C.P.PATEL & F.H.SHAH COMMERCE COLLEGE
(MANAGED BY SARDAR PATEL EDUCATION TRUST)
BCA, BBA (ITM) & PGDCA PROGRAMME:
BCA Semester III -Paper Code: US03CBCA23

UNIT 1- **Object Oriented Programming (OOP) Concepts and Introduction to C++**

Sr. No.	Topics
1	**Object Oriented Programming (OOP) Concepts and Introduction to C++**
2	_ Structured programming vs. object oriented programming
3	_ Basic OOP concepts : objects , classes , encapsulation , data hiding , inheritance, polymorphism
4	_ Introduction to C++: structure of a C++ program , data types , variables, constants, expressions, statements and operators
5	_ Usage of header files
6	_ Control flow statements : if else, for loop, while loop, do while loop, switch, break and continue

Structured programming vs. object oriented programming

Structured Programming Language

1. Follow top-down approach to program design.
2. Data and Functions don't tide with each other.
3. Large programs are divided into smaller self contained program segment known as functions.
4. Data moves openly around the system from function to function.
5. Functions are dependent so reusability is not possible
6. Structured Programming is designed which focuses on process/ logical structure and then data required for that process.
7. Structured Programming is also known as Modular Programming and a subset of procedural programming language.
8. Structured Programming is less secure as there is no way of data hiding.
9. Structured Programming can solve moderately complex programs.
10. Structured Programming provides less reusability, more function dependency and less abstraction and less flexibility.

Object Oriented Programming Language

1. Follow bottom-up approach in program design.
2. Functions and data are tied together.
3. Programs are divided into entity called Objects.
4. Data is hidden and can't be accessed by the external world.
5. Functions are not dependent so reusability is possible.
6. Object Oriented Programming is designed which focuses on data.
7. Object Oriented Programming supports inheritance, encapsulation, abstraction, polymorphism, etc.
8. Object Oriented Programming is more secure as having data hiding feature.
9. Object Oriented Programming can solve any complex programs.

10. Object Oriented Programming provides more reusability, less function dependency and more abstraction and more flexibility.

Basic OOPs concepts:

Objects:

Objects are the basic runtime entities in an object-oriented system. They may represent a person, a place, a bank account, a table of data or any item that the program has to handle. They may also represent user defined data such as vectors, time and lists. Programming problem is analyzed in terms of objects and the nature of communication between them. Program objects should be chosen such that they match closely with the real-world objects. Objects take up space in the memory and have as associated address like a record in Pascal, or a structure in C.
When a program is executed, the objects interact by sending messages to one another. For example, if customer and account are two objects in a program, then the customer object may send a message to the account object requesting for the bank balance. Each object contains data, and code to manipulate the data. Objects can interact without having to know details of each other's data or Code. It is sufficient to know the type of message accepted, and the type of response returned by the objects. Although different authors represent them differently, Fig. 1.7 shows two notations that are popularly used in object oriented analysis and design.

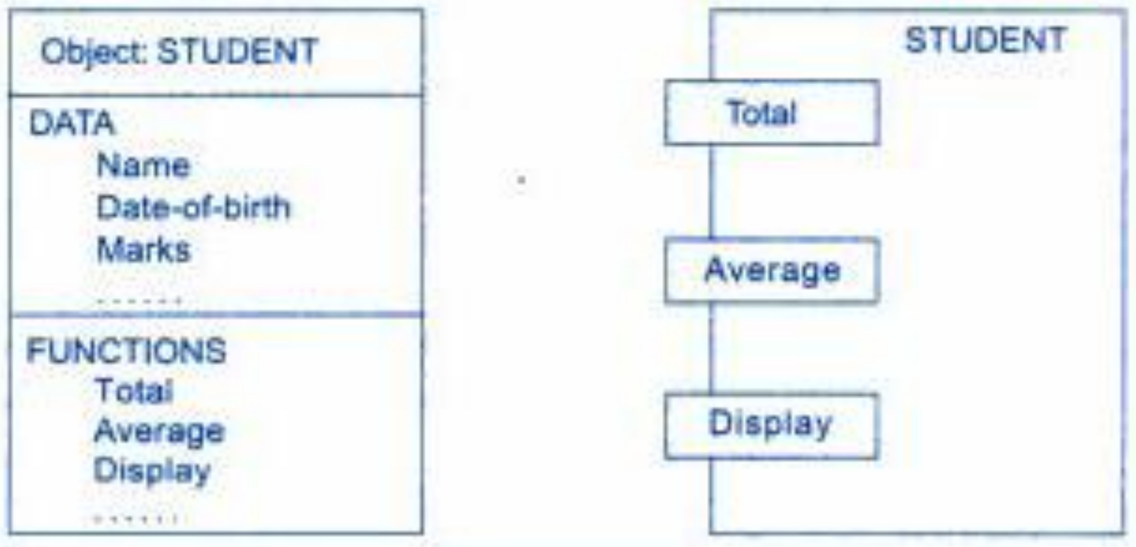


Fig. 1.7 ⇔ Two ways of representing an object

Classes

We just mentioned that objects contain data, and code to manipulate that data. The entire set of data and code of an object can be made a user-defined data type with the help of a class. In fact, objects are variables of the type class. Once a class has been defined, we can create any number of objects belonging to that class. Each object is associated with the data of type class with which they are created. A class is thus a collection of objects of similar type. For example, mango, apple and orange are members of the class fruit Classes are user-defined data types and behave like the built-in types of a programming language. The syntax used to create an object is no different than the syntax used to create an integer object in C. If fruit bas been defined as a class, then the statement will create an object mango belonging to the class fruit.

fruit mango;

Data Abstraction and Encapsulation

The wrapping up of data and functions into a single unit (called c1ass) is known as **Encapsulation**. Data encapsulation is the most striking feature of a class. The data is not accessible to the outside world, and only those functions which are wrapped in the class can access it. These functions provide the interface between the object's data and the program. This insulation of the data from direct access by the program **is called data hiding or information hiding.** Abstraction refers to the act of representing essential features without including the background details or explanations. Classes use the concept of abstraction and are defined as a list of abstract attributes such as size, weight and cost and functions to operate on these attributes. They encapsulate all the essential properties of the objects that are to be created. The attributes are sometimes called data numbers because they hold information. The functions that operate on these data are sometimes called methods or member functions. Since the classes use the concept of data abstraction, they are known as Abstract Data Types (ADT).

Inheritance

Inheritance is the process by which objects of one class acquire the properties of objects of another class. It supports the concept of hierarchical classification. For example, the bird 'robin' is a part of the class 'flying bird' which is again a part of the class 'bird'. The principle behind this sort of division is that each derived class shares common characteristics with the class from which it. is derived as illustrated in Fig. 1.8

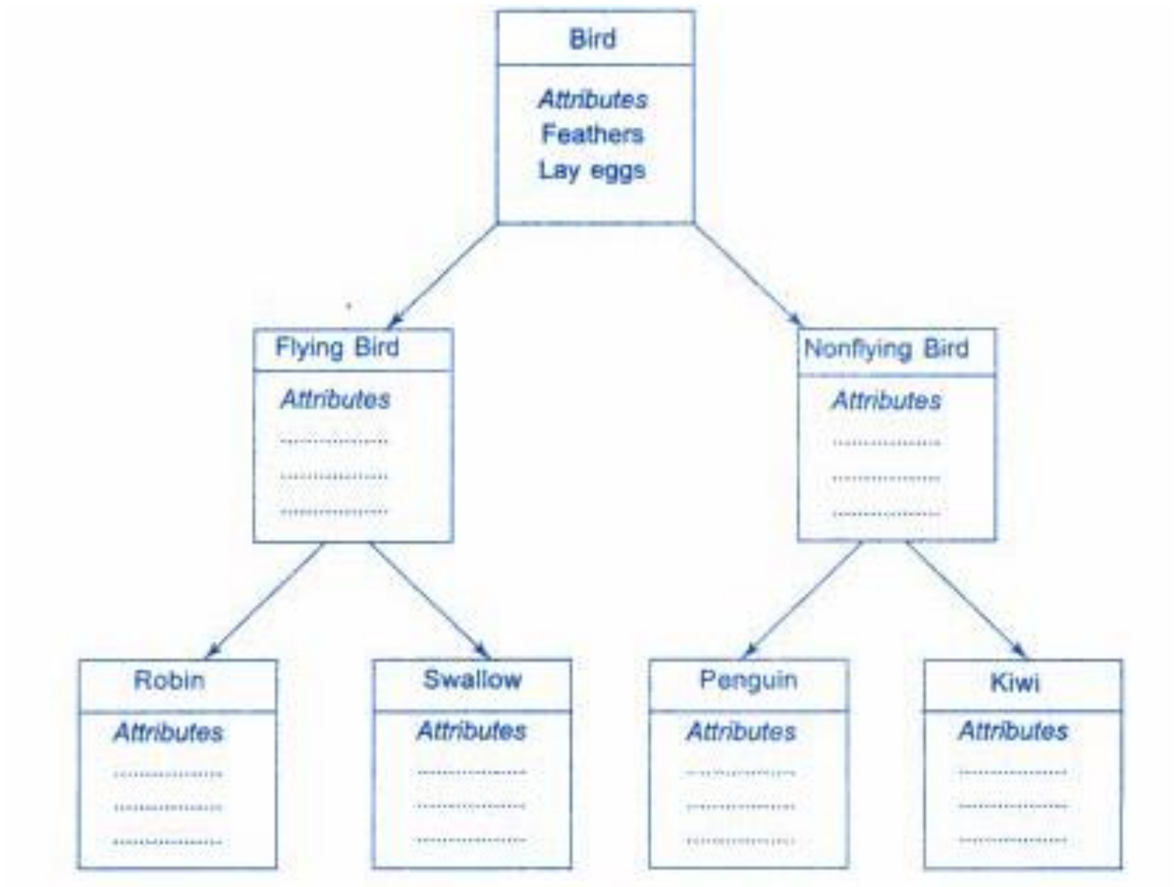


Fig. 1.8 ⇔ Property inheritance

In OOP, the concept of inheritance provides the idea of reusability. This means that we can add additional features to an existing class without modifying it. This is possible by deriving a new class from the existing one. The new class will have the combined features of both the classes. The real appeal and power of the inheritance mechanism is that it allows the programmer to reuse a class that is almost, but not exactly, what he wants, and to tailor the class in such a way that it does not introduce any undesirable side-effects into the rest of the classes. Note that each subclass defines only those features that are unique to it. Without the use of classification, each class would have to explicitly include all of its features.

Polymorphism

Polymorphism is another important OOP concept. Polymorphism. a Greek term, means the ability to take more than one form. An operation may exhibit different behaviours in different instances. The behaviour depends upon the types of data used in the operation. For example, consider the operation of addition. For two numbers, the operation will generate a sum if the operands are strings then the operation would produce a third string by concatenation. The process of making an operator to exhibit different behaviours in different instances is known as operator overloading. Figure 1.9 illustrates chat a single function name can be used to handle different number and different types of arguments. This is something similar to a particular word having several different meanings depending on the context. Using a single function name to perform different types of tasks is known as function overloading. Polymorphism plays an important role in allowing objects having different internal structures to share the same external interface. This means that general class of operations may be accessed in the same manner even though specific actions associated with each operation may differ. Polymorphism is extensively used in implementing inheritance.

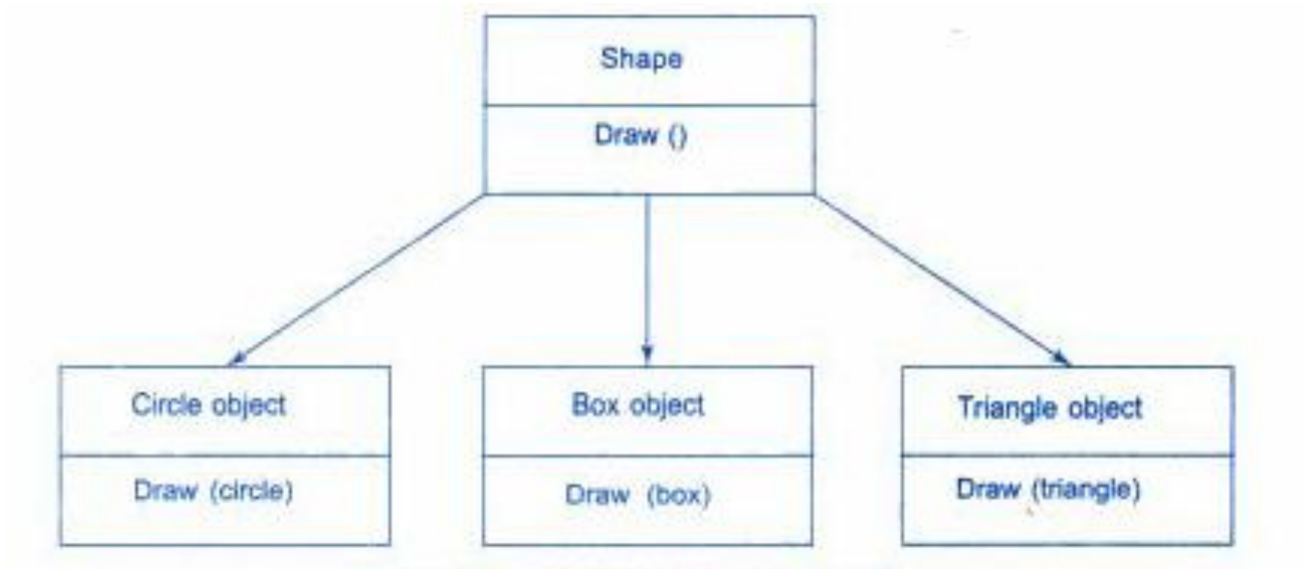
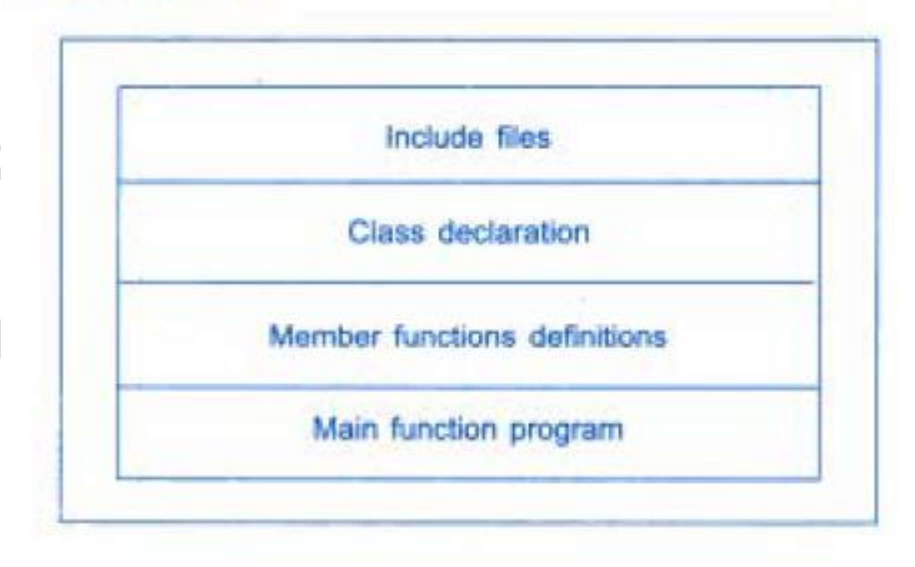


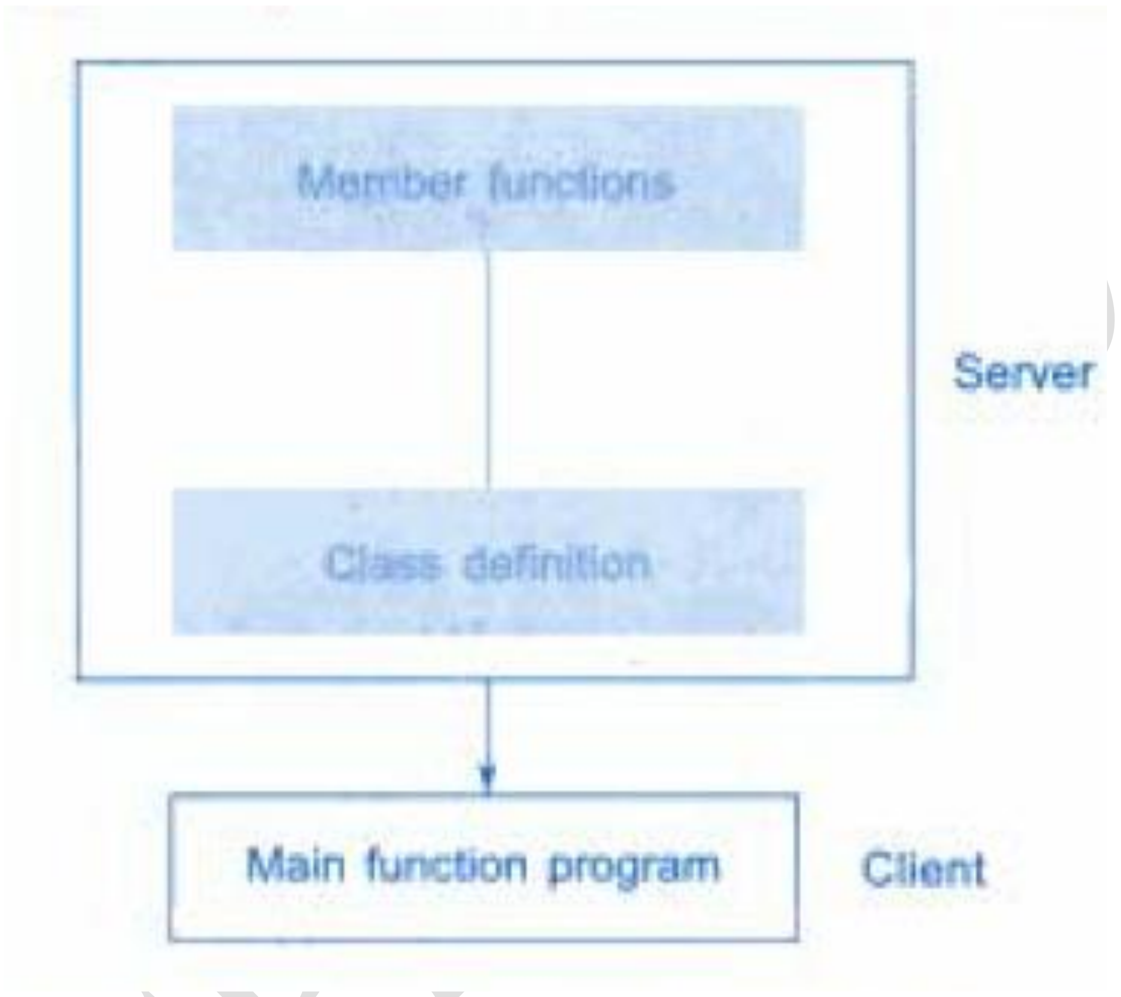
Fig. 1.9 ⇔ *Polymorphism*

Introduction to C++:
Structure of a C++ program:

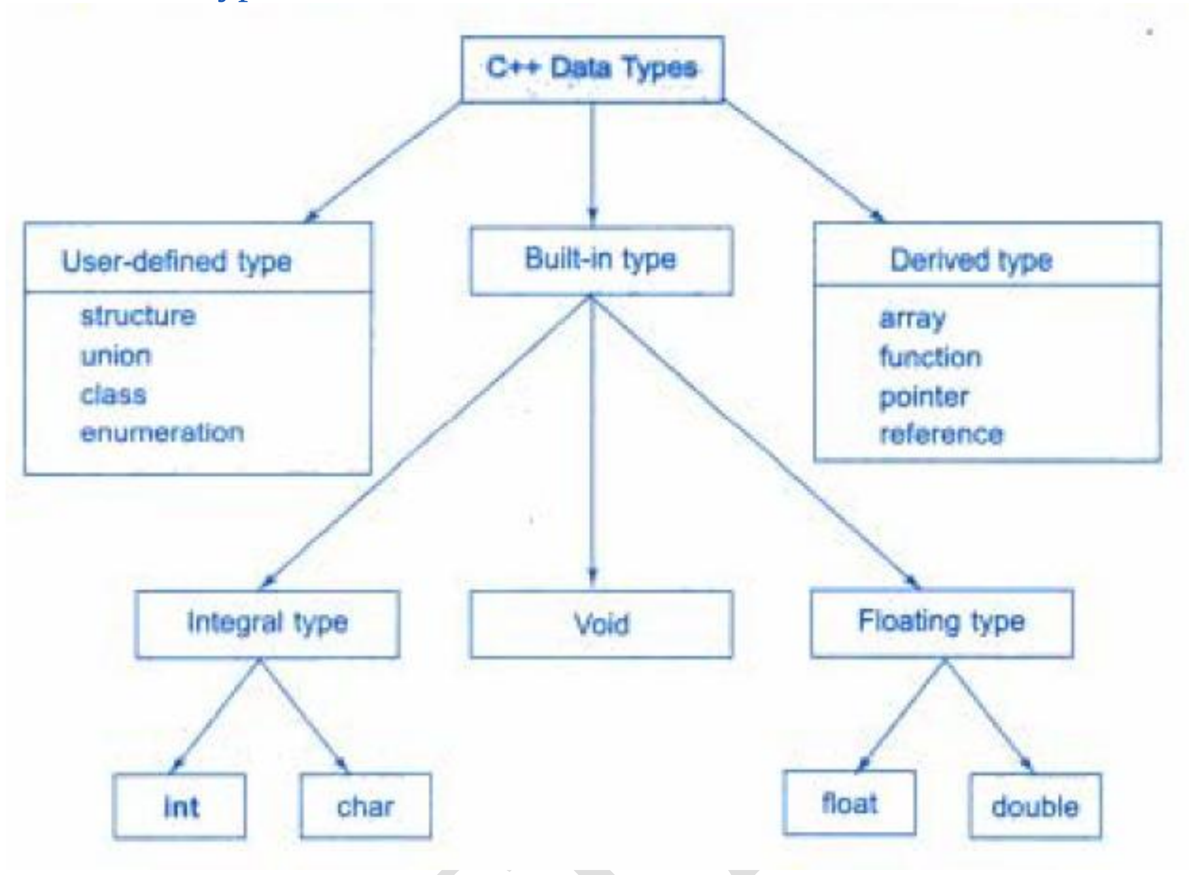
A typical C++ program would contain four sections as shown in Figure. These sections may be placed in separate rode files and then compiled independently or jointly.



It is a common practice to organize a program into three separate files. The class declarations are placed in a header file and the definitions of member functions go into another file. This approach enables the programmer to separate the abstract specification of the interface (class definition) from the implementation details (member functions definition). Finally, the main program that uses the class is placed in a third file which 11includes' the previous two files as well as any other files required. This approach is based on the concept of client-.server model as shown in Fig. 2.4. The class definition including the member functions constitute the server that provides services to the main program known as client. The client uses the server through the public interface of the class.



Basic Data Types



Variables in C++

What are Variables?

Variables are used in C++ where you will need to store any type of values within a program and whose value can be changed during the program execution. These variables can be declared in various ways each having different memory requirements and storing capability. Variables are the name of memory locations that are allocated by compilers, and the allocation is done based on the data type used for declaring the variable.

Variable Definition in C++

A variable definition means that the programmer writes some instructions to tell the compiler to create the storage in a memory location. The syntax for defining variables is:

Syntax:

```
data_type variable_name;
data_type variable_name, variable_name, variable_name;
```

Here *data_type* means the valid C++ data type which includes int, float, double, char, wchar_t, bool and variable list is the lists of variable names to be declared which is separated by commas.

Example:

```
/* variable definition */int    width, height, age;
char   letter;
float  area;
double d;
```

Variable Initialization in C++

Variables are declared in the above example, but none of them has been assigned any value. Variables can be initialized, and the initial value can be assigned along with their declaration.

Syntax:

```
data_type variable_name = value;
```

Example:

```
/* variable definition and initialization */int    width, height=5, age=32;
char   letter='A';
float  area;
double d;
/* actual initialization */width = 10;
area = 26.5;
```

There are some rules that must be in your knowledge to work with C++ variables.
Rules of Declaring variables in C++

1. A variable name can consist of Capital letters A-Z, lowercase letters a-z, digits 0-9, and the underscore character.
2. The first character must be a letter or underscore.
3. Blank spaces cannot be used in variable names.
4. Special characters like #, $ are not allowed.
5. C++ keywords cannot be used as variable names.
6. Variable names are case-sensitive.
7. A variable name can be consisting of 31 characters only if we declare a variable more than one characters compiler will ignore after 31 characters.
8. Variable type can be bool, char, int, float, double, void or wchar_t.

Example:

```
#include <iostream>
using namespace std;
int main()
{
   int x = 5;
   int y = 2;
   int Result;
   Result = x * y;
   cout << Result;
}
```

Constants - Type of Constants in C++

Constants refer to fixed values that the program may not alter. Constants can be of any of the basic data types. The way each constant is represented depends upon its type. Constants are also called **literals**.

Definition: "A constant value is the one which does not change during the execution of a program."

Constant uses the secondary storage area. Constants are those quantities whose value does not vary during the execution of the program i.e. constant has fixed value. Constants in every language are the same. For example, in the C++ language some valid constant are: 53.7 , -44.4, +83.65, "Dinesh", 'M', "2013" , "\n" etc.

In C++ language constants are of two types:

1. Numeric Constant

2. Non-Numeric Constant (Character Constant)

These are further sub-divided into more categories.

Numeric Constant

These have numeric value having combination of sequence of digits i.e. from 0-9 as alone digit or combination of 0-9 with or without decimal point (precision value) having positive or negative sign. These are further sub-divided into two categories as:

(i) Integer Numeric Constant

(ii) Float or Real Numeric Constant

(i) Integer Numeric Constant

An Integer Numeric Constant is a sequence of digits (combination of 0-9 digits without any decimal point or without precision), optionally preceded by a plus or minus sign.

There are 3 types of integer numeric constant namely decimal integer, octal integers and hexadecimal integer.

(a) Decimal Integer Numeric Constant: These have no decimal point in it and are either be alone or be the combination of 0-9 digits. These have either +ve or -ve sign. For example: 1214, -1321, 10,254, -78, +99 etc.

(b) Octal Integer Numeric Constant: These consist of combination of digits from 0-7 with positive or negative sign. It has leading with 0 or 0 (upper or lower case) means Octal or octal. For example: 0317,003, -045 etc.

(c) Hexadecimal Integer Numeric Constant: These have hexadecimal data. It has leading ox, OX, Ox or x, X. These have combination of 0-9 and A-F (a-f) or alone. The letters a-f or A-F represents the numbers 10-15. For example: 0x232, 0x92, 0xACD, 0xAEF etc.

(ii) Float or Real Numeric Constant

Float Numeric Constants consists of a fractional part in their representation; Integer constants are inadequate to represent quantities that vary continuously. These quantities are represented by numbers containing fractional parts like 26.082.

Examples of real constants are: 0.0026, -0.97, 435.29, +487.0, 3.4E-2, 4.5E5

A floating-point constant consists of a sequence of decimal digits, a decimal point, and another sequence of decimal digits. A minus sign can precede the value to denote a negative value. Either the sequence of digits before the decimal point or after the decimal point can be omitted, but not both.

Float Numeric constants are further divided into two parts. One is Mantissa Part and the other is Exponent Part.

(a) Mantissa part: The part without E and having a decimal point is called Mantissa Real part e. g. 45, -22.43, 0.5 etc. it is also called without exponent part.

(b) Exponent part: The exponent part has an E within it. It is also called a scientific notation. Here E has base value 10. it computes the power. For example: 4.2xl02 can be written as 4.2E2, 4.2xlO-5 can be written as 4.2E- 5.-Similarly some more valid real numeric constant are as: 54.73 E -4,51.9 E +11 etc.

Character Constant

Character constants have either a single character or group of characters or a character with backslash used for special purpose. These are further subdivided into three types:

(i) Single Character Constant

(ii) String Character Constant

(iii) Backslash Character Constant

(i) Single Character Constant

Single Character constants are enclosed between single quotes('). For example, 'a' and '%' are both character constants. So these are also called single quote character constant.

For example: 'a', 'M', '5', '+', '1' etc. are some valid single character constant.

(ii) String Character Constant

C++ supports another type of constant: the string. A string is a set of characters enclosed in double quotes ("). For example: "Punar Deep" is a string.

You must not confuse strings with characters. A single character constant is enclosed in single quotes, as in 'a'. However, "a" is a string containing only one letter.

For example: "Dinesh", "Hello", "2013", "2013-2020", "5+3", "?+!" etc. are some valid string character constant. These are used for printing purpose or display purpose in the C++ program's output statements. These can also be used for assigning the string data to the character (string) type variables.

(iii) Backslash Character Constants

Enclosing character constants in single quotes works for most printing characters. A few, however, such as the carriage return, can't be. For this reason, C++ includes the special backslash character constants, shown below, so that you may easily enter these special characters as constants. These are also referred to as escape sequences. You should use the backslash codes instead of their ASCII equivalents to help ensure portability. These are used for special purpose in the C++ language. These are used in output statements like cout etc.

What is Expressions in C++?

A combination of variables, constants and operators that represents a computation forms an expression. Depending upon the type of operands involved in an expression or the result obtained after evaluating expression, there are different categories of an expression. These categories of an expression are discussed here. Constant expressions: The expressions that comprise only constant values are called constant expressions. Some examples of constant expressions are 20, ‘ a ‘ and 2/5+30 .

• **Integral expressions:** The expressions that produce an integer value as output after performing all types of conversions are called **integral expressions.** For example, x, 6*x-y and 10 +int (5.0) are integral expressions. Here, x and yare variables of type into

• **Float expressions:** The expressions that produce floating-point value as output after performing all types of conversions are called **float expressions.** For example, 9.25, x-y and 9+ float (7) are float expressions. Here, x 'and yare variables of type float.

• **Relational or Boolean expressions:** The expressions that produce a bool type value, that is, either true or false are called **relational or Boolean expressions.** For example, x + y<100, m + n==a-b and a>=b + c .are relational expressions.

• **Logical expressions:** The expressions that produce a bool type value after combining two or more relational expressions are called **logical expressions.** For example, x==5 &&m==5 and y>x I I m<=n are logical expressions.

• **Bitwise expressions:** The expressions which manipulate data at bit level are called **bitwise expressions.** For example, a >> 4 and b<< 2 are bitwise expressions.

• **Pointer expressions:** The expressions that give address values as output are called **pointer expressions.** For example, &x, ptr and -ptr are pointer expressions. Here, x is a variable of any type and ptr is a pointer.

• **Special assignment expressions:** An expression can be categorized further depending upon the way the values are assigned to the variables.

What is Statements in C++?

Statements are fragments of the C++ program that are executed in sequence. The body of any function is a sequence of statements. For example:

```
int main()
{
   int n = 1;                        // declaration statement
   n = n + 1;                        // expression statement
   std::cout << "n = " << n << '\n'; // expression statement
   return 0;                         // return statement
}
```

C++ includes the following types of statements:

1) expression statements;

2) compound statements;

3) selection statements;

4) iteration statements;

5) jump statements;

6) declaration statements;

7) try blocks;

What is Operators in C++?

An operator is a symbol that tells the compiler to perform specific mathematical or logical manipulations. C++ is rich in built-in operators and provide the following types of operators

- Arithmetic Operators
- Relational Operators
- Logical Operators
- Bitwise Operators
- Assignment Operators
- Misc Operators

This chapter will examine the arithmetic, relational, logical, bitwise, assignment and other operators one by one.

Arithmetic Operators

There are following arithmetic operators supported by C++ language −

Assume variable A holds 10 and variable B holds 20, then

Show Examples

Operator	Description	Example
+	Adds two operands	A + B will give 30
-	Subtracts second operand from the first	A - B will give -10
*	Multiplies both operands	A * B will give 200
/	Divides numerator by de-numerator	B / A will give 2
%	Modulus Operator and remainder of after an integer division	B % A will give 0
++	**Increment operator**, increases integer value by one	A++ will give 11
--	**Decrement operator**, decreases integer value by one	A-- will give 9

Relational Operators

There are following relational operators supported by C++ language Assume variable A holds 10 and variable B holds 20, then −

Show Examples

Operator	Description	Example
==	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(A == B) is not true.
!=	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B) is true.

>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is true.

Logical Operators

There are following logical operators supported by C++ language.

Assume variable A holds 1 and variable B holds 0, then −

Show Examples

Operator	Description	Example
&&	Called Logical AND operator. If both the operands are non-zero, then condition becomes true.	(A && B) is false.
||	Called Logical OR Operator. If any of the two operands is non-zero, then condition becomes true.	(A || B) is true.
!	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true, then Logical NOT operator will make false.	!(A && B) is true.

Bitwise Operators

Bitwise operator works on bits and perform bit-by-bit operation. The truth tables for &, |, and ^ are as follows −

p	q	p & q	p | q	p ^ q
0	0	0	0	0
0	1	0	1	1
1	1	1	1	0
1	0	0	1	1

Assume if A = 60; and B = 13; now in binary format they will be as follows −

A = 0011 1100

B = 0000 1101

A&B = 0000 1100

A|B = 0011 1101

A^B = 0011 0001

~A = 1100 0011

The Bitwise operators supported by C++ language are listed in the following table. Assume variable A holds 60 and variable B holds 13, then −

Show Examples

Operator	Description	Example
&	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) will give 12 which is 0000 1100
|	Binary OR Operator copies a bit if it exists in either operand.	(A | B) will give 61 which is 0011 1101
^	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) will give 49 which is 0011 0001
~	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.	(~A) will give -61 which is 1100 0011 in 2's complement form due to a signed binary number.
<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 will give 240 which is 1111 0000
>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 will give 15 which is 0000 1111

Assignment Operators

There are following assignment operators supported by C++ language −

Show Examples

Operator	Description	Example
=	Simple assignment operator, Assigns values from right side operands to left side operand.	C = A + B will assign value of A + B into C
+=	Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand.	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand.	C -= A is equivalent to C = C - A
*=	Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand.	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand.	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand.	C %= A is equivalent to C = C % A
<<=	Left shift AND assignment operator.	C <<= 2 is same as C = C << 2
>>=	Right shift AND assignment operator.	C >>= 2 is same as C = C >> 2
&=	Bitwise AND assignment operator.	C &= 2 is same as C = C & 2
^=	Bitwise exclusive OR and assignment operator.	C ^= 2 is same as C = C ^ 2
|=	Bitwise inclusive OR and assignment operator.	C |= 2 is same as C = C | 2

Misc Operators

The following table lists some other operators that C++ supports.

Sr. No	Operator & Description
1	**sizeof** **sizeof operator** returns the size of a variable. For example, sizeof(a), where 'a' is integer, and will return 4.
2	**Condition ? X : Y** **Conditional operator (?)**. If Condition is true then it returns value of X otherwise returns value of Y.
3	**,** **Comma operator** causes a sequence of operations to be performed. The value of the entire comma expression is the value of the last expression of the comma-separated list.
4	**. (dot) and -> (arrow)** **Member operators** are used to reference individual members of classes, structures, and unions.
5	**Cast** **Casting operators** convert one data type to another. For example, int(2.2000) would return 2.
6	**&** **Pointer operator &** returns the address of a variable. For example &a; will give actual address of the variable.
7	***** **Pointer operator *** is pointer to a variable. For example *var; will pointer to a variable var.

Usages of Header Files in C++

Header files contain definitions of **Functions and Variables**, which is imported or used into any C++ program by using the pre-processor #include statement. Header file have an extension ".h" which contains C++ function declaration and macro definition.

Each header file contains information (or declarations) for a particular group of functions. Like **stdio.h** header file contains declarations of standard input and output functions available in C++ which is used for get the input and print the output. Similarly, the header file **math.h** contains declarations of mathematical functions available in C++.

Types of Header Files in C++

- **System header files:** It is comes with compiler.
- **User header files:** It is written by programmer.

Why need of header files

When we want to use any function in our C++ program then first we need to import their definition from C++ library, for importing their declaration and definition we need to include header file in program by using #include. Header file include at the top of any C++ program.

For example if we use clrscr() in C++ program, then we need to include, conio.h header file, because in conio.h header file definition of clrscr() (for clear screen) is written in conio.h header file.

Syntax

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

See another simple example why use header files

Syntax

```
#include<iostream>

int main()
{
   using namespace std;
   cout << "Hello, world!" << endl;
   return 0;
}
```

In above program print message on scree hello world! by using cout but we don't define cout here actually already cout has been declared in a header file called **iostream**.

Decision Making Statement

Decision making statement is depending on the condition block need to be executed or not which is decided by condition.

If the condition is "true" statement block will be executed, if condition is "false" then statement block will not be executed.

In this section we are discuss about if-then (if), if-then-else (if else), and switch statement. In C++ language there are three types of decision making statement.

- if
- if-else
- switch

if-then Statement

if-then is most basic statement of Decision making statement. It tells to program to execute a certain part of code only if particular condition or test is true.

Syntax

```
if(condition)
{
..........
..........
}
```

Example

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

void main()
{
int a=10;
if(a<10)
{
cout<<"a is less than 10";
}
getch();
}
```

Output

```
a is less than 10
```

- Constructing the body of "if" statement is always optional, Create the body when we are having multiple statements.
- For a single statement, it is not required to specify the body.
- If the body is not specified, then automatically condition part will be terminated with next semicolon (;).

else

It is a keyword, by using this keyword we can create a alternative block for **if** part. Using else is always optional i.e, it is recommended to use when we are having alternate block of condition.

In any program among if and else only one block will be executed. When if condition is false then else part will be executed, if part is executed then automatically else part will be ignored.

Syntax

```
if(condition)
{
........
statements
........
}
else
{
........
statements
........
}
```

Example

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

void main()
{
int age=40;
if(a<18)
{
cout<<"you are child";
}
else
{
cout<<"you are young";
}
getch();
}
```

Output

```
you are young
```

Output

```
Good morning
```

Switch Case in C++

The **switch** statement in C++ language is used to execute the code from multiple conditions or case. It is same like if else-if ladder statement.

A switch statement work with byte, short, char and int primitive data type, it also works with enumerated types and string.

Syntax

```
switch(expression or variable)
{
case  value:
//statements
// any number of case statements
break;  //optional
default: //optional
//statements
}
```

Rules for apply switch

- The switch expression must be of integer or character type.
- With switch statement use only byte, short, int, char data type.
- With switch statement not use float data type.
- You can use any number of case statements within a switch.
- The case value can be used only inside the switch statement.
- Value for a case must be same as the variable in switch.

Valid statement for switch

Switch Case in Example C++

```
int x;
byte y;
short z;
char a,b;
// Invalid
```

Syntax

```
switch(ch)
{
case 1:
statement 1;
break;
case 2:
statement 2;
break;
}
```

In this ch can be integer or char and can not be float or any other data type.

Example of Switch Case in C++

```
#include<iostream.h>
#include<conio.h>
void main()
{
int ch;
clrscr();
cout<<"Enter any number (1 to 7)";
cin>>ch;
switch(ch)
{
case  1:
cout<<"Today is Monday";
break;
case  2:
cout<<"Today is Tuesday";
break;
case  3:
cout<<"Today is Wednesday";
break;
case  4:
cout<<"Today is Thursday";
break;
case  5:
cout<<"Today is Friday";
break;
case  6:
cout<<"Today is Saturday";
break;
case  7:
cout<<"Today is Sunday";
break;
default:
cout<<"Only enter value 1 to 7";
}
getch();
}
```

Output

```
Enter any number (1 to 7): 5
Today is Friday
```

Note: In switch statement default is optional but when we use this in switch default is executed at last whenever all cases are not satisfied the condition.

Example of Switch without break in C++

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

void main()
{
int ch;
clrscr();
cout<<"Enter any number (1 to 7)";
cin>>ch;
switch(ch)
{
case  1:
cout<<"Today is Monday";
case  2:
cout<<"Today is Tuesday";
case  3:
cout<<"Today is Wednesday";
case  4:
cout<<"Today is Thursday";
case  5:
cout<<"Today is Friday";
case  6:
cout<<"Today is Saturday";
case  7:
cout<<"Today is Sunday";
default:
cout<<"Only enter value 1 to 7";
}
getch();
}
```

Output

```
Enter any number (1 to 7): 4
Today is Thursday
Today is Friday
Today is Saturday
Today is Sunday
```

Note: In switch statement when you not use break keyword after any statement then all the case after matching case will be execute. In above code case 4 is match and after case 4 all case are execute from case-4 to case-7.

C++ program for calculator

Example Calculator

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

void main()
{
char choice;
int a,b,res=0;
clrscr();
cout<<"Enter first value: ";
cin>>a;
cout<<"\n Enter operator: ";
choice=getch();
cout<<"\n Enter second value: ";
cin>>b;

switch(choice)
{
case '+':
  res=a+b;
  cout<<"Sum: ",res;
break;
case '-':
  res=a-b;
  cout<<"Difference: ",res;
break;
case '*':
  res=a*b;
  cout<<"Product: ",res;
break;
case '/':
  res=a/b;
  cout<<"Quotient: ",res;
break;
default:
  cout<<"Enter Valid Operator!";
}
getch();
}
```

Output

```
Enter First number: 6
Enter operator : +
Enter Second number: 10
Sum: 16
```

Control Flow Statement

Set of instructions given to the compiler to execute set of statements until condition becomes false is called loops. The basic purpose of loop is code repetition that means same code repeated again and again.

Why use Loop

Where need repetition of same code a number of times at that place use Loop in place of writing more than one statements. The way of the repetition will be forms a circle that's why repetition statements are called loops.

Advantage with looping statement

- Reduce length of Code
- take less memory space.

Types of Loops.

There are three type of Loops available in 'C' programming language.

- while loop
- for loop
- do-while

Difference Between Conditional and Looping Statement in C++

Conditional statement executes only once in the program where as looping statements executes repeatedly several number of time.

While loop

In **while loop** First check the condition if condition is true then control goes inside the loop body other wise goes outside the body. **while loop** will be repeats in clock wise direction.

Syntax

```
Assignment;
while(condition)
{
Statements;
............
Increment/decrement (++ or --);
}
```

Note: If while loop condition never false then loop become infinite loop.

Example of while loop

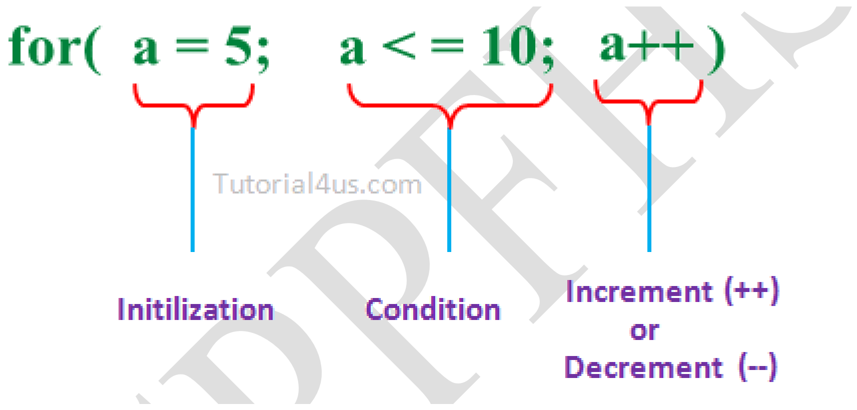
```
#include<iostream.h>
#include<conio.h>

void main()
{
int i;
clrscr();
i=1;
while(i<5)
{
cout<<endl<<i;
i++;
}
getch();
}
```

Output

```
1
2
3
4
```

For loop
For loop contains 3 parts.

- Initialization
- Condition
- Iteration



- When we are working with for loop always execution process will start from initialization block.
- After initialization block control will pass to condition block, if condition is evaluated as true then control will pass to statement block.
- After execution of the statement block control will pass to iteration block, from iteration it will pass back to the condition.
- Always repetitions will happen beginning condition, statement block and iteration only.
- Initialization block will be executed only once when we are entering into the loop first time.

- When we are working with for loop everything is optional but mandatory to place 2 semicolons (; ;)

Example

```
while()              // Error
for( ; ; )                        // valid
```

Example of for loop

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

void main()
{
int i;
clrscr();
for(i=1;i<5;i++)
{
cout<<endl<<i;
}
getch();
}
```

Output

```
1
2
3
4
```

- when we are working with the for loop if condition part is not given then it will repeats infinite times, because condition part will replace it non-zero. So it is always true like.
  for(; 1;)
- Whenever we are working with for loop it repeats in antilock direction.
- In for loop also rechecking process will be occurred that is before execution of the statement block, condition part will evaluated.

Example

```
while(0)        // no repetition
for( ; 0; )     // it will repeats 1 time
```

In for loop whenever the condition part is replace with constant zero then it will repeats 1 st become nember of instance become 1 at the time of compilation.

Example

```
int i;
i=0;
while(i)        // no repetition

for( ; ; )      // no repetition
```

Always execution process is faster than while loop when we are working with for.

do-while

In do-while loop post checking of the statement block condition part will be executed.

Syntax

```
do
{
Statements;
........
Increment/decrement (++ or --)
}
while();
```

When use Do-While Loop

When we need to repeat the statement block atleast 1 time then we use do-while loop.

Example of do-while Loop

Example

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

void main()
{
```

```
int i;
clrscr();
i=1;
do
{
cout<<endl<<i;
i++;
}
while(i<5);
getch();
}
```

Output

```
1
2
3
4
```

C++ break Statement

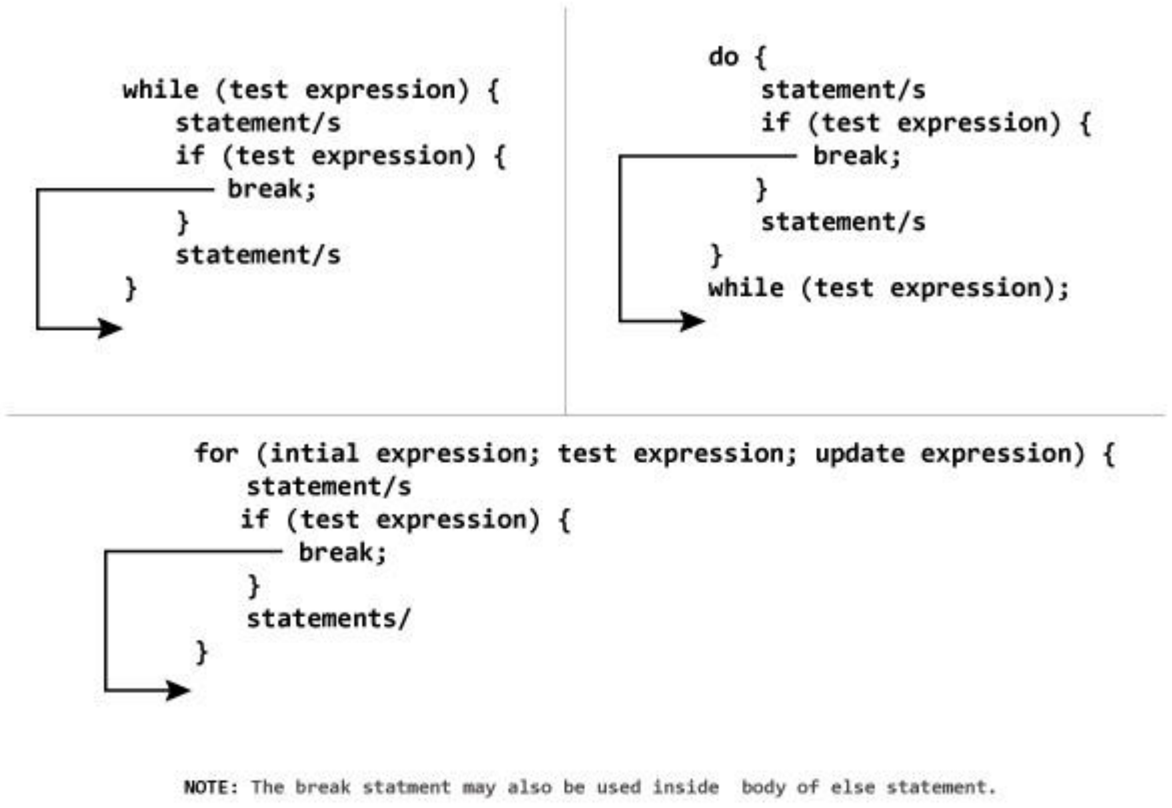
The break; statement terminates a loop (for, while and do..while loop) and a switch statement immediately when it appears.

Syntax of break

```
break;
```

In real practice, break statement is almost always used inside the body of conditional statement (if...else) inside the loop.

How break statement works?



```
while (test expression) {
    statement/s
    if (test expression) {
        break;
    }
    statement/s
}
```

```
do {
    statement/s
    if (test expression) {
        break;
    }
    statement/s
}
while (test expression);
```

```
for (intial expression; test expression; update expression) {
    statement/s
    if (test expression) {
        break;
    }
    statements/
}
```

NOTE: The break statment may also be used inside body of else statement.

Example 1: C++ break
C++ program to add all number entered by user until user enters 0.

```
// C++ Program to demonstrate working of break statement

#include <iostream>
using namespace std;
int main() {
    float number, sum = 0.0;

    // test expression is always true
    while (true)
    {
        cout << "Enter a number: ";
        cin >> number;

        if (number != 0.0)
        {
            sum += number;
        }
        else
        {
            // terminates the loop if number equals 0.0
            break;
        }

    }
    cout << "Sum = " << sum;

    return 0;
}
```

Output

```
Enter a number: 4
Enter a number: 3.4
Enter a number: 6.7
Enter a number: -4.5
Enter a number: 0
Sum = 9.6
```

C++ continue Statement

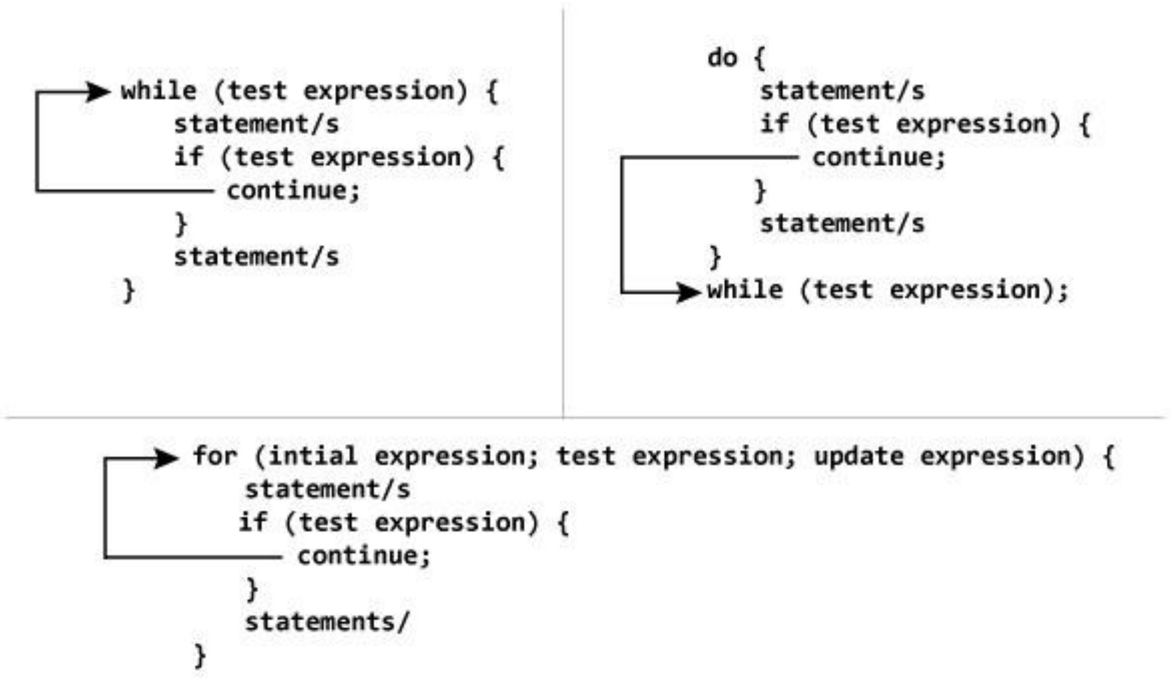
It is sometimes necessary to skip a certain test condition within a loop. In such case, continue; statement is used in C++ programming.

Syntax of continue

```
continue;
```

In practice, continue; statement is almost always used inside a conditional statement.

Working of continue Statement



```
while (test expression) {
    statement/s
    if (test expression) {
        continue;
    }
    statement/s
}
```

```
do {
    statement/s
    if (test expression) {
        continue;
    }
    statement/s
}
while (test expression);
```

```
for (intial expression; test expression; update expression) {
    statement/s
    if (test expression) {
        continue;
    }
    statements/
}
```

Example 2: C++ continue
C++ program to display integer from 1 to 10 except 6 and 9.

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

int main()
{
   for (int i = 1; i <= 10; ++i)
   {
      if ( i == 6 || i == 9)
      {
         continue;
      }
      cout << i << "\t";
   }
   return 0;
}
```

Output

```
1        2        3        4        5    7    8        10
```

In above program, when i is 6 or 9, execution of statement cout << i << "\t"; is skipped inside the loop using continue; statement.