Marwadi
education foundation

Meghnesh Jayswal

UNIT 4
Classes and Objects

Objective of this Unit

- Revisited structure in C,
- Limitation of Structure in C,
- Introduction to Class,
- Scope of class and accessing members of class,
- Class Object,
- Constructors,
- Types of constructors,
- Destructors
- Friend Function

Revisited structure in C

- We know that one of the unique feature of the C language is structures. They provide a method for packing together data of different types.

- A structure is a convenient tool for handling a group of logically related data items. It is a user define data type with a template that serves to define its data properties.

- Once the structure type has been defined, we can create variables of that type using declaration that are similar to the built in type declaration.

- For example, consider the following declaration:

```
struct student
{
char name[20];
int roll_number;
float total_marks;
};
```

  The keyword struct declare student as a new data type that can hold three fields of different data types. These field are known as structure member or elements. The identifier student, which is referred to as structure name or structure tag, can be used to create variables of type student.

Example:

```
struct student A; // C declaration
```

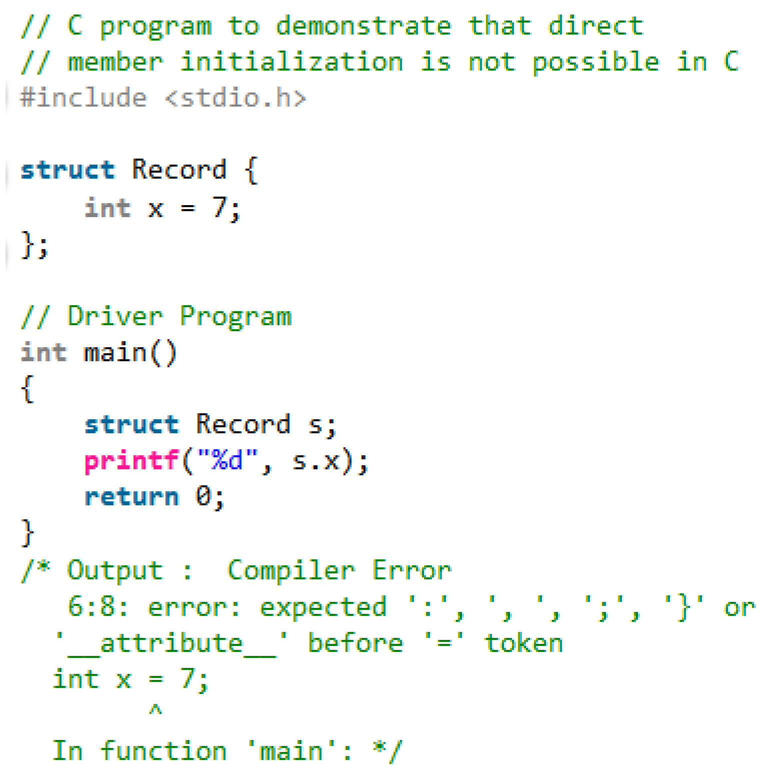
A is a variable of type student and has three member variables as defined by the template. Member variable can be accessed using the dot or period operator as fallows:

```
strcpy(A.name, "John");
A.roll_number = 999;
A.total_marks = 595.5;
final_total = A.total_marks + 5;
```

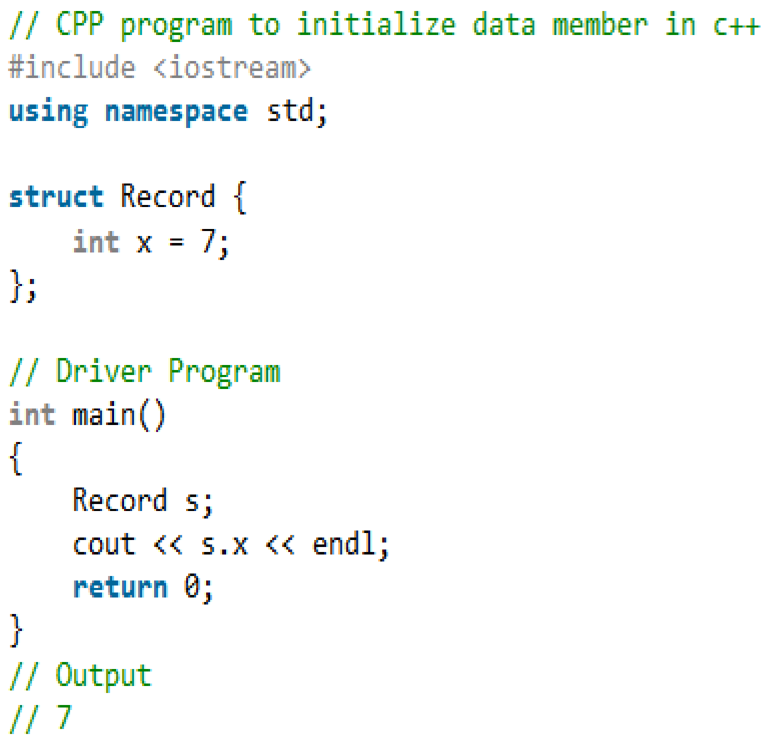
Structure can have arrays, pointer or structure as members.

Limitation of Structure in C

- **Member functions inside structure**: Structures in C cannot have member functions inside structure but Structures in C++ can have member functions along with data members.
- **Direct Initialization:** We cannot directly initialize structure data members in C but we can do it in C++.

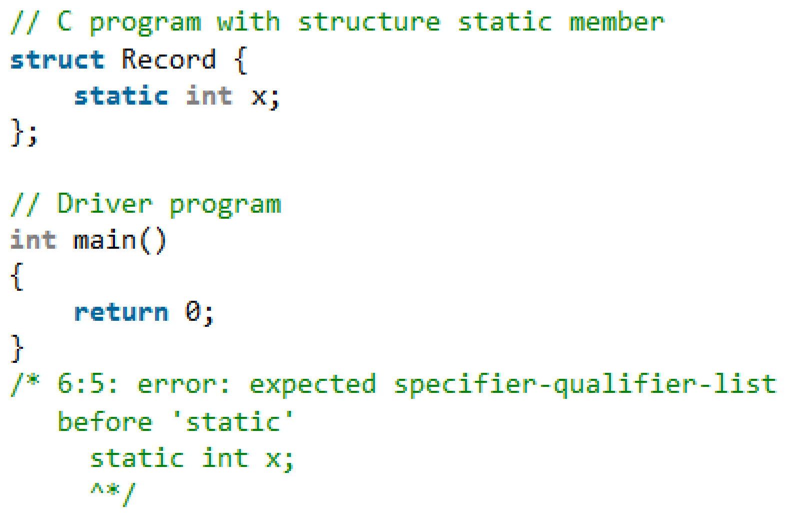


```
// C program to demonstrate that direct
// member initialization is not possible in C
#include <stdio.h>

struct Record {
    int x = 7;
};

// Driver Program
int main()
{
    struct Record s;
    printf("%d", s.x);
    return 0;
}
/* Output :  Compiler Error
   6:8: error: expected ':', ',', ';', '}' or
  '__attribute__' before '=' token
  int x = 7;
        ^
  In function 'main': */
```

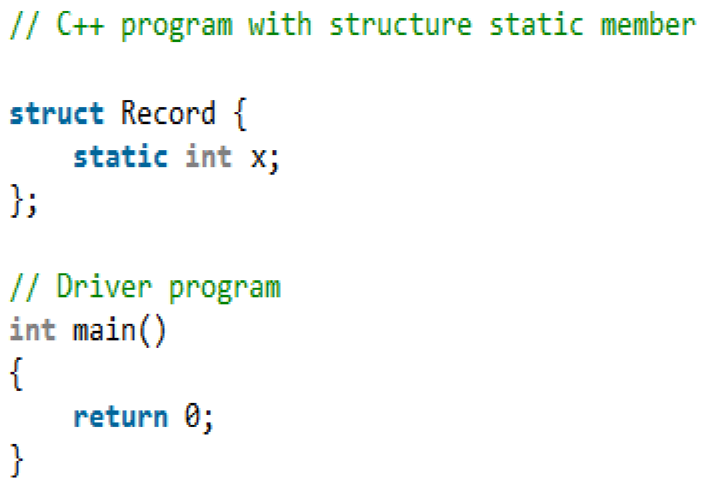


```
// CPP program to initialize data member in c++
#include <iostream>
using namespace std;

struct Record {
    int x = 7;
};

// Driver Program
int main()
{
    Record s;
    cout << s.x << endl;
    return 0;
}
// Output
// 7
```

- **Using struct keyword:** In C, we need to use struct to declare a struct variable. In C++, struct is not necessary. For example, let there be a structure for Record. In C, we must use “struct Record” for Record variables. In C++, we need not use struct and using ‘Record‘ only would work.



```
// C program with structure static member
struct Record {
    static int x;
};

// Driver program
int main()
{
    return 0;
}
/* 6:5: error: expected specifier-qualifier-list
   before 'static'
     static int x;
     ^*/
```



```
// C++ program with structure static member

struct Record {
    static int x;
};

// Driver program
int main()
{
    return 0;
}
```

- **Static Members:** C structures cannot have static members but is allowed in C++.
- **Constructor creation in structure**: Structures in C cannot have constructor inside structure but Structures in C++ can have Constructor creation.
- **Data Hiding:** C structures do not allow concept of Data hiding but is permitted in C++ as C++ is an object oriented language whereas C is not.
- **Access Modifiers:** C structures do not have access modifiers as these modifiers are not supported by the language. C++ structures can have this concept as it is inbuilt in the language**.**

Introduction to class

- **Class:** The building block of C++ that leads to Object Oriented programming is a **Class**. It is a user defined data type, which holds its own data members and member functions, which can be accessed and used by creating an instance of that class. A class is like a blueprint for an object.

- For Example: Consider the Class of **Cars**. There may be many cars with different names and brand but all of them will share some common properties like all of them will have *4 wheels*, *Speed Limit*, *Mileage range* etc. So here, Car is the class and wheels, speed limits, mileage are their properties.

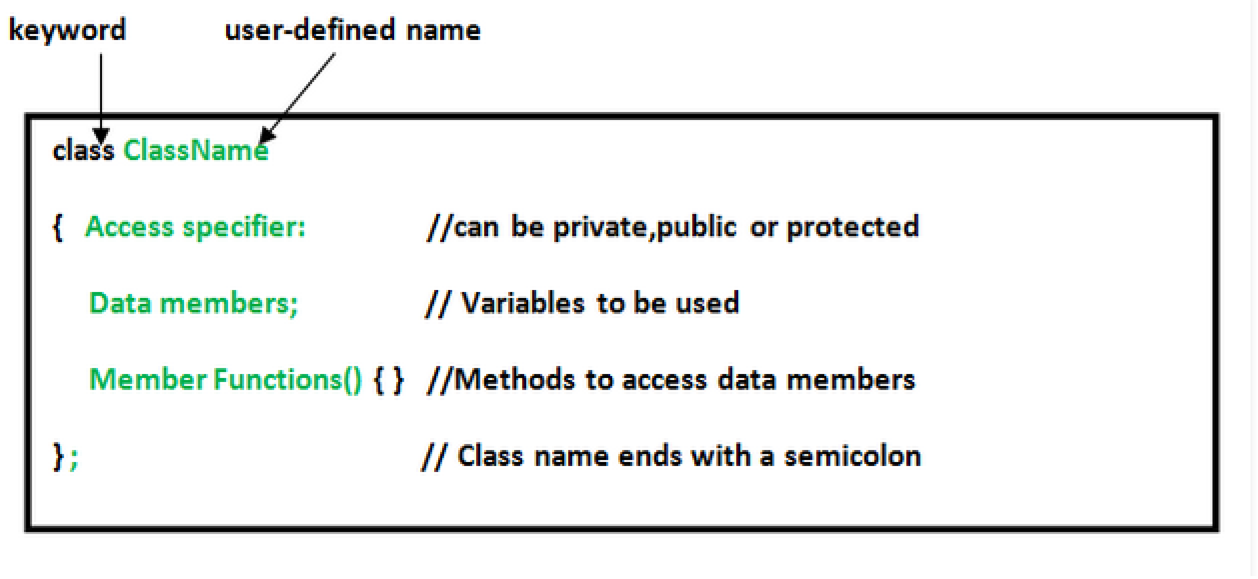
- A Class is a user defined data-type which has data members and member functions.

- Data members are the data variables and member functions are the functions used to manipulate these variables and together these data members and member functions defines the properties and behavior of the objects in a Class.

- In the above example of class *Car*, the data member will be *speed limit*, *mileage* etc. and member functions can be *apply brakes*, *increase speed* etc.

Class Object

- 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.
- **Defining Class and Declaring Objects**
- A class is defined in C++ using keyword class followed by the name of class. The body of class is defined inside the curly brackets and terminated by a semicolon at the end.



```
class ClassName

{  Access specifier:           //can be private,public or protected

   Data members;               // Variables to be used

   Member Functions() { }      //Methods to access data members

};                             // Class name ends with a semicolon
```

- **Accessing data members and member functions**: The data members and member functions of class can be accessed using the dot('.') operator with the object.
- For example if the name of object is *obj* and you want to access the member function with the name *printName()* then you will have to write *obj.printName()* .
- **Accessing Data Members**
- The public data members are also accessed in the same way given however the private data members are not allowed to be accessed directly by the object. Accessing a data member depends on the access control of that data member. This access control is given by Access modifiers in C++. There are three access modifiers : **public, private and protected**.

Access Specifier

- **Private members** of a class are accessible only from within other members of the same class or from their friends.

- **Protected members** are accessible from members of their same class and from their friends, but also from members of their derived classes.

- **Public members** are accessible from anywhere where the object is visible.

Defining Member Function

- A member function can be defined in two place in the class.

1. Inside the class
2. Outside the class

Member function Inside the class

- A member function of a class can also be defined inside the class. However, when a member function is defined inside the class, the class name and the scope resolution operator are not specified in the function header. Moreover, the member functions defined inside a class definition are by default inline functions.

```
class book
{
char title[30];
float price;
public:
void getdata(char [],float); II declaration
void putdata()//definition inside the class
{
cout<<"\nTitle of Book: "<<title;
cout<<"\nPrice of Book: "<<price;
} ;
```

Member function Outside the class

- The definition of member function outside the class differs from normal function definition, as the function name in the function header is preceded by the class name and the scope resolution operator (: :). The scope resolution operator informs the compiler what class the member belongs to. The syntax for defining a member function outside the class is

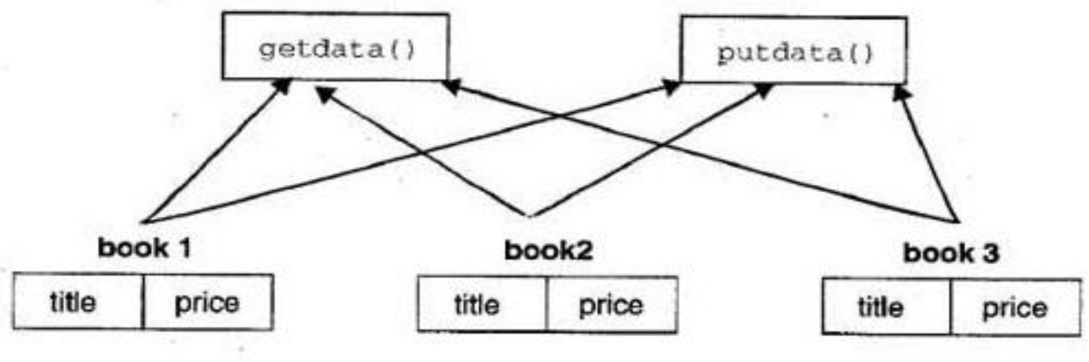
```
Return_type class_name :: function_name (parameter_list)
{
// body of the member function
}
```

Example

```
class book
{
// body of the class
} ;
void book :: getdata(char a[],float b)
{
// defining member function outside the class
strcpy(title,a):
price = b:
}
void book :: putdata ()
{
cout<<"\nTitle of Book: "<<title;
cout<<"\nPrice of Book: "<<price;
}
```

Specifying a Class

- **Memory Allocation for Object**
- The memory space for objects are allocated when they are declared and not when the class is specified.
- For about member function, when member functions are created, it will occupy memory space only once when they are defined in a class. So all objects created for that class can use same member functions, so no separate space is allocated for member functions when the objects are created



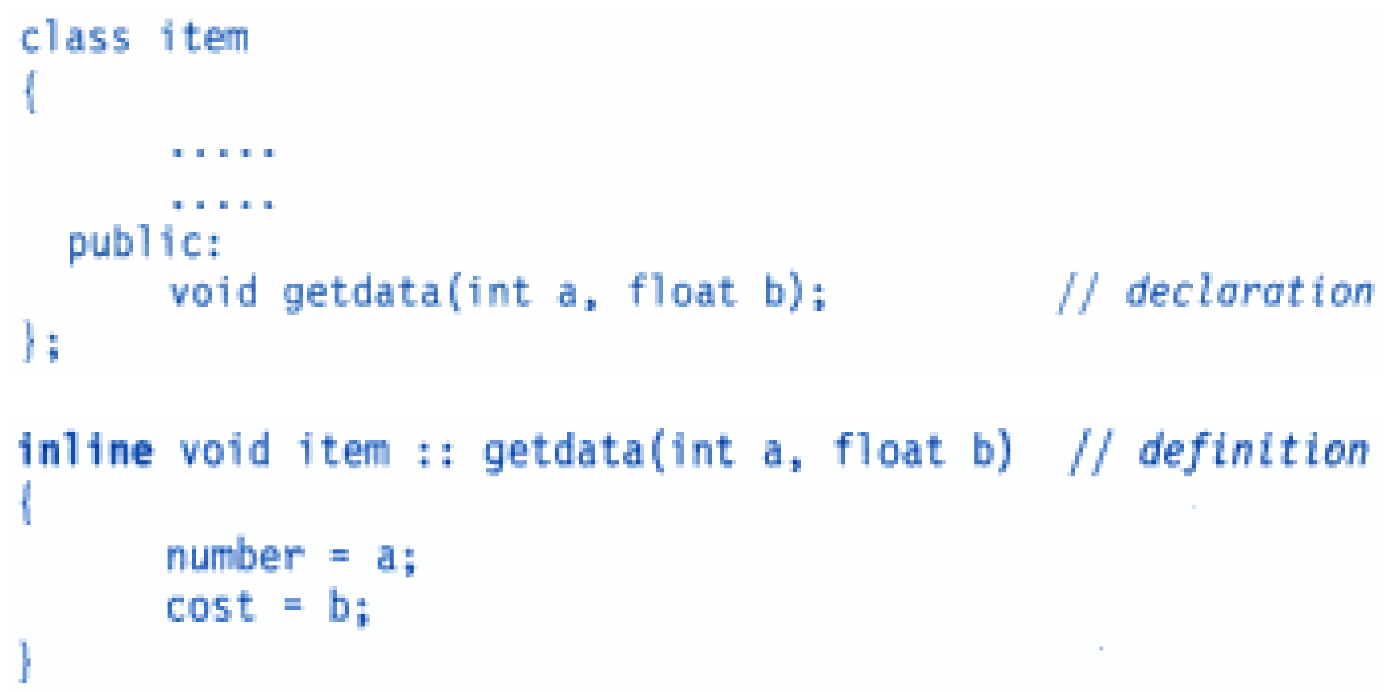
Memory Allocation for the Objects of the Class book

- **Example:** A Class student have three data members such are rollno, age, per and two member functions getdata() and show().If we create three objects s1, s2, and s3.

- So object s1 take up space for: rollno, age, per
- object s2 take up space for: rollno, age, per
- object s3 take up space for: rollno, age, per
- But it will access common member function getdata() and show(), so it will take up space only **one time when class student is created.**

Specifying a Class

- **Making outside function Inline**



```
class item
{
        .....
        .....
   public:
        void getdata(int a, float b);           // declaration
};

inline void item :: getdata(int a, float b)  // definition
{
        number = a;
        cost = b;
}
```

Specifying a Class

- **Private Member Function**
- We can not call directly from the main(), because It was private function. So the solution of these problem is, we have to declare public function of that class and we call this private function inside the public member function.
- So a public member function are called by object from main().
- **Example:**

```
class test {
private: int a, b;
void getdata(); //private member function
public: void show();
}; //end of class
void test::getdata()
{
cout<<"enter two numbers :";
cin>>a>>b;
}
void test::show(){
getdata(); //calling a private member function here
cout<<"\n two numbers are :";
cout<<a<<" "<<b;
}
int main()
{
test x; // x is object of class test
 x.getdata(); // illegal, compiler give error it is private member function, so can not call from
outside class.
x.show(); // it will call getdata() function direct
return (0);
```

Specifying a Class

- **Nesting of Member Function**
- A member function can be called by using its name inside another member function of the same class is called nesting member functions.
- **Example:**
- To Add and subtract two number using nesting function

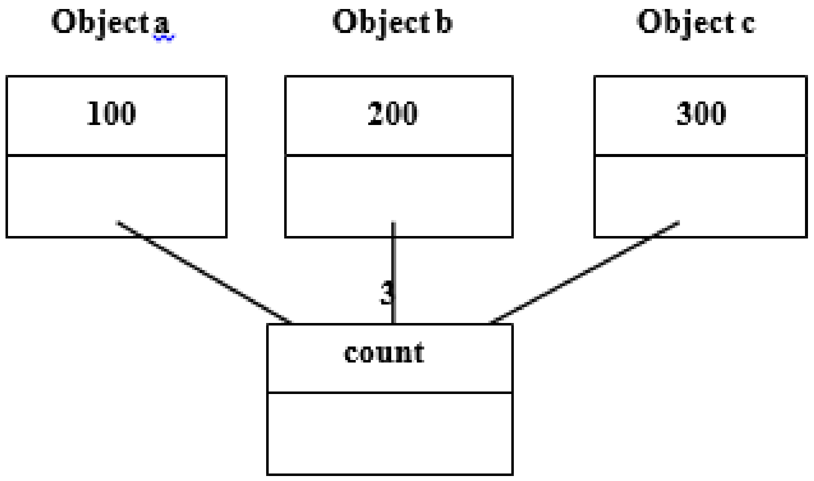
```
Class number
{         int a,b,s1,s2;
          public:
                    void getdata(int m, int n);
                    int sum();
                    int sub();
void show()
{         cout<<”\n add two num”;
          cout<<a<< “ “<<b;
          cout<< “add and sub are”;
          cout<<  sum()<<  “  “<<sub();    //calling  sum  and  sub  member
                                                      fun in show()
}  }
```

Static data member and Static member function

- **Static data members:**
- A static member variable has certain special characteristics:

1. It is automatically initialized to zero when the first object created, no other initialization is permitted. Where a simple variable have initially garbage value.
2. Only one copy of that member is created for entire class and shared by all objects of that class, no matter how many objects are created.
3. It is visible only within a class, but its life time is the entire program.

- Lets we see the example of static data member, using static data member we will count how many objects are created for class. Without static data member it is not possible to count the number objects because a simple variable can not store the value permanently.

Example:
```
#include<iostream.h>
#include<conio.h>
class item
{
        int number;
        static int count;                    \\ static variable declaration public:
                void getdata(int a)
                {
                        number = a;
                        count++;
                }
                void getcount()
                {
                          cout<<"Count : "<<count;
                }
};
int item :: count;                  \\ static variable definition
int main()
{
        item a,b,c;
        a.getdata(100);
        b.getdata(200);
        c.getdata(300);
        a.getcount();
        b.getcount();
        c.getcount();
        getch();
        return 0;
}
```


- In above program , the static count is **initialize to 0**,when object is created and we will see the value using getcount(). Then after data is read three times by using getdata(), the variable count increment three times, so value of **count is 3** after end.

- Because count is define as static data member, so only one copy of count shared by all three objects, all the output statements cause the value display is 3.

- **Static member function**
- Static member functions are associated with a class, not with any object.
- They can be invoked(call) using class name, not object.
- They can access only static members of the class.
- They cannot be virtual.
- They cannot be declared as constant or volatile(The value of the variable may change without any code of yours changing it).
- There cannot be static and non-static version of the same function.
- A static member function does not have this pointer. So they can not access nonstatic member data using the **. Or ->** operator.
- **Syntax:**
- classname:: functionname

```
#include<iostream.h>
#include<conio.h>
class item
{
        int number;
        static int count;
        public:
                void getdata(int a)
                {
                        number = a;
                        count++;
                }
                static void getcount()
                {
                        cout<<"Count : "<<count;
                }
};
int item :: count;
int main()
{
        item a,b,c;
        a.getdata(100);
        b.getdata(200);
        c.getdata(300);
        item::getcount();
        getch();
        return 0;
}
```

Output=3

Array of object

- An object of class represents a single record in memory, if we want more than one record of class type, we have to create an array of class or object. As we know, an array is a collection of similar type, therefore an array can be a collection of same class type.

Array of object

```
#include <iostream.h>
class sample
{
int roll_no;
char name[20];
public:
void read()
{
cout<<"Enter the Roll number=";
cin>>roll_no;
cout<<"Enter the Name=";
cin>>name;
}
void disp()
{
cout<<"\nRoll_no="<<roll_no<<"\tName="<<name;
}
};
```

```
void main()
{
sample S[5];
int i;
for(i=0;i<5;i++)
S[i].read();
for(i=0;i<5;i++)
S[i].disp();
}
```

Passing objects as an argument and return object

- In C, we write argument inside function bracket, but in C++ using class and objects, we can use argument as objects.
- We can pass object as an argument like any other data type and also we can return it.
- We can pass by two methods:
  - A copy of the entire object is passed to the function.
  - Only the address of the object is transferred to the function.

- We can return object by return keyword and set return type of function is class_name.
- Syntax:function_name(object_name);

```
#include <iostream.h>
#include <conio.h>
class trial
{
        int a;
        public:
        void getdata()
        {
                a=10;
        }
        void putdata()
        {
                cout<<"\nValue of a="<<a;
        }
        trial square(trial tt)
        {
                trial ttt; a=tt.a*tt.a;
                ttt.a=tt.a*tt.a*tt.a; return ttt;
        }
};
void main()
{
        trial t1,t2,t3; clrscr();
        t1.getdata(); t3=t2.square(t1);
        t1.putdata();
        t2.putdata();
        t3.putdata();
        getch();
}
```

Output

```
    Value of a=10
    Value of a=100
    Value of a=1000
```

Constructor

A constructor is a “special” member function which initializes the objects of class.

- Constructor is invoked(Process/call) automatically whenever an object of class is created.
- Constructor name must be same as class name.(**Default constructors**)
- Constructors that can take arguments are called **parameterized constructors.**
- Constructor which accepts a reference to its own class as a parameter is called **copy constructor.**
- A copy constructor is used to declare and initialize an object from another object.

- Constructors should be declared in the **public section** because private constructor cannot be invoked from outside the class so they are useless.

- Constructors do not have return types and they cannot return values, not even void.

- They cannot be inherited though derived class, because they can call the first base class constructor

Syntax:

```
class A
{
            int x;
 public:
            A();  //Constructor
};
```

```
int main()

{

        rectangle r1;                       // Invokes default constructor rectangle
        r2(10,10);                          // Invokes parameterizedconstructor
        rectangle r3(r2);            // Invokes copy constructor

        rectangle r4=rectangle();           // Invokes default constructorexplicitly
        r1=rectangle(20,20);          // Dynamic initialization of constructor

        return 0;

}
```

Destructor

- Destructor is used to destroy the objects that have been created by a constructor.
- Destructor is a member function whose name must be same as class name but is preceded by a tilde (~).
- Destructor **never** takes any **argument** nor it returns any value **nor i**t has **return type**.
- Destructor is invoked automatically by the complier upon exit from the program.
- Destructor should be declared in the public section.
- In a class there is **only one destructor.**

```
#include<iostream.h>
class sample
{
        public:
        sample()            //Constructor
        {     cout<<"Constructor\n";    }
        ~sample()           //destructor
        {     cout<<"Destructor\n";     }
};
int main()
{
        cout<<"Main block\n";
        sample s1;
        {
                cout<<"Inner block\n";
                sample s2;
        }
        cout<<"Return back to mainblock\n";
        return 0;
}
```

```
/*  OUTPUT
Main block

Constructor

Inner block

Constructor

Destructor

Return back to main block

Destructor
```

Friend Function

- As we know that a class cannot access the private members of other class. Similarly a class that doesn’t inherit another class cannot access its protected members.

- **Friend Class:**
  A **friend class** is a class that can access the private and protected members of a class in which it is declared as **friend**. This is needed when we want to allow a particular class to access the private and protected members of a class.

- **Friend Function:**
  Similar to friend class, this function can access the private and protected members of another function. A global function can also be declared as friend as shown in the example below:

Syntax:

```
class className
{
            friend return_type functionName(argument/s);
}
return_type functionName(argument/s)
{
            // Private and protected data of className can be accessed from
            // this function because it is a friend function of className.
}
```

Friend Function Example

```
#include<iostream.h>
using namespace std;
class XYZ
{
private:
    int num=100;
   char ch='Z';
   public:
   friend void disp(XYZ obj);
};
//Global Function
void disp(XYZ obj){
   cout<<obj.num<<endl;
   cout<<obj.ch<<endl;
}
int main() {
   XYZ obj;
   disp(obj);
   return 0;
}
```

THANK YOU