# C++

Date ___/___/___

## Friend Function

Not a member function of class to which it is friend
Declared with friend keyword
Must be defined Outside the class to which it is friend
Can access private members to which it is friend
Cannot access members of class directly
No caller object as it is not a member function
We donot have to specify Membership label like
void Class_name :: func() X
c1.fun() X like this we cannot do

Code

```
class Complex {
  private : int a,b;
  public :
  void setData(int x, int y)
  { a=x; b=y; }
  void showData() { cout << " a,b" << a << b << endl;
  friend void fun (Complex);
};
void fun (Complex c) // we pass an object
{ cout << " Sum is " << c.a+c.b << endl; }
void main ()
{ Complex c1, c2, c3, c4;
  c1.setData (4,5)
  fun(c1);
  return 0;
}
```

Friend func^n can become friend to more than one class
We can declare friend fun^n anywhere, public private it does not matter
When we want access private members of different classes and do some computation we use friend function.

Class B; // formal Decloration [for compulation]

Date / /

Code: A member funⁿ can access pvt members of its own class, But it cannot access members(pvt) of othe class so we use FRIEND function.

```
class A {
  int a;
  Public: friend void fun(A, B);
};

class B {
  int b;
  Public: friend void fun(A, B);
};
  void fun( A o1, B o2)
  { cout << "Sum is" << o1.a + o2.b; }
void main() {
  A obj1;
  B obj2;
  fun(obj1, obj2);
}
```

# Overloading of operators as a FRIEND Funⁿ

★ operator Overloading without friend

```
class Complex {
  int a, b;
  Public: void setData( int x, int y) { a=x; b=y; }
  void showData() { cout<<a<<" "<<"b"<<endl;
  Complex operator+ ( Complex c )
    { Complex temp;
      temp.a = a + c.a;
      temp.b = b + c.b;
      return temp;
    }
```

c1 c2

This is member function.

```
void main ()
  { Complex c1, c2, c3;
    c1.setData(3,4); c2.setData(5,6);
    c3 = c1 + c2; // c1.operator+(c2)
```

passed as parameter

This is caller object

vijeta

Date / /

★ With friend func$^n$ Operator Overloading

```
Class Complex {
    int a,b;
    void setData( int x, int y) { ~~x=c~~ a=x; b=y; }
    void showData () {        }
    friend Complex operator + ( Complex c1, Complex c2 );
// friend fun$^n$ is never defined inside the class
}


Complex operator + ( Complex c1, Complex c2 )
{   Complex temp;
    temp.a = c1.a + c2.a;
    temp.b = c1.b + c2.b;
    return temp;
}
void main() {
    Complex c1, c2, c3;
    c1.setData(3,4); c2.setData(5,6);
    c3 = c1+c2; // here both c1 & c2 are parameters
}   // + is a friend function
```

★ Unary Operator Overloading with friend function

```
class Complex { int a,b;
Public: void setData (--) {--};
void showData () {---};
Complex operator - () {
    Complex temp;      belongs to c1 object
    temp.a = -a;
    temp.b = -b;
};
void main () { Complex c1, c2;
    c1.setData( 3,4);      caller object
    c2 = -c1; // c1.operator - ();
}
```

# As friend function.

Date / /

```
class Complex {
int a,b;
Public : void setData( int x, int y ) { a=x ; b=y ; }
friend Complex operator - ( Complex ~~a~~ ) ;
}// friend return_type operator [+,-,/,*,--] ( parameter Datatype);
~~vo~~ Complex operator - ( Complex o1 ) {
    Complex temp ;
    temp.a = - o1.a ;
    temp.b = - o1.b ;
    return temp ;
}

void main () {
    Complex c1, c2 ;
    c1.setData( 4,5);
    c2 = - c1 ; // here c1 is not caller object it is a
} parameter passed to operator - () function.
```

# Insertion Extraction operator overloading (Input / Output)

```
Complex c1 ;
cout << "Enter a Complex number " ;
cin >> c1 ;
```

X this won't work But we can achieve it by operator overloading

```
cout << c1
```

X this also won't work

// we make friend func^n in those classes from where we want to access there private members.

```
class Complex { int a,b;
    public: void setData( int x, int y ) { a=x ; b=y ; }
    friend ostream & operator << ( ostream & , Complex ) ;
} friend istream & operator >> ( istream & , Complex & ) ;
    ~~void~~
};
```

Date / /

return type for cout is declared in ostream

```
ostream & operator << ( ostream & dout , Complex c)
{
    cout << c.a << c.b ; return ( dout ) ;
}
```

giv any name.

```
istream & operator >> ( ~~ostream & cout~~ istream & cin , Complex & c)
{
    cin >> c.a >> c.b ; ~~return ( cin~~ return (cin) ;
}
```

// object for cout & cin cannot be made so we use refrence.

```
void main () {
    Complex c1 ; cin>> c1 ; // operator >> ( cin , c1) ;
    cout << c1 ; // operator << (cout , c1) ;
}
```

// without friend function we cannot overload >>, <<
as we cannot make objects of cout & cin & also
we cannot access those classes.

★ Member function of one class can become friend to another class

```
class A {
    public : void fun() { ... } ;
};

class B {
    friend void fun() ;
}
void fun() { } ;
```

This fun does not know whic fun to access, from A class or the one declared outside.

// so to resolve this we need to add scope resolution operator ( :: )

friend void A :: fun () ; now this will acess fun function from A class & not from outside

// If we want to make member functions of A as friend in B then syntax

```
class B {
    friend class A;
}
```

# Inheritance

Date / /

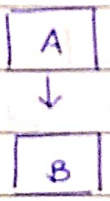
**Class** : is used to describe properties + behaviour of object.
क्या info  क्या opn performed

**Encapsulation** : oop ek entity se related sari info ek hi object me rakhna, like normal car + Sports car ke liye alag alag classes nhi banani

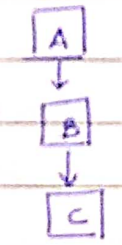
**Code** :

```
class Base {

}

Class Derived : Visibility mode (Public, Protected, Private) Base
{

}
```

Single Inheritance

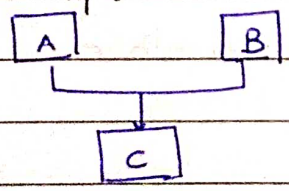



Multiple



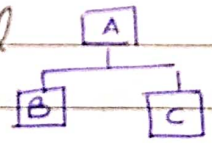

furniture
dining table
Round Dining table

Multiple Inheritance




```
Class A { };
Class B {};
Class C : public A , public B { };
```

Hierarchial

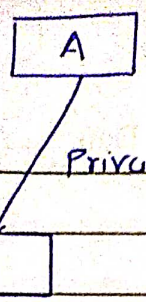



| Availibility + | Accessibility |
|---|---|
| When object made + even if it is not allowed to access pvt members then also pvt mem exist in its memory | only Public members are accessibly |

```
U1 {  }
U2 : pub U1 {  }
```

Protected → U1 cannot access its protected mem but U2 can access

Date __/__/__




→ B can access pvt & protected mem of A but by applying specifier whatever objects of B are made for them Inherited Pvt, Protect mem of A will seem as Private mem only.

B b1, b2;

Visibility mode hume ye bolata ki Derived class ke objects ke liye Parent class ke members jo inherit kiye vo kya hai unke liye – Pvt, Pub, Pro

Public Inheritance me Protected mem, derived me bhi Protected Banta

<u>Code</u>

```
class A {
    private : int a;
    protected : void setValue (int k) { a = k; }
};
class B : public A {
  public :
      void setData (int x) { setValue (x); }
};
void main () {
  B obj;
  obj. setValue(4);
}
```

→ This will give error as it will act as protected member for obj so cannot access even if public specifier is applied

<u>Inheritance</u> → is a

Association → relaⁿ btw two classes

→ Aggregation, Composition, Inheritance → BMW is a Car

Student is a faculty X

faculty has a student X

faculty teaches student

vijeta

is a relationship me Public specifier hi use Karte hai

Date / /

Why?

```
class Car
{ int gear;
  Public:
  void increamentGear()
  { if(gear <5)
        gear++;
  }
};

class SportsCar : public Car
{
}
```

Put ya protected me parent ki public classes put, pro ban jaati hai isiliye hum inheritance me Public specifier use karte hai

→ <u>When to use</u> <u>put</u> & <u>Protecd</u>

```
class Array
{ put: int a[10];
  public: void insert (int i, int val) { a[index] = val; }
};

class Stack : public Array
{ int top;
  public: void push (int val) // after making private push will be
  { insert (top, val); } able to call insert because insert fun^n
};
```

× → private

has become private Member of Stack

also: Child Class can access BUT! Child object cannot access insert func^n.

```
int main () {
  Stack s1;
  s1.push(34);
  s1.insert(2, 56); // violating stack concept as num 56 is
  directly going to index 2 & not 1, But this should
  not happen. So now we need protected, private specifier.
}
```

// vijeta //

Date / /

# Constructor in Inheritance.

→ Child class constructor calls Parent class constructor.

→ Parent to child [Execution]

→ Child to Parent [Calling of Constructor]

**Code**

```
Class A {
  int a;
  public :   A (int k)  // constructor
             {  a = k ; }
};
Class B : Public A {
   Public :
    B() : A()x  // Defalt constructor of B will call
   { }          defalt const of A BUT A does not have
}               defalt Constructor. But as we have parameterized
void main() {   constructor so we can call in
  B obj ;       B() : A(5)  way ✓
```

→ If a parameterized constructor is made in Parent class then we strictly have to make constructor in child class to call that parent class parameterised constructor.

```
Class A { int a ;
Public: A( int k) { a = k ; }
};
Class B : Public A {
   int b ;
   public :  B (int x , int y ) : A(x)
             { b = y ; }
};
void main () { B obj (2,3) ; }
```

goes to a

goes to b

Date / /

## DESTRUCTOR in Inheritance

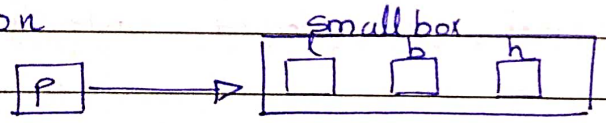
→ Child to Parent [ Execution ]
→ Child to Parent [ Calling ]
→ Compiler itself calls destructor of Parent

## Object Pointer

→ stores object address
→ Box *p , smallbox ;
p = &smallbox
p→setDimension ( 12 , 10,5 );
→ we use arrow to access functions thru pointers
p → showDimension ();

## This pointer

→ is a [local] object pointer in [every instance]/ object [member] [function] containing address of caller object
function {
this ← I am a local pointer + has access to current object, I am accessible inside ~~obje~~ instance mem function.
}
→ this pointer cannot be Modified
→ [static] void setDimension() is not an instance member function




→ It is used to [refer caller object] in Member Function
this → l = l ;

## New , Delete keyword

→ SMA : Static Memory Allocation
DMA : Dynamic memory allocation
→ SMA → int a; float y;
Compile time memory alloca$^n$ : At compile time compiler knows what amount of memory will be given to a variable
we cannot destroy SMA Variable before its destruction

// vijeta //

Date / /

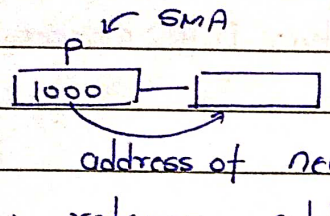
→ At compile time, what will be lifetime of var & how much amount of mem is given to a block is fixed in SMA

DMA → At runtime, how much memory needed is decided

int *p = new int; C++ malloc calloc → C

float *q = new float;

Complex * ptr = new Complex;




new returns address

→ float * q = new float [5];

→ int x; cin>>x; int *p = new int [x];

→ delete p; // used for releasing DMA variables memory as DMA variable memory will remain througout the program until program is terminated.

→ delete [] p;

→ p jisko point kar raha hai, jis bhi address block ko vo delete ho rha hai

Method Overriding

```
class A {
public: void f1() {};
        void f2() { };
};
class B : public A
{ void f1() { }   // Method Overriding
  void f2 (int a) { }   // Method Hiding
}
void main () {  B obj;
                obj.f1(); // B f1() runs
                obj.f2() // It goes to B class checks if this f2() is present as it is, then it will not go to Parent class, BUT as f2(int x) needs var input so Error.
```

Early Binding

→ Its compilers duty which f1() should run.

→ Compiler sees the object type of f1() & calls that f1() so it checks if f1() is present in B class or not

vijeta

Date / /

obj.f2(4) // B Class ka f2() chalega
obj.f2() // error

function Overloading
→ In one class we have two or many functions with same name & different parameters

→ If two functions with same name is present two Diff classes then that's not function overloading

overriding → same name different implementation

```
class Car {
  public:
  void shiftGear() {  }

}
class SportsCar : public Car
 { public:
     void shiftGear() {  }
 }
void main() {
    SportsCar obj;
    obj.shiftGear(); // SportsCar ShiftGear() called
    ~~obj~~
}
```

→ If we want to change the implementation of a service in Parent class we use overriding in Child class
Virtual Function.
→ Base class pointer can point to object of any of it descendents class
→ But Converse not true.

Date ___/___/___

```
class A {
Public : void f1();
};

class B : public A
{ public :
    void f1();    // function overriding
}

void main () {
  A * p , O1 ;
  B  O2 ;
  p = & O2 ;
  O2 . f1() // Due to Early Binding [B class] f1 is called.
  p → f1() // This calls f1() of [A class] coz it sees the data type of
                p & it is A but p contains address of B type object
                due to Early Binding.
```

Early Binding : It is a concept related to overriding, ~~which~~ compiler decides which class func^n to call by seeing the Datatype of caller object.

// Solution : is to stop Early Binding , Do Late Binding i.e at Runtime so we write virtual Keyword in Parent class

virtual void f1() in class A

helps in Late Binding so at runtime pointer's Datatype is not seen instead the content ka type is seen

so p ⇒ O2 ; p → f1() , after applying virtual keyword f1() of B class is called pointer's contents Datatype is seen

Virtual functions

```
class A
{ * vptr
  public :
    void f1() { }
    virtual void f2() { }
    virtual void f3() { }
    virtual void f4() { }
```

→ It points to vtable, Each class has its own vtable.

→ this ptr is made in class which has virtual functions. It is only made in parent class & not derived class

vtable

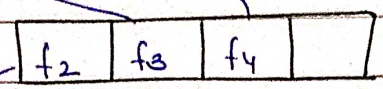
| | | | | |
|---|---|---|---|---|
| f2 | f3 | f4 | | |

This vptr makes a static array which is of pointers type & it contains virtual function Address

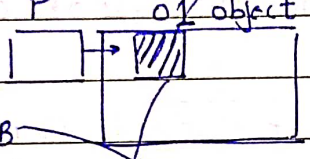
Inherited from A class

Date / /

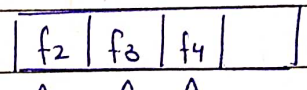
```
class B : Public A        this will also have vtable
{ public :
    void f1() { }
    void f2() { }
    void f4(int a) { } ;  // Method Hiding
};

void main()
{
    A *p, o1;
    p = &o1;
    p -> f1(); // Early B
    p -> f2(); // Late B
    p -> f3(); // Late B
    p -> f4(); // Late B
    p -> f4(5); // Early B / normal
}
```




Contains address of vtable of A object




as




* vptr

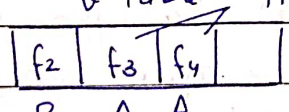
Class A f2
f3
f4

this ~~does~~ has a ~~Data param~~ argument but parent does not has any f4(int a) type function so it will give Error

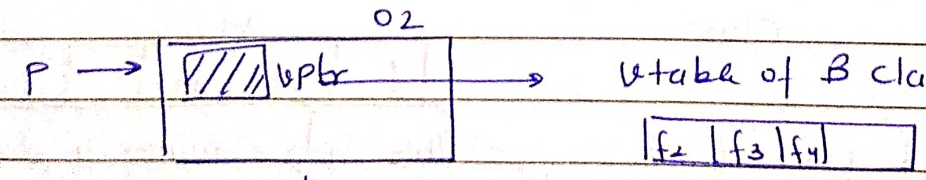
```
{ A *p, o1;
  B o2;
  p = &o2;
  p -> f1() // EB , f1() of A
  p -> f2() // LB , f2() of B
  p -> f3() // LB , f3() of A as B does not contain f3()
  p -> f4(); // LB , f4 of A as B does not have f4()
  p -> f4(5); // EB , f4(5) of B as it is present in B.
```

p → vptr of o2 object that is B class

vtable




these two were inherited from class A




EB → ~~takes func^n of that class that~~ b matches Datatype of Pointer / object & calls function from that class.

LB → ~~makes~~ Datatype of contents of pointer & calls that Class function

// vijeta //

Date / /

# Pure virtual function

→ a virtual function which has no defination is called pure virtual function.

Void fun() = 0; // Do nothing function.

```
class Person {
  public :
      Void fun() = 0;
}
```

```
class student : Public Person
{ Public :
  void fun () {  }
}
```

→ It is illegal to make object of Person & call fun() because it is not defined yet

→ So if in a class we make a virtual pure function we cannot make that class's object

→ ~~So this class is Abstract Class~~

→ We can make Child class of Person class & if we create pointer of Child class then also it will be able to access fun() from Parent

→ So it is must that whatever class we make pure virtual function, it is necessary to define that func$^n$ in Child class i.e do overriding of pure virtual function.

→ But if we make Person * ptr;

Student s; $\boxed{\text{ptr} = \&s;}$

$\boxed{\text{ptr.fun();}}$ → Due to Early Binding it will call fun() from Person class so this is also illegal.

So we add virtual key word

```
class Person {
Public :
  virtual void fun() = 0;
}
```

pure virtual function.

```
class Student : public Person
{ public :
  void fun () {  }
}
```

```
main() { Person * ptr;  Student s;
  ptr = &s;   ptr.fun();
```

// Calls Student's fun()

Date / /

# ABSTRACT CLASS

→ Class which contains atleast one pure virtual function called Abstract Class

→ pure virtual function → func^n that has no body.

```
virtual void fun () = 0;
```

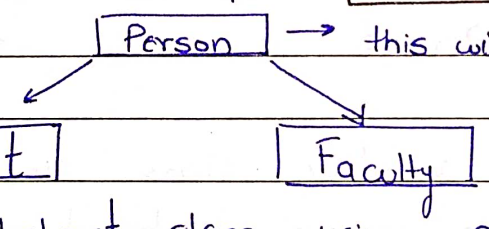
→ Cannot Make objects of Abstract Class

→ It is necessary to define virtual pure function inside derived class.

→ It is must to make Child class of Abstract Class then only we will be make objects & use methods.

→ Why to Make Abstract Class ?

→ Abstract Class is used for Generalization




→ this will contain common methods.

→ Make Person Abstract class using pure virtual function.

## Static functions

→ These function does not belong to object, they belong to class

→ we ~~define it~~ + declare it inside class & define outside.

```
static void func();

void A :: func() { }
```

→ They do not have caller object.

## Initializer list :

```
class Dummy {
  private : int a, b;
  Public : Dummy () : a(5), b(6) { }
```

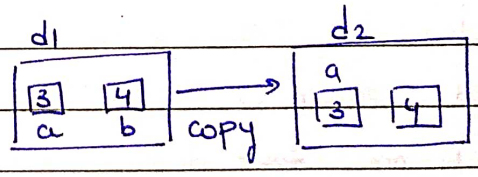
// vijeta //

Date ___/___/___

```
class Dummy
{ pvt :
    int a,b;
    const int x; // when const declared it should be defined also
public :          // we cannot initialize inside class to a const
  Dummy () : [x (5)] → const variable is initialized using list
{                      initializers.
    x = 5; // X this cannot be done as it is const
    // So here is the need for LIST INITIALISERS
} };
Class Dummy {
  Pvt : int &y; → refrence variable, these also should be
                  initialized when declared.
  Public :
    Dummy (int &n) : y(n) { }
};
void main () {
    int m = 6;        → passing refrence of m not value.
    Dummy d1(m);
}
```

→ Initializers are important for non-static const data members  const int x;

→ For Refrence members  int &y;




Deep & Shallow Copy

```
Class Dummy {
  private : int a,b;
  public : void setData (int x, int y) { a = x; b = y; }
    void showData() { }
}
void main() {
    Dummy d1;   d1.setData(3,4);
    Dummy d2 = d1;   d2.showData();
}
```

Date / /

How copy created?

→ Copy constructor is called

→ Defalt assignment operator.

→ an object when initialized & it does not have any constructor, then two constructor are made
1) Default Constructor 2) Copy Constructor

→ Dummy d2 = d1; // copy constructor will be called
~~d2~~ =

→ Dumy d2;
d2 = d1; → // copy constructor not called
// here assignment operator is called

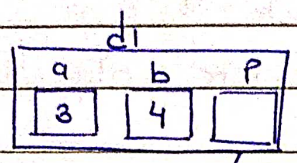
→
```
Class Dummy {
  pvt: int a,b;
  public:
  void setData (int x, int y) { a = x; b = y; }
  Dummy ( Dummy &d)  → refrencing to (d1)
  {                  → relate to (d2)
    a = d.a;  b = d.b
  }
void main ()
{ Dummy ~~d2~~ d1;
  d1.setData( 3,4);
  Dummy d2 = d1; // copy Constructor called
  d2. showData();
}
```

Copy Constructor

Shallow Copy: Creating Copy of object by copying data of all member variables as it is.

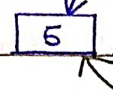
Date __/__/__

## Deep Copy :



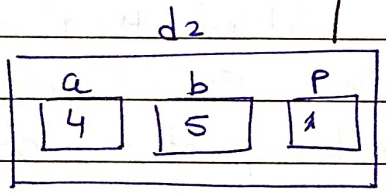

This object has resources residing outside the object but handled by object only as p hold address of this block

```
Class Dummy {
  put: int a,b;
  int *p;
public:
  Dummy () {
    p = new int;
}
```




int type block

```
  ~~Dummy~~ void SetData (int x, int y, int z)
    { a=x; b=y; *p = z; }
Dummy (Dummy &d) → Copy Constructor
  { a = d.a;
    b = d.b
    p = d.p
  }
void main () {
  Dummy d1;
  d1.setData( 3,4,5);
  Dummy d2 = d1;
}
```




This p also pointing to the block created

so if we change value of p in d2 then value of d1 will also be changed, This is a problem.

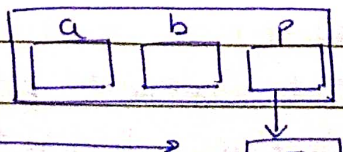
This is Shallow Copy.

→ Member variable pointer i.e (p) is pointing to same resource for d2 & d1, so it is necessary that for d2 there should be another block created.

→ How to achieve? DEEP COPY → Copy Constructor should be made by Us

So inside copy Constructor

```
  Dummy ( Dummy &d)
  { a = d.a, b = d.b,
    p = new int;
    *p = *(d.p);
  }
```







new block created

(d.p) → address

*(d.p) → value at address

refrencing d1 object

vijeta

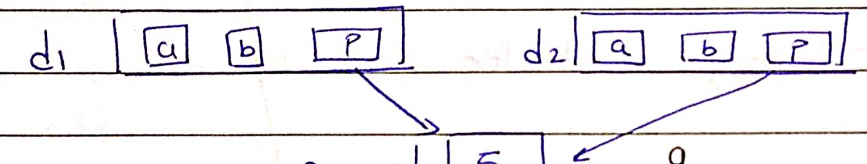
~ Dummy () Destructor

Date / /

{ delete p; delete the block that p points to before object gets destroyed
}
};

main()
{ Dummy d1;
d1.setData( 3, 4, 5);
Dummy d2 = d1; // Copy Constructor called.
}

→ If we made Shallow copy the p of d1 & p of d2 would point to same block.




In Shallow Copy

& if d2 is deleted then this memory block [5] also gets deleted, So if we try to access p from d1 it would point to nothing, would give garbage Value

So [p of d1] would be called DANGLING POINTER point that points to nothing

→ Deep Copy can be achieved thru Copy Constructor. by making new memory block for object

p = new int;
*p = *(d.p)

# Primitive type to Class type can be achieved thru parameterised Constructor.

class Complex { put: int a, b;
public: Complex() {}
Complex (int k) { a=k; b=0; }
void setData (int x, int y) { a=x; b=y; }
} void main() { Complex c1; int x = 5;
c1 = x // c1.Complex(x)
}

// vijeta //