

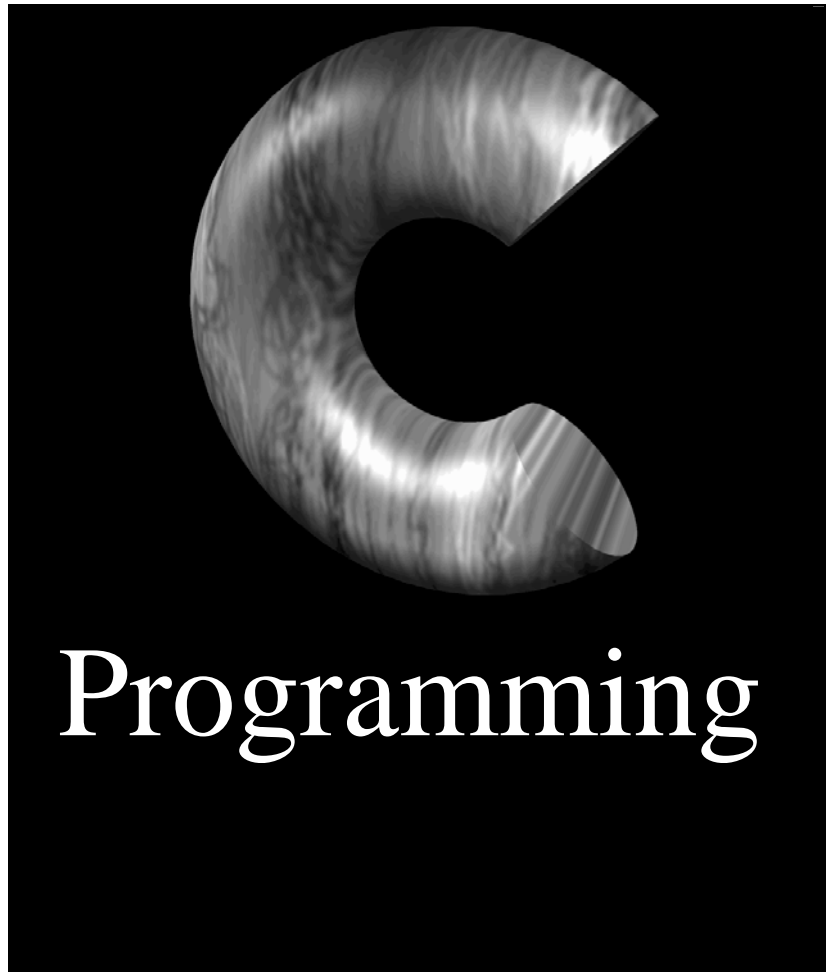


Queen Mary  
University of London

Computing  
Services

© Computing Services  
Queen Mary  
University of London

Permission to use material in  
this document for any  
purpose other than personal  
use should be obtained from  
[d.lexton@qmul.ac.uk](mailto:d.lexton@qmul.ac.uk)



Brian Littlechild  
Computing Services  
March 13, 1997



## Contents

Introduction .....	4
History of C.....	4
What is included .....	4
What is not included .....	4
Example program.....	5
Compiler and executing a program .....	5
about this booklet.....	6
Some C basics.....	7
What's in a C program .....	7
Lets get started .....	8
My First program .....	8
Exercise 1.....	8
Simple Arhythmic .....	9
Exercise 2a, 2b.....	9
Variable Types .....	9
Getting some input.....	10
Exercise 3a, 3b.....	10
I/O Field types.....	11
Loops .....	12
Exercise 4a, 4b.....	12
Relational Operators .....	12
Special characters .....	13
Exercise4c.....	13
Quick Test.....	14
your turn to write a program .....	15
Exercise 5.....	15
Decisions .....	16
Exercise 6.....	16
Function.....	17
Exercise 7.....	17
Arrays & Strings .....	18
Exercise 8.....	18
Strings Functions.....	19
Files (input and output).....	20
Exercise 9a, 9b.....	20
File Modes .....	21
Did we win the lottery.....	22
Exercise 10.....	22
Where to get help from Books, Web Pages, and FTP sites .....	25

## Introduction

---

### History of C

C is a general-purpose programming language with emphasis on economy of expression. It is not an 'all-included' nor 'high-level' language, but its absence of restrictions makes it more useful and popular.

C was designed by Dennis Ritchie for the UNIX operating system on PDP-11. The operating system and most UNIX applications programs including the C compiler are written in C. C is not tied to any particular hardware and is widely available making C programming very portable.

C design was influenced by programming languages B and BCPL.

C does not include some of the features other languages offer (see following paragraphs), it makes C small and easy to install and to learn.

---

### What is included

C provides essential tools to construct structured programs:

- statement grouping
- decision making (if clause)
- condition selection (switch)
- loops and its control (while, for, do, break)
- functions
  - may returning values
  - may be recursive
- modular programming

---

### What is not included

C does not provide operations to manipulate composite objects such as strings or arrays.

There is no storage allocation (heap) other than static definitions. Other than the use of local variables of function to provide the stack facilities.

C does not provides input/output facilities! There is not READ or WRITE statements. These must be provided by writing your own functions, or, more commonly, provided by the most C implementations.

C offers only straightforward, single-thread control flow, but not multiprogramming, parallel, co-routines, etc.

Example program

```
#include <stdio.h>

main()
{
int j,i=0;

while (i<1||i>99)
{
printf("Enter a number from 1 to 99 ");
scanf("%d",&i);
}
for(j=1;j<10;j++)
{
printf("%d x %d = %d\n",j,i,j*i);
}
return(0);
}
```

progdemo

Compile and execute the program

### **Compilation**

This is done by the compiler, it take a source program, which is in human readable form (see above example) and creates a set of machine instruction.

### **Linking**

Most compilers need an external program for this process. during the linking process all the necessary links to routine that program needs are added and an executable file is created.

### **Execution**

Once the program has been compiled and lined, it only remain to execute or run it.

About this booklet.

This document has been created by Brian Littlechild, part of the User Services Group within Computing Services, it has not been created as an instruction manual, but is intended to be a supplement to the Computing Services Programming in C short course.

## A Couple of Rules

All C program have a function main(), this is also the entry point of the program. Statements for a function are enclosed between curly braces, this includes the main function.

```
main()  
{  
    program code  
}
```

C is a free form language so each instruction must be terminated, in C a semicolon ; is used to terminate instruction.

*\*note\** because each instruction needs to be terminates, you may find that when compiling a program, the compiler will report errors in lines of code that don't have any ! This is normally caused by the proceeding line of code which has not been terminated !

because C is a free form language we can indent instruction by putting space and line breaks within the program to make it much clearer to read. So use this to your advantage and make the write your programs in a structured and clear manner.

## Comments

Comments are very important, try to put as many comments in your program as possible, you may easily forget what your program is doing at various places, but if you comment your program as you are writing it, there will be no confusion.

comments are surrounded by /\* at the start and \*/ at the end, you can comment a single word, line or whole paragraph. You can also comment out parts of your program while you are testing and debugging it.

## What is a C program

In simple terms it's just a list of instructions that once compiled and linked the computer will attempt to follow. The C language is used in a wide range of applications from games to operating systems, many of the computer programs you use today started their life as a C or C++ program.

The commands used in the C language are very simple to use and understand singly, but when used in conjunction with one another can produce very useful routines and procedures, to perform complex and difficult tasks.

## Let get started.

### Exercise 1

- Use the DOS editor, create a source file `prog1.c` containing the following program  
Note a source file should always have a file extension `.c`

```
#include <stdio.h>

main()
{
    printf("Hello World");
}
```

prog1.c

- Compile and link the source program, use the command **bcc prog1.c**

With most compiler this is done in 2 steps, step 1 compiling and step 2 linking, the first step creates a object file with the file extension **.obj**, the second step links the object file with all the appropriate routines and creates an executable file with the file extension **.exe**.

we are using BORLAND C/C++ Version 4 compiler and are able to perform both steps with one single command.

- Run the program, simply type the file name you have given to the program, in this case **prog1** and press return.

## Not so simple Arithmetic.

### Exercise 2

- Enter, Compile and Run this program.
- Then change the values of a, b & c to 4, 5 & 8
- Recompile the program and run once again.

What is going wrong with this program ?

```
#include <stdio.h>
main()
{
int a,b,c,d;

a=5;
b=4;
c=2;
d=a*b/c;

printf("%d * %d / %d = %d",a,b,c,d);
}
```

prog2a.c

- Enter, Compile and Run this program.

Note the change in this statement `d=(float)a*b/c;`

We have cast the variable, in fact we are temporarily changing a,b & c variables from the integer type to a floating decimal point type of variable. This will not effect the variables only the outcome

```
#include <stdio.h>
main()
{
int a,b,c;
float d;

a=5;
b=6;
c=9;
d=(float)a*b/c;

printf("%d * %d / %d = %f",a,b,c,d);
}
```

prog2b.c

### Variables Types

We have now seen 2 types of variables, Integers and floating points (also know as real), take a look at the next table for some of the most common types.

Type	Description
float	for single precision floating point numbers, such as 3.33333. These are expressed in 32 bits and their range is $\pm 3.4E\pm 38$ ;
double	for double precision floating point numbers, which are twice as large as variables of type float, expressed in 64 bits within the range $\pm 1.7E\pm 308$ ;
int	for integer numbers, which are whole numbers without a decimal point, expressed in 16 bits within the range -32768 to 32767.
Long	for long integer numbers, in 32 bits within the range -2147483648 to 2147483647
char	for storing one byte or one ASCII character, which are any of the characters appearing on the keyboard. These are expressed in 8 bits and their range is - 128 to 127.
Unsigned int	unsigned integer numbers, expressed in 16 bits within the range 0 to 65535
unsigned long	unsigned long integer numbers expressed in 32 bits within the range 0 to 4294967295
unsigned char	unsigned character, expressed in 8 bits within the range 0 to 255.

## Getting some input.

In the next program we are using an array of characters (char name[20]) to store some input from the user, you can think of this as 21 (as the computer will count from 0) little boxes that each store a single character.

B	R	I	A	N															
---	---	---	---	---	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--

Note the 6<sup>th</sup> character is a space, and that it also take-up a element in the array, when calculating the number of element you will need in your programs, you must also account for an extra 1 used by the compiler to make the end of the data (the NULL character)

```
#include <stdio.h>

main()
{
    char name[20];

    printf("Please enter your name ");
    scanf("%s",&name);
    printf("Hello %s",name);
}
```

prog3a.c

In this program we have created a variable called name, and we use the scanf command in the stdio library to get input from the user and place it directly into the computers memory that has been allocated for the variable name. Note the & in front of the variable, this tell the compiler to use the address memory address of the variable.

Don't worry to much about this for now, just name a mental note that input may be directed to an address in memory

### Exercise 3a

- Compile and run the program.
- Type you name and look at the results.
- Now run the program again and type a name that is over 20 characters long !

In the next program we are the user for a number, working with numbers is a little bit simpler !

### Exercise 3b

- Enter , Compile and run the next program,
- Now try to add a third number to the argument !

```
#include <stdio.h>

main()
{
    int a,b;

    printf("Enter a two numbers ");
    scanf("%d %d",&a,&b);

    printf("%d + %d = %d",a,b,a+b);
}
```

prog3b.c

So far we have used %d and %f to format the output and scan for input, in the exercise 3 we have also used %s to indicate a string of characters, the next table give you the list of I/O fields and there meaning.

### **Table of I/O Field Types**

<b>Type</b>	<b>Meaning</b>
%c	one character output. The prefix u (%uc) can be used for unsigned char type constants.
%d	decimal (base 10), integer values. The prefix u or l can be used - (%ud) for unsigned int type number, or (old) for long int type number.
%e	scientific notation (3.6E-5) for expressing very large or very small real numbers.
%f	floating point value - must include a decimal point.
%g	general format to represent values in either e or f format, whichever has the shortest form.
%o	octal (base 8) values.
%p	pointer values.
%s	string variables.
%u	unsigned integers.
%x	hexadecimal (base 16) values.

## Loops

loops are one of the most useful parts of a program, they allow a program to repeat things many time, in a controlled fashion. The next 2 programs use a FOR loop and a WHILE loop.

A for loop is controlled by three parameters

- 1) The initialisation - done once at the beginning.
- 2) Condition of the loop - checked at the beginning of each iteration.
- 3) The increment - done at the end of each interaction.

A while loop only has one parameter.

- 1) Condition - checked at the beginning of each iteration.

and will continue to repeat while that condition is TRUE.

### Exercise 4a & 4b

- Compile and run the next 2 programs
- Then try to increase the number of time the loop is performed.

```
#include <stdio.h>
int i;
main()
{
    for (i=1;i<11;i++)
    {
        printf("5 x %d =%d \n",i,5*i);
    }
}
```

prog4a.c

```
#include <stdio.h>
int i;
main()
{
    I=1;
    while(i<11)
    {
        printf("5 x %d =%d \n",i,5*i);
        i++;
    }
}
```

prog4b.c

We have user the command while(i<11) note the I<11 this means I is less than 11 and is known as a relational operator, take a look at the next table for more relation operators.

### Table of Relational Operators

C symbol	Example	Meaning
==	a==b	a equal to b
<	a<b	a less than b
<=	a<=b	a less than or equal to b
>	a>b	a greater than b
>=	a>=b	a greater than or equal to b
!=	a!= b	a not equal to b

## SPECIAL CHARACTERS

<code>\n</code>	Newline
<code>\t</code>	Tab
<code>\b</code>	Backspace
<code>\"</code>	Double Quotes
<code>\\</code>	Back Slash

The next program show a nested loop (a loop within a loop) there is an out loop counting from 1 to 5 and an inner loop also counting from 1 to 10, this will make the commands within both loops to be repeated 50 times.

### Exercise 4c

- Enter and compile the next program,
- Try to Increase the outer loop so that it will count from 1 to 10, making the number of repeats 100.

We have also sneaked an IF command into the program, we will be looking into the IF command in greater detail later.

```
#include <stdio.h>
int i,j;
main()
{
    for (i=1;i<6;i++)
    {
        for (j=1;j<11;j++)
        {
            printf("%d ",i*j);
            if ((i*j)<10)
            {
                printf(" ");
            }
        }
        printf("\n");
    }
}
```

prog4c.c

## QUICK TEST.

Which is a valid C program ?

a)	main()	
b)	main() { }	
c)	main{ }	

What command can be used to count from 1 to 10 ?

a)	for (i=1;i<10;i++)	
b)	for(i=1;i<11;i++)	
c)	for(i=0;i<10;i++)	

What is %s used for in a printf statement ?     printf(“%s”);

a)	to declare a sum	
b)	to declare a string	
c)	to declare a space	

What is %d used for in a printf statement ?     printf(“%d”);

a)	to declare a dot	
b)	to declare a delimiter	
c)	to declare a decimal	

Which statement will assign the value 5 to a ?

a)	5=a;	
b)	a=5;	
c)	a,5	

Which statement will add 1 to a ?

a)	a=1;	
b)	a++;	
c)	a+1;	

Find the three bugs in this program (use the compiler if you need help)

```

int a,b,c;

main()
{
    a=4
    b=6;
    c=5.2;
    printf(“The sum of %d %d = %d”,a,b,c,a+b+c);
}
```

1)	
2)	
3)	

## Is it cold in here ?

### Exercise 5

- Write a program to ask the user for a temperature in centigrade and then print the same temperature in Fahrenheit.

Break the program down like this.

- 1) ask the user for the temperature.
- 2) store the answer in a variable.
- 3) calculate the temperature and store in a second variable.
- 4) display the calculated temperature(s).

## Decisions.

The most command used for decision making is IF, the syntax is very simple

```
if ( <condition> ) <statement1>;
```

or

```
if ( <condition> ) <statement1>;  
    else <statement2>;
```

### Exercise 6.

- Enter, compile and run the next program.
- Now add an else statement to print X is an odd number

```
#include <stdio.h>  
  
main()  
{  
    int a,b;  
    float c;  
    printf("Enter a number ");  
    scanf("%d",&a);  
    b=a/2;  
    c=(float) a/2;  
    if ( b==c )  
    {  
        printf("%d is an even number\n",a);  
    }  
}
```

prog6.c

## Functions.

So far we have only had one function in our program. In more complexed program you can group lines of your source program and define them as a function, once defined you can call upon them many time.

### Exercise 7.

- Enter, compile and run the next program.
- Modify the program 5 so it will ask for a temperature then use a function to convert it.

```
#include <stdio.h>

float convert (int);

#define COUNT 31
main()
{
    int degf;
    float degc;

    for(degf=0;degf<100;degf=degf+10)
    {
        degc=convert(degf);
        printf("%d fahrenheit = %5.2f celsius \n",degf, degc);
    }
    return(0);
}

/*convert function*/
float convert(int heat)
{
    float temp_value;
    temp_value = (5.0/9.0)*(heat-32);
    return temp_value;
}
```

prog7.c

## Strings.

Most of the program we have written have been using numerical variables, but strings are just as important, so the next program will demonstrate some of the command used in conjunction with character arrays (strings)

```
#include <stdio.h>

main()
{
  int test_value;
  char user_name[50];

  printf("\n\nPlease type your name :");
  scanf("%s",user_name);
  test_value=(strcmp(user_name,"brian"));

  if ( test_value == 0 )
    printf("Hello %s nice to see you again",user_name);
  else
    printf("Hello %s",user_name);
}
```

prog8.c

In this program we are asking the user to enter a name, then test that name, and display an appropriate greeting.

The command strcmp is used to test two strings and return a TRUE or FALSE value (1 or 0).

### Exercise 8.

- Enter, Compile and run program 8.
- Try typing a name with mixed case, to see what happens !
- Now find a better test we can perform from the table of string function on the next page.

## Table of String Functions

Function Call	Operation
strcat(str1 ,str2)	Concatenates (appends) string str2 to str1
strncat(str1 ,str2,n)	Concatenates n number f bytes from string str2 to str1
strcmp(str1 ,str2)	Compares string str2 to str1. Returns a value (<0, 0 or >0) based on result of comparison
stricmp(str1 ,str2)	Compares string str2 to str1, without case sensitivity
strnicmp(str1 ,str2,n)	Compares n number of bytes from string str2 to str1, without case sensitivity
strcpy(str1 ,str2)	Copies string str2 into str1
strdup(pnt,strn)	Copies a string into a new location, returning a pointer
strncpy(str1 ,str2,n)	Copies n number of bytes from str2 to str1
strlwr(strn)	Converts upper case letters in a string to lower-case
strupr(strn)	Converts lower case letters in a string to upper case
strlen(strn)	Returns the length of a string
strrev(strn)	Reverses a string and returns a pointer
strtod(strn)	Converts a string to a <b>double</b> value
strtol(strn)	Converts a string to a <b>long</b> value
strchr(strn,chr)	Scans string strn for the first occurrence of a character in chr, returning a pointer
strcspn(str1 ,str2)	Scans string str1 for the first segment not containing any subset of characters in str2, and returns a pointer
strpbrk(str1 ,str2)	Scans string str1 for the first occurrence of any character from string str2
strrchr(strn,chr)	Scans a string (in the reverse direction) for the last occurrence of a given character
strspn(str1 ,str2)	Scans string str1 for the first segment that is a subset of Str2, returning its length
strstr(str1 ,str2)	Scans string Str2 for occurrence of string str1, returning a pointer to the element in str2
strtok(str1 ,str2)	Scans string str1 for delimited tokens which are defined in string
strnset(strn,n,chr)	Sets n number of bytes in string strn to a given character in chr
strset(strn,chr)	Sets all characters in string strn to a given character in chr

## FILES.

Probably THE MOST IMPORTANT part of C programming is the ability to read and write a file.

The next program asks for a name and stores it into a file.

### Exersie 9a

- Enter , Compile and run program 9a
- Take a look at the output file, use the DOS editor to open and look at it.

```
#include "stdio.h"

main()
{
FILE *fp;
char letter;
char name[20];

printf("Enter your name");
scanf("%s",&name);

fp = fopen("TESTFILE","wt");

fputs(name, fp);

fclose(fp);
}
```

prog9a.c

Now we need a program to read the stored data from the file.

### Exersie 9b

- Enter , Compile and run program 9b
- Add a couple of name to the file, using the DOS editor or by modifying program9a, and re-run the program, NOTE you do not have to re-compile the program9b.

```
#include "stdio.h"

main()
{
FILE *fp;
char letter;
char name[20];

fp = fopen("TESTFILE","rt");

while (!feof(fp))
{
fgets(name,20, fp);
printf("%s",name);
}

fclose(fp);
}
```

prog9b.c

There are many ways to access files and we have only scratched the surface of this topic, below is a table of access modes, most will become self-explanatory when you come to writing a program that needs a particular mode of access.

#### **Table of Values for file Mode**

<b>Symbol</b>	<b>Meaning</b>
---------------	----------------

"r"	allows reading from a text file
"w"	allows writing to a text file, but could overwrite existing data
"a"	allows data to be appended to the existing data of a text file
"rb"	allows reading from a binary file
"wb"	allows writing to a binary file, but could overwrite existing data
"ab"	allows data to be appended to the existing data of a binary file
"r+"	allows read/write from a text file
"w+ "	creates a text file for reading/writing
"a+ "	allows read/write or creates a text file
"r+b"	allows read/write from a binary file
"w+b"	creates a binary file for reading/writing
"a+b"	allows read/write or creates a binary file

## Did we win the lottery.

Here is a little program I wrote to check my lottery ticket every week, as I trust the computer to check my ticket more than I trust myself , especially when there is a million pounds at stake !

The lottery program is based on 2 sets of number, the number we have bet on and the number that have been selected by the national lottery !

In this program the number we bet on will be stored in a file called LOTO.DAT so the first problem is to get the number into a file that the computer can read and understand !

So let look at some possible solutions.



```
#include <stdio.h>

main()
{
int a,b,c,d,e,f;
FILE *fopen(), *fptr;

fptr=fopen("loto.dat","wb");

a=14;
putw(a,fptr);
a=20;
putw(a,fptr);
a=6;
putw(a,fptr);
a=34;
putw(a,fptr);
a=36;
putw(a,fptr);
a=8;

a=7; putw(a,fptr);
a=41; putw(a,fptr);
a=43; putw(a,fptr);
a=25; putw(a,fptr);
a=11; putw(a,fptr);
a=9; putw(a,fptr);

fclose(fptr);
}
```

make1.c

```

#include <stdio.h>

main()
{
int i,j,a;
FILE *fopen(), *fptr;

fptr=fopen("loto.dat","wb");

for(i=1;i<=2;i++)
{
for(j=1;j<=6;j++)
{
printf("Enter Number for game %d, Number %d ",i,j);
scanf("%d",&a);
putw(a,fptr);
}
}

fclose(fp);
}

```

make2.c

```

#include <stdio.h>

main()
{
int i,j,a;
FILE *fopen(), *fptr;

fptr=fopen("loto.dat","wb");

for(i=1;i<=12;i++)
{
printf("Enter Number for game %d, Number %d ",((i-1)/6)+1,i-
((i-1)/6)*6 );
scanf("%d",&a);
putw(a,fptr);
}
fclose(fp);
}

```

make3.c

Now for the fun part, the second problem is to get the program to read the file that we have created and cross check the numbers with a new set of numbers that have been selected by the national lottery.

```
#include <stdio.h>
#define NUMBER_OF_GAMES=2;
main()
{
int winning_balls[7];
int numbers[12];
int i,j,k;
float score;
FILE *fopen(), *fptr;
char keyp;
/* read the number we have selected this week */
fptr=fopen("loto.dat","rb");
for(i=1;i<=12;i++)
{
numbers[i]=getw(fptr);
}
fclose(fptr);
/* get this weeks winning balls from the user */
printf("Please enter this weeks winning balls\n\n");
for (i=1;i<=7;i++)
{
printf("Ball No.%d ",i);
scanf("%d",&winning_balls[i]);
}
printf("\n\n\n\n");/* print some blank lines to clear the screen */

/* OK check all the balls */
for (i=1;i<=2;i++) /* loop for number of games played */
{
score = 0;
printf("Game No.%d\t",i);
for (j=1;j<=6;j++) /* loop for each number played in game */
{
if (numbers[(((i-1)*6)+j)] == winning_balls[7])
score=score+0.5;
for (k=1;k<=6;k++) /* loop for each winning ball not bonus */
{
if (numbers[(((i-1)*6)+j)] == winning_balls[k])
{
score=score+1.0;
printf("*");
}
}
printf("%d ",numbers[(((i-1)*6)+j)]);
}
}
/* well how did we do */
if (score == 3.0 || score == 3.5)
printf("\tYou've won æ 10.00\n");
if (score == 4.0 || score == 4.5)
printf("\tYou've won æ 50.00 ish\n");
if (score == 5.0)
printf("\tWow you won a lot\n");
if (score == 5.5)
printf("\tYou had better sit down\n");
if (score == 6.0)
printf("\tYou are now a millionaire\n");
printf("\n");
}
}
```

## **Where to get help from !**

Computing Services Help desk will always try to help and assist you, but by far the best sources of information are Books, there are many books covering C and C++ in most library's including the QMW main library.

You can also find lots of information on the World Wide Web, there are many sources programs available on the WEB along with freeware and shareware C and C++ compiler.

A good place to start is the nation archives at Lancs., you can access this with a Web browser or FTP program.

Useful Address and URL's

<http://micros.hensa.ac.uk>

<ftp://micros.hensa.ac.uk>