Object & Classes

Sisoft Technologies Pvt Ltd
SRC E7, Shipra Riviera Bazar, Gyan Khand-3, Indirapuram, Ghaziabad
Website: www.sisoft.in Email:info@sisoft.in
Phone: +91-9999-283-283

Class:

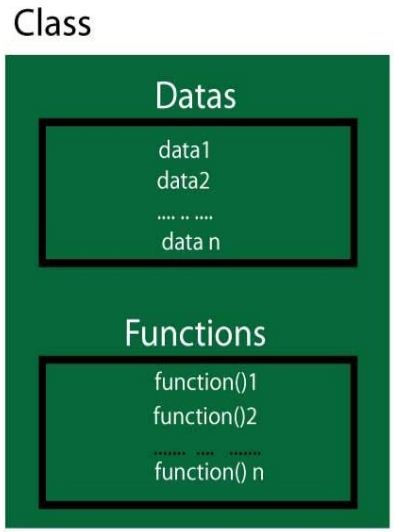
Classes are the most important feature of C++ that leads to Object Oriented programming language.

Class is a user defined data type, which holds its own data members and member functions, which can be accessed and used by object of that class.

The variables inside class definition are called as data members and the functions are called member functions of the class.

Example:

Class of birds, all birds can fly and they all have wings and beaks. So here flying is a behavior and wings and beaks are part of their characteristics.



Defining the Class in C++:-

In C++, Class is defined using keyword class followed by an identifier (name of class).

Body of class is defined inside curly brackets an terminated by semicolon at the end like structure.

Syntax:

```
class class_name
{
    // body of class
};
```

Access specifiers in C ++:

Access specifiers in C++ class defines the access control rules. It means, they set boundaries for availability of members of class i.e. (data members and member functions).

C++ used 3 keywords in classes:

1. private
2. public
3. protected

Access specifiers are followed by a colon. User can use either one, two or all the 3 specifiers in the same class to set different boundaries for different class members.

This feature in OOP is known as data hiding. Generally, data are private and functions are public.

Private keyword:

Keyword private makes data and functions is private. Private data and functions are accessible inside that class only.

Private , means that no one can access the class members outside that class. If someone tries to access the private member, they will get a compile time error.

Public keyword:

Keyword public makes data and functions is public. public data and functions are accessible both inside and outside the class.

Public, means all the class members declared under public will be available to everyone. The data members and member functions declared public can be accessed by other classes too. Hence there are chances that they might change them. So the key members must not be declared public.

Protected Keyword:

Protected, is the last access specifier , and it is similar to private, it makes class member inaccessible outside the class. But they can be accessed by any subclass of that class.

Objects :

Objects are instances of class, which holds the data, for variables of the class and the member functions of the class work on these class objects.

Each object has different data variables.

Objects are initialized using special class functions called Constructors , and whenever the object is out of its scope, another special class member function called Destructor, to release the memory which is reserved by the object.

C++ doesn't have Automatic Garbage Collector like in JAVA, so in C++ Destructor performs this task.

Simple class Example :

```
class student
{
private:
int marks;
char name[30];

public:
int getdata()
{
cout<<“Enter the name & marks of student\n";
gets(name);//cin>>name;
cin>>marks;
return marks;
}
void printdata()
{
cout<<"name:"<<"\t"<<name<<endl;
cout<<"marks:"<<"\t"<<marks<<endl;
}
};
```

```
int main()
{
student s;
s. getdata();
cout<<"Marks is :"<<s . marks<<endl;
s . printdata ();
return 0;
}
```

Output:

```
Enter the name & marks of student :
Sunita
30

Marks is: 30
Name: Sunita
Marks: 30
```

Program show difference b/w Private & public data:

```
class box
{
int length;

public:
int width;
void setdata (int l);
void printdata();
};
void box :: setdata (int l)
{
length = l;
}
void box :: printdata()
{
cout<<"length & width is:"<<"\t"<< length<<
      "\t"<<width<<endl;
}
```

```
int main()
{
box b;
b. setdata (20);
// b. length=20;      //  Show error.  Here
                          length is private.
b. width=10;
b. printdata();
getch();
}
```

Output:

Length & width is : 20 10

Protected Keyword Example :

```
class box
{
protected:
double width;
};

class smallbox : box
{
public:
void setwidth(double w)
{
width=w;
}

double print()
{
return width;
}
};
```

```
void main()
{
smallbox b;

b.setwidth(40.2);
cout<<"Width is "<<b.print();
getch();
}
```

Output:

Width is 40.2

Defining Member Function Outside the Class :

A large program may contain many member functions. For the clarity of the code, member functions can be defined outside the class.

To do so, member function should be declared inside the class(means, function prototype should be inside the class). Then, the function definition can be defined using scope resolution operator **::**

Example :

```
class student
{
public:
int marks;
char name[30];
int getdata();
void printdata ();
};

int student :: getdata()
{
cout<<"enter the name & marks of student\n";
gets(name);//cin>>name;
cin>>marks;

return marks;
}
```

```
void student :: printdata()
{
cout<<"name:"<<"\t"<<name<<endl;
cout<<"marks:"<<"\t"<<marks<<endl;
}};

int main()
{
student s;
s. getdata();
cout<<"Marks is :"<<s . marks<<endl;
s . printdata ();
return 0;
}
```

Output:

```
Enter the name & marks of student :
Sunita
30
Marks is: 30
Name: Sunita
Marks: 30
```

Types of Member Functions in C++

- Simple functions
- Inline functions
- Static functions
- Const functions
- Friend functions

Simple Member functions:

These are the basic member function, which don't have any special keyword like static etc as a prefix.

Syntax:

```
return_type function_Name(parameter list)
{
        //   function body
}
```

Inline functions:

All the member functions defined inside the class definition are by default declared as Inline.

Static Member functions:

Static is a keyword which can be used with data members as well as the member functions. A function or variable is made static by using static keyword.

Static functions are class type functions , so they can be called using the object or called directly using the class name.

These functions cannot access ordinary data members and member functions, they can accessed only static data members and static member functions.

Const Member functions:

Const keyword makes variables and functions constant, that means once defined, there values can't be changed . When used with member function, such member functions can never modify the object or its related data members.

Syntax: void fun() const { }

Friend functions:

Friend functions are actually not class member function. They are made to give private access to non-class functions.

You can declare a global function as friend, or a member function of other class as friend.

Inline Functions in C++

All the member functions defined inside the class definition are by default declared as Inline.

With an inline function, the compiler tries to expand the code in the body of the function in place of a call to the function.

Inline functions in C++ do the same thing what Macros do in C. Preprocessors were not used in C++ because they had some drawbacks.

Example:

```
class Number
{
public:
inline int max(int x, int y)
{
return (x>y)?x:y;
}
};

int main()
{
Number n;
cout<< “max(40,50)”<<n.max(40,50)<<endl;
}
```

Output: 50

Nesting of Member functions

A member function of a class can be called only by an object of that class using a dot operator.

A member function also can be called by using its name inside another member function of the same class. This is known as nesting of member functions.

```
class set
{
int m,n;
public:
void input(void);
void display(void);
int largest(void);
};

int set :: largest(void)
{
if(m >= n)
return(m);
else
return(n);
}
void set :: input(void)
{
cout << "Input value of m and n"<<"\n";
cin >> m>>n;
}
void set :: display(void)
{
cout << "largest value=" << largest()
<<"\n";
}

int main()
{
set A;
A.input();
A.display();

return 0;
}
```

The output of program would be:

```
Input value of m and n
25 18
Largest value=25
```

Array within a Class in C++

Arrays can be declared as the members of a class. The arrays can be declared as private, public or protected members of the class.

Example:

```
const int size=10;

class array
{
int a[size];
public:
void setval(void);
void display(void);
};
```

Here array variable a[] declared as private member of the class array can be used in the member function, like any other array variable.

We can perform any operations on it. In this class, the member function setval() sets the value of element of the array a[], and display() function displays the values. Similarly, we may use other member functions to perform any other operation on the array values.

```
const int size=5;
class student
{
int roll_no;
int marks[size];
public:
void getdata ();
void tot_marks ();
} ;
void student :: getdata ()
{
cout<<"\nEnter roll no: ";
cin>>roll_no;
for(int i=0; i<size; i++)
{
cout<<"Enter marks in subject"<<(i+1)<<": ";
cin>>marks[i] ;
}
}
```

```
void student :: tot_marks() //calculating total
      marks
{
int total=0;
for(int i=0; i<size; i++)
total+ = marks[i];
cout<<"\n\nTotal marks "<<total;
}
int main()
{
student stu;
stu.getdata() ;
stu.tot_marks() ;
return 0;
}
```

Array of Objects in C++

Arrays of variables having type "**class**" is known as "**Array of objects**". Here the "identifier" used to refer the array of objects is an user defined data type.

Consider the following class definition:

```
class employee
{
char name[30];
float age;
public:
void getdata(void);
void putdata(void);
};
```

Here the identifier employee is a user defined data type, which used to create objects that relate to different categories of the employees.

Ex: employee manager[3]; // An array of manager
employee worker [4]; // An array of worker

```
class student
{
char name[30];
int rollno;

public:
int getdata()
{
cout<<"Enter name & age\n";
cin>>name>>rollno;
return 0;
}
int printdata()
{
cout<<"name:"<<name<<endl;
cout<<"rollno:"<<rollno<<endl;
return 0;
}
};
```

```
int main()
{
student s[3];
int i;
cout<<"List is\n";
for(i=0;i<3;i++)
{
s[i].getdata();
}
cout<<endl;

for(i=0;i<3;i++)
{
cout<< "student "<<i+1<<endl;
s[i].printdata();
}
}
```

Objects as function arguments in C++

The objects of a class can be passed as arguments to member functions as well as nonmember functions either by value or by reference.

1) When an object is passed by value, a copy of the actual object is created inside the function. This copy is destroyed when the function terminates. Moreover, any changes made to the copy of the object inside the function are not reflected in the actual object.

2) On the other hand, in pass by reference, only a reference to that object (not the entire object) is passed to the function. Thus, the changes made to the object within the function are also reflected in the actual object.

```
class A
{
private:
        int num1;
        int num2;
public:
        A():num1(1),num2(1)
        {}
        void get ()
        {
                cout<<"enter num1";
                cin>>num1;
                cout<<"enter num2";
                cin>>num2;
        }
        void print ()
        {
 cout<<"num1is:"<<num1<<"\t"<<"num2
     is:"<<num2<<endl;
        }
```

```
void multi(A r1, A r2)
        {
                num1=r1.num1*r2.num1;
                num2=r1.num2*r2.num2;
        }
};
int main ()
{
        A r1,r2,r3;
        r1.get();
        r2.get();
        r3.multi(r1,r2);

        r3.print();

}
```

Returning Objects in C++

A function cannot only receive objects as arguments but also can return them.

The function could return

1) A reference to an object
2) A constant reference to an object
3) An object
4) A constant object

If a method or function returns a local object, it should return an object, not a reference.

If a method or function returns an object of a class for which there is no public copy constructor, such as **ostream** class, it must return a reference to an object.

Example:

```
class data
{
 int a , b ;
 public:
 void get();
 friend data sum (data , data);
void show();
 };

void data::get()
{
 cout<<"PLEASE ENTER FOR A:->";
cin>>a;
cout<<"PLEASE ENTER FOR B:->";
cin>>b;
 }
```

```
 void data::show()
 {
 cout<<"A= "<<a<<endl;
 cout<<"B= "<<b<<endl;
 }

 data sum(data a1,data a2)
{
 data a3;
a3.a=a1.a+a2.a;
 a3.b=a1.b+a2.b;
 return a3;
}
main()
 {
 data a , b , c;
 a. get();
 b . get();
 c=sum(a , b );
 c . show();
}
```

Const member functions in C++

If a member function does not alter any data in the class, then we may declare it as a const member function.

Example:

1) void multiply (int , int) const;
2) void getdata () const;

The qualifier const is must append with function in both definitiion & declaration.

The compiler will generate an error message if such functions try to alter the data value.

```
class student
{
public:
    int marks;
 student(int x)   // Constructor
 {
     marks=x;
 }

 int printdata() const    // Constant function
 {
  //   marks++;      //   Can't modify value of marks
       cout<< marks;
 }

 int output()
 {
    marks++;
    cout<< marks;
 }
};
```

```
int main()
{
student obj1(10);         // Non const Object
const student obj2(20);    // Const Object

obj1.printdata();   // No error
obj2.printdata();   // No error

cout << obj1.marks << obj2.marks ;
 obj1.output();   // No error
//   obj2.result();   // Compile time error
       because obj2 is const object
}
```

Output:
10 20 10 20 11

Here, we can see, that const member function never changes data members of class, and it can be used with both const and non-const object.
But a const object can't be used with a member function which tries to change its data members.
*/

Local Class in C++

Classes can be defined and used inside a function or a block. Such classes are called local classes.

Example:

```
void test (int a)
{
……
……
Class Student
{
……
……
};
……
……
Student s1(a);
……
}
```

```
int j = 8;
void fun()
{
    int y;      //  Non-static variables
    static int x;          // static variable

    enum {i = 1, j = 2};

    class Test    // Local class
    {
       int z =7;

     public:
       void method() {
          cout << "x = " << x << endl;
          cout << "i = " << i << endl;
//          cout << "y = " << y << endl;  // Member is non
    static so not accessible inside local classes.
          cout << "z = " << z << endl;
          cout << "j = " << j << endl;
    }
    };

    Test t;
    t . method();
}

int main()
{
   fun();
   return 0;
}
```

Output:

x=0

i=1

z=7

j=8

Friend Function in C++

We know that the private members cannot be accessed from outside the class. It means that a nonmember function cannot have an access to the private data of a class.

However, they could be a situation where we could like two classes to share a particular function.

In such situations, C++ allows the common function to be made friendly with both the classes, which allow the function to access the private data of these classes. Such a function need not be a member of any of these classes.

To make an outside function “friendly” to a class, we simply declare this function as a friend of the class.

Declaration of Friend Function:

```
class A
{
……
……
public:
……
……
friend void print (void)          //        Declaration
};
```

Characteristics of Friend Function:

1) It is not in the scope of the class to which it has been declared as friend.

2) Since it is not in the scope of the class, it cannot be called using the object of that class.

3) It can be invoked like a normal function without the help of any object.

4) Unlike member function, it cannot access the member names directly and has to use an object name and dot membership operator with each member name.

5) It can be declared either in the public or the private part of a class without affecting its meaning.

6) Usually, it has the objects as arguments.

Program

```
class sum
{
private:
   float a , b;
   friend  float mean (sum s);
public:
   void result()
   {
    cout<<"Enter the values";
    cin>>a>>b;
   }
//   friend  float mean (sum s);
};
```

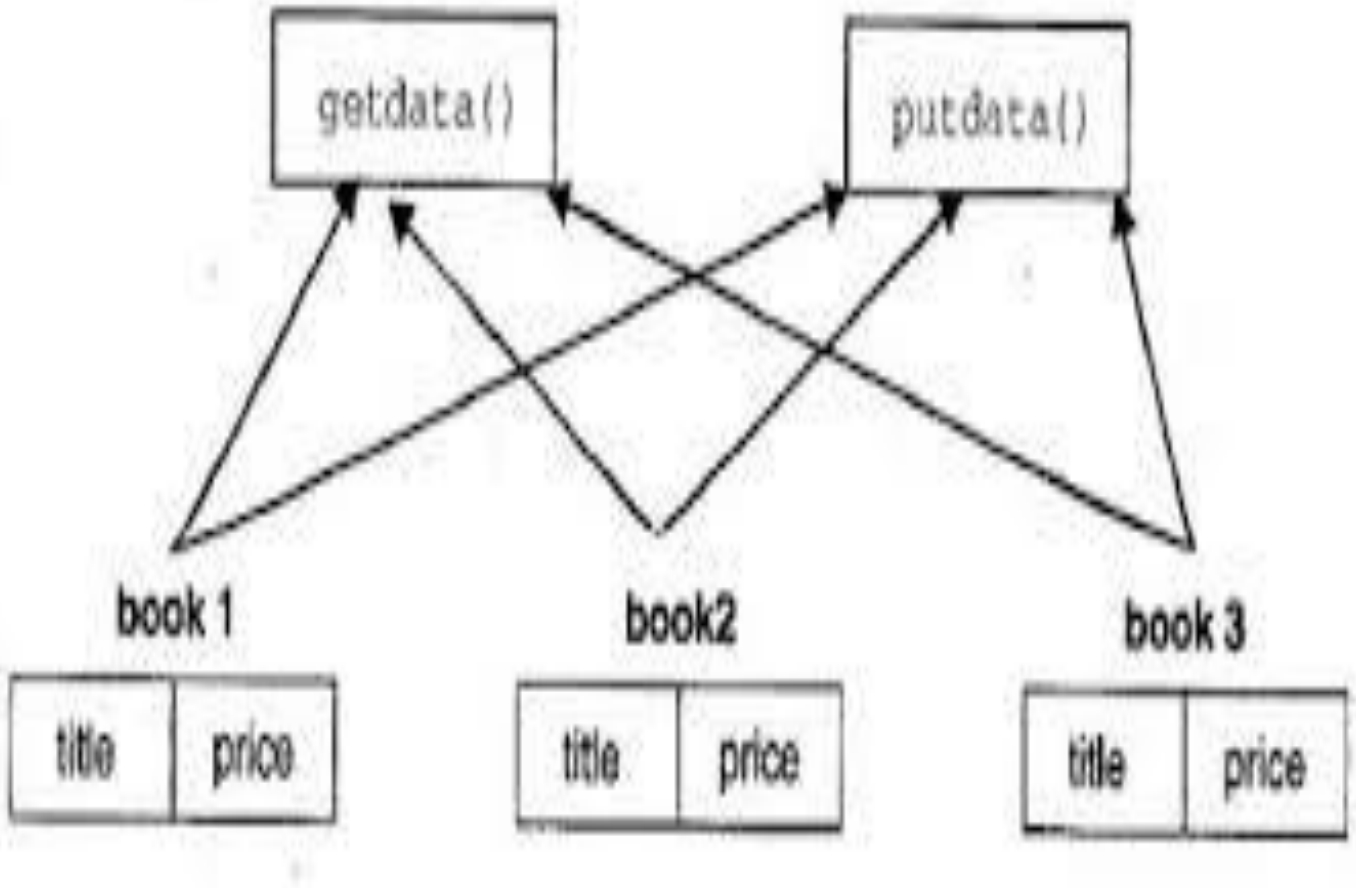
```
float mean (sum s)
{
   return float(s .a + s .b)/2;
}

int main()
{
   sum s1;
   s1.result();
   cout<<mean(s1);
 }
```

Memory Allocation for Objects

- We know that the memory space for objects is allowed when they are declared and not when the class is specified. This statement is partly true.

- Actually , the members functions are created and placed in the memory space only once when they are defined as a part of a class specification.

- Since all the object belonging to that class use the same member function, no separate space is allocated for member functions when the objects are created. Only space for member variables is allocated separately for each object, becoz each variables hold different data for each object.



Memory Allocation for the Objects of the Class book

Static Data Members in C++

Static is a keyword in C++ used to give special characteristics to an element. The properties of a static member variable are similar to that of a C static variable.

A static member variable has certain special characteristics. These are:

1) It is initialized to zero when the first object of its class is created. No other initialization is permitted.

2) Only one copy of that member is created for the entire class and is shared by all the objects of that class, no matter how many objects are created.

3) It is visible only within the class, but its lifetime is the entire program.

Static data member in class

Static data members of class are those members which are shared by all the objects. Static data member has a single piece of storage, and is not available as separate copy with each object, like other non-static data members.

Static member variables (data members) are not initialized using constructor, because these are not dependent on object initialization.

Static variables are normally used to maintain values common to the entire class. For example, a static data member can be used as a counter that record the occurrences of all the objects . Program illustrates the use of a static data member.

```
using namespace std;

class item
{
static int count;
int number;
public:
void getdata (int a)
{
number = a;
count ++;
}
void getcount (void)
{
cout << "Count: ";
cout << count <<"\n";
}
};
int item :: count;

int main()
{
item a , b ,c;
a . getcount();
b. getcount();

a.getdata(100);
b . getdata(200);

cout << "After reading data"<<"\n";
a . getcount();
b . getcount();
return 0;
}
```

Static class Objects :

Static keyword works in the same way for class objects too. Objects declared static are allocated storage in static storage area, and have scope till the end of program.

Static objects are also initialized using constructors like other normal objects. Assignment to zero, on using static keyword is only for primitive data types, not for user defined data types.

Static Member Functions

Like static member variable, we can also have static member function. A member function that is declared static has the following properties:

A static function can have access to only static members declared in the class.

A static member function can be called the class name as follows:

class-name::function-name;

```
class item
{
   static int count;
   int number;
   int data;

   public:
      void getdata (int a)
      {
         number = a;
         count ++;
      }
   static void getcount (void)
      {
         cout << "Count: ";
         cout << count <<"\n";
//       cout<<"Data is: "<<"\t"<< data <<endl;
//   Static function access only static data.
      }
};
```

```
int item :: count;  // Definition of static data member

int main()
{
   item a , b ,c;          // Count is initialized to zero

   a.getdata(100);
   b.getdata(200);

   item :: getcount();     // Access using Class Name

   a.getcount();           // Access using Objects
   b.getcount();
}
```

Output:

```
Count: 2
Count: 2
Count: 2
```