Date:
Page No:

Private member functions

a private member function can be called by another function that is the member of its class. even an object can't invoke a private member using a dot operator.

function

Static Data Members

→ It is similar to c-static variables.
→ It is initialized to zero. when first object of its class is created no other initialization is permitted.
→ Only one copy of that member is created for entire class and is shared by all the object of that class. no matter how many objects are created.
→ It is visible only within the class but it's lifetime is the entire program.

Ex

```
class Item
{ static int count;
    int number;
Public:
  void getdata (int a)
    { number = a;
      count ++;
    }
```

Date:
Page No:

```
void getcount()
{ cout<<count;
};
int item::count; // we have declare static member

int main()
{
 item a,b,c;
 a.getcount(); 0
 b.getcount(); 0
 c.getcount(); 0
 a.getdata(100); 100,1
 b.getdata(200); 200,2
 c.getdata(300); 300,3
 a.getcount(); 3

 return;
}
```

Date:
Page No:

→ Type and scope of each static member must be defined outside the class definition.
→ Static data members are stored separately.

Static Member Functions

→ A static function can have access only other static members whether it is function or variable declared in same class.
→ A static member function can be called using class name instead of object.

```
class_name :: static_function
```

Array of objects

```
class item
{
};
item a, b, c[10];
```

// c is a array of 10 objects name as c[0], c[1], .. c[9];

Date:
Page No

Object as function argument

Call by reference ⇒ C++

```
swap ( int &a, int &b)
{ int t;
    t=a;
    a=b;
    b=t;
}
swap (m,n);
```

We can access function as a argument in two ways

i) A copy of entire object is pass to the function.
ii) Only the address of object is transffered to the function.

```
Class Time
{
  int H;
  int M;
Public:
    Void getTime (int h, int m)
    { H=h;
      M=m;
    }
```

Date:
Page No:

```
void Puttime( )
{
    cout<<H;
    cout<<M;
}

void sum (Time, Time) ;  // passing objects
};                  T1  T2

Void Time :: Sum (Time t1, Time t2)
{
    M = t1.M + t2.M;
    H = M/60;
    M = M%60;
    H = H + t1.H + t2.H;
} return (H,M);

int main()
{ Time t1,t2,t3;
t1.gettime (2,45);
t2.gettime (3,45);
t2.go sum (t1,t2);
t1.Puttime();    = 2,45
t2 : put time();  = 3,45
t3 : puttime();   = 6;30
}
```

Date:
Page No:

Friend Function

An Ordinary function that is not the member function of class has no privilages to access to the private data member but the friend function dose have the capability thto access any private date members

We declare the friend function using friend keyword inside body of class.

So

Characteristics of friend

→ It is not in the scope of the class to which it has been declared as friend.
→ Since it is not in the scope of class it can not be called using the object of that class.
→ It can be invoked like normal function without the help of any object.
→ It can be declared either private or public part of the class. without affecting its meaning.
→ Usually it has object as argument

```
class item
{ int number;
    int cost;
public:
    void set value ( )
    {
```

Date:
Page No:

```
    number =10;
    cost = 100;
}

friend float mean (item s);
};

float mean (item s)
{ return float ( s.number + s.cost )/20;
}

int main()
{ item x;
  x.setvalue (100, 50)
  cout << mean(x);
}
```

The friend function accesses the class variable no. and cost by using the dot (.) and object pass to it.

The function mean passes object s by value to the friend function.

Date:
Page No.

Constructure

→ A constructure is a special member function or automatic initialization of an object
→ Whenever an object is created the constructure will be executed automatically.
→ A constructor function can be overloaded to accomodate many different forms of intialization.

Sytax of constructor function

→ A constructor name must be same as that of its class name.
→ It is declared with no return type

```
Class abc{
    Public:
    abc ()
    {
        cout<<"constructor Call";
    }
    void main()
    { abc obj1;
    }
```

Date:
Page No:

Types of Constructor

1) Default Constructor
2) Paramatries Constructor
3) Copy Constructor

Default Constructor

It is a constructor with no argument or all argument should have default values.

→ It can be explicitely return in a program in case if constructor is not defined in a class the C++ compiler will automaticaly generates in a program.

```
Class Student {
    Privale:
        Char name [20];
        int roll no;
    Public:
        student () { }  — Case-1
        Student (int x=10) { roll no = x }  → Case-2
        default →  Case-3
f void main()
    f Student S1;
    }
```

Date:
Page No.

2) Paramatrized Constructor

→ It is a constructor with paramaters is defined in the class then C++ will not add any default constructor for intialization.

→ ~~overloading~~

Overloading of Constructor

```
Student {
  Private:
    char name[50];
    int rollno;
  Public:
    Student (int x)
    { roll no. = x }

  Student ()
    { cout << "without argument"; }

  Student ( Char x, int y)
  { cout << "two argument"; }

  Student (char x) {
    { cout << "char argument";
```

0 1 1 2 3 5 8 13 - —

Date:
Page No:

WAP to generate series of fibbonacci numbers using constructor

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

class fibbonacci {
        int n;
        int first;
        int second;
    fibbonacci(int x) {
        n=x;
        first=0;
        second=1;
    }
};

int main()
{
    int y;
    cout << "Enter no of fibbonacci series";
    cin>> y;

    fibbonacci num(y);
```

Date:
Page No:

```
class fibbonacci{
        int n;

        fibbonacci(val){
                n=val;
                int first=0;
                int second=1;
                if(n==1)
                cout<<"0_";
                if(n==2)
                cout<<"1=";
        }

int main(){
        int val;
        cout<<"Enter of terms in fibbonacci series";
        cin>>val;

        fibbonacci ~~(val)~~;
        fibbonacci new(val);

fibbonacci. fibbo(val);

}
```

Date:
Page No:

```
void fibbo ( int x)
{
  if (X>=2)
  {
    sum= first + second;
    second= first+ second;
    second= sum;
  }
```

WAP to find area of circle and rectangle using class where all the data member should be kept in private.

```
class Area {
        int r;
        int l;
        int w;

        Area (int R, int L, int W)
        { R=r; r=R;
          L=l; l=L;
          W=w; w=W;

public:
void area of circle (){

        cout << "Area of Circle:" << 3.14*r*r;
            return;
          }
void area of rectangle (){
        cout << "Area of rectangle :" << l*w;
        return;
        }
};
```

Date:
Page No:

```
int main()
{ Int R, L, W;
  Area A;
  cout << "Enter r" << endl;
  cin >> R;
  cout << "Enter l" << endl;
  cin >> L;
  cout << "Enter w" << endl;
  cin >> W;
  Area( R, L, W);
  void areaofcircle();
      areaofrectangle;
    return 0;
  }
```

Copy Constructor

The copy constructor is constructor which is used to initialize an object by another object of the same class, which has been created previously.

The general format of copy constructor-

```
Class_name (Const Classname & obj)
{
    // body of const;
}
```

Date:
Page No:

```
Class Test
{
  int x,y;
Public:
   Test (int a, int b)
     {
       x=a;
       y=b;
     }
   Test (const Test &obj)
     {
       X=obj.X;
       y=obj.y;
     }
};
int main()
{ Test obj1(10,20);

// calling copy constructor
  Test obj 2= obj 1;
  Test obj 3(obj 1);
```

If we dont define our own copy constructor the C++ compiler creates a default copy constructor for each class which does a member-wise copy between objects.

friend fun Date:
Page No:

In c++ a copy constructor called in following cases -

1) When an object is constructed based on another object of the same class.
2) When an object of a class is passed by an argument.
3) When an object of a class in returned by value.

Destructor:-

→ A destructor is a function that automatically execute when object is destroyed.
→ The primary usage of destructor function is to release space occupied by an object.
→ A destructor function may be invoked explicitly.

→ Rules of writing a destructor function

→ A destructor function name is the same as that of class except the first character of the name must be tilt sign. (~).

→ It is declared with no written type.
→ It not takes any arguments and therefore can't be overloaded.
→ If we don't write our own destructor in class. compiler creates a destructor for each class.

Date:
Page No:

```
class ABC
{
  Public:
  ABC(){ }        Constructor
  ~ABC(){ };      destructor
};
```

How to return

```
class Ex
{
  Public:
  int a;
  Ex add( Ex x, Ex y)
  {
    Ex Z;
    Z.a = x.a + y.a;
    return z;
  }
};

int main()
{
  Ex E1, E2, E3;
  E1.a = 50;
  E2.a = 100;
  E3.a = 0;
```

Date:
Page No

```
cout << E1.a << E2.a << E3.a;
E3 = E3.add (E1,E2);
cout << E1.a << E2.a << E3.a;
return 0;
}
```

50 100 0

50 100 150

Friend Function

Member function of a one class can be friend function of another in such case they are defined using scope resolution operator.

```
Class A
{ Public:
  Void show() { };
};
Class B
{ Public:
  Friend Void A:: show();
};
```

Date:
Page No:

```
Class B;
Class A {
    Public:
    void showB(B&);
};

Class B {
    int b;
  Public:
    B() {b=0;}
  friend void A::showB(B&x);
};

Void A::showB(B&x)
{ Cout << x.b;
}
int main()
{ A a;
  B x;
  a.Show(x);
}
```

→ friend Class A;

We can also declare all the member function of one class as the friend function of another class in such case class is called friend ~~class~~ class.

Date:
Page No:

```
class B;
class A { int x;
  public:
    void set (int a)
        { x=a; }
  friend void max (A, B); };
  class B {
    int b;
  public
  void set(int x) { b=x; }
  friend void max (A, B);
    };
  max (A obj1, B obj2)
  { if (obj1.x > obj2.b)
    { cout << obj1.x; }
    } else
        { cout << obj2.x }
main()
{ A a;
  B x;
  a.set (50); x.set (100);
  max(a, x);
  return 0;
}
```

Date:
Page No:

Q: WAP to convert a decimal into binary no. using friend function.

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

class A{
        int val;
        public;
        void set(int x)
          { x=val;}

        int
    Friend conversion(A);
    }

int conversion(A)
  { string int s[20];
    int temp, i=0;
while(A.x!=0)
  { temp= A.x%2;
    A.x= A.x/10;
    i++ s[i]=temp;
        i++;

  }
```

Date:
Page No:

Classes within class

C++ permits declaration of a class within another class. A class declared as a member of another class is called as Nested Class.

The nested class is in the scope of its enclosing class.

```
class stud {
    private:
        char name[20];
        int rollno;
    public:
        class dat {
        private:
          int day, month, year;
        public:
          void setdata (int d, int m, int y)
          {
            day = d;
            month = m;
            year = y;
          }
        };
        void display ();
};
```

Date:
Page No:

```
int main()
{
    Stud obj;
    [stud:: date obj 1;] → Scope resolution operator
    obj 1.setdata(2,4,2023);
}
```

→ The member of an enclosing class have no special acc excess to members of a nested class. The usual access rule shall be obeyed.

→ The inner class can access private data of enclosing class.

Eg: int x, int y class → public

```
Class outer {
    int x;
    public;
    class inner{
        int y;
        public
        void add (int a, int b)
        {
            x=a;  → can't access direct x or y accessing x we need to pass object-
            y=b;
            cout<<"x+y<<endl;
        };
};
int main()
{ int a+b;
```

Errors →

Date:
Page No:

```
int main(){
    outer obj1;
    outer::inner obj2;
    int a,b;
    cin>>a;obj1.x;
    cin>>b; obj2.y;
outer::add(obj1.x, obj2.y);

Class outer{
    int x;
    public:
    outer(int a)
    {
        x=a;
    }
    class inner{
        int y;
        pulic:
        inner(int b)
        {
            y=b;
        }

    void add(obj1.x + obj2.y)
    {
        return obj1x + obj2y ;
    }
```

Date:
Page No:

```
int main()
{ outer obj1(2)
  outer::inner obj2(3);
  inner.add(obj1, obj2);
  return 0;
}
```

Inheritance

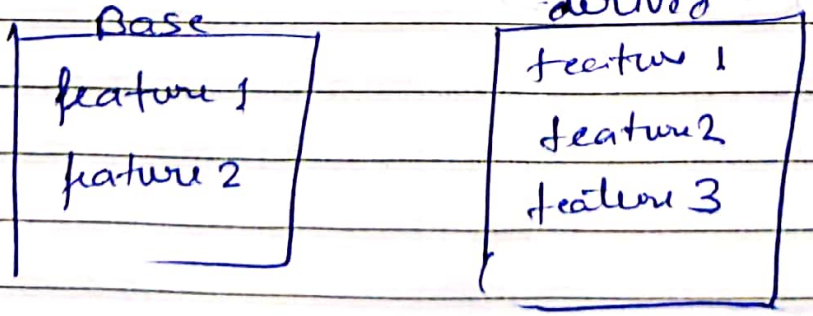
→ Inheritance is the process of creating a new class from an existing class.

→ It is a form of software reuse in which you create a class that built on existing class capabilities then customize and enhance them

→ The existing class is known as base class and newly created class is called as derived class.
Ex- Person, student, Teacher, Vehicle

→ Inheritance represent is a relationship

→ The main advantage of inheritance are
1) Reusability of code
2) To increase reliability of code
3) To add some enhancement to base class
4)

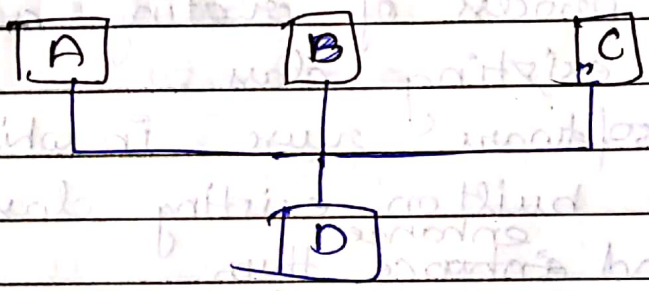


Date:
Page No:

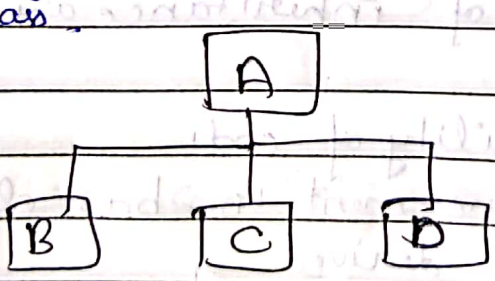
Types of Inheritance

1) Single Inheritance - A derived with only one base class.

2) Multiple Inheritance - Derived class with several base classes.



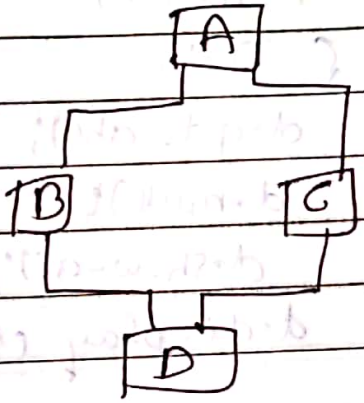
3) Hierarchical Inheritance - Property of one base class may be inherited by more than one class.



Date:
Page No:

4) Multilevel Inheritance :- Deriving a class from another derived class.

5) Hybrid Inheritance :- It is a combination of 2 or more Inheritance.



6) General Syntax of Derived Class

```
Class derived_class_name : Visibility_Mode Baseclass_name
{
    Data members;
    Datafunction;
};
```

Ex

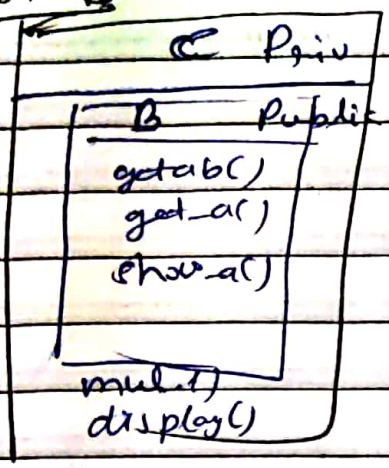
```
Class B
{
  int a;
  Public:
  int b;
  void get ab();
int void get_a();
  void show_a();
};
```

```
[ Class D: Public B ]
  {
    int c;
    Public:
    void mul();
    void display ();
};

void B :: get_ab()
   { a=10;
     b=20;
   }
int
void B :: get_a()
   { return a; }

void B :: show_ a()
   { cout <<a;
   }
```

```
void D :: mul()
  {
    c= b* get_a();
  }
void D :: display()
  { cout <<get_a();
    cout <<b;
    cout <<c;
  }

int main()
{ D d;
  d.get_ab();
  d.mul();
  d.show_a();
  d.display ();
}
```

Date:
Page No:

Public D



Private D