CLASSES & OBJECTS

Class Specification:

- A class is a collection of objects. The keyword 'class' is used to define a class template.
- A class is a grouping of variables of different data types with function.
- Each variable of a class is called as member variable(data member) and function are called as member function.
- An Object is an instance of a Class. When a class is defined, no memory is allocated but when it is instantiated (i.e. an object is created) memory is allocated.

```
class Class_Name
{
access specifier:
data member;
data member;
access specifier:
member function();
member function()
{
//function body;
}
};
void main()
{
Class_Name Object_Name;
}
```

Access Specifier in c++:

1) **Public keyword:** The public members of a class can be accessed from anywhere in the program using the direct member access operator (.) with the object of that class.
2) **Private keyword:** The class members declared as 'private' can be accessed only by the member functions inside the class. Only the member functions or the friend functions are allowed to access the private data members of a class.
3) **Protected keyword:** It is same as private. It is frequently used in inheritance

Member Functions in Classes

There are 2 ways to define a member function:

- Inside class definition
- Outside class definition

```
#include<iostream.h>
#include<conio.h>
class A
{
public:
int sum;
void Add(int num1, int num2)
{
sum=num1+num2;
cout<<"sum="<<sum;
}
void Sub(int num1,int num2);
};
// Definition outside class using ::
void A::Sub(int num1,int num2)
{
int subtract;
subtract=num1-num2;
cout<<"subtract="<<subtract;
}
void main()
{
A obj1;
int x,y;
cout<<"enter the values";
cin>>x;
cin>>y;
obj1.Add(x,y);
obj1.Sub(x,y);
getch();
}
```

Static data member in C++

When we declare a normal variable (data member) in a class, different copies of those data members create with the associated objects.

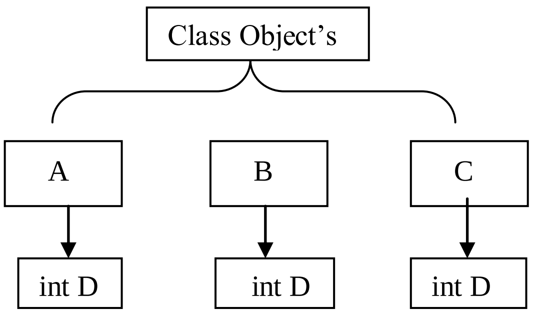


Fig: Normal variable with the associated class objects.

It is a variable which is declared with the 'static' keyword, it is also known as class member, thus only single copy of the variable creates for all objects. The static data members is initialized with zero when the first object of its class is created.

Any changes in the static data member through one member function will reflect in all other object's member functions.

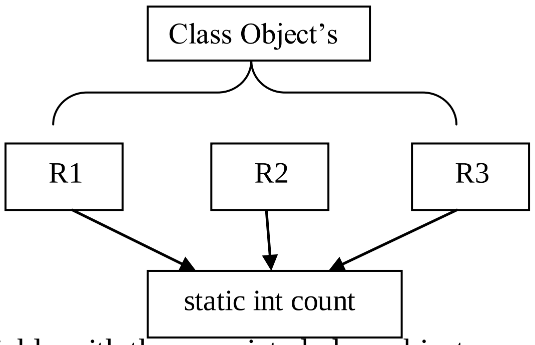


Fig: static variable with the associated class objects.

```
class Rectangle
{
static int count;
int area();
};
void main()
{
Rectangle  R1;
Rectangle  R2;
Rectangle  R3;
}
```

Declaration
static data_type variable;
Defining the static data member
It should be defined outside of the class following this syntax:
data_type class_name :: variable =value;

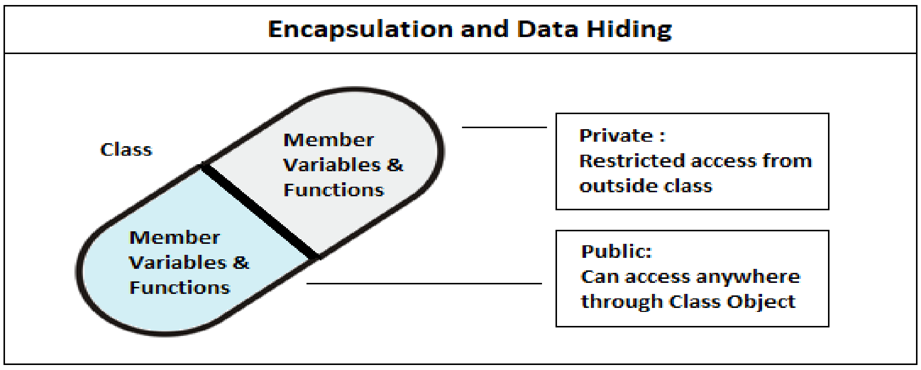
Static member function

When a function is defined as static it can access only static member variables & functions of same class. The non-static member are not available to these function.

```
#include <iostream.h>
class Demo
{
private:
static int X;
public:
static void fun()
{
cout <<"Value of X: " << X << endl;
}
};
//defining
int Demo :: X =10;
int main()
{
Demo X;
cout << X::count << endl; // use static member of class X
X.fun();
return 0;
}
```

If you are calling a static data member within a member function, member function should be declared as static (i.e. a static member function can access the static data members)

Difference Between Data Hiding and Encapsulation:


Encapsulation and Data Hiding
Class
Member Variables & Functions
Member Variables & Functions
Private : Restricted access from outside class
Public: Can access anywhere through Class Object

Data Hiding	Encapsulation
Data hiding concern about data security along with hiding complexity.	Encapsulation concerns about wrapping data to hide the complexity of a system.
Data Hiding focuses on restricting or permitting the use of data inside the capsule.	Encapsulation focuses on enveloping or wrapping the complex data.
The data under data hiding is always private and inaccessible.	The data under encapsulation may be private or public.
Data hiding is a process as well as technique.	Encapsulation is a sub-process in data hiding.

Empty Classes in C++:

- Empty class means, a class that does not contain any data members and members function.
- An Empty class's object will take only one byte in the memory. One byte is the minimum memory amount that could be occupied.
- The reason when we create an object of an empty class in C++ program, it needs some memory to get stored, and the minimum amount of memory that can be reserved is 1 byte. So, if we create multiple objects of an empty class, every object will have unique address.

```
#include <iostream.h>
class Empty_class
{

};
void main()
{
Empty_class objEmpty;
cout<<"Size of empty object is: "<<sizeof(objEmpty)<<endl;
}
```

Output:

```
Size of empty object is: 1
```

Friend Function in C++

- A friend function of a class is defined outside that class' scope but it has the right to access all private and protected members of the class. Even though the prototypes for friend functions appear in the class definition, friends are not member functions.

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

```
class className {
    ... .. ...
    friend returnType functionName(arguments);
    ... .. ...
}
```

Example :

```
// C++ program to demonstrate the working of friend function

#include <iostream.h>

class Distance {
    private:
        int meter;
        // friend function
        friend int addFive(Distance);
};
```

```
// friend function definition
int addFive(Distance d) {

    //accessing private members from the friend function
    d.meter = 5;
    return d.meter;
}

int main() {
    Distance D;
    cout << "Distance: " << addFive(D);
    return 0;
}
```

Output

```
Distance: 5
```

Characteristics of a Friend function:

- The function is not in the scope of the class to which it has been declared as a friend.
- It cannot be called using the object as it is not in the scope of that class.
- It can be invoked like a normal function without using the object.
- It can be declared either in the private or the public part.

Friend Class in C++

We can also use a friend Class in C++ using the friend keyword.
For example,

```
class ClassB;

class ClassA {
  // ClassB is a friend class of ClassA
  friend class ClassB;
  ... .. ...
}

class ClassB {
  ... .. ...
}
```

When a class is declared a friend class, all the member functions of the friend class become friend functions.

Example : C++ friend Class

```
// C++ program to demonstrate the working of friend class
#include <iostream.h>
// forward declaration
class ClassB;
class ClassA {
   private:
      int numA;

      // friend class declaration
      friend class ClassB;

   public:
      // constructor to initialize numA to 12
      ClassA() : numA(12) {}
};

class ClassB {
   private:
      int numB;
   public:
      // constructor to initialize numB to 1
      ClassB() : numB(1) {}

   // member function to add numA
   // from ClassA and numB from ClassB
   int add() {
      ClassA objectA;
```

```
        return objectA.numA + numB;
    }
};

int main() {
    ClassB objectB;
    cout << "Sum: " << objectB.add();
    return 0;
}
```

Output

```
Sum: 13
```

Constructor in C++:

- A constructor in C++ is a special method that is automatically called when an object of a class is created.
- To create a constructor, use the same name as the class, followed by parentheses ()
- It used to construct & initialize all the data members.

```
class A{
int x;
Public:
A();   //constructor
};
```

Types of constructor:
1) default constructor
2) parameterized constructor
3) copy constructor

1) default constructor: A constructor with no parameters is known as a default constructor.

Example:

```
class cube
```

```
{
int side;
public:
cube()
{
side=10;
}
};
void main()
{
cube c;
cout<<"side="<<c.side;
}
```

Output: side=10

2) Parameterized constructor: Constructor with parameters , used for initializing data members.

Example:

```
class cube
{
int side;
public:
cube( int x)
{
side=x;
}
};
void main()
{
cube c1(10);
cube c2(20);
cout<<"side1="<<c1.side;
cout<<"side2="<<c2.side;
}
```

Output:

```
side1=10
side2=20
```

3) Copy constructor: copy constructor is used, when one object of class initialize the other object (copy data of one object to another).

Example:

```
class A
{
int a,b;
public:
```

```
A(int x, int y)
{
a=x;
b=y;
}
void display()
{
cout<<a<<b;
}
};
void main()
{
A obj1(30,40);
A obj2=obj1; // copy constructor
obj1.display();
obj2.display();
}
```

Destructors in C++ :

- Destructors are members functions in a class that delete or destroy an object that have been created by a constructor.

- destructors have the same name as their class and their name is preceded by a tilde(~).

```
class Demo {
  private:
  int num1, num2;
  public:
  Demo(int n1, int n2) {
    cout<<"Inside Constructor"<<endl;
    num1 = n1;
    num2 = n2;
  }
  void display() {
    cout<<"num1 = "<< num1 <<endl;
    cout<<"num2 = "<< num2 <<endl;
  }
  ~Demo() {
    cout<<"Inside Destructor"; }
};
void main() {
  Demo obj1(10, 20);
  obj1.display();
}
```

Output

```
Inside Constructor
```

```
num1 = 10
num2 = 20
Inside Destructor
```

Constructor Overloading:

- Overloaded constructors have the same name (name of the class) but the different number of arguments.
- A constructor is called depending upon the number and type of arguments passed.
- While creating the object, arguments must be passed to let compiler know, which constructor needs to be called.

Example:

```
// C++ program to demonstrate constructor overloading
#include <iostream.h>
class Person {
  private:
   int age;
  public:
   // 1. Constructor with no arguments
   Person() {
      age = 20;
   }
   // 2. Constructor with an argument
   Person(int a) {
      age = a;
   }
   int getAge() {
      return age;
   }
};
int main() {
   Person person1, person2(45);
   cout << "Person1 Age = " << person1.getAge() << endl;
   cout << "Person2 Age = " << person2.getAge() << endl;
   return 0;
}
```

Output

```
Person1 Age = 20
Person2 Age = 45
```

Dynamic initialization of object Using Dynamic Constructor:

- Dynamic initialization of object refers to initializing the objects at a run time
- It can be achieved by using constructors and by passing parameters to the constructors.

Dynamic Constructor:

- The constructor used for allocating the memory at runtime is known as the dynamic constructor.
- The memory is allocated at runtime using a new operator.
- A new operator is used to dynamically initialized the variable in constructor and memory is allocated on the heap.
- The memory is deallocated or free at runtime using the delete operator.

Example:

```
// C++ program to dynamically
// deallocating the memory
#include <iostream.h>
class geeks {
	into ptr;
public:
	// Default constructor
	geeks()
	{
		ptr = 10;
	}

	// Function to display the value
	void display()
	{
		cout << "Value: " << ptr << endl;
	}
};

void main()
{
	// Dynamically allocating memory
	// using new operator
	geeks* obj1 = new geeks();
	geeks* obj2 = new geeks();

	// Assigning obj1 to obj2
	obj2 = obj1;

	// Function Call
	obj1->display();
	obj2->display();

	// Dynamically deleting the memory
	// allocated to obj1
	delete obj1;
}
```

Anonymous (Nameless) Object in C++:

- In a C++ programming an object are created with names but It is still possible to create object without name such object are known as **anonymous or nameless object**.
- When constructor and destructor are called, the data members are initialize and destroyed respectively without object.

Example:

```
#include <iostream.h>
class A {
        int ptr;
public:
// Default constructor
A()
{
cout << "Constructor executed" << endl;
ptr = 10;
cout<< “ptr value=”<<ptr<<endl;
}
~A() // Destructor
{
cout << "Destructor executed" << endl;
}
};
void main()
{
new A(); //constructor
A(); //constructor
}
```

Output:

```
Constructor executed
Ptr value=10
Destructor executed
Constructor executed
Ptr value=10
Destructor executed
```

Anonymous classes in C++:

Anonymous class is a class which has no name given to it.

Example:

```
// concept of Anonymous Class
#include <iostream.h>
// Anonymous Class : Class is not having any name
class
{
```

```
int i;
public:
void setData(int x)
{
i = x;
}
void print()
{
cout << "Value for i : " << i<< endl;
}
} obj1;  // object for anonymous class

void main()
{
        obj1.setData(10);
        obj1.print();
}
```