

Basic C Language

A simple C program is made up of four parts

- Libraries – to include standard functions necessary for the operation of the program, e.g. to use input and output functions a suitable library must be included
- Headers – to mark the beginning of the code, the standard header for the main function must be included
- Variable declaration – assigning names and data types to all the variables to be used in the code
- Code

```
#include <iostream>           // Library
using namespace std;
void main(void)               // Header
{
    cout<<"hello world! \n";    // Code
}
```

Libraries

The C language, and C++, supports a large number of standard libraries which provide a set of functions to be used in a program. For the simplest programs, the only library needed is **iostream**, which includes input and output functions necessary for reading the keyboard and displaying information on the screen.

To display information on the screen use **cout**, and to read input from the keyboard use **cin**. For example,

```
cout << "This is a test";
```

Anything inside quotes is displayed literally, and is not modified in any way by the computer. It is also possible to include escape sequences inside the quotes to format the information or display special characters, e.g.

```
cout << "This is a test \n";
```

In the above example, after the message is displayed, the cursor is moved to the beginning of the next line.

The following escape sequences can also be included in the literal string,

- 1) \n - newline.
- 2) \t - horizontal tab.
- 3) \v - vertical tab.
- 4) \b - backspace.
- 5) \r - carriage return.
- 6) \\ - backslash (\).
- 7) \' - single quote (').
- 8) \" - double quotes (").

To output the value stored in a variable, you must specify the variable name, as follows

```
cout << number;
```

Note that quotes are not used in this example, and must not be used if you wish to display the value stored in the variable.

To display more than one piece of information use the following syntax

```
cout << "The answer is " << number << "\n";
```

In this example, three items are displayed, the literal string, the value stored in the variable number and the newline character.

To input data from the keyboard use the following syntax

```
cin >> number;
```

When this statement is executed, the program will stop until a value is entered via the keyboard, and the return key is pressed. You should take care to enter the correct type of data into the variable as specified in its declaration, or your program may not work as you expect or intend.

Headers

Headers mark the beginning of section of code. In the simplest examples only the main header is required,

```
void main (void)
```

Note that this line does not end with a semi-colon(;

Variable declaration

Before a C (or C++) program can be run the computer needs to know how much memory it must allocate to the program so that it executes correctly. This is done by declaring all the variables to be used in the program before they are used for any calculation etc. Declaration requires that the type of data to be saved in the variable must be specified along with a unique name for that variable. The following data types are allowed

- 1) int - integers (32 bits -2,147,483,648 to 2,147,483,647)
- 2) long - may allow larger integers to be used on a 64 bit system (32 bits).
- 3) short – may save storage space when only small integers are required (16 bits - -32,768 to 32,767).
- 4) unsigned – can be used to qualify int, long or short so that only positive integers are used, so doubling the maximum value (32 bits - 0 to 4,294,967,295).

- 5) char - single characters (8 bits).
- 6) float - floating point or real numbers (32 bits, with a range of 3.4E +/- 38 (7 digits)).
- 7) double - double precision floating point numbers (64 bits, with a range of 1.7E +/- 308 (15 digits)).

A name for the variable must also be specified. The computer will use this name to point to the memory location where the corresponding data is stored. Variable names must start with a letter, which can be followed by any combination of digits, letters and the underscore character. It is good programming practice to give variables descriptive names so that they help with the understanding of the program. Typical examples of variable declarations are shown below

```
int count, value, number;

char reply, input;

float temperatureReading;
```

Code

This is the part of the structure where the program carries out the calculations etc. required. In a very simple program this would include data entry, arithmetic calculations and output of results.

Arithmetic Operations

The following arithmetic operators are supported

Binary

- = assignment e.g. x = 5;
- + addition e.g. x = 3 + 2;
- subtraction e.g. x = 3 - 2;
- * multiplication e.g. x = 3 * 2;
- / division e.g. x = 3 / 2;
- % modulus e.g. x = 3 % 2; (calculates the remainder of an integer division)

Unary

- ++ increment e.g. x++; (adds one to the value of x)
- decrement e.g. x--; (subtracts one from the value of x)

Compound Assignment

- += addition, x += 1 is equivalent to x = x + 1;
- = subtraction, x -= 1 is equivalent to x = x - 1;
- *= multiplication, x *= 1 is equivalent to x = x * 1;
- /= division, x /= 1 is equivalent to x = x / 1;
- %= modulus (remainder), x %= 1 is equivalent to x = x % 1;

Using these elements, a simple program can be written, such as the example shown below. This shows a sequence of statements which are executed in order.

```
#include <iostream>
using namespace std;

void main(void)
{
    float v,i,r;
    cout << "enter the resistance value (in ohms) : ";
    cin >> r;
    cout << "enter the current (in amps) : ";
    cin >> i;
    v = i * r;
    cout << "the voltage is " << v << " volts \n";
}
```

Control Structures

Most programmes require a more complex structure than the one shown above. Generally, programmes are required to choose between a range of options and to repeat calculations. C supports the usual control structures that you would expect in any programming language, selection and repetition. By combining these two structures with straightforward sequences of statements it is possible to write the program you need.

Selection

C supports two selection constructs, **if** and **switch**. As a general rule, use the **if** statement when there are only two or three options, and use the **switch** statement when there are more than three options to choose from.

if Statement

The general syntax of the **if** statement is shown below

```
if (Boolean expression)
{
    statement1;
}
else
{
    statement2;
}
```

A Boolean expression is one which returns a value of true or false. No other results are possible. They are used to control the flow of execution through the program, i.e. they determine which statement will be executed next.

In the general syntax shown above, statement1 and statement2 should be replaced by any valid C statement or group of statements. Statement1 would be executed if the Boolean expression is

true, and statement2 would be executed if the Boolean expression is false. The **else** part of the statement is optional and can be omitted as required.

Boolean expressions are created using the relational operators

>	greater than
<	less than
==	equal to
!=	not equal to
>=	greater than or equal to
<=	less than or equal to

For example, the following are valid Boolean expressions

x == 5	is true if the value stored in the variable x is 5
x > 5	is true if the value stored in the variable x is greater than 5
x != 5	is true if the value stored in the variable x is not equal to 5

Boolean expressions can be combined or modified using logical operators

	OR
&&	AND
!	NOT

For example,

if ((x > 5) || (y < 6)) is true if x is greater than 5 or y is less than 6

The table below shows a truth table for an OR (||) operation

if ((Boolean expression 1) || (Boolean expression 2))

Boolean expression 1	Boolean expression 2	Boolean expression 1 Boolean expression 2
true	true	true
true	false	true
false	true	true
false	false	false

The table below shows a truth table for an AND (&&) operation

if ((Boolean expression 1) && (Boolean expression 2))

Boolean expression 1	Boolean expression 2	Boolean expression 1 && Boolean expression 2
true	true	true
true	false	false
false	true	false
false	false	false

The table below shows a truth table for a NOT (!) operation

if (!(Boolean expression 1))

Boolean expression 1	!Boolean expression 1
true	false
false	true

Examples of the use of the if statement

Example 1

```
#include <iostream>
using namespace std;

// this program illustrates the use of the if statement

void main(void)
{
    int answer;
    cout << "do you want option 1 or 2 \n";
    cout << "enter 1 or 2 :- ";
    cin >> answer;
    if (answer == 1) cout << "you chose option 1 \n";
}
```

Example 2

```
#include <iostream>
using namespace std;

// this program illustrates the use of the if else statement

void main(void)
{
    int answer;
    cout << "do you want option 1 or 2 \n";
    cout << "enter 1 or 2 :- ";
    cin >> answer;
    if (answer == 1) cout << "you chose option 1 \n";
    else cout << "you chose option 2 \n";
}
```

Example 3

```
#include <iostream>
using namespace std;

// this program illustrates the use of a nested if else statement

void main(void)
{
    int answer;
    cout << "do you want option 1 or 2 \n";
    cout << "enter 1 or 2 :- ";
    cin >> answer;
    if (answer == 1) cout << "you chose option 1 \n";
    else if (answer == 2) cout << "you chose option 2 \n";
    else cout << "option not available \n";
}
```

Example 4

```
#include <iostream>
using namespace std;

// this program illustrates the use of if statements with compound statements

void main(void)
{
    int answer, value;
    cout << "do you want option 1 or 2 \n";
    cout << "enter 1 or 2 :- ";
    cin >> answer;
    if (answer == 1)
    {
        cout << "you chose option 1 \n";
        cout << " enter the value to be calculated :- ";
        cin >> value;
        value *= value;
        cout << "the square of the number entered is "
             << value << "\n";
    }
    else if (answer == 2)
    {
        cout << "you chose option 2 \n";
        cout << " enter the value to be calculated :- ";
        cin >> value;
        value += value;
        cout << "double the number entered is "
             << value << "\n";
    }
    else cout << "option not available \n";
}
```

Switch Statement

The **switch** statement is used when there are a number of possibilities available. The use of the **if** statement would be clumsy and complex, because of the number of choices available. It is often used when the user is presented with a menu of choices.

The general syntax of the **switch** statement is shown below

```
switch (controlVariable)
{
    case value1 : statement1;
                  break;
    case value2 : statement2;
                  break;
    case value3 : statement3;
                  break;
    case value4 : statement4;
                  break;
    default :    statement5;
}
```

In the general syntax shown above, statement1, statement2 etc. should be replaced by any valid C statement or group of statements. Statement1 would be executed if the controlVariable matches value1, statement2 would be executed if the controlVariable matches value2 etc. The controlVariable must be an integer type, which also includes **char**.

Switch statement example

```
#include <iostream>
using namespace std;

void main(void)
{
    float v,i,r;
    char reply;
    cout << " Do you want to calculate the voltage, current or resistance ?
\n" << " Enter v, i or r : ";
    cin >> reply;
    switch (reply)
    {
        case 'v' :          // user selects v
        case 'V' : {
            float v,i,r;
            cout << "enter the resistance value(in ohms):";
            cin >> r;
            cout << "enter the current (in amps) : ";
            cin >> i;
            v = i * r;
            cout << "the voltage is " << v << " volts \n";
        }
        break;
        case 'i' :          // user selects i
        case 'I' : {
            cout << "enter the resistance value(in ohms):";
            cin >> r;
            cout << "enter the voltage (in volts) : ";
            cin >> v;
            i = v / r;
            cout << "the current is " << i << " amps \n";
        }
        break;
        case 'r' :          // user selects r
        case 'R' : {
            cout << "enter the voltage (in volts) : ";
            cin >> v;
            cout << "enter the current (in amps) : ";
            cin >> i;
            r = v / i;
            cout << "the resistance is " << r << " ohms\n";
        }
        break;
        default : cout << "\n\nInvalid selection \n\n";
    }
}
```

Repetition

C supports three repetition constructs, **for**, **while** and **do....while**. As a general rule, use the **for** statement when you know how many repetitions are required, use the **do....while** loop when you require at least one repetition of the statements. The **for** and **while** statements are described as entry condition loops, i.e. the test for staying in the loop is done before entry. The **do....while** loop is described as an exit condition loop i.e. the test for staying in the loop is done at the exit point.

for Loop

The general syntax of the **for** loop is shown below

```
for (initialise; test; update)
{
    statement;
}
```

The syntax of the **for** loop is made up of three parts

initialise	set the initial value of the control variable (which must be of integer type)
test	check that the control variable lies within the specified range i.e. satisfies the Boolean expression
update	update the value of the control variable e.g. increment its value

for statement example

```
#include <iostream>
using namespace std;

void main(void)
{
    int multiplier, table, result;
    cout << "Enter the number for the multiplication table to be displayed :";
    cin >> table;
    for (multiplier = 1; multiplier <= 12; multiplier++)
    {
        result = multiplier * table;
        cout << multiplier << " X " << table << " = " << result << "\n";
    }
}
```

while Loop

The general syntax of the **while** loop is shown below

```
while (Boolean expression)
{
    statement;
}
```

The statement inside the loop will be repeated as long as the Boolean expression is true. The number of iterations for this type of loop varies between 0 and n (where n is an integer value). You must ensure that the value of the control variable within the Boolean expression can be changed inside the loop or the program will remain inside the loop forever (known as an infinite loop).

Example of the use of the while loop

```
#include <iostream>
using namespace std;

void main(void)
{
    int answer, value;
    char repeat = 'y';
    while (repeat == 'y')
    {
        cout << "do you want option 1 or 2 \n";
        cout << "enter 1 or 2 :- ";
        cin >> answer;
        if (answer == 1)
        {
            cout << "you chose option 1 \n";
            cout << " enter the value to be calculated :- ";
            cin >> value;
            value = value * value;
            cout << "the square of the number entered is "
                << value << "\n";
        }
        else if (answer == 2)
        {
            cout << "you chose option 2 \n";
            cout << " enter the value to be calculated :- ";
            cin >> value;
            value = value + value;
            cout << "double the number entered is "
                << value << "\n";
        }
        else cout << "option not available \n";
        cout << " would you like to choose another option ? ";
        cin >> repeat;
    }
    cout << "end of program \n";
}
```

do....while Loop

The general syntax of the **do....while** loop is shown below

```
do
{
    statement;
}
while (Boolean expression);
```

The statement inside the loop will be repeated as long as the Boolean expression is true. The number of iterations for this type of loop varies between 1 and n (where n is an integer value). You must ensure that the value of the control variable within the Boolean expression can be changed inside the loop or the program will remain inside the loop forever (known as an infinite loop).

Example of the use of the do....while loop

```
#include <iostream>
using namespace std;

void main(void)
{
    int answer, value;
    char repeat;
    do
    {
        cout << "do you want option 1 or 2 \n";
        cout << "enter 1 or 2 :- ";
        cin >> answer;
        if (answer == 1)
        {
            cout << "you chose option 1 \n";
            cout << " enter the value to be calculated :- ";
            cin >> value;
            value = value * value;
            cout << "square of the number entered is "<< value << "\n";
        }
        else if (answer == 2)
        {
            cout << "you chose option 2 \n";
            cout << " enter the value to be calculated :- ";
            cin >> value;
            value = value + value;
            cout << "double the number entered is " << value << "\n";
        }
        else cout << "option not available \n";
        cout << " would you like to choose another option ? ";
        cin >> repeat;
    }
    while (repeat == 'y')
    cout << "end of program \n";
}
```

Structured Data Types

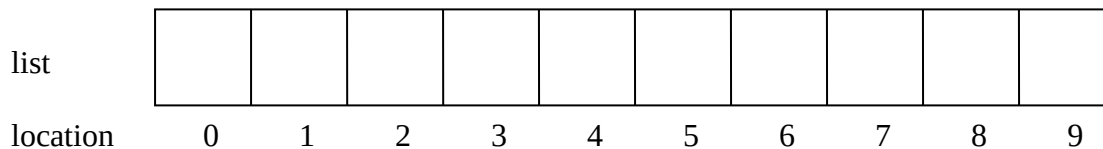
The simple variables described earlier can hold one item of data only. There are many circumstances when programs are developed where it is necessary to group related data together. The simplest way of achieving this is to use a structured data type known as an array. An array is a collection of related items, all of which have the same data type. The variables are identified as arrays by the square brackets which follow the name. To declare a one dimensional array, the following general syntax is used

```
type arrayName[size];
```

where type can be any of the standard types listed above, and size can be any integer value. For example,

```
int list[10];
```

declares an array called list which can hold 10 integer values. The simple diagram below illustrates the array's structure



It shows 10 memory locations allocated to the array, each able to store one integer value. To access these locations, an index is used where 0 corresponds to the first location and 9 corresponds to the tenth location. For example, to fill each element of the array with the value zero, a loop would be used as shown below

```
for (i = 0; i < 10; i++)
{
    list[i] = 0;
}
```

To enter information into the array number from the keyboard, code similar to that shown below could be used

```
for ( i= 0 ; i < 10 ; i++)
{
    cout << "\n enter number["<< i << "] : ";
    cin >> list[i]);
}
```

The indices, i.e. the numbers used to identify the elements of an array, must be integers.

Arrays can also be multi-dimensional, for example a two-dimensional array could be declared as follows

```
int table[rows][columns];
```

C distinguishes between an array of characters and a string by placing a null character (\0) at the end of the string. A string is a character array in which the last character is the null character. The null character takes up one element of the array, and allowance must be made for this character when dimensioning the array. A complete string can be read into an array with one cin statement. For example,

```
char name[50];  
cout << "what's your name ? ";  
cin >> name;  
cout << "\n hello " << name << endl;
```

The output of this block of code would be

what's your name ? Fred Bloggs

hello Fred

In this example of output, the information printed by the computer is shown in bold type. Only the first word of the name has been placed in the variable. cin only reads single words, and stops when it encounters a "white space" (space, tab or newline).

Array example

This program calculates the sum and average of a list of numbers.

```
#include <iostream>
using namespace std;

void main (void)
{
    // the next three lines declare the variables
    // that this program will use
    int count, i;
    float average,sum,value[20]; // value is an array variable which
    char again;                  //can hold 20 floating point values

    cout << "This program will calculate the sum and average of a list of
numbers \n";
    cout << "the maximum number of values that can be entered is 20 \n";
    cout << "the user can end the list by entering a value of zero \n";

    do
    {
        average = 0; // these 3 lines initialise the variables
        count = 0;   // used to calculate the sum and average
        sum = 0;

        for (i = 0; i < 20; i++) // this loop places -1 in every
            value[i] = -1;       // element of the array

        do // this loop asks the user to enter values and calculates
        { // the sum of these values
            cout << "\nEnter value [" << count+1 << "] : ";
            cin >> value[count];
            sum = sum + value[count]; // this line sums the values entered
            if (value[count] != 0) count++; //this line keeps count of how
                                           //many numbers have been entered
        }
        while ((count < 20) && (value[count] != 0)); //this line checks for a
                                                    //full array or a zero entry
        average = sum/count; // this line calculates the average of the numbers

        cout << "you entered the following " << count << " numbers \n";
        for (i=0; i < count; i++) // this loop displays the numbers entered
            cout << "\nvalue [" << i+1 << "] = " << value[i];

        cout << "\n\nThe sum of these numbers is " << sum;
        cout << "\n\nThe average of these numbers is " << average;

        cout << "\n\nDo you want to enter another list of numbers ? ";
        cin >> again;
    }
    while ((again == 'y') || (again == 'Y')); // this line checks to see if
                                              // the user wants another go

    cout << "\n\n End of program    \n\n";
}
```

