# C++ Programming Language

All computer systems consist of similar hardware devices and software components.

- **Hardware**: Hardware refers to the physical components that a computer is made of. A computer is not an individual device, but a system of devices. A typical computer system consists of the following major components:
  1. The central processing unit (CPU)
  2. Main memory (RAM)
  3. Secondary storage devices
  4. Input devices
  5. Output devices

- **Software**: Software refers to the programs that run on a computer. There are two general categories of software: operating systems and application software. An operating system is a set of programs that manages the computer's hardware devices and controls their processes. Application software refers to program that the computer useful to the user. These programs solve specific problems or perform general operations that satisfy the needs of the user.

## Programming Languages

There are two categories of programming languages: low level and high level. A low level language is close to the level of the computer, which means it resembles the numeric machine language of the computer more than natural language of humans. The easiest languages for people to learn are high level language. They are called "high level" because they are closer to the level of human readability than computer-readability like C++.

# How C++ Programming Works

The C++ programming language is a popular and widely used programming language for creating **computer programs**. Programmers around the world embrace C++ because it gives maximum control and efficiency to the programmer.

If you are a programmer, or if you are interested in becoming a programmer, there are a couple of benefits you gain from learning C++:

- You will be able to read and write code for a large number of platforms -- everything from microcontrollers to the most advanced scientific systems can be written in C++, and many modern operating systems are written in C++.
- C++ is an object oriented language. C++ is an extension of C, C++ also includes several other improvements to the C language, including an extended set of library routines.

We will walk through the entire language and show you how to become a C++ programmer, starting at the beginning. You will be amazed at all of the different things you can create once you know C++!

**What is C++?**

C++ is a **computer programming language**. That means that you can use C++ to create lists of instructions for a computer to follow. C++ is one of thousands of programming languages currently in use. It gives programmers maximum control and efficiency. C++ is an easy language to learn. It is a bit more cryptic in its style than some other languages, but you get beyond that fairly quickly.

C++ is what is called a **compiled language**. This means that once you write your C++ program, you must run it through a C++ **compiler** to turn your program into an **executable** that the computer can run (execute). The C++ program is the human-readable form, while the executable that comes out of the compiler is the machine-readable and executable form.

When a C++ program is written, it must be typed into the computer and saved to a file. A text editor, which is similar to a word processing program, is used for this task. The statements written by the programmer are called **source code**, and the file they are saved in is called the source file. After the source code is saved to a file, the process of translating it to machine language can begin.

During the first phase of this process, a program called the **preprocessor** reads the source code. The preprocessor searches for special lines that begin with the (#) symbol. These lines contain commands that cause the preprocessor to modify the source code in some way as we will see later.

During the next phase the compiler steps through the preprocessed source code, translating each source code instruction into the appropriate machine language instruction. This process will uncover any syntax errors that may be in the program. Syntax errors are illegal uses of keywords, operators, punctuation, and other language elements. If the program is free of syntax errors, the compiler stores the translated machine language instructions, which are called **object code**, in an **object file**.

Although an **object file** contains machine language instructions, it is not a complete program, because C++ is conveniently equipped with a library of prewritten code for performing common operations or sometimes difficult tasks. For example, the library contains mathematical functions, such as calculating the square root of a number. The collection of code, called the **run time library**, is extensive. Programs almost always use some part of it. When the compiler generates an **object file**, however, it does not include machine code for any run time library routines the programmer might have used. During the last phase of the translation process, another program called the linker combines the object file with the necessary library routines. Once the linker has finished with this step, an **executable file** is created. The executable file contains machine language instructions, or **executable code**, and is ready to run on the computer.

**Libraries**

Libraries are very important in C++ because the C++ language supports only the most basic features that it needs. C++ does not even contain I/O functions to read from the keyboard and write to the screen. Anything that extends beyond the basic language must be written by a programmer. The resulting chunks of code are often placed in **libraries** to make them easily reusable. We have seen the standard I/O, or **stdio**, library already: Standard libraries exist for

standard I/O, math functions, string handling, time manipulation, and so on. You can use libraries in your own programs to split up your programs into modules. This makes them easier to understand, test, and debug, and also makes it possible to reuse code from other programs that you write.

**The Simplest C++ Program**

Let's start with the simplest possible C++ program and use it both to understand the basics of C++ and the C++ compilation process.

```
#include <iostream.h>

int main()
{
    cout << "Hello";
    return 0;
}
```

When executed, this program instructs the computer to print out the line "Hello" -- then the program quits. You can't get much simpler than that!

You should see the output "Hello" when you run the program. Here is what happened when you compiled the program:

If you mistype the program, it either will not compile or it will not run. If the program does not compile or does not run correctly, edit it again and see where you went wrong in your typing. Fix the error and try again.

The Simplest C++ Program: What's Happening?

- **#include <iostream.h>** tells the preprocessor to include the iostream standard file. This specific file (iostream) includes the declarations of the basic standard input-output library in C++, and it is included because its functionality is going to be used later in the program.
- The line **int main()** declares the main function. Every C++ program must have a function named **main** somewhere in the code. At run time, program execution starts at the first line of the main function.
- In C++, the **{** and **}** symbols mark the beginning and end of a block of code. In this case, the block of code making up the main function contains two lines.
- cout << "Hello"; This line is a C++ statement. A statement is a simple or compound expression that can actually produce some effect. In facts, this statement performs the only action that generates a visible effect in our first program. cout represents the standard output stream in C++, and the meaning of the entire statement is to insert a sequence of characters (in this case, the Hello sequence of characters) into the standard output stream (which usually is the screen).
- The **return 0;** line causes the function to return an error code of 0 (no error) to the shell that started execution.

- We can little update the previous program to become

```
#include <iostream.h>
#include <conio.h>
int main()
{
clrscr();
cout<<"Hello";
getch();
return 0;
}
```

□ **#include <conio.h>**. This line **includes** the "console I/O library" into your program.
□clrscr() clears the current text window and places the cursor in the upper left-hand corner (at position 1,1).
□getch() reads a single character directly from the keyboard, without echoing to the screen.

- **To output multiple lines**

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

int main()
{
    clrscr();
    cout << "Hello";
    cout<<"nice to meet you";
    getch();
    return 0;
}
```

```
The output becomes:
Hellonice to meet you
```

You can use \n for newline at the end of the 1st message, so that the 2nd message appear in the next line, and the program becomes:

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

int main()
{
    clrscr();
    cout << "Hello\n";
    cout<<"nice to meet you";
    getch();
    return 0;
}
```

```
The output becomes:
Hello
nice to meet you
```

which is the same as to

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

int main()
{
    clrscr();
    cout << "Hello\nnice to meet you";
    getch();
    return 0;
}
```

- You may also use the endl manipulator to add a new line:

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

int main()
{
    clrscr();
    cout << "Hello"<<endl<<"nice to meet you";
    getch();
    return 0;
}
```

## Comments

Comments may be of the form:

// *comment*

or

/* *comment* */

The first form allows a trailing comment on a single line, while the second form allows comments that span multiple lines.

Comments may appear anywhere.

## Function Declarations

Provides the full definition of a function, while
*storage_class type identifier (formal_argument_list);*

provides a *prototype* of a function which may be used to provide code which uses the function with the minimal essential parts of the definition to permit compilation. In the prototype the *formal_argument_list* must contain the types, but need not contain names for the formal arguments.

In older versions of C++, different syntax was used to declare formal arguments, placing them after the closing parenthesis and before the function body.

Example:

```
void main(void);

void main(void)

{

}
```

## Constants and Variables

**Constants** are data items whose values cannot change while the program is running.

### Literals

Literals are used to express particular values within the source code of a program. We have already used these previously to give concrete values to variables or to express message we wanted our programs to print out, for example, when we wrote:

A=5;

The 5 in the piece of code was a literal constants.

Literal constants can be divided in integer numerals, floating-point numerals, character, strings and Boolean values.

#### Integer Numerals

1776

707

-273

They are numerical constants that identify decimal values. Notice that to express a numerical constant we do not have to write quotes (") nor any special character. There is no doubt that it is a constant: whenever we write 1776 in a program, we will be referring to the value 1776.

In addition to decimal numbers (those that all of us are used to use every day) C++ allows the use as literal constants of octal numbers (base 8) and hexadecimal numbers (base 16). If we want to express an octal number we have to precede it with a 0 (zero character). And in order to express a hexadecimal number we have to precede it with the characters 0x (zero, x). For example, the following literal constants are all equivalent to each other:

```
75          //decimal
0113        //octal
0x4b        //hexadecimal
```

All of these represent the same number: 75 expressed as a base-10 numeral, octal numeral and hexadecimal numeral, respectively.

#### Floating point numbers

They express numbers with decimals and/or exponents. They can include either a decimal point, an e character (that express "by ten at the xth height", where x is an integer value that follows the e character), or both a decimal point and an e character:

3.14159 // 3.14159

6.02e23 // $6.02 \times 10^{23}$

1.6e-19 // $1.6 \times 10^{-19}$

3.0 // 3.0

## Character and string literals

There also exist non-numerical constants, like:

'Z'

"Hello"

The 1st expression represent single character constant, and the 2nd expression represent string literals composed of several character. Notice that to represent a single character we enclose it between quotes (') and to express a string (which generally consists of more than one character) we enclose it between double quotes (").

When writing both single character and string literals, it is necessary to put the quotation marks surrounding them to distinguish them from possible variable identifiers or reserved keywords. Notice the difference between these two expressions:

x

'x'

x alone would refer to a variable whose identifier is x, whereas 'x' (enclosed within single quotation marks) would refer to the character constant 'x'.

Characters and string literals have certain peculiarities, like the escape codes. These are special characters that are difficult or impossible to express otherwise in the source code of program, like newline (\n) or tab (\t). All of them are preceded by a backslash (\). Here you have a list of some of such escape codes:

| | |
|---|---|
| \n | Newline |
| \r | Carriage return |
| \t | Tab |
| \v | Vertical tab |
| \b | Backspace |
| \f | Form feed (page feed) |
| \a | Alert (beep) |
| \' | Single quote (') |
| \" | Double quote (") |
| \? | Question mark (?) |
| \\ | Backslash (\) |

For example:

'\n'

'\t'

"Left \t righr"

"one\ntwo\nthree"

String literals can extend to more than a single line of code by putting a backslash sign (\) at the end of each unfinished line.

"string expressed in \

two lines"

**#define (directive)**

Defines a macro

Syntax: #define <id1>[ ( <id2>, ... ) ] <token string>

The #define directive defines a macro.

Macros provide a mechanism for token replacement with or without a set of formal, function-line parameters.

All subsequent instances of the identifier <id1> in the source text will be replaced by the text defined by <token string>.

If <id1> is followed IMMEDIATELY by a "(", the identifiers following the "(" (<id2>, etc.) are treated like parameters to a function instead of as part of <token string>.

All instances of <id2> in <token string> will be replaced with the actual text defined for <id2> when <id1> is referenced in the source.

To continue the definitions on another line, end the current line with \

```
Example:
 #define MAXINT 32767
 #define ctrl(x) ((x) \
   - 64) /* continued from preceding line */
```

It is legal but ill-advised to use Turbo C++ keywords as macro identifiers.

**Example**

```
#include <iostream.h>
#define x 5

int main()
{
    cout << "x:="<<x;   //x=5
    return 0;
}
```

In the above program, the value of x is fixed and can't be changed later

## Variables

As a programmer, you will frequently want your program to "remember" a value. For example, if your program requests a value from the user, or if it calculates a value, you will want to remember it somewhere so you can use it later. The way your program remembers things is by using **variables**. Variables represent storage locations in the computer's memory. For example:

```
int b;
```

This line says, "I want to create a space called b that is able to hold one integer value." A variable has a **name** (in this case, b) and a **type** (in this case, int, an integer). You can store a value in b by saying something like:

```
b = 5;
```

You can use the value in b by saying something like:

```
cout << b
The output will be:
5

While writing:
cout << "b"
The output will be:
b

and writing:
cout << "b=" << b
The output will be:
b=5

To display a text line No.= with the value of b use:
cout << "No.=" << b
The output will be:
No.=5
```

In C++, there are several standard types for variables:

- **Integer: int**
- **Floating point: float**
- **Character: char**

An int is a 2-byte integer value. A float is a 4-byte floating point value. A char is a 1-byte single character (like "a" or "A") or any other 8-bit quantity, when a character is stored in memory, it is actually the numeric code is stored. A string is declared as an array of characters.

There are a number of derivative types:

- **double** (8-byte floating point value)
- **short** (2-byte integer)
- **unsigned short** or **unsigned int** (positive integers, no sign bit)

## Data Types

| Type | Length | Range |
|---|---|---|
| unsigned char | 8 bits | 0 to 255 |
| char | 8 bits | -128 to 127 |
| enum | 16 bits | -32,768 to 32,767 |
| unsigned int | 16 bits | 0 to 65,535 |
| short int | 16 bits | -32,768 to 32,767 |
| int | 16 bits | -32,768 to 32,767 |
| unsigned long | 32 bits | 0 to 4,294,967,295 |
| long | 32 bits | -2,147,483,648 to 2,147,483,647 |
| float | 32 bits | 3.4 * (10**-38) to 3.4 * (10**+38) |
| double | 64 bits | 1.7 * (10**-308) to 1.7 * (10**+308) |
| long double | 80 bits | 3.4 * (10**-4932) to 1.1 * (10**+4932) |
| void | - | valueless |

Note: void data type specifies a valueless expression.

Examples:

```
char a='d';

int b=50;

float c=5.3;
```

Q) write a program to calculate the area of circle, radius=4.5

```
#include <iostream.h>
#include <conio.h>
#define pi 3.1415
int main()
{
 float r=4.5,area;
 clrscr();
 area=r*r*pi;
 cout << "Area of circle="<<area;
 getch();
}
```

## Arithmetic Operators

### 1- Plus and minus operators (+ and -)

+ cast-expression: Value of the operand after any required integral promotions.

- cast-expression: Negative of the value of the operand after any required integral promotions.

**Example: y = -y;**

**Example:**

**z=x+y;  c=a-b;**

### 2-Multiplicative operators ( * / % )

There are three multiplicative operators:

* (multiplication), / (division), % (modulus or remainder)

```
void main()
{
    int a=5;
    a=a%2;
    cout <<"a="<<a; //a=1 remainder of division a/2
 }
```

The / operator performs integer division if both operands are integers, and performs floating point division otherwise. For example:

```
void main()
{
    float a;
    a=10/3;
    cout <<"a="<<a;
}
```

This code prints out a floating point value since **a** is declared as type **float**, but **a** will be 3.0 because the code performed an integer division.

**Typecasting**

C++ allows you to perform type conversions on the fly. You do this especially often when using pointers. Typecasting also occurs during the assignment operation for certain types. For example, in the code above, the integer value was automatically converted to a float.

You do typecasting in C++ by placing the type name in parentheses and putting it in front of the value you want to change. Thus, in the above code, replacing the line **a=10/3;** with **a=(float)10/3;** produces 3.33333 as the result because 10 is converted to a floating point value before the division, also you can say a=10.0/3;

### Precedence of Operators

**Operator precedence** in C++ is also similar to that in most other languages. Division and multiplication occur first, then addition and subtraction. The result of the calculation 5+3*4 is 17, not 32, because the * operator has higher precedence than + in C++. You can use parentheses to change the normal precedence ordering: (5+3)*4 is 32. The 5+3 is evaluated first because it is in parentheses. We'll get into precedence later -- it becomes somewhat complicated in C++ once pointers are introduced.

## Input/ Output technique

Until now, the example programs of previous sections provide very little interaction with the user, if any at all. Using the standard input and output library, we will be able to interact with the user by printing messages on the screen and getting the user's input from the keyboard.

C++ uses a convenient abstraction called streams to perform input and output operations in sequential media such as the screen or the keyboard. A stream is an object where a program can either insert or extract characters to/from it. We do not really need to care about many specifications about the physical media associated with the stream – we only need to know it will accept or provide characters sequentially.

The standard C++ library include the header file iostream, where the standard input and output stream object are declared.

### Standard output (cout)

By default, the standard output of a program is the screen, and the C++ stream object defined to access it is cout.

cout is used in conjunction with the insertion operator, which is written as << (two "less than" signs).

```
cout << "output sentence";          //prints output sentence on screen

cout << 120;                        // print number 120 on screen

cout << x;                          // prints the content of x on screen
```

The << operator inserts the data that follows it into the stream preceding it. In the example above it inserted the constant string output sentence, the numerical constant 120 and variable x into the standard output stream cout. Notice that the sentence in the first instruction is enclosed between double quotes (") because it is a constant string of characters. Whenever we want to use constant strings of characters we must enclose them between quotes (") so that they can be clearly distinguished from variable names. For example, these two sentences have very different results:

```
cout << "Hello";            // prints Hello

cout << Hello;              // prints the content of Hello variable
```

The insertion operator (<<) may be used more than once in a single statement:

```
cout << " Hello," << "I am " << "a C++ statement";
```

This last statement would print the message Hello, I am a C++ statement on the screen. The utility of repeating the insertion operator (<<) is demonstrated when we want to print out a combination of variables and constants or more than one variable:

```
cout << "Hello, Iam "<< age<< " years old and my zipcode is "<< zipcode;
```

If we assume the age variable to contain the value 24 and the zipcode variable to contain 90064 the output of the previous statement would be:

```
Hello, I am 24 years old and my zipcode is 90064
```

It is important to notice that cout does not add a line break after its output unless we explicitly indicate it, therefore, the following statements:

```
cout << "This is a sentence."

cout << "This is another sentence.";
```

Will be shown on the screen one following the other without any line break between them:

```
This is a sentence. This is another sentence.
```

Even though we had written them in two different insertions into cout. In order to perform a line break on the output we must explicitly insert a new-line character into cout. In C++ a new-line character can be specified as \n (backslash ,n):

```
cout << "First sentence.\n";

cout << "Second sentence.\nThird sentence.";
```

This produces the following output:

```
First sentence.

Second sentence.

Third sentence.
```

Additionally, to add a new-line, you may also use the endl manipulator. For example:

```
cout << "First sentence."<<endl;

cout << "Second sentence."<<endl;
```

Would print out:

```
First sentence.

Second sentence.
```

The endl manipulator produces a newline character, exactly as the insertion of '\n' does, but it also has an additional behavior when it is used with buffered streams: the buffer is flushed. Anyway, cout will be an unbuffered stream in most cases, so you can generally use both the \n escape character and the endl manipulator in order to specify a new line without any difference in its behavior.

## Standard Input (cin)

The standard input device is usually the keyboard. Handing the standard input in C++ is done by applying the overloaded operator of extraction (>>) on the cin stream. The operator must be followed by the variable that will store the data that is going to be extracted from the stream. For example:

```
int age;

cin >> age;
```

The first statement declares a variable of type int called age, and the second one waits for an input from cim (the keyboard) in order to store it in this integer variable.

cin can only process the input from the keyboard once the return key has been pressed. Therefore, even if you request a single character, the extraction from cin will not process the input until the user presses return after the character has been introduced.

You must always consider the type of the variable that you are using as container with cin extractions. If you request an integer you will get an integer, if you request a character you will get a character and if you request a string of character you will get a string of character.

```
// I/O example

#include <iostream.h>

int main()

{

    int i;

    cout << "Please enter an integer value: ";

    cin >> i;

    cout << "The value you entered is " << i;

    cout << " and its double is " << i*2 << ".\n";

    return 0;

}
```

This produces the following output:

```
Please enter an integer value: 702

The value you entered is 702 and its double is 1404.
```

The user of a program may be one of the factors that generate errors even in the simplest programs that use cin (like the one we have just seen). Since if you request an integer value and the user introduce a name (which generally is a string of characters), the result may cause your program to misoperate since it is not what we were expecting from the user. So when

you use the data input provided by cin extractions you will have to trust that the user of your program will be cooperative and that he/she will not introduce his/her name or something similar when an integer value is requested. A little ahead, when we see the string stream class we will see a possible solution for the errors that can be caused by this type of user input.

You can also use cin to request more than one datum input from the user:

```
cin >> a >> b;
```

Is equivalent to:

```
cin >> a;
cin >> b;
```

```
Q) write a program to calculate the area of circle
#include <iostream.h>
#include <conio.h>
#define pi 3.1415
int main()
{
 float r,area;
 clrscr();
 cout << "radius=";
 cin >> r;
 area=r*r*pi;
 cout << "Area of circle="<<area;
 getch();
}
```

Q) exchange two numbers

```
#include <iostream.h>
#include <conio.h>
int main()
{
 int x, y,z;
 clrscr();
 cout <<"x=";
 cin >>x;
 cout <<"y=";
 cin >>y;
 z=x;
 x=y;
 y=z;
 cout <<"New x="<<x;
 cout <<"\nNew y="<<y;
 getch();
}
```

## Relational operators ( < > <= and >= )

The relational operators are used to compare relative values.

E1 < E2 Gives 1 (true) if the value of E1 is less than value of E2; otherwise, the result is 0 (false)

E1 <= E2 Gives 1 (true) if the value of E1 is less than or equal to the value of E2; otherwise, the result is 0 (false).

E1 > E2 Gives 1 (true) if the value of E1 is greater than the value of E2; otherwise, the result is 0 (false)

E1 >= E2 Gives 1 (true) if the value of E1is greater than or equal to the value of E2; otherwise, the result is 0 (false).

## Equality operators ( == and != )

The operators == and != are used to test for equality or inequality between arithmetic or pointer values, following rules similar to those for the relational operators.

The equality operators have lower precedence than the relational operators, however, and you can also compare certain pointer types not allowed with relational operations.

If E1 == E2 gives 1 (true), then either E1 and E2 point to the same function, or they are both null.

The expression E1 != E2 follows the same rules, except that the result is 1 (true) if the operands are unequal, and 0 (false) if the operands are equal.

## The if statement (keyword)

The first type of branching statement we will look at is the **if statement**. The if statement can cause other statements to execute only under certain conditions. An **if statement** has the form:

if ( <expression> ) <statement1>;

If <expression> is non-zero when evaluated, <statement1> is executed.

Example:
if (count < 50) count++;

Here is a simple C++ program demonstrating an if statement:

```
#include <iostream.h>

int main()
{
    int b;
    cout <<"Enter a value:";
    cin >> b;
    if (b < 0)
        cout <<"The value is negative";
    return 0;
}
```

This program accepts a number from the user. It then tests the number using an if statement to see if it is less than 0. If it is, the program prints a message `"The value is negative"`. Otherwise, the program is silent. The **(b < 0)** portion of the program is the Boolean expression. C++ evaluates this expression to decide whether or not to print the message. If the Boolean expression evaluates to **True**, then C++ executes the single line immediately following the if statement (or a block of lines within braces immediately following the if statement). If the Boolean expression is **False**, then C++ skips the line or block of lines immediately following the if statement.

If (condition)

{

Statement1;

Statement2;

.

.

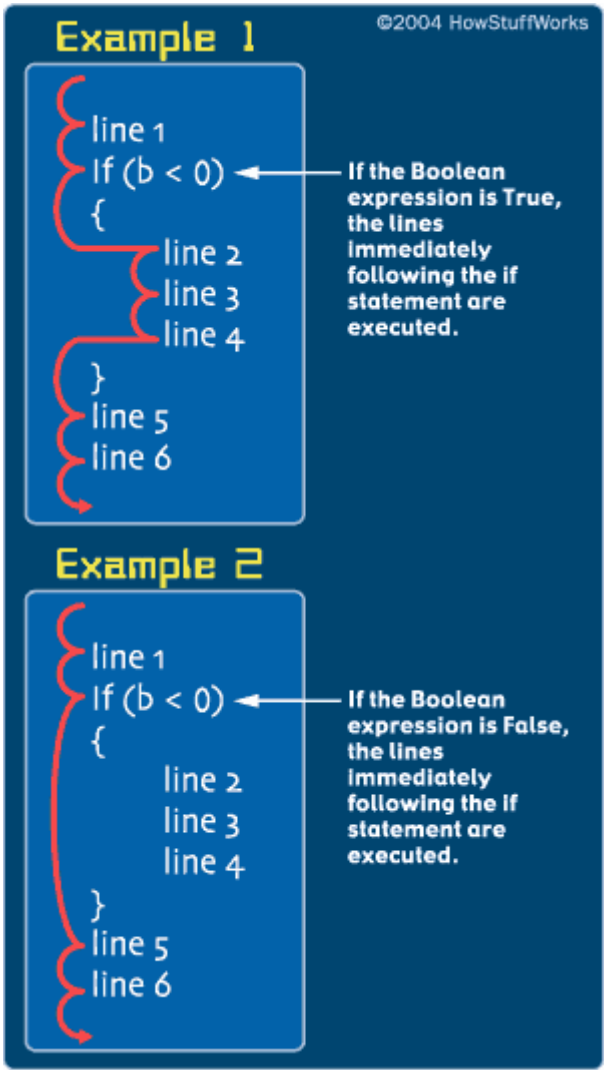
Statementn;

}

Example: to update the previous program, if the value of b is negative then add 50 to b then print the new value of b. The program will be:

```
#include <iostream.h>

int main()
{
    int b;
    cout << "Enter a value:";
    cin >> b;
    if (b < 0)
        {
         b+=50;
        cout <<"b="<<b;
        }
    return 0;
}
```



**Q) Write a program to check if the no. is positive, then print positive number**

```
#include <iostream.h>
#include <conio.h>
int main()
{
int x;
clrscr();
cout << "Number:=";
cin >> x;
 if (x > 0)
  cout << " --- Positive Number ---";
cout << "\n\nHit any key to continue";
getch();
return 0;
}
```

**Q) Write a program to check  if the student pass in English language**

```
#include <iostream.h>
#include <conio.h>
int main()
{
int x;
clrscr();
cout << "Degree:=";
cin >> x;
 if (x >=50)
  cout << " --- Pass ---";
cout << "\n\nHit any key to continue";
getch();
return 0;
}
```

**Q) Write a program to check the no., if it is even then print even number**

```
#include <iostream.h>
#include <conio.h>
int main()
{
int x;
clrscr();
cout << "Number:=";
cin >> x;
 if ((x % 2)==0)       // if (!(x % 2))
  cout << " --- Even Number ---";
cout << "\n\nHit any key to continue";
getch();
return 0;
}
```

**Q) Write a program to check if the no. accept division by 5 without remainder, then print that**

```
#include <iostream.h>
#include <conio.h>
int main()
{
int x;
clrscr();
cout << "Number:=";
cin >> x;
 if ((x % 5)==0)       // if (!(x % 5))
  cout << " --- Number accept division by 5 ---";
cout << "\n\nHit any key to continue";
getch()
return 0;
}
```

**Q) Write a program to solve the following equation**

y=x+1-y  when x>=20

```
#include <iostream.h>
#include <conio.h>
int main()
{
int x,y;
clrscr();
cout << "x:=";
cin >> x;
cout << "y:=";
cin >> y;

 if (x >=20)
  {
  y=x+1-y;
  cout << " y:="<<y;
  }
cout << "\n\nHit any key to continue";
getch();
return 0;
}
```

**The If –else statement**

 if ( <expression> ) <statement1>;
 else <statement2>;

If <expression> is non-zero when evaluated, <statement1> is executed. Otherwise <statement2> is executed if the expression is 0.

An optional else can follow an if statement, but no statements can come between an if statement and an else.

Example:

```
if (x < y)
  z = x;
else
  z = y;
```

The #if and #else preprocessor statements (directives) look similar to the if and else statements, but have very different effects.

They control which source file lines are compiled and which are ignored.

- In an **if statement**, *condition* is a value or an expression that is used to determine which code block is executed, and the curly braces act as "begin" and "end" markers.

```
if (condition)
{
  // code to execute if condition is true
}
else
{
  // code to execute if condition is false
}
```

Here is a full C++ program as an example:

```
#include <iostream.h>  //include this file for cout
int main() {
  // define two integers
  int x = 3;
  int y = 4;

  //print out a message telling which is bigger
  if (x > y)
    cout << "x is bigger than y" << endl;

  else
    cout << "x is not bigger than y" << endl;

  return 0;
}
```

In this case *condition* is equal to "(x > y)" which is equal to "(3 > 4)" which is a *false* statement. So the code within the **else** clause will be executed. The output of this program will be:

```
x is not bigger than y
```

If instead the value for x was 6 and the value for y was 2, then *condition* would be "(6 > 2)" which is a *true* statement and the output of the program would be:

```
x is bigger than y
```

**Q) Write a program to check if the no. is positive or not positive**

```
#include <iostream.h>
#include <conio.h>
int main()
}
int x;
clrscr();
cout << "Number:=";
cin >> x;
 if (x > 0)
  cout << " --- Positive Number ---";
 else
  cout << " --- The number is not positive ---";

cout << "\n\nHit any key to continue";
getch();
return 0;
}
```

**Q) Write a program to check  if the student pass in English language or not**

```
#include <iostream.h>
#include <conio.h>
int main()
}
int x;
clrscr();
cout << "Degree:=";
cin >> x;
 if (x >=50)
  cout << " --- Pass ---";
else
  cout << " --- Fail ---";

cout << "\n\nHit any key to continue";
getch();
return 0;
}
```

**Q) Write a program to check if the NO. is even or odd**

```
#include <iostream.h>
#include <conio.h>
int main()
}
int x;
clrscr();
cout << "Number:=";
cin >> x;
 if ((x % 2)==0)       // if (!(x % 2))
  cout << " --- Even Number ---";
else
  cout << " --- Odd Number ---";

cout << "\n\nHit any key to continue";
getch();
return 0;
}
```

**Q) Write a program to check if the no. accept division by 5 without remainder, then print that**

```
#include <iostream.h>
#include <conio.h>
int main()
}
int x;
clrscr();
cout << "Number:=";
cin >> x;
 if ((x % 5)==0)       // if (!(x % 5))
cout << " --- Number accept division by 5 without remainder ---";
else
cout << " --- Number don't accept division by 5 without remainder ---";

cout << "\n\nHit any key to continue";
getch()
return 0;
}
```

**Q) Write a program to solve the following equations**

Y=x+1-y  when x>=20

Y=(x-1)*y  when x<20

```
#include <iostream.h>
#include <conio.h>
int main()
{
int x,y;
clrscr();
cout << "x:=";
cin >> x;
cout << "y:=";
cin >> y;

 if (x >=20)
  {
  y=x+1-y;
  cout << " y:="<<y;
  }
 else
 {
   y=(x-1)*y;
  cout << " y:="<<y;

  }
cout << "\n\nHit any key to continue";
getch();
return 0;
}
```

**Note:** It is better to write the following program rather the previous one

```
#include <iostream.h>
#include <conio.h>
int main()
{
int x,y;
clrscr();
cout << "x:=";
cin >> x;
cout << "y:=";
cin >> y;

 if (x >=20)
  y=x+1-y;
else
   y=(x-1)*y;

cout << " y:="<<y;//because in both cases, y is the output
cout << "\n\nHit any key to continue";
getch();
return 0;
}
```

## If/else if/ else statement

**The (if/else if) statement is a chain of if statements. They perform their tests, one after the other, until one of them is found to be true.**

**This construction is like a chain of (if/else) statements. The else part of one statement is linked to the if part of another. When put together this way, the chain of if/else becomes one long statement.**

```
if (condition1)
{
   // code to execute if condition1 is true
}
else if (condition2)
{
//code to execute if condition1=false but condition2=true
}
Else
{
//code to execute if both condition1 and condition2=false
}
```

- Or it can be written as:

```
if (condition1)
{
   // code to execute if condition1 is true
}
Else
{
 if (condition2)
{
//code to execute if condition1=false but condition2=true
}
Else
{
//code to execute if both condition1 and condition2=false
}
}
```

Here's slightly more complex example:

```
#include <iostream.h>

int main()
{
    int b;
    cout << "Enter a value:";
    cin >> b;
    if (b < 0)
        cout << "The value is negative";
    else if (b == 0)
        cout << "The value is zero";
    else
        cout << "The value is positive";
    return 0;
}
```

## Q) Write a program to solve the following equations

y=x+1-y when x>20

y=(x-1)*y when x<20

y=x when x=20

```
#include <iostream.h>
#include <conio.h>
int main()
{
int x,y;
clrscr();
cout << "x:=";
cin >> x;
cout << "y:=";
cin >> y;

 if (x >20)
  y=x+1-y;
 else if (x<20)
   y=(x-1)*y;
  else
   y=x;

  cout << " y:="<<y;
  cout << "\n\nHit any key to continue";
getch();
return 0;
}
```

## Q) Write a program to solve the following equations

a=a/10 when a>b

b=a/3 when a<b

a=a*b when a=b

```
#include <iostream.h>
#include <conio.h>
int main()
{
float a,b;
clrscr();
cout << "a:=";
cin >> a;
cout << "b:=";
cin >> b;

 if (a >b)
 {
  a/=10.0;
  cout << " a:="<<a;
 }
 else if (a<b)
  {
   b=a/3.0;
   cout << " b:="<<b;
  }
  else
  {
   a=a*b;
   cout << " a:="<<a;
  }
  cout << "\n\nHit any key to continue";
getch();
return 0;
}
```

## Nested if

Nested if used when we need to test condition after another condition happened. It has the form:

```
if (condition1)
{
    If (condition2)
    {
    }
  Else
    {
    }
}
Else
{
}
```

**Q) Write a program to check if the no. is even and accept division by 5 without remainder**

```
#include <iostream.h>
#include <conio.h>
int main()
{
int a;
clrscr();
cout << "Number:=";
cin >> a;

 if ((a % 2)==0)
  if ((a % 5)==0)
  cout << " even number accept division by 5";

 cout << "\n\nHit any key to continue";
getch();
return 0;
}
```

**Q) Write a program to check if the no. is odd and negative**

```
#include <iostream.h>
#include <conio.h>
int main()
{
int a;
clrscr();
```

```
cout << "Number:=";
cin >> a;

 if ((a % 2)!=0)
  if (a<0)
  cout << "--- negative odd number ---";

 cout << "\n\nHit any key to continue";
getch();
return 0;
}
```

**Q) Write a program to solve the following equations**

When a>=5

b=a-1 when b>=20

b=a+1 when b<20

When a<5

b=a*5 when b>=20

b=a/5 when b<20

```
#include <iostream.h>
#include <conio.h>
int main()
{
float a,b;
clrscr();
cout << "a:=";
cin >> a;
cout << "b:=";
cin >> b;

 if (a >=5)
  if (b>=20)
   b=a-1;
  else
   b=a+1;
 else
  if (b>=20)
   b=a*5;
  else
   b=a/5.0;

  cout << " b:="<<b;
  cout << "\n\nHit any key to continue";
getch();
return 0;
}
```

## Logical operator

### 1- Logical operator (&&, ||)

Turbo C++ offers these logical operators:

&& logical AND
|| logical OR

Syntax:
logical-AND-expr && inclusive-OR-expression
logical-OR-expr || logical-AND-expression

In these expressions, both operands must be of scalar type.

E1 && E2 E1 || E2

The usual arithmetical conversions are performed on E1 and E2.

Unlike the bitwise operators, && and || guarantee left-to-right evaluation.

E1 is evaluated first; if E1 is zero, E1 && E2 gives 0 (false), and E2 is not evaluated.

With E1 || E2, if E1 is nonzero, E1 || E2 gives 1 (true), and E2 is not evaluated.

### 2- Logical negation operator !

In the expression

! cast-expression

the cast-expression operand must be of scalar type.

The result is of type int and is the logical negation of the operand:

- 0 if the operand is nonzero
- 1 if the operand is 0

The expression !E is equivalent to (0 == E).

### If with logical operator

Here is a more complicated Boolean expression:

```
if ((x==y) && (j>k))
    z=1;
else
    q=10;
```

This statement says, "If the value in variable x equals the value in variable y, and if the value in variable j is greater than the value in variable k, then set the variable z to 1, otherwise set the variable q to 10." You will use if statements like this throughout your C++ programs to

make decisions. In general, most of the decisions you make will be simple ones like the first example; but on occasion, things get more complicated.

Notice that C++ uses = = to **test for equality**, while it uses = to **assign a value** to a variable. The **&&** in C++ represents a Boolean AND operation.

Boolean expressions evaluate to integers in C++, and integers can be used inside of Boolean expressions. The integer value 0 in C++ is False, while any other integer value is True. The following is legal in C++:

```
#include <iostream.h>

int main()
{
    int a;

    cout << "Enter a number:";
    cin >> a;
    if (a)
    {
        cout << "The value is True";
    }
    return 0;
}
```

If **a** is anything other than 0, the printf statement gets executed.

In C++, a statement like **if (a=b)** means, "Assign **b** to **a**, and then test **a** for its Boolean value." So if **a** becomes 0, the if statement is False; otherwise, it is True. The value of **a** changes in the process. This is not the intended behavior if you meant to type = = (although this feature is useful when used correctly), so be careful with your = and = = usage.

**Q) Write a program to check if the no. is even and accept division by 5 without remainder**

```
#include <iostream.h>
#include <conio.h>
int main()
{
int a;
clrscr();
cout << "Number:=";
cin >> a;

 if ((!(a % 2)) && (!(a % 5)))
  cout << " even number accept division by 5";

 cout << "\n\nHit any key to continue";
getch();
return 0;
}
```

**Q) Write a program to check if the no. is odd and negative**

```
#include <iostream.h>
#include <conio.h>
int main()
{
int a;
clrscr();
cout << "Number:=";
cin >> a;

 if ((a % 2) && (a<0))
  cout << "--- negative odd number ---";

 cout << "\n\nHit any key to continue";
getch();
return 0;
}
```

**Q) Write a program to solve the following equations**

When a>=5

b=a-1 when b>=20

b=a+1 when b<20

When a<5

b=a*5 when b>=20

b=a/5 when b<20

```
#include <iostream.h>
#include <conio.h>
int main()
{
float a,b;
clrscr();
cout << "a:=";
cin >> a;
cout << "b:=";
cin >> b;

 if ((a >=5) && (b>=20))
   b=a-1;
 else if ((a >=5) && (b<20))
   b=a+1;
 else if ((a<5) && (b>=20))
   b=a*5;
 else if ((a<5) && (b<20))
   b=a/5.0;

  cout << " b:="<<b;
  cout << "\n\nHit any key to continue";
getch();
return 0;
}
```

**Q) Write a program to check the mark of the student**

```
#include <iostream.h>
#include <conio.h>
int main()
{
int a;
clrscr();
cout << "Mark:=";
cin >> a;

 if (a >=90)
    cout << " Excellent";
 else if ((a >=80) && (a<90))
     cout << " Very good";
 else if ((a >=70) && (a<80))
     cout << " Good";
 else if ((a >=60) && (a<70))
     cout << " Medium";
 else if ((a >=50) && (a<60))
     cout << " Accepted";
 else
     cout << " Fail";

  cout << "\n\nHit any key to continue";
getch();
return 0;
}
```

## The switch statement

The next branching statement is called a **switch statement**. A **switch statement** is used in place of many **if statements**.

If you want to test whether a variable takes one of a series of values, it's easier to use a switch statement than an if statement.

switch (<variable>) {

case <value 1>:

<one or more statements>

break;
case <value 2>:

<one or more statements>

break;
default:
<one or more statements>

break;
} //end switch

**switch, case, and default (keywords)**

Branches control

Syntax:
- switch ( <expression> ) <statement>
- case <constant expression> :
- default :

**switch**

Causes control to branch to one of a list of possible statements in the block defined by <statement>.

The branched-to statement is determined by evaluating <expression>, which must return an integral type.

**case**
The list of possible branch points within <statement> is determined by preceding sub statements with

case <constant expression> :

where <constant expression> must be an int and must be unique.

Once a value is computed for <expression>, the list of possible <constant expression> values determined from all case statements is searched for a match.

If a match is found, execution continues after the matching case statement and continues until a break statement is encountered or the end of <statement> is reached.

**default**

If a match is not found and the "default :" statement prefix is found within <statement>, execution continues at this point.

The **break** keyword means "jump out of the switch statement, and do not execute any more code." To show how this works, examine the following piece of code:

```
int value = 0;
switch(input){
  case 1:
    value=value+4;
  case 2:
    value=value+3;
  case 3:
    value=value+2;
  default:
    value=value+1;
}
```

If *input* is 1 then 4 will be added to *value*. Since there is no **break** statement, the program will go on to the next line of code which adds 3, then the line of code that adds 2, and then the line of code that adds 1. So *value* will be set to 10!

The code that was intended was probably:

```
int value = 0;
switch(input){
  case 1:
    value=value+4;
    break;
  case 2:
    value=value+3;
    break;
  case 3:
    value=value+2;
    break;
  default:
    value=value+1;
}
```

**Q) Write a program to calculate the area or circumference of rectangle using menu**

```
#include <iostream.h>
#include <conio.h>
int main()
{
int a;
int l,w,r;
clrscr();
cout << "*********************************\n";
cout << "* 1: Area of rectangle          *\n";
cout << "* 2: Circumference of rectangle *\n";
cout << "*********************************\n\n";

cout << "Length:=";
cin >> l;
cout << "Width:=";
cin >> w;

cout << "\nChoice:=";
cin >> a;
switch (a)
{
 case 1:
   r=l*w;
   cout << "\n   Area of rectangle:="<<r;
 break;
 case 2:
   r=(l+w)*2;
   cout << "\n   Circumference of rectangle:="<<r;
 break;
}

  cout << "\n\nHit any key to continue";
getch();
return 0;
}
```

*** You can write the same previous program depending on character**

```
#include <iostream.h>
#include <conio.h>
#include <ctype.h>
int main()
{
char a;
int l,w,r;
clrscr();
cout << "*********************************\n";
cout << "* A: Area of rectangle          *\n";
cout << "* C: Circumference of rectangle *\n";
cout << "*********************************\n\n";

cout << "Length:=";
cin >> l;
cout << "Width:=";
cin >> w;

cout << "\nChoice:=";
cin >> a;
switch (toupper(a)) // to convert the character to upper case
{
 case 'A':
   r=l*w;
   cout << "\n   Area of rectangle:="<<r;
 break;
 case 'C':
   r=(l+w)*2;
   cout << "\n   Circumference of rectangle:="<<r;
 break;
 default:
  cout << "\n   Error Choice";
}

  cout << "\n\nHit any key to continue";
getch();
return 0;
}
```

## The iteration (looping) statement : (if with counter)

Looping is the repetition of a statement or block of statements in a program.

**goto (keyword)**

Transfers control

Syntax: goto <identifier> ;

Control is unconditionally transferred to the location of a local label specified by identifier.

Example:

```
Again:
;
.
goto Again;
```

NOTE: Labels must be followed by a statement (;).

**Q) Write a program to print number from 1..5 in ascending order**

```
#include <iostream.h>
#include <conio.h>
int main()
{
int i;
clrscr();
  i=1;
 lab:
    cout <<i<<" ";
    i=i+1;
    if (i<=5) goto lab;
cout << "\n\nHit any key to continue"; //i=6
getch();
return 0;
}
```

**Q) Write a program to print number from 5..1 in descending order**

```
#include <iostream.h>
#include <conio.h>
int main()
{
int i;
clrscr();
  i=5;
 lab:
    cout <<i<<" ";
    i=i-1
    if (i>=1) goto lab;
cout << "\n\nHit any key to continue"; //i=0
getch();
return 0;
}
```

## For (keyword)

For loop

Syntax: for ( [<expr1>] ; [<expr2>] ; [<expr3>] ) <statement>

<statement> is executed repeatedly UNTIL the value of <expr2> is 0.

BEFORE the first iteration, <expr1> is evaluated. This is usually used to initialize variables for the loop.

AFTER each iteration of the loop, <expr3> is evaluated. This is usually used to increment a loop counter.

In C++, <expr1> can be an expression or a declaration.

The scope of any declared identifier extends to the end of the control statement only.

All the expressions are optional. If <expr2> is left out, it is assumed to be 1.

The following statement is used for infinite loop: ***for ( ;;)***

**Q) Write a program to print number from 1..5 in ascending order**

```
#include <iostream.h>
#include <conio.h>
int main()
{
clrscr();
  for (int i=1;i<=5;i=i+1)
    cout <<i<<" ";

   //out the loop i=6
cout << "\n\nHit any key to continue";
getch();
return 0;
}
```

**Q) Write a program to print number from 5..1 in descending order**

```
#include <iostream.h>
#include <conio.h>
int main()
{
clrscr();
  for (int i=5;i>=1;i=i-1)
    cout <<i<<" ";
   //out the loop i=0
cout << "\n\nHit any key to continue";
getch();
return 0;
}
```

**Q) Write a program to find the factorial of any number**

```
#include <iostream.h>
#include <conio.h>
int main()
{
int n;
long int f;
clrscr();
cout << "Number:=";
cin >> n;
 if (n>=1)
  {
  for (int i=1,f=1;i<=n;i=i+1)
    f=f*i;
  cout <<"Factorial number:="<<f;
  }

cout << "\n\nHit any key to continue";
getch();
return 0;
}
```

**Q) Write a program to find the factorial of 3 different numbers**

```
#include <iostream.h>
#include <conio.h>
int main()
{
int n;
long int f;
clrscr();
for (int j=1;j<=3;j=j+1)
{
cout << "Number:=";
cin >> n;
 if (n>=1)
  {
  for (int i=1,f=1;i<=n;i=i+1)
    f=f*i;
  cout <<"Factorial number:="<<f<<"\n\n";
  }
}
cout << "\n\nHit any key to continue";
getch();
return 0;
}
```

**Q) Write a program to solve the following equation:**

$$\boldsymbol{sum = 4 + \frac{7}{2} + \frac{5}{3} + \frac{3}{4}}$$

```
#include <iostream.h>
#include <conio.h>
int main()
{
float sum=4.0;
clrscr();
for (int i=2,j=7;i<=4;i=i+1,j=j-2)
 {
    cout <<"i="<<i<<" j="<<j<<"\n"; // for test
```

```
    sum=sum+(float )j/i;
  }
    cout<<"sum="<<sum;
cout << "\n\nHit any key to continue";
getch();
return 0;
}
```

**Q) Write a program to solve the following equation:**

$$sum = 4 - \frac{7}{2} + \frac{5}{3} - \frac{3}{4}$$

```
#include <iostream.h>
#include <conio.h>
int main()
{
float sum=4.0;
clrscr();
for (int i=2,j=7,s=-1;i<=4;i=i+1,j=j-2,s=-s)
 {
    cout <<"i="<<i<<" j="<<j<<" s="<<s<<"\n"; // for test
    sum=sum+s*(float )j/i;
  }
    cout<<"sum="<<sum;
cout << "\n\nHit any key to continue";
getch();
return 0;
}
```

**Q) Write a program to solve the following equation:**

$$sum = 3 + \frac{2!}{3} + \frac{5!}{9} + \frac{8!}{27}$$

```
#include <iostream.h>
#include <conio.h>
int main()
{
float sum=3.0;
long int f;
clrscr();
for (int n=2,j=3;n<=8;n=n+3,j=j*3)
 {
    f=1;
    cout <<"n="<<n<<" j="<<j<<"\n";
    for (int i=1;i<=n;i=i+1)
     f=f*i;
 cout<<"Number="<<n<<" Factorial number="<<f<<"\n\n"; // for test
    sum=sum+(float )f/j;
  }
    cout<<"sum="<<sum;
cout << "\n\nHit any key to continue";
getch();
return 0;
}
```

**Q) Write a program to find even numbers among group of numbers**

```
#include <iostream.h>
#include <conio.h>
int main()
{
int n,n1;
clrscr();
cout << "Number of numbers:=";
cin >> n;

for (int j=1;j<=n;j=j+1)
{
cout << "Number:=";
cin >> n1;
 if (!(n1%2))
  cout <<" --- Even number --- \n\n";
 }
cout << "\n\nHit any key to continue";
getch();
return 0;
}
```

**Q) Write a program to find the number and summation of even numbers which accept division by 4 without remainder among group of numbers**

```
#include <iostream.h>
#include <conio.h>
int main()
{
int n,n1;
clrscr();
cout << "Number of numbers:=";
cin >> n;

for (int j=1,n2=0,sum=0;j<=n;j=j+1)
{
cout << "Number:=";
cin >> n1;
 if ((!(n1%2)) && (!(n1%4)))
  {
   sum=sum+n1;
   n2=n2+1;
  }
}
cout <<"\nSum. of even numbers which accept division by 4:="<<sum;
cout<<"\nNo. of even numbers which accept division by 4:="<<n2;
cout << "\n\nHit any key to continue";
getch();
return 0;
}
```

**Q) Write a program to find largest number between 5 different numbers**

```
#include <iostream.h>
#include <conio.h>
int main()
{
int n,large;
clrscr();
cout << "Number1:=";
cin >> n;
large=n;

for (int i=2;i<=5;i=i+1)
{
cout << "Number"<<i<<":=";
cin >> n;
 if (n>large)
   large=n;
}
cout <<"\nLargest No.:="<<large;
cout << "\n\nHit any key to continue";
getch();
return 0;
}
```

**Q) Write a program to find smallest number between 5 different numbers**

```
#include <iostream.h>
#include <conio.h>
int main()
{
int n,small;
clrscr();
cout << "Number1:=";
cin >> n;
small =n;

for (int i=2;i<=5;i=i+1)
{
cout << "Number"<<i<<":=";
cin >> n;
 if (n < small)
   small =n;
}
cout <<"\n smallest No.:="<< small;
cout << "\n\nHit any key to continue";
getch();
return 0;
}
```

## While(keyword)

Repeats execution
Syntax: while ( <expression> ) <statement>
<statement> is executed repeatedly as long as the value of <expression> remains non-zero. The test takes place before each execution of the <statement>.

You'll find that **while** statements are just as easy to use as if statements. For example:

```
while (a < b)
{
    cout << a;
    a = a + 1;
}
```

This causes the two lines within the braces to be executed repeatedly until **a** is greater than or equal to **b**. The while statement in general works as illustrated to the right.

The **for loop** in C++ is simply a shorthand way of expressing a while statement. For example, suppose you have the following code in C++:

```
x=1;
while (x<10)
{

    X=x+1;
}
```

You can convert this into a for loop as follows:

```
for(x=1; x<10; x=x+1)
{

}
```

Note that the while loop contains an initialization step (**x=1**), a test step (**x<10**), and an increment step (**x++**). The for loop lets you put all three parts onto one line, but you can put anything into those three parts. For example, suppose you have the following loop:

```
a=1;
b=6;
while (a < b)
{
    a=a+1;
    cout << a;
}
```

You can place this into a for statement as well:

```
for (a=1,b=6; a < b; a=a+1,cout <<a);
```

It is slightly confusing, but it is possible. The **comma operator** lets you separate several different statements in the initialization and increment sections of the for loop (but not in the test section). Many C++ programmers like to pack a lot of information into a single line of C++ code; but a lot of people think it makes the code harder to understand, so they break it up.

**Q) Write a program to print number from 1..5 in ascending order**

```
#include <iostream.h>
#include <conio.h>
int main()
{
int i;
clrscr();

     i=1;
  while (i<=5)
    {
     cout <<i<<" ";
     i=i+1;
    }

cout << "\n\nHit any key to continue";
getch();
return 0;
}
```

**Q) Write a program to print number from 5..1 in descending order**

```
#include <iostream.h>
#include <conio.h>
int main()
{
int i;
clrscr();

     i=5;
  while (i>=1)
    {
     cout <<i<<" ";
     i=i-1;
    }

cout << "\n\nHit any key to continue";
getch();
return 0;
}
```

**Q) Write a program to search for first number 6 between 5 different numbers**

```
#include <iostream.h>
#include <conio.h>
int main()
{
int n,i;
clrscr();

     cout <<"Number:=";
     cin>> n;
     i=1;
  while ((n!=6) && (i<=4))
    {
     i=i+1;
     cout <<"Number:=";
     cin>> n;
    }

     if (n==6)
       cout <<" --- Number 6 is found ---";
cout << "\n\nHit any key to continue";
getch();
return 0;
}
```

**Q) Write a program to find the summation of numbers of group of numbers and stop when the result exceed 50 :**

```
#include <iostream.h>
#include <conio.h>
int main()
{
int sum=0,n;
clrscr();

  while (sum<=50)
    {
     cout <<"Number:=";
     cin>> n;
     sum=sum+n;
     }
     cout <<"Sum.:="<<sum;
cout << "\n\nHit any key to continue";
getch();
return 0;
}
```

**Q) Write a program to find the summation of the following series and stop when the result exceed 7 :** $\boldsymbol{sum = 4 + \frac{1}{2} + \frac{3}{3} + \frac{5}{4}}$ **….**

```
#include <iostream.h>
#include <conio.h>
int main()
{
float sum=4;
int i=1,j=2;
clrscr();

  while (sum<=7)
    {
     sum=sum+(float )i/j;
     cout <<"j:="<<j<<" i="<<i<<"\n"; // for test
     i=i+2;
     j=j+1;
     }
     cout <<"Sum.:="<<sum;
cout << "\n\nHit any key to continue";
getch();
return 0;
}
```

## Notes:

```
While (condition)
{
        While (condition)
        {
        }
}
// or
While (condition)
{
            for ( expression_1; expression_2; expression_3 )
           {
           }
}
// or
  for ( expression_1; expression_2; expression_3 )
 {
            While (condition)
            {
             }
}
```

## Do…while loop

Syntax: do <statement> while ( <expression> );

<statement> is executed repeatedly as long as the value of <expression> remains non-zero.

The test takes place AFTER each execution of the <statement>.

Example1:

```
i = 1; n = 1;
do {
  n =n* i;
  i=i+1;
} while (i <= factorial);
```

Example2:

```
do
{
    cout << a;
    a = a + 1;
}
while (a < b);
```

**Q) Write a program to print number from 1..5 in ascending order**

```
#include <iostream.h>
#include <conio.h>
int main()
{
int i;
clrscr();

     i=1;
   do
    {
     cout <<i<<" ";
     i=i+1;
    } while (i<=5);

cout << "\n\nHit any key to continue";
getch();
return 0;
}
```

**Q) Write a program to print number from 5..1 in descending order**

```
#include <iostream.h>
#include <conio.h>
int main()
{
int i;
clrscr();

     i=5;
   do
    {
     cout <<i<<" ";
     i=i-1;
    } while (i>=1);

cout << "\n\nHit any key to continue";
getch();
return 0;
}
```

**Q) Write a program to calculate the area or circumference of rectangle using menu, add choice 3 for exit**

```
#include <iostream.h>
#include <conio.h>
int main()
{
int a;
int l,w,r;
clrscr();
cout << "Length:=";
cin >> l;
cout << "Width:=";
cin >> w;

do
{
clrscr();
cout << "*********************************\n";
cout << "* 1: Area of rectangle          *\n";
cout << "* 2: Circumference of rectangle *\n";
cout << "* 3: Exit                       *\n";
cout << "*********************************\n\n";

cout << "\nChoice:=";
cin >> a;
switch (a)
{
 case 1:
   r=l*w;
   cout << "\n   Area of rectangle:="<<r;
 break;
 case 2:
   r=(l+w)*2;
   cout << "\n   Circumference of rectangle:="<<r;
 break;
 case 3:
  cout << "\n\nGood bye at exit";
  break;
}
 cout << "\n\nHit any key to continue";
    getch();

} while (a!=3);
    return 0;

}
```

**Exit**

exit terminates the program

Remarks:

exit terminates the calling process. Before termination, exit does the following:

- closes all files
- writes buffered output (waiting to be output)
- calls any registered "exit functions" (posted with at exit)

**Q) Write a program to calculate the area or circumference of rectangle using menu, add choice 3 for exit**

```
#include <iostream.h>
#include <conio.h>
#include <stdlib.h>
int main()
{
int a;
int l,w,r;
clrscr();
cout << "Length:=";
cin >> l;
cout << "Width:=";
cin >> w;
for (;;)
{
clrscr();
cout << "*********************************\n";
cout << "* 1: Area of rectangle          *\n";
cout << "* 2: Circumference of rectangle *\n";
cout << "* 3: Exit                       *\n";
cout << "*********************************\n\n";
cout << "\nChoice:=";
cin >> a;
switch (a)
{
 case 1:
   r=l*w;
   cout << "\n   Area of rectangle:="<<r;
 break;
 case 2:
   r=(l+w)*2;
   cout << "\n   Circumference of rectangle:="<<r;
 break;
 case 3:
    cout << "\n\nGood bye at exit, hit any key to continue";
    getch();
    exit (0);
break;
}
 cout << "\n\nHit any key to continue";
    getch();
}

}
```

## Break (keyword)

Passes control

Syntax: break ;

The break statement causes control to pass to the statement following the innermost enclosing while, do, for, or switch statement. It causes a loop to terminate early. When it is encountered, the loop stops and the program jump to the statement immediately following the loop.

**Q) Write a program to search for first number 6 between 5 different numbers**

```
#include <iostream.h>
#include <conio.h>
int main()
{
int n;
clrscr();
  for (int i=1;i<=5;i=i+1)
    {
     cout <<"Number:=";
     cin>> n;
     if (n==6)
      {
       cout <<" --- Number 6 is found ---";
       break;
      }
    }
cout << "\n\nHit any key to continue";
getch();
return 0;
}
```

**Continue** (keyword)

Syntax: continue ;
Causes control to pass to the end of the innermost enclosing while, do, or for statement, at which point the loop continuation condition is re-evaluated. It causes a loop to stops its current iteration and begin the next one. When continue is encountered, all the statements in the body of the loop that appear after it are ignored, and the loop prepares for the next iteration.

**Q) Write a program to find the summation of 4 different numbers except number 3 or 5**

```
#include <iostream.h>
#include <conio.h>
int main()
{
int n;
clrscr();
  for (int i=1,sum=0;i<=4;i=i+1)
    {
     cout <<"Number:=";   cin>> n;
     if ((n==3) || (n==5))
       continue;
    sum=sum+n;
    }
    cout<<"\n\nSum.:="<<sum;
cout << "\n\nHit any key to continue"; getch();
return 0;
}
```

**Math. Functions**

**pow (real) <math.h>**

Power function, x to the y
Declaration:
double pow(double x, double y);

**Q) Write a program to calculate the cubic number for 4 different numbers**

```
#include <iostream.h>
#include <conio.h>
#include <math.h>
int main()
{
double n,c;
clrscr();

  for (int i=1;i<=4;i=i+1)
    {
     cout <<"Number:=";
     cin>> n;
     c=pow(n,3);
    cout <<"Cubic Number:="<<c;
    }
cout << "\n\nHit any key to continue";
getch();
return 0;
}
```

**cos, sin, tan (real) <math.h>**

Cosine, sine, and tangent functions

Declaration:
double cos(double x);
double sin(double x);
double tan(double x);

Remarks:

- cos compute the cosine of the input value
- sin compute the sine of the input value
- tan calculate the tangent of the input value

Angles are specified in radians.

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

int main(void)
{
   double result, x;

   x = 90;
   result = sin(x*3.14/180);
   cout <<"The sin of “<<x<<” is "<< result;
//the oputput is: The sin of 90.00 is 1.00
   return 0;
}
```

**acos, asin, atan, atan2 (real) <math.h>**

Arc cosine, arc sine, and arc tangent functions

Declaration:

double acos(double x);
double asin(double x);
double atan(double x);
double atan2(double y, double x);

Remarks:

- acos of a real value compute the arc cosine of that value
- asin of a real value compute the arc sine of that value
- atan calculate the arc tangent of the input value
- atan2 also calculate the arc tangent of the input value

Real arguments to acos, and asin must be in the range -1 to 1.

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

int main(void)
{
   double result;
   double x = 1.00;

   result = asin(x)/(3.14/180);
  cout <<"The arc sin of “<<x<<” is "<< result;
//the oputput is: The arc sin of 1.00 is 90.00

   return(0);
}
```

**exp (real) <math.h>**

- Real exp calculates e to the xth power

Declaration:

double exp(double x);

Remarks:

exp calculates the exponential function e**x.

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

int main(void)
{
   double result;
   double x = 4.0;

   result = exp(x);
cout <<" 'e' raised to the power of "<<x<<" (e ^ "<<x<<") = "<< result;
      return 0;
}
The output is: 'e' raised to the power of 4.000000e^4.000000)=54.598150
```

**sqrt (real) < math.h >**

Calculates square root

Declaration:
- Real: double sqrt(double x);
  long double sqrtl(long double @E(x));

Remarks:
sqrt calculates the positive square root of the input value.

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

 int main(void)
 {
   double x = 4.0, result;

   result = sqrt(x);
   cout <<"The square root of  "<<x<<" is "<< result;
   //the oputput is: The square root of 4.000000 is 2.000000
         return 0;
}
```

**abs <math.h, stdlib.h, complex.h>**
**fabs < math.h >**
**labs < math.h, stdlib.h >**

- abs (a macro) gets the absolute value of an integer
- fabs calculate the absolute value of a floating-point number
- labs calculates the absolute value of a long number

Declaration:
- abs
  int abs(int x);
- double fabs(double x);
- long int labs(long int x);

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

int main(void)
{
  int number = -1234;

  cout <<"number: "<<number<<" absolute value: " << abs(number);
  //the oputput is: number: -1234 absolute value: 1234
  return 0;
}
```

**randomize <stdlib.h>**

Initializes random number generator. time.h must included

**random < stdlib.h >**

Returns a random number: random(num);
random returns a random number between 0 and (num-1).

```
#include <stdlib.h>
#include <iostream.h>
#include <time.h>
/* prints a random number in the range 0 to 99 */
int main(void)
{
   randomize();
   cout <<"Random number in the 0-99 range: "<< random (100);
   return 0;
}
```

**Q) Write a program to solve the following equations, where Q in degree**

$$y = \begin{cases} |\tan Q - e^{Q}| & x < 0 \\ \dfrac{-b + \sqrt{b^2 - 4ac}}{2a} & x \geq 0 \end{cases}$$

```
#include <iostream.h>
#include <conio.h>
#include <math.h>
int main()
{
 double y,x,a,b,c,Q;
 clrscr();
 cout <<"x="; cin >>x;
 if (x<0)
 {
   cout <<"Q="; cin >>Q;
   y=fabs(tan(Q*3.14/180)-exp(Q));
 }
 else
 {
  cout <<"a="; cin >>a;
  cout <<"b="; cin >>b;
  cout <<"c="; cin >>c;
  double bs=b*b-4*a*c;
  if ((a) && (bs>0))
   y=(-b+sqrt(bs))/2*a;
  else
   {
    cout <<"Error illegal values";
    getch();
    return 1;
   }
 }
 cout <<"y="<<y;
 getch();
 return 0;
}
```