OBJECT ORIENTED PROGRAMMING LAB

Lab Manual # 01

Introduction to C++

Instructor: Fariba Laiq

Semester Spring 2021

Course Code: CL217

Fast National University of Computer and Emerging Sciences Peshawar

Department of Computer Science

Table of Contents

C++ Introduction

- C++ pronounced as "See Plus Plus". It is a powerful computer programing language. It is the advance version of C language.
- The C is a procedural programing language while C++ is an Object-Oriented Programing language. Procedural means step by step process or step by step order.

C++ History

- In 1972 Dennis Ritche developed C language at Bell Laboratory.
- In 1980 C++ language was developed by "Bjarne Stroustrup" .
- C++ was an extension of C.

Writing a C++ program

- Extension of C++ program is cpp i.e. filename.cpp
- Text editor/Dev C++ or any other IDE is required to write and edit C++ source code.
- C++ Compiler is required for compilation of source code into machine code. This machine code is called object code. It is stored in a new file with extension ***obj***. The object code is then linked to the libraries. After linking the object code to the libraries, an executable file with extension ***exe*** is created. The executable program is then run from operating system command line.
- For example, a source code of program in C++ is written and stored with first.cpp.
- After compilation, the object code is saved in file first.obj and executable code is stored in file first.exe.
- Before compilation code is called **source code.**
- After compilation code is called **object code.**

Structure of C++ program

- A C++ program consists of three main parts. These are:

  1. Preprocessor directives
  2. The main() function
  3. C++ statements

1. Preprocessor directives

- ❖ Provides instructions to the compiler before the beginning of the actual program. Also called compiler directives.
- ❖ The preprocessor directives consist of instructions for the compiler.
- ❖ The compiler adds special instructions or code from these directives into program at the time of compilation.
- ❖ The preprocessor directives normally start with a number sign (#) and the keyword "include" or "define". For example preprocessor directives are used to include header files in the program.
- ❖ A program example is given below. The first statement of the program is a preprocessor directive. The preprocessor directive has been written to include the iostream.h header file.

```
#include <iostream>
int main()
 {
   cout << "Hello C++ Programming";
   return 0;
  }
```

Header File

- ❖ Header file is a C++ source file that contains definitions of library functions/objects.
- ❖ A header files are added into the program at the compilation of the program.
- ❖ A header file is added if the function/object defined in it is to be used the program.
- ❖ The preprocessor directive "**include**" is used to add a header file into the program. The name of the file is written in single brackets (<>) after "**include**" directive. The C++ has a large number of header files in which library functions are defined. The header file must be included in the program before calling its function in the program. A single header file may contain a large number of built-in library functions.
- ❖ For example the header file iostream.h has definitions of different built in input and output objects and functions. It is included in the above program because its object "cout" is used in the program.

Syntax is given by

#include<name of the header file>

Eg: #include<iostream.h> input and output stream

#include<conio.h> console input and output (console means screen)

For example cout<< is defined in iostream.h and getch() is defined in conio.h.

2. The main() function

- The main() function indicates the beginning of C++ program.
- When we run a program first of all OS runs main function.
- Main function is built in function.
- It is automatically called.
- Without it programs is not executed.

Syntax

```
int main()

{

program statements…

}
```

3. C++ statements

The statements of the program are written under the main() function between the curly braces {}.

These statements are the body of the program. Each statement in C++ ends with semicolon (;).

C++ is case sensitive language. The C++ statements are normally written in lowercase letters but in some exceptional cases, these can also be written in upper case.

C++ Program Syntax

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

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

Example explained

- **Line 1**: #include <iostream> is a header file library that lets us work with input and output objects, such as cout (used in line 5). Header files add functionality to C++ programs.
- **Line 2:** using namespace std means that we can use names for objects and variables from the standard library.
- Don't worry if you don't understand how #include <iostream> and using namespace std works. Just think of it as something that (almost) always appears in your program.
- **Line 3:** A blank line. C++ ignores white space.
- **Line 4:** Another thing that always appear in a C++ program, is int main(). This is called a **function**. Any code inside its curly brackets {} will be executed.
- **Line 5:** cout (pronounced "see-out") is an **object** used together with the *insertion operator* (<<) to output/print text. In our example it will output "Hello World".
- **Note:** Every C++ statement ends with a semicolon ;.
- **Note:** The body of int main() could also been written as:
  int main () { cout << "Hello World! "; return 0; }
- **Remember:** The compiler ignores white spaces. However, multiple lines makes the code more readable.
- **Line 6:** return 0 ends the main function.
- **Line 7:** Do not forget to add the closing curly bracket } to actually end the main function.

C++ Output

The cout object, together with the << operator, is used to output values/print text:

```
#include <iostream>
using namespace std;
int main() {
  cout << "Hello World!";
  return 0;
}
```

- You can add as many cout objects as you want. However, note that it does not insert a new line at the end of the output:

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

```
int main() {
  cout << "Hello World!";
  cout << "I am learning C++";
  return 0;
}
```

C++ New Lines

To insert a new line, you can use the \n character:

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

int main() {
  cout << "Hello World! \n";
  cout << "I am learning C++";
  return 0;
}
```

Tip: Two \n \n characters after each other will create a blank line:

```
#include <iostream>
using namespace std;
int main() {
  cout << "Hello World! \n\n";
  cout << "I am learning C++";
  return 0;
}
```

Another way to insert a new line, is with the endl manipulator:

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

int main() {
  cout << "Hello World!" << endl;
  cout << "I am learning C++";
  return 0;
}
```

C++ Comments

A well-documented program is a good practice as a programmer. It makes a program more readable and error finding become easier. One important part of good documentation is Comments.

- In computer programming, a comment is a programmer-readable explanation or annotation in the source code of a computer program

- Comments are statements that are not executed by the compiler and interpreter.

Types of comments

1. Single line comment
2. Multi-line comment

1. Single line Comment

Represented as **//** double forward slash. It is used to denote single line comment. It applies comment to a single line only. It is refereed as C++-style comments as it is originally part of C++ programming.

Example

```
#include<iostream>
int main()
{
    // Single line Welcome user comment
    cout<<"Welcome to PFlab";
    return 0;
}

/*
Output
Welcome to PFlab
*/
```

2. Multi-line Comment

Represented as /* any text */. Start with forward slash and asterisk (/*) and end with asterisk and forward slash (*/). It is used to denote multi-line comment. It can apply comment to more than a single line. It is referred as C-Style comment as it was introduced in C programming.

```
#include<iostream>
int main()
{
    /* Multi-line Welcome user comment
    written to demonstrate comments
    in C/C++ */
cout<<"Welcome to PF lab";
    return 0;
}

/*
Output
```

```
Welcome to PFlab
*/
```

Single or multi-line comments?

It is up to you which you want to use. Normally, we use // for short comments, and /* */ for longer.

C++ Keywords

The words that are used by the language for special purpose are called keywords. These are also called reserved words. For example, in a C++ the word main is used to indicate the starting of program, include is used to add header files, int is used to declare integer type variable. All these words are keywords of C++. The reserved words cannot be used as variable names in a program.

A list of 32 Keywords in C++ Language which are also available in C language are given below.

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

asm	dynamic_cast	namespace	reinterpret_cast	bool

explicit	new	static_cast	false	catch
operator	template	friend	private	class
this	inline	public	throw	const_cast
delete	mutable	protected	true	try
typeid	typename	using	virtual	wchar_t

Tokens

A program statement consists of variable names, keywords, constants, punctuations marks, operators etc. In C++ these elements of a statement are called tokens. In the following program segment:

```
main()

{

        int a, b;

}
```

main, {, }, int , a, b and punctuation marks (,) and (;) are tokens of the program.

Variables

❖ A named memory location where data is stored is called variable.

❖ A quantity whose value may change during execution of the program is called variable. It is represented by a symbol or name.

❖ **Variable** is name of *reserved area allocated in memory*. In other words, it is a *name of memory location*.

❖ It is a combination of "vary + able" that means its value can be changed.

❖ **int** data=10;   // Here data is variable

Rules for Naming/Writing Variables

1. The first character of variable must be an alphabetic character.
2. Underscore can be used as first character of variable name.
3. Blank spaces are not allowed in a variable name.
4. Special character such as arithmetic operators, #, ^, cannot be used in a variable name.
5. The maximum length of a variable name depends upon the compiler of C++.
6. A variable name declared for one data type cannot be used to declare another data type.

C++ is a case sensitive language. Thus, variable name with the same spellings but different cases are treated as different variable names. For example, variable “Pay” And “pay” are two different variables.

- C++ is a case sensitive language: number is different from Number or nUmber

Valid identifiers:

- Int abc, char aBc, int first_var, float first

Invalid identifiers:

Int 3bc, int a*b, int #a, int void

C++ Data Types

Data types specify the different sizes and values that can be stored in the variable.

Data Types tell us:

i) Type of data.

ii) Memory reserved (amount of memory).

iii) Range of value.

Keyword	Range		Bytes of Memory
	Low	High	
char	-128	127	1
short	-32,768	32768	2
int	-2,147,483,648	2,147,483,647	4
long	-2,147,483,648	2,147,483,647	4
float	$3.4*10^{-38}$	$3.4*10^{38}$	4
double	$1.7*10^{-308}$	$1.7*10^{308}$	8
long double	$3.4*10^{-4932}$	$3.4*10^{4932}$	10

sizeof() Data Type

sizeof() function is used to find size (bytes) of data type

e.g: sizeof(char)

```
#include<iostream>
using namespace std;
int main()
{
    cout << "Size of char : " << sizeof(char)  << " byte" << endl;
    cout << "Size of int : " << sizeof(int) << " bytes" << endl;


cout << "Size of short int : " << sizeof(short int)  << " bytes"
<<endl;
cout << "Size of long int : " << sizeof(long int) << " bytes" << endl;
cout << "Size of signed long int : " << sizeof(signed long int)
       << " bytes" << endl;
cout << "Size of unsigned long int : " << sizeof(unsigned long int)
       << " bytes" << endl;

cout << "Size of float : " << sizeof(float)  << " bytes" <<endl;
cout << "Size of double : " << sizeof(double)  << " bytes" << endl;
    return 0;
}

/*
```

```
Output
Size of char : 1 byte
Size of int : 4 bytes
Size of short int : 2 bytes
Size of long int : 8 bytes
Size of signed long int : 8 bytes
Size of unsigned long int : 8 bytes
Size of float : 4 bytes
Size of double : 8 bytes
*/
```

C++ Constant

When you do not want others (or yourself) to override existing variable values, use the const keyword (this will declare the variable as "constant", which means unchangeable and read-only)

Example

const int myNum = 15; // myNum will always be 15
myNum = 10; // error: assignment of read-only variable 'myNum'

You should always declare the variable as constant when you have values that are unlikely to change:

```
#include <iostream>
using namespace std;
int main() {
  const int minutesPerHour = 60;
  const float PI = 3.14;
  cout << minutesPerHour << “\n";
  cout << PI;
  return 0;
}
```

C++ Input

- You have already learned that cout is used to output (print) values. Now we will use cin to get user input.
- cin is a predefined variable that reads data from the keyboard with the extraction operator (>>).
- In the following example, the user can input a number, which is stored in the variable x. Then we print the value of x:

```
#include <iostream>
using namespace std;
int main() {
  int x;
  cout << "Type a number: "; // Type a number and press enter
  cin >> x; // Get user input from the keyboard
  cout << "Your number is: " << x;
  return 0;
}
```

Good To Know

- cout is pronounced "see-out". Used for output, and uses the insertion operator (<<)
- cin is pronounced "see-in". Used for input, and uses the extraction operator (>>)

Creating a Simple Calculator

```
#include <iostream>
using namespace std;
int main() {
  int x, y;
  int sum;
  cout << "Type a number: ";
  cin >> x;
  cout << "Type another number: ";
  cin >> y;
  sum = x + y;
  cout << "Sum is: " << sum;
  return 0;
}

/*
Output
Type a number:2
Type another number: 4
Sum is: 6
*/
```

Character Data Type

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

int main() {
    char x;
    x='a';
    cout<<"The character is :"<<x<<endl;
}
```

Output:

```
The character is :a
```

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

int main() {
    char x;
    cout<<"Enter any character: ";
    cin>>x;
    cout<<"The character you entered is :"<<x<<endl;
}
```

Output:

```
Enter any character: A
The character you entered is :A
```

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

int main() {
    char x;
    cout<<"Enter any character: ";
    cin>>x;
    cout<<"ASCII Value is :"<<int(x)<<endl;
}
```

Output:

```
Enter any character: F

ASCII Value is :70
```

Operators

- Operator is a symbol which is used to perform some operation. Operators are used to perform operations on variables and values. In the example below, we use the + **operator** to add together two values:
- int x = 100 + 50;
- Although the + operator is often used to add together two values, like in the example above, it can also be used to add together a variable and a value, or a variable and another variable:

Example

```
int sum1 = 100 + 50;        // 150 (100 + 50)
int sum2 = sum1 + 250;      // 400 (150 + 250)
int sum3 = sum2 + sum2;     // 800 (400 + 400)
```

Types of Operators

1. Unary operators
2. Binary operators
3. Ternary operators

1. Unary Operator

1. Increment (++)
2. Decrement (--)
3. Negation (!)

2. Binary Operator

1. Arithmetic (+, -, *, /, %)
2. Relational (>, <, >=, <=, !=, ==)
3. Logical (&&, ||)
4. Assignment (=)
5. Arithmetic Assignment operator (+=, -=, *=, /=, %=)

3. Ternary Operator

Conditional operator (?:)

Example (condition) ? statement 1 : statement 2;

int result= (n1>n2) ? n1 : n2;

```
#include<iostream>

using namespace std;

int main()
{
    int time = 20;
    string result = (time < 18) ? "Good day." : "Good evening.";
    cout << result;
    return 0;
}

/*
Output
Output: Good evening.
*/
```

Arithmetic Operators in C++

Arithmetic operators are used to perform common mathematical operations.

Operator	Name	Description	Example
+	Addition	Adds together two values	x + y
-	Subtraction	Subtracts one value from another	x - y
*	Multiplication	Multiplies two values	x * y
/	Division	Divides one value by another	x / y
%	Modulus	Returns the division remainder	x % y

Adding two integers

```
#include<iostream>
using namespace std;
int main()
{
    int n1, n2, sum;
    cout<<"Enter first number:\t";
    cin>>n1;
    cout<<"Enter 2nd number:\t";
    cin>>n2;
    sum=n1+n2;
    cout<<"The sum is:\t"<<sum<<endl;

}
/*
Output
Enter first number: 3
Enter 2nd number: 6
The sum is : 9
*/
```

Assignment Operator

Assignment operators are used to assign values to variables. In the example below, we use the assignment operator (=) to assign the value 10 to a variable called x:

Example

int x = 10;

The **addition assignment** operator (+=) adds a value to a variable:

Example

int x = 10;
x += 5;

A list of all arithmetic assignment operators:

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3

Relational/Comparison Operators

- Comparison operators are used to compare two values.
- **Note:** The return value of a comparison is either true (1) or false (0).
- In the following example, we use the **greater than** operator (>) to find out if 5 is greater than 3:

Example

```
int x = 5;
int y = 3;
cout << (x > y); // returns 1 (true) because 5 is greater than 3
```

A list of all relational operators:

Operator	Name	Example
==	Equal to	x == y
!=	Not equal	x != y
>	Greater than	x > y

<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

Logical Operators

Logical operators are used to determine the logic between variables or values:

Operator	**Name**	**Description**	**Example**
&&	Logical and	Returns true if both statements are true	x < 5 && x < 10
||	Logical or	Returns true if one of the statements is true	x < 5 || x < 4
!	Logical not	Reverse the result, returns false if the result is true	!(x < 5 && x < 10)

Increment and Decrement Operators

1) Increment Operator:

The operators that is used to add 1 to the value of a variable is called increment operator.

2) Decrement Operator :

The operator that is used to subtract 1 from the value of a variable is called decrement operator.

1) The Increment Operator (++)

❖ The increment operator is represented by a double plus (++) sign.

❖ It is used to add 1 to the value of an integer variable.

❖ This variable can be used before or after the variable name.

❖ For example, to add 1 to a value of variable xy, it is normally written as

```
xy = xy + 1;
```

- By using increment operator “++” it is written as

  xy++

- The increment operator can be written either before or after the variable.
- If it is written before the variable, it is known as **prefixing.**
- If it is written after the variable, it is known as **post fixing**.
- Prefix and postfix operators have different effects when they are used in expressions.

i) Prefix Increment Operator

When an increment operator is used in prefix mode in an expression, it adds 1 to the value of the variable before the values of the variable is used in the expression.

```
#include<iostream>

using namespace std;

int main()
{
    int a=2;
    int b=3;
    int c=2;
    int result=a+b+(++c);
    cout<<"Result is: "<<result;
    cout<<"\nValue of c is: "<<c;

}
/*
Output
Result is:8
nValue of c is : 3
*/
```

- In the above program, 1 will be added to the value of c before it is used in the expression.
- Thus, after execution, the result will be equal to 8 and the value of c will be 3.

ii) Postfix Increment Operator

- When an increment operator is used in postfix mode in an expression, it adds 1 to the value of the variable after the value of the variable is used in the expression.

- For Example, if in the above example, increment operator is used in postfix mode, the result will be different. The statement will be shown below:
    - result =a + b + c++;

- In this case, 1 will be added to the value of c after its existing value has been used in the expression. Thus, after execution, the result will be equal to 7 and the value of c will be 3.

```
#include<iostream>

using namespace std;

int main()
{
    int a=2;
    int b=3;
    int c=2;
    int result=a+b+(c++);
    cout<<"Result is: "<<result;
    cout<<"\nValue of c is: "<<c;

}
/*
Output
Result is:7
nValue of c is : 3
*/
```

2) The Decrement Operator (--)

- The decrement operator is represented by a double minus (--) sign.

- It is used to subtract 1 from the value of an integer variable.

- This variable can be used before or after the variable name.

- For example, to subtract 1 from the value of variable xy, the decrement statement is written as

    xy--;   or   --xy;

i) Prefix Decrement Operator

- When decrement operator is used in prefix mode in an expression, it subtracts 1 from the value of the variable **before** the values of the variable is used in the expression.

```
#include<iostream>
```

```
using namespace std;

int main()
{
    int a=2;
    int b=3;
    int c=2;
    int result=a+b+(--c);
    cout<<"Result is: "<<result;
    cout<<"\nValue of c is: "<<c;

}

/*
Output
Result is:6
nValue of c is : 1
*/
```

- In the above program, 1 will be subtracted from the value of **c** before it is used in the expression.
- Thus, after execution, the result will be equal to 6 and the value of **c** will be 1.

ii) Postfix Decrement Operator

- When an decrement operator is used in postfix mode in an expression, it subtracts 1 from the value of the variable **after** the values of the variable is used in the expression.
- For Example, if in the above example, decrement operator is used in postfix mode, the result will be different. The statement will be shown below:

    result =a + b + c--;

- In this case, 1 will be subtracted from the value of **c** after its existing value has been used in the expression. Thus, after execution, the result will be equal to 7 and the value of c will be 1.

```
#include<iostream>

using namespace std;

int main()
{
    int a=2;
    int b=3;
    int c=2;
    int result=a+b+(c--);
    cout<<"Result is: "<<result;
    cout<<"\nValue of c is: "<<c;

}

/*
Output
Result is:7
nValue of c is : 1
*/
```

Class Task-1

Ask user to enter a three digit number and then display the number in reverse order.

```
#include<iostream>
using namespace std;
int main()
{
    int number;
    cout<<"Enter 3 digit number:";
    cin>>number;
    cout<<number%10;
    number=number/10;
    cout<<number%10;
    number=number/10;
    cout<<number;
}
/*
Output
Enter 3 digit number:123
321
Enter 3 digit number:456
654
*/
```

Conditional Statements

Relational Operators/comparison operators

Algebraic	In C++	Example	Meaning
>	>	x>y	x is greater than y
<	<	x<y	x is less than y
≥	>=	x>=y	x is greater than or equal to y
≤	<=	x<=y	x is less than or equal to y
=	==	x==y	x is equal to y
≠	!=	x!=y	x is not equal to y

Blocks of Code

Whenever we write an if statement or a loop, if there is more than one statement of code which has to be executed, this has to be enclosed in braces, i.e. '{ …. }'

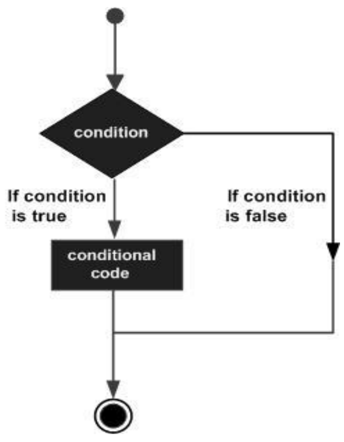
Conditional Statements

- ❖ Also called decision making statements decision control statements.
- ❖ Decision making structures have one or more conditions to be evaluated or tested by the program, along with a statement or statements that are to be executed if the condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false.
- ❖ Used to change the flow.
- ❖ In these statements (conditions) order or sequence of the statements are changed.
- ❖ Sometimes we need to execute a block of statements only when a particular condition is met or not met. This is called **decision making**, as we are executing a certain code after making a decision in the program logic.

1. If Statement
2. If-else Statement
3. If-else if- else statement
4. Switch statement
5. Nested Statement

- Use **if** to specify a block of code to be executed, if a specified condition is true.

- Use **else** to specify a block of code to be executed, if the same condition is false.
- Use **else if** to specify a new condition to test, if the first condition is false.
- Use **switch** to specify many alternative blocks of code to be executed.



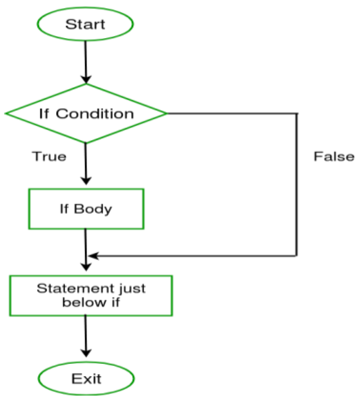
1) if Statement

if statement will execute or skip or ignore a block of code depending one condition

Syntax:

```
if(conditional expression)

{

 // Statements will execute if the Boolean expression is true

}
```

The statements inside **if** parenthesis (usually referred as if body) gets executed only when the given condition is true. If the condition is false than the statements inside if body are completely ignored.



```
#include<iostream>
using namespace std;
int main()
{
   if (20>18)
   {
       cout<<"20 is greater than 18";
   }

}

/*
Output
20 is greater than 18
*/
```

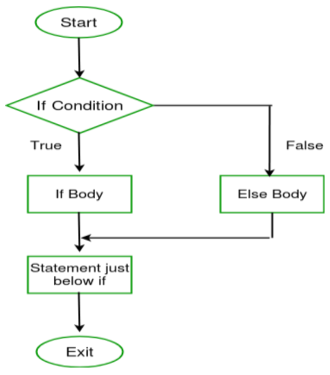
We can also test variables:

```
#include <iostream>
using namespace std;
int main() {
  int x = 20;
  int y = 18;
  if (x > y) {
    cout << "x is greater than y";
  }
  return 0;
}

/*
Output
20 is greater than 18
*/
```

2) if else Statement

- ❖ Used for making two way decision.
- ❖ It will execute if block if condition is true and will execute another block (else block) if condition is false.
- ❖ It will take one action if the condition is true and take another action if condition is false.

- Sometimes you have a condition and you want to execute a block of code if condition is true and execute another piece of code if the same condition is false. This can be achieved in C++ using if-else statement.
- This is how an if-else statement looks:



```
if(condition)
{
Statement(s);
}
else
{
Statement(s);
}
```

The statements inside "if" would execute if the condition is true, and the statements inside "else" would execute if the condition is false.

```
#include<iostream>
using namespace std;
int main()
{   int x=15;
    int y=18;

   if (x>y)
   {
       cout<<"x is greater than y";
   }
   else
   {
       cout<<"y is greater than x";
   }
   return 0;
}

/*
Output
y is greater than x
*/
```

3. if-else-if else Statement

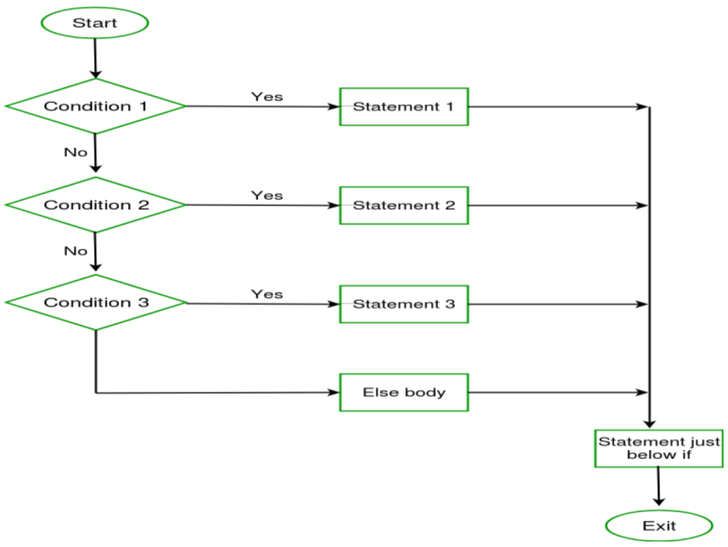
- This statement is used to check multiple conditions.
- Used to execute one condition from multiple statements.

❖ if-else-if statement is used when we need to check multiple conditions. In this control structure we have only one "if" and one "else", however we can have multiple "else if" blocks. This is how it looks:

```
if(condition_1)
{
/*if condition_1 is true execute this*/
statement(s);
}
else if(condition_2)
{
/*execute this if condition_1 is not met and condition_2 is met*/
statement(s);
}
else if(condition_3)
{   /* execute this if condition_1 & condition_2 are not met and
condition_3 is met    */
statement(s);
}
.
.
.
else {
/* if none of the condition is true then these statements gets
executed    */
statement(s);
}
```

Note:

❖ The most important point to note here is that in if-else-if, as soon as the condition is met, the corresponding set of statements get executed, rest gets ignored.
❖ If none of the condition is met then the statements inside "else" gets executed.



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

int main()
{
    int x=20;

   if (x==10)
   {
       cout<<"x is 10";
   }
   else if(x==15)
   {
       cout<<"x is 15";
   }
else if(x==20)
   {
       cout<<"x is 20";
   }
else
   {
            cout<<"sorry! x is not present";
   }

return 0;

}

/*
Output
x is 20
*/
```

```
#include<iostream>
using namespace std;
int main()
{
int number;
cout<<"Enter a number between 1 to 3:";
cin>> number;
if(number == 1)
{
cout<<"You pressed 1" <<endl;
}
else if(number == 2)
{
cout<<"You pressed 2" <<endl;
}
else if(number == 3)
{
cout<<"You pressed 3" <<endl;
}
else
{
cout<<"Invalid input";
}
}

/*
Output
Enter a number between 1 to 3: 3
You pressed 3
Enter a number between 1 to 3: 50
Invalid input
*/
```

4. Nested if statement

You can use one **if** or **else if** statement inside another **if** or **else if** statement(s).

```
#include<iostream>
using namespace std;
int main()
{
    int x=30;
    int y=10;
   if (x==30)
   {
       if(y==10)
       {
           cout<<"x=30 and y=10";
       }
   }
return 0;
}

/*
Output
x=30 and y=10
*/
```

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

int main()
{
bool job; char martialStatus; int age;
cout<<"Enter Martial Status:";
cin>>martialStatus;
cout<<"Enter age:";
cin>>age;
cout<<"Enter job Status:";
cin>>job;
if(martialStatus =='u')
{
    if(age <=25)
        if(job ==false)
            cout<<"Eligible for Loan";
}

else
    cout<<"Not eligible for Loan";
return 0;

}

/*
Output
Enter Marital Status: u
Enter age:20
Enter job Status: 0
Eligible for Loan

Enter Marital Status: u
Enter age:18
Enter job Status: 1
Not eligible for Loan
*/
```

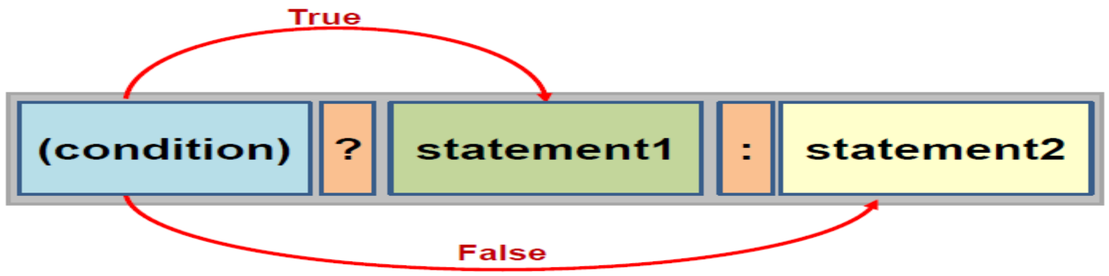
5. Conditional Operator (? :)

It is ternary operator and work on three operands. It works like if else statement.

Syntax:

(condition) ? statement 1 : statement 2;

Statement must be single and it is the limitation of conditional operator.



Conditional Operator (? :) Example

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

int main()
{
    int n1=5;
    int n2=4;
    int result = (n1>n2) ? n1 : n2;
    cout<<result;
return 0;

}

/*
Output
5
*/
```

Logical Operators

Used for compound condition or expression

- ❖ AND (&&)
- ❖ OR (||) pip sign

1. AND (&&)

```
#include<iostream>
using namespace std;
int main()
{
    int x=30;
    int y=10;

   if (x==30 && y==10)
   {
           cout<<"x=30 and y=10";
   }
   else
   {
    cout<<"Either x is not equal to 30 or y is not equal to 10 or both
 are not equal";
   }
return 0;
}

/*
Output

*/
```

2. OR (||)

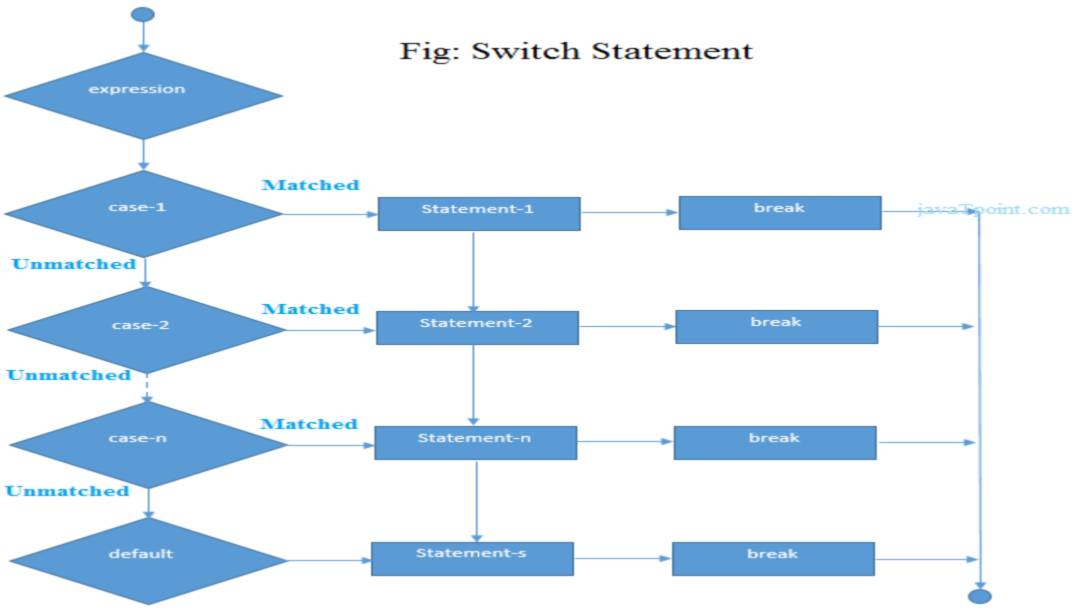
```
#include<iostream>
using namespace std;
int main()
{
    int x=30;
    int y=10;

   if (x==30 || y==10)
   {
           cout<<"x=30 or y=10";
   }
   else
   {
    cout<<"Both are not equal means x is not equal to 30 and y is not
equal to 10";
   }
return 0;
}
```

6. Switch statement

❖ Used when multiple choices are given and one is to be selected. It is like if else-if- else statement.

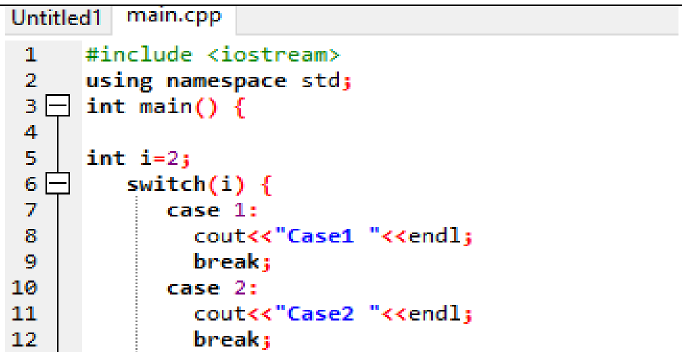
❖ Used to select one several actions based on the value of variable or expression.

❖ **Switch case statement** is used when we have number of options (or choices) and we may need to perform a different task for each choice.

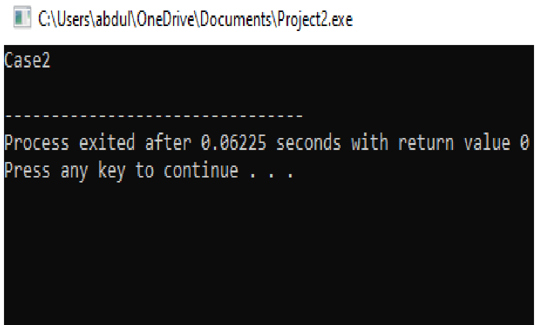


Switch statement Syntax

```
switch(variable/expression)
{
case value 1:
statement(s);  // code to be executed
break;
case value 2:
statement(s);   // code to be executed
break;
.
.
.
case value n:
Statement(s);      // code to be executed
break;
default:
statement(s);      // code to be executed if all cases are not matched
}
```



```
#include <iostream>
using namespace std;
int main() {

int i=2;
   switch(i) {
      case 1:
        cout<<"Case1 "<<endl;
        break;
      case 2:
        cout<<"Case2 "<<endl;
        break;
```



```
Case2

--------------------------------
Process exited after 0.06225 seconds with return value 0
Press any key to continue . . .
```

Switch statement VS if else if

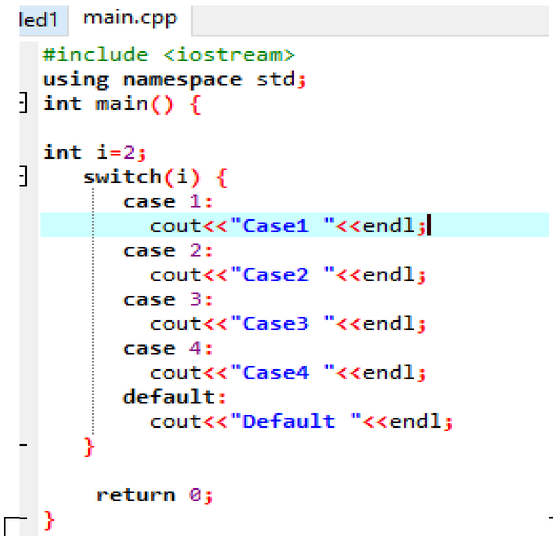
❖ If a program contains conditions or compound conditions then we use if else if else. If program contains single variable or expression then we use switch statement.

Compound conditions

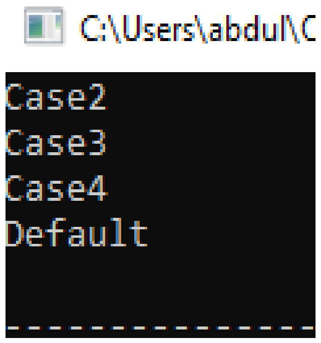
1) if(a>b && a>c)
2) if)(a>b || a>c)

C++ Switch statement is fall through

It means it executes all statements after match if break statement is not used with switch cases.



```
#include <iostream>
using namespace std;
int main() {

int i=2;
    switch(i) {
        case 1:
            cout<<"Case1 "<<endl;
        case 2:
            cout<<"Case2 "<<endl;
        case 3:
            cout<<"Case3 "<<endl;
        case 4:
            cout<<"Case4 "<<endl;
        default:
            cout<<"Default "<<endl;
    }

      return 0;
}
```



```
C:\Users\abdul\C
Case2
Case3
Case4
Default

---------------
```

```
#include <iostream>
using namespace std;
int main() {
cout<<"Enter alphabet: ";
char alphabet;
cin>>alphabet;
switch(alphabet)
{
    case 'a':
    case 'e':
    case 'i':
    case 'o':
    case 'u':
    case 'A':
    case 'E':
    case 'I':
    case 'O':
    case 'U':
    cout<<"You entered vowel";
    break;
    default:
    cout<<"You entered consonant";
}   // switch body Closed
return 0;
}

/*
Output
Enter alphabet: a
You entered vowel

Enter alphabet: I
You entered vowel

Enter alphabet: s
You entered consonant
*/
```

Break Statement

The break statement is used to exit from the body of the switch structure or loop structure.

The break statement terminates the execution of the loop when it is used inside the body of the loop.

Syntax: break;

boolean variable

Syntax:

bool variable_name;

e.g: bool even;

```
#include <iostream>
using namespace std;
int main() {
cout<<"Enter any number: ";
int number;
cin>>number;
bool even;   // bool variable declaration
even =(number % 2==0);
if(even)
{
```

```
cout<<"Even number";
}
else
{
cout<<"Odd Number";
}
return 0;
}

/*
Output
Enter any number: 6
Even number

Enter any number: 7
Odd Number
*/
```

Control Structure/Loop

- In computer programming, a loop is a sequence of instructions that is continually repeated until a certain condition is reached.
- A loop statement allows us to execute a statement or group of statements multiple times.
- Loops are used in programming to repeat a specific block until some end condition is met.

There are four type of loops in C++ programming:

1) for loop
2) while loop
3) do while loop
4) for-each loop (Enhanced for Loop)

Pretested Loop: Loop in which condition is checked first.

e.g. for loop and while loop

Post tested Loop: Loop in which condition is checked at the end.

e.g. do while loop

Determined loop/Definite loop: Loop for fixed repetition.

e.g. for loop

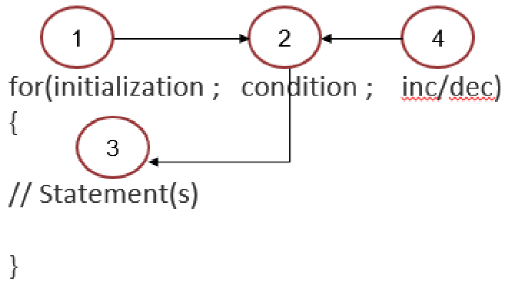
Undetermined loop/Indefinite loop: loop not for fixed repetition

E.g. while loop and do while loop

1. for loop

- for loop is used to a statement or group of statement for a fixed number of time.
- If the number of iterations is fixed then it is recommended to use for loop.
- A for loop is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times.

Syntax:



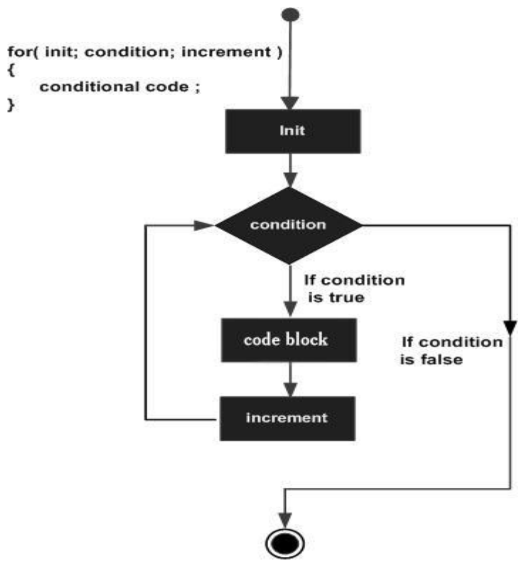
```
for(initialization ;  condition ;  inc/dec)
{
// Statement(s)

}
```

Order of Steps in for loop:

1. 1st initialization is performed.
2. Secondly condition is checked
3. In 3rd step statement is executed means control goes to body of the loop.
4. In 4th step incrementation or decrementation is performed.
5. Again condition is checked an so on.

Note: In loops variable is called counter variable.

i= i+1;   OR   i+=1;



```
#include <iostream>
using namespace std;
int main() {
for (int i = 0 ; i< 5 ; i++)
{
    cout << i << "\n";
}
  return 0;
}
/*
Output
0
1
2
3
4
*/
```

```
#include <iostream>
using namespace std;
int main ()
{
   // for loop execution
   for( int a = 10 ; a < 20; a++ )
   {
      cout << "value of a: " << a << endl ;
   }
return 0;
}

/*
output
value of a:10
value of a:11
value of a:12
value of a:13
value of a:14
value of a:15
value of a:16
value of a:17
value of a:18
value of a:19
*/
```

```
#include <iostream>
using namespace std;
int main ()
{
for( int a = 0; a < =10; a++ )
   {
    cout << "value of a: " << a << endl;
   }
return 0;
}

/*
Output
value of a: 0
value of a: 1
value of a: 2
value of a: 3
value of a: 4
value of a: 5
value of a: 6
value of a: 7
value of a: 8
value of a: 9
Value of a: 10
*/
```

```
#include <iostream>
using namespace std;
int main() {
  for (int i = 0 ; i <= 10; i = i + 2)
{
    cout << i << "\n";
}
  return 0;
}

/*
Output
0
2
4
8
10
*/
```

infinitive for loop

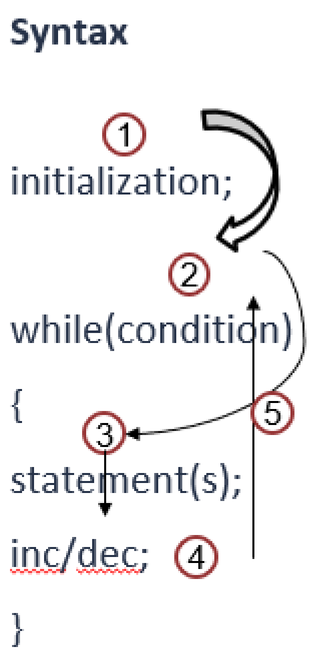
If you use two semicolons (; ;) in the for loop it will be infinitive for loop.

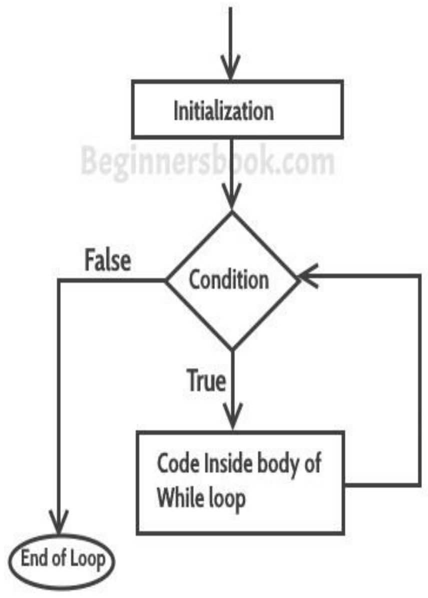
```
#include <iostream>
using namespace std;

int main () {
 for (;;)
 {
    cout<<“infinitive for loop";
 }

   return 0;
}
```

2. while loop

❖ Is used when number of iterations is not fixed.

❖ A **while** loop statement repeatedly executes a target statement as long as a given condition is true.





- Here, **statement(s)** may be a single statement or a block of statements. The **condition** may be any expression, and true is any non-zero value. The loop iterates while the condition is true.

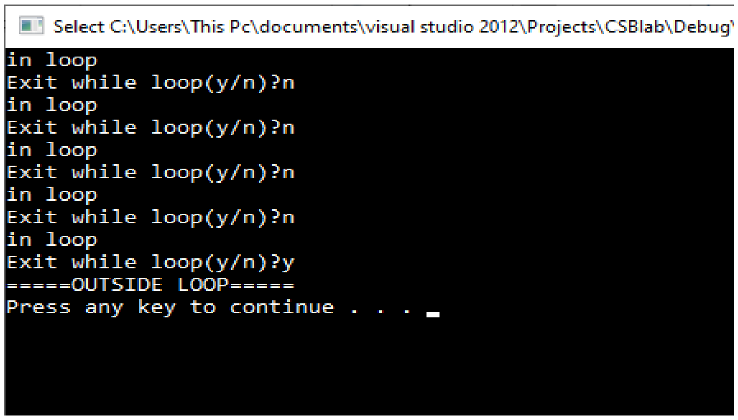
- When the condition becomes false, program control passes to the line immediately following the loop.

```
#include <iostream>
using namespace std;
int main() {
  int i = 0;
  while (i < 5) {

   cout << i << "\n";
   i++;
  }
  return 0;
}

/*
Output
0
1
2
```

```
3
4
*/
```

Note: Do not forget to increase the variable used in the condition, otherwise the loop will never end!

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

int main () {
   // Local variable declaration:
    char c = 'n';
   // while loop execution
   while( c !='y' )
   {
      cout << "in loop"<<endl;
      cout<<"Exit while loop(y/n)?";
      cin>>c;
   }
   cout<<"=====OUTSIDE LOOP====="<<endl;
   return 0;
}

/*
Output

*/
```



Select C:\Users\This Pc\documents\visual studio 2012\Projects\CSBlab\Debug\

```
in loop
Exit while loop(y/n)?n
in loop
Exit while loop(y/n)?n
in loop
Exit while loop(y/n)?n
in loop
Exit while loop(y/n)?n
in loop
Exit while loop(y/n)?y
=====OUTSIDE LOOP=====
Press any key to continue . . .
```

Sentinel Condition: Truthfulness or falseness depends upon user input.

infinitive while loop

If you pass true or true value in the while loop, it will be infinitive while loop.

Syntax:

while(true)

{

  Statement(s);

}

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

int main () {
 while(true)
 {
    cout<<"infinitive while loop";
 }

   return 0;
}
```

3. do while loop

An indefinite loop. Best used when the number of iteration is unknown. Used when you will execute the loop at least once. Unlike **for** and **while** loops, which test the loop condition at the top of the loop, the **do...while** loop checks its condition at the bottom of the loop. A **do...while** loop is similar to a while loop, except that a do...while loop is guaranteed to execute at least one time.

Syntax

initialization;

do

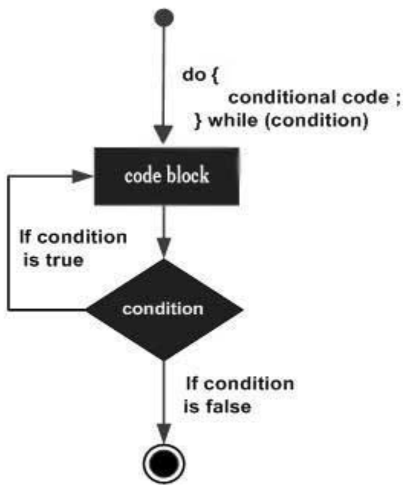
{

statement(s);

inc/dec;

}

while(condition) ;



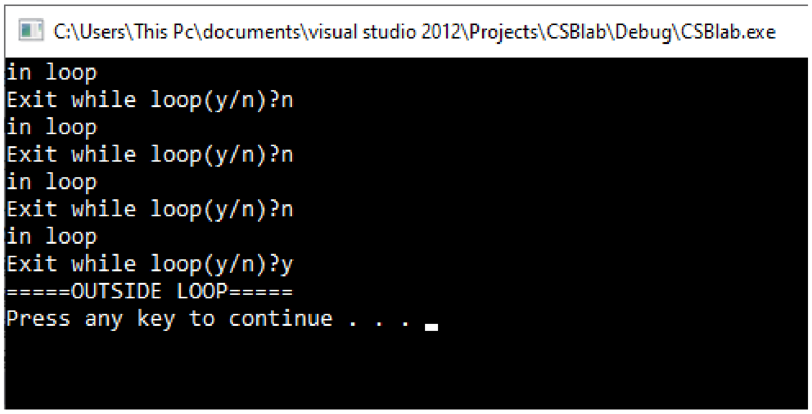
- ❖ Notice that the conditional expression appears at the end of the loop, so the statement(s) in the loop execute once before the condition is tested.
- ❖ If the condition is true, the flow of control jumps back up to do, and the statement(s) in the loop execute again. This process repeats until the given condition becomes false.

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

int main() {
  int i = 0;
  do {
    cout << i << "\n";
    i++;
  }
  while (i < 5);
  return 0;
}

/*
Output
0
2
3
4
*/
```

Do not forget to increase the variable used in the condition, otherwise the loop will never end!

```
#include <iostream>
using namespace std;
int main ()
{
   // Local variable declaration:
      char c ;
   // do-while loop execution
      do
      {
       cout << "in loop"<<endl;
       cout<<"Exit while loop(y/n)?";
       cin>>c;
   }
while( c !='y' );
```

```
   cout<<"=====OUTSIDE LOOP====="<<endl;
   return 0;
}
```


C:\Users\This Pc\documents\visual studio 2012\Projects\CSBlab\Debug\CSBlab.exe

```
in loop
Exit while loop(y/n)?n
in loop
Exit while loop(y/n)?n
in loop
Exit while loop(y/n)?n
in loop
Exit while loop(y/n)?y
=====OUTSIDE LOOP=====
Press any key to continue . . .
```

infinitive do while loop

If you pass true or true value in the do while loop, it will be infinitive do while loop.

Syntax:

do

{

Statement(s);

}

while(true);

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

int main () {
 while(true)
 {
    cout<<"infinitive do while loop";
 }

   return 0;
}
```

Break Statement

The break statement terminates the execution of the loop when it is used inside the body of the loop.

Syntax: break;

The break statement can also be used to jump out of a **loop**.

```
#include <iostream>
using namespace std;
int main() {
for (int i = 0; i <10; i++)
{
    if (i == 4) {
      break;
    }
    cout << i << "\n";
}
  return 0;
}


/*
Output
0
1
2
3
*/
```

Continue Statement

- ❖ The continue statement shifts the control back to the beginning of the loop.
- ❖ It is used inside the body of the loop.
- ❖ It is used to continue loop.
- ❖ It continues the current flow of the program and skips the remaining code at specified condition.

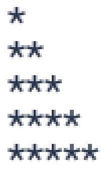
Syntax: continue;

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

int main() {
  for (int i = 0; i < 10; i++)
{
    if (i == 4) {
      continue;
    }
    cout << i << "\n";
  }
  return 0;
}


/*
Output
0
1
2
3
5
6
7
8
9
*/
```

4. Nested Loop

Loop within the body of another loop is called nested loop.

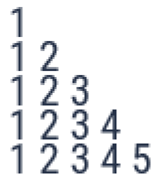
Program 1: (Triangular loop which make triangle using for nested loop with help of astricts (*)).



```
*
**
***
****
*****
```

```
#include <iostream>
using namespace std;
int main () {
 for (int i=1 ; i<=5 ; i++)
 {
    for (int j=0 ; j<i ; j++)
    {
        cout<<"*";
    }
 cout<<endl;
 }

   return 0;
}
```

Program 2: (Triangular loop which make triangle using for nested loop with help of numbers).



```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

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

int main () {
 for (int i=1 ; i<=5 ; i++)
 {
    for (int j=1 ; j<=i ; j++)
    {
        cout<<j;
        cout<<" ";
    }
 cout<<endl;
 }

   return 0;
}
```