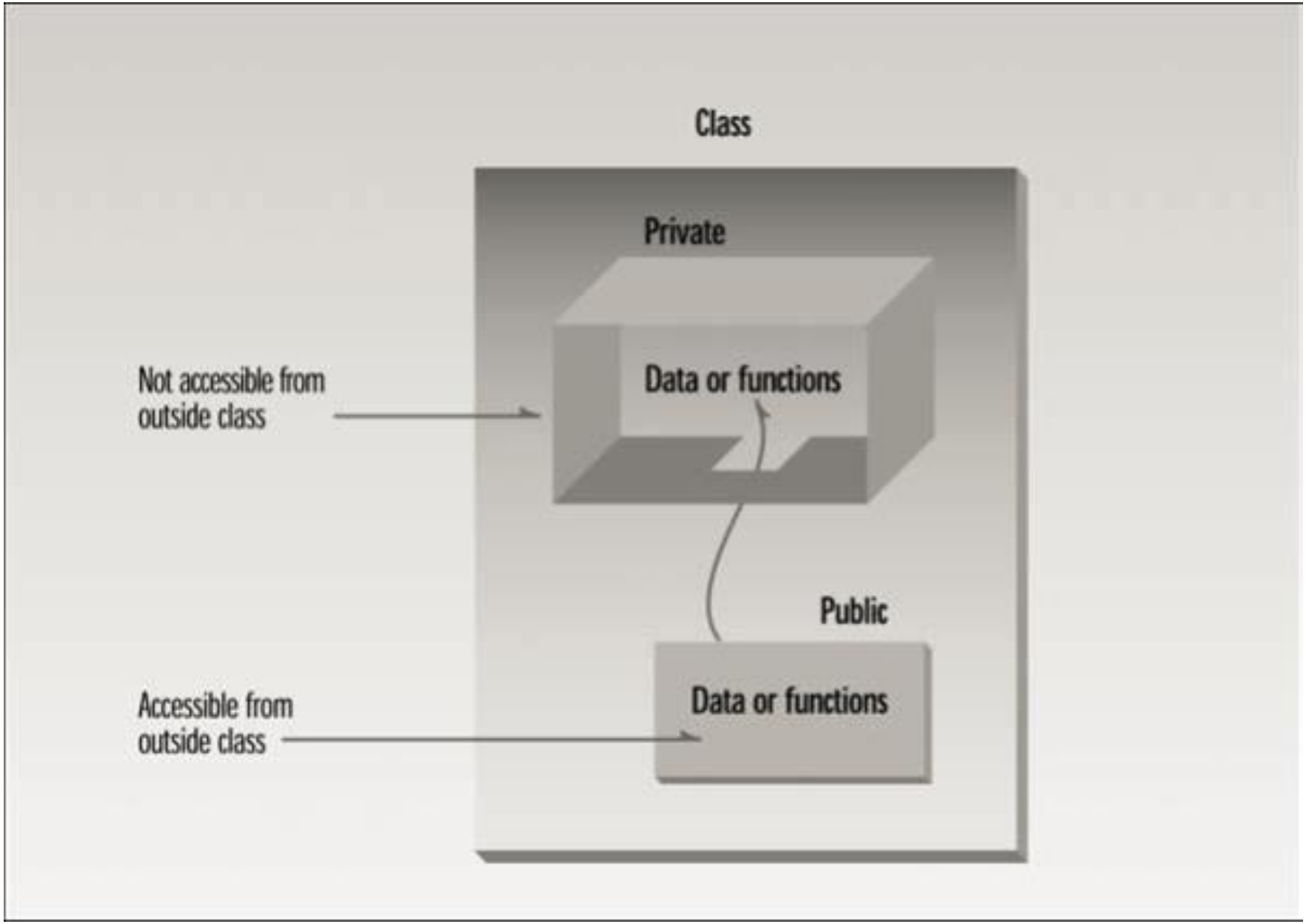
C++ : Object-Oriented Programming

- **Classes and Objects**

C++ class

```
class  class_name
{
    access_specifier:
        data and functions
            ...
    access_specifier:
        data and functions
            ...
} ;
```

Private and public



Class example-1

```
3   class My_class
4   {
5       private:
6
7           int data;
8
9       public:
10          void setdata(int x)
11          {
12              data = x;
13          }
14          void showdata()
15          {
16              cout<<"data = "<<data;
17          }
18  };
```

```
21  int main()
22  {
23      My_class ob1, ob2;
24      ob1.setdata( 11 );
25      ob2.setdata( 22 );
26      cout<<"\n ob1:";
27      ob1.showdata();
28      cout<<"\n ob2:";
29      ob2.showdata();
30      return 0;
31  }
```

OUTPUT:

```
ob1:data = 11
ob2:data = 22
```

member function definition outside the class:

Syntax:

```
return_type class_name :: function_name( parameters )
{
        function_body;
}
```

```
3  class My_class
4  {
5      private:
6
7          int data;
8
9      public:
10         void setdata(int);
11         void showdata();
12 };
```

```
13 void My_class :: setdata(int x)
14 {
15     data = x;
16 }
17 void My_class :: showdata()
18 {
19     cout<<"data = "<<data;
20 }
```

Class example-2

```
3   class My_class
4   {
5       private:
6
7           int data;
8           void setdata(int x)
9           {
10              data = x;
11          }
12      public:
13
14          void showdata()
15          {
16              cout<<"data = "<<data;
17          }
18  };
```

```
21  int main()
22  {
23      My_class ob1, ob2;
24      ob1.setdata( 11 );
25      ob2.setdata( 22 );
26      cout<<"\n ob1:";
27      ob1.showdata();
28      cout<<"\n ob2:";
29      ob2.showdata();
30      return 0;
31  }
```

Error ! Setdata() is not accessible

Class example-3

```
3   class My_class
4   {
5       private:
6
7           int data;
8           void setdata(int x)
9           {
10              data = x;
11          }
12      public:
13
14          void showdata(int x)
15          {
16              setdata(x);
17              cout<<"data = "<<data;
18          }
19  };
```

```
21  int main()
22  {
23      My_class ob1, ob2;
24      //ob1.setdata( 11 );
25      //ob2.setdata( 22 );
26      cout<<"\n ob1:";
27      ob1.showdata( 11 );
28      cout<<"\n ob2:";
29      ob2.showdata( 22 );
30      return 0;
31  }
```

OUTPUT:

```
ob1:data = 11
ob2:data = 22
```

Class example-4

```
3   class Class_Demo
4   {
5       int data1;
6   public:
7       int data2;
8       void setdata(int d)
9       {
10          data1 = d;
11          data2 = d*2;
12       }
13      void showdata()
14      {
15          cout<<"Data1="<<data1<<","
16              <<"Data2="<<data2<<"\n";
17      }
18  };
```

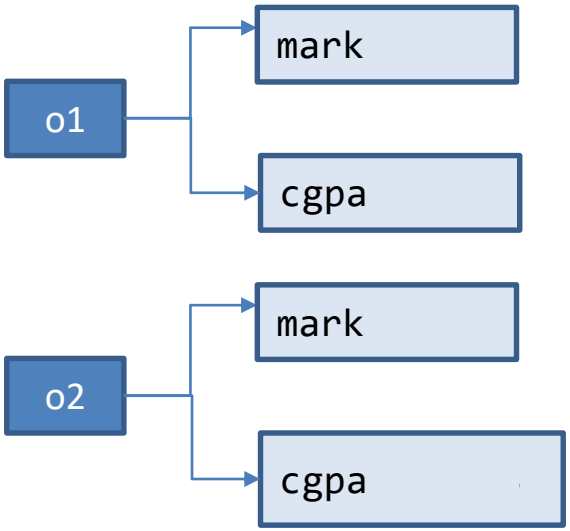
```
19  int main()
20  {
21      Class_Demo O1,O2;
22      O1.setdata( 5 );
23      O2.setdata( 10 );
24      O2.data2 = 100;
25      cout<<"O1:";
26      O1.showdata( );
27      cout<<"O2:";
28      O2.showdata();
29      return 0;
30  }
```

OUTPUT:

```
O1:Data1=5,Data2=10
O2:Data1=10,Data2=100
```

```
class Test
{
   private:
     int mark;
     float cgpa;
   public:
    void SetData()
    {
     cin>>mark;
     cin>>cgpa;
    }
    void DisplayData()
    {
     cout << "Mark= "<<mark;
     cout << "cgpa= "<<cgpa;
    }
} ;
```

```
int main()
{
   Test o1,o2;

   return 0;
}
```



Question-1

```
3   class Test
4   {
5       int a;
6   public:
7       void init()
8       {
9           a = 0;
10      }
11      void set()
12      {
13          a++;
14      }
15      void show()
16      {
17          cout<<" a="<<a;
18      }
19  };
```

```
20  int main()
21  {
22      Test ob1, ob2;
23      ob1.init();
24      ob2.init();
25      ob1.set();
26      ob2.set();
27      cout<<"\n ob1:";
28      ob1.show();
29      cout<<"\n ob2:";
30      ob2.show();
31      return 0;
32  }
```

OUTPUT:

```
ob1:  a=1
ob2:  a=1
```

Class example-5: Nesting of member functions

```
3   class set
4   {
5       int m, n;
6    public:
7       void input( );
8       void display( );
9       int largest( );
10  };
```

```
12  int set::largest( )
13  {
14      if(m>=n)  return m;
15      else return n;
16  }
17  void set::input( )
18  {
19      cout<<"Enter values for m and n:";
20      cin >> m >> n;
21  }
22  void set::display ( )
23  {
24      cout<<"largest="<<largest();
25  }
```

```
26  int main( )
27  {
28      set A;
29      A.input();
30      A.display();
31      return 0;
32  }
```

OUTPUT:

```
Enter values for m and n:10 20
largest=20
```

Question-2

```
3   class Class_Demo
4   {
5       int d1 , d2;
6   public:
7       void init()
8       {
9           d1 = 0; d2 = 0;
10      }
11      void incr()
12      {
13          d1++; d2 += 10;
14      }
15      void show()
16      {
17          incr();
18          cout<<d1<<"\t"<<d2;
19      }
20  };
```

```
21  int main()
22  {
23      Class_Demo O1,O2;
24      O1.init();
25      O2.init();
26      O2.incr();
27      cout<<"\nO1: ";
28      O1.show( );
29      cout<<"\nO2: ";
30      O2.show();
31      return 0;
32  }
```

OUTPUT:

```
O1: 1    10
O2: 2    20
```

Question-3

```
 3  int c = 0;
 4  class Test
 5  {
 6      int a;
 7  public:
 8      void set(int x)
 9      {
10          a = x;
11          c++;
12      }
13      void show()
14      {
15          cout<<" a="<< a <<"\t c="<<c;
16      }
17  };
```

```
18  int main()
19  {
20      Test ob1, ob2;
21      ob1.set( 25 );
22      ob2.set( 29 );
23      ob1.show();
24      ob2.show();
25      return 0;
26  }
```

OUTPUT:

```
a=25        c=2
a=29        c=2
```

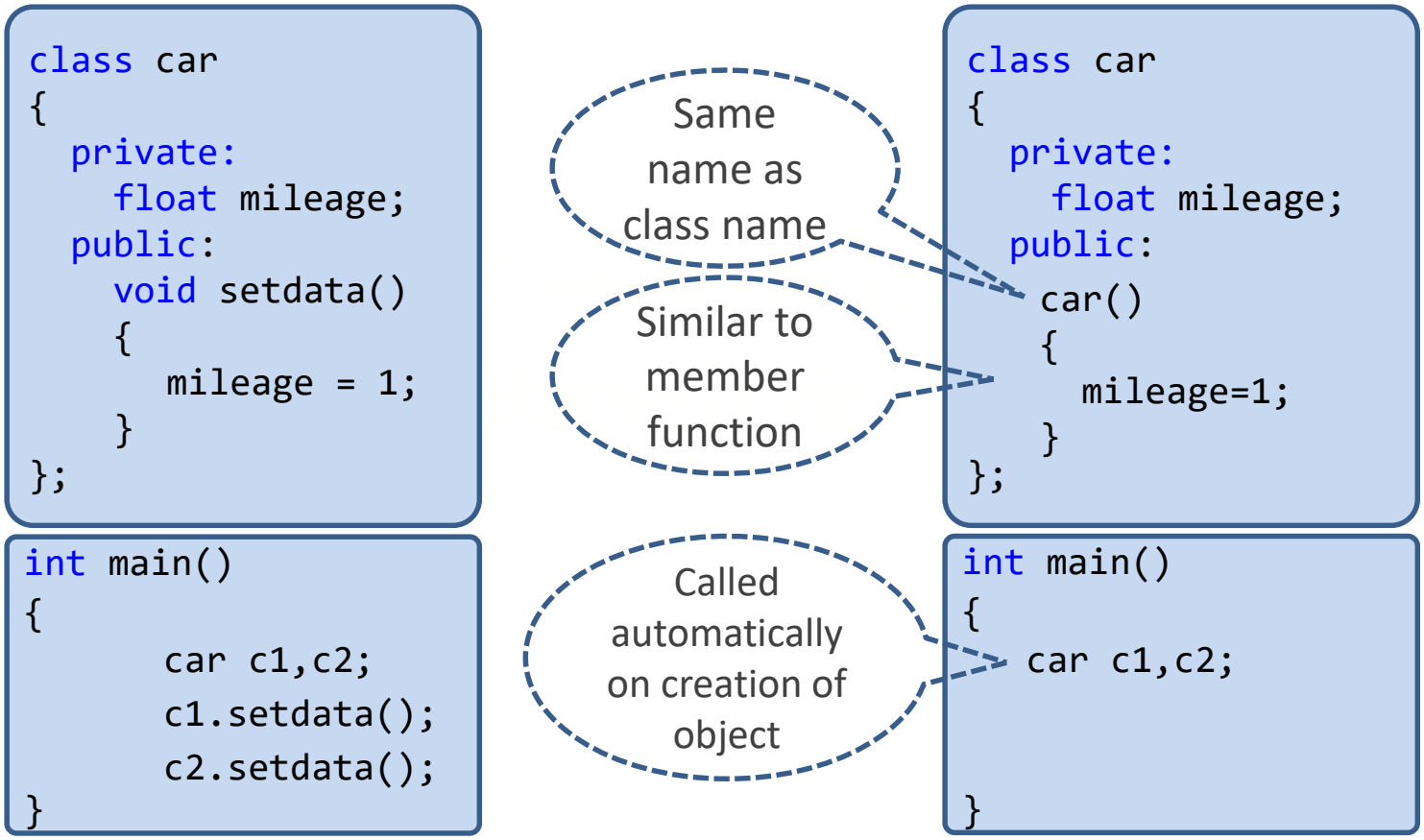
Initialization and Constructors

- *The constructor is responsible for initializing objects automatically whenever they are declared.*
- *The programmer has the option to provide his/her own constructors, which get called whenever a new object is declared.*
- *Otherwise, the compiler provides a default constructor.*

Constructors ...

- *The constructor is like any other method (i.e., member function) in that,*
  - *it takes arguments*
  - *it can be overloaded*
  - *its arguments can be defaulted*
- *But with one big difference: It has no return value, not even **void**.*
- *One other very important characteristic: the constructor's name is the same as the class name.*

Constructor



Constructor

```
class Rectangle
{
 int width,height;
 public:
 Rectangle(){
   width=5;
   height=6;
   cout<<”Constructor Called”;
 }
};
int main()
{
  Rectangle r1;
   return 0;
}
```

Types of Constructors

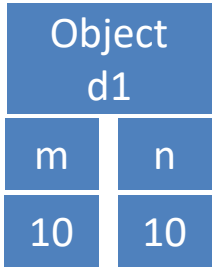
1) Default constructor
2) Parameterized constructor
3) Copy constructor

1) Default Constructor

- **Default constructor** is the one which is ***invoked by default*** when object of the class is created.
- It is generally used to ***initialize the default value*** of the data members.
- It is also called **no argument constructor**.

```
class demo{
  int m,n;
public:
  demo()
  {
    m=n=10;
  }
};
```

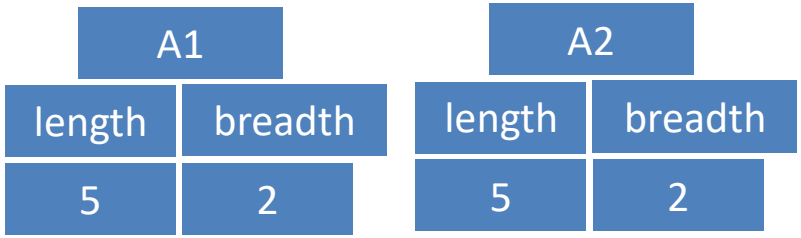
```
int main()
{
  demo d1;
}
```



Program Constructor

```
class Area
{
  private:
   int length, breadth;
  public:
   Area(){
   length=5;
   breadth=2;
   }
   void Calculate(){
    cout<<"\narea="<<length * breadth;
   }
};
```

```
int main(){
 Area A1;
 A1.Calculate();
 Area A2;
 A2.Calculate();
 return 0;
}
```



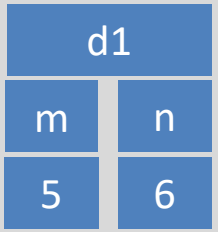
2) Parameterized Constructor

- Constructors that can take arguments are called **parameterized constructors.**
- Sometimes it is necessary ***to initialize the various data elements of different objects*** with ***different values*** when they are created.
- We can achieve this objective by ***passing arguments to the constructor function*** when the objects are created.

Parameterized Constructor

- Constructors that can take arguments are called **parameterized constructors**.

```
class demo
{
   int m,n;
   public:
   demo(int x,int y){ //Parameterized Constructor
     m=x;
     n=y;
     cout<<“Constructor Called“;
   }
};
int main()
{

}
```



3) Copy Constructor

- A **copy constructor** is used to declare and initialize an object from another object using an object as argument.
- For example:

```
demo(demo &d);   //declaration
demo d2(d1);          //copy object
```

OR

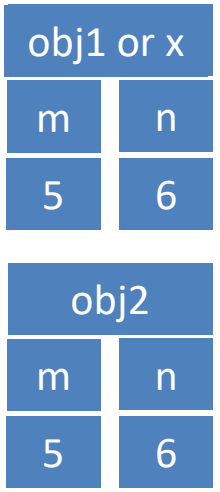
```
demo d2=d1;          //copy object
```

- Constructor which accepts a reference to its own class as a parameter is called **copy constructor**.

Copy Constructor

```
class demo
{
    int m, n;
    public:
    demo(int x,int y){
     m=x;
     n=y;
     cout<<"Parameterized Constructor";
    }
    demo(demo &x){
     m = x.m;
     n = x.n;
     cout<<"Copy Constructor";
    }
};
```

```
int main()
{
    demo obj1(5,6);
    demo obj2(obj1);
    demo obj2 = obj1;
}
```



```
3   class Rectangle
4   {
5       int length, width;
6    public:
7       Rectangle()
8       {
9         length=0; width=0;
10      }
11      Rectangle(int x, int y)
12      {
13        length = x;
14        width = y;
15      }
16     Rectangle(Rectangle &_r)
17      {
18        length = _r.length;
19        width = _r.width;
20      }
21      void show();
22  };
```

```
23  int main()
24  {
25      Rectangle r1;
26      Rectangle r2(10,20);
27      Rectangle r3(r2);
28      r1.show();
29      r2.show();
30      r3.show();
31  }
```

OUTPUT:

```
r1:  0   0
r2:  10  20
r3:  10  20
```

```
 3  class Test
 4  {
 5      int  a,  b;
 6  public:
 7      Test();
 8      Test( int );
 9      Test( int , int );
10      Test( Test & );
11      void show();
12  };
13
14  Test :: Test()
15  {
16      a = b = 0;
17  }
```

```
19  Test :: Test( int  p ):a(p),b(p)
20  {
21      //a = b = p;
22  }
23
24  Test :: Test( int  x, int  y ):a(x),b(y)
25  {
26      //a = x ;  b = y;
27  }
28
29  Test :: Test(  Test &T  )
30  {
31      a  =  T.a;
32      b  = T.b;
33  }
34  void Test :: show()
35  {
36      // Display statements
37  }
```

```
38  int  main( )
39  {
40      Test  T1 , T5;
41      Test  T2( 5 );
42      Test  T3( 10 ,20 );
43      Test  T4 (T3) ;
44      T5 = T2;
45  }
```

```
class Counter
   {
   private:
      unsigned int count;
   public:
      Counter() : count(0)
         { /*empty body*/ }
      void inc_count()
         { count++; }
      int get_count()
         { return count; }
   };
////////////////////////////////
```

```
int main()
   {
   Counter c1, c2;

   cout << "\nc1=" << c1.get_count();
   cout << "\nc2=" << c2.get_count();

   c1.inc_count();
   c2.inc_count();
   c2.inc_count();

   cout << "\nc1=" << c1.get_count();
   cout << "\nc2=" << c2.get_count();
```

Note: If multiple members must be initialized, they're separated by commas.

```
someClass() : m1(7), m2(33), m2(4)
   {  }
```

← initializer list

Example-3

```
3   class TEST
4   {
5   public:
6       TEST( )
7       {
8           cout<<"Def.constr called..\n";
9       }
10  };
11  int  main()
12  {
13      TEST  T[5];
14      return 0;
15  }
```

OUTPUT:

```
Def.constr called..
Def.constr called..
Def.constr called..
Def.constr called..
Def.constr called..
```

```
class Distance                          //English Distance class
   {
   private:
      int feet;
      float inches;
   public:
                                        //constructor (no args)
      Distance() : feet(0), inches(0.0)
         {  }
      //Note: no one-arg constructor
                                        //constructor (two args)
      Distance(int ft, float in)  : feet(ft), inches(in)
         {  }

      . . . . . .

int main()
   {
   Distance dist1(11, 6.25);      //two-arg constructor
   Distance dist2(dist1);         //one-arg constructor
   Distance dist3 = dist1;        //also one-arg constructor
```

Destructor

- **Destructor** is used to destroy the objects that have been created by a constructor.
- The syntax for **destructor** is same as that for the constructor,
  - the class name is used for the name of destructor,
  - with a **tilde (~)** sign as prefix to it.

```
class car
{
    float mileage;
  public:
   car(){
     mileage=0;
   }

   ~car(){
cout<<" destructor";
   }

};
```

Destructor

- never takes any argument nor it returns any value nor it has return type.
- is invoked automatically by the complier upon exit from the program.
- should be declared in the public section.

Program: Destructor

```
class rectangle
{
  int length, width;
  public:
  rectangle(){ //Constructor
   length=0;
   width=0;
   cout<<"Constructor Called";
  }
  ~rectangle() //Destructor
  {
   cout<<"Destructor Called";
  }
// other functions for reading, writing and
processing can be written here
};
```

```
int main()
{
   rectangle x;
// default
constructor is
called
}
```

Program: Destructor

```
class Marks{
public:
   int maths;
   int science;
   //constructor
   Marks() {
      cout << "Inside Constructor"<<endl;
      cout << "C++ Object created"<<endl;
   }
  //Destructor
   ~Marks() {
      cout << "Inside Destructor"<<endl;
      cout << "C++ Object destructed"<<endl;
   }
};
```

```
int main( )
{
   Marks m1;
   Marks m2;
   return 0;
}
```

Destructor Example-2:

```
3   int cnt=0;
4   class Test
5   {
6       int m;
7   public:
8       Test()
9       {
10          cnt++;
11          m = cnt;
12          cout<<"\n Object created: "<<m;
13      }
14      ~Test()
15      {
16          cout<<"\nDestructor for object: "<<m;
17      }
18  };
```

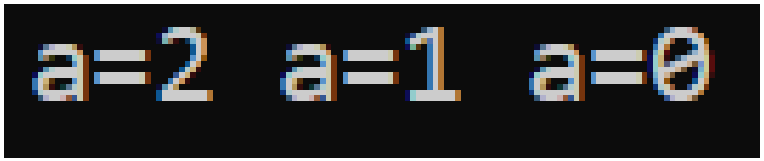
```
20  int main()
21  {
22      Test T1,T2,T3;
23      return 0;
24  }
```

```
 Object created: 1
 Object created: 2
 Object created: 3
Destructor for object: 3
Destructor for object: 2
Destructor for object: 1
```

Question

```
3   int count = 0;
4   class Test
5   {
6       int  a;
7   public:
8       Test()
9       {
10          a = count;
11          count++;
12      }
13      ~Test()
14      {
15          cout<<" a="<<a;
16      }
17  };
```

```
18  int  main( )
19  {
20      Test  T1, T2 , T3;
21      return 0;
22  }
```



Question

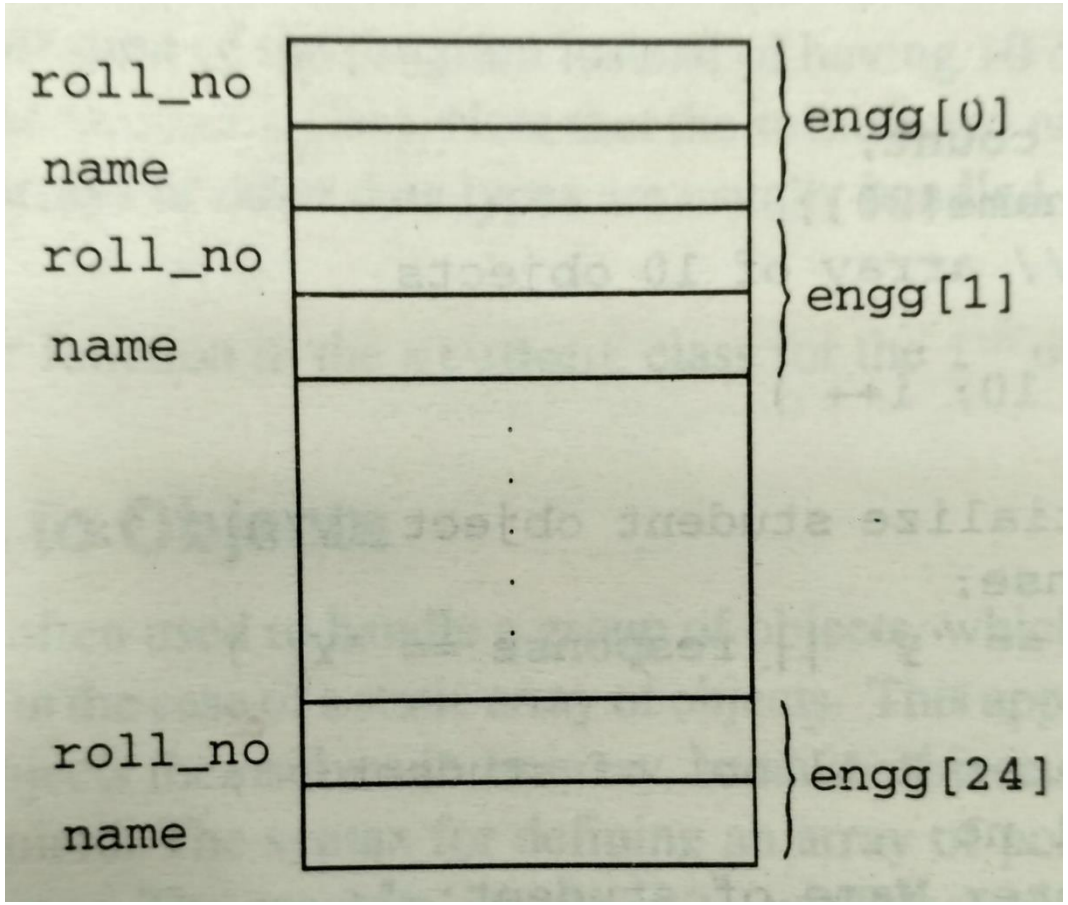
```
3  class A
4  {
5  public:
6      A()
7      {
8          cout<<"\n Constructor called.";
9      }
10     ~A()
11     {
12         cout<<"\n Destructor called.";
13     }
14 };

16 int main()
17 {
18     A a1, a2;
19     {
20         A a3;
21     }
22     return 0;
23 }
```

OUTPUT:

```
Constructor called.
Constructor called.
Constructor called.
Destructor called.
Destructor called.
Destructor called.
```

ARRAY OF OBJECTS

```
1  class student
2  {
3      int roll_no;
4      char name[20];
5      ......
6  };
7
8  student engg[25];
```



Array of objects:

```
3  class ArrayObjects
4  {
5      int i,j;
6  public:
7      ArrayObjects(int x, int y)
8      {
9          i = x; j = y;
10     }
11     void show()
12     {   cout<<i<<"\t"<<j; }
13
14 };
```

```
ArrayOb[0]: 10   20
ArrayOb[1]: 11   22
ArrayOb[2]: 100 200
```

```
16 int main()
17 {
18     ArrayObjects ob[3] = { ArrayObjects(10,20),
19                            ArrayObjects(11,22),
20                            ArrayObjects(100,200) };
21     for(int i = 0; i < 3; i++ )
22     {
23         cout<<"\nArrayOb["<<i<<"]: ";
24         ob[i].show();
25     }
26 }
```

Example: Objects as function argument

```
3   class Weight
4   {
5       int KG , gram;
6   public:
7       Weight() : KG(0), gram(0)
8       {  }
9
10      Weight(int kg, int gr ):KG(kg),gram(gr)
11      {  }
12
13      void input_weight()
14      {
15          cout <<"\nEnter KG: ";  cin >> KG;
16          cout << "Enter gram: ";  cin >> gram;
17      }
18      void show()                 //display Weight
19      { cout<<KG <<"KG+"<<gram<<"g"; }
20
21   void add_weight( Weight,Weight );
22  };
```

```
23  void Weight::add_weight( Weight  W1,Weight  W2 )
24  {
25      gram = W1.gram + W2.gram;
26      KG = W1.KG + W2.KG;;
27      if (gram >= 1000 )
28      {
29          KG +=  gram / 1000;
30          gram = gram % 1000;
31      }
32  }
33  int main()
34  {
35      Weight w1(2,500), w2, w3;
36      w2.input_weight();
37
38      w3.add_weight( w1 , w2 );
39
40      cout<<"\n w1:"; w1.show();
41      cout<<"\n w2:"; w2.show();
42      cout<<"\n w3:"; w3.show();
43  }
```

```
Enter KG: 4
Enter gram: 600

 w1:2KG+500g
 w2:4KG+600g
 w3:7KG+100g
```

Example-: Returning an object

```
3  class Weight
4  {
5      int KG , gram;
6  public:
7      Weight() : KG(0), gram(0)
8      {  }
9
10     Weight(int kg, int gr )  : KG(kg), gram(gr)
11     {  }
12
13     void input_weight()
14     {
15         cout <<"\nEnter KG: ";  cin >> KG;
16         cout << "Enter gram: ";  cin >> gram;
17     }
18     void show()              //display Weight
19     { cout<<KG <<"KG+"<<gram<<"g"; }
20
21     Weight add_weight( Weight );
22 };
```

```
23  Weight Weight::add_weight( Weight W )
24   {
25      Weight S;
26
27      S.gram = gram + W.gram;
28      S.KG = KG + W.KG;;
29      if (S.gram >= 1000 )
30      {
31          S.KG +=  S.gram / 1000;
32          S.gram = S.gram % 1000;
33      }
34      return S;
35   }
36  int main()
37  {
38       Weight w1(2,500); Weight w2, w3;
39       w2.input_weight();
40
41       w3 = w1.add_weight( w2 );
42
43       cout<<"\n w1:"; w1.show();
44       cout<<"\n w2:"; w2.show();
45       cout<<"\n w3:"; w3.show();
46  }
```

Constant member function

```
3  class Test
4  {
5      int  alpha;
6  public:
7
8      Test()
9      { alpha = 10; }
10
11     void any_function()
12     {
13         alpha = 25;
14     }
15     void const_function() const
16     {
17         alpha++; // Error
18     }
19 };
```

```
20  int main()
21  {
22      Test T;
23      T.any_function();
24      T.const_function();
25      return 0;
26  }
```

Constant member function...

```
3  class TEST
4  {
5      int  alpha;
6  public:
7
8      void constFunction()   const    //const member function
9      {
10         int x = 11;
11         x++;
12     }
13 };
14 int main()
15 {
16     TEST T;
17     T.constFunction();
18     return 0;
19 }
```

Constant objects

```
3  class Test
4  {
5      int  alpha;
6  public:
7
8      Test()
9      { alpha = 10; }
10
11     void any_function()
12     {
13         alpha = 25;
14     }
15     void const_function() const
16     {
17         cout<<alpha;
18     }
19 };
20 int main()
21 {
22     const Test T;
23     //T.any_function(); Error
24     T.const_function();
25     return 0;
26 }
```

Constant member function argument

```
3   class Test
4   {
5       int  a;
6   public:
7       Test()
8       { a = 10; }
9
10      void const_function_arg(const Test &T )
11      {
12          //T.a++; Error
13          a++;
14      }
15  };
16  int main()
17  {
18      Test T1, T2 ;
19      T1.const_function_arg( T2 );
20      return 0;
21  }
```

```
3  class Test
4  {
5      int  a;
6  public:
7      Test()
8      { a = 10; }
9
10     void const_function_arg(const Test &T ) const
11     {
12         //T.a++; Error
13         // a++; Error
14     }
15 };
16 int main()
17 {
18     Test T1, T2 ;
19     T1.const_function_arg( T2 );
20 }
```

Question-1:

```
3   class TEST
4   {
5       int  alpha;
6   public:
7
8       TEST() : alpha(10) { }
9
10      void show()
11      {
12          cout<<alpha;
13      }
14  };
15  int main()
16  {
17      const TEST T;
18      T.show();
19      return 0;
20  }
```

Error !

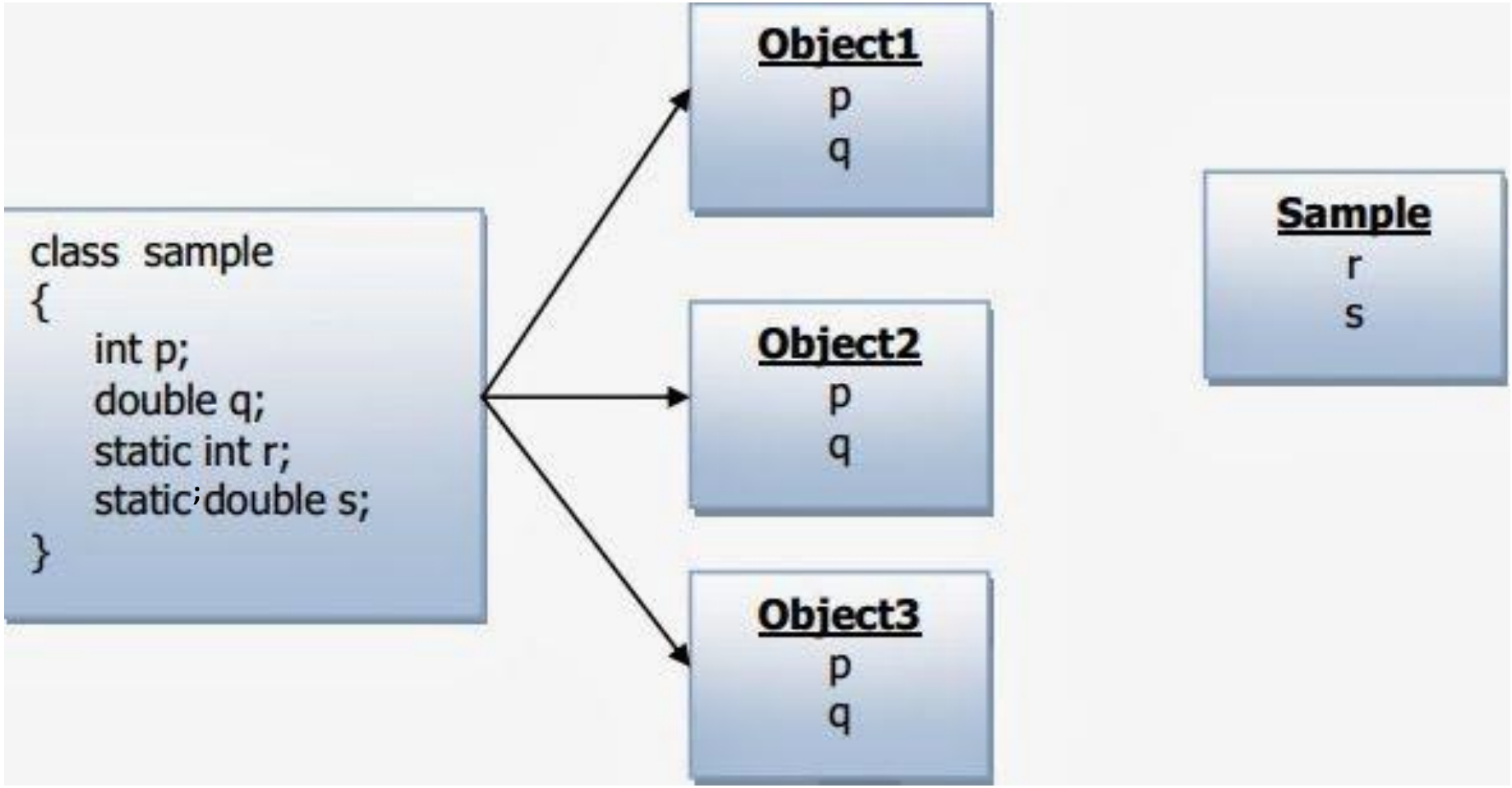
Question-2: Will it compile?

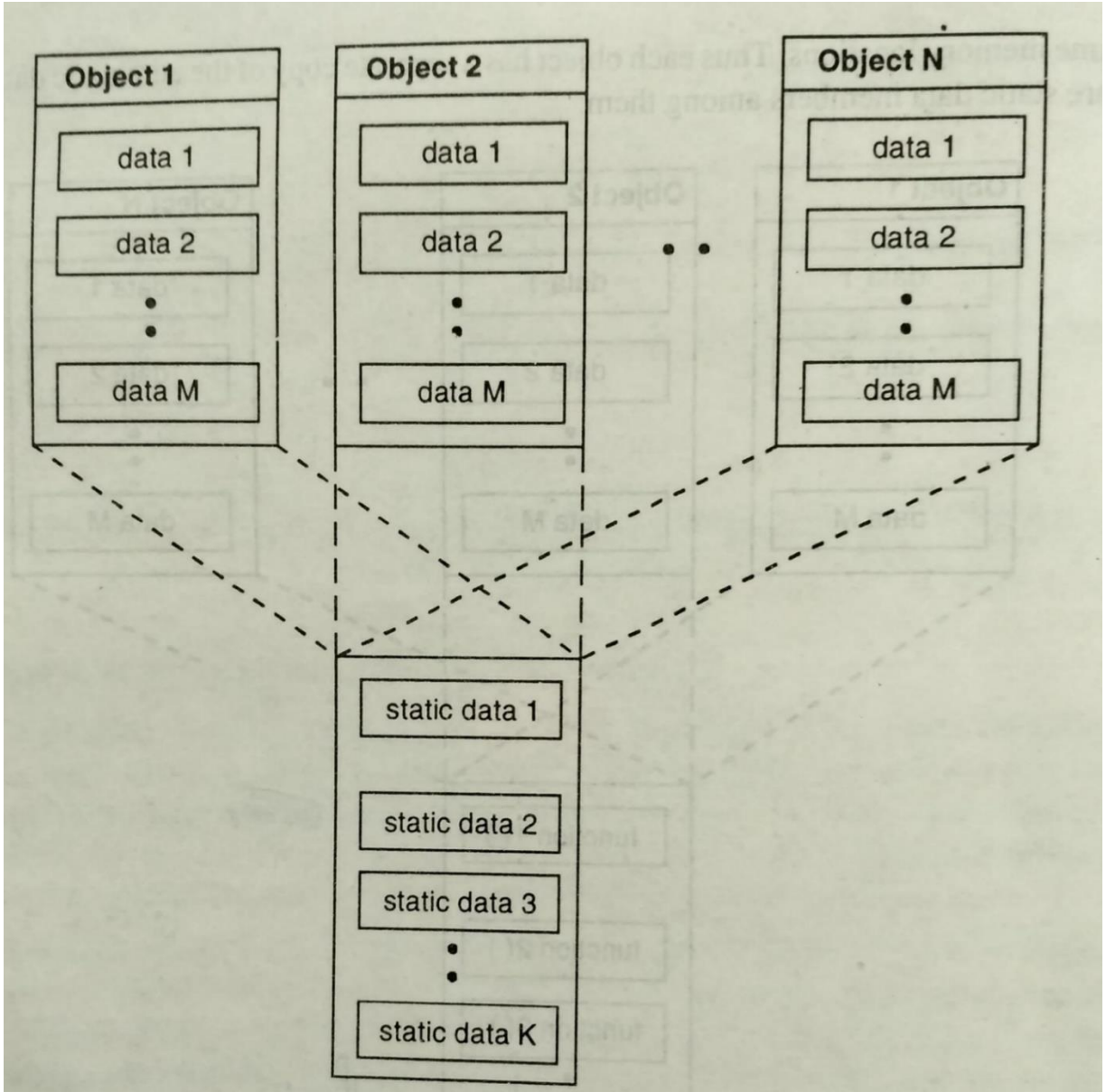
```
3   class Test
4   {
5       int  a;
6   public:
7       Test()
8       { a = 10; }
9
10      void const_function(Test &T ) const
11      {
12          T.a++;
13      }
14  };
15  int main()
16  {
17      Test T1, T2 ;
18      T1.const_function( T2 );
19      return 0;
20  }
```

Static Data members

A static data member is useful,
when all objects of the same class must **share a common information**.

- Just write static keyword prefix to regular variable
- It is initialized to zero when first object of class created
- Only one copy is created for each object
- Its life time is entire program

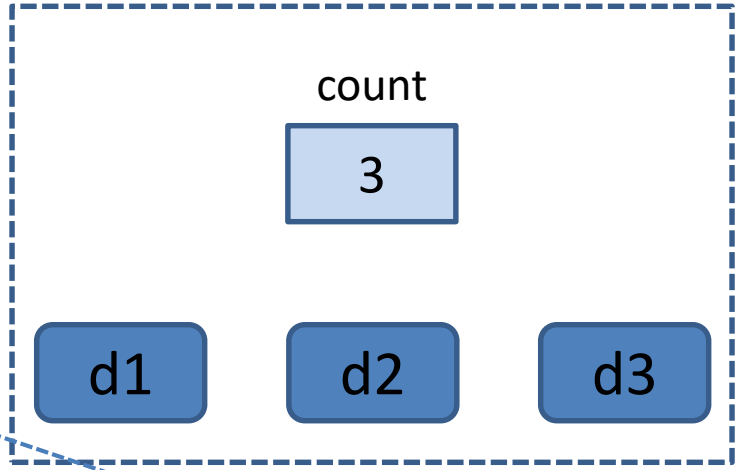

class sample
{
int p;
double q;
static int r;
static double s;
}
Object1
p
q
Object2
p
q
Object3
p
q
Sample
r
s



Separate memory for class's automatic data members
Shared memory for class's static data members

Static Data members

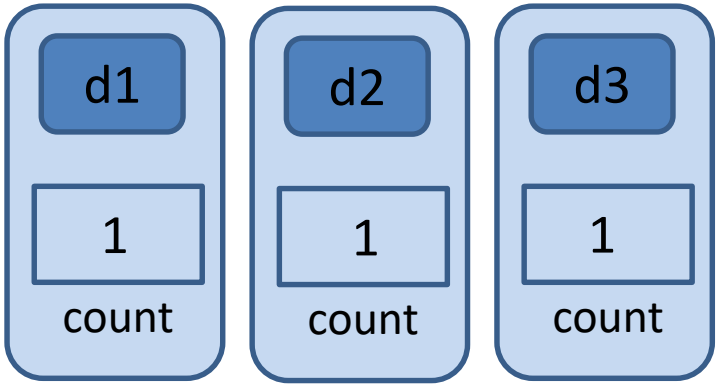
```
class demo
{
  static int count;
  public:
   void getcount()
   {
        cout<<"count="<<++count;
   }
};

int demo::count;

int main()
{
  demo d1,d2,d3;
  d1.getcount();
  d2.getcount();
  d3.getcount();
  return 0;
}
```



Static members are **declared** inside the class and **defined** outside the class.

```
class demo
{
  int count;

  public:
    void getcount()
    {
      count = 0;
      cout<<"count="<< ++count;
    }
};
int main()
{
  demo d1,d2,d3;
  d1.getcount();
  d2.getcount();
  d3.getcount();
  return 0;
}
```

Regular Data members



Static Data Members

- Data members of the class which are shared by all objects are known as **static** data members.
- **Only one copy** of a static variable is maintained by the class and it is common for all objects.
- **Static members** are **declared** inside the class and **defined** outside the class.
- It is initialized to **zero** when the first object of its class is created.
- you cannot initialize a static member variable inside the class declaration.
- It is visible only within the class but its lifetime is the entire program.
- **Static members** are generally used to maintain values common to the entire class.

Program : Static data member

```
3  class Static_Demo
4  {
5      static int a;
6      int b;
7  public:
8      void set(int i, int j)
9      {  a=i;  b=j;  }
10     void show();
11 };
12
13 int Static_Demo::a; // define a
15 void Static_Demo::show()
16 {
17     cout << "\n This is static a: " << a;
18     cout << "\n This is non-static b: " << b;
19 }
20 int main()
21 {
22     Static_Demo X, Y;
23     X.set(1, 1);
24     cout<<"\n Object X:";    X.show();
25     Y.set(2, 2);
26     cout<<"\n Object Y:";    Y.show();
27     cout<<"\n Object X:";    X.show();
28 }
```

```
Object X:
This is static a: 1
This is non-static b: 1
Object Y:
This is static a: 2
This is non-static b: 2
Object X:
This is static a: 2
This is non-static b: 1
```

```
3  class Static_Demo
4  {
5      static int a;
6      int b;
7  public:
8      Static_Demo()
9      {
10         a = 0; b = 0;
11     }
12     void incr()
13     {
14         a++; b++;
15     }
16     void show();
17 };
18 int Static_Demo::a; // define a
20 int main()
21 {
22     Static_Demo X, Y;
23     X.incr();
24     cout<<"\n Object X:";    X.show();
25     Y.incr();
26     cout<<"\n Object Y:";    Y.show();
27     cout<<"\n Object X:";    X.show();
28 }
```

```
Object X:
This is static a: 1
This is non-static b: 1
Object Y:
This is static a: 2
This is non-static b: 1
Object X:
This is static a: 2
This is non-static b: 1
```

```
 3  class Counter
 4  {
 5  public:
 6      static int count;
 7      Counter()
 8      {
 9          cout << "\nObjects in existence: obj"<< count;
10          count++;
11      }
12      ~Counter()
13      {
14          count--;
15          cout << "\nObjects destroying: obj"<< count;
16      }
17  };
```

```
19  int Counter::count;
20
21  int main()
22  {
23      Counter obj1;
24      Counter obj2;
25      return 0;
26  }
```

OUTPUT

```
Objects in existence: obj0
Objects in existence: obj1
Objects destroying: obj1
Objects destroying: obj0
```

Static Member Functions

Static Member Functions

- **Static member functions** can access only static members of the class.
- **Static member functions** can be invoked using class name, not object.
- There cannot be static and non-static version of the same function.

- A static member function does not have **this pointer**.

```
3   class static_type
4   {
5       static int i;
6   public:
7       static void init(int x)
8       {
9           i = x;
10      }
11      void show()
12      {
13          cout << i;
14      }
15  };
```

```
17  int static_type::i; // define i
18
19  int main()
20  {
21      static_type::init(100);
22      static_type x;
23      x.show();
24      return 0;
25  }
```

OUTPUT

```
100
```

```
 3  class static_type
 4  {
 5      int k;
 6      static int i;
 7  public:
 8      static void init(int x)
 9      {
10          i = x;
11          k = x; // Error: k is non-static
12      }
13      void show()
14      {
15          cout << i;
16      }
17  };
```

Question:

```
E1:100,27,40000
E2:101,35,75000
E3:102,40,90000
```

```
23  int main()
24  {
25      Emp E1(27 , 40000);
26      Emp E2(35,75000);
27      Emp E3(40 ,90000);
28      cout<<"\n E1:"; E1.show();
29      cout<<"\n E2:"; E2.show();
30      cout<<"\n E3:"; E3.show();
31  }
```

Create Emp class with data members for storing emp id , age , salary.

solution

```
4  class Emp
5  {
6      static int id;
7      int emp_id;
8      double salary;
9      int age;
10 public:
11
12     Emp( int a ,double sal )
13     {
14         emp_id = id;
15         id++;
16         age = a;
17         salary = sal;
18     }
19     void show()
20     { cout<<emp_id<<","<<age<<","<<salary; }
21 };
22 int Emp::id = 100;
```

Friend function

- ***A friend function*** is not in the scope of the class, in which it has been declared as friend.
- It cannot be called using the object of that class.
- It can be invoked like a normal function without any object.
- Unlike member functions, it cannot use the member names directly.
- It can be declared in ***public or private part*** without affecting its meaning.
- Usually, it has objects as arguments.

Example1: Friend function

```
3   class myclass
4    {
5       int a, b;
6   public:
7       friend  int  sum( myclass );
8       void set_ab()
9       {
10          a = 20 ; b = 25;
11      }
12  };
13
14  int  sum( myclass  x )
15  {
16      return x.a  +  x.b;
17  }
```

```
18  int main()
19  {
20      myclass n;
21      n.set_ab();
22      cout << sum( n );
23      return 0;
24  }
```

Output

```
Sum of data members: 45
```

Example 2: Friend function with two classes

```
3   class  beta;    // Forward reference
4   class  alpha     // First class
5   {
6       int data;
7   public:
8       alpha()
9       { data = 3;  }
10
11      friend int frn_fn ( alpha , beta );
12  };
13  class  beta    // Second class
14  {
15      int data;
16  public:
17      beta()
18      { data = 7;  }
19
20      friend  int  frn_fn ( alpha, beta );
21  };
```

```
22  int frn_fn(alpha a, beta b )
23  {
24      return( a.data + b.data );
25  }
```

```
26  int  main()
27  {
28      alpha aa;
29      beta bb;
30      cout << frn_fn( aa, bb ) ;
31      return  0;
32  }
```

```
Friend function output
10
```

Question: Replace member function by friend function

```
3   class TEST
4   {
5       int d1, d2;
6   public:
7       void setdata()
8       {
9           d1 = 10;
10          d2 = 20;
11      }
12      void showdata()
13      {
14          cout<<"d1="<<d1<<","
15              <<"d2="<<d2<<"\n";
16      }
17  };
```

```
18  int main()
19  {
20      TEST T;
21      T.setdata();
22      T.showdata();
23      return 0;
24  }
```

Solution.

```
3  class TEST
4  {
5      int d1, d2;
6  public:
7      friend void setdata( TEST& );
8      friend void showdata( TEST );
9  };
```

```
11 void setdata(TEST &A)
12 {
13         A.d1 = 10;
14         A.d2 = 20;
15 }
16
17 void showdata( TEST A )
18 {
19     cout<<"d1="<<A.d1<<","
20         <<"d2="<<A.d2<<"\n";
21 }
```

```
22 int main()
23 {
24     TEST T;
25     setdata( T );
26     showdata( T );
27     return 0;
28 }
```

Question: Define the friend function

```
3  class Weight
4  {
5      int KG , gram;
6  public:
7      Weight() : KG(0), gram(0)
8      {  }
9
10     Weight(int kg, int gr ):KG(kg),gram(gr)
11     {  }
12
13     void input_weight()
14     {
15         cout <<"\nEnter KG: ";  cin >> KG;
16         cout << "Enter gram: ";  cin >> gram;
17     }
18     void show()                 //display Weight
19     { cout<<KG <<"KG+"<<gram<<"g"; }
20
21     // friend function for adding 2 weights
22 };
```

Solution:

```
3   class Weight
4   {
5       int KG , gram;
6   public:
7       Weight() : KG(0), gram(0)
8       {  }
9
10      Weight(int kg, int gr ):KG(kg),gram(gr)
11      {  }
12
13      void input_weight()
14      {
15          cout <<"\nEnter KG: ";  cin >> KG;
16          cout << "Enter gram: ";  cin >> gram;
17      }
18      void show()                 //display Weight
19      { cout<<KG <<"KG+"<<gram<<"g"; }
20
21      friend Weight add_weight( Weight&,Weight& );
22  };
```

```
24  Weight add_weight( Weight  &W1,Weight  &W2 )
25   {
26       Weight sum;
27       sum.gram = W1.gram + W2.gram;
28       sum.KG = W1.KG + W2.KG;;
29       if (sum.gram >= 1000 )
30       {
31           sum.KG +=  sum.gram / 1000;
32           sum.gram = sum.gram % 1000;
33       }
34       return sum;
35   }
36  int main()
37  {
38      Weight w1(2,500), w2, w3;
39      w2.input_weight();
40
41      w3 = add_weight( w1 , w2 );
42
43      cout<<"\n w1:"; w1.show();
44      cout<<"\n w2:"; w2.show();
45      cout<<"\n w3:"; w3.show();
46  }
```