/* loop format string */
		ptr = output;
		bytes = 0;
		while(*format)
		{
			/* locate next argument */
			while(*format != '%')
			{
				*ptr++ = *format++;
				bytes++;
			}
			/* A % has been located */
			format++;
			switch(*format)
			{
				case '%' :  *ptr++ = '%';
				            break;

				case 'd' : /* integer expression follows */
				           x = va_arg(arg_ptr,int);
				           ptr = ITOS(x,ptr);
				           *ptr = 0;
				           format++;
				           bytes += strlen(output) - bytes;
				           break;

				case 's' : /* String expression follows */
				           y = va_arg(arg_ptr,char *);
				           strcat(output,y);
				           x = strlen(y);
				           format++;
				           ptr += x;
				           bytes += x;
				           break;

				case 'c' : /* Char expression follows */
				           z = va_arg(arg_ptr,char);
				           *ptr++ = z;
				           format++;
				           bytes++;
				           break;
			}

		}

		/* Clean stack just in case! */
```

```
        va_end(arg_ptr);

        /* Null terminate output string */
        *ptr = 0;

        /* Display what we got */
        printf("\nOUTPUT==%s",output);
}

void main()
{
        varfunc("%d %s %c",5,"hello world",49);
}
```

A simpler variation is to use vsprintf() rather than implementing our own variable argument list access. However, it is beneficial to understand how variable argument lists behave. The following is a simplification of the same program, but leaving the dirty work to the compiler;

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

void varfunc(char *format, ...)
{
        va_list arg_ptr;
        char output[1000];

        va_start(arg_ptr, format);

        vsprintf(output,format,arg_ptr);

        va_end(arg_ptr);

        /* Display what we got */
        printf("\nOUTPUT==%s",output);
}

void main()
{
        varfunc("%d %s %c",5,"hello world",49);
}
```

TRIGONOMETRY FUNCTIONS

The ANSI standard on C defines a number of trigonometry functions, all of which accept an angle parameter in radians;

FUNCTION	PROTOTYPE	DESCRIPTION
acos	double acos(double x)	arc cosine of x
asin	double asin(double x)	arc sine of x
atan	double atan(double x)	arc tangent of x
atan2	double atan2(double x, double y)	arc tangent of y/x
cos	double cos(double x)	cosine of x
cosh	double cosh(double x)	hyperbolic cosine of x
sin	double sin(double x)	sine of x
sinh	double sinh(double x)	hyperbolic sine of x
tan	double tan(double x)	tangent of x
tanh	double tanh(double x)	hyperbolic tangent of x

There are 2PI radians in a circle, therefore 1 radian is equal to 360/2PI degrees or approximately 57 degrees in 1 radian. The calculation of any of the above functions requires large floating point numbers to be used which is a very slow process. If you are going to use calls to a trig' function, it is a good idea to use a lookup table of values rather than keep on calling the function. This approach is used in the discussion on circle drawing later in this book.

ATEXIT

When ever a program terminates, it should close any open files (this is done for you by the C compiler's startup/termination code which it surrounds your program with), and restore the host computer to some semblance of order. Within a large program where exit may occur from a number of locations it is a pain to have to keep on writing calls to the cleanup routine. Fortunately we don't have to!

The ANSI standard on C describes a function, atexit(), which registers the specified function, supplied as a parameter to atexit(), as a function which is called immediately before terminating the program. This function is called automatically, so the following program calls 'leave()' whether an error occurs or not;

```
#include <stdio.h>

void leave()
{
        puts("\nBye Bye!");
}

void main()
{
        FILE *fp;
        int a;
        int b;
        int c;
        int d;
        int e;
        char text[100];

        atexit(leave);

        fp = fopen("data.txt","w");

        if(!fp)
        {
                perror("Unable to create file");
                exit(0);
        }

        fprintf(fp,"1 2 3 4 5 \"A line of numbers\"");

        fflush(fp);

        if (ferror(fp))
        {
                fputs("Error flushing stream",stderr);
                exit(1);
        }

        rewind(fp);
        if (ferror(fp))
        {
                fputs("Error rewind stream",stderr);
                exit(1);
        }

        fscanf(fp,"%d %d %d %d %d %s",&a,&b,&c,&d,&e,text);
```

```
                if (ferror(fp))
                {
                        /* Unless you noticed the deliberate bug earlier */
                        /* The program terminates here */
                        fputs("Error reading from stream",stderr);
                        exit(1);
                }

                printf("\nfscanf() returned %d %d %d %d %d %s",a,b,c,d,e,text);
        }
```

INCREASING SPEED

In order to reduce the time your program spends executing it is essential to know your host computer. Most computers are very slow at displaying information on the screen. And the IBM PC is no exception to this. C offers various functions for displaying data, printf() being one of the most commonly used and also the slowest. Whenever possible try to use puts(varname) in place of printf(“%s\n”,varname). Remembering that puts() appends a newline to the string sent to the screen.

When multiplying a variable by a constant which is a factor of 2 many C compilers will recognise that a left shift is all that is required in the assembler code to carry out the multiplication rapidly. When multiplying by other values it is often faster to do a multiple addition instead, so;

```
'x * 3' becomes 'x + x + x'
```

Don’t try this with variable multipliers in a loop because it becomes very slow! But, where the multiplier is a constant it can be faster. (Sometimes!) Another way to speed up multiplication and division is with the shift commands, << and >>.

The instruction x /= 2 can equally well be written x >>= 1, shift the bits of x right one place. Many compilers actually convert integer divisions by 2 into a shift right instruction. You can use the shifts for multiplying and dividing by 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024 &c. If you have difficulty understanding the shift commands consider the binary form of a number;

```
01001101        equal to   77
```

shifted right one place it becomes;

```
00100110        equal to   38
```

Try to use integers rather than floating point numbers where ever possible. Sometimes you can use integers where you didn’t think you could! For example, to convert a fraction to a decimal one would normally say;

```
percentage = x / y * 100
```

This requires floating point variables. However, it can also be written as;

```
z = x * 100;
percentage = z / y
```

Which works fine with integers, so long as you don’t mind the percentage being truncated. eg;

```
5 / 7 * 100 is equal to 71.43 with floating point
```

but with integers;

```
5 * 100 / 7 is equal to 71
```

(Assuming left to right expression evaluation. You may need to force the multiplication to be done first as with ‘z = x * 100’).

Here is a test program using this idea;

```
float funca(double x, double y)
{
        return(x / y * 100);
}

int funcb(int x,int y)
{
        return(x * 100 / y);
}

void main()
{
        int n;
        double x;
        int y;

        for(n = 0; n < 5000; n++)
        {
                x = funca(5,7);
                y = funcb(5,7);
        }
}
```

And here is the results of the test program fed through a profiler;

```
funca           1.9169 sec  96% |********************************************
funcb           0.0753 sec   3%  |*
```

You can clearly see that the floating point function is 25 times slower than the integer equivalent!

NB: Although it is normal practice for expressions to be evaluated left to right, the ANSI standard on C does not specify an order of preference for expression evaluation, and as such you should check your compiler manual.

Another way of increasing speed is to use pointers rather than array indexing. When you access an array through an index, for example with;

```
x = data[i];
```

the compiler has to calculate the offset of data[i] from the beginning of the array. A slow process. Using pointers can often improve things as the following two bubble sorts, one with array indexing and one with pointers illustrates;

```
void BUBBLE()
{
        /* Bubble sort using array indexing */

        int a;
        int b;
        int temp;

        for(a = lastone; a >= 0; a--)
        {
                for(b = 0; b < a; b++)
                {
                        if(data[b] > data[b + 1])
                        {
                                temp = data[b];
                                data[b] = data[b + 1];
                                data[b + 1] = temp;
                        }
                }
        }
}
```

```
void PTRBUBBLE()
{
        /* Bubble sort using pointers */

        int temp;
        int *ptr;
        int *ptr2;

        for(ptr = &data[lastone]; ptr >= data; ptr--)
        {
                for(ptr2 = data; ptr2 < ptr; ptr2++)
                {
                        if(*ptr2 > *(ptr2 + 1))
                        {
                                temp = *ptr2;
                                *ptr2 = *(ptr2 + 1);
                                *(ptr2 + 1) = temp;
                        }
                }
        }
}
```

Here are the profiler results for the two versions of the same bubble sort operating on the same 1000 item, randomly sorted list;

```
BUBBLE          3.1307 sec  59% |*******************************************
PTRBUBBLE       2.1686 sec  40% |***************************
```

Here is another example of how to initialise an array using first the common indexing approach, and secondly the pointer approach;

```
/* Index array initialisation */
int n;

for(n = 0; n < 1000; n++)
        data[n] = random(1000);


/* Pointer array initialisation */
int *n;

for(n = data; n < &data[1000]; n++)
        *n = random(1000);
```

Needless to say, the pointer approach is faster than the index. The pointer approach is only really of benefit when an array is going to be traversed, as in the above examples. In the case of say a binary search where a different and non-adjacent element is going to be tested each pass then the pointer approach is no better than using array indexing.

The exception to this rule of using pointers rather than indexed access, comes with pointer to pointers. Say your program has declared a table of static data, such as:

```
static char *colours[] = { "Black", "Blue", "Green", "Yellow", "Red", "White" };
```

It is faster to access the table with colours[n] than it is with a pointer, since each element in the table colours[], is a pointer. If you need to scan a string table for a value you can use this very fast approach instead;

First the table is changed into a single string, with some delimiter between the elements.

```
static char *colours = "Black/Blue/Green/Yellow/Red/White";
```

Then to confirm that a value is held in the table you can use strstr();

```
result = strstr(colours,"Cyan");
```

Using in-line assembler code can provide the greatest speed increase. Care must be taken however not to interfere with the compiled C code. It is usually safe to write a complete function with in-line assembler code, but mixing in-line assembler with C code can be hazardous. As a rule of thumb, get your program working without assembler code, and then if you want to use in-line assembler, convert small portions of the code at a time, testing the program at each stage. Video I/O is a very slow process with C, and usually benefits from in-line assembler, and we have used this principle quite widely in the example programs which follow later.

PC GRAPHICS

When programming graphics you should bear in mind that they are a machine dependant subject. Code to produce graphics on an IBM PC will not port to an Amiga, or VAX or any other type of computer.

Introduction To PC Graphics

The IBM PC and compatible range of computers display information on a visual display unit (VDU). To enable the computer to send information to the VDU a component is included within the computer called a "display card". There are various display cards and VDUs which have been produced for the IBM PC range of computers; monochrome display adapter (MDA), colour graphics adapter (CGA), Hercules graphics card (HGC), Enhanced graphics adapter (EGA), video graphics array (VGA), super video graphics array (SVGA), memory controller gate array (MCGA), 8514/A and the Txas Instruments Graphics Architecture (TIGA). For simplicity, this section will concern itself only with the three more common types of display;

CGA, EGA and VGA

Information about the VGA display is also relevant to the SVGA display which in simple terms can do the same and more. This section will not concern itself with monochrome displays since they are of limited use in graphics work.

Display Modes

When an IBM PC computer is first switched on is set to display 80 columns by 25 rows of writing. This measurement, 80 x 25 is called the "resolution". A display mode which is intended for displaying writing is called a "text" mode. Where as a display mode which is intended for displaying pictures is called a "graphics" mode.

If you look closely at the display you will see that each displayed character is comprised of dots. In reality the entire display is comprised of dots, called "pixels", which may be set to different colours. In text display modes these pixels are not very relevant, however, in graphics display modes each pixel may be selected and set by a program. The size of the pixels varies with the display resolution. In 80 x 25 text mode the pixels are half the width they are in 40 x 25 text display mode.

Depending upon the display card installed in the computer, there are a number of display modes which may be used;

MODE	TYPE	RESOLUTION	COLOURS
0	Text	40 x 25	4 (CGA), 16 (EGA, VGA) Shades of grey
1	Text	40 x 25	4 (CGA), 16 (EGA, VGA)
2	Text	80 x 25	4 (CGA), 16 (EGA, VGA) Shades of grey
3	Text	80 x 25	4 (CGA), 16 (EGA, VGA)
4	Graphics	320 x 200	4
5	Graphics	320 x 200	4 (grey on CGA and EGA)
6	Graphics	640 x 200	2
7	Text	80 x 25	Mono (EGA, VGA)
13	Graphics	320 x 200	16 (EGA, VGA)
14	Graphics	640 x 200	16 (EGA, VGA)
15	Graphics	640 x 350	Mono (EGA, VGA)
16	Graphics	640 x 350	16 (EGA, VGA)
17	Graphics	640 x 480	2 (VGA)
18	Graphics	640 x 480	16 (VGA)
19	Graphics	320 x 200	256 (VGA)

The term resolution in graphics modes refers to the number of pixels across and down the VDU. The larger the number of pixels, the smaller each is and the sharper any displayed image appears. As you can see from the table, the VGA display can produce a higher resolution than the other display cards, resulting in sharper images being produced.

The CGA display card can produce a maximum resolution of 320 x 200 pixels, where as the VGA display card can produce a resolution of 640 x 480 pixels. This is why writing on a VGA VDU looks so much sharper than the writing displayed on a CGA VDU.

Accessing The Display

Inside the IBM PC computer is a silicon chip called the BIOS ROM, this chip contains functions which may be called by an external computer program to access the display card, which in turn passes the information on to the VDU. The BIOS display functions are all accessed by generating interrupt 10 calls, with the number of the appropriate function stored in the assembly language AH register.

A programmer interested in creating graphics displays must first be able to switch the display card to an appropriate graphics mode. This is achieved by calling the BIOS display function 0, with th number of the desired display mode from the table stored in the assembly language AL register thus the following assembly language code segment will switch the display card to CGA graphics mode 4, assuming that is that the display card is capable of this mode;

```
        mov     ah , 00
        mov     al , 04
        int     10h
```

A C function for selecting video display modes can be written;

```
#include <dos.h>

void setmode(unsigned char mode)
{
        /* Sets the video display mode */

        union REGS inregs outreg;

        inreg.h.ah = 0;
        inreg.h.al = mode;
        int86(0x10,&inreg,&outregs);
}
```

Any graphics are created by setting different pixels to different colours, this is termed “plotting”, and is achieved by calling BIOS display function 12 with the pixel’s horizontal coordinate in the assembly language CX register and it’s vertical coordinate in the assembly language DX register and the required colour in the assembly language AL register thus;

```
        mov     ah, 12
        mov     al, colour
        mov     bh, 0
        mov     cx, x_coord
        mov     dx, y_coord
        int     10h
```

The corresponding C function is;

```
#include <dos.h>

void plot(int x_coord, int y_coord, unsigned char colour)
{
        /* Sets the colour of a pixel */

        union REGS regs;

        regs.h.ah = 12;
        regs.h.al = colour;
        regs.h.bh = 0;
        regs.x.cx = x_coord;
        regs.x.dx = y_coord;
        int86(0x10,&regs,&regs);
}
```

The inverse function of plot is to read the existing colour setting of a pixel. This is done by calling BIOS ROM display function 13, again with the pixel’s horizontal coordinate in the assembly language CX register and it’s vertical coordinate in the assembly language DX register. This function then returns the pixel’s colour in the assembly language AL register;

```
#include <dos.h>

unsigned char get_pixel(int x_coord, int y_coord)
{
        /* Reads the colour of a pixel */

        union REGS inreg, outreg;

        inreg.h.ah = 13;
        inreg.h.bh = 0;
        inreg.x.cx = x_coord;
        inreg.x.dx = y_coord;
        int86(0x10,&inreg,&outreg);
        return(outreg.h.al);
}
```

Colour And The CGA

The CGA display card can display a maximum of 4 colours simultaneously at any time. However, the display card can generate a total of 8 colours. There are two sets of colours, called “palettes”. The first palette contains the colours;

background, cyan, magenta and white.

the second palette contains the colours;

background, green, red and yellow.

Colour 0 is always the same as the background colour.

The pixels displayed on the VDU take their colours from the currently active palette, and are continuously being refreshed. So, if the active palette changes, so to do the colours of the displayed pixels on the VDU.

Selection of the active CGA palette is achieved by calling the BIOS display function 11 with the number of the desired palette (either 0 or 1) in the assembly language BH register;

```
          mov     ah, 11
          mov     bh, palette
          int     10h
```

The C function for selecting the CGA palette is;

```
void palette(unsigned char palette)
{
          union REGS inreg, outreg;

          inreg.h.ah = 11;
          inreg.h.bh = palette;
          int86(0x10,&inreg,&outreg);
}
```

The background colour may be selected independantly from any of the eight available colours by calling the same BIOS display function with a value of 0 stored in the assembly language BH register and the desired background colour in the assembly language BL register;

```
mov     ah, 11
mov     bh, 0
mov     bl, colour
int     10h
```

In C this function can be written;

```
void background(unsigned char colour)
{
          union REGS inreg, outreg;

          inreg.h.ah = 11;
          inreg.h.bh = 0;
          inreg.h.bl = colour;
          int86(0x10,&inreg,&outreg);
}
```

The background colours available are;

0	Black
1	Blue
2	Green
3	Cyan
4	Red
5	Magenta
6	Yellow
7	White

Colour And The EGA

The EGA display card can display a maximum of 16 colours simultaneously at any time. The colour of all pixels is continuously refreshed by the display card by reading the colour from the EGA palette. Unlike the CGA display card, the EGA display card allows you to redefine any or all of the colours in the palette. Unfortunately only the first 8 colours may be loaded into other palette colours.

The colours are;

0	Black
1	Blue
2	Green
3	Cyan
4	Red
5	Magenta
6	Brown
7	Light grey
8	Dark grey
9	Light blue
10	Light green
11	Light cyan
12	Light red
13	Light magenta
14	Yellow
15	White

Changing a palette colour is achieved by calling the BIOS display function 16 with a value of 0 in the assembly language AL register, the colour value (0 to 7) in the assembly language BH register and the number of the palette colour (0 to 15) in the assembly language BL register thus;

```
        mov     ah,16
        mov     al,0
        mov     bl,palette
        mov     bh,colour
        int     10h
```

In C this function may be written;

```
        void ega_palette(unsigned char colour, unsigned char palette)
        {
                union REGS inreg,outreg;

                inreg.h.ah = 16;
                inreg.h.al = 0;
                inreg.h.bl = palette;
                inreg.h.bh = colour;

                int86(0x10,&inreg,&outreg);
        }
```

Colour And The VGA

The VGA display card can display a maximum of 256 colours on the VDU at any one time, these colours are defined by information held in 256 special registers called "DAC" registers. As with the CGA and EGA displays, the colour of displayed pixels is continuously being refreshed, and as such any change to a DAC register is immediately visible. Each DAC register has three component data parts which record the amount of green, blue and red colours which make up the displayed colour. Each of these seperate data components can hold a value between 0 and 63 giving the VGA display card the ability to display 262,144 colours! Although only a small subset of them can be displayed at any one time.

Setting the value of a DAC register is achieved by calling the BIOS display function 16 with a value of 16 stored in the assembly language AL register, the green value stored in the assembly language CH register, the blue value stored in the assembly language CL register and the red value stored in the assembly language DH register and the number of the DAC register to be set stored in the assembly language BX register;

```
        mov     ah,16
        mov     al,16
        mov     ch,green
        mov     cl,blue
        mov     dh,red
        mov     bx,dac
        int     10h
```

The C function to set a DAC register looks lik this;

```
    void set_dac(int dac, unsigned char green, unsigned char blue, unsigned char red)
    {
        union REGS regs;

        regs.h.ah = 16;
        regs.h.al = 16;
        regs.x.bx = dac;
        regs.h.ch = green;
        regs.h.cl = blue;
        regs.h.dh = red;
        int86(0x10,&regs,&regs);
    }
```

Displaying Text

The BIOS ROM provides three functions for displaying a single character. The first function to consider is the one used extensively by DOS for displaying messages, this is function 14 called "write text in teletype mode". This function interprets some control characters; bell (ascii 7), backspace (ascii 8), carriage return (ascii 10) and line feed (ascii 13) but all other ascii codes are displayed, and the current cursor position updated accordingly, moving down a row when a character is displayed in the far right column. To call this function the assembly language register AL holds the ascii code of the character to be displayed and assembly language register BL holds the foreground colour for the character to be displayed in if a graphics mode is active;

```
        mov     ah,14
        mov     al,character
        mov     bh,0
        mov     bl,foreground
        int     10h
```

A C function for accessing the write text in teletype mode may be written like this;

```
    #include <dos.h>

    void teletype(unsigned char character, unsigned char foreground)
    {
        union REGS inreg, outreg;

        inreg.h.ah = 14;
        inreg.h.al = character;
```

```
                inreg.h.bh = 0;
                inreg.h.bl = foreground;
                int86(0x10,&inrg,&outreg);
        }
```

The second BIOS ROM display function for displaying a character allows the foreground and background colours of the displayed character to be defined. It also allows multiple copies of the character to be displayed one after another automatically displaying subsequent characters at the next display position, although the current cursor position is not changed by this function.

This function is called "write character and attribute", and is BIOS ROM display function number 9. It is called with the ascii code of the character to be displayed in the assembly language AL register, the display page in assembly language register BH, the foreground colour in the first four bits of the assembly language register BL and the background colour in the last four bits of the assembly language register BL, the number of times the character is to be displayed is stored in the assembly language CX register thus;

```
                mov     ah,9
                mov     al,character
                mov     bh,0
                mov     bl,foreground + 16 * background
                mov     cx,number
                int     10h
```

And in C;

```
        #include <dos.h>

        void char_attrib(unsigned char character, unsigned char foreground, unsigned char background, int number)
        {
                union REGS inreg,outreg;

                inreg.h.ah = 9;
                inreg.h.al = character;
                inreg.h.bh = 0;
                inreg.h.bl = (background << 4) + foreground;
                inreg.x.cx = number;
                int86(0x10,&inreg,&outreg);
        }
```

The last BIOS ROM display function for displaying a character retains the foreground and background colours of the display position.

This function is called "write character", and is BIOS ROM display function number 10. It is identical to BIOS ROM display function 9 except that the colours of the displayed character are those which are prevalent at the display position, except in graphics modes when the foreground colour of the character is determined by the value in the assembly language BL register. Its use is as follows;

```
                mov     ah,10
                mov     al,character
                mov     bh,0
                mov     bl,foreground   ; For graphics modes ONLY
                mov     cx,number
                int     10h
```

And in C;

```
        #include <dos.h>
```

```
void char_attrib(unsigned char character, unsigned char foreground, int number)
{
        union REGS inreg,outreg;

        inreg.h.ah = 10;
        inreg.h.al = character;
        inreg.h.bh = 0;
        inreg.h.bl = foreground;  /* For graphics modes ONLY */
        inreg.x.cx = number;
        int86(0x10,&inreg,&outreg);
}
```

Positioning of the text cursor is provided for by the ROM BIOS display function number 2. It is called with the row number in the assembly language register DH and the column number in the assembly language register DL;

```
        mov     ah,2
        mov     bh,0
        mov     dh,row
        mov     dl,column
        int     10h
```

The corresponding function in C looks like this;

```
#include <dos.h>

void at(unsigned char row, unsigned char column)
{
        union REGS regs;

        regs.h.ah = 2;
        regs.h.bh = 0;
        regs.h.dh = row;
        regs.h.dl = column;
        int86(0x10,&regs,&regs);
}
```

From these basic functions a more useful replacement for the C language's "printf()" function can be written which allows data to be displayed at the current cursor position, previously set by a call to "at()", with prescribed attributes;

```
#include <dos.h>
#include <stdarg.h>

void at(unsigned char row, unsigned char column)
{
        union REGS regs;

        regs.h.ah = 2;
        regs.h.bh = 0;
        regs.h.dh = row;
        regs.h.dl = column;
        int86(0x10,&regs,&regs);
}

void xprintf(unsigned char foreground, unsigned char background, char *format,...)
{
        union REGS inreg,outreg;

        va_list arg_ptr;
        static char output[1000];
        unsigned char col;
        unsigned char row;
        unsigned char n;
        unsigned char p;
        unsigned char text;
        unsigned char attr;

        /* Convert foreground and background colours into a single attribute */
        attr = (background << 4) + foreground;

        /* Copy data into a single string */
        va_start(arg_ptr, format);
        vsprintf(output, format, arg_ptr);

        /* Determine number of display columns */
        inreg.h.ah = 15;
        int86(0x10,&inreg,&outreg);
        n = outreg.h.ah;

        /* Determine current cursor position */
        inreg.h.ah = 3;
        inreg.h.bh = 0;
        int86(0x10,&inreg,&outreg);
        row = outreg.h.dh;
        col = outreg.h.dl;

        /* Now display data */
        p = 0;
        while (output[p])
        {
                /* Display this character */
                inreg.h.bh = 0;
                inreg.h.bl = attr;
                inreg.x.cx = 01;
                inreg.h.ah = 9;
                inreg.h.al = output[p++];
                int86(0x10,&inreg,&outreg);

                /* Update cursor position */
                /* moving down a row if required */
                col++;
                if (col < (n - 1))
                        at(row, col);
                else
                {
                        col = 0;
                        at(++row, col);
                }
        }
}
```

This function, "xprintf()" illustrates two more functions of the BIOS ROM. The first is the call to function 15 which returns the number of text display columns for the currently active display mode.

The other function illustrated, but not yet discussed, is BIOS ROM function 3 which returns information about the cursor. The cursor’s row is returned in the assembly language register DH, and it’s column in the assembly language register DL.

ADVANCED GRAPHICS ON THE IBM PC

This section aims to reveal more about the graphics facilities offered by the IBM PC, in particular topics which are of a more complex nature than those addressed in the previous section.

Display Pages

The information for display by the display card is stored in an area of memory called the "video RAM". The size of the video RAM varies from one display card to another, and the amount of video RAM required for a display varies with the selected display mode. Display modes which do not require all of the video RAM use the remaining video RAM for additional display pages.

MODE	PAGES
0	8
1	8
2	4 (CGA) 8 (EGA, VGA)
3	4 (CGA) 8 (EGA, VGA)
4	1
5	1
6	1
7	8 (EGA, VGA)
13	8 (EGA, VGA)
14	4 (EGA, VGA)
15	2 (EGA, VGA)
16	2 (EGA, VGA)
17	1 (VGA)
18	1 (VGA)
19	1 (VGA)

Many of the BIOS ROM display functions allow selection of the display page to be written to, regardless of which page is currently being displayed.

The display card continuously updates the VDU from the information in the active display page. Changing the active display page instantaneously changes the display.

This provides the graphics programmer with the means to build a display on an undisplayed page, and to then change the active display page so that the viewer does not see the display being drawn.

Selection of the active display page is achieved by calling BIOS ROM display function 5 with the number of the required display page stored in the assembly language register AL;

```
                mov     ah,5
                mov     al,page
                int     10h
```

Or, from C this function becomes;

```
        #include <dos.h>

        void set_page(unsigned char page)
        {
                union REGS inreg, outreg;

                inreg.h.ah = 5;
                inreg.h.al = page;
                int86(0x10,&inreg,&outreg);
        }
```

The display page to which BIOS ROM display functions write is decided by the value stored in the assembly language BH register. The functions for setting a pixel’s colour may be amended thus;

```
                mov     ah, 12
                mov     al, colour
                mov     bh, page
                mov     cx, x_coord
                mov     dx, y_coord
                int     10h
```

And the corresponding C function becomes;

```
        #include <dos.h>

        void plot(int x_coord, int y_coord, unsigned char colour, unsigned char page)
        {
                /* Sets the colour of a pixel */

                union REGS inreg, outreg;

                inreg.h.ah = 12;
                inreg.h.al = colour;
                inreg.h.bh = page;
                inreg.x.cx = x_coord;
                inreg.x.dx = y_coord;
                int86(0x10,&inreg,&outreg);
        }
```

The currently active display page can be determined by calling BIOS ROM display function 15. This function returns the active display page in the assembly language register BH;

```
                mov     ah,15
                int     10h
                                                ; BH now holds active page number
```

Advanced Text Routines

When the IBM PC display is in a text mode a blinking cursor is displayed at the current cursor position. This cursor is formed of a rectangle which is one complete character width, but it’s top and bottom pixel lines are definable within the limits of the character height by calling BIOS ROM display function 1. A CGA display has a character height of 8 pixel lines, an EGA display has a character height of 14 lines and a VGA display has a character height of 16 lines.

BIOS ROM function 1 is called with the top pixel line number of the desired cursor shape in assembly language register CH and the bottom pixel line number in assembly language register CL;

```
        mov     ah,1
        mov     ch,top
        mov     cl,bottom
        int     10h
```

A C function to set the cursor shape may be be written thus;

```
    #include <dos.h>

    void setcursor(unsigned char top, unsigned char bottom)
    {
            union REGS inreg, outreg;

            inreg.h.ch = start;
            inreg.h.cl = end;
            inreg.h.ah = 1;
            int86(0x10, &inreg, &outreg);
    }
```

If the top pixel line is defined as larger than the bottom line, then the cursor will appear as a pair of parallel rectangles.

The cursor may be removed from view by calling BIOS ROM display function 1 with a value of 32 in the assembly language CH register.

The current shape of the cursor can be determined by calling BIOS ROM function 3, which returns the top pixel line number of the cursor shape in the assembly language CH register, and the bottom line number in the assembly language register CL.

Two functions are provided by the BIOS ROM for scrolling of the currently active display page. These are function 6, which scrolls the display up and function 7 which scrolls the display down.

Both functions accept the same parameters, these being the number of lines to scroll in the assembly language register AL, the colour attribute for the resulting blank line in the assembly language BH register, the top row of the area to be scrolled in the assembly language CH register, the left column of the area to be scrolled in the assembly language CL register, the bottom row to be scrolled in the assembly language DH register and the right most column to be scrolled in the assembly language DL register.

If the number of lines being scrolled is greater than the number of lines in the specified area, then the result is to clear the specified area, filling it with spaces in the attribute specified in the assembly language BH register.

Scrolling

A C function to scroll the entire screen down one line can be written thus;

```
    #include <dos.h>
```

```
void scroll_down(unsigned char attr)
{
        union REGS inreg, outreg;

        inreg.h.al = 1;
        inreg.h.cl = 0;
        inreg.h.ch = 0;
        inreg.h.dl = 79;
        inreg.h.dh = 24; /* Assuming a 25 line display */
        inreg.h.bh = attr;
        inreg.h.ah = 7;
        int86(0x10, &inreg, &outreg);
}
```

Clear Screen

A simple clear screen function can be written in C based upon the "scroll_down()" function simply by changing the value assigned to inreg.h.al to 0;

```
#include <dos.h>

void cls(unsigned char attr)
{
        union REGS inreg, outreg;

        inreg.h.al = 0;
        inreg.h.cl = 0;
        inreg.h.ch = 0;
        inreg.h.dl = 79;
        inreg.h.dh = 24; /* Assuming a 25 line display */
        inreg.h.bh = attr;
        inreg.h.ah = 7;
        int86(0x10, &inreg, &outreg);
}
```

Windowing

Windowing functions need to preserve the display they overwrite, and restore it when the window is removed from display. The BIOS ROM provides a display function which enables this to be done.

Function 8 requires the appropriate display page number to be stored in assembly language register BH, and then when called it returns the ascii code of the character at the current cursor position of that display page in the assembly language AL register, and the display attribute of the character in the assembly language AH register.

The following C functions allow an area of the display to be preserved, and later restored;

```
#include <dos.h>

void at(unsigned char row, unsigned char column, unsigned char page)
{
        /* Position the cursor */

        union REGS inreg,outreg;

        inreg.h.ah = 2;
        inreg.h.bh = page;
        inreg.h.dh = row;
```

```
                inreg.h.dl = column;
                int86(0x10,&inreg,&outreg);
        }

        void get_win(unsigned char left, unsigned char top, unsigned char right,unsigned char bottom, unsigned char
        page,           char *buffer)
        {
                /* Read a text window into a variable */

                union REGS inreg,outreg;

                unsigned char old_left;
                unsigned char old_row;
                unsigned char old_col;

                /* save current cursor position */
                inreg.h.ah = 3;
                inreg.h.bh = page;
                int86(0x10,&inreg,&outreg);
                old_row = outreg.h.dh;
                old_col = outreg.h.dl;

                while(top <= bottom)
                {
                        old_left = left;
                        while(left <= right)
                        {
                                at(top,left,page);
                                inreg.h.bh = page;
                                inreg.h.ah = 8;
                                int86(0x10,&inreg,&outreg);
                                *buffer++ = outreg.h.al;
                                *buffer++ = outreg.h.ah;
                                left++;
                        }

                        left = old_left;
                        top++;
                }

                /* Restore cursor to original location */
                at(old_row,old_col,page);
        }

        void put_win(unsigned char left, unsigned char top, unsigned char right,  unsigned char bottom, unsigned char
                                page, char *buffer)
        {
                /* Display a text window from a variable */

                union REGS inreg,outreg;

                unsigned char old_left;
                unsigned char chr;
                unsigned char attr;
                unsigned char old_row;
                unsigned char old_col;

                /* save current cursor position */
```

```
		inreg.h.ah = 3;
		inreg.h.bh = page;
		int86(0x10,&inreg,&outreg);
		old_row = outreg.h.dh;
		old_col = outreg.h.dl;

		while(top <= bottom)
		{
			old_left = left;
			while(left <= right)
			{
				at(top,left,page);
				chr = *buffer++;
				attr = *buffer++;
				inreg.h.bh = page;
				inreg.h.ah = 9;
				inreg.h.al = chr;
				inreg.h.bl = attr;
				inreg.x.cx = 1;
				int86(0x10,&inreg,&outreg);
				left++;
			}
			left = old_left;
			top++;
		}

		/* Restore cursor to original location */
		at(old_row,old_col,page);
	}
```

DIRECT VIDEO ACCESS WITH THE IBM PC

Accessing video RAM directly is much faster than using the BIOS ROM display functions. There are problems however. Different video modes arrange their use of video RAM in different ways so a number of functions are required for plotting using direct video access, where as only one function is required if use is made of the BIOS ROM display function.

The following C function will set a pixel in CGA display modes 4 and 5 directly;

```
void dplot4(int y, int x, int colour)
{
        /* Direct plotting in modes 4 & 5 ONLY! */

        union mask
        {
                char c[2];
                int i;
        }bit_mask;

        int index;
        int bit_position;

        unsigned char t;
        char xor;

        char far *ptr = (char far *) 0xB8000000;

        bit_mask.i = 0xFF3F;

        if ( y < 0 || y > 319 || x < 0 || x > 199)
                return;

        xor = colour & 128;

        colour = colour & 127;

        bit_position = y % 4;

        colour <<= 2 * (3 - bit_position);

        bit_mask.i >>= 2 * bit_position;

        index = x * 40 + (y / 4);

        if (x % 2)
                index += 8152;



        if (!xor)
        {
                t = *(ptr + index) & bit_mask.c[0];
                *(ptr + index) = t | colour;
        }
        else
```

```
		{
			t = *(ptr + index) | (char)0;
			*(ptr + index) = t ^ colour;
		}
	}
```

Direct plotting in VGA mode 19 is very much simpler;

```
	void dplot19(int x, int y, unsigned char colour)
	{
		/* Direct plot in mode 19 ONLY */
		char far *video;

		video = MK_FP(0xA000,0);
		video[x + y * 320] = colour;
}
```

ADVANCED GRAPHICS TECHNIQUES WITH THE IBM PC

Increasing Colours

The EGA display is limited displaying a maximum of 16 colours, however in high resolution graphics modes (such as mode 16) the small physical size of the pixels allows blending of adjacent colours to produce additional shades.

If a line is drawn straight across a black background display in blue, and then a subsequent line is drawn beneath it also in blue but only plotting alternate pixels, the second line will appear in a darker shade of the same colour.

The following C program illustrates this idea;

```
#include <dos.h>

union REGS inreg, outreg;

void setvideo(unsigned char mode)
{
        /* Sets the video display mode */

        inreg.h.al = mode;
        inreg.h.ah = 0;
        int86(0x10, &inreg, &outreg);
}

void plot(int x, int y, unsigned char colour)
{
        /* Sets a pixel at the specified coordinates */

        inreg.h.al = colour;
        inreg.h.bh = 0;
        inreg.x.cx = x;
        inreg.x.dx = y;
        inreg.h.ah = 0x0C;
        int86(0x10, &inreg, &outreg);
}

void line(int a, int b, int c, int d, unsigned char colour)
{
        /* Draws a straight line from (a,b) to (c,d) */

        int u;
        int v;
        int d1x;
        int d1y;
        int d2x;
        int d2y;
        int m;
        int n;
        int s;
        int i;
        u = c - a;
        v = d - b;
        if (u == 0)
        {
                d1x = 0;
                m = 0;
        }
        else
        {
                m = abs(u);
                if (u < 0)
                        d1x = -1;
                else
                if (u > 0)
                        d1x = 1;
        }
        if ( v == 0)
        {
                d1y = 0;
                n = 0;
        }
        else
        {
                n = abs(v);
                if (v < 0)
                        d1y = -1;
                else
                if (v > 0)
                        d1y = 1;
        }
        if (m > n)
        {
                d2x = d1x;
                d2y = 0;
        }
        else
        {
                d2x = 0;
                d2y = d1y;
                m = n;
                n = abs(u);
        }
        s = m / 2;

        for (i = 0; i <= m; i++)
        {
                plot(a,b,colour);
                s += n;
                if (s >= m)
                {
                        s -= m;
                        a += d1x;
                        b += d1y;
                }
                else
                {
                        a += d2x;
                        b += d2y;
                }
        }
}

void dot_line(int a, int b, int c, int d, int colour)
{
        /* Draws a dotted straight line from (a,b) to (c,d) */

        int u;
        int v;
        int d1x;
        int d1y;
        int d2x;
        int d2y;
        int m;
        int n;
        int s;
        int i;

        u = c - a;
        v = d - b;
        if (u == 0)
        {
                d1x = 0;
                m = 0;
        }
        else
        {
                m = abs(u);
                if (u < 0)
                        d1x = -2;
                else
                if (u > 0)
                        d1x = 2;
        }
        if (v == 0)
        {
                d1y = 0;
                n = 0;
        }
        else
        {
                n = abs(v);
                if (v < 0)
                        d1y = -2;
                else
                if (v > 0)
                        d1y = 2;
        }
        if (m > n)
        {
                d2x = d1x;
                d2y = 0;
        }
        else
        {
                d2x = 0;
                d2y = d1y;
                m = n;
                n = abs(u);
        }
        s = m / 2;

        for (i = 0; i <= m; i+=2)
        {
                plot(a,b,colour);
                s += n;
                if (s >= m)
                {
                        s -= m;
                        a += d1x;
                        b += d1y;
                }
                else
                {
                        a += d2x;
                        b += d2y;
                }
        }
}


void main(void)
{
        int n;

        /* Display different colour bands */

        setvideo(16);

        for(n = 1; n < 16; n++)
        {
                line(0,n * 20,639,n * 20,n);
                line(0,1 + n * 20,639,1 + n * 20,n);
                line(0,2 + n * 20,639,2 + n * 20,n);

                dot_line(0,4 + n * 20,639,4 + n * 20,n);
                dot_line(1,5 + n * 20,639,5 + n * 20,n);
                dot_line(0,6 + n * 20,639,6 + n * 20,n);

                dot_line(0,8 + n * 20,639,8 + n * 20,n);
                dot_line(1,9 + n * 20,639,9 + n * 20,n);
                dot_line(0,10 + n * 20,639,10 + n * 20,n);

                dot_line(1,8 + n * 20,639,8 + n * 20,7);
                dot_line(0,9 + n * 20,639,9 + n * 20,7);
                dot_line(1,10 + n * 20,639,10 + n * 20,7);

                dot_line(1,12 + n * 20,639,12 + n * 20,n);
                dot_line(0,13 + n * 20,639,13 + n * 20,n);
                dot_line(1,14 + n * 20,639,14 + n * 20,n);

                dot_line(0,12 + n * 20,639,12 + n * 20,14);
                dot_line(1,13 + n * 20,639,13 + n * 20,14);
                dot_line(0,14 + n * 20,639,14 + n * 20,14);
        }
```

```
	}
```

This technique can be put to good use for drawing three dimensional boxes;

```
	void box3d(int xa,int ya, int xb, int yb, int col)
	{
		/* Draws a box for use in 3d histogram graphs etc */

		int xc;
		int xd;

		int n;

		xd = (xb - xa) / 2;
		xc = xa + xd;

		/* First draw the solid face */
		for(n = xa; n < xb; n++)
			line(n,ya,n,yb,col);

		/* Now "shaded" top and side */
		for(n = 0; n < xd; n++)
		{
			dotline(xa + n,yb - n ,xc + n,yb - n,col);
			dotline(xa + xd + n,yb - n ,xc + xd + n,yb - n,col);
			dotline(xb +n ,ya - n ,xb + n,yb - n,col);
		}
	}
```

Displaying Text At Pixel Coordinates

When using graphics display modes it is useful to be able to display text not at the fixed character boundaries, but at pixel coordinates. This can be achieved by implementing a print function which reads the character definition data from the BIOS ROM and uses it to plot pixels to create the shapes of the desired text. The following C function, “gr_print()” illustrates this idea using the ROM CGA (8x8) character set;

```
	void gr_print(char *output, int x, int y, unsigned char colour)
	{
		unsigned char far *ptr;
		unsigned char chr;
		unsigned char bmask;
		int i;
		int k;
		int oldy;
		int p;
		int height;

		/* The height of the characters in the font being accessed */
		height = 8;

		/* Set pointer to start of font definition in the ROM */
		ptr = getfontptr(3);

		oldy = y;
		p = 0;
```

```
                /* Loop output string */
                while(output[p])
                {
                        /* Get first character to be displayed */
                        chr = output[p];

                        /* Loop pixel lines in character definition */
                        for(i = 0; i < height; i++)
                        {
                                /* Get pixel line definition from the ROM */
                                bmask = *(ptr + (chr * height) + i);

                                /* Loop pixel columns */
                                for (k = 0; k < 8; ++k, bmask <<= 1)
                                {
                                        /* Test for a set bit */
                                        if(bmask & 128)
                                                /* Plot a pixel if appropriate */
                                                plot(x,y,colour);
                                        else
                                                plot(x,y,0);
                                        x++;
                                }
                                /* Down to next row and left to start of character */
                                y++;
                                x -= 8;
                        }
                        /* Next character to be displayed */
                        p++;

                        /* Back to top row of the display position */
                        y = oldy;

                        /* Right to next character display position */
                        x += 8;
                }
        }
```

The following assembly language support function is required to retrieve the address of the ROM font by calling the BIOS ROM display function which will return the address;

```
        ; GET FONT POINTER
        ; Small memory model
        ; compile with tasm /mx

        _TEXT  segment          byte public 'CODE'
                                 assume          cs:_TEXT,ds:NOTHING

        _getfontptr     proc    near
                        push    bp
                        mov      bp,sp
                        mov      ax,1130h
                        mov      bh, [bp+4]              ; Number for font to be retrieved
                        int      10h
                        mov      dx,es
                        mov      ax,bp
```

```
                pop     bp
                ret
_getfontptr     endp

_TEXT   ends

        public  _getfontptr
        end
```

The font number supplied to "getfontptr()" can be one of;

2	ROM EGA 8 x 14 font
3	ROM CGA 8 x 8 font
6	ROM VGA 8 x 16 font

A Graphics Function Library For Turbo C

```
/* Graphics library for 'Turbo C'  (V2.01) */

/* (C)1992 Copyright Servile Software */

#include <stdio.h>
#include <dos.h>
#include <stdlib.h>
#include <stdarg.h>
#include <string.h>
#include <math.h>

/* Global variables */
static unsigned char attribute;
int _dmode;
int _lastx;
int _lasty;

/* Maximum coordinates for graphics screen */
static int maxx;
static int maxy;

/* Sprite structure */
struct SP
{
        int x;
        int y;
        char data[256];
        char save[256];
};
typedef struct SP SPRITE;


/* Sine and cosine tables for trig' operations */

static double sintable[360] = {0.000000000000000001,0.017452406437283512,
                              0.034899496702500969,0.052335956242943835,
                              0.069756473744125302,0.087155742747658166,
                              0.104528463267653457,0.121869343405147462,
                              0.139173100960065438,0.156434465040230869,
                              0.173648177666930331,0.190808995376544804,
                              0.207911690817759315,0.224951054343864976,
                              0.241921895599667702,0.258819045102520739,
                              0.275637355816999163,0.292371704722736769,
                              0.309016994374947451,0.325568154457156755,
                              0.342020143325668824,0.358367949545300379,
                              0.374606593415912181,0.390731128489273882,
                              0.406736643075800375,0.422618261740699608,
                              0.438371146789077626,0.453990499739547027,
                              0.469471562785891028,0.484809620246337225,
                              0.500000000000000222,0.515038074910054489,
                              0.529919264233205234,0.544639035015027417,
                              0.559192903470747127,0.573576436351046381,
                              0.587785252292473470,0.601815023152048600,
                              0.615661475325658625,0.629320391049837835,
0.642787609686539696,0.656059028990507720,
0.669130606358858682,0.681998360062498921,
0.694658370458997698,0.707106781186548017,
0.719339800338651636,0.731353701619170904,
0.743144825477394688,0.754709580222772458,
0.766044443118978569,0.777145961456971346,
0.788010753606722458,0.798635510047293384,
0.809016994374947895,0.819152044288992243,
0.829037572555042179,0.838670567945424494,
0.848048096156426512,0.857167300702112778,
0.866025403784439152,0.874619707139396296,
0.882947592858927432,0.891006524188368343,
0.898794046299167482,0.906307787036650381,
0.913545457642601311,0.920504853452440819,
0.927183854566787868,0.933580426497202187,
0.939692620785908761,0.945518575599317179,
0.951056516295153975,0.956304755963035880,
0.961261695938319227,0.965925826289068645,
0.970295726275996806,0.974370064785235579,
0.978147600733805911,0.981627183447664198,
0.984807753012208353,0.987688340595137992,
0.990268068741570473,0.992546151641322205,
0.994521895368273512,0.996194698091745656,
0.997564050259824309,0.998629534754573944,
0.999390827019095762,0.999847695156391270,
1.000000000000000000,0.999847695156391159,
0.999390827019095651,0.998629534754573833,
0.997564050259824087,0.996194698091745323,
0.994521895368273179,0.992546151641321761,
0.990268068741570029,0.987688340595137437,
0.984807753012207687,0.981627183447663532,
0.978147600733805245,0.974370064785234802,
0.970295726275996029,0.965925826289067757,
0.961261695938318228,0.956304755963034880,
0.951056516295152865,0.945518575599316069,
0.939692620785907651,0.933580426497200966,
0.927183854566786536,0.920504853452439486,
0.913545457642599978,0.906307787036649048,
0.898794046299166149,0.891006524188367122,
0.882947592858926211,0.874619707139395186,
0.866025403784438042,0.857167300702111778,
0.848048096156425624,0.838670567945423717,
0.829037572555041513,0.819152044288991688,
0.809016994374947451,0.798635510047293051,
0.788010753606722236,0.777145961456971346,
0.766044443118978569,0.754709580222772680,
0.743144825477395132,0.731353701619171459,
0.719339800338652302,0.707106781186548794,
0.694658370458998808,0.681998360062500142,
0.669130606358860014,0.656059028990509274,
0.642787609686541472,0.629320391049839833,
0.615661475325660845,0.601815023152051043,
0.587785252292476135,0.573576436351049268,
0.559192903470750236,0.544639035015030637,
0.529919264233208676,0.515038074910058152,
0.500000000000004219,0.484809620246341444,
0.469471562785895413,0.453990499739551634,
0.438371146789082455,0.422618261740704715,
0.406736643075805704,0.390731128489279489,
0.374606593415918010,0.358367949545306430,
0.342020143325675152,0.325568154457163306,
0.309016994374954279,0.292371704722743819,
0.275637355817006491,0.258819045102528289,
0.241921895599675502,0.224951054343872997,
0.207911690817767558,0.190808995376553270,
0.173648177666939019,0.156434465040239751,
0.139173100960074542,0.121869343405156802,
0.104528463267663005,0.087155742747667922,
0.069756473744135267,0.052335956242954007,
0.034899496702511350,0.017452406437294093,
0.000000000000010781,-0.017452406437272534,
-0.034899496702489805,-0.052335956242932476,
-0.069756473744113756,-0.087155742747646453,
-0.104528463267641564,-0.121869343405135402,
-0.139173100960053198,-0.156434465040218462,
-0.173648177666917786,-0.190808995376532092,
-0.207911690817746464,-0.224951054343851986,
-0.241921895599654574,-0.258819045102507472,
-0.275637355816985785,-0.292371704722723225,
-0.309016994374933796,-0.325568154457142933,
-0.342020143325654891,-0.358367949545286335,
-0.374606593415898026,-0.390731128489259616,
-0.406736643075785997,-0.422618261740685175,
-0.438371146789063082,-0.453990499739532427,
-0.469471562785876373,-0.484809620246322570,
-0.499999999999985512,-0.515038074910039723,
-0.529919264233190468,-0.544639035015012540,
-0.559192903470732361,-0.573576436351031616,
-0.587785252292458593,-0.601815023152033834,
-0.615661475325643859,-0.629320391049823069,
-0.642787609686525041,-0.656059028990493065,
-0.669130606358844027,-0.681998360062484377,
-0.694658370458983265,-0.707106781186533584,
-0.719339800338637314,-0.731353701619156804,
-0.743144825477380699,-0.754709580222758580,
-0.766044443118964691,-0.777145961456957690,
-0.788010753606709025,-0.798635510047280062,
-0.809016994374934795,-0.819152044288979364,
-0.829037572555029412,-0.838670567945412060,
-0.848048096156414188,-0.857167300702100676,
-0.866025403784427272,-0.874619707139384750,
-0.882947592858916108,-0.891006524188357352,
-0.898794046299156713,-0.906307787036639945,
-0.913545457642591208,-0.920504853452430938,
-0.927183854566778320,-0.933580426497192972,
-0.939692620785899990,-0.945518575599308742,
-0.951056516295145871,-0.956304755963028108,
-0.961261695938311900,-0.965925826289061651,
-0.970295726275990256,-0.974370064785229362,
-0.978147600733800138,-0.981627183447658869,
-0.984807753012203468,-0.987688340595133552,
-0.990268068741566587,-0.992546151641318764,
-0.994521895368270514,-0.996194698091743103,
-0.997564050259822310,-0.998629534754572390,
-0.999390827019094763,-0.999847695156390714,
-1.000000000000000000,-0.999847695156391714,
                                -0.999390827019096761,-0.998629534754575388,
                                -0.997564050259826307,-0.996194698091748099,
                                -0.994521895368276398,-0.992546151641325647,
                                -0.990268068741574470,-0.987688340595142433,
                                -0.984807753012213349,-0.981627183447669860,
                                -0.978147600733812128,-0.974370064785242240,
                                -0.970295726276004022,-0.965925826289076417,
                                -0.961261695938327665,-0.956304755963044872,
                                -0.951056516295163523,-0.945518575599327393,
                                -0.939692620785919530,-0.933580426497213511,
                                -0.927183854566799748,-0.920504853452453253,
                                -0.913545457642614411,-0.906307787036664148,
                                -0.898794046299181804,-0.891006524188383331,
                                -0.882947592858942976,-0.874619707139412506,
                                -0.866025403784455916,-0.857167300702130208,
                                -0.848048096156444498,-0.838670567945443146,
                                -0.829037572555061497,-0.819152044289012227,
                                -0.809016994374968434,-0.798635510047314479,
                                -0.788010753606744219,-0.777145961456993772,
                                -0.766044443119001550,-0.754709580222796106,
                                -0.743144825477418891,-0.731353701619195773,
                                -0.719339800338677060,-0.707106781186574107,
                                -0.694658370459024455,-0.681998360062526232,
                                -0.669130606358886548,-0.656059028990536253,
                                -0.642787609686568784,-0.629320391049867478,
                                -0.615661475325688934,-0.601815023152079465,
                                -0.587785252292504890,-0.573576436351078467,
                                -0.559192903470779767,-0.544639035015060502,
                                -0.529919264233238985,-0.515038074910088794,
                                -0.500000000000035083,-0.484809620246372641,
                                -0.469471562785926888,-0.453990499739583386,
                                -0.438371146789114541,-0.422618261740737022,
                                -0.406736643075838289,-0.390731128489312296,
                                -0.374606593415951039,-0.358367949545339737,
                                -0.342020143325708625,-0.325568154457197001,
                                -0.309016994374988196,-0.292371704722777903,
                                -0.275637355817040797,-0.258819045102562761,
                                -0.241921895599710085,-0.224951054343907719,
                                -0.207911690817802419,-0.190808995376588242,
                                -0.173648177666974129,-0.156434465040274973,
                                -0.139173100960109847,-0.121869343405192190,
                                -0.104528463267698463,-0.087155742747703435,
                                -0.069756473744170822,-0.052335956242989604,
                                -0.034899496702546981,-0.017452406437329739
                };

static double costable[360] = { 1.000000000000000000,0.999847695156391270,
                                0.999390827019095762,0.998629534754573833,
                                0.997564050259824198,0.996194698091745545,
                                0.994521895368273290,0.992546151641321983,
                                0.990268068741570362,0.987688340595137770,
                                0.984807753012208020,0.981627183447663976,
                                0.978147600733805689,0.974370064785235246,
                                0.970295726275996473,0.965925826289068312,
                                0.961261695938318894,0.956304755963035436,
                                0.951056516295153531,0.945518575599316735,
                                0.939692620785908317,0.933580426497201743,
                                0.927183854566787313,0.920504853452440264,
0.913545457642600867,0.906307787036649826,
0.898794046299166927,0.891006524188367788,
0.882947592858926766,0.874619707139395630,
0.866025403784438486,0.857167300702112112,
0.848048096156425846,0.838670567945423828,
0.829037572555041513,0.819152044288991576,
0.809016994374947229,0.798635510047292607,
0.788010753606721681,0.777145961456970569,
0.766044443118977680,0.754709580222771681,
0.743144825477393911,0.731353701619170127,
0.719339800338650748,0.707106781186547129,
0.694658370458996810,0.681998360062498032,
0.669130606358857682,0.656059028990506721,
0.642787609686538697,0.629320391049836836,
0.615661475325657626,0.601815023152047601,
0.587785252292472471,0.573576436351045382,
0.559192903470746128,0.544639035015026307,
0.529919264233204124,0.515038074910053378,
0.499999999999999112,0.484809620246336115,
0.469471562785889862,0.453990499739545861,
0.438371146789076460,0.422618261740698442,
0.406736643075799154,0.390731128489272661,
0.374606593415910960,0.358367949545299158,
0.342020143325667547,0.325568154457155479,
0.309016994374946230,0.292371704722735493,
0.275637355816997887,0.258819045102519463,
0.241921895599666398,0.224951054343863643,
0.207911690817757955,0.190808995376543389,
0.173648177666928888,0.156434465040229398,
0.139173100960063939,0.121869343405145950,
0.104528463267651903,0.087155742747656584,
0.069756473744123679,0.052335956242942190,
0.034899496702499304,0.017452406437281822,
-0.000000000000001715,-0.017452406437285253,
-0.034899496702502732,-0.052335956242945618,
-0.069756473744127107,-0.087155742747659998,
-0.104528463267655317,-0.121869343405149350,
-0.139173100960067325,-0.156434465040232784,
-0.173648177666932274,-0.190808995376546747,
-0.207911690817761285,-0.224951054343866974,
-0.241921895599669701,-0.258819045102522793,
-0.275637355817001217,-0.292371704722738768,
-0.309016994374949450,-0.325568154457158698,
-0.342020143325670822,-0.358367949545302322,
-0.374606593415914124,-0.390731128489275825,
-0.406736643075802318,-0.422618261740701329,
-0.438371146789079125,-0.453990499739548303,
-0.469471562785892083,-0.484809620246338169,
-0.500000000000000999,-0.515038074910054933,
-0.529919264233205567,-0.544639035015027528,
-0.559192903470747127,-0.573576436351046159,
-0.587785252292473026,-0.601815023152048045,
-0.615661475325657848,-0.629320391049836947,
-0.642787609686538697,-0.656059028990506499,
-0.669130606358857238,-0.681998360062497477,
-0.694658370458996033,-0.707106781186546240,
-0.719339800338649749,-0.731353701619168906,
-0.743144825477392579,-0.754709580222770238,
-0.766044443118976237,-0.777145961456968903,
-0.788010753606719905,-0.798635510047290720,
-0.809016994374945231,-0.819152044288989578,
-0.829037572555039404,-0.838670567945421719,
-0.848048096156423625,-0.857167300702109891,
-0.866025403784436265,-0.874619707139393410,
-0.882947592858924435,-0.891006524188365345,
-0.898794046299164484,-0.906307787036647494,
-0.913545457642598424,-0.920504853452437932,
-0.927183854566784982,-0.933580426497199412,
-0.939692620785906096,-0.945518575599314515,
-0.951056516295151311,-0.956304755963033326,
-0.961261695938316785,-0.965925826289066314,
-0.970295726275994586,-0.974370064785233358,
-0.978147600733803912,-0.981627183447662310,
-0.984807753012206577,-0.987688340595136327,
-0.990268068741569030,-0.992546151641320873,
-0.994521895368272291,-0.996194698091744657,
-0.997564050259823532,-0.998629534754573389,
-0.999390827019095318,-0.999847695156391048,
-1.000000000000000000,-0.999847695156391381,
-0.999390827019096095,-0.998629534754574499,
-0.997564050259825086,-0.996194698091746544,
-0.994521895368274622,-0.992546151641323537,
-0.990268068741572027,-0.987688340595139658,
-0.984807753012210241,-0.981627183447666418,
-0.978147600733808353,-0.974370064785238244,
-0.970295726275999804,-0.965925826289071865,
-0.961261695938322669,-0.956304755963039654,
-0.951056516295157972,-0.945518575599321509,
-0.939692620785913424,-0.933580426497207072,
-0.927183854566793086,-0.920504853452446370,
-0.913545457642607195,-0.906307787036656598,
-0.898794046299173921,-0.891006524188375226,
-0.882947592858934649,-0.874619707139403846,
-0.866025403784447034,-0.857167300702120993,
-0.848048096156435061,-0.838670567945433487,
-0.829037572555051505,-0.819152044289001902,
-0.809016994374957998,-0.798635510047303709,
-0.788010753606733227,-0.777145961456982559,
-0.766044443118990004,-0.754709580222784449,
-0.743144825477407012,-0.731353701619183672,
-0.719339800338664737,-0.707106781186561450,
-0.694658370459011576,-0.681998360062513242,
-0.669130606358873337,-0.656059028990522708,
-0.642787609686555128,-0.629320391049853711,
-0.615661475325674945,-0.601815023152065254,
-0.587785252292490457,-0.573576436351063812,
-0.559192903470765002,-0.544639035015045625,
-0.529919264233223886,-0.515038074910073473,
-0.500000000000019651,-0.484809620246357043,
-0.469471562785911123,-0.453990499739567510,
-0.438371146789098498,-0.422618261740720869,
-0.406736643075822024,-0.390731128489295920,
-0.374606593415934497,-0.358367949545323083,
-0.342020143325691917,-0.325568154457180181,
-0.309016994374971266,-0.292371704722760861,
-0.275637355817023644,-0.258819045102545497,
                              -0.241921895599692793,-0.224951054343890372,
                              -0.207911690817784989,-0.190808995376570756,
                              -0.173648177666956560,-0.156434465040257376,
                              -0.139173100960092194,-0.121869343405174496,
                              -0.104528463267680741,-0.087155742747685686,
                              -0.069756473744153044,-0.052335956242971805,
                              -0.034899496702529162,-0.017452406437311916,
                              -0.000000000000028605,0.017452406437254715,
                              0.034899496702471985,0.052335956242914670,
                              0.069756473744095979,0.087155742747628689,
                              0.104528463267623842,0.121869343405117708,
                              0.139173100960035545,0.156434465040200865,
                              0.173648177666900216,0.190808995376514606,
                              0.207911690817729033,0.224951054343834611,
                              0.241921895599637282,0.258819045102490264,
                              0.275637355816968632,0.292371704722706127,
                              0.309016994374916809,0.325568154457126058,
                              0.342020143325638126,0.358367949545269682,
                              0.374606593415881484,0.390731128489243240,
                              0.406736643075769733,0.422618261740669021,
                              0.438371146789047095,0.453990499739516551,
                              0.469471562785860608,0.484809620246306972,
                              0.499999999999970080,0.515038074910024402,
                              0.529919264233175369,0.544639035014997663,
                              0.559192903470717484,0.573576436351016961,
                              0.587785252292444271,0.601815023152019624,
                              0.615661475325629759,0.629320391049809191,
                              0.642787609686511385,0.656059028990479520,
                              0.669130606358830815,0.681998360062471387,
                              0.694658370458970387,0.707106781186521038,
                              0.719339800338624991,0.731353701619144592,
                              0.743144825477368709,0.754709580222746812,
                              0.766044443118953255,0.777145961456946477,
                              0.788010753606698033,0.798635510047269292,
                              0.809016994374924359,0.819152044288969150,
                              0.829037572555019531,0.838670567945402290,
                              0.848048096156404752,0.857167300702091572,
                              0.866025403784418391,0.874619707139376090,
                              0.882947592858907782,0.891006524188349247,
                              0.898794046299148941,0.906307787036632395,
                              0.913545457642583991,0.920504853452423943,
                              0.927183854566771659,0.933580426497186644,
                              0.939692620785893884,0.945518575599302968,
                              0.951056516295140320,0.956304755963022890,
                              0.961261695938306904,0.965925826289056988,
                              0.970295726275985926,0.974370064785225365,
                              0.978147600733796474,0.981627183447655538,
                              0.984807753012200360,0.987688340595130776,
                              0.990268068741564034,0.992546151641316543,
                              0.994521895368268627,0.996194698091741548,
                              0.997564050259821089,0.998629534754571502,
                              0.999390827019094097,0.999847695156390381

      int dtoi(double x)
      {
            /* Rounds a floating point number to an integer */

            int y;
            y = x;
            if (y < (x + 0.5))
                  y++;
            return(y);
      }

      int getmode(int *ncols)
      {
            /* Returns current video mode and number of columns in ncols */

            union REGS inreg,outreg;

            inreg.h.ah = 0x0F;
            int86(0x10, &inreg, &outreg);
            *ncols = outreg.h.ah;
            return(outreg.h.al);
      }

      void setvideo(unsigned char mode)
      {
            /* Sets the video display mode and clears the screen */
            /* (modes above 127 on VGA monitors do not clear the screen) */

            asm mov ax , mode;
            asm int 10h;

            /* Set global variables */
            switch(mode)
            {
                  case 0:
                  case 1:
                  case 2:
                  case 3: break;
                  case 4:
                  case 9:
                  case 13:
                  case 19:
                  case 5: maxx = 320;
                        maxy = 200;
                        break;
                  case 14:
                  case 10:
                  case 6: maxx = 640;
                        maxy = 200;
                        break;
                  case 7: maxx = 720;
                        maxy = 400;
                        break;
                  case 8: maxx = 160;
                        maxy = 200;
                        break;
                  case 15:
                  case 16: maxx = 640;
                         maxy = 350;
                         break;
                  case 17:
                  case 18: maxx = 640;
```

```
                        maxy = 480;
                        break;

            }
            _dmode = mode;
    }

    void getcursor(int *row, int *col)
    {
            /* Returns the cursor position and size for the currently active
               video display page */

            union REGS inreg,outreg;

            inreg.h.bh = 0;
            inreg.h.ah = 0x03;
            int86(0x10, &inreg, &outreg);
            *row = outreg.h.dh;
            *col = outreg.h.dl;
    }

    void setcursor(char status)
    {
            /* Set cursor size, if status is 0x20 the cursor is not displayed */

            asm mov ah,1
            asm mov ch,status
            asm mov cl,7
            asm int 10h
    }

    void at(int row, int col)
    {
            /* Position text cursor on current screen */

            asm mov bh , 0;
            asm mov dh , row;
            asm mov dl , col;
            asm mov ah , 02h;
            asm int 10h;
    }

    void clsc(unsigned char attrib)
    {
            /* Clear display and fill with new attribute */

            asm mov ax , 073dh;
            asm mov bh , attrib;
            asm mov cx , 0;
            asm mov dx , 3c4fh;
            asm int 10h;

            /* Now move text cursor to origin (0,0) */
            asm mov bh , 0;
            asm mov dx , 0;
            asm mov ah , 02h;
            asm int 10h;
    }
```

```
void clrwindow(int x1, int y1,int x2, int y2, unsigned char attrib)
{
	/* Clear a text window */

	union REGS inreg,outreg;

	inreg.h.al = (unsigned char)(y2 - y1 + 1);
	inreg.h.bh = attrib;
	inreg.h.cl = (unsigned char)x1;
	inreg.h.ch = (unsigned char)y1;
	inreg.h.dl = (unsigned char)x2;
	inreg.h.dh = (unsigned char)y2;
	inreg.h.ah = 0x06;
	int86(0x10, &inreg, &outreg);
}

void scrollsc(void)
{
	/* Scroll the text screen */

	asm mov al,01h;
	asm mov cx,0
	asm mov dx,3c4fh;
	asm mov bh,attribute;
	asm mov ah, 06h;
	asm int 10h;
}

void winscr(unsigned char left,unsigned char top, unsigned char right,unsigned char bottom,
		unsigned char direction, unsigned char numlines)
{
	/* Scroll a text window */

	asm mov al , numlines;
	asm mov cl , left;
	asm mov ch , top;
	asm mov dl , right;
	asm mov dh , bottom;
	asm mov bh , attribute;
	asm mov ah , direction;
	asm int 10h;
}

unsigned char attr(int foregrnd, int backgrnd)
{
	/* Convert a colour pair into a single attribute */

	return((char)((backgrnd << 4) + foregrnd));
}

void shadewin(unsigned char left, unsigned char top, unsigned char right,   unsigned char bottom)
{
	/* Shade a text window */

	int row;
	int col;
	int oldrow;
```

```
        int oldcol;

        /* Preserve existing coords */
        getcursor(&oldrow,&oldcol);

        col = right + 1;
        for (row = top + 1; row <= bottom + 1; row++)
        {
                /* Move to row,col */
                asm mov bh , 0;
                asm mov dh , row;
                asm mov dl , col;
                asm mov ah , 02h;
                asm int 10h;
                /* Get character */
                asm mov ah , 0Fh;
                asm int 10h;
                asm mov ah , 08h;
                asm int 10h;

                /* Write in attribute 7 */
                asm mov ah , 09h;
                asm mov bl, 07h;
                asm mov cx, 01h;
                asm int 10h;
        }
        bottom++;
        for (col = left + 1; col <= right + 1; col++)
        {
                /* Move to row,col */
                asm mov bh , 0;
                asm mov dh , bottom;
                asm mov dl , col;
                asm mov ah , 02h;
                asm int 10h;

                /* Get character */
                asm mov ah , 0Fh;
                asm int 10h;
                asm mov ah , 08h;
                asm int 10h;

                /* Write in attribute 7 */
                asm mov ah , 09h;
                asm mov bl, 07h;
                asm mov cx, 01h;
                asm int 10h;
        }

        /* Return to original position */
        /* Move to row,col */
        asm mov bh , 0;
        asm mov dh , oldrow;
        asm mov dl , oldcol;
        asm mov ah , 02h;
        asm int 10h;
}
```

```
void bprintf(char *format, ...)
{
	/* print text to graphics screen correctly */

	va_list arg_ptr;
	char output[1000];
	int c, r, n, p = 0;
	unsigned char text;
	unsigned char page;

	va_start(arg_ptr, format);
	vsprintf(output, format, arg_ptr);

	if (strcmp(output,"\r\n") == 0)
		fputs(output,stdout);

	else
	{
		asm mov ah , 0Fh;
		asm int 10h;
		asm mov page, bh;

		getmode(&n);
		getcursor(&r,&c);

		while (output[p])
		{
			text = output[p++];
			asm mov bh , page;
			asm mov bl , attribute;
			asm mov cx , 01h;
			asm mov ah , 09h;
			asm mov al , text;
			asm int 10h;

			c++;
			if (c < (n-1))
				at( r, c);
			else
			{
				c = 0;
				at(++r,0);
			}
		}
	}
}

void wrtstr(char *output)
{
	/* TTY text output. The original screen attributes remain unaffected */

	int p = 0;
	unsigned char page;
	unsigned char text;

	asm mov ah , 0Fh;
	asm int 10h;
	asm mov page, bh;
```

```
        while (output[p])
        {
                text = output[p++];
                asm mov bh , page;
                asm mov ah , 0Eh;
                asm mov al , text;
                asm int 10h;
        }
}

void getwin(int left, int top, int right, int bottom, char *buffer)
{
        /* Read a text window into a variable */

        int oldleft;
        unsigned char page;

        asm mov ah , 0Fh;
        asm int 10h;
        asm mov page, bh;

        while(top <= bottom)
        {
                oldleft = left;
                while(left <= right)
                {
                        at(top,left);
                        asm mov bh , page;
                        asm mov ah , 08h;
                        asm int 10h;
                        *buffer++ = _AL;
                        *buffer++ = _AH;
                        left++;
                }
                left = oldleft;
                top++;
        }
}

void putwin(int left, int top, int right, int bottom,char *buffer)
{
        /* Display a text window from a variable */

        int oldleft;
        unsigned char chr;
        unsigned char attr;
        unsigned char page;

        asm mov ah , 0Fh;
        asm int 10h;
        asm mov page, bh;

        while(top <= bottom)
        {
                oldleft = left;
                while(left <= right)
                {
                        at(top,left);
                        chr = *buffer++;
                        attr = *buffer++;
                        asm mov bh , page;
                        asm mov ah , 09h;
                        asm mov al, chr;
                        asm mov bl, attr;
                        asm mov cx,1;
                        asm int 10h;
                        left++;
                }
                left = oldleft;
                top++;
        }
}

void setpalette(unsigned char palno)
{
        /* Sets the video palette */

        asm mov bh,01h;
        asm mov bl,palno;
        asm mov ah,0Bh;
        asm int 10h;
}

void setborder(unsigned char x)
{
        /* Set border colour */

        asm mov bh, x;
        asm mov ax ,1001h;
        asm int 10h;
}

void setlines(unsigned char x)
{
        /* Set text display number of lines */

        asm mov ah,11h;
        asm mov al,x;
        asm mov bl,0;
        asm int 10h;
}

void graphbackground(unsigned char colour)
{
        /* Selects the background colour for a graphics mode */

        asm mov bh,0;
        asm mov bl,colour;
        asm mov ah, 0Bh;
        asm int 10h;
}

void plot(int x, int y, unsigned char colour)
{
        /* Sets a pixel at the specified coordinates to the specified colour.
```

```
	  The variables _lastx and _lasty are updated. */

	unsigned char far *video;

	_lastx = x;
	_lasty = y;

	if (_dmode == 19)
	{
		video = MK_FP(0xA000,0);
		video[x+y*320] = colour;
	}
	else
	{
		asm mov al , colour;
		asm mov bh , 00;
		asm mov cx , x;
		asm mov dx , y;
		asm mov ah , 0Ch;
		asm int 10h;
	}
}

int pixset(int x, int y)
{
	/* Returns the colour of the specified pixel */

	asm mov cx ,x;
	asm mov dx ,y;
	asm mov ah ,0Dh;
	asm int 10h;
	return(_AL);
}

void move(int x, int y)
{
	/* Sets the value of the variables _lastx and _lasty */

	_lastx = x;
	_lasty = y;
}

void line(int a, int b, int c, int d, int col)
{
	/* Draws a straight line from (a,b) to (c,d) in colour col */

	int u;
	int v;
	int d1x;
	int d1y;
	int d2x;
	int d2y;
	int m;
	int n;
	int s;
	int i;

	if (a < 0)
```

```
                a = 0;
        else
        if (a > maxx)
                a = maxx;

        if (c < 0)
                c = 0;
        else
        if (c > maxx)
                c = maxx;

        if (b < 0)
                b = 0;
        else
        if (b > maxy)
                b = maxy;

        if (d < 0)
                d = 0;
        else
        if (d > maxy)
                d = maxy;

        u = c - a;
        v = d - b;

        if (u == 0)
        {
                d1x = 0;
                m = 0;
        }
        else
        {
                m = abs(u);
                if (u < 0)
                        d1x = -1;
                else
                if (u > 0)
                        d1x = 1;
        }
        if ( v == 0)
        {
                d1y = 0;
                n = 0;
        }
        else
        {
                n = abs(v);
                if (v < 0)
                        d1y = -1;
                else
                if (v > 0)
                        d1y = 1;
        }
        if (m > n)
        {
                d2x = d1x;
                d2y = 0;
        }
        else
        {
                d2x = 0;
                d2y = d1y;
                m = n;
                n = abs(u);
        }
        s = m / 2;

        for (i = 0; i <= m; i++)
        {
                asm mov al , col;
                asm mov bh , 0;
                asm mov ah ,0Ch;
                asm mov cx ,a;
                asm mov dx ,b;
                asm int 10h;

                s += n;
                if (s >= m)
                {
                        s -= m;
                        a += d1x;
                        b += d1y;
                }
                else
                {
                        a += d2x;
                        b += d2y;
                }
        }
        _lastx = a;
        _lasty = b;
}

void ellipse(int x, int y, int xrad, int yrad,double incline,int col)
{
        /* Draws an ellipse */

        int f;
        float a;
        float b;
        float c;
        float d;
        int cols;
        double div;

        incline = 1 / sintable[(int)incline];

        if (getmode(&cols) == 6)
                div = 2.2;
        else
                div = 1.3;

                /* Compensate for pixel shape */

        a = x + xrad;
```

```
		b = y + sintable[0] * yrad + xrad/incline / div;

		for(f = 5; f < 360; f += 5)
		{
			c = x + costable[f] * xrad;
			d = y + sintable[f] * yrad + (costable[f] * xrad)/incline/div;

			line(a,b,c,d,col);

			a = c;
			b = d;
		}
		/* Ensure join */
		line(a,b,x + xrad,y + sintable[0] * yrad + xrad/incline / div,col);
	}

	void polygon(int x, int y, int rad, int col, int sides, int start)
	{
		/* Draws a regular polygon */

		double f;
		double div;
		double a;
		double b;
		double c;
		double d;
		double aa;
		double bb;
		int cols;
		double step;

		step = 360 / sides;

		if (getmode(&cols) == 6)
			div = 2.2;
		else
			div = 1.3;
		aa = a = x + costable[start] * rad;
		bb = b = y + sintable[start] * rad / div;

		for(f = start + step; f < start + 360; f += step)
		{
			c = x + costable[(int)f] * rad;
			d = y + sintable[(int)f] * rad / div;
			line(a,b,c,d,col);
			a = c;
			b = d;
		}
		line(a,b,aa,bb,col);
	}

	void arc(int x, int y, int rad, int start, int end,int col)
	{
		/* Draw an arc */

		int f;
		float a;
		float b;
```

```
                float c;
                float d;
                int cols;
                float div;

                if (getmode(&cols) == 6)
                        div = 2.2;
                else
                        div = 1.3;
                a = x + costable[start] * rad;
                b = y + sintable[start] * rad / div;

                for(f = start; f <= end; f ++)
                {
                        c = x + costable[f] * rad;
                        d = y + sintable[f] * rad / div;
                        line(a,b,c,d,col);
                        a = c;
                        b = d;
                }
        }

        void segm(int x, int y, int rad, int start, int end,int col)
        {
                /* Draw a segment of a circle */

                int f;
                float a;
                float b;
                float c;
                float d;
                int cols;
                double div;

                if (getmode(&cols) == 6)
                        div = 2.2;
                else
                        div = 1.3;
                a = x + costable[start] * rad;
                b = y + sintable[start] * rad / div;

                line(x,y,a,b,col);

                for(f = start; f <= end ; f ++)
                {
                        c = x + costable[f] * rad;
                        d = y + sintable[f] * rad / div;
                        line(a,b,c,d,col);
                        a = c;
                        b = d;
                }
                line(x,y,a,b,col);
        }

        void box(int xa,int ya, int xb, int yb, int col)
        {
                /* Draws a box for use in 2d histogram graphs etc */
                line(xa,ya,xa,yb,col);
                line(xa,yb,xb,yb,col);
                line(xb,yb,xb,ya,col);
                line(xb,ya,xa,ya,col);
        }

        void tri(int xa,int ya, int xb, int yb, int xc, int yc,int col)
        {
                /* Draw a triangle */

                line(xa,ya,xb,yb,col);
                line(xb,yb,xc,yc,col);
                line(xc,yc,xa,ya,col);
        }

        void fill(int x, int y, int col,int pattern)
        {
                /* Fill a boundered shape using a hatch pattern */

                int xa;
                int ya;
                int bn;
                int byn;

                int hatch[10][8] = {    255,255,255,255,255,255,255,255,
                                        128,64,32,16,8,4,2,1,
                                        1,2,4,8,16,32,64,128,
                                        1,2,4,8,8,4,2,1,
                                        238,238,238,238,238,238,238,238,
                                        170,85,170,85,170,85,170,85,
                                        192,96,48,24,12,6,3,1,
                                        62,62,62,0,227,227,227,0,
                                        129,66,36,24,24,36,66,129,
                                        146,36,146,36,146,36,146,36
                                };

                /* Patterns for fill, each integer describes a row of dots */

                xa = x;
                ya = y;  /* Save Origin */

                if(pixset(x,y))
                        return;

                bn = 1;
                byn = 0;

                do
                {
                        if (hatch[pattern][byn] != 0)
                        {
                                /* If blank ignore */
                                do
                                {
                                        if ((bn & hatch[pattern][byn]) == bn)
                                        {
                                                asm mov al , col;
                                                asm mov bh , 00;
                                                asm mov cx , x;
                                                asm mov dx , y;
                                                asm mov ah , 0Ch;
                                                asm int 10h;
                                        }
                                        x--;
                                        bn <<= 1;
                                        if (bn > 128)
                                                bn = 1;
                                }
                                while(!pixset(x,y) && (x > -1));

                                x = xa + 1;
                                bn = 128;

                                do
                                {
                                        if ((bn & hatch[pattern][byn]) == bn)
                                        {
                                                asm mov al , col;
                                                asm mov bh , 00;
                                                asm mov cx , x;
                                                asm mov dx , y;
                                                asm mov ah , 0Ch;
                                                asm int 10h;
                                        }
                                        x++;
                                        bn >>=1;
                                        if (bn <1)
                                                bn = 128;
                                }
                                while((!pixset(x,y)) && (x <= maxx));
                        }
                        x = xa;
                        y--;
                        bn = 1;
                        byn++;
                        if (byn > 7)
                                byn = 0;

                }
                while(!pixset(x,y) && ( y > -1));

                /* Now travel downwards */

                y = ya + 1;

                byn = 7;
                bn = 1;
                do
                {
                        /* Travel left */
                        if (hatch[pattern][byn] !=0)
                        {
                                do
                                {
                                        if ( (bn & hatch[pattern][byn]) == bn)
                                        {
```

```
                                        asm mov al , col;
                                        asm mov bh , 00;
                                        asm mov cx , x;
                                        asm mov dx , y;
                                        asm mov ah , 0Ch;
                                        asm int 10h;
                                }

                                x--;
                                bn <<= 1;
                                if (bn > 128)
                                        bn = 1;
                        }
                        while(!pixset(x,y) && (x > -1));

                        /* Back to x origin */
                        x = xa + 1 ;
                        bn = 128;

                        /* Travel right */
                        do
                        {
                                if ((bn & hatch[pattern][byn]) == bn)
                                {
                                        asm mov al , col;
                                        asm mov bh , 00;
                                        asm mov cx , x;
                                        asm mov dx , y;
                                        asm mov ah , 0Ch;
                                        asm int 10h;
                                }
                                x++;
                                bn >>=1;
                                if (bn <1)
                                        bn = 128;
                        }
                        while((!pixset(x,y)) && (x <= maxx));
                }
                x = xa;
                bn = 1;
                y++;
                byn--;
                if (byn < 0)
                        byn = 7;
        }
        while((!pixset(x,y)) && (y <= maxy));
}

void invert(int xa,int ya, int xb, int yb, int col)
{
        /* Invert a pixel window */

        union REGS inreg,outreg;

        inreg.h.al = (unsigned char)(128 | col);
        inreg.h.ah = 0x0C;
        for(inreg.x.cx = (unsigned int)xa;            inreg.x.cx <= (unsigned int)xb; inreg.x.cx++)
                for(inreg.x.dx = (unsigned)ya; inreg.x.dx <= (unsigned)yb; inreg.x.dx++)
```

```
                        int86(0x10, &inreg, &outreg);
        }

        void circle(int x_centre , int y_centre, int radius, int colour)
        {
                int x,y,delta;
                int startx,endx,x1,starty,endy,y1;
                int asp_ratio;

                if (_dmode == 6)
                        asp_ratio = 22;
                else
                        asp_ratio = 13;

                y = radius;
                delta = 3 - 2 * radius;
                for(x = 0; x < y; )
                {
                        starty = y * asp_ratio / 10;
                        endy = (y + 1) * asp_ratio / 10;
                        startx = x * asp_ratio / 10;
                        endx = (x + 1) * asp_ratio / 10;
                        for(x1 = startx; x1 < endx; ++x1)
                        {
                                plot(x1+x_centre,y+y_centre,colour);
                                plot(x1+x_centre,y_centre - y,colour);
                                plot(x_centre - x1,y_centre - y,colour);
                                plot(x_centre - x1,y + y_centre,colour);
                        }

                        for(y1 = starty; y1 < endy; ++y1)
                        {
                                plot(y1+x_centre,x+y_centre,colour);
                                plot(y1+x_centre,y_centre - x,colour);
                                plot(x_centre - y1,y_centre - x,colour);
                                plot(x_centre - y1,x + y_centre,colour);
                        }

                        if (delta < 0)
                                delta += 4 * x + 6;
                        else
                        {
                                delta += 4*(x-y)+10;
                                y--;
                        }
                        x++;
                }

                if(y)
                {
                        starty = y * asp_ratio / 10;
                        endy = (y + 1) * asp_ratio / 10;
                        startx = x * asp_ratio / 10;
                        endx = (x + 1) * asp_ratio / 10;
                        for(x1 = startx; x1 < endx; ++x1)
                        {
                                plot(x1+x_centre,y+y_centre,colour);
                                plot(x1+x_centre,y_centre - y,colour);
```

```
                    plot(x_centre - x1,y_centre - y,colour);
                    plot(x_centre - x1,y + y_centre,colour);
                }

                for(y1 = starty; y1 < endy; ++y1)
                {
                    plot(y1+x_centre,x+y_centre,colour);
                    plot(y1+x_centre,y_centre - x,colour);
                    plot(x_centre - y1,y_centre - x,colour);
                    plot(x_centre - y1,x + y_centre,colour);
                }
        }
}

void draw(int x, int y, int colour)
{
        /* Draws a line from _lastx,_lasty to x,y */

        line(_lastx,_lasty,x,y,colour);
}

void psprite(SPRITE *sprite,int x,int y)
{
        int origx;
        int origy;
        int z;
        int count;
        int col;
        int pos;
        unsigned char far *video;

        if (_dmode == 19)
        {
                /* Super fast direct video write in mode 19 for sprites */

                video = MK_FP(0xA000,0);

                origx = x;
                origy = y;

                if (sprite->x != -1)
                {
                        /* This sprite has been displayed before */
                        /* replace background */
                        /* This must be done first in case the sprite overlaps itself */
                        x = sprite->x;
                        y = sprite->y;
                        col = 0;
                        pos = x + y * 320;
                        for(count = 0; count < 256; count++)
                        {
                                video[pos] = sprite->save[count];
                                col++;
                                if (col == 16)
                                {
                                        pos += 305;
                                        col = 0;
                                }
                                else
                                        pos++;
                        }
                }

                x = origx;
                y = origy;
                col = 0;

                pos = x + y * 320;

                for(count = 0; count < 256; count++)
                {
                        sprite->save[count] = video[pos];
                        z = sprite->data[count];
                        if (z != 255)
                                video[pos] = z;

                        col++;
                        if (col == 16)
                        {
                                pos += 305;
                                col = 0;
                        }
                        else
                                pos++;
                }
                sprite->x = origx;
                sprite->y = origy;

                return;
        }

        origx = x;
        origy = y;

        if (sprite->x != -1)
        {
                /* This sprite has been displayed before */
                /* replace background */
                /* This must be done first in case the sprite overlaps itself */
                x = sprite->x;
                y = sprite->y;
                col = 0;
                for(count = 0; count < 256; count++)
                {
                        if ((x >= 0) && (y >= 0) && (x < maxx) && (y < maxy))
                        {
                                z = sprite->save[count];
                                asm mov al , z;
                                asm mov bh , 00;
                                asm mov cx , x;
                                asm mov dx , y;
                                asm mov ah , 0Ch;
                                asm int 10h;
                        }
                        col++;
                        if (col == 16)
```

```
                        {
                                y++;
                                x = sprite->x;
                                col = 0;
                        }
                        else
                                x++;
                }
        }

        x = origx;
        y = origy;
        col = 0;

        for(count = 0; count < 256; count++)
        {
                if ((x >= 0) && (y >= 0) && (x < maxx) && (y < maxy))
                {
                        asm mov cx , x;
                        asm mov dx , y;
                        asm mov ah , 0Dh;
                        asm int 10h;
                        asm mov z ,al;
                        sprite->save[count] = z;
                        z = sprite->data[count];

                        if (z != 255)
                        {
                                asm mov al , z;
                                asm mov bh , 0;
                                asm mov cx , x;
                                asm mov dx , y;
                                asm mov ah , 0Ch;
                                asm int 10h;
                        }
                }
                col++;
                if (col == 16)
                {
                        y++;
                        x = origx;
                        col = 0;
                }
                else
                        x++;
        }
        sprite->x = origx;
        sprite->y = origy;
        return;
}
```

Displaying A PCX File

The following program is offered as a practical example of graphics with the IBM PC. It reads a file of the 'PCX' format and displays the image on the screen.

```
/* Read a PCX file and display image */

#include <dos.h>
#include <io.h>
#include <fcntl.h>

typedef struct
{
        unsigned char man;
        unsigned char version;
        unsigned char encoding;
        unsigned char bpp;
        int xmin;
        int ymin;
        int xmax;
        int ymax;
        int hdpi;
        int vdpi;
        int colormap[24];
        char reserved;
        unsigned char planes;
        int bpl;
        int palette;
        int hss;
        int vsize;
        char pad[54];
}
PCX_HEADER;

PCX_HEADER header;

int x;
int y;

union REGS inreg,outreg;

void setvideo(unsigned char mode)
{
        /* Sets the video display mode   and clears the screen */

        inreg.h.al = mode;
        inreg.h.ah = 0x00;
        int86(0x10, &inreg, &outreg);
}

void plot(int x, int y, unsigned char colour)
{

          if (x < header.xmax && y < header.ymax)
          {

          /* Direct video plot in modes 16 & 18 only! */
          asm mov   ax,y;
          asm mov   dx,80;
          asm mul   dx;
          asm mov   bx,x;
          asm mov   cl,bl;
```

```
                asm shr   bx,1;
                asm shr   bx,1;
                asm shr   bx,1;
                asm add   bx,ax;

                asm and   cl,7;
                asm xor   cl,7;
                asm mov   ah,1;
                asm shl   ah,cl;

                asm mov   dx,3ceh;
                asm mov   al,8;
                asm out   dx,ax;

                asm mov   ax,(02h shl 8) + 5;
                asm out   dx,ax;

                asm mov   ax,0A000h;
                asm mov   es,ax;

                asm mov   al,es:[bx];
                asm mov   al,byte ptr colour;
                asm mov   es:[bx],al;

                asm mov   ax,(0FFh shl 8 ) + 8;
                asm out   dx,ax;

                asm mov   ax,(00h shl 8) + 5;
                asm out   dx,ax;
        }
}

void DISPLAY(unsigned char data)
{
        int n;
        int bit;

        bit = 32;

        for (n = 0; n < 6; n++)
        {
                if (data & bit)
                        plot(x,y,1);
                else
                        plot(x,y,0);
                bit >>= 1;
                x++;
        }
}

main(int argc, char *argv[])
{
        int fp;
        int total_bytes;
        int n;
        unsigned char data;
        int count;
        int scan;

        if (argc != 2)
        {
                puts("USAGE IS getpcx <filename>");
                exit(0);
        }

        setvideo(16);

        x = 0;
        y = 0;

        fp = open(argv[1],O_RDONLY|O_BINARY);

        _read(fp,&header,128);

        total_bytes = header.planes * header.bpl;

        for(scan = 0; scan <= header.ymax; scan++)
        {
                x = 0;

                /* First scan line */

                for(n = 0; n < total_bytes; n++)
                {
                        /* Read byte */
                        _read(fp,&data,1);

                        count = data & 192;

                        if (count == 192)
                        {
                                count = data & 63;
                                n += count - 1;
                                _read(fp,&data,1);
                                while(count)
                                {
                                        DISPLAY(data);
                                        count--;
                                }
                        }
                        else
                                DISPLAY(data);

                }
                x = 0;
                y++;
        }
}
```

Drawing Circles

What has drawing circles got to do with advanced C programming? Well quite a lot, it is a task which is often desired by modern programmers, and it is a task which can be attacked from a number of angles. This example illustrates some of the ideas already discussed for replacing floating point numbers with integers, and using lookup tables rather than repeat calls to maths functions.

A circle may be drawn by plotting each point on its circumference. The location of any point is given by;

```
Xp = Xo + Sine(Angle) * Radius
Yp = Yo + Cosine(Angle) * Radius
```

Where Xp,Yp is the point to be plotted, and Xo,Yo is the centre of the circle.

Thus, the simplest way to draw a circle is to calculate Xp and Yp for each angle between 1 and 360 degrees and to plot these points. There is however one fundamental error with this. As the radius of the circle increases, so also does the length of the arc between each angular point. Thus a circle of sufficient radius will be drawn with a dotted line rather than a solid line.

The problem of the distance between the angular points may be tackled in two ways;

1) The number of points to be plotted can be increased, to say every 30 minutes.
2) A straight line may be drawn between each angular point.

The simplest circle drawing pseudo-code may then look like this;

```
FOR angle = 1 TO 360
    PLOT Xo + SINE(angle) * radius, Yo + COSINE(angle) * radius
NEXT angle
```

This code has two major disadvantages;

1) It uses REAL numbers for the sine and cosine figures
2) It makes numerous calculations of sine and cosine values

Both of these disadvantages result in a very slow piece of code. Since a circle is a regular figure with two axis of symmetry, one in both the X and Y axis, one only needs to calculate the relative offsets of points in one quadrant of the circle and then these offsets may be applied to the other three quadrants to produce a faster piece of code. Faster because the slow sine and cosine calculations are only done 90 times instead of 360 times;

```
FOR angle = 1 TO 90
    Xa = SINE(angle) * radius
    Ya = COSINE(angle) * radius
    PLOT Xo + Xa, Yo + Ya
    PLOT Xo + Xa, Yo - Ya
    PLOT Xo - Xa, Yo + Ya
    PLOT Xo - Xa, Yo - Ya
NEXT angle
```

A further enhancement may be made by making use of sine and cosine lookup tables instead of calculating them. This means calculating the sine and cosine values for each required angle and storing them in a table. Then, instead of calculating the values for each angle the circle drawing code need only retrieve the values from a table;

```
FOR angle = 1 TO 90
        Xa = SINE[angle] * radius
        Ya = COSINE[angle] * radius
        PLOT Xo + Xa, Yo + Ya
        PLOT Xo + Xa, Yo - Ya
        PLOT Xo - Xa, Yo + Ya
        PLOT Xo - Xa, Yo - Ya
NEXT angle
```

Most computer languages work in RADIANS rather than DEGREES. There being approximately 57 degrees in one radian, 2 * PI radians in one circle. This implies that to calculate sine and cosine values of sufficient points to draw a reasonable circle using radians one must again use real numbers, that is numbers which have figures following a decimal point. Real number arithmetic, also known as floating point arithmetic, is horrendously slow to calculate. Integer arithmetic on the other hand is very quick to calculate, but less precise.

To use integer arithmetic in circle drawing code requires ingenuity. If one agrees to use sine and cosine lookup tables for degrees, rather than radians. Then the sine value of an angle of 1 degree is;

0.0175

Which, truncated to an integer becomes zero! To overcome this the sine and cosine values held in the table should be multiplied by some factor, say 10000. Then, the integer value of the sine of an angle of 1 degree becomes;

175

If the sine and cosine values have been multiplied by a factor, then when the calculation of the point’s offset is carried out one must remember to divide the result by the same factor. Thus the calculation becomes;

```
Xa = SINE[angle] * radius / factor
Ya = COSINE[angle] * radius / factor
```

The final obstacle to drawing circles on a computer is the relationship between the width of the display screen and its height. This ratio between width and height is known as the “aspect ratio” and varies upon video display mode. The IBM VGA 256 colour mode for example can display 320 pixels across and 200 up the screen. This equates to an aspect ratio of 1:1.3. If the circle drawing code ignores the aspect ratio, then the shape displayed will often be ovalar to a greater or lesser degree due to the rectangular shape of the display pixels. Thus in order to display a true circle, the formulae to calculate each point on the circumference must be amended to calculate a slight ellipse in compensation of the distorting factor of the display.

The offset formulae then become;

```
Xa = SINE[angle] * radius / factor
Ya = COSINE[angle] * radius / (factor * aspect ratio)
```

The following short C program illustrates a practical circle drawing code segment, in a demonstrable form;

```
/* Circles.c   A demonstration circle drawing program for the IBM PC */


#include <stdlib.h>

int sintable[91] = {0,175,349,523,698,
                    872,1045,1219,1392,
                    1564,1736,1908,2079,
                    2250,2419,2588,2756,
                    2924,3090,3256,3420,
                    3584,3746,3907,4067,
                    4226,4384,4540,4695,
                    4848,5000,5150,5299,
                    5446,5592,5736,5878,
                    6018,6157,6293,6428,
                    6561,6691,6820,6947,
                    7071,7193,7314,7431,
                    7547,7660,7771,7880,
                    7986,8090,8192,8290,
                    8387,8480,8572,8660,
                    8746,8829,8910,8988,
                    9063,9135,9205,9272,
                    9336,9397,9455,9511,
                    9563,9613,9659,9703,
                    9744,9781,9816,9848,
                    9877,9903,9925,9945,
                    9962,9976,9986,9994,
                    9998,10000
};

int costable[91] = { 10000,9998,9994,9986,9976,
                    9962,9945,9925,9903,
                    9877,9848,9816,9781,
                    9744,9703,9659,9613,
                    9563,9511,9455,9397,
                    9336,9272,9205,9135,
                    9063,8988,8910,8829,
                    8746,8660,8572,8480,
                    8387,8290,8192,8090,
                    7986,7880,7771,7660,
                    7547,7431,7314,7193,
                    7071,6947,6820,6691,
                    6561,6428,6293,6157,
                    6018,5878,5736,5592,
                    5446,5299,5150,5000,
                    4848,4695,4540,4384,
                    4226,4067,3907,3746,
                    3584,3420,3256,3090,
                    2924,2756,2588,2419,
                    2250,2079,1908,1736,
                    1564,1392,1219,1045,
                    872,698,523,349,
                    175,0
};

void setvideo(unsigned char mode)
{
        /* Sets the video display mode for an IBM PC */

        asm mov al , mode;
        asm mov ah , 00;
        asm int 10h;
}

void plot(int x, int y, unsigned char colour)
{
        /* Code for IBM PC BIOS ROM */
        /* Sets a pixel at the specified coordinates to a specified colour */

        /* Return if out of range */
```

```
		if (x < 0 || y < 0 || x > 320 || y > 200)
			return;

		asm mov al , colour;
		asm mov bh , 0;
		asm mov cx , x;
		asm mov dx , y;
		asm mov ah, 0Ch;
		asm int 10h;
	}

	void Line(int a, int b, int c, int d, int col)
	{
		/* Draws a straight line from point a,b to point c,d in colour col */

		int u;
		int v;
		int d1x;
		int d1y;
		int d2x;
		int d2y;
		int m;
		int n;
		double s;	/* The only real number variable, but it's essential */
		int i;

		u = c - a;
		v = d - b;
		if (u == 0)
		{
			d1x = 0;
			m = 0;
		}
		else
		{
			m = abs(u);
			if (u < 0)
				d1x = -1;
			else
			if (u > 0)
				d1x = 1;
		}

		if ( v == 0)
		{
			d1y = 0;
			n = 0;
		}
		else
		{
			n = abs(v);
			if (v < 0)
				d1y = -1;
			else
			if (v > 0)
				d1y = 1;
		}
		if (m > n)
```

```
        {
                d2x = d1x;
                d2y = 0;
        }
        else
        {
                d2x = 0;
                d2y = d1y;
                m = n;
                n = abs(u);
        }
        s = m / 2;

        for (i = 0; i <= m; i++)
        {
                plot(a,b,col);
                s += n;
                if (s >= m)
                {
                        s -= m;
                        a += d1x;
                        b += d1y;
                }
                else
                {
                        a += d2x;
                        b += d2y;
                }
        }
}


void Circle(int x, int y, int rad, int col)
{
        /* Draws a circle about origin x,y */
        /* With a radius of rad */
        /* The col parameter defines the colour for plotting */

        int f;
        long xa;
        long ya;
        int a1;
        int b1;
        int a2;
        int b2;
        int a3;
        int b3;
        int a4;
        int b4;


        /* Calculate first point in each segment */

        a1 = x + ((long)(costable[0]) * (long)(rad) + 5000) / 10000;
        b1 = y + ((long)(sintable[0]) * (long)(rad) + 5000) / 13000;

        a2 = x - ((long)(costable[0]) * (long)(rad) + 5000) / 10000;
        b2 = y + ((long)(sintable[0]) * (long)(rad) + 5000) / 13000;
        a3 = x - ((long)(costable[0]) * (long)(rad) + 5000) / 10000;
        b3 = y - ((long)(sintable[0]) * (long)(rad) + 5000) / 13000;

        a4 = x + ((long)(costable[0]) * (long)(rad) + 5000) / 10000;
        b4 = y - ((long)(sintable[0]) * (long)(rad) + 5000) / 13000;

        /* Start loop at second point */
        for(f = 1; f <= 90; f++)
        {
                /* Calculate offset to new point */
                xa = ((long)(costable[f]) * (long)(rad) + 5000) / 10000;
                ya = ((long)(sintable[f]) * (long)(rad) + 5000) / 13000;

                /* Draw a line from the previous point to the new point in
                   each segment */
                Line(a1,b1,x + xa, y + ya,col);
                Line(a2,b2,x - xa, y + ya,col);
                Line(a3,b3,x - xa, y - ya,col);
                Line(a4,b4,x + xa, y - ya,col);

                /* Update the previous point in each segment */
                a1 = x + xa;
                b1 = y + ya;
                a2 = x - xa;
                b2 = y + ya;
                a3 = x - xa;
                b3 = y - ya;
                a4 = x + xa;
                b4 = y - ya;
        }
}

main()
{
        int n;

        /* Select VGA 256 colour 320 x 200 video mode */
        setvideo(19);

        /* Draw some circles */
        for(n = 0; n < 100; n++)
                Circle(160,100,n,n + 20);
}
```

Vesa Mode

The VESA BIOS provides a number of new, and exciting video modes not supported by the standard BIOS. These modes vary from one video card to another, but most support the following modes:

Mode	Display
0x54	Text 16 colours 132 x 43
0x55	Text 16 colours 132 x 25
0x58	Graphics 16 colours 800 x 600

0x5C	Graphics 256 colours 800 x 600
0x5D	Graphics 16 colours 1024 x 768
0x5F	Graphics 256 colours 640 x 480
0x60	Graphics 256 colours 1024 x 768
0x64	Graphics 64k colours 640 x 480
0x65	Graphics 64k colours 800 x 600
0x6A	Graphics 16 colours 800 x 600
0x6C	Graphics 16 colours 1280 x 1024
0x70	Graphics 16m colours 320 x 200
0x71	Graphics 16m colours 640 x 480

These modes are in addition to the standard BIOS video modes described earlier.

Setting a VESA video mode requires a call to a different BIOS function than the standard BIOS, as illustrated in the following example which enables any VESA or standard display mode to be selected from the DOS command line.

```
#include <dos.h>
#include <ctype.h>

void setvideo(int mode)
{
        /* Sets the video display to a VESA or normal mode and clears the screen */

        union REGS inreg,outreg;

        inreg.h.ah = 0x4f;
        inreg.h.al = 0x02;
        inreg.x.bx = mode;
        int86(0x10, &inreg, &outreg);
}

main(int argc, char *argv[])
{
        setvideo(atoi(argv[1]));
}
```

Plotting pixels in a VESA mode graphics display can be acgieved with the standard BIOS plot functiona call, as illustrated here;

```
void plot(int x, int y, unsigned char colour)
{
        asm mov al , colour;
        asm mov bh , 00;
        asm mov cx , x;
        asm mov dx , y;
        asm mov ah , 0Ch;
        asm int 10h;
}
```

Or, in a 800 x 600 resolution mode you can use this fast direct video access plot function;

```
void plot( int x, int y, unsigned char pcolor)
{
        /*
                Fast 800 x 600 mode (0x58 or 0x102) plotting
        */

        asm mov   ax,y;
        asm mov   dx,800/8;
        asm mul   dx;
        asm mov   bx,x;
        asm mov   cl,bl;

        asm shr   bx,1;
        asm shr   bx,1;
        asm shr   bx,1;
        asm add   bx,ax;

        asm and   cl,7;
        asm xor   cl,7;
        asm mov   ah,1;
        asm shl   ah,cl;

        asm mov   dx,03CEh;
        asm mov   al,8;
        asm out   dx,ax;

        asm mov   ax,(02h shl 8) + 5;
        asm out   dx,ax;

        asm mov   ax,0A000h;
        asm mov   es,ax;

        asm mov   al,es:[bx];
        asm mov   al,byte ptr pcolor;
        asm mov   es:[bx],al;

        asm mov   ax,(0FFh shl 8 ) + 8;
        asm out   dx,ax;

        asm mov   ax,(00h shl 8) + 5;
        asm out   dx,ax;
}
```

There are lots more functions supported by the VESA BIOS, but this will get you going with the basic operations. Remember though that when using VESA display modes, that the direct console I/O functions in the C compiler library may not function correctly.

DIRECTORY SEARCHING WITH THE IBM PC AND DOS

Amongst the many functions provided by DOS for programmers, are a pair of functions; “Find first” and “Find next” which are used to search a DOS directory for a specified file name or names. The first function, “Find first” is accessed via DOS interrupt 21, function 4E. It takes an ascii string file specification, which can include wildcards, and the required attribute for files to match. Upon return the function fills the disk transfer area (DTA) with details of the located file and returns with the carry flag clear. If an error occurs, such as no matching files are located, the function returns with the carry flag set.

Following a successful call to “Find first”, a program can call “Find next”, DOS interrupt 21, function 4F, to locate the next file matching the specifications provided by the initial call to “Find first”. If this function succeeds, then the DTA is filled in with details of the next matching file, and the function returns with the carry flag clear. Otherwise a return is made with the carry flag set.

Most C compilers for the IBM PC provide non standard library functions for accessing these two functions. Turbo C provides “findfirst()” and “findnext()”. Making use of the supplied library functions shields the programmer from the messy task of worrying about the DTA. Microsoft C programmers should substitue findfirst() with _dos_findfirst() and findnext() with _dos_findnext().

The following Turbo C example imitates the DOS directory command, in a basic form;

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

void main(void)
{
        /* Display directory listing of current directory */

        int done;
        int day;
        int month;
        int year;
        int hour;
        int min;
        char amflag;
        struct ffblk ffblk;
        struct fcb fcb;

        /* First display sub directory entries */
        done = findfirst("*.",&ffblk,16);

        while (!done)
        {
                year = (ffblk.ff_fdate >> 9) + 80;
                month = (ffblk.ff_fdate >> 5) & 0x0f;
                day = ffblk.ff_fdate & 0x1f;
                hour = (ffblk.ff_ftime >> 11);
                min = (ffblk.ff_ftime >> 5) & 63;

                amflag = 'a';

                if (hour > 12)
                {
                        hour -= 12;
                        amflag = 'p';
                }
```

```
				printf("%-11.11s  <DIR> %02d-%02d-%02d  %2d:%02d%c\n",
						ffblk.ff_name,day,month,year,hour,min,amflag);
				done = findnext(&ffblk);
			}

			/* Now all files except directories */
			done = findfirst("*.*",&ffblk,231);

			while (!done)
			{
				year = (ffblk.ff_fdate >> 9) + 80;
				month = (ffblk.ff_fdate >> 5) & 0x0f;
				day = ffblk.ff_fdate & 0x1f;
				hour = (ffblk.ff_ftime >> 11);
				min = (ffblk.ff_ftime >> 5) & 63;

				amflag = 'a';

				if (hour > 12)
				{
					hour -= 12;
					amflag = 'p';
				}

				parsfnm(ffblk.ff_name,&fcb,1);

				printf("%-8.8s %-3.3s %8ld  %02d-%02d-%02d  %2d:%02d%c\n",
						fcb.fcb_name,fcb.fcb_ext,ffblk.ff_fsize,
						day,month,year,hour,min,amflag);
				done = findnext(&ffblk);
			}
		}
```

The function "parsfnm()" is a Turbo C library command which makes use of the DOS function for parsing an ascii string containing a file name, into its component parts. These component parts are then put into a DOS file control block (fcb), from where they may be easily retrieved for displaying by printf().

The DOS DTA is comprised as follows;

Offset	Length	Contents
00	15	Reserved
15	Byte	Attribute of matched file
16	Word	File time
18	Word	File date
1A	04	File size
1E	0D	File name and extension as ascii string

The file time word contains the time at which the file was last written to disc and is comprised as follows;

Bits	Contents
0 - 4	Seconds divided by 2
5 - 10	Minutes

11 - 15	Hours

The file date word holds the date on which the file was last written to disc and is comprised of;

Bits	Contents
0 - 4	Day
5 - 8	Month
9 - 15	Years since 1980

To extract these details from the DTA requires a little manipulation, as illustrated in the above example.

The DTA attribute flag is comprised of the following bits being set or not;

Bit	Attribute
0	Read only
1	Hidden
2	System
3	Volume label
4	Directory
5	Archive

ACCESSING EXPANDED MEMORY

Memory (RAM) in an IBM PC comes in three flavours; Conventional, Expanded and Extended. Conventional memory is the 640K of RAM which the operating system DOS can access. This memory is normally used. However, it is often insufficient for todays RAM hungry systems. Expanded memory is RAM which is addressed outside of the area of conventional RAM not by DOS but by a second program called a LIM EMS driver. Access to this device driver is made through interrupt 67h.

The main problem with accessing expanded memory is that no matter how much expanded memory is fitted to the computer, it can only be accessed through 16K blocks refered to as pages. So for example. If you have 2mB of expanded RAM fitted to your PC then that is comprised of 128 pages (128 * 16K = 2mB).

A program can determine whether a LIM EMS driver is installed by attempting to open the file ‘EMMXXXX0’ which is guarranteed by the LIM standard to be present as an IOCTL device when the device driver is active.

The following source code illustrates some basic functions for testing for and accessing expanded memory.

```
/*
Various functions for using Expanded memory
*/

#include <dos.h>
#define EMM    0x67

char far *emmbase;
emmtest()
{
        /*
        Tests for the presence of expnaded memory by attempting to
        open the file EMMXXXX0.
        */

        union REGS regs;
        struct SREGS sregs;
        int error;
        long handle;

        /* Attempt to open the file device EMMXXXX0 */
        regs.x.ax = 0x3d00;
        regs.x.dx = (int)"EMMXXXX0";
        sregs.ds = _DS;
        intdosx(&regs,&regs,&sregs);
        handle = regs.x.ax;
        error = regs.x.cflag;

        if (!error)
        {
                regs.h.ah = 0x3e;
                regs.x.bx = handle;
                intdos(&regs,&regs);
        }
        return error;
}

emmok()
{
        /*
```

```
            Checks whether the expanded memory manager responds correctly
            */

            union REGS regs;

            regs.h.ah = 0x40;
            int86(EMM,&regs,&regs);

            if (regs.h.ah)
                  return 0;

            regs.h.ah = 0x41;
            int86(EMM,&regs,&regs);

            if (regs.h.ah)
                  return 0;

            emmbase = MK_FP(regs.x.bx,0);
            return 1;
      }

      long emmavail()
      {
        /*
        Returns the number of available (free) 16K pages of expanded memory
        or -1 if an error occurs.
        */

              union REGS regs;

            regs.h.ah = 0x42;
            int86(EMM,&regs,&regs);
            if (!regs.h.ah)
                  return regs.x.bx;
            return -1;
      }

      long emmalloc(int n)
      {
            /*
            Requests 'n' pages of expanded memory and returns the file handle
            assigned to the pages or -1 if there is an error
            */

            union REGS regs;

            regs.h.ah = 0x43;
            regs.x.bx = n;
            int86(EMM,&regs,&regs);
            if (regs.h.ah)
                  return -1;
            return regs.x.dx;
      }

      emmmap(long handle, int phys, int page)
      {
            /*
            Maps a physical page from expanded memory into the page frame in the
```

```
		conventional memory 16K window so that data can be transfered between
		the expanded memory and conventional memory.
		*/

		union REGS regs;

		regs.h.ah = 0x44;
		regs.h.al = page;
		regs.x.bx = phys;
		regs.x.dx = handle;
		int86(EMM,&regs,&regs);
		return (regs.h.ah == 0);
	}

	void emmmove(int page, char *str, int n)
	{
		/*
		Move 'n' bytes from conventional memory to the specified expanded memory
		page
		*/

		char far *ptr;

		ptr = emmbase + page * 16384;
		while(n-- > 0)
			*ptr++ = *str++;
	}

	void emmget(int page, char *str, int n)
	{
		/*
		Move 'n' bytes from the specified expanded memory page into conventional
		memory
		*/

		char far *ptr;

		ptr = emmbase + page * 16384;
		while(n-- > 0)
			*str++ = *ptr++;
	}

	emmclose(long handle)
	{
		/*
		Release control of the expanded memory pages allocated to 'handle'
		*/

		union REGS regs;

		regs.h.ah = 0x45;
		regs.x.dx = handle;
		int86(EMM,&regs,&regs);
		return (regs.h.ah == 0);
	}

	/*
	Test function for the EMM routines
```

```
*/

void main()
{
        long emmhandle;
        long avail;
        char teststr[80];
        int i;

        if(!emmtest())
        {
                printf("Expanded memory is not present\n");
                exit(0);
        }

        if(!emmok())
        {
                printf("Expanded memory manager is not present\n");
                exit(0);
        }

        avail = emmavail();
        if (avail == -1)
        {
                printf("Expanded memory manager error\n");
                exit(0);
        }
        printf("There are %ld pages available\n",avail);

        /* Request 10 pages of expanded memory */
        if((emmhandle = emmalloc(10)) < 0)
        {
                printf("Insufficient pages available\n");
                exit(0);
        }

        for (i = 0; i < 10; i++)
        {
                sprintf(teststr,"%02d This is a test string\n",i);
                emmmap(emmhandle,i,0);
                emmmove(0,teststr,strlen(teststr) + 1);
        }

        for (i = 0; i < 10; i++)
        {
                emmmap(emmhandle,i,0);
                emmget(0,teststr,strlen(teststr) + 1);
                printf("READING BLOCK %d: %s\n",i,teststr);
        }

        emmclose(emmhandle);
}
```

ACCESSING EXTENDED MEMORY

Extended memory has all but taken over from Expanded Memory now (1996). It is faster and more useable than expanded memory. As with Expanded memory, Extended memory cannot be directly accessed through the standard DOS mode, and so a transfer buffer in conventional or "real-mode" memory needs to be used. The process to write data to Extended memory then involves copying the data to the transfer buffer in conventional memory, and from there copying it to Extended memory.

Before any use may be made of Extended memory, a program should test to see if Extended memory is available. The following function, XMS_init(), tests for the presence of Extended memory, and if available calls another function, GetXMSEntry() to initialise the program for using Extended Memory. The function also allocates a conventional memory transfer buffer.

```
/*
        BLOCKSIZE will be the size of our real-memory buffer that
        we'll swap XMS through (must be a multiple of 1024, since
        XMS is allocated in 1K chunks.)
*/

#ifdef __SMALL__
#define BLOCKSIZE (16L * 1024L)
#endif


#ifdef __MEDIUM__
#define BLOCKSIZE (16L * 1024L)
#endif


#ifdef __COMPACT__
#define BLOCKSIZE (64L * 1024L)
#endif

#ifdef __LARGE__
#define BLOCKSIZE (64L * 1024L)
#endif


char XMS_init()
{
        /*
                returns 0 if XMS present,
                        1 if XMS absent
                        2 if unable to allocate conventional memory transfer buffer
        */
        unsigned char status;
        _AX=0x4300;
        geninterrupt(0x2F);
        status = _AL;
        if(status==0x80)
        {
                GetXMSEntry();
                XMSBuf = (char far *) farmalloc(BLOCKSIZE);
                if (XMSBuf == NULL)
                        return 2;
                return 0;
        }
        return 1;
}

void GetXMSEntry(void)
{
        /*
                GetXMSEntry sets XMSFunc to the XMS Manager entry point
                so we can call it later
        */

        _AX=0x4310;
        geninterrupt(0x2F);
        XMSFunc= (void (far *)(void)) MK_FP(_ES,_BX);
}
```

Once the presence of Extended memory has been confirmed, a program can find out how much Extended memory is available;

```
void XMSSize(int *kbAvail, int *largestAvail)
{
        /*
                XMSSize returns the total kilobytes available, and the size
                in kilobytes of the largest available block
        */

        _AH=8;
        (*XMSFunc)();
        *largestAvail=_DX;
        *kbAvail=_AX;
}
```

The following function may be called to allocate a block of Extended memory, like you would allocate a block of conventional memory.

```
char AllocXMS(unsigned long numberBytes)
{
        /*
                Allocate a block of XMS memory numberBytes long
                Returns 1 on success
                        0 on failure
        */

        _DX = (int)(numberBytes / 1024);
        _AH = 9;
        (*XMSFunc)();
        if (_AX==0)
        {
                return 0;
        }
        XMSHandle=_DX;
        return 1;
}
```

Allocated Extended memory is not freed by DOS. A program using Extended memory must release it before terminating. This function frees a block of extended memory previously allocated by AllocXMS. Note, XMSHandle is a global variable of type int.

```
void XMS_free(void)
{
	/*
		Free used XMS
	*/
	_DX=XMSHandle;
	_AH=0x0A;
	(*XMSFunc)();
}
```

Two functions are now given. One for writing data to Extended memory, and one for reading data from Extended memory into conventional memory.

```
/*
	XMSParms is a structure for copying information to and from
	real-mode memory to XMS memory
*/

struct parmstruct
{
	/*
		blocklength is the size in bytes of block to copy
	*/
	unsigned long blockLength;

	/*
		sourceHandle is the XMS handle of source; 0 means that
		sourcePtr will be a 16:16 real-mode pointer, otherwise
		sourcePtr is a 32-bit offset from the beginning of the
		XMS area that sourceHandle points to
	*/

	unsigned int sourceHandle;
	far void *sourcePtr;

	/*
		destHandle is the XMS handle of destination; 0 means that
		destPtr will be a 16:16 real-mode pointer, otherwise
		destPtr is a 32-bit offset from the beginning of the XMS
		area that destHandle points to
	*/

	unsigned int destHandle;
	far void *destPtr;
}
XMSParms;


char XMS_write(unsigned long loc, char far *val, unsigned length)
{
	/*
```

```
                    Round length up to next even value
          */
          length += length % 2;

          XMSParms.sourceHandle=0;
          XMSParms.sourcePtr=val;
          XMSParms.destHandle=XMSHandle;
          XMSParms.destPtr=(void far *) (loc);
          XMSParms.blockLength=length;/* Must be an even number! */
          _SI = FP_OFF(&XMSParms);
          _AH=0x0B;
          (*XMSFunc)();
          if (_AX==0)
          {
                    return 0;
          }
          return 1;
}


void *XMS_read(unsigned long loc,unsigned length)
{
          /*
                    Returns pointer to data
                    or NULL on error
          */

          /*
                    Round length up to next even value
          */
          length += length % 2;

          XMSParms.sourceHandle=XMSHandle;
          XMSParms.sourcePtr=(void far *) (loc);
          XMSParms.destHandle=0;
          XMSParms.destPtr=XMSBuf;
          XMSParms.blockLength=length;            /* Must be an even number */
          _SI=FP_OFF(&XMSParms);
          _AH=0x0B;
          (*XMSFunc)();
          if (_AX==0)
          {
                    return NULL;
          }
          return XMSBuf;
}
```

And now putting it all together is a demonstration program.

```
/* A sequential table of variable length records in XMS */

#include <dos.h>
#include <stdio.h>
#include <stdlib.h>
#include <alloc.h>
#include <string.h>
```

```
#define TRUE 1
#define FALSE 0

/*
        BLOCKSIZE will be the size of our real-memory buffer that
        we'll swap XMS through (must be a multiple of 1024, since
        XMS is allocated in 1K chunks.)
*/

#ifdef __SMALL__
#define BLOCKSIZE (16L * 1024L)
#endif


#ifdef __MEDIUM__
#define BLOCKSIZE (16L * 1024L)
#endif


#ifdef __COMPACT__
#define BLOCKSIZE (64L * 1024L)
#endif

#ifdef __LARGE__
#define BLOCKSIZE (64L * 1024L)
#endif


/*
        XMSParms is a structure for copying information to and from
        real-mode memory to XMS memory
*/

struct parmstruct
{
        /*
                blocklength is the size in bytes of block to copy
        */
        unsigned long blockLength;

        /*
                sourceHandle is the XMS handle of source; 0 means that
                sourcePtr will be a 16:16 real-mode pointer, otherwise
                sourcePtr is a 32-bit offset from the beginning of the
                XMS area that sourceHandle points to
        */

        unsigned int sourceHandle;
        far void *sourcePtr;

        /*
                destHandle is the XMS handle of destination; 0 means that
                destPtr will be a 16:16 real-mode pointer, otherwise
                destPtr is a 32-bit offset from the beginning of the XMS
                area that destHandle points to
        */

        unsigned int destHandle;
```

```
		far void *destPtr;
	}
	XMSParms;

	void far (*XMSFunc) (void);	/* Used to call XMS manager (himem.sys) */
	char GetBuf(void);
	void GetXMSEntry(void);

	char *XMSBuf;	/* Conventional memory buffer for transfers */

	unsigned int XMSHandle;		/* handle to allocated XMS block */


	char XMS_init()
	{
		/*
			returns 0 if XMS present,
				1 if XMS absent
				2 if unable to allocate transfer buffer
		*/
		unsigned char status;
		_AX=0x4300;
		geninterrupt(0x2F);
		status = _AL;
		if(status==0x80)
		{
			GetXMSEntry();
			XMSBuf = (char far *) farmalloc(BLOCKSIZE);
			if (XMSBuf == NULL)
				return 2;
			return 0;
		}
		return 1;
	}

	void GetXMSEntry(void)
	{
		/*
			GetXMSEntry sets XMSFunc to the XMS Manager entry point
			so we can call it later
		*/

		_AX=0x4310;
		geninterrupt(0x2F);
		XMSFunc= (void (far *)(void)) MK_FP(_ES,_BX);
	}


	void XMSSize(int *kbAvail, int *largestAvail)
	{
		/*
			XMSSize returns the total kilobytes available, and the size
			in kilobytes of the largest available block
		*/

		_AH=8;
		(*XMSFunc)();
		*largestAvail=_DX;
		*kbAvail=_AX;
	}

	char AllocXMS(unsigned long numberBytes)
	{
		/*
			Allocate a block of XMS memory numberBytes long
		*/

		_DX = (int)(numberBytes / 1024);
		_AH = 9;
		(*XMSFunc)();
		if (_AX==0)
		{
			return FALSE;
		}
		XMSHandle=_DX;
		return TRUE;
	}

	void XMS_free(void)
	{
		/*
			Free used XMS
		*/
		_DX=XMSHandle;
		_AH=0x0A;
		(*XMSFunc)();
	}

	char XMS_write(unsigned long loc, char far *val, unsigned length)
	{
		/*
			Round length up to next even value
		*/
		length += length % 2;

		XMSParms.sourceHandle=0;
		XMSParms.sourcePtr=val;
		XMSParms.destHandle=XMSHandle;
		XMSParms.destPtr=(void far *) (loc);
		XMSParms.blockLength=length;/* Must be an even number! */
		_SI = FP_OFF(&XMSParms);
		_AH=0x0B;
		(*XMSFunc)();
		if (_AX==0)
		{
			return FALSE;
		}
		return TRUE;
	}


	void *XMS_read(unsigned long loc,unsigned length)
	{
		/*
			Returns pointer to data
			or NULL on error
```

```
        */

        /*
                Round length up to next even value
        */
        length += length % 2;

        XMSParms.sourceHandle=XMSHandle;
        XMSParms.sourcePtr=(void far *) (loc);
        XMSParms.destHandle=0;
        XMSParms.destPtr=XMSBuf;
        XMSParms.blockLength=length;/* Must be an even number */
        _SI=FP_OFF(&XMSParms);
        _AH=0x0B;
        (*XMSFunc)();
        if (_AX==0)
        {
                return NULL;
        }
        return XMSBuf;
}


/*
        Demonstration code
        Read various length strings into a single XMS block (EMB)
        and write them out again
*/

int main()
{
        int kbAvail,largestAvail;
        char buffer[80];
        char *p;
        long pos;
        long end;

        if (XMS_init() == 0)
                printf("XMS Available ...\n");
        else
        {
                printf("XMS Not Available\n");
                return(1);
        }

        XMSSize(&kbAvail,&largestAvail);
        printf("Kilobytes Available: %d; Largest block: %dK\n",kbAvail,largestAvail);

        if (!AllocXMS(2000 * 1024L))
                return(1);


        pos = 0;

        do
        {
                p = fgets(buffer,1000,stdin);
                if (p != NULL)
                {
                        XMS_write(pos,buffer,strlen(buffer) + 1);
                        pos += strlen(buffer) + 1;
                }
        }
        while(p != NULL);

        end = pos;

        pos = 0;

        do
        {
                memcpy(buffer,XMS_read(pos,100),70);
                printf("%s",buffer);
                pos += strlen(buffer) + 1;
        }
        while(pos < end);

        /*
                It is VERY important to free any XMS before exiting!
        */
        XMS_free();
        return 0;
}
```

TSR PROGRAMMING

Programs which remain running and resident in memory while other programs are running are the most exciting line of programming for many PC developers. This type of program is known as a "Terminate and Stay Resident" or "TSR" program and they are very difficult to program sucessfuly.

The difficulties in programming TSRs comes from the limitations of DOS which is not a multi-tasking operating system, and does not react well to re-enterant code. That is it's own functions (interrupts) calling themselves.

In theory a TSR is quite simple. It is an ordinary program which terminates not through the usual DOS terminate function, but through the DOS "keep" function - interrupt 27h. This function reserves an area of memory, used by the program so that no other programs will overwrite it. This in itself is not a very difficult task, excepting that the program needs to tell DOS how much memory to leave it!

The problems stem mainly from not being able to use DOS function calls within the TSR program once it has "gone resident".

There are a few basic rules which help to clarify the problems encountered in programming TSRs:

1. Avoid DOS function calls
2. Monitor the DOS busy flag, when this flag is nonzero, DOS is executing an interrupt 21h function and MUST NOT be disturbed!
3. Monitor interrupt 28h. This reveals when DOS is busy waiting for console input. At this time you can disturb DOS regardless of the DOS busy flag setting.
4. Provide some way of checking whether the TSR is already loaded to prevent multiple copies occuring in memory.
5. Remember that other TSR programs may be chained to interrupts, and so you must chain any interrupt vectors that your program needs.
6. Your TSR program must use its own stack, and NOT that of the running process.
7. TSR programs must be compiled in a small memory model with stack checking turned off.
8. When control passes to your TSR program, it must tell DOS that the active process has changed.

The following three source code modules describe a complete TSR program. This is a useful pop-up address book database which can be activated while any other program is running by pressing the key combination 'Alt' and '.'. If the address book does not respond to the key press, it is probably because DOS cannot be disturbed, and you should try to pop-it-up again.

```
/*
  A practical TSR program (a pop-up address book database)
  Compile in small memory model with stack checking OFF
*/

#include <dos.h>
#include <stdio.h>
#include <string.h>
#include <dir.h>

static union REGS rg;

/*
  Size of the program to remain resident
  experimentation is required to make this as small as possible
```

```
*/
unsigned sizeprogram = 28000/16;

/* Activate with Alt . */
unsigned scancode = 52;          /* . */
unsigned keymask = 8;            /* ALT */

char signature[]= "POPADDR";
char fpath[40];

/*
        Function prototypes
*/

void curr_cursor(int *x, int *y);
int resident(char *, void interrupt(*)());
void resinit(void);
void terminate(void);
void restart(void);
void wait(void);
void resident_psp(void);
void exec(void);

/*
  Entry point from DOS
*/

main(int argc, char *argv[])
{
        void interrupt ifunc();
        int ivec;

        /*
          For simplicity, assume the data file is in the root directory
          of drive C:
        */
        strcpy(fpath,"C:\\ADDRESS.DAT");

        if ((ivec = resident(signature,ifunc)) != 0)
        {
                /* TSR is resident */
                if (argc > 1)
                {
                        rg.x.ax = 0;
                        if (strcmp(argv[1],"quit") == 0)
                                rg.x.ax = 1;
                        else if (strcmp(argv[1],"restart") == 0)
                                rg.x.ax = 2;
                        else if (strcmp(argv[1],"wait") == 0)
                                rg.x.ax = 3;
                        if (rg.x.ax)
                        {
                                int86(ivec,&rg,&rg);
                                return;
                        }
                }
                printf("\nPopup Address Book is already resident");
        }
		else
		{
			/* Initial load of TSR program */
			printf("Popup Address Book Resident.\nPress Alt . To Activate....\n");
			resinit();
		}
	}

	void interrupt ifunc(bp,di,si,ds,es,dx,cx,bx,ax)
	{
		if(ax == 1)
			terminate();
		else if(ax == 2)
			restart();
		else if(ax == 3)
			wait();
	}

	popup()
	{
		int x,y;

		curr_cursor(&x,&y);

		/* Call the TSR C program here */
		exec();
		cursor(x,y);
	}

	/*
		Second source module
	*/

	#include <dos.h>
	#include <stdio.h>

	static union REGS rg;
	static struct SREGS seg;
	static unsigned mcbseg;
	static unsigned dosseg;
	static unsigned dosbusy;
	static unsigned enddos;
	char far *intdta;
	static unsigned intsp;
	static unsigned intss;
	static char far *mydta;
	static unsigned myss;
	static unsigned stack;
	static unsigned ctrl_break;
	static unsigned mypsp;
	static unsigned intpsp;
	static unsigned pids[2];
	static int pidctr = 0;
	static int pp;
	static void interrupt (*oldtimer)();
	static void interrupt (*old28)();
	static void interrupt (*oldkb)();
	static void interrupt (*olddisk)();
```

```
        static void interrupt (*oldcrit)();

        void interrupt newtimer();
        void interrupt new28();
        void interrupt newkb();
        void interrupt newdisk();
        void interrupt newcrit();

        extern unsigned sizeprogram;
        extern unsigned scancode;
        extern unsigned keymask;

        static int resoff = 0;
        static int running = 0;
        static int popflg = 0;
        static int diskflag = 0;
        static int kbval;
        static int cflag;

        void dores(void);
        void pidaddr(void);

        void resinit()
        {
                segread(&seg);
                myss = seg.ss;

                rg.h.ah = 0x34;
                intdos(&rg,&rg);
                dosseg = _ES;
                dosbusy = rg.x.bx;

                mydta = getdta();
                pidaddr();
                oldtimer = getvect(0x1c);
                old28 = getvect(0x28);
                oldkb = getvect(9);
                olddisk = getvect(0x13);

                setvect(0x1c,newtimer);
                setvect(9,newkb);
                setvect(0x28,new28);
                setvect(0x13,newdisk);

                stack = (sizeprogram - (seg.ds - seg.cs)) * 16 - 300;
                rg.x.ax = 0x3100;
                rg.x.dx = sizeprogram;
                intdos(&rg,&rg);
        }

        void interrupt newdisk(bp,di,si,ds,es,dx,cx,bx,ax,ip,cs,flgs)
        {
                diskflag++;
                (*olddisk)();
                ax = _AX;
                newcrit();
                flgs = cflag;
                --diskflag;
```

```
}

void interrupt newcrit(bp,di,si,ds,es,dx,cx,bx,ax,ip,cs,flgs)
{
        ax = 0;
        cflag = flgs;
}

void interrupt newkb()
{
        if (inportb(0x60) == scancode)
        {
                kbval = peekb(0,0x417);
                if (!resoff && ((kbval & keymask) ^ keymask) == 0)
                {
                        kbval = inportb(0x61);
                        outportb(0x61,kbval | 0x80);
                        outportb(0x61,kbval);
                        disable();
                        outportb(0x20,0x20);
                        enable();
                        if (!running)
                                popflg = 1;
                        return;
                }
        }
        (*oldkb)();
}

void interrupt newtimer()
{
        (*oldtimer)();
        if (popflg && peekb(dosseg,dosbusy) == 0)
                if(diskflag == 0)
                {
                        outportb(0x20,0x20);
                        popflg = 0;
                        dores();
                }
}

void interrupt new28()
{
        (*old28)();
        if (popflg && peekb(dosseg,dosbusy) != 0)
        {
                popflg = 0;
                dores();
        }
}

resident_psp()
{
        intpsp = peek(dosseg,*pids);
        for(pp = 0; pp < pidctr; pp++)
                poke(dosseg,pids[pp],mypsp);
}
```

```
interrupted_psp()
{
        for(pp = 0; pp < pidctr; pp++)
                poke(dosseg,pids[pp],intpsp);
}

void dores()
{
        running = 1;
        disable();
        intsp = _SP;
        intss = _SS;
        _SP = stack;
        _SS = myss;
        enable();
        oldcrit = getvect(0x24);
        setvect(0x24,newcrit);
        rg.x.ax = 0x3300;
        intdos(&rg,&rg);
        ctrl_break = rg.h.dl;
        rg.x.ax = 0x3301;
        rg.h.dl = 0;
        intdos(&rg,&rg);
        intdta = getdta();
        setdta(mydta);
        resident_psp();
        popup();
        interrupted_psp();
        setdta(intdta);
        setvect(0x24,oldcrit);
        rg.x.ax = 0x3301;
        rg.h.dl = ctrl_break;
        intdos(&rg,&rg);
        disable();
        _SP = intsp;
        _SS = intss;
        enable();
        running = 0;
}

static int avec = 0;

unsigned resident(char *signature,void interrupt(*ifunc)())
{
        char *sg;
        unsigned df;
        int vec;

        segread(&seg);
        df = seg.ds-seg.cs;
        for(vec = 0x60; vec < 0x68; vec++)
        {
                if (getvect(vec) == NULL)
                {
                        if (!avec)
                                avec = vec;
                        continue;
                }
                for(sg = signature; *sg; sg++)
                if (*sg != peekb(peek(0,2+vec*4)+df,(unsigned)sg))
                        break;
                if (!*sg)
                        return vec;
        }
        if (avec)
                setvect(avec,ifunc);
        return 0;
}

static void pidaddr()
{
        unsigned adr = 0;

        rg.h.ah = 0x51;
        intdos(&rg,&rg);
        mypsp = rg.x.bx;
        rg.h.ah = 0x52;
        intdos(&rg,&rg);
        enddos = _ES;
        enddos = peek(enddos,rg.x.bx-2);
        while(pidctr < 2 && (unsigned)((dosseg<<4) + adr) < (enddos <<4))
        {
                if (peek(dosseg,adr) == mypsp)
                {
                        rg.h.ah = 0x50;
                        rg.x.bx = mypsp + 1;
                        intdos(&rg,&rg);
                        if (peek(dosseg,adr) == mypsp + 1)
                                pids[pidctr++] = adr;
                        rg.h.ah = 0x50;
                        rg.x.bx = mypsp;
                        intdos(&rg,&rg);
                }
                adr++;
        }
}

static resterm()
{
        setvect(0x1c,oldtimer);
        setvect(9,oldkb);
        setvect(0x28,old28);
        setvect(0x13,olddisk);
        setvect(avec,(void interrupt (*)()) 0);
        rg.h.ah = 0x52;
        intdos(&rg,&rg);
        mcbseg = _ES;
        mcbseg = peek(mcbseg,rg.x.bx-2);
        segread(&seg);
        while(peekb(mcbseg,0) == 0x4d)
        {
                if(peek(mcbseg,1) == mypsp)
                {
                        rg.h.ah = 0x49;
                        seg.es = mcbseg+1;
                        intdosx(&rg,&rg,&seg);
```

```
                    }
                    mcbseg += peek(mcbseg,3) + 1;
            }
    }

    terminate()
    {
            if (getvect(0x13) == (void interrupt (*)()) newdisk)
                    if (getvect(9) == newkb)
                            if(getvect(0x28) == new28)
                                    if(getvect(0x1c) == newtimer)
                                    {
                                            resterm();
                                            return;
                                    }
            resoff = 1;
    }

    restart()
    {
            resoff = 0;
    }

    wait()
    {
            resoff = 1;
    }

    void cursor(int y, int x)
    {
            rg.x.ax = 0x0200;
            rg.x.bx = 0;
            rg.x.dx = ((y << 8) & 0xff00) + x;
            int86(16,&rg,&rg);
    }

    void curr_cursor(int *y, int *x)
    {
            rg.x.ax = 0x0300;
            rg.x.bx = 0;
            int86(16,&rg,&rg);
            *x = rg.h.dl;
            *y = rg.h.dh;
    }

    /*
      Third module, the simple pop-up address book
      with mouse support
    */

    #include <stdio.h>
    #include <stdlib.h>
    #include <io.h>
    #include <string.h>
    #include <fcntl.h>
    #include <sys\stat.h>
    #include <dos.h>
    #include <conio.h>
```

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

/* left cannot be less than 3 */
#define left        4

/* Data structure for records */
typedef struct
{
            char name[31];
            char company[31];
            char address[31];
            char area[31];
            char town[31];
            char county[31];
            char post[13];
            char telephone[16];
            char fax[16];
}
data;

extern char fpath[];

static char scr[4000];

static char sbuff[2000];
char stext[30];
data rec;
int handle;
int recsize;
union REGS inreg,outreg;

/*
            Function prototypes
*/
void FATAL(char *);
void OPENDATA(void);
void CONTINUE(void);
void EXPORT_MULTI(void);
void GETDATA(int);
int GETOPT(void);
void DISPDATA(void);
void ADD_REC(void);
void PRINT_MULTI(void);
void SEARCH(void);
void MENU(void);

int GET_MOUSE(int *buttons)
{
            inreg.x.ax = 0;
            int86(0x33,&inreg,&outreg);
            *buttons = outreg.x.bx;
            return outreg.x.ax;
}

void MOUSE_CURSOR(int status)
{
            /* Status = 0 cursor off */
```

```
          /*                    1 cursor on */

          inreg.x.ax = 2 - status;
          int86(0x33,&inreg,&outreg);
     }

     int MOUSE_LOCATION(int *x, int *y)
     {
          inreg.x.ax = 3;
          int86(0x33,&inreg,&outreg);

          *x = outreg.x.cx / 8;
          *y = outreg.x.dx / 8;

          return outreg.x.bx;
     }

     int GETOPT()
     {
          int result;
          int x;
          int y;

          do
          {
               do
               {
                    result = MOUSE_LOCATION(&x,&y);
                    if (result & 1)
                    {
                         if (x >= 52 && x <= 53 && y >= 7 && y <= 15)
                              return y - 7;
                         if (x >= 4 && x <= 40 && y >= 7 && y <= 14)
                              return y + 10;

                         if (x >= 4 && x <= 40 && y == 15)
                              return y + 10;
                    }
               }
               while(!bioskey(1));

               result = bioskey(0);
               x = result & 0xff;
               if (x == 0)
               {
                    result = result >> 8;
                    result -= 60;
               }
          }
          while(result < 0 || result > 8);
          return result;
     }

     void setvideo(unsigned char mode)
     {
          /* Sets the video display mode   and clears the screen */

          inreg.h.al = mode;
```

```
        inreg.h.ah = 0x00;
        int86(0x10, &inreg, &outreg);
}


int activepage(void)
{
        /* Returns the currently selected video display page */

        union REGS inreg,outreg;

        inreg.h.ah = 0x0F;
        int86(0x10, &inreg, &outreg);
        return(outreg.h.bh);
}

void print(char *str)
{
        /*
          Prints characters only directly to the current display page
          starting at the current cursor position. The cursor is not
          advanced.
          This function assumes a colour display card. For use with a
          monochrome display card change 0xB800 to read 0xB000
        */

        int page;
        int offset;
        unsigned row;
        unsigned col;
        char far *ptr;

        page = activepage();
        curr_cursor(&row,&col);

        offset = page * 4000 + row * 160 + col * 2;

        ptr = MK_FP(0xB800,offset);

        while(*str)
        {
                *ptr++= *str++;
                ptr++;
        }
}


void TRUESHADE(int lef, int top, int right, int bottom)
{
        int n;

        /* True Shading of a screen block */

        gettext(lef,top,right,bottom,sbuff);
        for(n = 1; n < 2000; n+= 2)
                sbuff[n] = 7;
        puttext(lef,top,right,bottom,sbuff);
}
```

```
void DBOX(int l, int t, int r, int b)
{
        /* Draws a double line box around the described area */

        int n;

        cursor(t,l);
        print("É");
        for(n = 1; n < r - l; n++)
        {
                cursor(t,l + n);
                print("Í");
        }
        cursor(t,r);
        print("»");

        for (n = t + 1; n < b; n++)
        {
                cursor(n,l);
                print("º");
                cursor(n,r);
                print("º");
        }
        cursor(b,l);
        print("È");
        for(n = 1; n < r - l; n++)
        {
                cursor(b,l+n);
                print("Í");
        }
        cursor(b,r);
        print("¼");
}

int INPUT(char *text,unsigned length)
{
        /* Receive a string from the operator */

        unsigned key_pos;
        int key;
        unsigned start_row;
        unsigned start_col;
        unsigned end;
        char temp[80];
        char *p;

        curr_cursor(&start_row,&start_col);

        key_pos = 0;
        end = strlen(text);
        for(;;)
        {
                key = bioskey(0);
                if ((key & 0xFF) == 0)
                {
                        key = key >> 8;
                        if (key == 79)
                        {
                                while(key_pos < end)
                                        key_pos++;
                                cursor(start_row,start_col + key_pos);
                        }
                        else
                        if (key == 71)
                        {
                                key_pos = 0;
                                cursor(start_row,start_col);
                        }
                        else
                        if ((key == 75) && (key_pos > 0))
                        {
                                key_pos--;
                                cursor(start_row,start_col + key_pos);
                        }
                        else
                        if ((key == 77) && (key_pos < end))
                        {
                                key_pos++;
                                cursor(start_row,start_col + key_pos);
                        }
                        else
                        if (key == 83)
                        {
                                p = text + key_pos;
                                while(*(p+1))
                                {
                                        *p = *(p+1);
                                        p++;
                                }
                                *p = 32;
                                if (end > 0)
                                        end--;
                                cursor(start_row,start_col);
                                cprintf(text);
                                cprintf(" ");
                                if ((key_pos > 0) && (key_pos == end))
                                        key_pos--;
                                cursor(start_row,start_col + key_pos);
                        }
                }
                else
                {
                        key = key & 0xFF;
                        if (key == 13 || key == 27)
                                break;
                        else
                        if ((key == 8) && (key_pos > 0))
                        {
                                end--;
                                key_pos--;
                                text[key_pos--] = '\0';
                                strcpy(temp,text);
                                p = text + key_pos + 2;
                                strcat(temp,p);
                                strcpy(text,temp);
```

```
                                        cursor(start_row,start_col);
                                        cprintf("%-*.*s",length,length,text);
                                        key_pos++;
                                        cursor(start_row,start_col + key_pos);
                                }
                                else
                                if ((key > 31) && (key_pos < length)  &&
                                   (start_col + key_pos < 80))
                                {
                                        if (key_pos <= end)
                                        {
                                                p = text + key_pos;
                                                memmove(p+1,p,end - key_pos);
                                                if (end < length)
                                                        end++;
                                                text[end] = '\0';
                                        }
                                        text[key_pos++] = (char)key;
                                        if (key_pos > end)
                                        {
                                                end++;
                                                text[end] = '\0';
                                        }
                                        cursor(start_row,start_col);
                                        cprintf("%-*.*s",length,length,text);
                                        cursor(start_row,start_col + key_pos);
                                }
                        }
                }
                text[end] = '\0';
                return key;
        }

        void FATAL(char *error)
        {
                /* A fatal error has occured */

                printf("\nFATAL ERROR: %s",error);
                exit(0);
        }

        void OPENDATA()
        {
                /* Check for existence of data file and if not create it */
                /* otherwise open it for reading/writing at end of file */

                handle = open(fpath,O_RDWR,S_IWRITE);

                if (handle == -1)
                {
                        handle = open(fpath,O_RDWR|O_CREAT,S_IWRITE);
                        if (handle == -1)
                                FATAL("Unable to create data file");
                }
                /* Read in first rec */
                read(handle,&rec,recsize);
        }
```

```
void CLOSEDATA()
{
        close(handle);
}

void GETDATA(int start)
{
        /* Get address data from operator */

        textcolor(BLACK);
        textbackground(GREEN);
        gotoxy(left,8);
        print("Name ");
        gotoxy(left,9);
        print("Company ");
        gotoxy(left,10);
        print("Address ");
        gotoxy(left,11);
        print("Area ");
        gotoxy(left,12);
        print("Town ");
        gotoxy(left,13);
        print("County ");
        gotoxy(left,14);
        print("Post Code ");
        gotoxy(left,15);
        print("Telephone ");
        gotoxy(left,16);
        print("Fax ");

        switch(start)
        {
                case 0: gotoxy(left + 10,8);
                        if(INPUT(rec.name,30) == 27)
                                break;
                case 1: gotoxy(left + 10,9);
                        if(INPUT(rec.company,30) == 27)
                                break;
                case 2: gotoxy(left + 10,10);
                        if(INPUT(rec.address,30) == 27)
                                break;
                case 3: gotoxy(left + 10,11);
                        if(INPUT(rec.area,30) == 27)
                                break;
                case 4: gotoxy(left + 10,12);
                        if(INPUT(rec.town,30) == 27)
                                break;
                case 5: gotoxy(left + 10,13);
                        if(INPUT(rec.county,30) == 27)
                                break;
                case 6: gotoxy(left + 10,14);
                        if(INPUT(rec.post,12) == 27)
                                break;
                case 7: gotoxy(left + 10,15);
                        if(INPUT(rec.telephone,15) == 27)
                                break;
                case 8: gotoxy(left + 10,16);
                        INPUT(rec.fax,15);
                        break;
        }
        textcolor(WHITE);
        textbackground(RED);
        gotoxy(left + 23,21);
        print("                                                         ");
}

void DISPDATA()
{
        /* Display address data */
        textcolor(BLACK);
        textbackground(GREEN);
        cursor(7,3);
        cprintf("Name      %-30.30s",rec.name);
        cursor(8,3);
        cprintf("Company   %-30.30s",rec.company);
        cursor(9,3);
        cprintf("Address   %-30.30s",rec.address);
        cursor(10,3);
        cprintf("Area      %-30.30s",rec.area);
        cursor(11,3);
        cprintf("Town      %-30.30s",rec.town);
        cursor(12,3);
        cprintf("County    %-30.30s",rec.county);
        cursor(13,3);
        cprintf("Post Code %-30.30s",rec.post);
        cursor(14,3);
        cprintf("Telephone %-30.30s",rec.telephone);
        cursor(15,3);
        cprintf("Fax       %-30.30s",rec.fax);
}

int LOCATE(char *text)
{
        int result;

        do
        {
                /* Read rec into memory */
                result = read(handle,&rec,recsize);
                if (result > 0)
                {
                        /* Scan rec for matching data */
                        if (strstr(strupr(rec.name),text) != NULL)
                                return(1);
                        if (strstr(strupr(rec.company),text) != NULL)
                                return(1);
                        if (strstr(strupr(rec.address),text) != NULL)
                                return(1);
                        if (strstr(strupr(rec.area),text) != NULL)
                                return(1);
                        if (strstr(strupr(rec.town),text) != NULL)
                                return(1);
                        if (strstr(strupr(rec.county),text) != NULL)
                                return(1);
                        if (strstr(strupr(rec.post),text) != NULL)
                                return(1);
                        if (strstr(strupr(rec.telephone),text) != NULL)
                                return(1);
                        if (strstr(strupr(rec.fax),text) != NULL)
                                return(1);
                }
        }
        while(result > 0);
        return(0);
}

void SEARCH()
{
        int result;

        gotoxy(left,21);
        textcolor(WHITE);
        textbackground(RED);
        cprintf("Enter data to search for ");
        strcpy(stext,"");
        INPUT(stext,30);
        if (*stext == 0)
        {
                gotoxy(left,21);
                cprintf("%70c",32);
                return;
        }
        gotoxy(left,21);
        textcolor(WHITE);
        textbackground(RED);
        cprintf("Searching for %s Please Wait....",stext);
        strupr(stext);
        /* Locate start of file */
        lseek(handle,0,SEEK_SET);
        result = LOCATE(stext);
        if (result == 0)
        {
                gotoxy(left,21);
                cprintf("%70c",32);
                gotoxy(left + 27,21);
                cprintf("NO MATCHING RECORDS");
                gotoxy(left + 24,22);
                cprintf("Press RETURN to Continue");
                bioskey(0);
                gotoxy(left,21);
                cprintf("%70c",32);
                gotoxy(left,22);
                cprintf("%70c",32);
        }
        else
        {
                lseek(handle,0 - recsize,SEEK_CUR);
                read(handle,&rec,recsize);
                DISPDATA();
        }
        textcolor(WHITE);
        textbackground(RED);
        gotoxy(left,21);
        cprintf("%70c",32);
```

```
                textcolor(BLACK);
                textbackground(GREEN);
        }

        void CONTINUE()
        {
                int result;
                long curpos;

                curpos = tell(handle) - recsize;

                result = LOCATE(stext);
                textcolor(WHITE);
                textbackground(RED);
                if (result == 0)
                {
                        gotoxy(left + 24,21);
                        cprintf("NO MORE MATCHING RECORDS");
                        gotoxy(left + 24,22);
                        cprintf("Press RETURN to Continue");
                        bioskey(0);
                        gotoxy(left,21);
                        cprintf("%70c",32);
                        gotoxy(left,22);
                        cprintf("%70c",32);
                        lseek(handle,curpos,SEEK_SET);
                        read(handle,&rec,recsize);
                        DISPDATA();
                }
                else
                {
                        lseek(handle,0 - recsize,SEEK_CUR);
                        read(handle,&rec,recsize);
                        DISPDATA();
                }
                textcolor(WHITE);
                textbackground(RED);
                gotoxy(left,21);
                cprintf("%70c",32);
                gotoxy(left,22);
                cprintf("                                                                      ");
                textcolor(BLACK);
                textbackground(GREEN);
        }

        void PRINT_MULTI()
        {
                data buffer;
                char destination[60];
                char text[5];
                int result;
                int ok;
                int ok2;
                int blanks;
                int total_lines;
                char *p;
                FILE *fp;
                textcolor(WHITE);
                textbackground(RED);
                gotoxy(left + 23,21);
                cprintf("Enter selection criteria");

                /* Clear existing rec details */
                memset(&rec,0,recsize);

                DISPDATA();
                GETDATA(0);

                textcolor(WHITE);
                textbackground(RED);
                gotoxy(left,21);
                cprintf("Enter report destination PRN");
                strcpy(destination,"PRN");
                gotoxy(left,22);
                cprintf("Enter Address length in lines 18");
                strcpy(text,"18");
                gotoxy(left + 25,21);
                INPUT(destination,40);
                gotoxy(left +30,22);
                INPUT(text,2);
                gotoxy(left,21);
                cprintf("%72c",32);
                gotoxy(left,22);
                cprintf("%72c",32);

                total_lines = atoi(text) - 6;
                if (total_lines < 0)
                        total_lines = 0;

                fp = fopen(destination,"w+");
                if (fp == NULL)
                {
                        gotoxy(left,21);
                        cprintf("Unable to print to %s",destination);
                        gotoxy(left,22);
                        cprintf("Press RETURN to Continue");
                        bioskey(0);
                        gotoxy(left,21);
                        cprintf("%78c",32);
                        gotoxy(left,22);
                        cprintf("                                        ");
                }

                /* Locate start of file */
                lseek(handle,0,SEEK_SET);

                do
                {
                        /* Read rec into memory */
                        result = read(handle,&buffer,recsize);
                        if (result > 0)
                        {
                                ok = 1;
                                /* Scan rec for matching data */
                                if (*rec.name)
```

```
				if (stricmp(buffer.name,rec.name))
					ok = 0;
			if (*rec.company)
				if (stricmp(buffer.company,rec.company))
					ok = 0;
			if (*rec.address)
				if (stricmp(buffer.address,rec.address))
					ok = 0;
			if (*rec.area)
				if (stricmp(buffer.area,rec.area))
					ok = 0;
			if (*rec.town)
				if (stricmp(buffer.town,rec.town))
					ok = 0;
			if (*rec.county)
				if (stricmp(buffer.county,rec.county))
					ok = 0;
			if (*rec.post)
				if (stricmp(buffer.post,rec.post))
				ok = 0;
			if (*rec.telephone)
				if (stricmp(buffer.telephone,rec.telephone))
					ok = 0;
			if (*rec.fax)
				if (stricmp(buffer.fax,rec.fax))
					ok = 0;
			if (ok)
			{
				blanks = total_lines;
				p = buffer.name;
				ok2 = 0;
				while(*p)
				{
					if (*p != 32)
					{
						ok2 = 1;
						break;
					}
					p++;
				}
				if (!ok2)
					blanks++;
				else
					fprintf(fp,"%s\n",buffer.name);
				p = buffer.company;
				ok2 = 0;
				while(*p)
				{
					if (*p != 32)
					{
						ok2 = 1;
						break;
					}
					p++;
				}
				if (!ok2)
					blanks++;
				else
```

```
		fprintf(fp,"%s\n",buffer.company);
	p = buffer.address;
	ok2 = 0;

	while(*p)
	{
		if (*p != 32)
		{
			ok2 = 1;
			break;
		}
		p++;
	}
	if (!ok2)
		blanks++;
	else
		fprintf(fp,"%s\n",buffer.address);
	p = buffer.area;
	ok2 = 0;
	while(*p)
	{
		if (*p != 32)
		{
			ok2 = 1;
			break;
		}
		p++;
	}
	if (!ok2)
		blanks++;
	else
		fprintf(fp,"%s\n",buffer.area);
	p = buffer.town;
	ok2 = 0;
	while(*p)
	{
		if (*p != 32)
		{
			ok2 = 1;
			break;
		}
		p++;
	}
	if (!ok2)
		blanks++;
	else
		fprintf(fp,"%s\n",buffer.town);
	p = buffer.county;
	ok2 = 0;

	while(*p)
	{
		if (*p != 32)
		{
			ok2 = 1;
			break;
		}
		p++;
```

```
                                                }
                                                if (!ok2)
                                                        blanks++;
                                                else
                                                        fprintf(fp,"%s\n",buffer.county);
                                                p = buffer.post;
                                                ok2 = 0;
                                                while(*p)
                                                {
                                                        if (*p != 32)
                                                        {
                                                                ok2 = 1;
                                                                break;
                                                        }
                                                        p++;
                                                }
                                                if (!ok2)
                                                        blanks++;
                                                else
                                                        fprintf(fp,"%s\n",buffer.post);
                                                while(blanks)
                                                {
                                                        fprintf(fp,"\n");
                                                        blanks--;
                                                }
                                        }
                                }
                        }
                        while(result > 0);
                        fclose(fp);
                        lseek(handle,0,SEEK_SET);
                        read(handle,&rec,recsize);
                        DISPDATA();
                }

                void EXPORT_MULTI()
                {
                        data buffer;
                        char destination[60];
                        int result;
                        int ok;
                        FILE *fp;

                        textcolor(WHITE);
                        textbackground(RED);
                        gotoxy(left + 23,21);
                        cprintf("Enter selection criteria");

                        /* Clear existing rec details */
                        memset(&rec,0,recsize);

                        DISPDATA();
                        GETDATA(0);

                        textcolor(WHITE);
                        textbackground(RED);
                        gotoxy(left,21);
                        cprintf("Enter export file address.txt");
```

```
strcpy(destination,"address.txt");
gotoxy(left + 18,21);
INPUT(destination,59);
gotoxy(left,21);
cprintf("%70c",32);

fp = fopen(destination,"w+");
if (fp == NULL)
{
        gotoxy(left,21);
        cprintf("Unable to print to %s",destination);
        gotoxy(left,22);
        cprintf("Press RETURN to Continue");
        bioskey(0);
        gotoxy(left,21);
        cprintf("%78c",32);
        gotoxy(left,22);
        cprintf("                                        ");
}
/* Locate start of file */
lseek(handle,0,SEEK_SET);

do
{
        /* Read rec into memory */
        result = read(handle,&buffer,recsize);
        if (result > 0)
        {
                ok = 1;
                /* Scan rec for matching data */
                if (*rec.name)
                        if (stricmp(buffer.name,rec.name))
                                ok = 0;
                if (*rec.company)
                        if (stricmp(buffer.company,rec.company))
                                ok = 0;
                if (*rec.address)
                        if (stricmp(buffer.address,rec.address))
                                ok = 0;
                if (*rec.area)
                        if (stricmp(buffer.area,rec.area))
                                ok = 0;
                if (*rec.town)
                        if (stricmp(buffer.town,rec.town))
                                ok = 0;
                if (*rec.county)
                        if (stricmp(buffer.county,rec.county))
                                ok = 0;
                if (*rec.post)
                        if (stricmp(buffer.post,rec.post))
                        ok = 0;
                if (*rec.telephone)
                        if (stricmp(buffer.telephone,rec.telephone))
                                ok = 0;
                if (*rec.fax)
                        if (stricmp(buffer.fax,rec.fax))
                                ok = 0;
                if (ok)
```

```
                        {
                                fprintf(fp,"\"%s\",",buffer.name);
                                fprintf(fp,"\"%s\",",buffer.company);
                                fprintf(fp,"\"%s\",",buffer.address);
                                fprintf(fp,"\"%s\",",buffer.area);
                                fprintf(fp,"\"%s\",",buffer.town);
                                fprintf(fp,"\"%s\",",buffer.county);
                                fprintf(fp,"\"%s\",",buffer.post);
                                fprintf(fp,"\"%s\",",buffer.telephone);
                                fprintf(fp,"\"%s\"\n",buffer.fax);

                        }
                }
        }

        while(result > 0);
        fclose(fp);
        lseek(handle,0,SEEK_SET);
        read(handle,&rec,recsize);
        DISPDATA();
}

void MENU()
{
        int option;
        long result;
        long end;
        int new;

        do
        {
                cursor(21,26);
                print("Select option (F2 - F10)");
                cursor(7,52);
                print("F2 Next record");
                cursor(8,52);
                print("F3 Previous record");
                cursor(9,52);
                print("F4 Amend record");
                cursor(10,52);
                print("F5 Add new record");
                cursor(11,52);
                print("F6 Search");
                cursor(12,52);
                print("F7 Continue search");
                cursor(13,52);
                print("F8 Print address labels");
                cursor(14,52);
                print("F9 Export records");
                cursor(15,52);
                print("F10 Exit");
                MOUSE_CURSOR(1);
                option = GETOPT();
                MOUSE_CURSOR(0);

                switch(option)
                {
                        case 0 : /* Next rec */
                        result = read(handle,&rec,recsize);
                        if (!result)
                        {
                                lseek(handle,0,SEEK_SET);
                                 result = read(handle,&rec,recsize);
                        }
                        DISPDATA();
                        break;

                case 1 : /* Previous rec */
                        result = lseek(handle,0 - recsize * 2,SEEK_CUR);
                        if (result <= -1)
                                lseek(handle,0 - recsize,SEEK_END);
                        result = read(handle,&rec,recsize);
                        DISPDATA();
                        break;

                case 3 : /* Add rec */
                        lseek(handle,0,SEEK_END);
                        memset(&rec,0,recsize);
                        DISPDATA();

                case 2 : /* Amend current rec */
                        new = 1;
                        if (*rec.name)
                                new = 0;
                        else
                        if (*rec.company)
                                new = 0;
                        else
                        if (*rec.address)
                                new = 0;
                        else
                        if (*rec.area)
                                new = 0;
                        else
                        if (*rec.town)
                                new = 0;
                        else
                        if (*rec.county)
                                new = 0;
                        else
                        if (*rec.post)
                                new = 0;
                        else
                        if (*rec.telephone)
                                new = 0;
                        else
                        if (*rec.fax)
                                new = 0;
                        result = tell(handle);
                        lseek(handle,0,SEEK_END);
                        end = tell(handle);

                        /* Back to original position */
                        lseek(handle,result,SEEK_SET);

                        /* If not at end of file, && !new rewind one rec */
```

```
                                if (result != end || ! new)
                                        result = lseek(handle,0 - recsize,SEEK_CUR);
                                result = tell(handle);
                                gotoxy(left + 22,21);
                                print(" Enter address details  ");
                                GETDATA(0);
                                if (*rec.name || *rec.company)
                                        result =  write(handle,&rec,recsize);
                                break;

                        case 4 : /* Search */
                                gotoxy(left + 22,21);
                                print("                                          ");
                                SEARCH();
                                break;

                        case 5 : /* Continue */
                                gotoxy(left + 22,21);
                                print("                                          ");
                                CONTINUE();
                                break;

                        case 6 : /* Print */
                                gotoxy(left + 22,21);
                                print("                                          ");
                                PRINT_MULTI();
                                break;

                        case 7 : /* Export */
                                gotoxy(left + 22,21);
                                print("                                          ");
                                EXPORT_MULTI();
                                break;

                        case 8 : /* Exit */
                                break;

                        default: /* Amend current rec */
                                new = 1;
                                if (*rec.name)
                                        new = 0;
                                else
                                if (*rec.company)
                                        new = 0;
                                else
                                if (*rec.address)
                                        new = 0;
                                else
                                if (*rec.area)
                                        new = 0;
                                else
                                if (*rec.town)
                                        new = 0;
                                else
                                if (*rec.county)
                                        new = 0;
                                else
                                if (*rec.post)
```

```
                                        new = 0;
                                else
                                if (*rec.telephone)
                                        new = 0;
                                else
                                if (*rec.fax)
                                        new = 0;
                                result = tell(handle);
                                lseek(handle,0,SEEK_END);
                                end = tell(handle);

                                /* Back to original position */
                                lseek(handle,result,SEEK_SET);

                                /* If not at end of file, && !new rewind one rec */
                                if (result != end || ! new)
                                        result = lseek(handle,0 - recsize,SEEK_CUR);
                                result = tell(handle);
                                gotoxy(left + 22,21);
                                print(" Enter address details  ");
                                GETDATA(option - 17);
                                if (*rec.name || *rec.company)
                                        result = write(handle,&rec,recsize);
                                option = -1;
                                break;

                }
        }

        while(option != 8);
}

void exec()
{
        gettext(1,1,80,25,scr);
        setvideo(3);
        textbackground(WHITE);
        textcolor(BLACK);
        clrscr();
        recsize = sizeof(data);

        OPENDATA();

        TRUESHADE(left,3,79,5);
        window(left - 2,2 ,78, 4);
        textcolor(YELLOW);
        textbackground(MAGENTA);
        clrscr();
        DBOX(left - 3, 1, 77, 3);
        gotoxy(3,2);
        print("Servile Software                PC ADDRESS BOOK 5.2                (c)
1994");

        TRUESHADE(left,8,left + 43,18);
        window(left - 2,7 , left + 42, 17);
        textcolor(BLACK);
        textbackground(GREEN);
        clrscr();
        DBOX(left - 3, 6, left + 41, 16);

        TRUESHADE(left + 48,8,79,18);
        window(left + 46, 7 , 78, 17);
        textbackground(BLUE);
        textcolor(YELLOW);
        clrscr();
        DBOX(left + 45,6,77,16);

        TRUESHADE(left ,21,79,24);
        window(left - 2, 20 , 78, 23);
        textbackground(RED);
        textcolor(WHITE);
        clrscr();
        DBOX(left - 3,19,77,22);

        window(1,1,80,25);
        textcolor(BLACK);
        textbackground(GREEN);
        DISPDATA();

        MENU();

        CLOSEDATA();
        puttext(1,1,80,25,scr);
        return;
}
```

INTERFACING C WITH CLIPPER

The Clipper programming language is a popular xBase environment for the PC. However, it lacks many of the facilities available to programmers of other languages, and it is quite slow compared to C. Because of this there are a large number of third party add-on libraries available for Clipper which provide the facilities lacked.

As a programmer you probably want to write your own library for Clipper, or perhaps individual functions to cater for circumstances which Clipper cannot handle, such as high resolution graphics.

Throughout this section, Clipper refers to the Summer '87 Clipper compiler, although initial tests show that the functions described here work perfectly well with the new Clipper 5 compiler also, we are not in a position to guarrantee success!

COMPILING AND LINKING

The Clipper extend functions allow user defined functions to be written in C, linked with and used by the Clipper application. The first problem a programmer must address when writing functions in C to link with a Clipper application is that of the C compiler's run time libraries.

If one is writing functions with Microsoft C, then most of the required run time library functions will be found in the Clipper.lib and Extend.lib libraries which are part of Clipper.

If, however, one is using a different C compiler, such as Borland's Turbo C then the run time library routines must be supplied on the link line.

All C functions must be compiled using the large memory model the following line is used with Microsoft C

```
cl /c /AL /Zl /Oalt /FPa /Gs <program.c>
```

and this compile line may be used with Turbo C

```
tcc -c -ml <program>
```

simply substitute <program> for the program name to be compiled.

Having compiled a C function it must be linked in with the application. If the C function was compiled with Microsoft C then the link line will look a little like this;

```
LINK /SE:500 /NOE program.obj cfunc.obj,,,Clipper Extend
```

If the C function was linked with another C compiler you will also need to link in the C run time libraries, for example to link in the Turbo C large memory mode library use the following link line;

```
LINK /SE:500 /NOE program.obj cfunc.obj,,,Clipper Extend cl
```

If one is using a number of separately compiled C functions it is a good idea to collect them in a library. If you are using Microsoft C then you can simply create the library by using Microsoft Lib.exe with the following command line;

```
LIB mylib +prog1 +prog2, NUL, NUL
```

This tells the librarian to add prog1.obj and prog2.obj to a library called mylib.lib, creating it if it does not exist. The NUL parameter is for supressing the listing file.

If you have been using another C compiler you should copy the C large memory model run time library before adding your functions to it for example;

```
 COPY C:\TURBOC\LIB\cl.lib mylib.lib
 LIB mylib +prog1 +prog2, NUL, NUL
```

Then when you link your Clipper application you will use a link line similar to;

```
LINK /SE:500 /NOE myprog,,,Clipper Extend Mylib
```

Often when linking C functions with Clipper applications link errors will occur such as those shown below;

```
Microsoft (R) Overlay Linker  Version 3.65
Copyright (C) Microsoft Corp 1983-1988.  All rights reserved.


LINK : error L2029: Unresolved externals:


FIWRQQ in file(s):
 M:SLIB.LIB(TEST)
FIDRQQ in file(s):
 M:SLIB.LIB(TEST)

There were 2 errors detected
```

Example Link Errors

The errors shown here are ‘Unresolved externals’, that is they are references to functions which are not found in any of the object modules or libraries specified on the link line. These occur because the C compilers often scatter functions and variables through a number of libraries. In tracking these functions down use may be made of the Microsoft librarian list file option. If you run Lib.Exe on the Turbo C ‘emu.lib’ library file and specify a listing file as follows;

```
LIB emu,emu.lst
```

The librarian will create an ascii file which contains the names of each object module contained in the specified library file, and the names of each function and public variable declared in each object module, as shown in this listing of Borland’s EMU.LIB library.

```
e086_Entry........EMU086          e086_Shortcut.....EMU086
e087_Entry........EMU087          e087_Shortcut.....EMU087
FIARQQ............EMUINIT         FICRQQ............EMUINIT
FIDRQQ............EMUINIT         FIERQQ............EMUINIT
FISRQQ............EMUINIT         FIWRQQ............EMUINIT
FJARQQ............EMUINIT         FJCRQQ............EMUINIT
FJSRQQ............EMUINIT         __EMURESET........EMUINIT


EMUINIT         Offset: 00000010H  Code and data size: 1a2H
FIARQQ          FICRQQ             FIDRQQ             FIERQQ
FISRQQ          FIWRQQ             FJARQQ             FJCRQQ
FJSRQQ          __EMURESET
```

```
EMU086              Offset: 00000470H  Code and data size: 2630H
  e086_Entry        e086_Shortcut

EMU087              Offset: 00003200H  Code and data size: 417H
  e087_Entry        e087_Shortcut
```

Receiving Parameters

Clipper provides six different functions for receiving parameters in a C function. These functions are;

Receive a string	char * _parc(int,[int])
Receive a Date string	char * _pards(int,[int])
Receive a logical	int _parl(int,[int])
Receive an integer	int _parni(int,[int])
Receive a long	long _parnl(int,[int])
Receive a double	double _parnd(int,[int])

To illustrate simple parameter receiving in a C function I offer the following simple C function which receives two numeric parameters from the calling Clipper program, and uses these two numeric parameters to set the size of the cursor.

```
#include <nandef.h>                     /* Clipper header files */
#include <extend.h>
#include <dos.h>                                /* Header file to define REGS */

CLIPPER s_curset()
{
        /* Demonstration function to set cursor shape */

        union REGS inreg,outreg;

        inreg.h.ah = 0x01;
        inreg.h.ch = _parni(1);    /* Get integer parameter 1 */
        inreg.h.cl = _parni(2);    /* Get integer parameter 2 */
        int86(0x10,&inreg,&outreg);
        _ret();                                 /* Return to Clipper */
}
```

Clipper provides four more functions for dealing with received parameters;

_parclen(int,[int]) which returns the length of a string including imbedded '\0's, _parcsiz(int[int]) which returns the length of a character string passed by reference from Clipper, _parinfa(int,[int]) which returns the type of a specified array element or the length of an array, and finally _parinfo(int) whic returns the type of a parameter.

The following example function uses _parinfa() to determine both the length of an array passed from a Clipper program, and the type of each element in the array. The function then returns to Clipper an integer representing the number of defined elements in the array.

```
#include <nandef.h>
#include <extend.h>

CLIPPER s_alen()
{
        int total;
        int n;
        int defined;
        int type;

        /* Return the number of defined elements in an array */
        /* From Clipper use defined = s_alen(arr) */

        total = _parinfa(1,0); /* Get declared number of elements in array */

        defined = 0;

        for (n = 1; n <= total; n++){
                type = _parinfa(1,n);   /* Get array parameter type */
                if (type)
                        defined++;
        }
        _retni(defined);                /* Return an integer to Clipper */
}
```

This function goes one step further to return the mean average of all numeric values in an array. Notice the use of _parnd() to retrieve the numeric values as doubles. You may find that because of the floating point arithmetic in this function that it will only work if compiled with Microsoft C.

```
#include <nandef.h>
#include <extend.h>

CLIPPER s_aave()
{
        int total;
        int defined;
        int n;
        int type;
        double sum;

        /* Return the mean average value of numbers in array */
        /* From Clipper use mean = s_aave(arr)


        total = _parinfa(1,0);                  /* Get declared number of elements */

        defined = 0;

        for (n = 1; n <= total; n++){           /* Determine number of defined */
                type = _parinfa(1,n);                   /* elements */
                if (type == 2)
                        defined++;
        }

        sum = 0;
```

```
		for (n = 1; n <= total; n++){
			type = _parinfa(1,n);
			if (type == 2)				/* Only sum numeric values */
				sum += _parnd(1,n);
		}
		_retnd(sum / defined);			/* Return a double to Clipper */
	}
```

Returning Values

The Clipper manual lists seven functions for returning from a function written in another language. These return functions for C are as follows;

character	_retc(char *)
date	_retds(char *)
logical	_retl(int)
numeric (int)	_retni(int)
numeric (long)	_retnl(long)
numeric (double)	_retnd(double)
nothing	_ret(void)

Omitted from the Clipper manual is the information that you may return different types of value back from a function! For example, you may wish to return a character string under normal circumstances, fine use _retc(). On error occurences however you can return a logical using _retl(). The Clipper program will assign the received value to the receiving variable in the correct manner.

The following simple C function returns a random number. Notice the use of integers which limits the range of the function to +-32767. For larger values you should use longs instead of integers.

```
	#include <nandef.h>
	#include <extend.h>
	#include <dos.h>

	CLIPPER s_random()
	{
		/* Returns a random number between 0 and param1 - 1 */
		/* From Clipper use x = s_random(param1) */

		int param1;
		int x;

		param1 = _parni(1);

		x = rand() % param1;
		_retni(x);
	}
```

This function receives a string from Clipper, and passes back an upper case copy of the string, leaving the original unchanged. The maximum length of the string which can be processed is determined by the size of target[], here set to 5000 characters.

```
        #include <nandef.h>
        #include <extend.h>

        CLIPPER s_upper()
        {
                /* Returns an upper case copy of string */
                /* From Clipper use ? s_upper("this is a string")

                char *p;
                char *q;
                char *string;
                char target[5000];
                int n;

                string = _parc(1);

                p = string;
                q = target;

                while(*string){
                        *q++ = toupper(*string);
                        string++;
                }
                *q = '\0';
                string = p;
                _retc(target);
        }
```

This function may be used to change the current DOS directory. If it is successful it returns .T. to the calling Clipper program, otherwise it returns .F.

```
        #include <nandef.h>
        #include <extend.h>
        #include <dos.h>

        CLIPPER s_chdir()
        {
                /* Attempts to change the current DOS directory */
                /* From Clipper use result = s_chdir(path) */

                union REGS inreg,outreg;
                struct SREGS segreg;

                char *path;
                int x;

                path = _parc(1);                /* Retrieve string from Clipper */

                inreg.h.ah = 0x3b;
                segreg.ds = FP_SEG(path);
                inreg.x.dx = FP_OFF(path);
                intdosx(&inreg,&outreg,&segreg);

                x = outreg.x.ax;

                if (x == 3)
                        _retl(0);   /* Return logical .F. back to Clipper */
                else
                        _retl(1);    /* Return logical .T. back to Clipper */
        }
```

Avoiding Unresolved Externals

As we have already seen, a common problem plaguing the programmer interfacing C functions with Clipper programs is Unresolved Externals.

The following example C function called s_print() will not link with Clipper.

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

CLIPPER s_print()
{
        char *x;

        x = _parc(1);

        printf("\nI received %s from Clipper.\n",x);

        _ret();
}
```

The linker gives you the following reply;

```
Microsoft (R) Overlay Linker  Version 3.65
Copyright (C) Microsoft Corp 1983-1988.  All rights reserved.

M:SLIB.LIB(IOERROR) : error L2025: __doserrno : symbol defined more than once
 pos: 16C6F Record type: 53C6

LINK : error L2029: Unresolved externals:



__RealCvtVector in file(s):
 M:SLIB.LIB(REALCVT)
_abort in file(s):
 M:SLIB.LIB(CVTFAK)

There were 3 errors detected
```

The error L2025 'symbol defined more than once' can in this case be ignored. However, the unresolved externals 'RealCvtVector' and 'abort' cannot be ignored. These two functions are referenced by the function printf() which has been included in the C function. The answer is to use as few of the compiler's run time library functions as possible, use ROM calls instead with INT86() and INTDOSX() etc.

Adding High Resolution Graphics To Clipper With C

The most annoying omission from Clipper, in my opinion, is the lack of high resolution graphics facilities. The following functions, written in Turbo C, provide high resolution graphics to Clipper.

First we require a means to change the video display mode to a high resolution graphics mode, and back to text mode. The IBM PC BIOS provides the means for this and can be called from C as follows;

```
/*                      Servile Software Library For Clipper                     */

#include <nandef.h>
#include <extend.h>
#include <dos.h>

CLIPPER s_smode()
{
        /* Set Video Mode */
        /* From Clipper use s_smode(mode) */

        union REGS inreg,outreg;

        inreg.h.al = _parni(1);
        inreg.h.ah = 0x00;
        int86 (0x10, &inreg, &outreg);


/*  1 40x25 colour text
        2 40x25 bw text
        3 80x25 colour text
        4 320x200 4 colour graphics
        5 320x200 4 colour graphics colour burst off
        6 640x200 2 colour graphics
        etc
*/
        _ret();
}
```

Having set the computer into graphics mode, how about setting pixels to a specified colour?

```
/*                      Servile Software Library For Clipper                     */

#include <nandef.h>
#include <extend.h>
#include <dos.h>

CLIPPER s_plot()
{

        union REGS inreg,outreg;

        /* Sets a pixel at the specified coordinates to the specified colour. */

        inreg.h.bh = 0x00;
        inreg.x.cx = _parni(1);
        inreg.x.dx = _parni(2);
        inreg.h.al = _parni(3);
```

```
                inreg.h.ah = 0x0C;
                int86(0x10, &inreg, &outreg);
        }
```

Line drawing and circles are handled by these two functions;

```
        /*              Servile Software Library For Clipper                    */

        #include <nandef.h>
        #include <extend.h>
        #include <dos.h>

        CLIPPER s_line()
        {

                union REGS inreg,outreg;

                /* Draws a straight line from (a,b) to (c,d) in colour col */

                int a;
                int b;
                int c;
                int d;
                int col;
                int u;
                int v;
                int d1x;
                int d1y;
                int d2x;
                int d2y;
                int m;
                int n;
                int s;
                int i;

                a = _parni(1);
                b = _parni(2);
                c = _parni(3);
                d = _parni(4);
                col = _parni(5);

                u = c - a;
                v = d - b;
                if (u == 0)
                {
                        d1x = 0;
                        m = 0;
                }
                else
                {
                        m = abs(u);
                        if (u < 0)
                                d1x = -1;
                        else
                                if (u > 0)
                                        d1x = 1;
                }
```

```
          if ( v == 0)
          {
                    d1y = 0;
                    n = 0;
          }
          else
          {
                    n = abs(v);
                    if (v < 0)
                              d1y = -1;
                    else
                              if (v > 0)
                                        d1y = 1;
          }
          if (m > n)
          {
                    d2x = d1x;
                    d2y = 0;
          }
          else
          {
                    d2x = 0;
                    d2y = d1y;
                    m = n;
                    n = abs(u);
          }
          s = (m / 2);

          inreg.h.al = (unsigned char)col;
          inreg.h.bh = 0x00;
          inreg.h.ah = 0x0C;
          for (i = 0; i <= m; i++)
          {
                    inreg.x.cx = (unsigned int)(a);
                    inreg.x.dx = (unsigned int)(b);
                    int86(0x10, &inreg, &outreg);
                    s += n;
                    if (s >= m)
                    {
                              s -= m;
                              a += d1x;
                              b += d1y;
                    }
                    else
                    {
                              a += d2x;
                              b += d2y;
                    }
          }
}
```

This circle drawing function uses in-line assembler to speed up the drawing process. It can easily be replaced with inreg and outreg parameters as in the other functions, or the other functions can be changed to in-line assembler. Both methods are shown to illustrate different ways of achieving the same result.

```
/*                    Servile Software Library For Clipper                           */

#include <nandef.h>
#include <extend.h>
#include <dos.h>


void plot(int x, int y, unsigned char colour)
{
        asm mov al , colour;
        asm mov bh , 00;
        asm mov cx , x;
        asm mov dx , y;
        asm mov ah , 0Ch;
        asm int 10h;
}

int getmode()
{
        /* Returns current video mode  and number of columns in ncols */

        asm mov ah , 0Fh;
        asm int 10h;
        return(_AL);
}


CLIPPER s_circle()
{
        int x_centre;
        int y_centre;
        int radius;
        int colour;
        int x,y,delta;
        int startx,endx,x1,starty,endy,y1;
        int asp_ratio;

        x_centre = _parni(1);
        y_centre = _parni(2);
        radius = _parni(3);
        colour = _parni(4);




        if (getmode() == 6)
                asp_ratio = 22;
        else
                asp_ratio = 13;

        y = radius;
        delta = 3 - 2 * radius;

        for(x = 0; x < y; )
        {
                starty = y * asp_ratio / 10;
                endy = (y + 1) * asp_ratio / 10;
                startx = x * asp_ratio / 10;
```

```
                        endx = (x + 1) * asp_ratio / 10;

                        for(x1 = startx; x1 < endx; ++x1)
                        {
                                plot(x1+x_centre,y+y_centre,colour);
                                plot(x1+x_centre,y_centre - y,colour);
                                plot(x_centre - x1,y_centre - y,colour);
                                plot(x_centre - x1,y + y_centre,colour);
                        }

                        for(y1 = starty; y1 < endy; ++y1)
                        {
                                plot(y1+x_centre,x+y_centre,colour);
                                plot(y1+x_centre,y_centre - x,colour);
                                plot(x_centre - y1,y_centre - x,colour);
                                plot(x_centre - y1,x + y_centre,colour);
                        }

                        if (delta < 0)
                                delta += 4 * x + 6;
                        else
                        {
                                delta += 4*(x-y)+10;
                                y--;
                        }
                        x++;
                }



                if(y)
                {
                        starty = y * asp_ratio / 10;
                        endy = (y + 1) * asp_ratio / 10;
                        startx = x * asp_ratio / 10;
                        endx = (x + 1) * asp_ratio / 10;
                        for(x1 = startx; x1 < endx; ++x1)
                        {
                                plot(x1+x_centre,y+y_centre,colour);
                                plot(x1+x_centre,y_centre - y,colour);
                                plot(x_centre - x1,y_centre - y,colour);
                                plot(x_centre - x1,y + y_centre,colour);
                        }

                        for(y1 = starty; y1 < endy; ++y1)
                        {
                                plot(y1+x_centre,x+y_centre,colour);
                                plot(y1+x_centre,y_centre - x,colour);
                                plot(x_centre - y1,y_centre - x,colour);
                                plot(x_centre - y1,x + y_centre,colour);
                        }
                }
        }
```

The Clipper facilities for displaying text on the screen, @....SAY and ? do not work when the monitor is in graphics mode. You then need the following function to allow text to be displayed in a graphics mode;

```
/*                    Servile Software Library For Clipper                    */

#include <nandef.h>
#include <extend.h>
#include <dos.h>

int sgetmode(int *ncols)
{
        /* Returns current video mode  and number of columns in ncols */

        union REGS inreg,outreg;

        inreg.h.ah = 0x0F;
        int86(0x10, &inreg, &outreg);
        *ncols = outreg.h.ah;
        return(outreg.h.al);
}

void at(int row, int col)
{
        asm mov bh , 0;
        asm mov dh , row;
        asm mov dl , col;
        asm mov ah , 02h;
        asm int 10h;
}



CLIPPER s_say()
{
        char *output;
        int p = 0;
        unsigned char page;
        unsigned char text;
        int n;
        int r;
        int c;
        int attribute;

        output = _parc(1);
        r = _parni(2);
        c = _parni(3);
        attribute = _parni(4);

        asm mov ah , 0Fh;
        asm int 10h;
        asm mov page, bh;

        sgetmode(&n);

        at(r,c);

        while (output[p])
        {
                text = output[p++];
```

```
                        asm mov bh , page;
                        asm mov bl , attribute;
                        asm mov cx , 01h;
                        asm mov ah , 09h;
                        asm mov al , text;
                        asm int 10h;
                        c++;
                        if (c < (n-1))
                                at( r, c);
                        else
                        {
                                c = 0;
                                at(++r,0);
                        }
                }
        }
```

When drawing graphs, it is often required to fill in areas of the graph in different patterns. This is a graphics function to fill boundered shapes with a specified hatching pattern providing a means to achieve more usable graphs;

```
        /*              Servile Software Library For Clipper                    */

        #include <nandef.h>
        #include <extend.h>
        #include <dos.h>

        int pixset(int x, int y)
        {
                /* Returns the colour of the specified pixel */

                asm mov cx ,x;
                asm mov dx ,y;
                asm mov ah ,0Dh;
                asm int 10h;
                return(_AL);
        }

        CLIPPER s_fill()
        {
                /* Fill a boundered shape using a hatch pattern */

                int mode;
                int xa;
                int ya;
                int bn;
                int byn;
                int x;
                int y;
                int col;
                int pattern;
                int maxx;
                int maxy;
                int hatch[10][8] = { 255,255,255,255,255,255,255,255,
                                    128,64,32,16,8,4,2,1,
                                    1,2,4,8,16,32,64,128,
                                    1,2,4,8,8,4,2,1,
                                    238,238,238,238,238,238,238,238,
                                    170,85,170,85,170,85,170,85,
                                    192,96,48,24,12,6,3,1,
                                    62,62,62,0,227,227,227,0,
                                    129,66,36,24,24,36,66,129,
                                    146,36,146,36,146,36,146,36};

        /* Patterns for fill, each integer describes a row of dots */




        x = _parni(1);
        y = _parni(2);
        col = _parni(3);
        pattern = _parni(4);

        mode = getmode();

        switch(mode)
        {
                case 0:
                case 1:
                case 2:
                case 3: break;
                case 4:
                case 9:
                case 13:
                case 19:
                case 5: maxx = 320;
                        maxy = 200;
                        break;
                case 14:
                case 10:
                case 6: maxx = 640;
                        maxy = 200;
                        break;
                case 7: maxx = 720;
                        maxy = 400;
                        break;
                case 8: maxx = 160;
                        maxy = 200;
                        break;
                case 15:
                case 16: maxx = 640;
                         maxy = 350;
                         break;
                case 17:
                case 18: maxx = 640;
                         maxy = 480;
                         break;

        }

        xa = x;
        ya = y;  /* Save Origin */

        if(pixset(x,y))
                return;

        bn = 1;
        byn = 0;




        do
        {
                if (hatch[pattern][byn] != 0)
                {  /* If blank ignore */
                        do
                        {
                                if ((bn & hatch[pattern][byn]) == bn)
                                {
                                        asm mov al , col;
                                        asm mov bh , 00;
                                        asm mov cx , x;
                                        asm mov dx , y;
                                        asm mov ah , 0Ch;
                                        asm int 10h;
                                }
                                x--;
                                bn <<= 1;
                                if (bn > 128)
                                        bn = 1;
                        }
                        while(!pixset(x,y) && (x > -1));

                        x = xa + 1;
                        bn = 128;

                        do
                        {
                                if ((bn & hatch[pattern][byn]) == bn)
                                {
                                        asm mov al , col;
                                        asm mov bh , 00;
                                        asm mov cx , x;
                                        asm mov dx , y;
                                        asm mov ah , 0Ch;
                                        asm int 10h;
                                }
                                x++;
                                bn >>=1;
                                if (bn <1)
                                        bn = 128;
                        }
                        while((!pixset(x,y)) && (x <= maxx));
                }
                x = xa;
                y--;
                bn = 1;
                byn++;
                if (byn > 7)
                        byn = 0;




        }
        while(!pixset(x,y) && ( y > -1));

        /* Now travel downwards */

        y = ya + 1;

        byn = 7;
        bn = 1;
        do
        {
                /* Travel left */
                if (hatch[pattern][byn] !=0)
                {
                        do
                        {
                                if ((bn & hatch[pattern][byn]) == bn)
                                {
                                        asm mov al , col;
                                        asm mov bh , 00;
                                        asm mov cx , x;
                                        asm mov dx , y;
                                        asm mov ah , 0Ch;
                                        asm int 10h;
                                }
                                x--;
                                bn <<= 1;
                                if (bn > 128)
                                        bn = 1;
                        }
                        while(!pixset(x,y) && (x > -1));

                        /* Back to x origin */
                        x = xa + 1 ;
                        bn = 128;

                        /* Travel right */
                        do
                        {
                                if ((bn & hatch[pattern][byn]) == bn)
                                {
                                        asm mov al , col;
                                        asm mov bh , 00;
                                        asm mov cx , x;
                                        asm mov dx , y;
                                        asm mov ah , 0Ch;
                                        asm int 10h;
                                }
                                x++;
                                bn >>=1;
                        if (bn <1)
                                bn = 128;
                }
                while((!pixset(x,y)) && (x <= maxx));
            }
            x = xa;
            bn = 1;
            y++;
            byn--;
            if (byn < 0)
                byn = 7;
        }
        while((!pixset(x,y)) && (y <= maxy));
    }
```

SPELL - AN EXAMPLE PROGRAM

It has been said that example programs provide a good way of learning a new computer language. On that basis the following simple program is offered as an example of making use of the dynamic memory allocation provided by DOS.

This is a spell checker for ASCII (text) documents, written in, and making use of Borland's Turbo C text graphics facilities for displaying windows of text.

```
/* Spell checker for ascii documents */
/* Compile with -mc (compact memory model) and unsigned characters */


#include <stdio.h>
#include <conio.h>
#include <fcntl.h>
#include <io.h>
#include <dos.h>
#include <string.h>
#include <alloc.h>
#include <ctype.h>
#include <stdlib.h>
#include <bios.h>
#include <dir.h>
#include <stat.h>

#define  ON              0x06
#define  OFF             0x20
#define  MAXIMUM         15000
#define  WORDLEN         20
#define  LEFTMARGIN  1

union REGS inreg,outreg;

char *dicname;
char *dic[MAXIMUM];                /* Array of text lines */
char word[31];
char comp[31];
char fname[160];
int lastelem;
char changed;
char *ignore[100];
int lastign;
int insert;
int n;
int bp;
int mp;
int tp;
int result;




char *text;
char *textsav;

void AT(int, int);
void BANNER(void);
int COMPARE(void);
void CORRECT(void);
void FATAL(char *);
void FILERR(char *);
void GETDIC(void);
void IGNORE(void);
void INSERT(void);
int MATCHSTR(char *, char *);
void SPELL(void);
void UPDATE(void);

void CURSOR(char status)
{
	/* Toggle cursor display on and off */

	union REGS inreg,outreg;

	inreg.h.ah = 1;
	inreg.h.ch = (unsigned char)status;
	inreg.h.cl = 7;
	int86(0x10,&inreg,&outreg);
}

void DISPLAY(char *text)
{
	/* Display 'text' expanding tabs and newline characters */

	while(*text)
	{
		switch(*text)
		{
			case '\n':  cputs("\r\n");
						break;
			case '\t':  cputs("                ");
						break;
			default: putch(*text);
		}
		text++;
	}
}




void GETDIC()
{
	/* Read dictionary into memory */

	FILE *fp;
	char *p;
	int poscr;
	int handle;

	window(1,22,80,24);
	clrscr();
	gotoxy(28,2);
	cprintf("Reading Dictionary....");
```

```
                changed = 0;
                lastelem = 0;

                dicname = searchpath("spell.dic");
                handle = open(dicname,O_RDWR);
                if (handle < 0)
                        FILERR("spell.dic");

                fp = fdopen(handle,"r");
                if (fp == NULL)
                        FILERR("spell.dic");

                do
                {
                        dic[lastelem] = calloc(WORDLEN,1);
                        if (dic[lastelem])
                        {
                                p = fgets(dic[lastelem],79,fp);
                                /* Remove carriage return from end of text line */
                                poscr = (int)strlen(dic[lastelem]) - 1;
                                if (dic[lastelem][poscr] == '\n')
                                        dic[lastelem][poscr] = 0;
                        }
                        else
                                FATAL("Unable To Allocate Memory");
                }
                while((p != NULL) && (lastelem++ < MAXIMUM));

                lastelem--;

                fclose(fp);
        }



        void UPDATE()
        {
                FILE *fp;
                int n;

                if (changed)
                {
                        window(1,22,80,24);
                        clrscr();
                        gotoxy(27,2);
                        cprintf("Updating Dictionary....");

                        fp = fopen(dicname,"w+");
                        if (fp == NULL)
                                FILERR("spell.dic");

                        for(n = 0; n <= lastelem; n++)
                                fprintf(fp,"%s\n",dic[n]);

                        fclose(fp);
                }
```

```
    }

    void IGNORE()
    {
        /* Add a word to the ignore table */

        if (lastign < 100)
        {
            ignore[lastign] = calloc(strlen(word) + 1,1);
            if (ignore[lastign])
                strcpy(ignore[lastign++],comp);
            else
            {
                clrscr();
                cprintf("No available memory for new words!\r\nPress A key....");
                bioskey(0);
            }
         }
         else
         {
            clrscr();
            cprintf("No available memory for new words!\r\nPress A key....");
            bioskey(0);
         }
    }



    void FATAL(char *text)
    {
        /* Fatal error drop out */

        textcolor(LIGHTGRAY);
        textbackground(BLACK);
        window(1,1,80,25);
        clrscr();
        printf("SERVILE SOFTWARE\n\nSPELL V1.7\nFATAL ERROR: %s\n\n",text);
        CURSOR(ON);
        exit(0);
    }

    void FILERR(char *fname)
    {
        char text[60];

        strcpy(text,"Unable To Access: ");
        strcat(text,fname);
        FATAL(text);
    }

    int COMPARE()
    {
        char **p;

        /* Check Ignore table */
        for(p = ignore; p <= &ignore[lastign]; p++)
            if (strcmp(comp,*p) == 0)
```

```
                        return(1);

                /* Binary search of dictionary file */
                bp = 0;
                tp = lastelem;
                mp = (tp + bp) / 2;




                while((result = strcmp(dic[mp],comp)) != 0)
                {
                        if (mp >= tp)
                        {
                                /* Not found! */
                                insert = mp;
                                if (result > 0)
                                        insert--;
                                return(0);
                        }
                        if (result < 0)
                                bp = mp + 1;
                        else
                                tp = mp - 1;

                        mp = (bp + tp) / 2;
                }
                return(1);
        }

        void INSERT()
        {
                int n;

                changed = 1;
                lastelem++;
                n = lastelem;

                dic[n] = calloc(WORDLEN,1);

                if (dic[n] == NULL)
                {
                        clrscr();
                        cprintf("No available memory for new words!\r\nPress A key....");
                        bioskey(0);
                        free(dic[n]);
                        lastelem--;
                        return;
                }

                while(n > (insert + 1))
                {
                        strcpy(dic[n],dic[n-1]);
                        n--;
                };

                strcpy(dic[insert + 1],comp);
        }
```

```
void SPELL()
{
        FILE *target;
        FILE *source;
        char *p;
        char *x;
        char temp[256];
        char dat1[1250];
        char dat2[1250];
        int c;
        int m;
        int found;
        int curpos;
        int key;
        int row;
        int col;
        int srow;
        int scol;

        window(1,1,80,20);
        textcolor(BLACK);
        textbackground(WHITE);

        /* Open temporary file to take spell checked copy */
        target = fopen("spell.$$$","w+");

        source = fopen(fname,"r");

        if (source == NULL)
                FILERR(fname);

        lastign = 0;

        do
        {
                clrscr();

                text = dat1;

                p = text;

                textsav = dat2;

                strcpy(text,"");

                /* Display read text */
                row = wherey();
                col = wherex();



                for(m = 0; m < 15; m++)
                {
```

```
                x = fgets(temp,200,source);
                if (x)
                {
                        strcat(text,temp);
                        DISPLAY(temp);
                }
                if (wherey() > 18)
                        break;
        }

        /* return cursor to start position */
        gotoxy(col,row);

        do
        {
                memset(word,32,30);
                curpos = 0;
                do
                {
                        c = *text++;
                        if ((isalpha(c)) || (c == '-') && (curpos != 0))
                                word[curpos++] = c;
                }
                while(((isalpha(c)) || (c == '-') && (curpos != 0))
                          && (curpos < 30));
                word[curpos] = 0;
                strcpy(comp,word);
                strupr(comp);

                if (*comp != 0)
                {
                        found = COMPARE();
                        if (!found){
                                textbackground(RED);
                                textcolor(WHITE);
                        }
                }
                else
                        found = 1;

                srow = wherey();
                scol = wherex();

                cputs(word);
                textbackground(WHITE);
                textcolor(BLACK);




                switch(c)
                {
                        case '\n': cputs("\r\n");
                                  break;
                        case '\t': cputs("          ");
                                  break;
                        default: putch(c);
                }

                row = wherey();
                col = wherex();

                if (!found)
                {
                        window(1,22,80,24);
                        clrscr();
                        cputs("Unknown word ");
                        textcolor(BLUE);
                        cprintf("%s ",word);
                        textcolor(BLACK);
                        cputs("[A]dd  [I]gnore  [C]orrect  [S]kip");
                        do
                        {
                                key = toupper(getch());
                                if (key == 27)
                                        key = 'Q';
                        }
                        while(strchr("AICSQ",key) == NULL);

                        switch(key)
                        {
                                case 'A':INSERT();
                                                break;

                                case 'C':CORRECT();
                                                break;

                                case 'I':IGNORE();
                                                break;
                        }



                        if (key == 'C')
                        {
                                clrscr();
                                gotoxy(1,1);
                                strcpy(textsav,--text);
                                /* Delete old word */
                                text -= strlen(comp);
                                *text = 0;
                                /* Insert new word */
                                strcat(text,word);
                                /* Append remainder of text */
                                strcat(text,textsav);
                                text += strlen(word);
                                text++;
                                /* Length of text may have changed ! */
                                if (strlen(word) < strlen(comp))
                                        col -= (strlen(comp) - strlen(word));
                                window(1,1,80,20);
                                clrscr();
                                DISPLAY(p);
                        }
```

```
                                        else
                                        {
                                                clrscr();
                                                gotoxy(29,2);
                                                cputs("Checking Spelling....");
                                                window(1,1,80,20);
                                                gotoxy(scol,srow);
                                                cputs(word);
                                        }
                                        window(1,1,80,20);
                                        gotoxy(col,row);
                                }
                        }
                        while((*text) && (key != 'Q'));
                        fprintf(target,"%s",p);
                }
                while((x != NULL) && (key != 'Q'));

                window(1,22,80,24);
                clrscr();
                gotoxy(27,2);
                cprintf("Writing Updated File....");




                do
                {
                        p = fgets(temp,200,source);
                        if (p)
                                fprintf(target,"%s",temp);
                }
                while(p);

                fclose(target);
                fclose(source);

                /* Now transfer spell.$$$ to fname */
                unlink(fname);
                rename("SPELL.$$$",fname);
        }

        void CORRECT()
        {
                /* Locate a good match and return word */

                char text[51];
                int m;
                int n;
                int key;

                window(1,22,80,24);
                clrscr();
                gotoxy(25,2);
                cprintf("Searching For Alternatives....");

                /* Remove any pending key strokes from keyboard buffer */
                while(kbhit())
                        getch();

                for(n = 0; n <= lastelem; n++)
                {
                        if (MATCHSTR(dic[n],comp))
                        {
                                strcpy(text,dic[n]);
                                if (strlen(word) <= strlen(text))
                                {
                                        for (m = 0; m < strlen(word); m++)
                                        {
                                                if (isupper(word[m]))
                                                        text[m] = toupper(text[m]);
                                                else
                                                        text[m] = tolower(text[m]);
                                        }




                                        for(m = strlen(word); m < strlen(text); m++)
                                                if (isupper(word[strlen(word)]))
                                                        text[m] = toupper(text[m]);
                                                else
                                                        text[m] = tolower(text[m]);
                                }
                                else
                                {
                                        for (m = 0; m < strlen(text); m++)
                                        {
                                                if (isupper(word[m]))
                                                        text[m] = toupper(text[m]);
                                                else
                                                        text[m] = tolower(text[m]);
                                        }
                                }
                                clrscr();
                                cprintf("Replace ");
                                textcolor(BLUE);
                                cprintf("%s ",word);
                                textcolor(BLACK);
                                cprintf("With ");
                                textcolor(BLUE);
                                cprintf("%s",text);
                                textcolor(BLACK);
                                cprintf(" Yes No Continue");
                                do
                                {
                                        key = toupper(getch());
                                }
                                while(strchr("YNC",key) == NULL);
                                if (key == 'Y')
                                {
                                        strcpy(word,text);
                                        return;
                                }
                                clrscr();
                                gotoxy(25,2);
```

```
                    cprintf("Searching For Alternatives....");

                    /* Remove any pending key strokes from keyboard buffer */
                    while(kbhit())
                         getch();

                    if (key == 'C')
                         return;
               }
          }
          clrscr();
          gotoxy(23,2);




          cprintf("NO ALTERNATIVES FOUND! (Press a key)");
          bioskey(0);
          return;
     }


     int MATCHSTR(char *src, char *tgt)
     {
          /* Compare two words and return non zero if they are similar */

          int match;
          int result;
          int strsrc;
          int strtgt;
          int longest;

          strtgt = strlen(strupr(tgt));
          strsrc = strlen(strupr(src));

          longest = max(strtgt,strsrc);

          match = 0;

          if(strtgt > strsrc)
          {
               for(; *src ; match += (*src++ == *tgt++))
                    ;
          }
          else
          {
               for(; *tgt ; match += (*src++ == *tgt++))
                    ;
          }

          result = (match * 100 / longest);

          /* result holds percentage similarity */

          if (result > 50)
               return(1);
          return(0);
     }
```

```
void AT(int row, int col)
{
        /* Position the text cursor */
        inreg.h.bh = 0;
        inreg.h.dh = row;
        inreg.h.dl = col;
        inreg.h.ah = 0x02;
        int86 (0x10, &inreg, &outreg);
}

void WRTCHA (unsigned char ch, unsigned char attrib,  int num)
{
        /* Display a character num times in colour attrib */
        /* via the BIOS */

        inreg.h.al = ch;
        inreg.h.bh = 0;
        inreg.h.bl = attrib;
        inreg.x.cx = num;
        inreg.h.ah = 0x09;
        int86 (0x10, &inreg, &outreg);
}

void SHADE_BLOCK(int left,int top,int right,int bottom)
{
        int c;

        AT(bottom,right);
        WRTCHA(223,56,1);
        AT(top,right);
        WRTCHA('ß',7,1);
        for (c = top+1; c < bottom; c++)
        {
                AT(c,right);
                WRTCHA(' ',7,1);
        }
        AT(bottom,left+1);
        WRTCHA('Ü',7,right-left);
}




void BOX(int l, int t, int r, int b)
{
        /* Draws a single line box around a described area */

        int n;
        char top[81];
        char bottom[81];
        char tolc[5];
        char torc[5];
        char bolc[5];
        char borc[5];
        char hoor[5];

        sprintf(tolc,"%c",218);
```

```
		sprintf(bolc,"%c",192);
		sprintf(hoor,"%c",196);
		sprintf(torc,"%c",191);
		sprintf(borc,"%c",217);

			strcpy(top,tolc);
		strcpy(bottom,bolc);
		for(n = l + 1; n < r; n++)
		{
			strcat(top,hoor);
			strcat(bottom,hoor);
		}
		strcat(top,torc);
		strcat(bottom,borc);

		window(1,1,80,25);
			gotoxy(l,t);
		cputs(top);
		for (n = t + 1; n < b; n++)
		{
			gotoxy(l,n);
			putch(179);
			gotoxy(r,n);
			putch(179);
		}
		gotoxy(l,b);
		cputs(bottom);
	}



	void BANNER()
	{
		window (2,2,78,4);
		textcolor(BLACK);
		textbackground(GREEN);
		clrscr();
		SHADE_BLOCK(1,1,78,4);
		BOX(2,2,78,4);
		gotoxy(4,3);
		cprintf("Servile Software                    SPELL CHECKER V1.7
				(c)1992");
	}


	void main(int argc, char *argv[])
	{
		char *p;
		char tmp_name[160];
		char tmp_fname[160];

		if (argc != 2)
		{
			puts("\nERROR: Usage is SPELL document");
			exit(1);
		}
		else
```

```
                        strcpy(fname,argv[1]);

                CURSOR(OFF);

                GETDIC();

                window(1,22,80,24);
                clrscr();
                gotoxy(28,2);
                cprintf("Making Backup File....");

                strcpy(tmp_fname,argv[1]);

                /* Remove extension from tmp_fname */
                p = strchr(tmp_fname,'.');
                if(p)
                        *p = 0;

                /* Create backup file name using DOS */
                sprintf(tmp_name,"copy %s %s.!s! > NUL",argv[1],tmp_fname);

                system(tmp_name);

                window(1,1,80,25);




                textcolor(WHITE);
                textbackground(BLACK);
                clrscr();
                gotoxy(29,2);
                cprintf("Checking Spelling....");

                SPELL();

                UPDATE();
                window(1,1,80,25);
                textcolor(LIGHTGRAY);
                textbackground(BLACK);
                clrscr();
                CURSOR(ON);
        }
```

APPENDIX A - USING LINK

General Syntax:

```
LINK [options] obj[,[exe][,[map][,[lib]]]][;]
```

'obj' is a list of object files to be linked. Each obj file name must be separated by a + or a space. If you do not specify an extension, LINK will assume .OBJ. 'exe' allows you to specify a name for the executable file. If this file name is ommited, LINK will use the first obj file name and suffix it with .EXE. 'map' is an optional map file name. If you specify the name 'NUL', no map file is produced. 'lib' is a list of library files to link. LINK searches each library file and only links in modules which are referenced.

eg:

```
LINK filea+fileb,myfile,NUL;
```

Links .obj files 'filea.obj' and 'fileb.obj' into .exe file 'myfile.exe' with no map file produced. The ; at the end of the line tells LINK that there are no more parameters.

Using Overlays

Overlay .obj modules are specified by encasing the .obj name in parenthesis in the link line.

eg:

```
LINK filea + (fileb) + (filec),myfile,NUL;
```

Will link filea.obj fileb.obj and filec.obj with modules fileb.obj and filec.obj as overlay code.

Overlay modules must use FAR call/return instructions.

Linker Options

All LINK options commence with a forward slash '/'. Options which accept a number can accept a decimal number or a hex number prefixed 0X. eg: 0X10 is interpreted as 10h, decimal 16.

Pause during Linking (/PAU)

Tells LINK to wait before writing the .exe file to disk. LINK displays a message and waits for you to press enter.

Display Linker Process Information (/I)

Tells LINK to display information about the link process.

Pack Executable File (/E)

Tells LINK to remove sequences of repeated bytes and to optimise the load-time relocation table before creating the executable file. Symbolic debug information is stripped out of the file.

List Public Symbols (/M)

Tells LINK to create a list of all public symbols defined in the object files in the MAP file.

Include Line Numbers In Map File (/LI)

Tells LINK to include line numbers and associated addresses of the source program in the MAP file.

Preserve Case Sensitivity (/NOI)

By default LINK treats uppercase and lowercase letters as the same. This option tells LINK that they are different.

Ignore Default Libraries (/NOD)

Tells LINK not to search any library specified in the object files to resolve external references.

Controlling Stack Size (/ST:n)

Specifies the size of the stack segment where 'n' is the number of bytes.

Setting Maximum Allocation Space (/CP:n)

Tells LINK to write the parameter 'n' into the exe file header. When the exe file is executed by DOS, 'n' 16 byte paragraphs of memory are reserved. If 'n' is less than the minimum required, it will be set to the minimum. This option is ESSENTIAL to free memory from the program. C programs free memory automatically on start-up, assembly language programs which want to use dynamic memory allocation must be linked with this option set to a minimum.

Setting Maximum Number Of Segments (/SE:n)

Tells LINK how many segments a program is allowed to have. The default is 128 but 'n' can be any number between 1 and 3072.

Setting Overlay Interrupt (/O:n)

Tells LINK which interrupt number will be used for passing control to overlays. The default is 63. Valid values for 'n' are 0 through 255.

Ordering Segments (/DO)

Tells LINK to use DOS segment ordering. This option is also enabled by the MASM directive .DOSSEG.

Controlling Data Loading (/DS)

By default LINK loads all data starting at the low end of the data segment. At run time the DS register is set to the lowest possible address to allow the entire data segment to be used. This option tells LINK to load all data starting at the high end of the data segment.

Control Exe File Loading (/HI)

Tells LINK to place the exe file as high as possible in memory.

Prepare for Debugging (/CO)

Tells LINK to include symbolic debug information for use by codeview.

Optimising Far Calls (/F)

Tells LINK to translate FAR calls to NEAR calls where possible. This results in faster code.

Disabling Far Call Optimisation (/NOF)

Tells LINK not to translate FAR calls. This option is specified by default.

Packing Contiguous Segments (/PAC:n)

Tells LINK to group together neighbouring code segments, providing more oportunities for FAR call translation. 'n' specifies the maximum size of a segment. By default 'n' is 65530. This option is only relevant to obj files created using FAR calls.

Disabling Segment Packing (/NOP)

Disables segment packing. This option is specified by default.

Using Response Files

Linker options and file names may be specified in a response file. Each file list starting on a new line instead of being separated by a comma.

eg:

```
filea.obj fileb.obj
myfile.exe
NUL
liba.lib libb.lib
```

A response file is specified to LINK by prefixing the response file name with '@'.
eg:

```
LINK @response
```