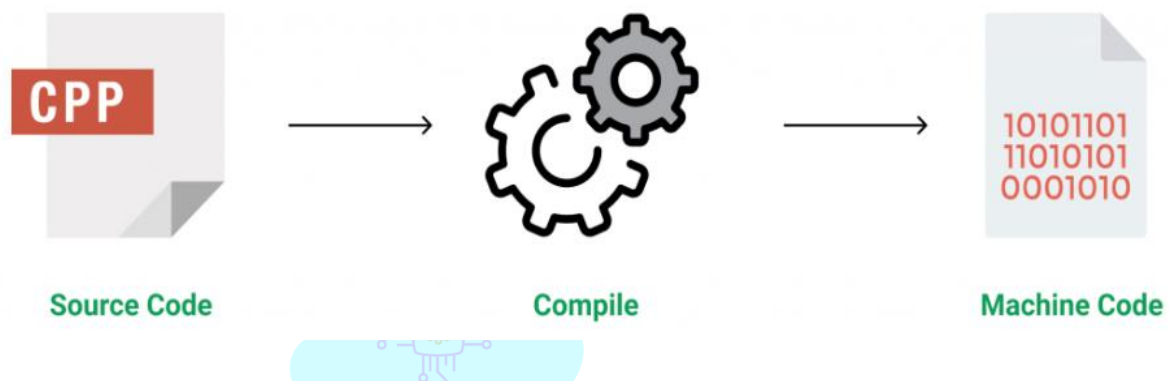


# Object Oriented Programming Language Using C++

## Introduction to C++:

C++ was developed by Bjarne Stroustrup in 1978 at Bell Labs, C++ is an extension to the C language. C++ is a general-purpose programming language. C++ introduces Object-Oriented Programming (OOP's), OOP's concepts not present in C. C++ supports the four primary features of OOP: encapsulation, polymorphism, abstraction, and inheritance.

In name C++, “++” is the C increment operator.



## Applications of C++:

C++ finds varied usage in applications such as:

- Operating Systems & Systems Programming. e.g. Linux-based OS (Ubuntu etc.)
- Browsers (Chrome & Firefox)
- Graphics & Game engines (Photoshop, Blender, Unreal-Engine)
- Database Engines (MySQL, MongoDB, Redis etc.)
- Cloud/Distributed Systems

There are many C++ compilers available which you can use to compile and run C++ program:

- Apple C++. Xcode
- Bloodshed Dev-C++
- Clang C++
- Cygwin (GNU C++)
- Mentor Graphics
- MINGW - "Minimalist GNU for Windows"
- GNU GCC source

- IBM C++
- Intel C++
- Microsoft Visual C++
- Oracle C++
- HP C++
- TURBO C++
- BORLAND C++

Popular IDE's include Code::Blocks, Eclipse, and Visual Studio. These are all free, and they can be used to both edit and debug C++ code. **Online Compiler** Web-based IDE's can work as well, but functionality is limited.

## Language:

Characters → Tokens (Identifiers/Constant/Reserved Words) → Instructions (Statements) → Program → Software.

## Features of OOP'S

- Class / Object
- Encapsulation
- Abstraction
- Data hiding
- Inheritance (Is-A Relationship)
- Polymorphism

## Class:

A class is an encapsulated entity which binds the properties and behaviour into a single unit.

Or

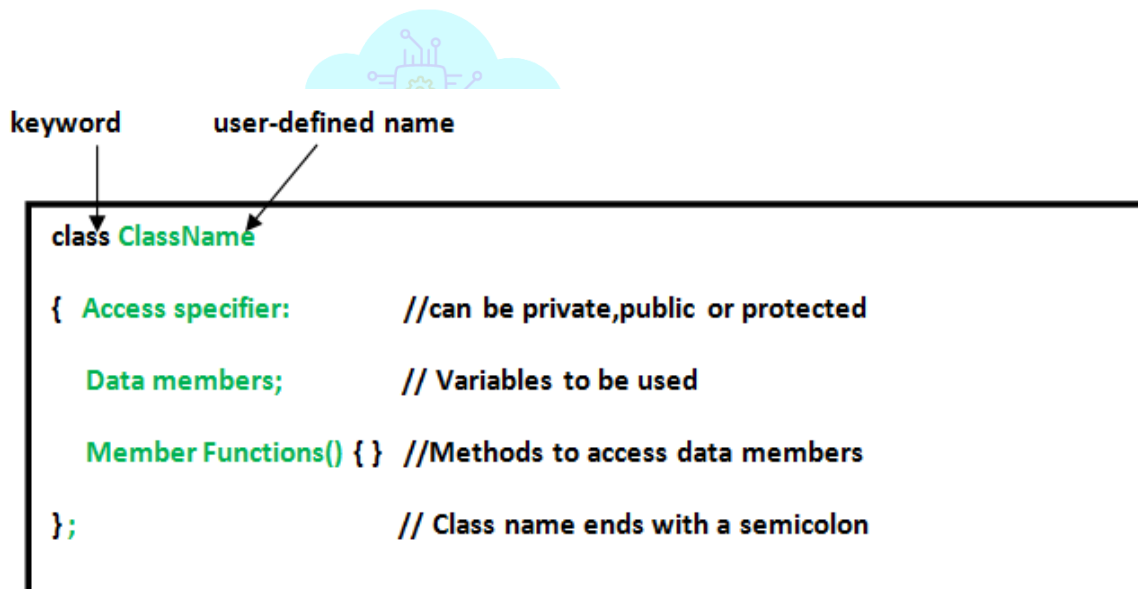
A class is an encapsulated entity which binds the data-member and member-function into a single unit is called class.

A class is user define data type. A C++ class is like a blueprint for an object. By using class keyword we can define a class.

A class is defined in C++ using keyword class followed by the name of class. The body of class is defined inside the curly brackets and terminated by a semicolon at the end.

### Syntax:

```
class <class-name>
{
    Property1 or data-member1 or variable1;
    Property2 or data-member2 or variable2;
    :
    :
    Behaviour1 or member-function1 or function1();
    Behaviour1 or member-function1 or function1();
};
```



### Example:

```
class Student
{
    public:
        int rollno;
```

```
float fees;

public:

void setDetail()

{

    cout<<"Enter the rollno and fees";

}

void getDetail()

{

    cout<<rollno<<fees;

}

};
```

## Object:

Object is a real world entity which takes some space in computer memory (heap memory). An **Object** is an instance of a Class. When a class is defined, no memory is allocated but when it is instantiated (i.e. an object is created) memory is allocated.

**Student s1,s2,s3;**

## How to Access Class members:

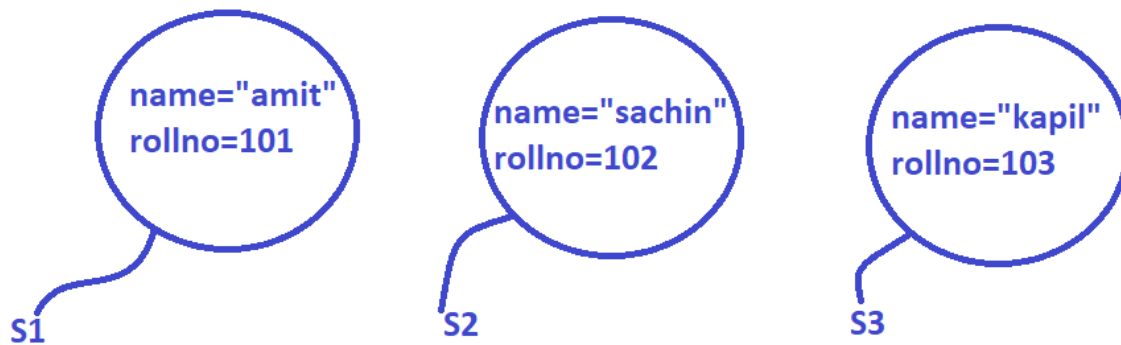
By using dot operator (.) and with the help of reference variable we can access class members.

```
s1.name="abc";

s1.rollno=101;

s1.getDetail();

s2.setDetail();
```



## Encapsulation:

Encapsulation is the process to binds the data member (variables) and member function (function or method) into a single unit is called encapsulation.

Class is an example of encapsulation in which we binds the variables and functions.

### Example:

```
class Student
```

```
{
```

```
    public:
```

```
        int rollno;
```

```
        float fees;
```

```
    public:
```

```
        void setDetail()
```

```
        {
```

```
            cout<<"Enter the rollno and fees";
```

```
        }
```

```
        void getDetail()
```

```
        {
```

```
            cout<<rollno<<fees;
```

```
        }
```

```
};
```

## Abstraction:

In Abstraction highlight the set of services and hide the internal implementation of an application is called abstraction.

### Example:

ATM Machine, in ATM Screen we show set of services (like withDraw, mini stm, bal query etc) and hide the internal implementation (How ATM machine work).

## Data hiding:

Hide the data from the outside side world and only authorized person can access that data. So the process of hiding data from outside world is known as data hiding.

### Example:

Email Account, only authorized Person (Which has user name and Password) can access your email.

So we can hide the data from outside world by declare variables as private.

### Note:

**Encapsulation = Abstraction + data Hiding.**

## Types of Variable:

There are three types of variable.

- 1) Instance variable or object level variable
- 2) Static variable or class level variable.
- 3) Local variable.

### 1) Instance variable or object level variable:

Instance variable are loaded into the memory when we create the object of that class.

For every object separate copy of instance variable will be created.

They are declared inside the class but outside the function, constructor or block without static keyword.

By default value of instance variable is Garbage.

```
class Employee
```

```
{
```

```
public:
```

```
int eid;
```

```
char ename[20];
```

```
float esal;
```

**Instance Variable**



```
public:
```

```
void setDetail()
```

```
{
```

```
cout<<"Enter the eid ename and esal";
```

```
cin>>eid>>ename>>esal;
```

```
}
```

```
void getDetail()
```

```
{
```

```
cout<<eid<<ename<<esal;
```

```
}
```

```
}
```

```
int main()
```

```
{
```

```
Employee e;
```

```
cout<<"Enter the id";
```

```
cin>>e.eid;
```

```
cout<<"Enter the Ename: ";
```

```
cin>>e.ename;
```

```
cout<<"Enter the salary":
```



```
cin>>e.esal;

cout<<e.eid<<" "<<e.ename<<" "<<e.esal;

}
```

**Note:** We can't access instance variable directly. They can access with reference variable and by using dot (.) operator.

### static variable:

- static variable are loaded into the memory at the time of class load. So they are loaded into the memory firstly.
- For every object a common copy will be created if any object changes the value of static variable then these changes will be reflect to every object.
- They are declared inside class and function, constructor or block with static keyword.
- By default value of static variable is zero.

### Static variables in a Function:

We can declare a static variable inside the function. If the function is called multiple times, space for the static variable is allocated only once and the value of variable in the previous call gets carried through the next function call.

```
#include <iostream.h>

#include <string.h>

using namespace std;

void demo()

{

    static int count;

    cout << count << " ";

    count++;

}

int main()
```

```
{  
    for (int i=0; i<5; i++)  
        demo();  
    return 0;  
}
```

**OUTPUT:** 0 1 2 3 4

### Static variables in a class:

We can declare a static variable inside the class but outside the function, constructor and block. Every object share the static variables if any object change the value of static variable than changes will be reflect to every object.

**Note:** we can access static variable by using reference variable, with class name and directly.

### Example:

```
#include <iostream>  
#include <string>  
using namespace std;  
class Test  
{  
    public:  
        static int i;  
};  
int Test::i = 1;  
int main()  
{  
    Test obj;  
    cout << obj.i;  
}
```

### 3) Local Variable:

Local variable are declared inside the function, constructor, block. For temporary requirement we declare local variable in a block. The Scope of Local variables within a block where they are declare. By default value of local variable is Garbage.

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

int multiply(int a, int b){
    return a * b;
}

int main() {
    int x = 3, y = 5;
    int z;
    z = multiply( x, y );
    cout << z << endl;
    return 0;
}
```

## Types of variables in C++

```
class GFG {
public:
    static int a; — Static Variable
    int b; — Instance Variable
public:
    func()
    {
        int c; — Local Variable
    };
};
```

## Constructor:

Constructor is a special function which names same as class name. Constructor is used to initialize an Object or initialize instance variable. Constructor calls automatically when we create the object of that class.

## Example:

```
class Test
{
    public:
        Test() // constructor
        {
            cout<<"Hello";
        }
};
```

## Example:

```
class Test
{
    public:
        int x,y;
        Test(int a,int b)
        {
            x=a;
            y=b;
        }
};
```

## Properties of Constructor:

1) Constructor must be declared in public section only.

- 2) Constructor must be executing when we create the object of that class.
- 3) Constructor name should be same as class name.
- 4) Constructor does not return any value even void.
- 5) Constructor can't be inherited.
- 6) Constructor can't be virtual.
- 7) Every class must contain at least one constructor.

**Note:**

Every class must contain at least one constructor. If Programmer not define any constructor inside the class then compiler will create a default constructor inside the class. If programmer define a constructor inside the class then compiler not create default constructor.

**Types of constructor:**

There are three types of constructor.

- 1) No-arg constructor or default constructor
- 2) Argument constructor or parameterized constructor.
- 3) Copy Constructor

**Note: Java support no-arg and parameterized constructor.**

**1) No-arg constructor or default constructor:**

Such constructor which does not accept any parameter is called no-arg constructor.

```
class Test
{
    public:
        Test()
        {
            cout<<"Welcome";
        }
}
```

```
};  
  
int main()  
{  
    Test t1,t2,t3;  
}
```

**Note: If programmer not defines any constructor inside the class the compiler will create no-argument constructor (default constructor).**

## 2) Parameterized constructor:

Such constructor which accepts a list of parameter is called parameterized constructor.

```
class Test  
{  
    public :  
        Test(int x, float y)  
        {  
            cout<<x+y;  
        }  
};
```

We can pass the initial value to the constructor in two ways.

### a) By calling the constructor explicitly.

```
Test t=Test (10,10.5);
```

### b) By calling the constructor implicitly.

```
Test t(10,10.5);
```

## 3) Copy Constructor:

Copy constructor is used to create a exactly same copy of an object.

Or

Copy Constructor is used to initialize an Object to another Object.

Or

Such constructor which accept the address of an Object.

```
class Test
{
    public:
        int x;
        int y;
    public:
        Test(int a, int b)
        {
            x=a;
            y=b;
        }
        Test(Test &t)
        {
            x=t.x;
            y=t.y;
        }
};

int main()
{
    Test t1(10,20);
    Test t2=t1 or
    Test t2(t1);
}
```

## Note:

In the above Program we can copy one object content to another Object with the help of copy constructor.

Java does not support copy constructor.

## Constructor Overloading:

When we declare more than one constructor in class with different signature. This concept is known as constructor overloading.

class Test

```
{  
    int a,b;  
    Test() // no-argument constructor  
    {  
        cout<<"Welcome"<<endl;  
    }  
    Test(int x,int y) // Parameterized constructor  
    {  
        a=x;  
        b=y;  
    }  
    Test(int a,int b,int c) // Parameterized constructor  
    {  
        cout<<a+b+c;  
    }  
};
```

## Constructor with default arguments:



In C++ we can pass one or more variable with default value in a constructor. Such constructor which accept default argument variable is called constructor with default argument.

**Example:**

```
class Test
{
    public:
        Test(int x, int y=10)
        {
            cout<<x+y;
        }
        Test(int x=10, int y, int z)
        {
            cout<<x+y+z;
        }
};
```

```
int main()
{
    Test t(5);
    Test t1(5,9);
}
```

**Dynamic Constructor:**

Dynamic constructor is used to allocate different-different size for each object according to requirement. For Dynamic memory allocation we new keyword.

If we use new keyword in a constructor then that constructor is called dynamic constructor.

**Example:**

```
char *name;
```

```
name=new char[size];
```

**Example:**

```
int *x;
```

```
x=new int[10];
```

```
class Test
```

```
{
```

```
    public:
```

```
        int x;
```

```
        char *name;
```

```
    Test(int l)
```

```
    {
```

```
        x=l;
```

```
        name =new char[x+1];
```

```
    }
```

```
};
```

```
int main()
```

```
{
```

```
    Test t1(5), t2(6),t3(10);
```

```
}
```

**Dynamic initialization of an Object:**

```
class Test
```

```
{
```

```
    public:
```

```
        int x;
```

```
float y;

Test(int a,float b)
{
    x=a;
    y=b;
}

};

int main()
{
    Test t1;
    int p;
    float q;
    cout<<"Enter the two variable";
    cin>>p>>q;
    t1=Test(p,q);

    cout<<t1.p+t1.q;
}
```

## **Destructor:**

A Destructor is a special function which name is same as class name but it will start with ~ tilde sign. As the name implies, it is used to destroy the object.

## **Properties of Destructor:**

- \* It will call automatically when a block is completed or program is terminated.
- \* Destructor never takes any argument or not accepts any parameter. Only one type of destructor is possible in c++. Which is used to destroy an object which is useless?
- \* Destructor is used to free the memory from the useless object.
- \* Destructor does not return any value even void.

- \* Destructor cannot be inherited.
- \* Destructor cannot be virtual.
- \* Destructor always declared in public section only.

```
class Test
{
    public:
        int x,int y;
        Test(int a,int b)
        {
            x=a;
            y=b;
        }
    public:
        ~Test() // Destructor
        {
            cout<<"Object is Destroyed";
        }
};
```

**Example:**

```
#include<conio.h>
#include<iostream.h>
class Test
{
    public:
        int x,y;
        Test(int a,int b)
```

```
{  
    cout<<"Object Created"<<endl;  
    x=a;  
    y=b;  
}  
  
public:  
    ~Test()  
    {  
        cout<<"Object is Destroyed"<<endl<<endl;  
    }  
};  
  
int main()  
{  
// clrscr();  
  
    Test t1(10,20),t2(100,200);  
    cout<<t1.x+t1.y<<endl;  
    getch();  
    return 0;  
}
```

**Output:**

Object Created

Object Created

30

Object is Destroyed

Object is Destroyed

**Example:**

```
#include<conio.h>

#include<iostream.h>

class Test
{
    public:
        static int count;

    public:
        Test()
        {
            count++;
            cout<<count<<" object is Created"<<endl;
        }
        ~Test()
        {
            cout<<count<<"Object is Destroy"<<endl;
            count--;
        }
};

int Test::count;

int main()
{
    clrscr();

    Test t1,t2,t3;
```

```
{  
    Test t4;  
}  
  
{  
    Test t5;  
}  
getch();  
return 0;  
}
```

### Output:

```
1 object is Created  
2 object is Created  
3 object is Created  
4 object is Created  
4 object is Destroy  
4 object is Created  
4 object is Destroy  
3 object is Destroy  
2 object is Destroy  
1 object is Destroy
```

### Inheritance or is-a relationship or Derivation:

Inheritance is the important concept of OOP's.

When Child Class extends or inherits Parent class Properties (data member) and behaviour

(member function) is called inheritance.

Inheritance is known as "is-a relationship".

Through inheritance Child class can access Parent class members(data variable & member function)

Note:

----

- 1) The main Advantage of inheritance is reusability of Code.
- 2) Through inheritance we can save memory space as well as time.

Which class is inherited is called Parent class or Super class or Base class.

Which inherits the Parent class is called Child class or Sub class or Drived class.

Syntax:

-----

```
class <Parent class>
```

```
{
```

```
    data member;
```

```
    +
```

```
    member function();
```

```
};
```

```
class <Child> : visibility mode <Parent class>
```

```
{
```

```
    data member;
```

```
    +
```

```
    member function();
```



```
};
```

Note:

-----

There are three types of visibility mode (public private protected) it is also known as access specifier.

Example:

-----

```
class Parent
```

```
{
```

```
    public:
```

```
        int x,y;
```

```
    public:
```

```
        void m1()
```

```
        {
```

```
        }
```

```
        void m2()
```

```
        {
```

```
        }
```

```
};
```

```
class Child : public Parent
```

```
{
```

```
    public :
```

```
        float a;
```

```
    public :
```

```
void m3()
{
}
};
```

Types of Inheritance:

-----

There are 5 types of inheritance in C++.

- 1) Single level inheritance.
- 2) Multilevel inheritance.
- 3) Hierarchical inheritance.
- 4) Multiple inheritance.
- 5) Hybrid inheritance.

Note: Java support only single, multilevel and hierarchical.

----

- 1) Single level inheritance:

-----

In inheritance only two classes are involved, one is Parent and another one is Child.

or Such a single child have only one Parent is called. single inheritance.

Example:

-----

```
class Parent
{
```

```
public:
    int x,y;
public:
    void m1()
    {
        cout<<x+y<<end;
    }
};
```

```
class Child : public Parent
```

```
{
public:
    int a;
public:
    void m2()
    {
        cout<<a;
    }
};
```

Example:

-----

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

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

```
class Parent
```

```
{
```

```
public:
    int x,y;
public:
    void m1()
    {
        cout<<x+y<<endl;
    }
};
class Child : public Parent
{
public:
    int a;
public:
    void m2()
    {
        cout<<a<<endl;
    }
    void m3()
    {
        cout<<a+x+y<<endl;
    }
};
int main()
{
    clrscr();
    Parent p;
```

Child c;

p.x=10;

p.y=20;

c.a=30;

c.x=100;

c.y=200;

p.m1();

c.m1();

c.m2();

c.m3();

getch();

return 0;

}

Output:

-----

30

300

30

330

2) Multilevel inheritance:

-----

In Multilevel inheritance Parent Class inherits GrandParent class, Child class inherits Parent class and soon. This type of inheritance is known as multilevel inheritance.

class GrandParent

```
{  
    public:  
        int x;  
    public:  
        void m1()  
        {  
            cout<<x;  
        }  
};  
class Parent : public GrandParent  
{  
    public:  
        int y;  
    public:  
        void m2()  
        {  
            cout<<x+y<<endl;  
        }  
};  
class Child : public Parent  
{  
    public :  
        int z;  
    public:  
        void m3()  
        {
```

```
    cout<<x+y+z;
}
};
```

Example:

-----

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

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

```
class GrandParent
```

```
{
```

```
    public:
```

```
        int x;
```

```
    public:
```

```
        void m1()
```

```
        {
```

```
            cout<<x<<endl;
```

```
        }
```

```
};
```

```
class Parent : public GrandParent
```

```
{
```

```
    public:
```

```
        int y;
```

```
    public:
```

```
        void m2()
```

```
        {
```

```
            cout<<x+y<<endl;
```

```
    }  
};  
class Child : public Parent  
{  
    public :  
        int z;  
    public:  
        void m3()  
        {  
            cout<<x+y+z<<endl;  
        }  
};  
int main()  
{  
    clrscr();  
    GrandParent gp;  
    Parent p;  
    Child c;  
    gp.x=10;  
  
    p.x=100;  
    p.y=200;  
  
    c.x=1000;  
    c.y=2000;  
    c.z=3000;
```



```
gp.m1();  
p.m1();  
p.m2();  
c.m1();  
c.m2();  
c.m3();  
getch();  
return 0;  
  
}
```

Output:

-----

10  
100  
300  
1000  
3000  
6000



3) Hierarchical inheritance:

-----

When One Parent have multiple childs or when multiple childs inherits a single parent is called

Hierarchical inheritance.

Example:

-----

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

```
#include<iostream.h>

class Parent
{
    public:
        int x,y;
    public:
        void m1()
        {
            cout<<x+y<<endl;
        }
};

class Child1 : public Parent
{
    public:
        int a;
    public:
        void m2()
        {
            cout<<a+x+y<<endl;
        }
};

class Child2 : public Parent
{
    public:
        int b;
    public:
```

```
void m3()
{
    cout<<b+x+y<<endl;
}
};
```

```
int main()
```

```
{
```

```
    Parent p;
```

```
    Child1 c1;
```

```
    Child2 c2;
```

```
    clrscr();
```

```
    p.x=100;
```

```
    p.y=200;
```

```
    c1.x=10;
```

```
    c1.y=20;
```

```
    c1.a=30;
```

```
    c2.x=1000;
```

```
    c2.y=2000;
```

```
    c2.b=3000;
```

```
    p.m1();
```

```
    c1.m2();
```

```
    c2.m3();
```

```
getch();  
  
return o;  
  
}
```

output:

-----

300

60

6000

Example:

-----

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

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

```
class Parent
```

```
{
```

```
    public:
```

```
        int x,y;
```

```
        static int z;
```

```
    public:
```

```
        void m1()
```

```
        {
```

```
            cout<<x+y+z<<endl;
```

```
        }
```

```
};
```

```
class Child1 : public Parent
```

```
{
```

```
public:
    int a;
public:
    void m2()
    {
        cout<<a+x+y+z<<endl;
    }
};
class Child2 : public Parent
{
public:
    int b;
public:
    void m3()
    {
        cout<<b+x+y+z<<endl;
    }
};
int Parent::z;
int main()
{
    Parent p;
    Child1 c1;
    Child2 c2;
    clrscr();
```

```
p.x=100;
```

```
p.y=200;
```

```
p.z=1;
```

```
c1.x=10;
```

```
c1.y=20;
```

```
c1.a=30;
```

```
c1.z=2;
```

```
c2.x=1000;
```

```
c2.y=2000;
```

```
c2.b=3000;
```

```
    c2.z=3;
```

```
p.m1();
```

```
c1.m2();
```

```
c2.m3();
```

```
cout<<p.z+c1.z+c2.z;
```

```
getch();
```

```
return 0;
```

```
}
```

```
output:
```

```
-----
```

```
303
```

```
63
```

6003

9

#### 4) Multiple Inheritance:

-----

In multiple inheritance one child inherits multiple parents.

or

when Multiple parents have only one child, such type of inheritance is called multiple inheritance.

Syntax:

-----

```
class <Child Name> : visibility mode <Parent1 Name>, Visibility Mode <Parent2 Name>,.....  
{  
    data member and member functions.  
};
```

Example:

-----

```
#include<conio.h>  
#include<iostream.h>  
class Parent1  
{  
    public:  
        int x;  
    public:  
        void m1()
```

```
{
    cout<<x<<endl;
}

};

class Parent2
{
    public:
        int y;
    public:
        void m2()
        {
            cout<<y<<endl;
        }
};

class Child:public Parent1, public Parent2
{
    public:
        int z;
    public:
        void m3()
        {
            cout<<x+y+z<<endl;
        }
};

int main()
{
```



```
clrscr();
```

```
Parent1 p1;
```

```
Parent2 p2;
```

```
Child c;
```

```
p1.x=100;
```

```
p2.y=200;
```

```
c.x=1000;
```

```
c.y=2000;
```

```
c.z=3000;
```

```
p1.m1();
```

```
p2.m2();
```

```
c.m3();
```

```
getch();
```

```
return 0;
```

```
}
```

output:

-----

100

200

6000

Example:

-----

```
#include<conio.h>

#include<iostream.h>

class Parent1
{
    public :
        int a;
        static int b;

    public:
        void m1()
        {
            cout<<a+b<<endl;
        }
};

class Parent2
{
    public:
        int x;
        static int y;

    public:
        void m2()
        {
            cout<<x+y<<endl;
        }
};

class Child:public Parent1,public Parent2
```

```
{  
    public:  
        int p;  
    public:  
        void m3()  
        {  
            cout<<x+y+a+b+p;  
        }  
};  
int Parent1::b;  
int Parent2::y;  
int main()  
{  
    clrscr();  
    Parent1 p1;  
    Parent2 p2;  
    Child c;  
  
    p1.a=10;  
    p1.b=20;  
  
    p2.x=100;  
    p2.y=200;  
  
    c.a=1000;  
    c.b=2000;
```

```
c.x=3000;
```

```
c.y=4000;
```

```
c.p=5000;
```

```
p1.m1();
```

```
p2.m2();
```

```
c.m3() ;
```

```
getch();
```

```
return 0;
```

```
}
```

Output:

```
-----
```

```
2010
```

```
4100
```

```
15000
```



5) Hybrid inheritance:

```
-----
```

Such inheritance in which two or more inheritance are involved are called hybrid inheritance.

when two or more inheritance are involved in a single program such program is called

Hybrid inheritance program.

Example:

```
-----
```

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

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

```
class A
{
    public:
        int x;
};

class B:public A
{
    public:
        int y;
};

class C
{
    public:
        int z;
};

class D: public B,public C
{
    public:
        int p;

    public:
        void m1()
        {
            cout<<x+y+z+p;
        }
};

int main()
```

```
{  
    clrscr();  
    D d;  
    d.x=1;  
    d.y=2;  
    d.z=3;  
    d.p=4;  
  
    d.m1();  
    getch();  
    return 0;
```

```
};
```

output:

-----

10

-----

Ambiguity problem in multiple inheritance:

-----

When two or parent contain same signature function than child class inherits both classes,

So it contain two copy of same function with same signature. when we call that function with child class reference variable then compiler will generate an error which is ambiguity problem.

it mean compiler is in confusion which function will be call.

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

```
#include<iostream.h>

class Parent1
{
    public:
        void m1()
        {
            cout<<"Parent1 m1 function"<<endl;
        }

};

class Parent2
{
    public:
        void m1()
        {
            cout<<"Parent2 m1 function"<<endl;
        }

};

class Child:public Parent1,public Parent2
{
    public:
        void m2()
        {
            cout<<"child class function";
        }

};
```

```
int main()
{
    Child c;
    clrscr();
    // c.m1();
    c.m2();
    getch();
    return 0;
}
```

In the above program Parent1 and Parent2 contain m1() with same signature.

When child class try to access m1() then it will generate ambiguity problem.

We can solve ambiguity problem in multiple inheritance by using Scope resolution operator.

We can define which Parent class m1() will be call inside the child class m1();

```
class Child:public Parent1,public Parent2
{
    public:
        void m1()
        {
            Parent1::m1(); //Here Parent1 class m1() will be call
        }
};
```



Program:

-----

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

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

```
class Parent1
```

```
{
```

```
    public:
```

```
        void m1()
```

```
        {
```

```
            cout<<"Parent1 m1 function"<<endl;
```

```
        }
```

```
};
```

```
class Parent2
```

```
{
```

```
    public:
```

```
        void m1()
```

```
        {
```

```
            cout<<"Parent2 m1 function"<<endl;
```

```
        }
```

```
};
```

```
class Child:public Parent1,public Parent2
```

```
{
```

```
    public:
```

```
void m1()
{
    Parent1::m1();
    Parent2::m1();
}
};
```

```
int main()
{
    Child c;
    clrscr();
    c.m1();
    getch();
    return 0;
}
```



-----  
Ambiguity Problem in hybrid inheritance:

-----  
It can be explain with the help of following Program.

```
#include<conio.h>
#include<iostream.h>
class GrandParent
{
    public:
        void m1()
```

```
{  
    cout<<"Hello"<<endl;  
}  
};  
class Parent1: public GrandParent  
{  
    public:  
        void m2()  
        {  
            cout<<"m2"<<endl;  
        }  
};  
class Parent2:public GrandParent  
{  
    public:  
        void m3()  
        {  
            cout<<"m3"<<endl;  
        }  
};  
};  
class Child:public Parent1,public Parent2  
{  
    public:
```

```
void m4()
{
    cout<<"m4"<<endl;
}

};

int main()
{
    clrscr();
    Child c;
    c.m1();
    getch();
    return 0;
}
```

In the above program Child class contain two copy of m1() from the path Parent1 and Parent2.

When we try to access m1() with the help of Child class reference variable then compiler will generate an error( ambiguity problem).

So we can solve the above problem with the help of virtual base class.

At the time of inheritance we inherits the GrandParent as virtual.

So due to this only single copy of GrandParent class function will be delivered to the Child class directly.

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

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

```
class GrandParent
```

```
{  
  
    public:  
  
        void m1()  
  
        {  
  
            cout<<"Hello"<<endl;  
  
        }  
  
};  
  
class Parent1: public virtual GrandParent  
{  
  
    public:  
  
        void m2()  
  
        {  
  
            cout<<"m2"<<endl;  
  
        }  
  
};  
  
class Parent2:public virtual GrandParent  
{  
  
    public:  
  
        void m3()  
  
        {  
  
            cout<<"m3"<<endl;  
  
        }  
  
};
```

```
class Child:public Parent1,public Parent2
```

```
{
```

```
    public:
```

```
        void m4()
```

```
        {
```

```
            cout<<"m4"<<endl;
```

```
        }
```

```
};
```

```
int main()
```

```
{
```

```
    clrscr();
```

```
    Child c;
```

```
    c.m1();
```

```
    getch();
```

```
    return 0;
```

```
}
```

-----  
Access specifier:

-----

In C++ there are three types of access specifier.

- 1) public
- 2) protected
- 3) private

1) public:

-----  
inside the class    Outside the inherit class    outside the non inherit class

-----  
YES

YES

YES

Example:

-----  
#include<conio.h>

#include<iostream.h>

class A

{

  public:

    int x;

  public:

    void m1()

    {

      cout<<x<<endl;

    }

};

class B: public A

{

  public:

    void m2()

    {

      cout<<x<<endl;

      m1();

    }

```
};  
  
int main()  
{  
    clrscr();  
    A a1;  
    B b1;  
    b1.x=1000;  
    a1.x=100;  
    cout<<a1.x;  
    b1.m2();  
    getch();  
    return 0;  
}
```

output:

```
-----  
100  
1000  
1000
```

Example:

```
-----
```

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

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

```
class A
```

```
{
```



```
public:
    int x;
public:
    void m1()
    {
        cout<<x<<endl;
    }
};
class B
{
    public:
        void m2()
        {
            A a1;
            a1.x=10;
            cout<<a1.x<<endl;
            a1.m1();
        }
};
int main()
{
    clrscr();
    A a1;
    B b1;
    a1.x=100;
    cout<<a1.x<<endl;
```

```
b1.m2();
```

```
getch();
```

```
return 0;
```

```
}
```

output:

-----

100

10

10

Note:

----

Public member (data member and member function) of a class can be access from any where.

we can access within a class where they are declared, outside the inherited class, and outside the non

inherit class(but with that class reference variable).

Protected:

-----

We can access protected member of a class,within that class where they are declared and

inside the Child class but cannot access outside the non inherits class.

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

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

```
class Parent
```

```
{
```

protected:

```
int x;
```

```
int y;
```

protected:

```
void m1()
```

```
{
```

```
    x=10;
```

```
    y=20;
```

```
    cout<<x+y<<endl;
```

```
}
```

```
};
```

```
class Child:public Parent
```

```
{
```

```
public:
```

```
void m2()
```

```
{
```

```
    x=100;
```

```
    y=200;
```

```
    cout<<x+y;
```

```
    m1();
```

```
}
```

```
};
```

```
int main()
```

```
{
```

```
    Child c;
```

```
    clrscr();
```

```
c.m2();  
getch();  
return 0;  
}
```

output:

-----

300

30

private:

-----

private member of class can be access only within a class outside that class we can not access private member of that class.

Question: How to access private member outside that class where they are declared?

-----

Ans: we can access private member outside the class by using public function.

----

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

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

```
class Parent
```

```
{
```

```
    private:
```

```
        int x;
```

```
    public:
```

```
        void m1()
```

```
{  
    x=100;  
    cout<<x;  
}  
};
```

```
int main()  
{  
    Parent p;  
    clrscr();  
    cout<<p.x;  
    p.m1();  
    getch();  
    return 0;  
}
```



Note:

-----

It is recommended to declare variable as private and function as public.

By declaring variable as private we can achieve data hiding.

Inherits Parent class as public, private and protected:

-----

```
class Child : public Parent
```

-----

```
class Parent
```

```
{  
    public:  
        int x;  
    protected:  
        int y;  
    private:  
        int z;  
    public:  
        void m1()  
        {  
        }  
    protected:  
        void m2()  
        {  
        }  
    private:  
        void m3()  
        {  
        }  
};
```

```
class Child : public Parent
```

```
{  
    public:  
        int a,b;  
    protected:
```

```
int c;

private:

    int d;

public:

    void m11(){}

protected:

    void m12(){}

private:

    void m13(){}

};
```

If we inherits Parent class as Public then public member of parent class will goto in public section of Child class,protected member will goto in protected section of child class but private member can not be inherited.

```
class Child :protected Parent
```

-----

```
class Parent
{
    public:

        int x;

    protected:

        int y;

    private:
```

```
    int z;

public:
    void m1()
    {
        }

protected:
    void m2()
    {
        }

private:
    void m3()
    {
        }
};

class Child : protected Parent
{
public:
    int a,b;

protected:
    int c;

private:
    int d;

public:
    void m11(){ }

protected:
```



```
void m12(){  
  
private:  
  
void m13(){  
  
};
```

If we inherits Parent class as Protected, than public and protected member of Parent class will goto in protected section of Child class.

```
class Child : private Parent:  
  
-----
```

```
class Parent  
{  
public:  
    int x;  
protected:  
    int y;  
private:  
    int z;  
public:  
    void m1()  
    {  
    }  
protected:  
    void m2()  
    {
```

```
    }  
private:  
    void m3()  
    {  
    }  
};  
  
class Child : private Parent  
{  
    public:  
        int a,b;  
    protected:  
        int c;  
    private:  
        int d;  
    public:  
        void m11(){  
    protected:  
        void m12(){  
    private:  
        void m13(){  
};
```

If we inherits Parent class as private then public and protected member of Parent class will goto in private section of child class.

Note:

-----

private member of a Parent can not be inherited even if Parent inherits as public, private or protected.

Abstract class:

-----

Such a class in which at least one function which does not have implementation(without body) such class is known as abstract class.

class Shape

```
{  
    public:  
        void area()=0;  
}
```



Polymorphism:

-----

The word Polymorphism means having many forms.

The word Polymorphism is derived from two words.

Poly+ Morphism

Poly means many

&

Morphism means forms

Example:

-----

A Person in class room is act as a Student,

Same Person in Family act as Brother or Sister.

Same Person in Sports act as a Sportsman.

Here Same Person have multiple form in different different situation.

This is called Polymorphism.

Polymorphism is an important feature of OOP'S.

In C++ Polymorphism is divided into two Types:

-----

1) Comiple Time / static / Early Binding

2) Run time / Dynamic / Late Binding

1) Comiple Time / static / Early Binding:

-----

If we know at compile time which function will be execute at run time is called

compile time polymorphism.

Example:

-----

a) Function Overloading

b) Operator Overloading

a) Function Overloading:

-----

In function overloading, a class contain more than one same name but different Signature function

this concept is known as function overloading.

Note:

----


A class does not contain same signature function. if we try to declare the we will get compile time error.

Which m1() function will be execute it will be deside at compile time.

so it is known as compile time or static or early bind.

Example:

-----



```
#include<conio.h>
#include<iostream.h>
class Test
{
public:
    int x;
public:
    void m1()
    {
        cout<<x<<endl;
    }
    void m1(int a)
    {
```

```
    cout<<x+a<<endl;
}
void m1(int a,int b)
{
    cout<<a+b+x<<endl;
}
};
int main()
{
    Test t;
    clrscr();
    t.x=10;
    t.m1();
    t.m1(100);
    t.m1(100,200);
    getch();
    return 0;
}
```

## b) Operator Overloading:

-----

In operator overloading we can assign some special meaning to an operator is known as operator overloading.

Example:

-----

We known that plus + operator add two number but in c++ we can assign some extra work to

plus operator. Due to extra work plus operator is overload.

Syntax:

-----

return type class-name:: operator op-sign(list of parameter)

{

    // body

}

Rules for operator overloading:

-----

1) we can not overload class member access operator (. , .\*)

Scope resolution operator (::)

sizeof operator

Conditional operator (?:)

2) Only C++ existing operators can be overloaded. new operator can not be overloaded.

3) We cannot change the basic meaning of an operator.

4) We can not use following operators with friend keyword (=,(),[],->)

Overloading Unary Operator:

-----

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

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

```
class Test
```

```
{
```

```
    public:
```

```
int x,y,z;

public:

    void getData()

    {

        cout<<x<<" "<<y<<" "<<z;

    }

    void operator -()

    {

        x=-x;

        y=-y;

        z=-z;

        cout<<"Done";

    }

};

int main()

{

    Test t;

    t.x=10;

    t.y=-20;

    t.z=30;

    -t;

    t.getData();

    getch();

    return 0;

}
```



```
}
```

output:

-----

Done -10 20 -30

Example:

-----

```
class Test
```

```
{
```

```
    public:
```

```
        int x,y;
```

```
    public:
```

```
        void operator ++();
```

```
};
```

```
void Test::operator ++()
```

```
{
```

```
    x=x+5;
```

```
    y=y+10;
```

```
}
```

```
int main()
```

```
{
```

```
    Test t;
```

```
    t.x=10;
```

```
    t.y=20;
```

```
    t++;
```

```
++t;  
  
getch();  
  
return 0;  
  
}
```

Overloading Binary Operator:

```
-----  
  
#include<conio.h>  
  
#include<iostream.h>  
  
class Complex  
{  
public:  
    float x;  
    float y;  
    Complex()  
    {  
    }  
    Complex(float r,float i)  
    {  
        x=r;  
        y=i;  
    }  
    Complex operator +(Complex c4)  
    {  
  
        c4.x=x+c4.x;
```

```
        c4.y=y+c4.y;

        return c4;
    }

    void display()
    {
        cout<<x<<"+i"<<y<<endl;
    }

};

int main()
{
    Complex c1(2.5,3.5),c2(1.6,1.7),c3;
    clrscr();
    c3=c1+c2;
    c1.display();
    c2.display();

    cout<<"-----"<<endl;

    c3.display();

    getch();

}
```

output:

-----

2.5+i3.5

1.6+i1.7

-----  
4.1+i5.2

Assignment:  
-----

WAP to find the product of two complex number.

WAP to add the two time(5:20+4:10=9:30).

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

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

```
class Test
```

```
{
```

```
public:
```

```
int x,y;
```

```
Test operator -(Test);
```

```
};
```

```
Test Test:: operator -(Test t)
```

```
{
```

```
t.x=x+t.x;
```

```
t.y=y-t.y;
```

```
return t;
```

```
}
```

```
int main()
```

```
{  
    Test t1,t2,t3;  
    clrscr();  
    t1.x=10,t1.y=20;  
    t2.x=1,t2.y=2;  
  
    t3=t1-t2;  
    cout<<t3.x+t3.y;  
    getch();  
    return 0;  
}
```

Pointer: Pointer is a variable which hold the address of another variable.

-----

Syntax:

```
<pointer-Type> * <pointe-name>;
```

E.g:

```
int *p;
```

```
int x=10;
```

```
p=&x;
```

Note: We can access value of x directly or by using pointer.

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

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

```
int main()
```

```
{  
    int x=10;  
  
    int *p;  
  
    clrscr();  
  
  
    p=&x;  
  
  
  
    cout<<x<<endl;  
    cout<<*p<<endl;  
    cout<<p;  
    getch();  
    return 0;  
}
```



Note: We can increment/ decrement in pointer.

-----

```
#include<conio.h>  
  
#include<iostream.h>  
  
int main()  
{  
    int x=10;  
  
    int *p1,**p2;;  
  
    clrscr();  
  
  
    p1=&x;
```

```
p2=&p1;

cout<<"p1 address="<<p1<<endl;
cout<<"p2 address="<<p2<<endl;

p1=p1+2;
cout<<"p1 address after increment="<<p1<<endl;

p2=p2+2;
cout<<"p2 address after increment="<<p2;

getch();
return 0;
}
#include<conio.h>
#include<iostream.h>
int main()
{
    int x=10;
    int *p1,**p2,***p3;
    clrscr();

    p1=&x;
    p2=&p1;
    p3=&p2;
```

```
cout<<"p1 hold address="<<p1<<endl;
cout<<"p2 hold address="<<p2<<endl;
cout<<"p3 hold address="<<p3<<endl;

p1=p1+2;
cout<<"p1 address after increment="<<p1<<endl;

p2=p2+2;
cout<<"p2 address after increment="<<p2<<endl;

p3=p3+1;
cout<<"p3 address after increment="<<p3<<endl;
getch();
return 0;
}
```

Pointers with Array:

```
-----
#include<conio.h>
#include<iostream.h>
int main()
{
    int x[50],*p,n,i,sum=0;
    cout<<"Enter the no of elements in array"<<endl;
```



```
cin>>n;

for(i=0;i<n;i++)
{
    cin>>x[i];
}

p=x;

for(i=0;i<n;i++)
{
    if(*p%2==0)
        sum=sum + *p;
    p++;
}

cout<<sum<<endl;

getch();

return 0;
}
```

Array of Pointer:

-----

Example:

-----

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

```
#include<iostream.h>

int main()
{
    int *p[5],a=10,b=20,c=30,d=40,e=50,**p1;

    clrscr();

    p[0]=&a;
    p[1]=&b;
    p[2]=&c;
    p[3]=&d;
    p[4]=&e;

    p1=p;

    cout<<**p1;

    p1++;

    p1++;

    cout<<**p1;

    getch();

    return 0;

}
```

Example:

```
-----  
#include<conio.h>  
  
#include<iostream.h>  
  
int main()  
{  
    int *p[3],a[3]={10,20,30},b[4]={1,2,3,4},c[2]={6,9},**p1;  
  
    clrscr();  
  
    p[0]=a;  
    p[1]=b;  
    p[2]=c;  
  
    p1=p;  
  
    p1++;  
    (*p1)++;  
    (*p1)++;  
    **p1=10;  
  
    cout<<*((*p1)++)<<endl;  
  
    cout<<**p1;  
  
    getch();  
  
    return 0;  
  
}
```

Pointers to Objects:

```
-----  
#include<conio.h>  
  
#include<iostream.h>  
  
class Test  
{  
    public:  
        int x,y;  
    public:  
        void m1()  
        {  
            cout<<x+y<<endl;  
        }  
};  
  
int main()  
{  
    Test t;  
    Test *p;  
    clrscr();  
    p=&t;  
  
    p->x=10;  
    p->y=20;  
  
    t.x=100;  
    t.y=200;
```

```
p->m1();
```

```
    getch();
```

```
}
```

Note:

-----

A class Type Pointer can access class member by using arrow operator or Object Pointer.

There are following ways to access class members:

```
1)p->x=10;
```

```
    p->y=20;
```

```
2)(*p).x=1000;
```

```
3)t.x=100;
```

```
    t.y=200;
```

We can assign an address of object in two ways:

```
1)Test t;
```

```
    Test *p;
```

```
    p=&t;
```

```
2) Test *p=new Test;
```

Note: new keyword is used to assign memory to an object.

Example:

-----

Array of Objects:

-----

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

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

```
class Test
```

```
{
```

```
public:
```

```
int x,y;
```

```
public:
```

```
void m1()
```

```
{
```

```
cout<<x+y<<endl;
```

```
}
```

```
};
```

```
int main()
```

```
{
```

```
Test *p=new Test[5];
```

```
int i;
```

```
clrscr();
```

```
Test *p1=p;
```

```
for(i=0;i<5;i++)
{
    cout<<"enter the value of x and y";
    cin>>p1->x;
    cin>>p1->y;
    p1++;
}
p1=p;
for(i=0;i<5;i++)
{
    p1->m1();
    p1++;
}
getch();
}
```

output:

=====

Enter the value of x and y 10 20

Enter the value of x and y 20 30

Enter the value of x and y 30 40

Enter the value of x and y 40 50

Enter the value of x and y 50 60

30

50

70

90

110

WAP, you have 10 Items(T-shirt, Lower, Book, TV etc) which have following field

Item id, Item name, item price, congiguration.

To print the Item detail by Item id.

this pointer:

-----

"this" is a pointer, which is used inside the class only outside the class we cannot use this pointer.

this pointer is used to represent current class members(data member & member function).

We not need to create this pointer it is automatically created by compiler.

Example:

-----

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

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

```
class Test
```

```
{
```

```
    public:
```

```
        int x;
```

```
    public:
```



```
void m1()
{
    this->x=100;
    cout<<x<<endl;
}

};

int main()
{
    Test t;
    clrscr();
    // this->x=1000; // this can be used within member function(class)
    t.m1();
    getch();
    return 0;
}
```

Example:

-----

```
#include<conio.h>
#include<iostream.h>
class Test
{
public:
    int x;
public:
    Test(int x)
```

```
{
    this->x=x;
}

void m1()
{
    cout<<x<<endl;
}

};

int main()
{
    Test t(100);
    clrscr();
    // this->x=1000;
    t.m1();
    getch();
    return 0;
}
```

Note:

-----

this pointer is used to hide local variable, if both instance and local variable have same name.

Example:

-----

We can return this pointer also , which is explain in the below program

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

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

```
class Test
{
    public:
        int x,y;
    public:
        Test & m1()
        {
            this->x=10;
            this->y=20;
            return *this;
        }
};
int main()
{
    Test t,p;
    clrscr();
    // this->x=1000;
    p=t.m1();
    cout<<p.x+p.y<<endl;
    getch();
    return 0;
}
```

---

Function Overriding:

When Child class is not satisfy with Parent class member function

then child can change parent class member function internal implementation.

This concept is known as function Overriding.

For overriding inheritance is compulsory.

Parent class member function signature must be matched with Child class member function.

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

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

```
class Parent
```

```
{
```

```
public:
```

```
void property()
```

```
{
```

```
cout<<"Cash Gold Silver Power"<<endl;
```

```
}
```

```
void marry()
```

```
{
```

```
cout<<"Aarti"<<endl;
```

```
}
```

```
};
```

```
class Child : public Parent
```

```
{
```

```
public:
```

```
void marry()
```

```
{
```

```
cout<<"Pooja Cash Diamond Gold Power"<<endl;
}

};

int main()
{
    Parent p;
    Child c;

    p.property();
    p.marry();

    c.property();
    c.marry();
    getch();
}
```

Note: which marry function will be execute it we will be deside at runtime. So it is called Runtime polymorphism.

In the above program Child class marry() override the Parent class marry() at runtime(When Program is executed.).

Virtual function:

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

```
#include<iostream.h>

class Parent
{
    public:
        void property()
        {
            cout<<"Cash Gold Silver Power"<<endl;
        }
        void marry()
        {
            cout<<"Aarti"<<endl;
        }
};

class Child : public Parent
{
    public:
        void marry()
        {
            cout<<"Pooja Cash Diamond Gold Power"<<endl;
        }
};

int main()
{
    // Parent p;
    // Child *c1;
```

```
Parent *p;
```

```
Child c1;
```

```
clrscr();
```

```
// c1=&p; invalid
```

```
p=&c1;
```

```
p->property();
```

```
p->marry();
```

```
c1.property();
```

```
c1.marry();
```

```
getch();
```

```
}
```

Note:

-----

Parent class Pointer can hold Child class object address. but Child class Pointer can not hold Parent class Object address.

Example:

-----

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

```
#include<iostream.h>

class Parent
{
    public:
        void property()
        {
            cout<<"Cash Gold Silver Power"<<endl;
        }
        void virtual marry()
        {
            cout<<"Aarti"<<endl;
        }
};

class Child : public Parent
{
    public:
        void marry()
        {
            cout<<"Pooja Cash Diamond Gold Power"<<endl;
        }
};

int main()
{
    Parent *p;
    Child c1;
```



```
clrscr();  
  
p=&c1;  
  
p->property();  
p->marry();  
  
c1.property();  
c1.marry();  
  
getch();  
  
}
```

Note:

-----

If you want to access Child class function (marry()) with Parent class pointer (hold the address of

Child class) then you must be declared Parent class Function(marry()) as virtual.

Then that Child class marry() will be executed. It will be decided at Runtime which marry() will be executed. So virtual is an example of Runtime polymorphism.

-----

Constructor in Derived classes:

-----

Question : How to call Parent class Constructor without creating Parent class Object.

```
#include<conio.h>

#include<iostream.h>

class Parent1
{
public:
    int x;

public:
    Parent1(int a)
    {
        x=a;
        cout<<x<<endl;
    }
};

class Parent2
{
public:
    int y;

public:
    Parent2(int b)
    {
        y=b;
        cout<<y<<endl;
    }
};

class Child:public Parent2,public Parent1
```

```
{  
    public:  
        int z;  
    public:  
        Child(int a,int b,int c):Parent1(a), Parent2(b)  
        {  
            z=c;  
            cout<<z<<endl;  
        }  
};  
int main()  
{  
    clrscr();  
    Child c(10,20,30);  
    getch();  
}
```

Note: First Constructor of Parent class will be execute which inherits Child class firstly.

----

Example:

-----

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

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

```
class Parent1
```

```
{
```

```
public:
    Parent1()
    {
        cout<<"Parent1 cons"<<endl;
    }
};

class Parent2
{
public:
    Parent2()
    {
        cout<<"Parent2 cons"<<endl;
    }
};

class Child:public Parent1, public Parent2
{
public:
    Child()
    {
        cout<<"Child cons"<<endl;
    }
};

int main()
{
    clrscr();
```

```
Child c;  
    getch();  
}
```

Note:

-----

If we inherits Parent class as virtual then virtual Parent class constructor will be execute firstly, and sequence of inheritance by child class does not metter.

Question : Explain virtual keyword in detail?

-----

Managing I/O Operations:

-----

Stream:

-----

Flow of bits from input to program and Program to output device is known as bit stream.

Input Stream:

-----

Flow of bits/bytes from input device to Program by using extraction operation is known as input stream.

Output Stream:

-----

Flow of bits/bytes from Program to output device by using insertion operation is known as output stream.

C++ Stream classes:

-----

ios classes:

-----

This is the Parent class of all stream classes.

This class contains some basic facilities that are used in input/output operation.

istream class:

-----

istream class is the child of ios class.

It contain some basic input function such as:

get(), getline() and read().

It contain overloaded extraction operator >>

ostream:

-----

ostream class is the child of ios class.

It contain some basic output function such as:

put() write().

It contain overloaded insertion operator << .

iostream:

-----

This class inherits the istream and ostream classes.

It inherits all functionality of istream and ostream class.

and insertion and extraction operator also.

streambuf:

-----

It is the child of ios class.

It works between physical device(input/output device) and Program which store the no of bits/bytes.

Note:

cin and cout are object of input and output stream classes.

cin >> x;

">>" is an overloaded extraction operator which is define inside the istream class.

out << "Hello";

"<<" is an overloaded insertion operator which is define inside the ostream class.

get() and put() :

-----

get():

-----

get() is present inside the istream class. which handle the single character input operation.

By using cin object of itream class we can access get().

We can explain with the help of following Program.

```
#include<iostream.h>

#include<conio.h>

int main()

{

    char c;

    clrscr();

    cin.get(c); // cin is an object of istream class and we try to access get() od istream class by
using cin object.

    cout<<c;

    getch();

};
```

Example:

-----

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

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

```
int main()
```

```
{
```

```
    char c;
```

```
    clrscr();
```

```
    cin.get(c);
```

```
    while(c!='\t')
```

```
    {
```



```
cout<<"Print: "<<c<<endl;

cin.get(c);

}

getch();

};
```

Note:

-----

So we have two ways to get the character from the keyboard.

a) `cin>>c;` by using overloaded extractor operator we can get single char from the keyboard

b) `cin.get(c);` by using `get()` of `istream` class object.

`put()`:

-----

`put()` is present inside the `ostream` class.

By using `put()` we can print the single character on the screen.

By using `cout` object of `ostream` class, we can access `put()`.

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

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

```
int main()
```

```
{
```

```
    char c;
```

```
    clrscr();
```

```
cin.get(c);
```

```
while(c!='\t')
```

```
{
```

```
    cout.put(c);
```

```
    cin.get(c);
```

```
}
```

```
getch();
```

```
};
```

Note:

----

So we have two ways to display a single character on the screen.

```
cout<<c;
```

```
cout.put(c);
```

```
getline() and write():
```

-----

a) getline():

-----

-->getline() is present in istream class/iostream.

-->getline() read the whole line including white space from the keyboard.

while cin object read only single word if any white space occurs then it will stop read

the characters.

Program Without geline():

-----

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

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

```
int main()
```

```
{
```

```
    char name[20];
```

```
    clrscr();
```

```
    cout<<"Enter the name:";
```

```
    cin>>name;
```

```
    cout<<name;
```

```
    getch();
```

```
}
```

output:

-----

Enter the name: Mr amit kumar

Mr

Program with getline():

-----

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

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

```
int main()
```

```
{  
    char name[20];  
    clrscr();  
    cout<<"Enter the name:";  
    cin.getline(name,10); //cin>>name;  
    cout<<name;  
  
    getch();  
}
```

output:

-----

Enter the name: Mr amit kumar

Mr amit kumar

write():

-----

write() is used to display whole line at a time, on the console(output screen)

write() is present inside the ostream/iostream.

write() accept two parameters, one is array of char type and second parameter is size which you want to print.

e.g:

```
cout.write(name,10);
```

Program:

-----

```
#include <iostream>
```

```
int main()
```

```
{
```

```
// clrscr();
```

```
char name[20];
```

```
std::cout<<"enter the name = ";
```

```
std::cin.getline(name,20);
```

```
std::cout.write(name,10);
```

```
return 0;
```

```
}
```

-----

File Handling:

-----

File:

-----

File is a storage area where we can store data permanently.

File is store in secondary memory. Secondary memory store data permanently.

Where as main memory store data temporary.

File Handling:

-----

We can perform lots of operation on a file such as open a file, close a file, insert data into the file, update data to the file, delete data from the file and fatch the data from the file.

So we need to handle such operation on a file, need a programming language.

By using C++ we can handle a file.

File Stream classes:

-----

1) fstreambase:

-----

It provides some basic function which are applied to a file such as open() and close() etc.

2) ifstream:

-----

It provide such basic functions such as open(), close(), get(), getline(), read() seekg() tellg() etc.

3) ofstream:

-----  
It provides functions such as open() close() put() tellp() seekp() write().

Open a file:

-----

To load a file from harddrive to main memory(RAM) is known as  
open a file.

there are two ways to open a file.

- 1) Open a file by using open():
- 2) Open a file by using constructor of fstream/istream/ofstream.

Note: If you want to perform read/write operation on a file you must be include fstream.h

----- header file.

- 
- 1) Open a file by using open():
- 

syntax:

-----

```
open("File Path",mode);
```

E.g:

-----

```
open("D:/abc/first.txt",ios::out);
```

-->First parameter of open function specify the address(Location) of opened file.

-->Second parameter of open() specify the mode in which you want to open.

there are following modes are possible in c++.

ios::app:

-----

ios::app denotes to open a file in append mode it means new data will be add to the old data into the file.

If file is not exist in the HardDrive then append mode will create a new file.

```
#include<fstream.h>
```

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

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

```
int main()
```

```
{
```

```
    fstream f;
```

```
    clrscr();
```

```
    f.open("aryan.txt",ios::app);
```

```
    f<<"C++ Programmer";
```

```
    f.close();
```

```
    getch();
```

```
}
```

ios::out:

-----

ios::out denotes to write some data into the file, if old data is available inside the file then it will replace old data with new data into the file.



If file is not exist in harddrive then it will create a new file.

```
#include<fstream.h>

#include<conio.h>

#include<iostream.h>

int main()

{

    fstream f;

    clrscr();

    f.open("aryan123.txt",ios::out);

    f<<"Java Program";

    f.close();

    getch();

}
```

ios::in :

-----

This mode is used to read some data from the file.

if file is not exist then compiler will not create a file.

```
#include<fstream.h>

#include<conio.h>
```

```
#include<iostream.h>

int main()
{
    fstream f;
    int x;
    clrscr();
    f.open("aryan123.txt",ios::in);

    f>>x;
    cout<<x;
    f.close();
    getch();
}

ios::ate:
```



-----  
Open the file and moves the control to the end of the file.

ios::trunc : This mode will remove all data in the existing file.

-----

ios::nocreate:

-----

This mode open the file if file is already exists.

If file does not exist then it will not create a new file.

ios::noreplace:

-----

This mode open the file only if it does not already exist.

ios::binary:

-----

open the file in binary mode.

Note:

-----

We can apply more then one mode on a same file by using parallel operator(|)

Program 1:

-----

```
#include<fstream.h>
```

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

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

```
int main()
```

```
{
```

```
    fstream inf;
```

```
    int x,y;
```

```
    clrscr();
```

```
    inf.open("sun.txt",ios::out);
```

```
    cout<<"Enter the value: ";
```

```
    cin>>x;
```

```
inf<<x*5;
inf.close();

fstream outf;
outf.open("sun.txt",ios::in);
outf>>y;

cout<<"New Value is: "<<y*3;

outf.close();

getch();
}
```

WAP to open a file both in read/write mode:

```
-----
#include<fstream.h>
#include<iostream.h>
#include<conio.h>
int main()
{
    fstream inf;
    int x,y;
    clrscr();
    inf.open("sun.txt",ios::out | ios::in);
```

```
cout<<"Enter the value: ";  
  
cin>>x;  
  
inf<<x*5<<endl;  
  
inf.seekg(0);  
  
inf>>y;  
  
cout<<"New Value is: "<<y*3;  
  
inf.close();  
  
getch();  
}
```



Assignment:

-----

Copy one file content to another file.

2) Open a file by using constructor of fstream/istream/ofstream.:

-----

We can open a file by using ifstream or ofstream class constructor.

if we open a file by using ifstream class constructor then we can perform read operation from the file.

if we open a file by using ofstream class constructor then we can perform write operation

into the file.

Program 1:

-----

```
#include<fstream.h>
#include<iostream.h>
#include<conio.h>
int main()
{
    ofstream outf("shopitem.txt");
    char name[20],resname[20];
    int price,resprice;

    clrscr();
    cout<<"enter the item name";
    cin>>name;

    outf<<name;
    outf<<" ";
    cout<<"enter the Price:";
    cin>>price;

    outf<<price;

    outf.close();
```

```
ifstream inf("shopitem.txt");

inf>>resname;

inf>>resprice;

cout<<resname<<"\t"<<resprice;

getch();

}
```

---

Functions for manipulation of File pointers.

---

There are following functions which are used to check and change pointer position into the file.

seekg():

-----

It moves and get pointer(input) to a specified location.

```
#include<fstream.h>
```

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

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

```
int main()
```

```
{
```

```
    ofstream outf("shopitem.txt");
```

```
char name[20],resname[20];

int price,resprice;

clrscr();

cout<<"enter the item name";

cin>>name;

outf<<name;

outf<<" ";

cout<<"enter the Price:";

cin>>price;

outf<<price;

outf.close();

ifstream inf("shopitem.txt");

inf.seekg(4);

inf>>resprice;

cout<<"LCD Price is : "<<resprice;

getch();

}
```



output:

-----

enter the item nameLCD

enter the Price:2300

LCD Price is : 2300

seekp():

-----

It moves and get the pointer(write) to specified location.

```
#include<fstream.h>
```

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

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

```
int main()
```

```
{
```

```
  ofstream outf("shopitem.txt");
```

```
  char name[20],resname[20];
```

```
  int price,resprice;
```

```
  clrscr();
```

```
  cout<<"enter the item name";
```

```
  cin>>name;
```

```
  outf<<name;
```

```
  outf<<" ";
```

```
  cout<<"enter the Price:";
```

```
  outf.seekp(1);
```

```
cin>>price;
```

```
outf<<price;
```

```
outf.close();
```

```
ifstream inf("shopitem.txt");
```

```
inf>>resprice;
```

```
cout<<"LCD Price is : "<<resprice;
```

```
getch();
```

```
}
```

output:

-----

enter the item nameLCD

enter the Price:2300

LCD Price is : 0

Note: Inside the file content will be "L2300".

```
tellg():
```

-----

This function is used to give the current position of the pointer inside file. When file is opened in read mode.

```
#include<conio.h>

#include<iostream.h>

#include<fstream.h>

int main()

{

    int p;

    char msg[20];

    clrscr();

    ifstream inf("abcd.txt");

    inf.seekg(4);

    p=inf.tellg();

    cout<<p<<endl;

    inf>>msg;

    cout<<msg<<endl;

    p=inf.tellg();

    cout<<p;

    getch();

}
```

tellp():

-----

This function is used to give the current position of the pointer inside file. When file is opened in write mode.

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

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

```
#include<fstream.h>
```

```
int main()
```

```
{
```

```
int p;
```

```
char msg[20];
```

```
clrscr();
```

```
ofstream ofs("abcd.txt");
```

```
ofs<<"Welcome C++";
```

```
p=ofs.tellp();
```

```
cout<<p;
```

```
ofs.seekp(4);
```

```
ofs<<"done";
```

```
cout<<p;
```

```
getch();
```

```
}
```

put() and get() function:

-----

put():

-----

put() is used to put a character into the file.

We can write only one character at a time.

Syntax:

-----

```
file.put(char);
```

get():

-----

get() is used to get a single character at a time from the file.

we can read only single character at a time.

Program:

-----

```
#include<conio.h>

#include<iostream.h>

#include<fstream.h>

#include<string.h>

int main()

{

    char name[20];

    int i,l;

    char ch;

    clrscr();

    cout<<"Enter the name";

    cin>>name;

    l=strlen(name);

    fstream f;

    f.open("getputfile.txt",ios::in | ios::out);

    for(i=0;i<l;i++)

    {

        f.put(name[i]);

    }

    f.seekg(0);

    while(f)

    {
```

```
f.get(ch);  
cout<<ch;  
}  
getch();  
  
};
```

write() and read() function:

-----

write():

-----

This function is used to write data in binary format and store data in binary format.

This function is available in ofstream class.

```
outfile.write((char *) &v,sizeof(v));
```

read():

-----

This function is used to read data from the file which is stored in binary format.

This function is available in ifstream class.

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

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

```
#include<fstream.h>
```

```
int main()
```

```
{
```

```
int v=10,u;

clrscr();

ofstream of;

of.open("az.txt");

of.write((char *)&v,sizeof(v));

of.close();

ifstream ifs;

ifs.open("az.txt");

ifs.read((char *)&u,sizeof(u));

cout<<u;

getch();

}
```

How to save object into the file and read object from the file.

```
-----

#include<iostream.h>

#include<conio.h>

#include<fstream.h>

class Test

{

public:

    int x,y;
```



```
public:
    void display()
    {
        cout<<x<<endl;
        cout<<y<<endl;
    }
};

int main()
{
    Test t,t1;
    ofstream outfile;
    outfile.open("myfile.txt",ios::out);
    t.x=100;
    t.y=200;

    outfile.write((char *)&t,sizeof(t));
    outfile.close();

    ifstream infile;
    infile.open("myfile.txt",ios::in);

    infile.read((char *)&t1,sizeof(t1));

    t1.display();

    infile.close();
```

```
getch();
```

```
}
```

--> By using write() we can save an object into the file.

The Process of saving object into the file is called serialization.

-->By using read() we can read object from the file.

The process of read object from the file is called deserialization.

-->Object are always stored into the file in binary format.

Error handling Function:

-----

eof():

-----

eof() return true if end-of-file is encountered while reading,  
otherwise it return false.

fail():

-----

fail() return true if input or output operation has failed.

good():

-----

it return true if no error in the file read and write operation.

bad():

-----

it return true if an invalid operation attempted.

-----  
Exception Handling:

-----  
Exception:

-----  
An Exception is an unwanted unaspected event which disturb the normal flow of a program is called exception, Due to an exception your program will terminate abnormally. So we need to protect our program for abnormal termination.

Note:

-----  
int main()  
{  
    int x=10,y=0;  
    cout<<x/y;  
    getch();  
}

In the above program at line `cout<<x/y;` program will terminate abnormally.

Exception Handling:

-----  
To protect program for abnormal termination, we will use try-catch block.

Inside the try block we put the risky code or such lines of code which may genrate exception.

If any exception is arise in try block the it will throw by using throw keyword and will catch in catch block.

Syntax:

-----

```
try
{
    // risky code
}
catch(data-type)
{
    //Handling code
}
```

Example:

-----

```
// Online C++ compiler to run C++ program online
```

```
#include <iostream>
```

```
int main() {
    int x=10,y=0;
```

```
try
{
    if(y==0)
    {
        throw y;
    }
    else
    {
```

```
        std::cout<<x/y;
    }

}

catch(int a)
{
    std::cout<<"Dont divide by zero";
}

return 0;
}

// Online C++ compiler to run C++ program online
#include <iostream>

int main() {
    int x,y;
    std::cout<<"Enter first number";
    std::cin>>x;
    std::cout<<"Enter second number";
    std::cin>>y;
    try
    {
        if(y==0)
        {
            throw y;
        }
    }
}
```

```
}  
else  
{  
    std::cout<<x/y;  
}  
  
}  
catch(int a)  
{  
    std::cout<<"Dont divide by zero";  
}  
  
return 0;  
}
```



try with multiple catch block:

-----

There may be a chance of try block contains more than one exception than that case we will use multi catch blocks to handle each exception.

Syntax:

-----

```
try  
{  
    //risky code
```

```
}  
catch(type1)  
{  
}  
catch(type2)  
{  
}  
:  
:  
:  
catch(typeN)  
{  
}
```

Program:

-----  
// Online C++ compiler to run C++ program online

```
#include <iostream>
```

```
int main() {
```

```
    test(10);
```

```
    return 0;
```

```
}
```

```
void test(int x)
```

```
{
```

```
try
{
    if(x==0)
    {
        throw x;
    }
    else if(x==1)
    {
        throw 'a';
    }
    else if(x==-1)
    {
        throw 10.5;
    }
    else
    {
        std::cout<<x+x;
    }
}
catch(int p)
{
    std::cout<<"Dont take value is zero";
}
catch(char q)
{
    std::cout<<"Dont take value is one";
```



```
}  
catch(double r)  
{  
    std::cout<<"Dont take value is -1";  
}  
}
```

Program:

-----

// Online C++ compiler to run C++ program online

```
#include <iostream>
```

```
int main() {
```

```
    test(10);
```

```
    return 0;
```

```
}
```

```
void test(int x)
```

```
{
```

```
    try
```

```
    {
```

```
        if(x==0)
```

```
        {
```

```
            throw x;
```

```
        }
```

```
    else if(x==1)
```

```
{
    throw 'a';
}
else if(x===-1)
{
    throw 10.5;
}
else
{
    std::cout<<x+x;
}
}
catch(int p)
{
    std::cout<<"Dont take value is zero";
}
catch(char q)
{
    std::cout<<"Dont take value is one";
}
catch(double r)
{
    std::cout<<"Dont take value is -1";
}
}
```

Output:

-----  
Dont take value is one

Dont take value is zero

20

Note:

-----  
If exception handling there is possible to handle more than one exception  
into single catch block. This can be done with catch(...).

Syntax:

-----  
try

{

}

catch(...)

{

}

Program:

-----  
// Online C++ compiler to run C++ program online

#include <iostream>

void test(int x)

{

```
try
{
    if(x==0)
    {
        throw x;
    }
    else if(x==1)
    {
        throw 'a';
    }
    else if(x==-1)
    {
        throw 10.5;
    }
    else
    {
        std::cout<<x+x;
    }
}
catch(...)
{
    std::cout<<"Please Check your Code";
}

}

int main() {
```

```
test(1);  
test(0);  
test(10);  
return 0;  
}
```

output:

-----

Please Check your Code

Please Check your Code

20

Note:

-----

After catch(...), not takes any catch block because catch(...) is last handler block.

// Online C++ compiler to run C++ program online

```
#include <iostream>
```

```
void test(int x)
```

```
{
```

```
    try
```

```
    {
```

```
        if(x==0)
```

```
        {
```

```
            throw x;
```

```
}  
else if(x==1)  
{  
    throw 'a';  
}  
else if(x==-1)  
{  
    throw 10.5;  
}  
else  
{  
    std::cout<<x+x;  
}  
}  
catch(...)  
{  
    std::cout<<"Please Check your Code";  
}  
catch(int x)  
{  
    std::cout<<"Wrong int";  
}  
  
}  
int main() {
```

```
test(1);  
test(0);  
test(10);  
return 0;  
}
```

output:

-----

error: '...' handler must be the last handler for its try block

But we can put any number of catch block before catch(...).

// Online C++ compiler to run C++ program online

```
#include <iostream>
```

```
void test(int x)
```

```
{
```

```
try
```

```
{
```

```
if(x==0)
```

```
{
```

```
throw x;
```

```
}
```

```
else if(x==1)
```

```
{
```

```
throw 'a';
```

```
}
```

```
else if(x==-1)
```

```
{  
    throw 10.5;  
}  
else  
{  
    std::cout<<x+x;  
}  
}  
catch(int x)  
{  
    std::cout<<"Wrong int";  
}  
catch(...)  
{  
    std::cout<<"Please Check your Code";  
}  
  
}  
  
int main() {  
  
    test(1);  
    test(0);  
    test(10);  
    return 0;  
}
```



output:

-----

Please Check your Code

Wrong int

20

Specify Exception:

-----

It is possible to restrict a function to throw only certain specified exception.

This can be achieved by adding throw list in a function.

Syntax:

-----

```
function(arg-list) throw (throw-list)
```

```
{
```

```
// function body
```

```
}
```

Program:

-----

```
// Online C++ compiler to run C++ program online
```

```
#include <iostream>
```

```
void test(int x) throw(int,double)
```

```
{
```

```
if(x==0)
{
    throw 'a';
}
else if(x==1)
{
    throw x;
}
else if(x==-1)
{
    throw 1.5;
}
else
{
    std::cout<<x*x;
}
}
```

```
int main() {

    try
    {
        test(0);
    }

    catch(char ch)
```

```
{  
    std::cout<<"char catch block";  
}  
catch(int z)  
{  
    std::cout<<"int catch block";  
}  
catch(double d)  
{  
    std::cout<<"double catch block";  
}  
  
return 0;  
}  
output(Online Compiler):
```

-----  
terminate called after throwing an instance of 'char'

Aborted

Rethrowing an Exception:

-----  
It may be possible handler code(in catch block) can rethrow an exception.

In such situations, we can write only throw without argument in catch block.

Program:

-----

```
// Online C++ compiler to run C++ program online
```

```
#include <iostream>
```

```
void divide(double x,double y)
```

```
{
```

```
try
```

```
{
```

```
if(y==0.0)
```

```
{
```

```
throw y;
```

```
}
```

```
else
```

```
{
```

```
std::cout<<x/y;
```

```
}
```

```
}
```

```
catch(double z)
```

```
{
```

```
std::cout<<"divide function try catch";
```

```
throw;
```

```
}
```

```
}
```

```
int main() {
```

```
try
```

```
{
```

```
// divide(10.5,2.0);
```

```
    divide(20.0,0.0);  
}  
catch(double)  
{  
    std::cout<<"main try catch block";  
}  
return 0;  
}
```

---

## String

---

String:

-----

A string is a sequence of character or set of characters.

We can create a string in c++ in two ways.

1) by using array of characters which always terminated by null character ('\0').

E.g:

----

```
char name[20];
```

2) by creating string class object we can create a string.

E.g:

----

```
string s("abcd");
```

Note:

-----

string concept is come in ANSI C++.

We can perform following operation on a string.

- a) Creating a String and display a string.
- b) Modify the existing string.
- c) comparing two strings.
- d) Finding a substring from a string.
- e) Add two string
- f) Find the size of string.

string class contain three constructors:

A) `string();` it will create empty string

B) `string(char *str)` It will create a string which terminated by null character('\0').

C) `string(string &s)` It will create a string from another string.

string class contains some functions:

`append()` `append()` is used to appends a part of string to another.

`Assign()` Assigns a partial string.

`at()` `at()` is used to obtain a character at particular location.

`begin()` It return the address of string first character location.

`capacity()` `capacity()` is used to gives the total size.

compare() compare() is used to compare two string

empty() it return true if string have some characters otherwise it will return false.

end() It return the address of string's last character location.

erase() It removes the characters from the string.

find() searches for the occurrence of a specified substring.

insert() inserts characters at a specified location.

length() Gives the number of elements in a string.

replace() Replace the characters with new string.

swap() it will swap to substring to each other.

Operators for string class object:

= assignment

+ concate

+= concate assignment

== equality

!= not equality

< Less than

> greater than

<= Less than equal to

>= Greater than equal to.

etc..

Create string class object:

-----

i) string s1; It will create empty string.

ii) string s2("india");

iii) s1=s2;

iv) cin>>s1; Read only one word from the keyboard

v) getline(cin,s1) Read the whole line from keyboard with space.

Manipulate String Object:

-----

We can modify String class object by using insert(), replace(), append() erase() etc.

```
#include <iostream>
```

```
#include<string>
```

```
int main() {
```

```
    std::string s1("12345");
```

```
    std::string s2("abcde");
```

```
    s1.insert(4,s2) ;
```

```
    std::cout<<s1<<"\n";
```

```
    return 0;
```

```
}
```

output:

-----

12345

abcde

1234abcde5



```
-----  
#include <iostream>  
  
#include<string>  
  
int main() {  
    std::string s1("12345");  
    std::string s2("abcde");  
  
    s1.insert(4,s2) ;  
    std::cout<<s1<<"\n";  
  
    s1.erase(4,5);  
    std::cout<<s1<<"\n";  
    return 0;  
}
```

output:

-----  
1234abcde5

12345  
-----

```
#include <iostream>  
  
#include<string>  
  
int main() {  
    std::string s1("12345");  
    std::string s2("abcde");
```

```
s1.insert(4,s2) ;  
std::cout<<s1<<"\n";  
  
s1.erase(4,5);  
std::cout<<s1<<"\n";  
  
s2.replace(1,2,s1);  
std::cout<<s2;  
return 0;  
}
```

output:

-----

1234abcde5

12345

a12345de

-----

// Online C++ compiler to run C++ program online

```
#include <iostream>
```

```
#include<string>
```

```
int main() {
```

```
    std::string s1("ABC");
```

```
    std::string s2("XYZ");
```

```
    std::string s3=s1+s2;
```

```
if(s1!=s2)
{
    std::cout<<"s1 is not equal to s2"<<"\n";
}
if(s3==s1+s2)
{
    std::cout<<"s3 is equal to s1+s2"<<"\n";
}
int x=s1.compare(s2);
if(x==0)
    std::cout<<"s1==s2";
else if(x>0)
    std::cout<<"s1>s2";
else
    std::cout<<"s1<s2";

}
```

output:

-----

s1<s2

-----

// Online C++ compiler to run C++ program online

#include <iostream>

```
#include<string>

int main() {

    std::string s1("APPLY");

    std::string s2("APPLY");

    std::string s3=s1+s2;

    if(s1!=s2)

    {

        std::cout<<"s1 is not equal to s2"<<"\n";

    }

    if(s3==s1+s2)

    {

        std::cout<<"s3 is equal to s1+s2"<<"\n";

    }

    int x=s1.compare(s2);

    if(x==0)

        std::cout<<"s1==s2";

    else if(x>0)

        std::cout<<"s1>s2";

    else

        std::cout<<"s1<s2";

}
```

Output:

-----

s3 is equal to s1+s2

s1==s2

```
-----  
  
#include <iostream>  
  
#include<string>  
  
int main() {  
    std::string s1;  
  
    std::cout<<s1.size()<<"\n";  
    std::cout<<s1.length()<<"\n";  
    std::cout<<s1.capacity()<<"\n";  
    std::cout<<s1.max_size()<<"\n";  
    std::cout<<s1.empty()<<"\n";  
}
```

output:

```
-----  
0  
0  
15  
4611686018427387903  
1
```

-----  
WAP to find the number of character in a file.

WAP to replace word "is" to "are" in a given file.

Accessing Char in String:

-----  
at(),substr(),find(),find\_first\_of(), find\_last\_of().

```
#include <iostream>
```

```
#include<string>
```

```
int main() {
```

```
    std::string s("ONE TWO THREE");
```

```
    for(int i=0;i<s.length();i++)
```

```
    {
```

```
        std::cout<<s.at(i)<<"\n";
```

```
    }
```

```
    return 0;
```

```
}
```

output:

O

N

E

T

W

O

T

H

R

E

E

```
#include <iostream>

#include<string>

int main() {

    std::string s("ONE TWO THREE");

    int x = s.find("HREE");

    std::cout<<x<<"\n";

    std::cout<<s.substr(1,6)<<"\n";

    int x1 = s.find_first_of('T');
    int x2 = s.find_last_of('T');

    std::cout<<x1<<"\n";

    std::cout<<x2;

    return 0;

}
```

output:

-----

9

NE TWO

4

8

-----  
friend function:

-----  
We know that we can not access private variables outside the class,

but by using friend function we can access private member outside the class.

By using friend keyword we can declare a function as firendly.

A friend function is not a member of any class. But it can access all members of a class.

Properties of friend function:

-----  
->It is not in the scope of a class whatever it is declare inside the class.

->It can not be called by using the object of that class.

->It will be call like a normal function.

->It can be declare either in public or private or protected section in a class.

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

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

```
class Sample
```

```
{
```

```
    int a,b;
```

```
    public:
```

```
        void setValue()
```

```
    {
```



```
cout<<"enter the value of a and b: ";

cin>>a>>b;

}

friend int avg(Sample s);

};

int avg(Sample s)

{

    return (s.a+s.b)/2;

}

int main()

{

    Sample s1;

    clrscr();

    s1.setValue();

    cout<<avg(s1);

    getch();

}
```

output:

-----

enter the value of a and b: 10 20

15

A friend function of two classes:

-----

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

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

```
class Army;

class Manager
{
    int x;

    public :
    void setValue(int i)
    {
        x=i;
    }

    friend void max(Manager,Army);
};
```

```
class Army
{
    int y;
    public:
    void setValue(int i)
    {
        y=i;
    }

    friend void max(Manager,Army);
};
```

```
void max(Manager m,Army a)
{
    if(m.x>=a.y)
    {
```

```
    cout<<m.x;
}
else
{
    cout<<a.y;
}
}
int main()
{
    Manager mag;
    Army ar;
    clrscr();
    mag.setValue(12);
    ar.setValue(14);

    max(mag,ar);
    getch();
}
```

output:

-----

14

//WAP to find the simple interest of Manager and Army Person by using freind function.

Inline function:

-----

An inline function is a function that is expanded in line, When we call a function then compiler replaces the function call to the function body.

By using inline keyword we can declare a function as inline.

Inline function makes a program run faster.

syntax

-----

```
inline function-name()
```

```
{
```

```
    //function body
```

```
}
```

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

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

```
inline float multi(float x,float y)
```

```
{
```

```
    return x*y;
```

```
}
```

```
inline float divide(float x, float y)
```

```
{
```

```
    return x/y;
```

```
}
```

```
int main()
```

```
{
```

```
    float a=12.5;
```

```
    float b=2.5;
```

```
cout<<multi(a,b)<<endl;  
cout<<divide(a,b)<<endl;  
getch();  
}
```

Note: In the above program multi() function call will be replace with multi function code.

In the above program divide() function call will be replace with divide function code.

The End