DEVANSH GOYAL

C++ File

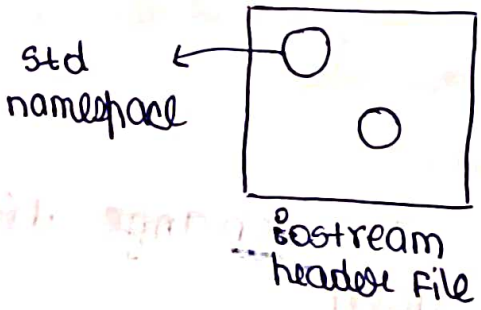
TARUN SIR

C++ ".cpp"

- C++ is an object oriented based programming language.
- Bjarne Stroustrup created this C++ in 1979. in same AT and TBell Lab in USA.
- C was based on POP thats why cpp created.
- C++ initial name was C with classes.
- C++ is an extention of C.
- File saved with .cpp extention

Header Files of C++ and namespace

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



This namespace contains → cin & cout functions

(There can be other namespaces)

In C++ functions are not stored directly in the headerfile we use namespaces.

- input and output →

  Syntax → cin >>
  cout <<

- Run commands →

  g++ Filename.cpp → compile
  a.out → Run

- Hello world →

```
#Inculde <iostream>
using namespace std;
int main()
{
  cout << "hello world!";
  return 0;
}
```

- Endl →

  is mainpulator who can change line and \n also can be used.

  Syntax → cout << endl;

WAP to add two numbers → (static)

```
# include <iostream>
using namespace std;
int main()
{  int a=10, b=20, c;
   c = a+b;
   cout << "sum of a,b = " << c << endl;
   return 0;
}
```

WAP to add 2 no's → (Dynamic)

```
# include <stdio.h>
using namespace std;
int main()
{  int a, b, c;
   cout << "Enter value of a, b";
   cin >> a >> b;
   c = a+b;
   cout << "sum of a, b = " << c << endl;
}
```

WAP to add 2 no's dynamic without using namespace →

```
# include <iostream>
// using namespace std;  (don't write)
{  int a, b, c;
   std :: cout << " Enter ";
   std :: cin << a >> b;
   c = a+b;
   std :: cout << " Sum a,b = " << c << std:: endl
   return 0;
}
```

std :: → scope Resolution operator

WAP to swap noumbers in cpp

```
# include <stdio.h>
# include < iostream>
using namespace std;
int main ()
{  int a,b,c;
   cout << " Enter " << endl;
   cin << a << b;
   c = a;
   a = b;
   b = c;
   cout << nos are << a << b;
}  return 0;
```

OOPs →

object oriented Programming

Encapsulation →

Binding the 'data members' (int a, b, c) and 'member functions' (eg → void add ()) in single unit is called data encapsulation.

wrapping things together is also called encapsulation.

In cpp we use CLASSES to encapsulate the data that means classes are example of encapsulation.

- Class — (user defined data type)

class is a container who provides some facility to store data members and member functions together.

class provides security to data member and member functions from access outside the class using Access Modifiers

- Access Modifiers →

In C++ there are 3 types of Access modifiers which are -

- Private
- Public
- Protected

- Public - If data member or member func. declared as public then they can be accessed anywhere in the algorithm.

- Protected - If we decleare Data member or member func. as Protected that mean it can be accessible in same class or child class and 'can't be accessed in the main function'

- Private - If we decleare Data member or member function as Prive it can't be accessede outside class. It always accessed with Public method of same class.

In cpp, by 'default class have private' access modifier.

Syntax of class Declaration –

```
# include < Postream >
Using namespace std;
class A          // like structure
{   Public :
    Void show ( )                    //methods
      {                                  by default
        cout << " hello" << endl;        private
      }
};

int main ( )
{
    A   ob;     // object → have power to access
} ob . show ;                the data of class
    return 0;
}
```

ob is object and ob is object name it may be ab, abc etc.

" ER class के अंदर कितने भी object हो सकते है "

* In class static declaration is not allowed

eg → int a=10, b=10, c;

prog. to add 2 nos access in cpp →

```
#include <stdio.h>
#include <iostream>
using namespace std;
class A
{
    int a, b, c;  // data member
    public:
    void set()    // member functions
    {
        cout << "Enter value of a, b";
        cin >> a >> b;
    }
    void show()
    {
        cout << "value of a = " << a << endl;
        cout << "value of b = " << b << endl;
    }
};
```

```
int main()
{
    A ob;
    ob.set();
    ob.show();
    return 0;
}
```

WAP to swap 2 no's without third var

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

int main()
{  int a = 5, b = 10;

    a = a+b;
    a = a-b;
    b = a-b;

    printf("a = %d, b = %d", a, b);

    return 0;

}
```

WAP to create student

```
#include <iostream>
using namespace std;
class student.
  {
      string name;
      int age;
      int rollno;
      Public:
        void set()
        { cout << "Enter name";
          cin >> name;

          cout << "Enter age" << endl;
          cin >> age;

          cout << "Enter roll" << endl;
          cin>> rollno;
        }

      void show()
{ cout << "name \t age \t Rodl " << endl;
  cout << name << "/t" << age "\t" << roll
                                  << "\t";
  }
};
```

```
int main ()
{  student ob;
   ob. set ();
   Ob. show ();
   return 0;
}
```

WAP to calculate simple intrest →

```
#include <iostream>
using namespace std;
class si
{  float p, r, t, si;
   Public:
   void set ()
   {  cout << "Enter value of p, r, t";
      cin >> p >> r >> t;
   }
   Void cal ()
   {  si = (p * r * t) / 100;
   }
  void show ()
   {  cout << "\n simple intrest = " << si;
   }
};
```

```
int main ()
{ Si obj;
   obj. set;
   obj. cal;
   Obj. show;
}
```

** How to use private –

```
class A.
{  int a, b, c;
   private :
        void set ()
          {
            cout << "Enter nos";
            cin >> a >> b;
            c = a + b;
          }
   public :
        void show ()
          {  set ();
             cout << "sum" << c;
          }
   };
int main ()
{ A obj;
  obj. show ();
```

" private can be accessed using public of the same class "

WAP to find max using encapsulation

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

class max
{   private :
        int a , b ;
        void set ()
          { cout << " Enter value of a,b ";
            cin >> a >> b;
          }

    Public :
        void cal()
          {  set ();
             if (a> b)
                 {
                   cout<<" a is max " <<endl;
                 }
             else
                 {
                   cout << " b is max "<<endl;
          }      }

} ;
```

```
int main ()
{
    max ob;
    ob.cal();
    return 0;
}
```

Object as array | Array of Object →

To store multiple records in object we use object as array.

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

class Student
{ Public:
    string name;
    int age;
    int rollno;
    int fees;

    Public:

        void set()
        {
            cout << "Enter name" << endl;
            cin >> name;
            cout << "Enter age" << endl;
            cin >> age;
```

```
        cout << " Enter roll no " << endl;
        cin >> rollno ;
        cout << " Enter the fees " << endl;
        cin >> fees ;
    }
    void show()
    {
    //cout << "name \t age \t rollno \t fees "
                                    << endl;
      cout << name << "\t" << age << "\t"
           << rollno << "\t" << fees << endl;
    }
};
int main()
{   student ob[3];
    int i;
    for ( i= 0 ; i<=2 ; i++)
        {
          ob[i].set();
        }
    cout << "name \t age \t rollno \t fees " << endl;
    for (i = 0 ; i<= 2 ; i++)
      {   if(ob[i].fees < 5000)
          ob[i].show();
      }
  return 0;
}
```

not here

→ we use fees with '.' so data members should also public

Scope Resolution operator (::)

If we want to access class data outside the class we use this operator.

Algorithm –

```
#include <iostream>
using namespace std;
class x
{
    int a,b,c;
    Public:
        void set();
        void mean();
};

void x::set()
{
    cout << "Enter values" << endl;
    cin >> a >> b;
}

void x::mean()
{
    c = (a+b) /2;
    cout << mean = << c;
}

int main()
{
    x ob;
    ob.set();
    ob.mean();
} return 0;
```

max of 3 nos using ternary operator

```
#include <iostream>
using namespace std;
class max
{ int a,b,c;
  Public:
  void set();
  void maxi(); // name can't same
};
void max :: set()
  {
    cout << "Enter value of a,b,c";
    cin >>a>> b>> c <<endl;
  }

voiol max :: maxi()
{
   a>b && a>c ? cout <<"a is max" :
   b>a && b>c ? cout <<"b is max" :
   c>a && c>b ? cout << "c is max" :
}                cout << "a,b,c are equal";

int main()
{
    max ob;
      ob. set();
      ob. maxi();
      return 0;
}
```

Default Arguments →

- ## This Pointer -

If a class have same class ve level variable and local var. then to distin-guish or differentiate between them we use this pointer. Syntax → (this →)

```
class salary
{
    int bs, gs, ta, da;
    Public :
        void set (int bs, int gs, int ta,
                                    int da)
        {
            this → bs = bs;
            this → gs = gs;
            this → ta = ta;
            this → da = da;
        }
    // bs is in paramatric fuc. set and
       also in class var. ( data members)
       so we use this pointer which
       indicates that this are class
       variables →
};
```

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

class salary
{ int bs, gs, ta, da;

  public:
  void setbasicsalary ( int bs )
    {
      this -> bs = bs;
    }
  void cal()
   {  ta = (bs * 10) /100;
      da = (bs * 20) /100;
      gs = ta+da+bs;
   }
  void show ()
     {  cout << " basic sal = " << bs << endl;
        cout << " ta =" << ta << endl;
        cout << " da = " << da << endl;
        cout << "gross sal = " << gs << endl;
     }
};
int main ()
 {  salary ob;
    ob. setbasicsalary (10000);
    ob. cal ();
    ob. show ();
    return 0;
 }
```

- Setw manipulator →

To use it we have to use the header file → #include <iomanip>

Setw function is used to set the with width of content from the Right alignment

Syntax → setw(n)

← will display from this side
nspaces

- For example in last question →

```
#include <iomanip>
#include <iostream>
using namespace std;
class salary
{
    int bs, gs, ta, da;
public:
    void setbasicsal(int bs)
    {
        this -> bs = bs;
    }
```

```
Void cal()
{   ta = (bs*10)/100;
    da = (bs*20)/100;
    gs = ta+bs+da;
}

void show()
{
 cout << setw(15) << " basic salary = " <<
                                setw(7) << bs \n;

 cout << setw(15) << " da = " << setw(7) << da
                                << endl;

 cout << setw(15) << " ta = " << setw(7) << ta
                                << endl;

 cout << setw(15) << "gross salary=" << setw(7) << gs
                                << endl;
}
};

int main()
{   Salary ob;
    ob. set basicsal (10000);
    Ob. cal();
    ob. show();
    return 0;
}
```

- Default Arguments / variables →

If you want to set the value of variable defaultly we can use default Argument.

Default Argument set always set default value of variable.

If user pass value for default argu argument then default value will not work.

** It should always declared at the Right most side

eg → void Sim(int p , int t , int r = 5);

default rate = 5
at Right most side

WAP for simple intrest using default rate = 5

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

class Sim
{
    float p, r, t, SI;
    Public :
    void set ( float p , float t , float r=5)
    {
        this -> p = p;
        this -> r = r;
        this -> t = t;
    }
```

```
    void cal()
    {   si = (p*r*t)/100;
        cout << "si = " << si << endl;
    }
};

int main()
{   sim ob;
    int cs;     // credit score
    cout<< "Enter the credit score";
    cin >> cs;
    if (cs>7 && cs<=10)
        {   ob.set(10000, 5);
            ob. cal();
        }
    else
        {   ob. set (10000, 5, 8);
            ob. cal();
        }
    return 0;
}
```

Constructor →

- constructor is a special function who have same name as class.
- constructor always declared as Public
- constructor is automatically called when object is created.
- constructor is used to set default value of variable.
- constructor has no return type that may be void, int or anything.

Types of constructor –

1) Default constructor –

```
X ()      // no parameter
{ a=1;
}
```

2) Paramatrica constructor –

```
X (int b.)
{ a = b;
}
```

3] copy constructor

```
x (x &ob)        // add of object is send
{ a = ob.a ;
}
```

Programme for Default constructor -

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

class cons
{ int not, f1, f2, f, i;
  public :
   cons ()          // we can initialize in class using constructor
    { not = 5 ;
      f1 = -1 ;     // if no value given so no of term will = 5
      f2 = 1 ;
    }

   void set()
    {
      cout << "Enter no of term" << endl;
      cin >> not ;
    }

   void cal ()
    {
      for (i = 1 ; i <= not ; i++)
       { f = f1 + f2 ;
         cout << f << " , " ;
         f1 = f2 ;
1:     } f2 = f ;
```

```
int main()
{
    cons ob;
    ob.cal();
}
```

OR

```
int main()
{
    cons ob;
    ob.cal();
    cons obj; // new ob.
    obj.set();
    obj.cal();
}
```

WAP for factorial using constructor -

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

Class fact
{ int f, no, i;
    Public:
    fact()
    { f = 1;
      no = 5;
    }
    void set()
    { cout<< Enter no <<endl;
      cin >> no;
    }
```

```
void facto ()
  {  for (i=1 ; i<= no ; i++)
        { f = f* i;
        }
  }
void show()
  {  cout << " fact = " << f ;
  }
};
int main ()
  {
     fact ob ;
     ob. set ;
     ob. facto ;
     ob. show ;
  }  return 0;
  }
```

- Parametrice constructor with constructor overloading →

(Example for factorial → after gcd)

constructor overloading – when more than one constructor is called or defined.

gcd without overloading

```
# Include <Postream>
Using namespace std;

Class Gcd
{   Int no1, no2, i, hcf;
    Public :
    Gcd ( int no1, int no2)
      {  this no1 = no1;
         this no2 = no2;
      }
    Void cal()
     {  for (i= 1; i<= no1 && i<=no2; i++)
        {   if ((no1 %. i==0) && (no2 %. i== 0))
            {  hcf = i;
            }
        }
       cout << " gcd = " << hcf;
     }
};
int main()
{  Gcd ob(2,8);
   Ob. cal()
}  return 0;
```

now gcd using default constructor

```
class Gcd
{ int no1, no2, i, hcf;
  Public:
    Gcd ()
    { no1 = 2;
      no2 = 8;
    }
    void set ()
    { cout << "Enter value of no1 & no2" << endl;
      cin >> no1 >> no2;
    }
    void cal ()
    { for (i=1; i<=no1 && i<=no2; i++)
      { if (no1 % i==0 && no2 % i==0)
        { hcf = i;
        }
      }
      cout << hcf;
    }
};
  int main ()
  { gcd ob;
    ob.cal ();
  }
// default
```

OR

```
{ gcd ob;
  ob.set;
  ob.cal ();
}
// user value
```

- Constructor overloading with parameterize & copy constructor →

copy constructor → एक ob के add पर दूसरे object पर copy करता है।

```
# Include <iostream>
Using namespace std;

class conover
{
    Public :
        int a;
        conover (int a)
        { this → a = a;
        }

        conover (conover & obj)
        {
            this → a = obj.a;
        }
                    it means value of a

        Void show ()
        {
          cout << " value of a = " << a << endl;
        }
};
```

```
int main ()
{  conover obj (5);
   obj. show ();
   conover ob (obj);
   ob . show;
}
```

another example - Palindrome no

```
# include <iostream>
using namespace std;
class Palin
{  int no, r, no1, sum;
   Public :
   Palin ( int no)
   {  this -> no = no;
   }
   Palin ( Palin & object)
   {
      this -> no = object.no
   }
```

(pro)

```
void check()
{
    no1 = no;
    while (no>0)
    {   r = no % 10;
        sum = sum * 10 + r;
        no = no/10;
    }
    if (no1 == sum)
      {
        cout << "no. is palindrome";
      }
    else
      {
        cout << " no is not palindrome";
      }
  }
};
int main()
 {
    Palin ob(121);
    Palin obj(ob);
    ob.check();
 }
```

Destructor →

- Destructor is special func whose name is same as class but its use tile symbol (~)
- Destructor is automatically called when obj is destroy or scope of object is finished.
- Destructor destroy memory which which was allocated by constructor.
- There is no types of destructor.

Example →

```
X ( )        // constructor of
{                 class named x
}

~X ( )       // Destructor of
{                 class named x
}
```

{ Programe PTO }

```
#include <iostream>
using namespace std;
int count = 0;   // global var

class A
{
    public:
      A()
      {
        count ++;
        cout << " object created = " << count << endl;
      }
     ~A()
      {
        count --;
        cout << " object deleted = " << count << endl;
      }
};

int main()
{
   A ob1, ob2, ob3;

   return 0;
}
```

Function Overloading → (Polymorphism)

If two or more then two func. have same fn names but diff. parameter list or parameters.

eg →
```
area (int side)
{
}
area (int length, int breadth)
{
}
```

Example -

```
#include <iostream>
using namespace std;
class Ar
{
  float l, b, a, r, pi;
  Public:
    Ar()
    {
      pi = 3.14;
    }

  }; (PTO)
```

```
    void area ( float l, float b)
    {
        a = l * b;
        cout << "area of rect = " << a << endl;
    }
    void aread (float r)
    {
        a = pi * r * r;
        cout << "area of circle " << a << endl;
    }
};

int main()
{
    Ar obj;
    obj.area (4,5);
    obj.area (3.62);
    return 0;
}
```

Polymorphism →

Poly → many
morphism → forms

If we create many forms of same thing (construc, func) then its called polymorphism.

There are 2 types of Polymorphism –

- compile time polymorphism –
  ↳ overloading (Early binding)
- Run time polymorphism
  ↳ overriding (late binding)

- overloading –
- If method constructor, operator or template have same name but diff. forms then it's called overloading.

  eg – If there is 2 area func.
  (last prog. func. overloading)

- If there is more than one constructor with diff parameters then it's called constructor overloading. and more than one fuenc than called function/method overloading.

- If class have 2 func of same name and one uses template argument as parameter and other uses primitive type parameter then its called template overloading.

- the basic nature of operators is to perform operations on the primitive type of data but if we can perform same operations on object type of data then its called operator overloading.

- overriding —

- when a method in subclass have same name, and same parameters and same return type as a method in its superclass, then method in subclass is said to override the method in superclass. It is called overriding which comes in Run time polymorphism.

WAP using func. overloading which calculate the perimeter -

```
#include <iostream>
using namespace std;
// int pi = 3.14;

class per;
{ float l, b, p, r;

  public:
    void per()
      { pi = 3.14;
      }
    void rec ( float l , float b)
      { p = 2 * (l+b);
        cout << "perimeter of □ = " << p << endl;
      }
    void cir ( float r)
      { p = 2 * pi * r;
        cout << " perimiter of O = " << p << endl;
  };

int main ()
{   per obj;
    obj.rec (2, 6);
    obj.cir (2.16);
    return 0;
}
```

FRIEND FUNCTION →

- To define a func as friend function we use 'Friend' keyword.
- Friend function can access the private data of class.
- In friend func we always pass object as a argument.
- As Friend func. is not member of class so it can't directly access the funcs / members of class. It can access with the help of Obj.
- The scope of friend func is outside the class
- As Friend Func is not member of class we can't call it with help of object. It can be called directly.
- Friend function can be friend of more than one class and we have to pass object of all classes in which it defines as friend.

- obj of class is passed as parameter in the friend function.

Implementation – (in one class) →

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

class Rev {      // Rev no.
    int no, r, sum;
    public: Rev() { sum = 0; }  // constructor
        void set()
        { cout << "Enter the no";
          cin >> no;
        }
        friend void rev (Rev ob);
                          //object as argument
    };
    //class over

void rev (Rev ob)          // we can access private
{    while(ob.no > 0)        members & methods
     {                       using object
       ob.r = ob.no % 10;
       ob.sum = ob.sum * 10 + ob.r
       ob.no = ob.no / 10;
     }
     cout << "reverse no = " << ob.sum;

}
```

```
int main()
{
    Rev obj;
    obj.set();
    rev(obj);   // called directly without using object but object passed as parameter
    return 0;
}
```

Implementation in 2 class →

```
#include <iostream>
using namespace std;
class B;   // Forward declaration
           2nd class to be declared before 1st class

class A
{   int a;
    Public:
    Void set()
    {
      cout << "Enter value of a" << endl;
      cin >> a;
    }

    friend Void max(A ob, B obj);
};
                    // Objects of both classes are passed
```

```
class B
{ int b;
  Public:
     Void set()
     { cout << "Enter value of b" << endl;
       cin >> b;
     }
     Friend Void max (A ob, B obj);
};

Void max (A ob, B obj)
  {
     If (a.ob > b.obj)
        cout << "a is max";
     else if (a.ob < b.obj)
        cout << "b is max";

     else
        cout << "both are equal;
  }
int main ()
  { A ob;
    B obj;
    ob.set()   // set of A class
    obj.set()  // set of B class

    max (ob, obj);
  }
```

syntax error

a.obj ✗

ob.a ✓

Programme for Restaurent using func. over

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

class Rest
{ int bill, qty;
  public:
    void order()
    { cout << "you order normal thali" << endl;
      bill = 250 * 1;
      cout << "bill amount =" << bill << endl;
    }
    void order (int thalino)
    { if (thalino == 1)
      { cout << "Enter no of thali
                you want " << endl;

        cin >> qty;
        cout << "you ordered normal
                  thali" << endl;
        bill = 250 * qty;
        cout << " bill amount = " << bill;

    }
```

```
        else if (thaliNo == 2)
        {   cout << "Enter the no of thali you
                        want" << endl;
            cin >> qty;
            cout << " you ordered vip thali" << endl;
            bill = 500 * qty;
            cout << "bill amount :" << bill;
        }
    }
};

int main()
{
    Rest ob;
    int ch, ch1;
    cout << " ..... MENU ..... " << endl;
    cout << " 1. It" << "Normal thali" << endl;
    cout << " 2. It" << " other thali with
                        selected qty" << endl;

    cin >> ch;
    system ("clear");   // clear screen
    switch (ch)
    {   case 1:
            ob. order ();
            break;
```

```
case 2 :
cout << "1. It" << "normal thali with
                    selected qnty " << endl;
cout << "2. It" << "Vip thali with selected
                    quanty " << endl;
cin << ch1 ;
        System (clear);
    ob.order (ch1);
    break;

dejault :
    cout << "invalid choice";
}
}
```

INHERITANCE →

- One class can access the property of another class is called inheritance. (inherit = access)

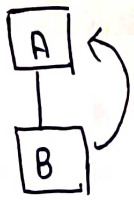
- Always only child class can inherit / access data of parent class but parent class never access the data of child class.

- that means child class obj can access parent class methods but parent class obj can't access child class method.

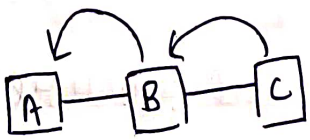
- Only Public and Protected data can be inherited by child class. Private data can't be inherited.

- In C++ there are 5 types of Inheritance.

1) Single level Inheritence
   - one parent and one child class

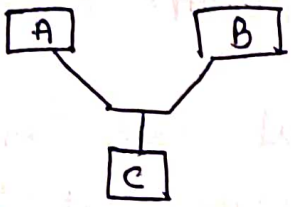


2) Multi level Inheritance
   - 2 parent & 2 childs



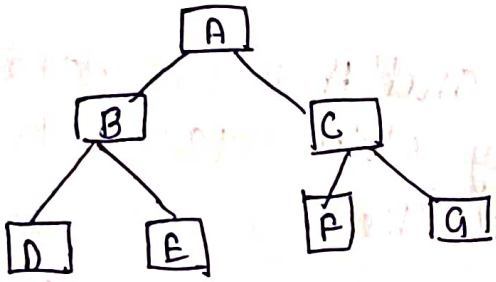
3] Multiple Inheritance

- 2parents one child



4] Hierarchical Inheritence

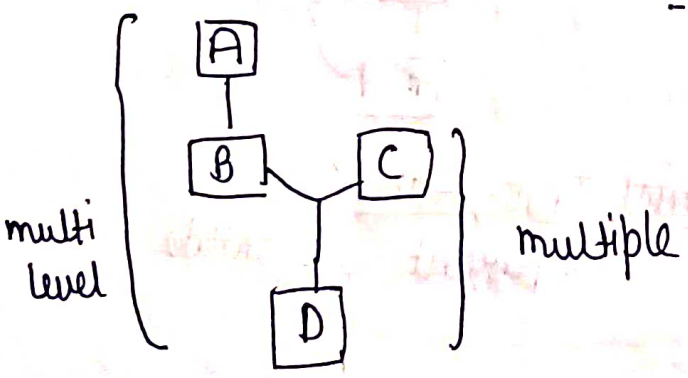
- Joint family / system



Tree structure

5] Hybrid Inheritance

- combination of multi level & multiple Inheri-tance



opposite tree structure

- By inheritance we [illegible] this is an application.

Example -

** Protected data → (If we inheriting some protected data) it can be accessed from public method of child class i.e. protected data member or member func. Can't be accessed with the help of object.

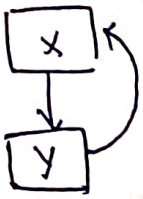
- If data member and member func declare as a public then it can be accessible thru the object.

Example → single level

```
class X
  {
    protected:
      int a,b;
    Public:
      void set()
    { cout<<"Enter";
      cin>>a>>b;
    }
};
```

```
x inherited by y
↑
class y: public x
{
    int c;
    Public:
    void add()
    { c=a+b;
      cout<<c;
    }
};
// ⇒ y→ child class
     x→ parent class
//so obj of y will access oo child class can accesses parent data.
```

```
int main()
{
    Y ob;
    ob.set();
    ob.add();
    return 0;
}
```



Single level

In same que we will make set() method also protected - (class A)

By doing this object can't call the set method so it will called by the child class →

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

class A
{   protected:
        int a, b;
        void set()
        { cout << "Enter";
          cin >> a << b;
        }

};
```

```
class B : public A
{ int c;
  public:
    void add()
    { set();
      c = a+b;
    }
    void show()
    {
      cout << "value =" << c;
    }
};

int main()
{
  B ob;
  ob.add();
  ob.show();
} return 0;
```

WAP for login class using single level inheritance

```
#include <iostream>
#include <string.h>
using Namespace std;
```

(PTO)

```
class Log
{
    protected :
    char username[20];
    char password [20];

    Public :
      Log ()    // constructor
      {
       strcpy (username, "admin");
       strcpy (password, "admin123");
      }
};

class User : public Log
{   protected :
    char Uname [20];
    char pass [20];

    Public :
      Void set ()
      { cout<< "Enter username";
        cin >> Uname;
        cout << " Enter password";
        cin >> pass;

      }
```

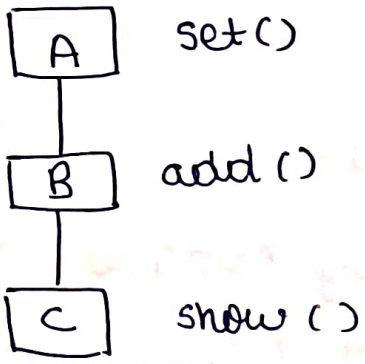
we use !
∴ ! ⇒ not
equal to
if ∴ strcmp
returns 0 if
strings are same

```
Void check()
{
 if (!(strcmp(username, uname))
     &&(!(strcmp(password, pass))))
 {  cout << "Login";
 }
 else
 {
   cout << "Login fail";
 }
 }
};

int main()
{
    user ob;
    ob.set();
    ob.check();
    return 0;
}
```

- Example → Multi level inheritance →

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

class A
{
    protected :
    int a, b;
    Public :
    void set ()
    { cout << "Enter a & b";
      cin >> a >> b;
    }
};

class B : public A
{
    protected :
        int c;
    Public :
        void add ()
        {
            c = a+b;
        }
};
```

```
class C : public B
{   public :
     void show ()
     {
       cout << "value of c =" << c ;
     }
};

int main ()
{
     C ob ;
     ob. set () ;
     ob. add () ;
     ob. show () ;
     return 0;
}
```



multi level

WAP for multilevel inheritance which have three class →

- class User → username & Pass
- class login → login sucess / fail
- class Bank → Net banking methods

```
#include <iostream>
#include <string.h>
#include <stdlib.h>
Using namespace std;

class User
{
    protected:
        char uname[20];
        char pass[20];
        public:
            User()
            {
              strcpy(uname, "admin");
              strcpy(pass, "admin123");
            }

};
```

```
class login : public
{
    protected :
    int flag ;
    char username[20];
    char password [20];

    public :
    login ()   // constructor
    {  flag = 0;
    }

    void set ()
    {
       cout << " Enter username" ;
       cin >> username;
       cout << " Enter password";
       cin >> password;
    }

  int userlogic ()
  {  if ((! strcmp (username, uname)) &&
                 (! strcmp (password, pass )))
       {
          flag = 1;
       }
```

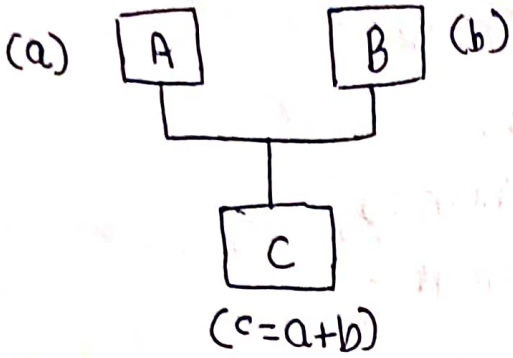
```
    else {
        cout << " login fail " << endl;
        flag = 0;
    }
    return flag;
  }
};

class Bank : Public Login
  {  Protected :
     int bal, amt;

     Public :
     Bank ()
     {
        bal = 5000;
     }
     void Deposit ()
     {  cout << " Enter amount " << endl;
        cin >> amt;
```

```
if (amt > 0)
  {  cout << " amt deposit = " << amt << endl;
     bal = bal + amt ;
     cout << " total bal = " << bal << endl;
  }
  else
  {  cout << " invalid amount " ;
  }
void withdrawal ()
  {  cout << " Enter amout for withdraw " ;
     cin >> amt ;
     if (amt > 0 && amt <= bal )
     {  cout << " amout withdrow " << amt;
        bal = bal - amt ;
        cout << " total bal = " << bal ;
     }
     else {
          cout << " invalid amount " ;
        }
}
```

```
void show()
{
    cout << "total bal" = << bal;
}

Void check()
{
    int f;
    int ch;
    set();
    f = userlogin();
    if (f == 1)
    {
        cout << "press 1 for deposit" << endl;
        cout << "press 2 for withdraw";
        cout << "press 3 for show balance";
        cout << "press 4 for exit";

        cin >> ch;

        switch (ch)
        {
            case 1:
                deposit();
                break;
```

```
            case 2 :
                withdrawal () ;
                break ;
            case 3 :
                show () ;
                break ;
            default :
                exit (0) ;
};  } } }

int main()
{   Bank ob ;
    ob . check () ;
    return 0 ;
}
```

• Example : Multiple Inheritance →



one child class have more than one Parent class

```
# Include <iostream>
Using namespace std;

class A
{
    protected :
     int a;
    Public :
     A ()
     { a= 10;
     }

};
```

⇒

```
class B
{   protected:
        int b;
    public:
    B()
    {   b = 34;
    }
};

class C : public A, public B
{   int c;
    public:
    void add()
    {   c = a + b;
    }
    void show()
    {   cout << "Sum =" << c;
    }
};
```

```
int main()
{
    c ob;
    ob.add();
    ob.show();
    return 0;
}
```

WAP using multiple inheritance →

Holiday / vaccation Programme

class flight → flight NO.
data Name, Date
Price

class Hotel → Hotel Name
data Name, Date
Price

class show/ → show all
custmer the details

`use multiple inheritance'

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

class flight          protected:
{    int flno ; price ; Date
   *(char passname , Date)*
    string passname , Date ; fname ;
    public :
        void set ()
        { cout << "Enter flight Name" ;
          cin >> fname ;
          cout << "Enter flno " ;
          cin >> flno ;
          cout << "Enter Date" ;
          cin >> Date ;
          cout << " Enter the Price"
           cin >> Price ;
        }

};
```

similarly make class hotel and inherit the classes →

Better version for the programe -

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

class flight
{
    protected:
        char fname[20];
        char date[20];
        int price;
        int ch;

    public:

    void set()
    {
        cout << "press 1 for indigo";
        cout << "press 2 for Air India";
```

```
cout << "press 3 for vistara";
cin >> ch;
switch(ch)
{
    case 1:
    strcpy (Fname, "indigo");
    cin.ignore();
    cout << "Enter the dep date";
    cin.getline (date, 19);
    price = 7000;
    break;

    case 2:
    strcpy (Fname, "AirIndia");
    cout << "Enter date";
    cin.getline (date, 19);
    price = 5000;
    break;
```

work as getch() in C

← cin.ignore();

string के बाद str or int के बाद str

19 char का Array

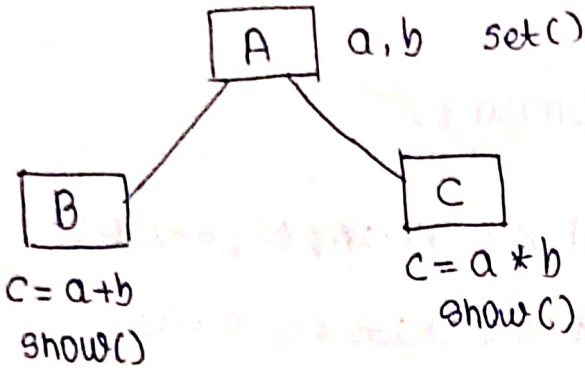
```
                case 3:
                strcpy ( frame , "Vistara" );
                cout << "Enter the date" ;
                cin . getline (date, 19) ;
                price = 9000 ;
                break ;

                default :
                cout << "invalid choice" ;
            }
        }
];


class Hotel
[   protected :
        char hname [20] ;
        int noofday nod ;   // no. of day
        int price1 ;
        int ch ;
        int bill ;
```

```
public :
void set1()
{ cout << "Press 1 for sheraton";
  cout << "Press 2 for wow";
  cout << "Press 3 for Taj";
  cin >> ch;
  switch (ch)
  {
     case 1 :
       strcpy (hnamo, "sheraton");
       cout << "Enter the nod";
       cin >> nod;
       price 1 = 10000;
       bill = price 1 * nod;
       break;

     case 2 :
       strcpy (hnamo, "wow");
       cout << "Enter no of days";
       Cin >> nod;
```

```
            price 1 = 8000;
            bill = price1 * nod;
            break;

        case 3:
            strcpy ( hname, "taj");
            cout << " Enter no. of day";
            cin >> nod;
            price 1 = 25000;
            bill = price 1 * nod;
            break;

        default:
            cout << "invalid choice";

        }

    }
};
```

```
class custmer : public F, public H
{  public:
     void show()
     {  cout << fname << endl;
        cout << date << endl;
        cout << price << endl;
        cout << hname << endl;
        cout << nod << endl;
        cout << bill << endl;
     }
};

int main()
{  custmer ob;
   ob.set();
   ob.set1();
   ob.show();

   return 0;
}
```

• Example → Hierarchical Inheritance →



Class A inherited by B & C so object of both B, C child classes will be created

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

class A
{
    protected:
   int a, b
    public:
    void set()
     { cout<< "Enter a & b";
       cin >> a >> b ;
     }

};
```

```
class B : Public A
{ protected :
  int c;
  void add()
  {
    c = a+b;
    cout << "Sum = " << c;
  }
};

class C : Public A
  { protected :
    int c;
    voi Public :
    void mul()
    {
      c = a * b;
    } cout << "mul = " << c;

  };

int main()
{   B ob;
    C obj;
    ob. set();
    ob. add();
    obj. set();
    obj. mul();
}
```

WAP using Hirarichical inheritance -

class grandfather → properties

flat → 5000000

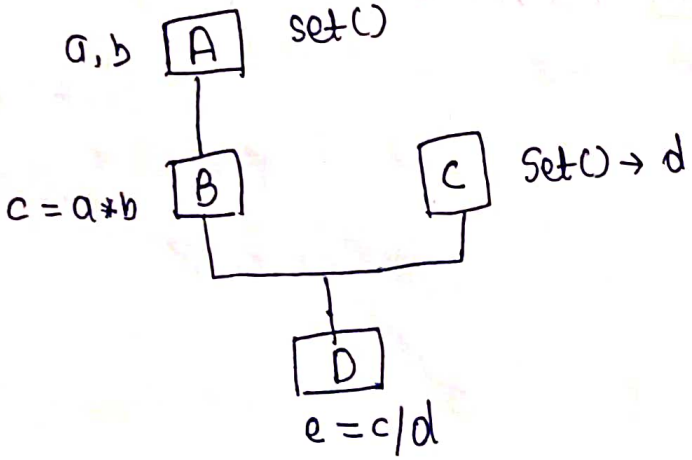
House → 1p,0p0000

class child A → should access flat
if not then deny

class child → B should Access house
if not then Deny

(code written after Hybrid)

- Example → Hybrid Inheritance →



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

class A
{   protected :
    int a, b;
    public :
    void set ()
    {   cout << "Enter a, b";
        cin >> a >> b;
    }
};

class B : public A
{   protected :
    int c;

    public :
    void mul ()
    {   c = a * b;
        cout << " c = " << c;
    }

};
```

```
class C
{
    protected:
    int d; Public:
    void setd()
    { cout<<"Enter value of d";
      cin>>d;
    }
};

class D : Public A, Public B, Public C
{     protected:
      int e;
      Public:
      void div()
      { e = c/d;
        cout<< e;
      }
};
```

```
int main()
{
    D ob;
    ob.set();
    ob.mul();
    ob.setd();
    ob.div();
    return 0;
}
```

EARLY BINDIND

Method Overwarding → (Inheritance)
(Run time polymorphism ≈ Overriding)

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

class A
{ public:
    void show()
    { cout << "super class" << endl;
    }
};
```

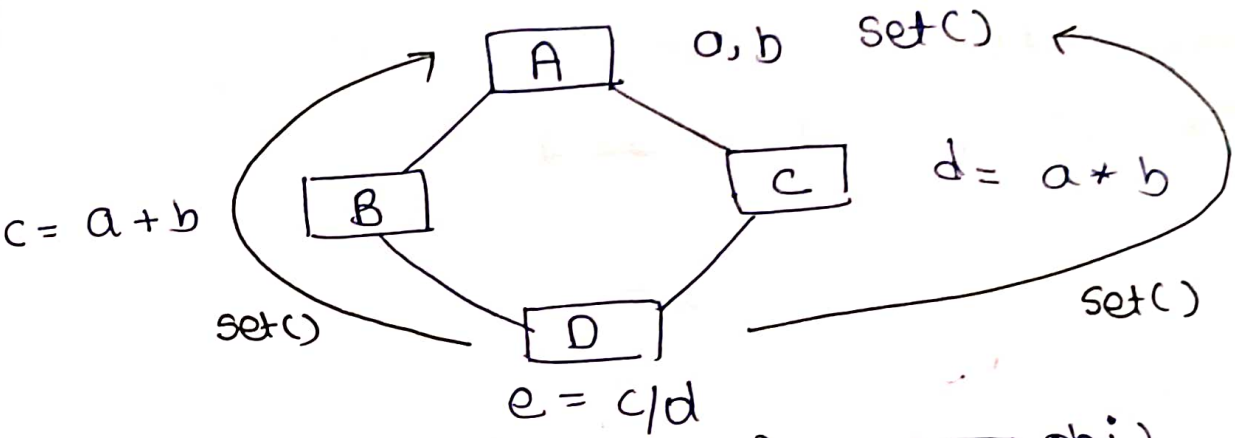
(using single level inheritance)

```
class B : public A
{
    public:
        void show()
        {
            cout << "subclass" << endl;
        }
};

int main()
{
    B obj;
    obj.show();
    obj.A::show();
}
```

when there is same function name in methods of child classes then compiler gets confused so we have to use scope resolution operator for parent class.

Diamond Problem → (inheritance)

We use virtual base class to solve diamond problem →

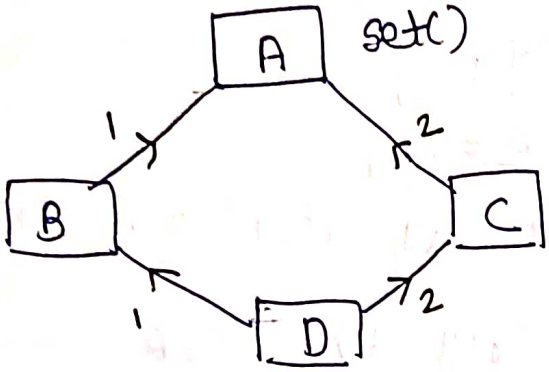


(2 bar set() को call करेगा D का obj)

Yeh when we try to work with Diamond Inheritance as in above fig. then there may be ambiguity comes (confussion) with methods of superclass A where two reference of can be created.

To Resolve this problem we can use virtual keyword with superclass A that is by creating a class A as virtual base class.

- Firstly when we create the ob. its search methods in that class only.
- If he couldn't find in that child class then it starts searching int the parent classes.



- Here ob of D will check for set by path ① ~~the~~ via class B and call set from A
- Then ob of D again will check for set in C means from path ② (via C) then again call set from A
- same ob. will called 2 times so ambiguity.

(If class Inherit [illegible] keyword use virtual)

```
# Include < Postream >
Using namespace std;

class A
{   Protected:
    int a, b;
    Public:
    Void set C)
    {
        cout << " Enter value of a and b";
        cin >> a >> b;
    }
};

class B : Virtual Public A
 {   Protected:
     int c;
     Public:
     void add C)
     {
         c = a + b;
         Cout << "sum =" << c;
     }
 };
```

```
class C : virtual public A
{
    protected :
    int d;
    public :
    void mul()
    {
      d = a * b;
      cout << "mul = " << d;
    }
};

class D : public B , public C
{
    int e;
    public :
    void div()
    { e = d / c;
      cout << "div = " << e;
    }
};
```

```
int main()
{   D ob;
    ob.set();
    ob.add();
    ob.mul()
    ob.div()
    return 0;
}
```

Early Binding in method overriding →

- overriding means Run time Polymorphism
- Ambiguity due to same method name in 2 different classes inheriting each other.
- when compiler know at compile time which method to be executed then its called Early Binding.
- Example we have seen earliear in method overriding is a Early Binding.

```
class A
{
    public:
    void show()
    {
     cout << "Hello";
    }
};

class B : public A
{
   public:
   void show()
  {  cout << "world";
  }
};

int main()
{
   B ob;
   ob.show();
   A::ob.show();
   return 0;
}
```

Base class pointer →

- Always pointer will be formed of base class or parent class.
- Base class pointer can hold reference of child class as well as parent class

Example →

```
# Include <iostream>
Using namespace std;

class A
{   Public :
    Void show ()
     {  cout << " base class | parent class " ;
     }
};

class B : Public A
{
    Public :
```

```
    void show()
    {
        cout << "child class";
    }
};

int main()
{
    A * bptr ;   // base class pointer

    bptr A ob ; // base class object

    bptr = &ob ;

    bptr -> show() ;
}
```

arrow

Output → Base class

But if we do like this

ALTER main func →

⇒

```
int main()
{
    A * bptr ;
    B ob ;   // child class obj
    bptr = &ob;
    bptr -> show();
}
```

output again → base class

" किसी का भी object बनाओ, हमेशा Base class का ही show call होगा as pointer is of base class"

child class का Pointer बन ही नहीं सकता।

To solve this problem we use late Binding.

Late Binding in method overriding and VIRTUAL function →

- In method overriding ; we use the 'VIRTUAL' keyword for late binding then method of that class is called whose object is created.
- Base class का function ही हमेशा Virtual function बनेगा ।

```
# include <Postream>
Using namespace std;
class A          // Base class
{  Public :
     Virtual Void Show C)
     { cout << " base class";
     }
};
```

```
class B : Public A
{ Public :
    void show ()
    { cout << "child class" ;
    }
};

int main ()
{   A * bptr ;
    A ob ;
    bptr = &ob ;
    bptr → show ();
}
```

output = Base class

ALTER

```
int main ()
{ A * bptr ;
  B ob ;
  bptr = &ob ;
  bptr → show ();
}
```

output = child class

virtual func बनाने से जीस class का object बनाएगे and उसे base pointer से call करेगे तब उसी class की show method call होगी "

Inheriting Types →

- while Inheriting syntax is →

  class B : Public A

  ↓

  we can do it privatly or Protectedly also

- IMP chart →

→ Data type / Access modifiers in Parents →

↓ Inherited by Value ↓

X	Private	Protected	Public
Private	X	Private	Private
Protected	X	Protected	Protected
Public	X	Protected	Public

X → Cant Access or Inherited

- Private data can't be accessed by any child class (inherited) only Public & Protected data can accessed.

- Protected and Public Data Inherited Privately are treated as Private so can't be accessed with object

example →

```
class A
{ Protected :
   int a, b ;
   Public :
   void set ()
   { cout << " Enter";
     cin>> a ; b;
   }
};

class B : Private A .
{
  // a,b and set() will
      treated as Private

   Public :
   set();
   void show()
   { set;
     cout << "value = " <<a, K<b;
   }

};
```

called using Public method ←

```
int main()
{
    B ob;
    // ob.set(); →
    ob.show();
}
```

will give error as set() considered as private so can't accessed by the object.

- Public and Protected Data inherited Protectedly treated is protected so can be called in public method of same or child classes.
- Protected data inherited Publicly treated as Protected.
- Public data inherited Publicly treated as Public.

Structure →

- Structure in C++ can have method, membere ands constructors.
- structure can't have access modifiers and Inheritance.
- structure data accessed by structure object or structure veriables.

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

Struct student
{       char name(20);
        int age ;
        student ()   // constructor
        {   strcpy (name, "mohit");
            age = 1008 ;
        }
```

```
    void show()
    { cout << name;
      cout << age;
    }
} ob;

int main()
{
    // struct student ob;
    ob.show();
    return 0;
}
```

Operator Overloading →

- compile time Polymorphism.
- Operator always works on primitive type of data. eg → int, float etc.

ex →
```
int a, b, c;
a = 10;
b = 20;
c = a + b;
```

- when operator can perform operations on object type of data then it called operator overloading -

eg →
```
class X
{
};

int main()
{
    X ob1, ob2, ob3;
    ob3 = ob1 + ob2;
    return 0;
}
```

- we have to do $ob3 = ob1 + ob2$ →

```
class x                          void add()
{  int a, b;
};
```

```
int main()
{   x ob1, ob2, ob3;
    ob3 = ob1.add(ob2);
}
```

ob1 ↓ call करने वाला है

ob2 ↓ passed as parameter

→ How to call the function

- method → `x add(x ob2)` will return object so class type the func

```
{
    x temp;   // new ob.
    temp.a = a + ob2.a;
    temp.b = b + ob2.b;
    return temp;
}
```

a, b ↓ जीस func ने call किया उसके a & b ⇒ ob1 के a, b

ob2.a, ob2.b ↓ parameter object के a and b ⇒ ob2 के a, b

- but we have to use '+' sign
  - x add (a, b) is a function
  - we can change the name →

eg →

x operator + (a, b)

↓

'use operator keyword'

and we can call just like

$ob_3 = ob_1 + ob_2$;

- c++ code → (Binary operator overload)

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

class x
{  int a,b ;
   Public :
   void set ( int a, int b)
   {
      this -> a = a ;
      this -> b = b ;
   }
```

```
X operator+ (X ob2)
{
    X temp;

    temp.a = a + ob2.a;
    temp.b = b + ob2.b;

    return temp;
}

Void show ()
{
    cout << "value of a = " << a;
    cout << "value of b = " << b;
}

};

int main ()
{
    X ob1, ob2, ob3;
    ob1.Set (5,2);
    ob2.Set (3,4);
    ob3 = ob1 + ob2;
    ob3.show ()
}
```

- Binary operator overloading →

These are → - , + , * , > , < , <= , >= , / operators which works on bits are etc. overloaded. 2 ~~opea~~ objects required.

- Uniary operator overloading →

This takes single or 1 object for overloading. eg → ++ , -- , — negation or NOT
+= , -= etc.

Example →

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

class x
{   int a, b;
    public ;
    void set (int a, int b)
    {
        this -> a = a ;
        this -> b = b ;

    }
```

```
x operator - ()
{
    x temp;
    temp.a = -a;
    temp.b = -b;
    return temp;
}
Void Show()
{
    cout<< "value of a=" << a;
    Cout << "value of b=" << b;
}
};

int main()
{
    x ob1, ob2;
    ob1. Set (5,2);
    ob2 = - ob1;
    ob2. Show();
    return 0;
}
```

Templates →

- Templates are used for generic programming.
- generic programming → no strict declaration means if we pass int, char, float anything as parameter code must executed
- In cpp → 'template' keyword.
- By template keyword we can create the variables of generic type that is they can handle all types of data.

Example →

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

- There are two types of templates →
  1) function Template
  2) class Template

Function Template —

```
# include < iostream >
using namespace std;
template < class t >   // t is a data type now
                       // not means that class, only a part of syntax

Void show (T a)        // T type का a pass किया
{
cout << "value of a =" << endl;
}

int main ()
{
    show (3);
    show ('a');
    show (7.9);
}
```

// we can pass any value Data type now

ee Another syntax → template < typename x >

WAP using 2 templates →

```
#include <iostream>
using namespace std;
template < class T ; class T1 >
void show (T a , T1 b)
{   cout << " value of a = " << a << endl;
    cout << " value of b = " << b << endl;
}
int main()
{   show (3 , 'a');
    show ("a' , 2);
    show (7.9 , 'y');
    return 0;
}
```

WAP which give max using template →

H. W.

- Template Overloading →

more than 1 func. with same name and minimum one should by of the template type.

Example →

```
# Include <iostream>
using namespace std;
template < class T, class T1 >
void show ( T a , T1 b)
  {
    cout << " template func " ;
    cout << " value of a = " << a ;
    cout << " value of b = " << b ;
  }
Void show ( int a , int b )
{     cout << " normal func " ;
      cout << " value of a = " << a ;
      cout << " value of b = " << b ;
}
```

```
int main()
{   show(3,5);  → will call normal fun
    show('a',2);
    show(7.9,3);  → will call template func.
}
```

- Function return template value →

```
#include<iostream>
using namespace std;
template<class T, class T1>
T max(T a, T1 b)   // will return a template
{   if(a>b)
        return a;
    else
        return b;
}
int main()
{   cout<<"max="<<int,int>(5,2)<<endl;
    cout<<"max="<<char,int>('a','z')<<endl;
}
```

tells the Data type of parameter

Class Template →

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

template <class T, class T1>

class A
{  public :
   void show (T a, T1 b)
    { cout <<a << b << endl ;
    }
};

int main ()
{
    A <int, char> ob ;     // Always use object while declaration only
    ob. show (6, 'z');
    return 0;
}
```

Exceptional Handling →

- Exceptions are unwanted events which disturb the normal flow of programmes.
- Exceptional Handling is a mechanism used to Handle the exceptions.
- In c++ there are 3 keywords used for Exceptional Handling →

1) Try → It is a block in which that code will written in which there will be possiblity of exception.

eg → c = a/b

exception when b = 0

2) catch → This block always handle the exception

3) Throw → Throw throws the exception to the catch block.

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

class Exp
{   int a,b,c;
    public:
    void set()
    {   cout<<"Enter values of a & b"<<endl;
        cin>>a>>b;
    }

    void exp()
    {   try{
            if(b!=0)
            {   c=a/b;
                cout<<"div="<<c;
            }
            else
            {
                throw 0;
            }
        }
```

```
    catch (int x)  (// will catch that 0 by throw)
    {
        cout << "can't divide by zero = " << x ;
    }
  }
};
int main()
{  Exp ob;
   ob.set();
   ob.exp();
  }
```

ALTER →

```
  else {
       throw " can't divide ";
      }
};
  catch ( char *msg)
    {
      cout << msg ;
    }
```

- universal catch block →

```
catch (...)
{
}
```

can catch any throw (no fixed Data Type)

Friend Class →

- A friend class can access private and protected data members & member func. of other class in which its declared as a friend.

- In friend class, A function of friend class is always a friend class.

```
#include <iostream>
Using namespace std;
class B;
```

```
class A
{   int x;
    Public:
    A()
    { x = 10;
    }
    friend class B;
};

class B
{   Public:
    void disp( A ob)
    { cout << " value of x= " << ob.x << endl;
    }
};

int main()
{   A ob;
    B obj;
   obj.disp(ob);
}
```

WAP for factorial using friend class

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

class A
{
    int no;
    public:
    void set()
    {
        cout << "Enter no";
        cin >> no;
    }
    friend class B;

};
```

```
class B
{   int i, f;
    public:
    B()
    {
      f=1;
    }
    void fact (A ob)
    {
       for (i=1 ; i<= ob.no ; i++)
        {     f = f * i ;
        }

       cout << "factorial = " << f ;
    }
};

int main ()
{    A ob;
     B obj ;
     ob. set ;
     obj. fact (ob);
}
```

NAME SPACE →

- Namespace in c++ is used to organise too many classes so that it can be easy to handle the application.

- To use namespace methods, use the following syntax →

" namespacename :: methodname ; "

- creating a namespace →

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

namespace xyz
{   int add (int a, int b)
    {   return a + b;
    }

}
```

```
int main()
{
    cout << xyz::add(2,3);
}
```

Example →

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

namespace xyz
{  int add(int a, int b)
    {  return a+b;
    }
}

namespace abc
{  int add(int a, int b, int c)
    {
      return a+b+c;
    }
}
```

```
int main()
{
    int x, y, z;
    x = xyz::add(2,3);
    y = abc::add(2,3,4);
    z = x+y;
    cout << z;
}
```

- without scope resolution operator →
  - 1st define
  - then use those namespace

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

namespace devansh
{   int add(int a, int b)
        {return a+b;
        }
}
```

```
using namespace Devansh;

int main()
{   int a;
    a = add(5,3);
    cout << a;
}
```

- Another way →

create a header file → using '.h'

eg → Devansh.h
↳ create namespace here

then in cpp file →

```
# include" Devansh.h"
using namespace xyz;
    :
```

use all namespaces declared in Devansh.h

Abstract class and The Pure Virtual Function →

- Hiding the implementation is called abstraction.
- In C++ abstraction can be achieved by abstract class or access modifier.
- inheritance used for it
- To create a class as Abstract class we have define one - 'Pure virtual function fn ft'
- no object of abs. class.
- Implementation is in the child class which inherit it.
- Can be called a method overriding.

```
class Abc
    {
      virtual void area () = 0;
    };
```

```
class xyz : Public Abc
  {  void area ()
     {
       Implementation 1
     }
  };
```

```
class Pair : Public Abc
  {    void area ()
       {
         Implementation 2
       }
  };
```

Example →

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

class Abc
{ public :
   virtual Void area () = 0;

};

class xyz : Public Abc
{   Rublic :
     Void area ()
     {  cout << "area of circle" << endl;
     }
};

int main()
{   xyz ob;
    ob.area();

}
```

FILE I/O HANDLING :-

File handling is used to store data permanently in a computer. Using file handling we can store our data in secondary memory (Hard disk)

How to achive file Handling →

For achiving file handling we need to follow the following steps →

Step 1 → Naming a file

Step 2 → opening a file

Step 3 → writing data into file

Step 4 → Reading data from file

Step 5 → closing a file

Streams in C++

we give input to executing programs and the execution program gives back the output. The sequence of bytes given as input to the executing program and the sequence of bytes that comes as output from the executing programe are called Stream. In other words, Streams are nothing but flow of data in the sequence.

The input and output operation between the executing programme and the devices like like keyboard and monitor are known as "console I/O operation". The input and output operation bet the executing programme and files are known as "Disk I/O operation"

classes for file stream operation →

The I/O system of C++ contains a set of classes which define the file handling methods. This include ifstream, ofstream and fstream classes. These classes ~~(designated to manage the disk files, and)~~ are derrived from fstream and from the corresponding iostream classes. These classes, designated to manage the disk files, are declared in fstream and therefore we must include this file in any program that uses files.

1] ios :- (iostream)

- ios stands for input output stream.
- This class is the base class for other classes in the class hierachy.

- This file contains the defination of cin and cout functions etc. its called iostream file

- This class contains the necessary facilities that are used by all the other derived classes for input and output operations.

- modes are written as →

```
ios::open
ios::in
ios::out
ios::app
```

2] istream :-

- istream stands for input stream
- This class is derived from the class 'ios'.
- This class handle input stream.
- The extraction operator (>>) is overloaded in this class to handle input streams from files to program execution.
- This class declares input functions such as get(), getline() and read().

3] Ostream :-

- ostream stands for output stream.
- This class is derived from class 'ios'.
- This class handle output stream.

- The insertion operator (<<) is overloaded in this class to handling output streams to files from programme execution.
- This class declares output functions such as put() and write()

4] fstreambase :-

- This class provides operations common to the file stream.
- serves as a base for fstream, ifstream and the gstream class.
- This class contains open() and close() functions.

5] ifstream :-

- This class provides input operations.
- It contains open() function with default input mode
- Inherits the function get(), getline(), read(), seekg() and tellg() functions from pstream.

6] ofstream :-

- This class provides output operations.
- It contains open() function default output mode.
- Inherits the function put(), write(), seekp() and tellp() func. hom the ostream.

7] fstream :-

- This class provides support for simultaneous input and output operations.
- Inherits all functions from istream and ostream classes through iostream.

modes for file handling table →

member constant	stands for	Access
in	input	File open for reading: the internal stream buffer supports input operations.
out	output	File open for writing: the internal stream buffer supports output operations.
binary	binary	operations are performed in binary mode rather than text.
app	append	All output operations happen at the end of the file, appending to its existing contents.

WAP to write a line in the file →

```
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
    ofstream obj ("B.txt");
    obj << "welcome to file";
    obj.close();
    return 0;
}
```

// ofstream का obj out mode use करेगा by default

OR obj ("A.txt", ios::out);

WAP to append line in file →

```
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
    fstream obj ("B.txt", ios::app);
    obj << "this is file handling";
    obj.close();
    return 0;
}
```

7. # WAP which reads data from file → (single word)

```
# include <iostream>
# include <fstream>
using namespace std;
int main()
{
    string str;
    ifstream obj("B.txt", ios::in);
    obj >> str;   // will read one word from file
    cout << str;  // print on terminal or console screen.
    ob.close();
    return 0;
}
```

WAP to read whole file →

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

int main()
{
    string str;
    ifstream obj("B.txt", ios::in);

    while (getline(obj, str))
    {
     cout << str;
    }

    obj.close();

    return 0;
}
```

obj → कहाँ से Read

str → किस में store

- write function →

This function is used to write the object type of data in the file.

Syntax ⇒

write (—— , ——)

↙ किस object का data file में enter करना है

↘ उस obj का size क्या है / (कितना data)

- we use this pointer to enter the current object data in the file.

- Now write() func. stores in char type so we have to typecast 'this' object in char type

for eg → object is obj. →

"obj.write((char *)this, sizeof(*this));"

↙

this object pointer points to obj

- Read

this function is used to read object type of data from the file.

syntax → Read (_ , _)

- _ : address of obj that data file से read
- _ : size of that obj

- we have to use this pointer and typecast it to char.

- It reads only only one line / record from the file so we have to use while / for loops to read whole file

for example →

object is obj →

" obj. Read ((char*) this , sizeof (*this)); "

this obj pointer point to object

Project in C++ → "Book Management"

```
#include <iostream>
# Include <fstream>
# include <string.h>
# include <stdio.h>
# include <stdlib.h>
using namespace std;

class Book
{
  private:
    int bookid;
    char title[20];
    float price;

  Public:

  Book()
  {
    bookid = 0;
    strcpy (title, "no title");
    price = 0;
  }
```

```
void getBookData()
{
    cout << "Enter bookid, title & price";
    cin >> bookid;
    cin.ignore();
    cin.getline(title, 19);
    cin >> price;
}

void showBookData()
{
    cout << "\n" << bookid << " " << title
                                << " " << price;
}

int storeBook();
void viewAllBook();
void searchBook(char *t);
void deleteBook(char *t);
void updateBook(char *t);

};
```

```
int Book::storeBook()
{
    ofstream fout;  // fout is object
    getBookData();
    if (bookid==0 && price==0)
    {
        cout << "book data is
                    not initialized";
        return 0;
    }
    else
    {
        fout.open("File.dat", ios::app|
                              ios::binary);
        fout.write((char*)this, sizeof(*this));
        fout.close();
        return 1;
    }
}
```

```
void Book :: viewAllBook()
{   ifstream fin;
    fin.open("file.dat", ios::in | ios::binary);
    if(!fin)
    {  cout << "File not found";
    }
    else
    {  fin.read((char*)this, sizeof(*this));
       while(!fin.eof())
       {   showBookData();
           fin.read((char*)this, sizeof(*this));
       }
       fin.close();
    }
}
```

```
void Book :: searchBook (char *t)
{  int count = 0;
   ifstream fin;
   fin.open ("file.dat", ios :: in |
                          ios :: binary);
   if (!fin)
     { cout << "\n file not found";
     }
   else
   { fin.read ((char *) this, sizeof (*this));
     while (!fin.eof())
     { if (!strcmp (t, title))
           { showBookData();
             count ++;
           }
       fin.read ((char*) this,
                          sizeof (*this));
     }
```

```
    if (count == 0)
    { cout << "\n record not found";
    }
  }
    fin.close();
}
void Book :: deleteBook (char *t)
{
    ifstream fin;
    ofstream fout;
    fin.open ("file.dat", ios::in | ios::binary);
    if (!fin)
    { cout << "\n file not found";
    }
    else
    {
      fout.open ("tempfile.dat", ios::out |
                                  ios::binary);
      fin.read ((char*)this, sizeof (*this));
```

```
while (!fin.eof())
{
    if (strcmp(title,t))
    {
        fout.write((char*)this, sizeof(*this));
    }
    fin.read((char*)this, sizeof(*this));
}
    fin.close();
    fout.close();
    remove("File.dat");
    rename("tempfile.dat", "File.dat");
}
}

void Book::updateBook(char *t)
{
    fstream file;
```

⇒

```
file.open("file.dat", pos::in | pos::out |
                              ios::ate | ios::binary);
file.seekg(0);
file.read((char*)this, sizeof(*this));
while (!file.eof())
{
    if (!strcmp(t,title))
    {
        getBookData();
        file.seekp(file.tellp() - sizeof(*this));
        file.write((char*)this; sizeof(*this));
    }
    file.read((char*)this, sizeof(*this));
}
}
```

(PTO)

```
int menu()
{
    int choice;
    cout << "\n Book management";
    cout << "\n 1. insert book record";
    cout << "\n 2. view all record";
    cout << "\n 3. search book record";
    cout << "\n 4. Delete book record";
    cout << "\n 5. update book record";
    cout << "\n 6. Exit";
    cout << "\n Enter your choice";
    cin >> choice;
    return choice;
}
```

```
int main()
{   system ("clear");
    Book b1;
    char title[20];
    while(1)
    {  system ("clear");
       Switch (menu())
       {
          case 1 :
          b1.storeBook();
          cout << "\n Record Inserted";
          break;

          case 2:
          b1.viewAllBook();
          break;
```

```
case 3:
cout << "\n Enter title of book to search";
cin.ignore();
cin.getline(title, 19);
b1.deleteBook(title);
b1.SearchBook(title);
break;

case 4:
cout << "\n Enter the book title to
                delete record";
cin.ignore();
cin.getline(title, 19);
b1.deleteBook(title);
break;
```

```
case 5:
cout << "Enter book title to update";
cin.ignore();
cin.getline(title, 19);
b1.updateBook(title);
break;

case 6:
cout << "\n Thankyou for using
        this application";
exit(0);
default:
cout << "\n invalid choice";
}
}

return 0;
}
```

some imp operations and functions →
(Also used in Project)

- Pipe Command (|) →

This command is used to execute the multiple statements →

eg- File.open ("File.dat", ios::in | ios::out
ios::app | ios::binary

It means open file in → in, out, app and binary mode simultaneously.

- getline (); →

Used to input a full line in cpp work like gets() in C.

Syntax →

cin.getline (title, 20);

title → variable in which string to be stored

20 → Size of the char array or string

- cin.ignore();→
- work just like getch() in the C language.
- This is used to be clear the buffer memory in programme.
- when we input a string after a int (or sometimes string after string) or can or char after int (or char after the char) we have to use it.
- since, int usese whole byte memory and nothing left for char/string.
- file.dat →

  obj type data stored in '.dat' extention and file is to be opened in binary mode also to store object type data.
- char *ptr→
  - char pointer is used to store the address of string.
  - By using get line we can input string with spaces eg- "my name is"

- If we send char array to function it will stop on space (only send "My")
- So we send full address of string to the function to access the full string for which char pointer is used. eg → char * name;

- rename () →
- used to rename a file
- include <stdio.h> to use / avail it.
- syntax →

rename ("file.dat", "temp.dat");

file.date को ⟶ temp.dat बनाओ

- remove () →
- used to remove / delete file.
- Syntax → remove ("file.dat");
- include <stdio.h> to avail it

- file.seekg() →

  seek the file pointer to the given position (as parameter)

- file.tellg() →

  Tells the position of the file pointer to where it is at present.

- exit(0) →

- exit the programme

- use <stdlib.h> to avail it