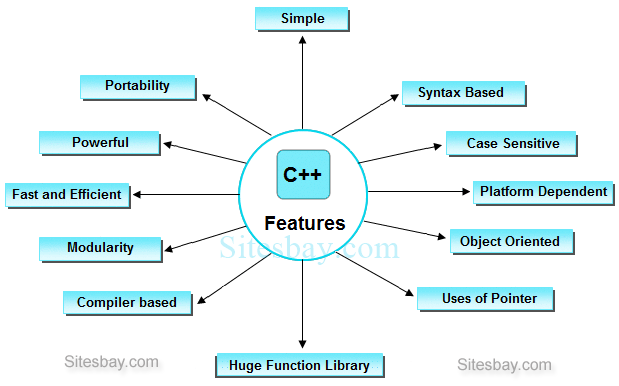
## Overview of C++

**C++** is a computer programming language developed in 1980 by **BjarneStroustrup** at the Bell Telephone Laboratories. C++ is an **Object Oriented Programming Language**, which follow oops concept like, inheritance, encapsulation, abstraction and polymorphism.

## Features of C++

C++ is object oriented programming language and it is a very simple and easy language, It is the enhanced form of C programming language. this language have following features and here we discuss some important features of C++.

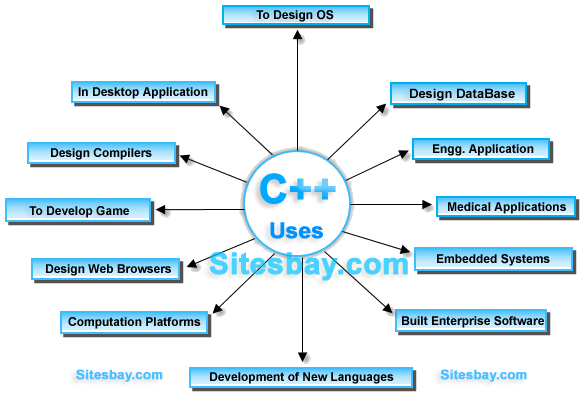


### Important Features of C++

* Simple
* Portability
* Powerful
* Platform dependent
* Object oriented oriented
* Case sensitive
* Compiler based
* Syntax based language
* Use of Pointers

**Applications of C++**

Mainly C++ Language is used for **Develop Desktop application** and **system software**. Some application of C++ language are given below.

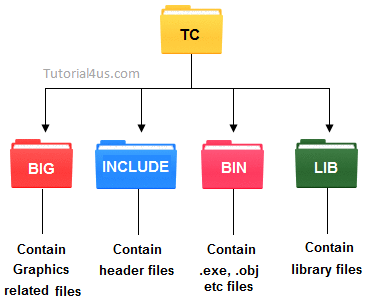


* For Develop Graphical related application like computer and mobile games.
* To evaluate any kind of mathematical equation use C++ language.
* C++ Language are also used for design OS. Like window xp.
* Google also use C++ for Indexing
* Few parts of apple OS X are written in C++ programming language.
* Internet browser Firefox are written in C++ programming language
* All major applications of adobe systems are developed in C++ programming language. Like Photoshop, ImageReady, Illustrator and Adobe Premier.
* Some of the Google applications are also written in C++, including Google file system and Google Chromium.
* C++ are used for design database like MySQL.

**How to Install C++ Editor**

Installation of TC is very simple just download turbo C or C++ and run .exe files

When you install the Turbo C or C++ compiler on your system, then TC directory is created on the hard disk and various sub directories such as INCLUDE, and LIB etc. are created under TC.



* **INCLUDE :**Contain the header files of C and C++.
* **LIB:**Contain the library files of C and C++.
* **BGI:**Contain Graphics related files.
* **BIN:**Contain .exe, .objetc files.

**Turbo C For Window(xp, 7, 8)**

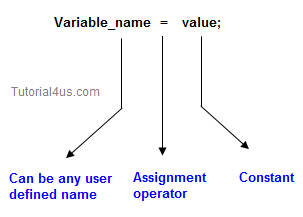
Previously Turbo C are not work properly on Window 7 and its above versions. Here you can get Turbo C for Window 7 and its above versions.

## Constant in C++

It is an identifier whose value can not be changed at the execution time of program. In general **constant** can be used to represent as fixed values in a C++ program. Constants are classified into following types.

## constant in c++Variable in C++

Variable is an identifier which holds data or another one variable is an identifier whose value can be changed at the execution time of program. Variable is an identifier which can be used to identify input data in a program.



## Syntax

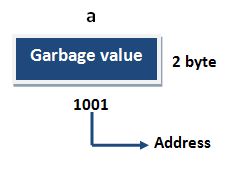
Variable\_name = value;

## Variable declarations

This is the process of allocating sufficient memory space for the data in term of variable.

## Syntax

Datatypevariable\_name; int a;



If no input values are assigned by the user than system will gives a default value called garbage value.

## Garbage value

Garbage value can be any value given by system and that is no way related to correct programs.  
It is a disadvantage and it can overcome using variable initialization.

## Variable initialization

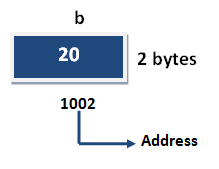
It is the process of allocating sufficient memory space with user defined values.

## Syntax

Datatypenariable\_name=value;

## Example

int b = 30;



## Variable assignment

It is a process of assigning a value to a variable.

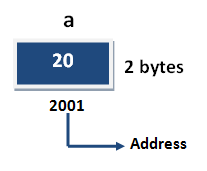
## Syntax

Variable\_Name = value

## Example

int a= 20;

int b;

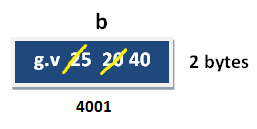


## Example

b = 25; // --> direct assigned variable

b = a; // --> assigned value in term of variable

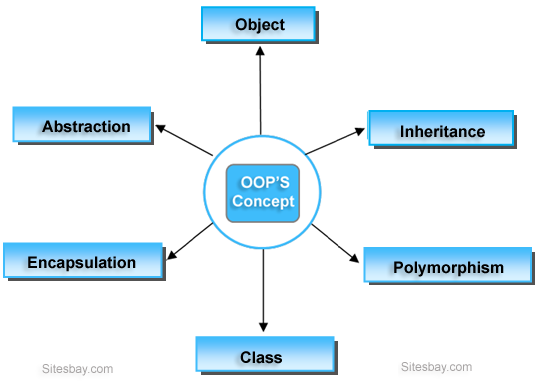
b = a+15; // --> assigned value as term of expression



**Oops Concept in C++**

The main purpose of C++ programming was to add object orientation to the C programming language, which is in itself one of the most powerful programming languages. If any programming language follow below oops concept then that language called object oriented programming language.

* Object
* Class
* Encapsulation
* Abstraction
* Inheritance
* Polymorphism



**Object**

**Object** is the physical as well as logical entity where as **class** is the only logical entity.

**Class**

**Class:** Class is a blue print which is containing only list of variables and method and no memory is allocated for them. A class is a group of objects that has common properties.

**Encapsulation**

**Encapsulation** is a process of wrapping of data and methods in a single unit is called encapsulation. Encapsulation is achieved in C++ language by class concept. The main advantage of using of encapsulation is to secure the data from other methods, when we make a data private then these data only use within the class, but these data not accessible outside the class.

**Abstraction**

**Abstraction** is the concept of exposing only the required essential characteristics and behavior with respect to a context.

Hiding of data is known as **data abstraction**. In object oriented programming language this is implemented automatically while writing the code in the form of class and object.

**Inheritance**

The process of obtaining the data members and methods from one class to another class is known as **inheritance**. It is one of the fundamental features of object-oriented programming.

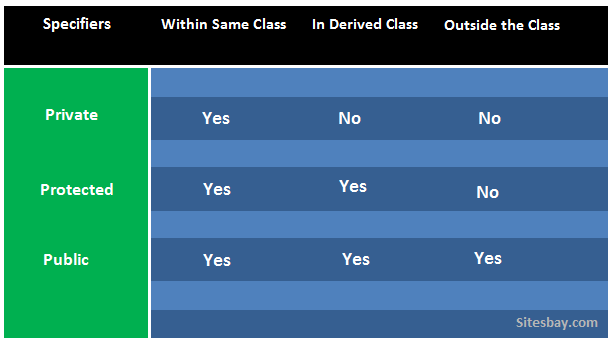
**Polymorphism**

The process of representing one Form in multiple forms is known as **Polymorphism**. Here one form represent original form or original method always resides in base class and multiple forms represents overridden method which resides in derived classes.

**Access Specifiers in C++**

**Access specifiers** in C++ define how the members of the class can be accessed. C++ has 3 new keywords introduced, namely.

* public
* private
* protected



The keywords public, private, and protected are called access specifiers. A class can have multiple public, protected, or private labeled sections.

**Note: By default, all members and function of a class is private i.e if no access specifier is specified**.

### Private and Public Access Specifier Example in C++

## Example

#include<iostream.h>

#include<conio.h>

**class** A

{

**private**:

**int** a;

**public**:

**int** b;

**public**:

**void** show()

{

a=10;

b=20;

clrscr();

//Every members can be access here, same class

cout<<"\nAccessing variable within the class"<<endl;

cout<<"Value of a: "<<a<<endl;

cout<<"Value of b: "<<b<<endl;

}

};

**void** main()

{

A obj;// create object

obj.show();

cout<<"\nAccessing variable outside the class"<<endl;

//'a' cannot be accessed as it is private

//cout<<"value of a: "<<obj.a<<endl;

//'b' is public as can be accessed from any where

cout<<"value of b: "<<obj.b<<endl;

getch();

}

**Note:**If here, we access variable ain side main method it will give compile time error

## Output

Accessing variable within the class

value of a: 10

value of b: 20

value of c: 30

Accessing variable outside the class

Value of b: 20

### Protected Access Specifier in C++

It is similar to private access specifier. It makes class member inaccessible outside the class. But they can be accessed by any subclass of that class.

#include<iostream.h>

#include<conio.h>

//using namespace std;

**class**Declaration

{

**private**:

**int** a;

**public**:

**int** b;

**protected**:

**int** c;

**public**:

**void** show()

{

a=10;

b=20;

c=30;

//Every members can be access here, same class

cout<<"\nAccessing variable within the class"<<endl;

cout<<"Value of a: "<<a<<endl;

cout<<"Value of b: "<<b<<endl;

cout<<"Value of c: "<<c<<endl;

}

};

**class**Sub\_class:**public**Declaration

{

**public**:

**void** show()

{

b=5;

c=6;

cout<<"\nAccessing variable in sub the class"<<endl;

// a is not accessible here it is private

//cout<<"Value of a: "<<a<<endl;

//b is public so it is accessible any where

cout<<"Value of b: "<<b<<endl;

//'c' is declared as protected, so it is accessible in sub class

cout<<"Value of c: "<<c<<endl;

}

};

**void** main()

{

clrscr();

Declaration d;// create object

d.show();

Sub\_class s;// create object

s.show();// Sub class show() function

cout<<"\nAccessing variable outside the class"<<endl;

//'a' cannot be accessed as it is private

//cout<<"value of a: "<<d.a<<endl;

//'b' is public as can be accessed from any where

cout<<"value of b: "<<d.b<<endl;

//'c' is protected and cannot be accesed here

//cout<<"value of c: "<<d.c<<endl;

getch();

}

## Output

Accessing variable within the class

value of a: 10

value of b: 20

value of c: 30

Accessing variable in sub class

value of b: 5

value of c: 6

Accessing variable outside the class

Value of b: 20

## Difference between Class and Object in C++

|  |  |  |
| --- | --- | --- |
|  | **Class** | **Object** |
| 1 | Class is a container which collection of variables and methods. | object is a instance of class |
| 2 | No memory is allocated at the time of declaration | Sufficient memory space will be allocated for all the variables of class at the time of declaration. |
| 3 | One class definition should exist only once in the program. | For one class multiple objects can be created. |

## Inheritance in C++

The process of obtaining the data members and methods from one class to another class is known as **inheritance**. It is one of the fundamental features of object-oriented programming.

### Important points

* In the inheritance the class which is give data members and methods is known as base or super or parent class.
* The class which is taking the data members and methods is known as sub or derived or child class.

## inheritance in C++

### Advantage of inheritance

If we develop any application using this concept than that application have following advantages,

* Application development time is less.
* Application take less memory.
* Application execution time is less.
* Application performance is enhance (improved).
* Redundancy (repetition) of the code is reduced or minimized so that we get consistence results and less storage cost.

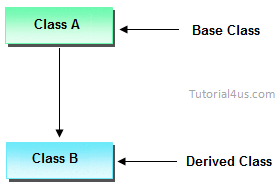
## Tpyes of Inheritance

Based on number of ways inheriting the feature of base class into derived class it have five types they are:

* Single inheritance
* Multiple inheritance
* Hierarchical inheritance
* Multiple inheritance
* Hybrid inheritance

### Single inheritance

In single inheritance there exists single base class and single derived class.



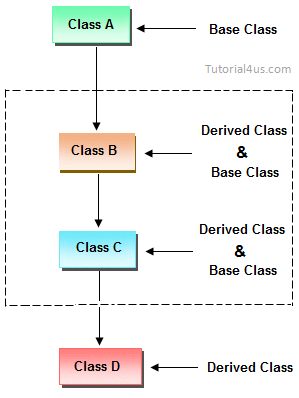
### Multiple inheritances

In multiple inheritances there exists single base class, single derived class and multiple intermediate base classes.

**Single base class + single derived class + multiple intermediate base classes.**

### Intermediate base classes

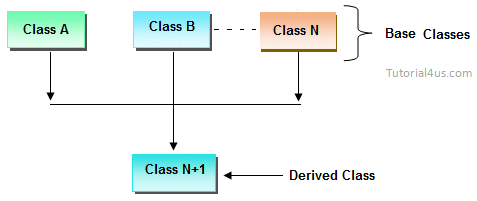
An intermediate base class is one in one context with access derived class and in another context same class access base class.



Hence all the above three inheritance types are supported by both classes and interfaces.

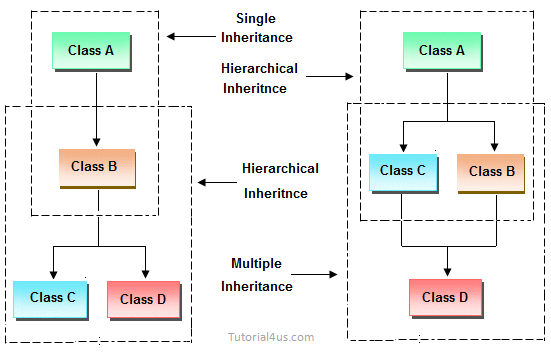
### Multiple inheritance

In multiple inheritance there exist multiple classes and singel derived class.



### Hybrid inheritance

Combination of any inheritance type



### Inheriting the feature from base class to derived class

In order to inherit the feature of base class into derived class we use the following syntax

## Syntax

class classname-2 : classname-1

{

variable declaration;

method declaration;

}

### Explanation

* classname-1 and classname-2 represents name of the base and derived classes respectively.
* **:**is operator which is used for inheriting the features of base class into derived class it improves the functionality of derived class.

## Example of Inheritance in C++

#include<iostream.h>

#include<conio.h>

**class** employee

{

**public**:

**int** salary;

};

**class** developer :**public** employee

{

employee e;

**public**:

**void** salary()

{

cout<<"Enter employee salary: ";

cin>>e.salary;// access base class data member

cout<<"Employee salary: "<<e.salary;

}

};

**void** main()

{

clrscr();

developerobj;

obj.salary();

getch();

}

## Output

Enter employee salary: 50000

Employee salary: 50000

## Encapsulation in C++

**Encapsulation** is a process of wrapping of data and methods in a single unit. It is achieved in C++ language by class concept.

**Benefits of encapsulation**

* Provides abstraction between an object and its clients.
* Protects an object from unwanted access by clients.
* Example: A bank application forbids a client to change an Account's balance.

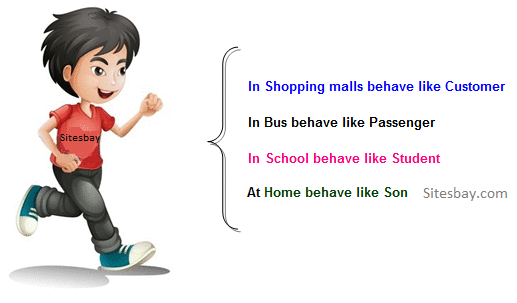
## Polymorphism in C++

The process of representing one Form in multiple forms is known as **Polymorphism**. Here one form represent original form or original method always resides in base class and multiple forms represents overridden method which resides in derived classes.

Polymorphism is derived from 2 greek words: **poly** and morphs. The word "poly" means many and **morphs** means forms. So polymorphism means many forms.

## Real life example of Polymorphism in C++

Suppose if you are in class room that time you behave like a student, when you are in market at that time you behave like a customer, when you at your home at that time you behave like a son or daughter, Here one person have different-different behaviors.



**Type of polymorphism**

* Compile time polymorphism
* Run time polymorphism

**Compile time polymorphism**

In C++ programming you can achieve compile time polymorphism in two way, which is given below;

* Method overloading
* Method overriding

## Method Overloading in C++

Whenever same method name is exiting multiple times in the same class with different number of parameter or different order of parameters or different types of parameters is known as **method overloading**. In below example method "sum()" is present in Addition class with same name but with different signature or arguments.

## Example of Method Overloading in C++

#include<iostream.h>

#include<conio.h>

**class**Addition

{

**public**:

**void** sum(**int** a,**int** b)

{

cout<<a+b;

}

**void** sum(**int** a,**int** b,**int** c)

{

cout<<a+b+c;

}

};

**void** main()

{

clrscr();

Additionobj;

obj.sum(10,20);

cout<<endl;

obj.sum(10,20,30);

}

## Output

30

60

## Method Overriding in C++

Define any method in both base class and derived class with same name, same parameters or signature, this concept is known as **method overriding**. In below example same method "show()" is present in both base and derived class with same name and signature.

## Example of Method Overriding in C++

#include<iostream.h>

#include<conio.h>

**Class** Base

{

**public**:

**void** show()

{

cout<<"Base class";

}

};

**Class** Derived: **public** Base

{

**public**:

**void** show()

{

cout<<"Derived Class";

}

}

**int**mian()

{

Base b;//Base class object

Derived d;//Derived class object

b.show();//Early Binding Ocuurs

d.show();

getch();

}

## Output

Base class

Derived Class

## Run Time Polymorphism

In C++ Run time polymorphism can be achieve by using [virtual function](http://www.sitesbay.com/cpp/cpp-virtual-function).

## Virtual Function in C++

A **virtual function** is a member function of class that is declared within a base class and re-defined in derived class.

When you want to use same function name in both the base and derived class, then the function in base class is declared as virtual by using the **virtual** keyword and again re-defined this function in derived class without using virtual keyword.

## Syntax

Virtual return\_type function\_name()

{

.......

.......

}

## Virtual Function Example

#include<iostream.h>

#include<conio.h>

**class** A

{

**public**:

**Virtual void** show()

{

cout<<"Hello base class";

}

};

**class** B :**public** A

{

**public**:

**void** show()

{

cout<<"Hello derive class";

}

};

**void** main()

{

clrsct();

Aaobj;

B bobj;

A \*bptr;

bptr=&aobj;

bptr->show();// call base class function

bptr=&bobj;

bptr->show();// call derive class function

getch();

}

## Output

Hello base class

Hello derive class

**Constructor in C++**

A **Constructor** is a special member method which will be called implicitly (automatically) whenever an object of class is created. In other words, it is a member function which initializes a class which is called automatically whenever a new instance of a class is created.

**Features of Constructor**

* The same name as the class itself.
* no return type.

**Syntax**

classname()

{

....

}

**Note:**If you do not specify a constructor, the compiler generates a default constructor for you (expects no parameters and has an empty body).

**Why use constructor ?**

The main use of constructor is placing user defined values in place of default values.

**How Constructor eliminate default values ?**

Constructor are mainly used for eliminate default values by user defined values, whenever we create an object of any class then its allocate memory for all the data members and initialize there default values. To eliminate these default values by user defined values we use constructor.

**Example of Constructor in C++**

#include<iostream.h>

#include<conio.h>

**class** sum

{

**int**a,b,c;

sum()

{

a=10;

b=20;

c=a+b;

cout<<"Sum: "<<c;

}

};

**void** main()

{

sum s;

getch();

}

**Output**

Sum: 30

In above example when we create an object of "Sum" class then constructor of this class call and initialize user defined value in a=10 and b=20. And here we no need to call sum() constructor.

**Destructor**

**Destructor** is a member function which deletes an object. A destructor function is called automatically when the object goes out of scope:

**When destructor call**

* when program ends
* when a block containing temporary variables ends
* when a delete operator is called

**Features of destructor**

* The same name as the class but is preceded by a tilde (~)
* no arguments and return no values

**Syntax**

~classname()

{

......

}

**Note:**If you do not specify a destructor, the compiler generates a default destructor for you.

**Example of Destructor in C++**

#include<iostream.h>

#include<conio.h>

**class** sum

{

**int**a,b,c;

sum()

{

a=10;

b=20;

c=a+b;

cout<<"Sum: "<<c;

}

~sum()

{

cout<<<<endl;"call destructor";

}

delay(500);

};

**void** main()

{

sum s;

cout<<<<endl;"call main";

getch();

}

**Output**

Sum: 30

call main

call destructor

**Explanation:**In above example when you create object of class sum auto constructor of class is call and after that control goes inside main and finally before end of program destructor is call.

**What is a copy constructor ?**

A copy constructor is a special constructor in the C++ programming language for creating a new object as a copy of an existing object.

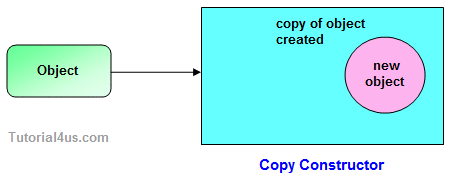
**Example**

class\_name (**const**class\_name&);

{

......

}



## Exception Handling in C++

The process of converting system error messages into user friendly error message is known as **Exception handling**. This is one of the powerful feature of C++ to handle run time error and maintain normal flow of C++ application.

## Exception

An exception is an event, which occurs during the execution of a program, that disrupts the normal flow of the program's Instructions.

## Handling the Exception

Handling the exception is nothing but converting system error message into user friendly error message. Use Three keywords for Handling the Exception in C++ Language, they are;

1. try
2. catch
3. throw

### Syntax for handling the exception

## Example of Exception Handling in C++

try

{

// causes executions code

}

catch(ExceptionName e1 )

{

// catch block

}

catch(ExceptionName e2 )

{

// catch block

}

catch(ExceptionNameeN )

{

// catch block

}

## Try Block

It is one of the block in which we write the block of statements which causes executions at run time in other words try block always contains problematic statements.

## Catch block

It is one of the block in which we write the block of statements which will generates user friendly error messages in other words catch block will suppose system error messages.

## Example without Exception Handling

## Example

#include<iostream.h>

#include<conio.h>

**void** main()

{

**int** a,ans;

a=10;

ans=a/0;

cout<<"Result: "<<ans;

}

## Output

Abnormally terminate program

## Example of Exception Handling

## Example

#include<iostream.h>

#include<conio.h>

**void** main()

{

**int** a=10,ans=0;

**try**

{

ans=a/0;

}

**catch**(**int** i)

{

cout<<"Denominator not be zero";

}

}

## Output

Denominator not be zero

## Header Files in C++

**Types of Header Files in C++**

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

# Enumeration (or enum) in C

Enumeration (or enum) is a user defined data type in C. It is mainly used to assign names to integral constants, the names make a program easy to read and maintain.

enum State {Working = 1, Failed = 0};

The keyword ‘enum’ is used to declare new enumeration types in C and C++. Following is an example of enum declaration.

// The name of enumeration is "flag" and the constant

// are the values of the flag. By default, the values

// of the constants are as follows:

// constant1 = 0, constant2 = 1, constant3 = 2 and

// so on.

enum flag{constant1, constant2, constant3, ....... };

Variables of type enum can also be defined. They can be defined in two ways:

// In both of the below cases, "day" is

// defined as the variable of type week.

enum week{Mon, Tue, Wed};

enum week day;

// **Or**

enum week{Mon, Tue, Wed}day;

|  |
| --- |
| // An example program to demonstrate working  // of enum in C  #include<stdio.h>    enum week{Mon, Tue, Wed, Thur, Fri, Sat, Sun};    int main()  {      enum week day;      day = Wed;      printf("%d",day);      return 0;  } |

Output:

2

In the above example, we declared “day” as the variable and the value of “Wed” is allocated to day, which is 2. So as a result, 2 is printed.

Another example of enumeration is:

|  |
| --- |
| // Another example program to demonstrate working  // of enum in C  #include<stdio.h>    enum year{Jan, Feb, Mar, Apr, May, Jun, Jul,            Aug, Sep, Oct, Nov, Dec};    int main()  {     int i;     for (i=Jan; i<=Dec; i++)        printf("%d ", i);       return 0;  } |

Output:

0 1 2 3 4 5 6 7 8 9 10 11

In this example, the for loop will run from i = 0 to i = 11, as initially the value of i is Jan which is 0 and the value of Dec is 11.

**Interesting facts about initialization of enum.**  
**1.** Two enum names can have same value. For example, in the following C program both ‘Failed’ and ‘Freezed’ have same value 0.

|  |
| --- |
| #include <stdio.h>  enum State {Working = 1, Failed = 0, Freezed = 0};    int main()  {     printf("%d, %d, %d", Working, Failed, Freezed);     return 0;  } |

Output:

1, 0, 0

**2.** If we do not explicitly assign values to enum names, the compiler by default assigns values starting from 0. For example, in the following C program, sunday gets value 0, monday gets 1, and so on.

|  |
| --- |
| #include <stdio.h>  enum day {sunday, monday, tuesday, wednesday, thursday, friday, saturday};    int main()  {      enum day d = thursday;      printf("The day number stored in d is %d", d);      return 0;  } |

Output:

The day number stored in d is 4

**3.** We can assign values to some name in any order. All unassigned names get value as value of previous name plus one.

|  |
| --- |
| #include <stdio.h>  enum day {sunday = 1, monday, tuesday = 5,            wednesday, thursday = 10, friday, saturday};    int main()  {      printf("%d %d %d %d %d %d %d", sunday, monday, tuesday,              wednesday, thursday, friday, saturday);      return 0;  } |

Output:

1 2 5 6 10 11 12

**4.** The value assigned to enum names must be some integeral constant, i.e., the value must be in range from minimum possible integer value to maximum possible integer value.

**5.** All enum constants must be unique in their scope. For example, the following program fails in compilation.

|  |
| --- |
| enum state  {working, failed};  enum result {failed, passed};    int main()  { return 0; } |

Output:

Compile Error: 'failed' has a previous declaration as 'state failed'

**Exercise:**  
Predict the output of following C programs

Program 1:

|  |
| --- |
| #include <stdio.h>  enum day {sunday = 1, tuesday, wednesday, thursday, friday, saturday};    int main()  {      enum day d = thursday;      printf("The day number stored in d is %d", d);      return 0;  } |

Program 2:

|  |
| --- |
| #include <stdio.h>  enum State {WORKING = 0, FAILED, FREEZED};  enum State currState = 2;    enum State FindState() {      return currState;  }    int main() {     (FindState() == WORKING)? printf("WORKING"): printf("NOT WORKING");     return 0;  } |

**Enum vs Macro**  
We can also use macros to define names constants. For example we can define ‘Working’ and ‘Failed’ using following macro.

filter\_none

brightness\_4

|  |
| --- |
| #define Working 0  #define Failed 1  #define Freezed 2 |

There are multiple advantages of using enum over macro when many related named constants have integral values.  
a) Enums follow scope rules.  
b) Enum variables are automatically assigned values. Following is simpler

|  |
| --- |
| enum state  {Working, Failed, Freezed}; |