OOPs

The only difference between class and structure :-

→ The members of structures are by default public.

→ The members of class are by default private.

Class

A class in C++ is building block that leads to object oriented programming. It is a user-define data type, which holds its own data member and member functions, which can be accessed by creating an instance of class. A C++ class is like a blueprint for an object.

Class is a description of an object.

```
Class className
{   Access specifier ;  // Private, public, protected
    Data Members ;      // Variable to be used
    Member Functions() { } ; // Methods to access data Member
};
```

→ Incase of structure, memory is allocated on stack, but for class memory is allocated on Heap.

→ Structure does not support Inheritance, but class supports Inheritance.

→ Structure may have only parameterized constructor but class have all types of constructor & destructor.

* **Object :-** An object is an instance of class. When a class is defined no memory is allocated but when it is instantiated (i.e. an object is created) memory is allocated.

ClassName objectName;

L-7 Part - 1

ex:
```
#include <bits/stdc++.h> using namespace std;
Class complexNum
{
private:
    int a, b;
public:
    void set_data (int x, int y)
    { a=x; b=y; }
    void show_data ()
    { cout<<"\n a = "<<a<<" b = "<<b; }
};
int main() {
complexNum c1;
c1.set_data(3, 4);
c1.show_data();
}
```

When we create an object of class, memory is allocated on Heap.

Object

* Outside declaration of Member f^n

```
Class ComplexNum
{ private:
    int a, b;
public:
    void set_data(int x, int y);
    void show_data();
{ cout<<"\n A="<< " B="<<b<<endl;}
};
```

Membership label

```
void complexNum :: set_data (int x; int y)
{ a=x; b=y;
```

L-7
Part-2.3

Camlin Page 05
Date / /

Classes & object in C++

Ex 2: Add two complex Number using class, object & member functions of class.

```
#Include<bits/stdc++.h>
using namespace std;
class complexNum
{   private:
      int a, b; -> Data members or Instance member variable
    public:
      void set_data (int x, int y) -> Instance member fn
      {  a = x;
         b = y;
      }
      void show_data ( ) Instance member fn
      {
      cout<<a<<"+i"<<b<<endl;
      }

      ComplexNum add (ComplexNum c)
      {
        ComplexNum temp;
        temp.a = a + c.a;
        temp.b = b + c.b;
        return temp;
      }
};
int main(){
    ComplexNum c1, c2, c3; - objects->
    c1.set_data (3, 4);
    c2.set_data (5, 6);
    c3 = c1 + c2;   (Wrong error)
    c3 = c1.add (c2);
    c3.show_data ();
    return 0;
}
```

NOTE:- 1. Without any object no member fn can perform any operation.
2. State of objects can be only change by its instance member fn.

L-8 Static Members

Part - 1

* Static local Variables

Static local variable is a variable whose initialization starts with keyword static is called static local variables.

For. ex

static int x;
Keyword

They are by default initialized with zero

Difference between :-

Static int x;	int y
1. By default initialized with zero.	1. By default initialized with garbage Value.
2. Qualify with static Keyword	2. No need of any Keyword
3. Memory allocated when we declare.	3. Memory will destroyed when block Scope will finish

* Static Member Variable

- Declared inside the class body.
- Also known as class Member variable.
- They must be define outside the class.
- Static member variable does not belong to any object but to the whole class.
- There will be one copy of static member variable for the whole class.
- Any object can use the same copy of class variable.
-

Static Member Variable not changes for different object it will be same for all objects of a class.

For ex

```
#Include<bits/stdc++.h>
using namespace std;

class Account
{
  private:
    int balance; //--> Instance member Variable
    static float roi; //--> Static Member variable /class variable.
                      --> Declared Inside class.
  public:
    void setBalance(int b)
    {
      balance = b;
    }
};
float Account :: roi = 3.5f ; Defined outside the class.
int main(){
{   Account a1, a2;
    a1.setBalance(22000);
    a2.setBalance(112000);
    return 0;
}
```

L-8

Part-2

Static Members

Camlin Page 08
Date / /

* Static Member Functions :-

- They are qualified with the keyword Static.
- They are also called class member function
- They can be invoked (Access) with or without object.
- They can only access static member of the class.

Need :- of static Member f^n

To access the Member f^n without any object we make f^n static Member f^n

for ex.

```
class Account
{   private:
    int balance;  // Instance Member Variable
    static float roi;  // Static Member Variable
public:
    void setBalance (int b)
    {  balance = b; }

static void setRoi (float r) // Static Member f^n
{  roi = r;  // without any object we can call static member
}
static void showRoi ()
{
    cout << " Rate of interest :: " << roi << endl;
}
};   float Account :: roi = 3.5f;
int main() {
{  Account a1, a2;
   a1.setRoi (4.5f);
   a1.showRoi ();
   Account :: setRoi (6.5f);
   a1.showRoi ();
   return 0;
```

Declaration →

Static

Instance

→ Defination of static member variable

a1.setRoi (4.5f); - Accessing static Member f^n with object

Account :: setRoi (6.5f); → Accessing static Member f^n without object.

L-9

Part-1

Constructor in C++

- Constructor is a member function of class.
- The name of the constructor is same as the name of class.
- Constructor has no return type, so can't use return keyword.
- It must be an instance member function that is, it can never be static.

When object of class is created then constructor called automatically.
that means, Constructor is implicitly invoked when an object is created.

Use :- Constructor is used to solve problem of initialization.

```
#Include<bits/stdc++.h>
using namespace std;
Class Complex
{
private:
int a, b;
public:
Complex()   // Default Constructor
{
cout << " Constructor called :: " << endl;
}
};
int main()
{
complex c1, c2, c3;
return 0;
}
```

if we creates n objects of class then n time Constructor will call.

output :-
Constructor called
Constructor called
Constructor called
} 3 Objects

L-9

Part-2

Object बनाता तो Constructor होता ही होता, Default Constructor Constructor in C++ implicit.

Types of constructor:-

- Default Constructor. → No parameter. (Not Necessary to write it, when No constructor in class.)
- Parameterized Constructor. → Contains parameter or Arguments.

ex

```
class Complex
{   private:
    int a, b;

Parameterize Constructor
    Complex (int x, int y)
    {
        a = x;   b = y;
    }

    Void show_Data ()
    {
        cout << a << " " << b << endl;
    }

Parameterized Constructor →
    Complex (int z)
    {
        a = z+z;
        b = z+z+z;
    }

    Complex ()  Default Constructor
    {
        cout << " Default Constructor " << endl;
    }
};

    int main() {
Complex c1(3,4), c2(5), c3 = Complex(5,6), c4;
    c1.show_data();
    c2.show_data();
    c3.show_data();

    return 0;
}
```

Constructor Overloading:
When same class having more than one constructor with different argument than this situation called Constructor Overloading.

NOTE: When we are making object without passing argument and class have both parameterized & default constructor and then it is necessary to write Default Constructor

L-9

Part-3

Camlin Page 1
Date / /

Imp point

→ यदि हमने class में कोई भी constructor बना लिया तो, Compiler Default Constructor नहीं बनाएगा।

* Copy Constructor

NOTE: Incase of copy constructor the reference of any object will pass as argument at the time of definition.

```
Class complex
{
  private:
    int a,b;
  public:
    Complex (int x, int y) // Parameterized Constructor
    {
      a=x; b=y;
    }
    Complex (int z)
    {a=z;}
    Complex () {} // Default Constructor

    Complex( complex &c) // → copy constructor
    {  a = c.a;
       b = c.b;
    }
    void show(){
      cout<< a <<" "<< b <<endl;
    }
};
int main(){
  Complex c1(3,4), c2, c3(5);
  Complex c4(c1); → In this situation
  c1.show();      Compiler automatic makes
  c4.show();      copy constructor.
  return 0;
}
```

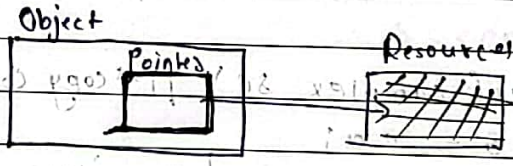
Q compiler makes copy constructor when we pass a reference as a object in another object.

Destructor in C++

Destructor:- Destructor is an instance member function of a class.

- Initialization :- ~ ClassName() { }.
- Destructor can never be static.
- Destructor has no return type.
- Destructor does not takes argument (No overloading is possible).
- It is invoked (Call) implicitly when object going to destroy.

Why Destructor:- It should be defined to release resources allocated to an object.



if object will decroy then pointer will also destroy so we can't access or realease Resources, thats why we use constructor to realease & access resources.

So Destructor is called when object of any class is going to destroy, or we can say that Destructor is last function which call and after that object will destroye.

```
Class Complex { private: int a,b;
  public:
  ~ Complex(){ cout<<"Destructor:: "<<endl;}
};  void fun() { complex obj; }
int main(){
    fun();
    return 0;
}
```

→ in this fn object (obj) is going to destroy so the Destructor is called, if we not makes any destructor, then compiler makes it, implicitly.

Operator Overloading in C++.

* When an operator is overloaded with multiple jobs, it is known as operator overloading.

* It is way to implement compile time polymorphism.

ex
```
class Complex {
private:
    int a, b;
public:
    void set_data(int x, int y)
    {
        a = x;
        b = y;
    }
    void show() {
        cout << a << "+i" << b << endl;
    }
    Complex operator+ (Complex c)
    {
        Complex temp;
        temp.a = a + c.a;
        temp.b = b + c.b;
        return temp;
    }
};
int main() {
    Complex c1, c2, c3;
    c1.set_data(3, 4);
    c2.set_data(5, 6);
    c3 = c1 + c2;   or   c3 = c1.operator+(c2);
    c3.show();
    return 0;
}
```

imp
You can not overload sizeof and ?: operator

- Any symbol can be used as function name :-
  1. if it is valid operator in c lang.
  2. It is preceded by operator keyword.

NOTE:
All the operators that exist in c++, we can overload them, with the keyword operator(sign).
{ +, -, *, /, ++, -- etc }

Here + operator, operator is non primitive complex types variables or objects in c++

(+) operator operates primitive variables or member so, it have two jobs, so one overload the other jobs, that why we call it operator overloading. imp

Part-2

Operator Overloading in C++

- overloading of unary operator

Incause of Binary operator, the left operand will be caller object.

- But Incause of unary operator there is one object i.e. caller object.

ex:

```
class Complex {
    private:
        int a, b;
    public:
        void SetData (int x, int y)
        {
            a = x;
            b = y;
        }
        void ShowData() {
            cout<<"a<<" "<<b<<endl;
        }
        Complex operator-() {          // No argument
            Complex temp;
            temp.a = -a;
            temp.b = -b;
            return temp;
        }
};
int main() {
    Complex c1,c2;
    c1.setData(3,4);
    c2 = c1.operator-(); or c2 = -c1;   // unary operator, c1 -> operand
    c2.showData();
    return 0;
}
```

In cause of unary operator overloading No argument will bepass in operators function

L-12

Part-3

Camlin Page 15
Date / /

Operator Overloading in C++

* Overloading with pre & post Increment operator

```
class Integer
{
private:
int x;
public:
void SetData(int a)
{ x=a; }
void ShowData()
{
cout<<" x="<<x<<endl;
}
Integer operator++() // --> PreIncrement
{
Integer i;
i.x = ++x;
return i;
}
Integer operator++(int)
{
Integer i;
i.x = x++;
return i;
}
};
int main(){
Integer i1, i2;
i1.SetData(3);
i1.ShowData();
i2 = ++i1 // i2 = i1.operator++(); - PreIncrement
i1.ShowData();
i2.ShowData();
i2 = i1++; // Post Increment
i1.ShowData();
i2.ShowData();
return 0;
}
```

Increment operator Overloading ++ (Pre & post)

i2 = i1++; in this case first the value of object i1 is copy into i2, & then i1 increment, but incase of i2 = ++i1 (PreIncrement) the value of i1 will increase 1 after that i1 will copy into (i2).

o/p:- No

o/p:-
=> 3
=> 4
=> 4
=> 5
=> 4

Part-1

Friend Function in C++

Friend function:- Friend function is not a member of a class to which it is a friend.

* A friend function is declared in the class with friend keyword, but define outside the class.
* Friend function can also access private members of class (like member fn), but not access directly.
* It has no caller object.
* It should not be define with membership label.

Example:-

```
Class Complex
{   private:
    int a, b;
    public:
    void set_data(int x, int y)
    {  a=x;   b=y;  }
    void show_data()
    {
    cout<<"A= "<<a<<" B= "<<b<<endl;
    }
    friend void fun(complex);
};
void fun(Complex c)
{
    cout<<"Sum is: "<<(c.a+c.b)<<endl;
}
int main()
{
    Complex c1;
    int p, q;
    cin>>p>>q;
    c1.set_data(p,q);
    fun(c1);
    return 0;
}
```

Inside class declaration

object as parameter

outside Definition

→ Calling of friend fn with object par

Camlin Page 17
Date / /

Friend function in c++

NOTE: We can make one unique friend fn for multiple classes.
Or friend function can become friend to more than one classes.

ex :-

```
#include<bits/stdc++.h>
using namespace std;
class B; // forword declaration of class B
class A {
private: int a;
public:
void set_data(int x) {a=x;}
friend void fun(A,B); } friend fn
};
class B {
private: int b;
public: void set_data(int y)
{ b=y; }
friend void fun(A,B); → friend fn declaration
};
void fun(A o1, B o2) } friend fn defination
{
cout<<"sum is:"<<(o1.a+o2.b)<<endl;
}
int main() {
A obj1;
B obj2;
obj1.set_data(33);
obj2.set_data(22);
fun(obj1, obj2); } friend fn Calling
return 0;
}
```

NOTE:
We can declare a friend fn in class as private, public & protected, it does not impact.

Benefit of friend fn:-
if we want to use data members of different class together, then we can use with help of friend fn.

Part-3

Friend function in c++

* Operator overloading using friend fn

Ex

```
Class Complex {
private:
    int a,b;
public:
    void set_data( int x, int y) { a=x; b=y; }
    void show_data(){ cout << a << " " << b << endl; }
    friend Complex operator+ (Complex, Complex);
};
```

→ Friend fn in operator overloading

```
Complex operator+ (complex c, Complex d)
{   Complex temp;
    temp.a = c.a + d.a;
    temp.b = c.b + d.b; return temp;
}

int main(){
    Complex c1,c2,c3;
    c1.set_data(3,4);
    c2.set_data(5,6);
    c3 = c1+c2;          // c3 = operator+ (c1,c2);
    c3.show_data(); // O/P :- 8, 10
    return 0;
}
```

operator overloading

L-12
Post-4

Camlin Page 19
Date / /

Friend function in C++.

* Overloading of unary operator as friend fn.

```
Class Complex {
    private:
        int a, b;
    public:
        void Set_data (int x, int y)
        { a = x; b = y; }
        void show_data ()
        { cout << a << " " << b << endl; }
        friend Complex operator-(Complex);   // Declaration friend fn
};
Complex operator-(Complex c)   // Defination friend fn as operator overload
{
    Complex temp;
    temp.a = -c.a;
    temp.b = -c.b;
    return temp;
}
int main()
{
    Complex c1, c2;
    c1.Set_data(3,4);
    c2 = -c1;   or   c2 = operator-(c1);
    c2.show_data();
    return 0;
}
```

Operator Overloading

→ When we use friend fn in operator overloading we've to pass one extra argument in overloading fn or call.

L-12
Part-5

Friend Function in c++

* Overloading of Insertion & Extraction operator using friend

```
#Include<bits/stdc++.h>
using namespace std;

Class Complex
{  private:
     int a,b;
   public:
     void set_data( int x, int y)
     {  a=x;  b=y; }
     void show_data(){
       cout << a << " " << b <<endl;
     }
     friend ostream& operator<<(ostream&, Complex);
     friend istream& operator>>(istream&, Complex&);
};
ostream& operator<<( ostream &dout, Complex C )
{
  cout << C.a << "  " << C.b << endl; return (dout);
}
istream & operator>> (istream &din, Complex &C)
{
  cin >> C.a >> C.b; return (din);
}

int main(){
Complex c1;
cout<<" Enter a complex Number :: "<<endl;
cin >> c1;
cout<<" You Entered "<<endl;
cout << c1;
return 0;
}
```

Extraction operator

Insertion operator

Friend fn of Insertion Operator

Friend fn of Extraction Operator

Calling extraction friend f^n

Calling insertion friend f^n

Part-1

Inheritance in C++

Inheritance :- It is a process of inheriting properties and behaviours of existing class into a new class.

- Existing class :- Old class, Parent class, Base class.
- New class = child class = Derived class.

Syntax :-

```
class Base_class.
{

};

class Derived_class : Visibility Mode {private, public, protected} Base_class
{

};
```

L-13

Part-2

Camlin Page 23
Date / /

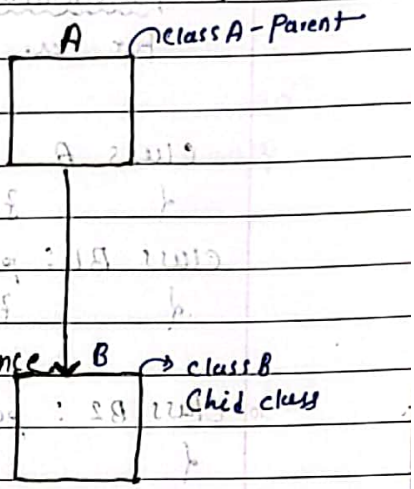
Inheritance in C++

* Types of Inheritance

* Single Inheritance :-

```
class A
{
};
class B : public A
{
};
```

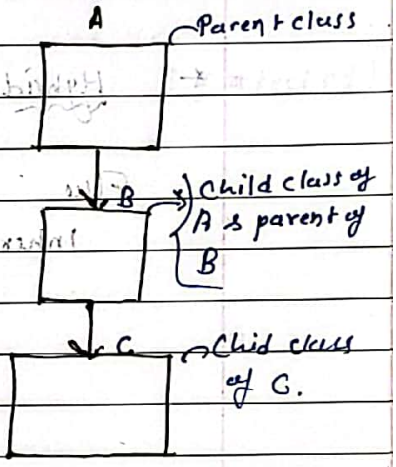
One parent
& One child class
∴ Single Inheritance



* Multilevel Inheritance :-

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

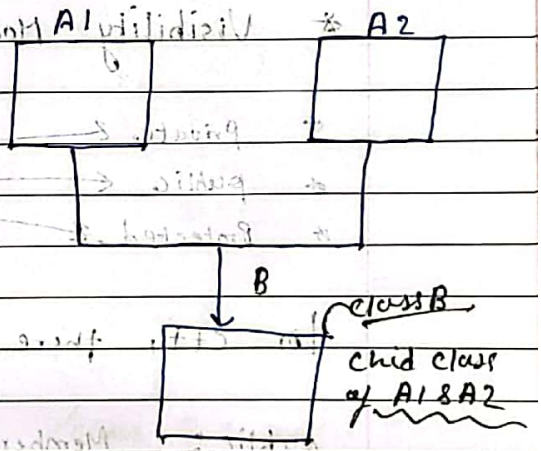
In Multilevel Inheritance lower child class have access of super parent class.



* Multiple Inheritance :-

More than one parent class existing.

```
Class A1 { };
Class A2 { };
Class B : public A1, public A2
{
};
```



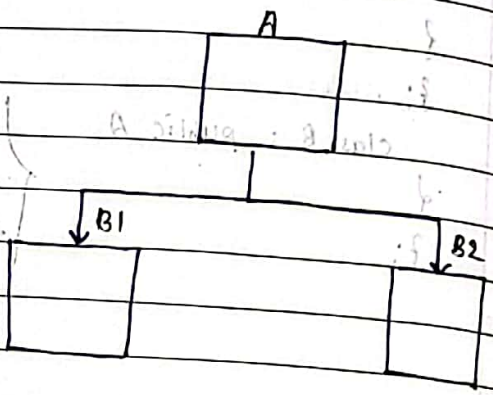
L-13

Part-2

Inheritance in C++

* Hierarchical Inheritance :→ popular Inheritance
For ex. Vehicle → for wheeler, Two wheeler.

```
class A
{   };
class B1 : public A
{   };
Class B2 : public A
{   }
```

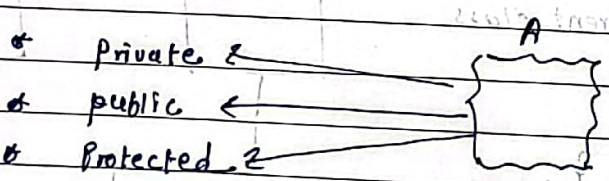


There is one parent class for several child class.

* Hybrid Inheritance :-

The mixture of two or more types of Inheritance is called Hybrid Inheritance.

* Visibility Modes or Access specifier :-



In C++, there are three access specifiers :-

public : Members are accessible from outside class.
private :- Members can't be accessed (or viewed) from outside class.
protected :- Members can not be accessed from outside the class, however they can be accessed in inherited class.

Part-2

Inheritance in C++

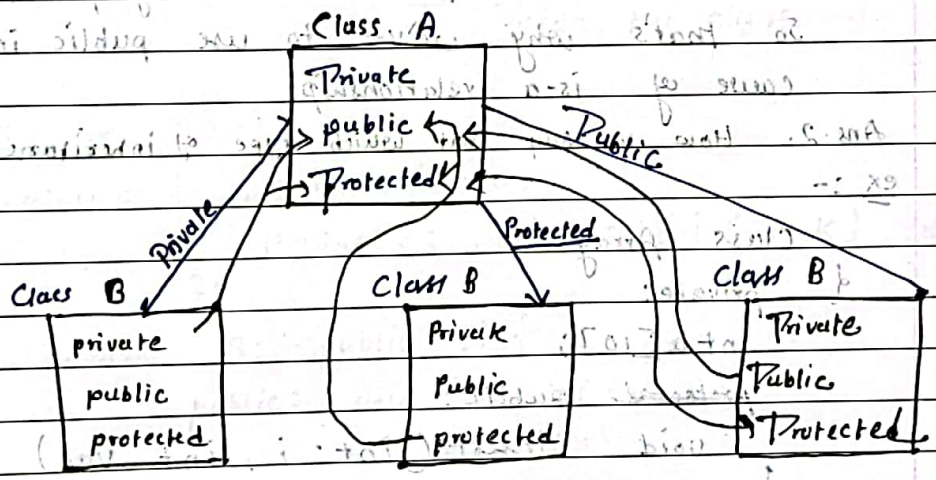
* Availability Vs Accessibility :-

If we're creating an object of class than all members, private, public & protected are Available, but we can access only public ~~private~~ members of class. So we can say that All Members are available but not Accessible.

Incause of Inheritance :-

If we deriving a class from existing class, (inherits) We can access only protected & public Members, but we've availability of All types (private, public & protected) of Members.

* Visibility Modes Incause of Inheritance




```
class A
{ };
Class B :private A
{ };
```

In this cause all public & protected members of class A will private for Class B.

```
Class A
{ };
class B : protected A
{ };
```

In above cause all public, & protected Members of Class A will protected For Class B.

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

In above cause public members are public & protected are protected Members.

L-19

Part-3

Inheritance in C++

Is a relationship :-

* Banana is-a fruit ∴ parent class Fruit, Child class - Ban
* Car is-a Vehicle ∴ Parent class :- Vehicle, Child class - C

* Three types of relationship Association, relationsh have :-
  - Aggregation
  - Composition
  - Inheritance.

→ if is-a relationship exist btw classes then Inheritance exist btw those classes.

→ Because of of is-a relationship is always implemented as a public inheritance.

↓ 1. But why ?

↓ 2. When to use private & protected Inheritance.

Ans 1 :- Because if we use private & protected Mem inheritance of in is-a relationship, we're not able to access public members of parent class So that's why we've to use public inheritance i cause of is-a relationship.

Ans 2. How we say that which type of inheritance we've to u

ex :-

```
Class Array
{ private:
  int a[10];
  protected: public:
  void Insert(int i, int Val)
  {
    a[i] = Val;
  }
};
Class STACK : private Array
{ int top;
```

Camlin Page 27
Date / /

```
public:
    void push(int val)
    {
        Insert(top, val);
    }
};
int main()
{
    STACK S1;
    S1.Insert(2,56);  // it gives error because we cant access insert fn, it declared private for class B.
    S1.push(33);
    return 0;
}
```

(It will work because, push fn is public for class B & inside it push fn is private for class B, But we knew that the members of same class can access other members inside the class.

→ Ex: for Part-2

```
class A { private: int a;
          protected: void setVal(int k) { a=k; }
};
class B: public A {
public: void setData(int x)
        { setVal(x);
        }
};
int main() {
    B b1;
    b1.setVal(4);   (It will Not work because setVal fn is protected for class B
    b1.setdata(5);  → This work because setdata is public fn for class B.
    return 0;
}
```

Part-1 Constructor in Inheritance

* In Inheritance, when we create object of derived class, what will happen.

Ans :- When we create object of derived or ~~base~~ Child class, then the constructor of child class will call but, the constructor of child class call the constructor of parent class & excute it, After that constructor of child class resume its execution.

OR

We can say that firstly the parent's class constructor will be execute after that child's class constructor will be execute.

Important NOTE :- If we declared the constructor of child class but not of parent class then Compiler automatically makes call of constructor of parent class, which is Default Constructor.

Syntax :-

```
Class
B() : A()
{
}
```

Child class constructor → Parent class constructor

if we not write this call Compiler implicitly call it, as default constructor.

Constructor execution

Obj
↓
Child const.
↓
Parent Const.
↓
Complete Parent
↓
Complete Child.

From example it will be More Clear. →

Camlin | Page 29
Date / /

Constructor in c++ with Inheritance:

Example 1:-

```
Class Z {
public:
Z(){ cout<<" Class Z Constructor:: "<<endl; }
};
Class A : public Z {
public:
A(){ cout<<"class A constructor::"<<endl; }
};
class B : public A {
public: B(){ cout<<" Class B Constructor :: "<<endl; }
};
int main(){
    B b1;
    return 0;
}
```

First Z constructor executes then, A's is the last the constructor of B will executes.

A() { by calling A(), compiler automatic calls Z() ↓ Making A():B()

Firstly for object class B, Constructor of B will call it calls

OutPut:-
Class Z Constructor::
Class A Constructor::
Class B Constructor::

Example : 2

↳ In class we've not any default Constructor.

```
class A{ int a;
public:
A(int p){ a= p; } // parameterized Constructor
};
class B: public A
{ int b;
public:
B(int x, int y): A(x) // Gives error → B(int x, int y): A()
{ b=y; }              // or → B(int x, int y) - error.
};
int main(){
B b(3,4);
return 0;
}
```

Because in class A no default Constructor so compiler will call default Constructor, So we've to make it suitable call so we passing x as argument in :A(x).

L-19

Camlin Page 38
Date

Part-1 Destructor In C++, with Inheritance

Ex:-

```
Class A {
  int a;
  private:- public:
    A(int x, int y) // parameterized Constructor
    { a = x+y; }
  ~A() {
  cout<<" Class A Destructor :: "<<endl;
  }
};

Class B : public A
{ int a, b;
  public:
  B(int p, int q) : A(pep, qeq) // parameterized Constructor with call stak. of Class A's
  { a=p; b=q; }
  ~B() {
  cout<<" Class B constructor :: "<<endl;
  }
};

int main() {
  B b1(4, 5);
  return 0;
}
```

Output :-

Class B Constructor ::
Class A Constructor ::

In Inheritance, incase of destructor firstly the destructor of ~~ele~~ child class (B) will call & execute and after that the parent class (A) Destructor will call & execute. {Opposite of Constructor}

Note:- Incase of destructor we've not need of state & declare (in child class) the state of calling of parent class destructor, Because In Destructor no parameter's, so always like the Default destructor, we don't need to pass any parameter & no need of predfine so compiler does the calling work of dutructor implicitly.

L-15
Part-1

This Pointer in C++

Camlin Page 31
Date / /

- Pointer can holds the type of its own type address.

- A pointer contains address of an object is called object pointer.

For ex :-

NOTE: Instance Member fn, Fn inside class are called instance member fn, which is not static Member fn

Instance Member fn = Nonstatic Member fn

```
Class Box
{ private:
    int l, b, h;
    void setDimensions(int x, int y, int z)
    { l = x; y = b; h = z; }
    void showDimensions()
    { cout << "Length :: " << l << "Breadth :: " << b << "Height :: " << h; }
};
int main() {
    Box *p;  // pointer object or object pointer
    Box b1;
    p = &b1; // p is an object pointer because holding the address of object b1.
    p -> setDimensions(3, 4, 5);
    p -> showDimensions();
    return 0;
}
```

* This Pointer :- - This is a KeyWord.

- This is a local object pointer in every instance member function containing address of the caller object.

- this pointer can not be modify.
- It is used to refer caller object in member function.

NOTE:
In every instance member fn of class a this pointer exist, which holds the address of caller object, it is inbuilt in c++, we don't need to build it inside instance member function.

This pointer in C++

NOTE: Each object gets its own copy of data Member and all objects shares a single copy of member functions.

Imp-line: As well Compiler supplies an Implicit pointer along with the name of functions as "this".

for eg:-

```
class Box {
    private: int l, b, h;
    public:
    void setDimension(int l, int b, int h)
    {this->l = l; this->b = b; this->h = h;} // Here the this local variable are in priority, so this pointer points the caller object bi. Access (i.e. Data number.
    void showDimension() {
        cout<<"length::"<<l<<"Breadth::"<<b<<"height::"<<this->l;
    }
};
int main() {
    Box b1;
    b1.setDimension(3,4,5); // this pointer is passed as a hidden argument to all nonstatic (instance) function calls.
    b1.showDimension();
    return 0;
}
```

hidden argument of this pointer

* Use - to refer to the invoking object.

Camlin Page 33
Date / /

New & delete In C++

* SMA :- Static Memory Allocation or Compile time
* DMA :- Dynamic Memory Allocation or Run time

```
int x;
float y;
Complex c1;
Student s1;
```

SMA :- At the time of compilation it is cleared & decide that how much memory (variables) are going to consume at Run time is called compile time or static memory allocation. & we are not flexible with SMA, because we dont know how much memory actually needed.

DMA :- Memory will be allocate at run time.

Keyword new :- new keyword allocate memory at run time (Dynamically) to the variables.

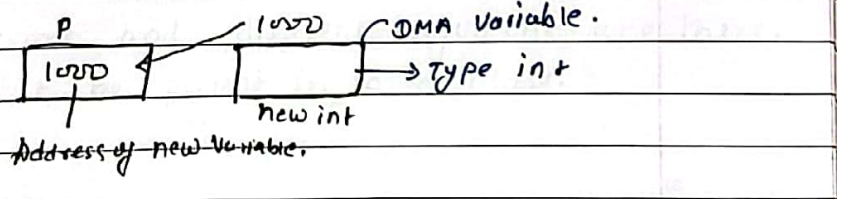
(in C++, a feature is that we can make some new variables, whose memory will allocate at run time, with new keyword we can do that.).

for

```
int *p = new int;
float *q = new float;
Complex *ptr = new Complex;
```

They should be Same.

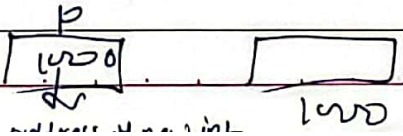
```
* ⟹  int *p = new int
```



How the new syntax keyword will work :-

(i) Step (i) :- Initialy the left part of statement → (int *p = new int;) will make int type pointer p & this pointer holds the address of new int.

left part



Part-1

New & delete keyword in c++

* We can make an array with the help of new keyword for eg

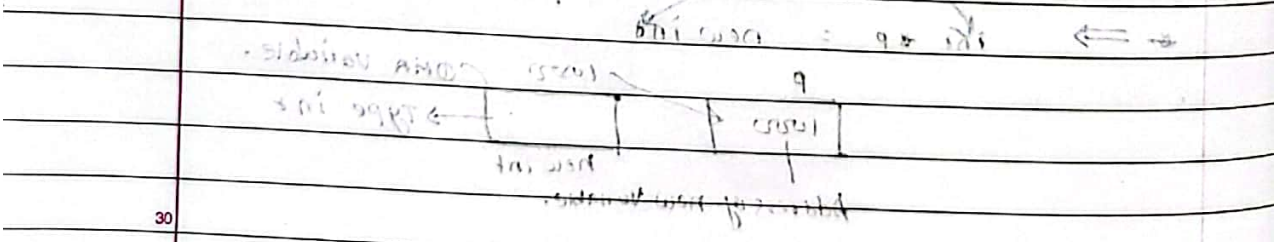
```
float *q = new float[5];   // size of array
```

or

```
int xe;
cin >> xe;
int *p = new int[xe];
```

* Keyword :- Delete (delete) - To realease memory delete keyword deallocate the memory of those variables, who have created with new keyword and those variables whose memory will've allocated been dynamically.

* delete p;
* delete []p;



L-17
Part-1

Camlin Page 35
Date / /

Method Overriding in C++

* Method Overloading – Same fn with different Argument with in same class
* Method Overriding → same f° & argument but in child class redefined
* Method hiding :- Same fn with differ argument but in child class

```
class A {
public:
    void f1() { }
    void f2() { }
};
class B : public A
{
public:
    void f1() { }  // Method overriding
    void f2(int x) { }  // Method hiding
};
int main()
{
    B b1;
    b1.f1();
    // b1.f2();  Gives error - explained in → Method hiding
    b1.f2(14);
    return 0;
}
```

* Method Overloading :- Incause of Method overloading same function name and different argument are there, but they must be present in same class.

For ex:

```
Class A {
public: void fn1(int a, int b) { }
        void fn1() { }
        int fn1(int x) { }
};
```

all the fn in same class with same name, having different arguments

L-17

Part-1

Method Overriding

Camlin Page 36 Date / /

* Method overriding :- When we write same named fn inside parent class & child class both, then the child's class fn overrides the fn which is present in parent class.

(It means that if we make a object of class child and call the fn then child's class fn will executes (always).

for ex
```
class A {  public:
            void f1() { }
};
class B : public A {
        public: void f1(): { }
};
int main() { B b1;
        b1.f1(); class B fn will call; return 0;
}
```

* Method hiding :- In this cause we write one fn in parent class and one in child class but with different arguments. So when we call the function without argument (through child class obj then child class fn will call & error occurs.
in child class compiler finds that same fn with different argument, & we've to call this fn with argument to resolve error.

for eg - example 1st of this lecture.

* What is Early Binding :- (compile time) polymorphism

When we do method overriding or method hiding or fn overloading the same fn in different class we declare, so when we ~~run or~~ compile this program, compiler associates the method to each fn call, it mean for each fn call Compiler bind a method, so we called early binding because this process done at compile time.

Part-1

L-17

Camlin Page 37
Date / /

Method Overriding

Example 2 → Common example :-

Here we are making two of class car is sportsCar. & defining a fn shiftGear in both classes, with different code. The child class shiftGear fn is updated version for new car, & for old cars, the car fn will execute, so we've a sports car then we will make object of class sportsCar & from we've to call shiftGear function.

```
Class Car { public:
        void shiftGear(){
        cout<< " Old way :: " <<endl;
        }
        void f2() { }
};
Class sportsCar : public Car
{ public:
    void shiftGear() {      Method Overriding
    cout<<" New way :: "<<endl;
    } void f2(int x) { } // Method hiding.
};
int main() {
    Class sportsCar sc;
    sc.shiftGear(); for class sportsCar
    // sc.f2(); error
    sc.f2(4); //for class " sportsCar
    return 0;
}
```

L-18

Part-1

Virtual Function in c++

* Base class pointer can point to the object of any of its descendant class. (वंशज)

For ex. if we've a parent class Car & a child class SportsCar

```
class Car {
};
class SportsCar : public Car {
};
int main() {
    Car *p;          // Base class pointer
    SportsCar sm;    // child class object
    p = &sm;
    return 0;
}
```

* Example of virtual function:-

```
class A {
public:
    virtual void show() {          // if we not write virtual keyword then compile time (early) binding will be there.
        cout << "This is a parent class" << endl;
    }
};
class B : public A {
public:
    void show()
    {
        cout << "This is a child class" << endl;
    }
};
int main() {
    A *p;
    B b;
    p = &b;          // run-time binded fn show()
    p->show();       // output: This is parent class.
    return 0;
}
```

Because of early binding compiler don't check that what the object pointer contains, it checks that which class object is there, and bind the fn according to it.

Virtual fn in C++

Defination of Virtual function :-

A virtual function is a member function which is declared within a base class and is re-defined (overridden) by derived class. When you refer to a derived class object using a pointer or a reference to the base class, you can call a virtual function for that object and executes the derived class's versions of the function.

- Virtual functions ensure that the correct function is called for an object, regardless of the type of reference (or pointer) used for function call.
- Virtual function mainly used to achieve Run-time polymorphism.

Imp
- Functions are declared with a virtual keyword in base class, and the function who have same prototype as virtual function are by default virtual function in derived class.
- The resolving of function call is done at runtime.

* Rules of Virtual fn :-
* Virtual fn can not be static.
* A virtual fn can be friend fn of another class.
* Virtual function should be accessed using pointer or reference of base class type to achieve runtime polymorphism.
* The prototype of virtual function should be same in base as well as derived class.
* A class may have virtual destructor but it can't have a virtual constructor.
* They are always defined in base class, & overriden in derived class, but it is not mandatory for derived class.

L-19

Pool-1

Camlin Page 41
Date / /

Abstract class in c++

* Pure Virtual function :- (Abstract fn)

A do nothing function is called pure virtual function.

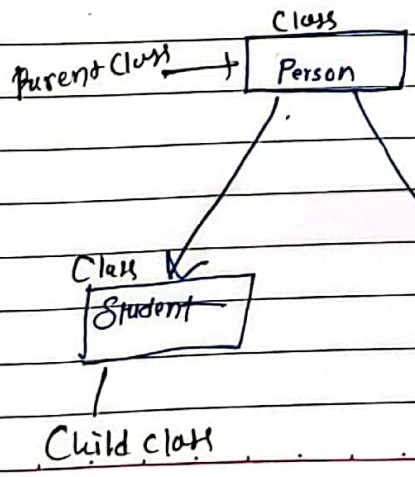
Or sometimes implementation of any function can not be provided in a base class, we don't know the implementation. Same in pure virtual fn in c++ is a virtual function which we can have implementation, But we must override that function in derived class. Syntax of pure virtual fn

{ virtual (return type) name of fn () = 0; }.

* Abstract class :-

A class having abstract fn or at least one pure virtual function is called Abstract class. And we can't create the object of Abstract class because we've a do-nothing fn (pure virtual fn) inside the class. so when we can't create ~~compiler is not able~~ the object of class, we're not able to access and call the pure virtual fn (it means c++ developers don't want that pure virtual fn have to be call so rule is that) we can not create the object of abstract class. But we can access this (pure virtual fn) with the help of inheritance (derived class). So it is neccessary to define the pure virtual fn in child or derived class.

For ex:-



So in this example Student & faculty can have some common properties like name, email-id, mob no etc, but some uncommon like roll no, salary so for we define those fn which have common use in person

L- 19

Part - 1

Abstract Class & Pure Virtual function in C++

* Example of Abstract Class & Pure virtual fn:

```
class Person // Abstract class
{
public:
virtual void func() = 0;
void f1()
{ cout<<"Normal fn in parent class"<<endl;
}
};
class student : public person
{ public:
void func() // Redefination of pure virtual fn (Overridd
{
cout<<"Implemented pure virtual fn"<<endl;
}
};
int main(){
Student s1; Person *p; p = &s1;
s1.f1();
s1.func(); // late Binded
return 0;
}
```

Pure virtual fn

if we remove virtual key word fro this declaration compiler will give error this fn defined to late binding

Means what type of Object pointer (Object p hold & according to binds the function.

// p->f1();
p->func();

for More Information See the laptop example