Classes

&

Objects

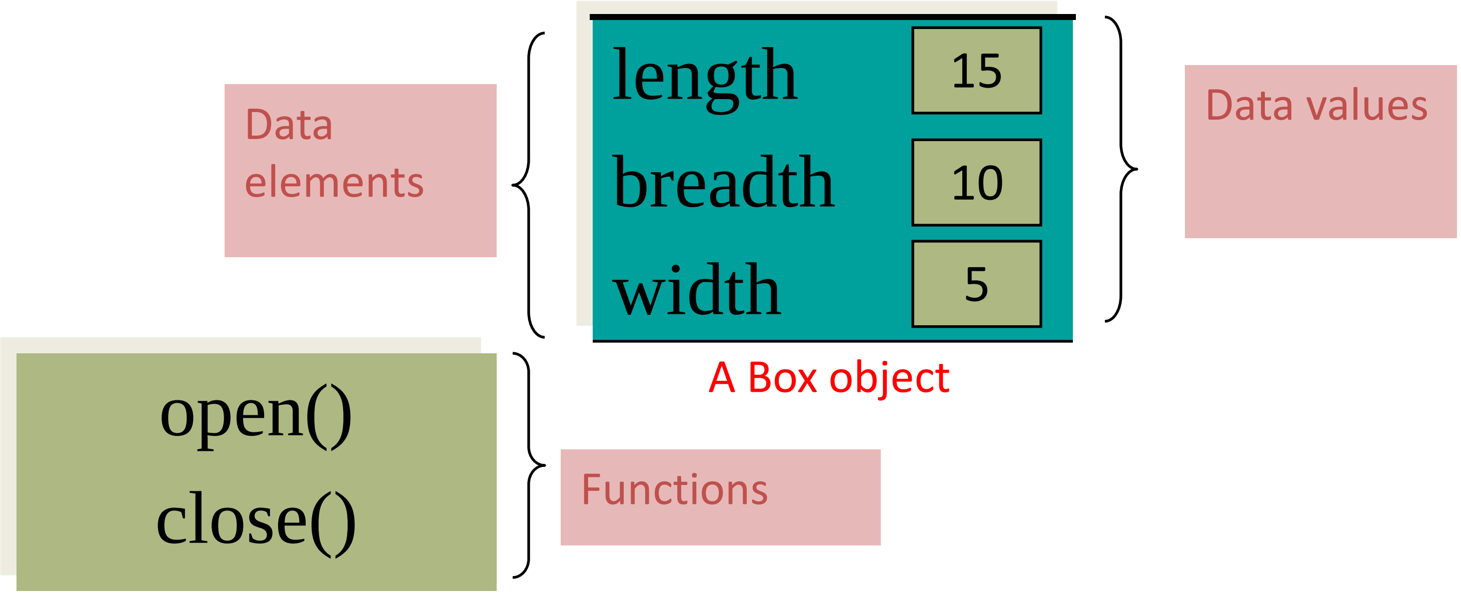
Object

- An object is a collection of some properties, behavior with some existence.
- Box object:

Properties:- *length , breadth, width.* (data elements)

Behavior:- *open, close.* (functions)

Existence:- *length=15, breadth=10, width=5* (data values)



Data elements

length 15

breadth 10

width 5

Data values

A Box object

open()

close()

Functions

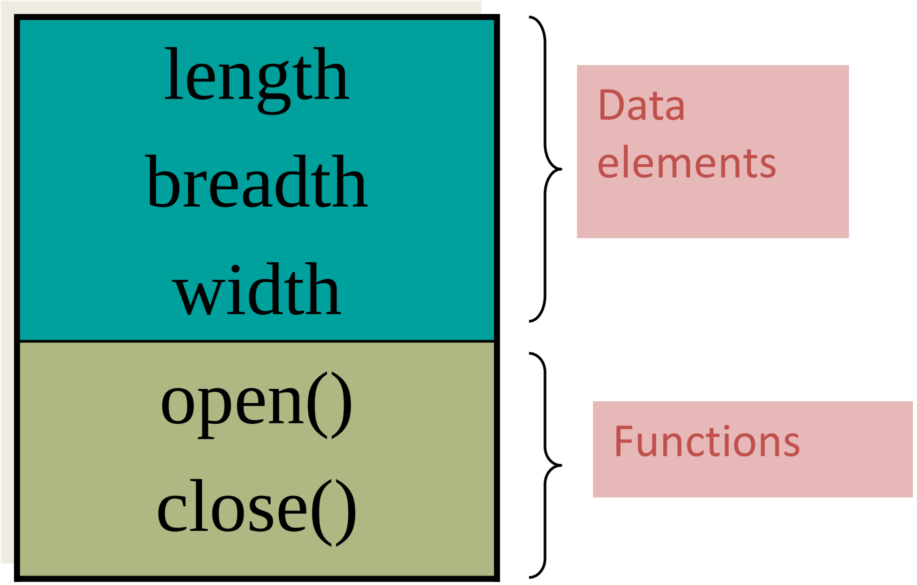
- ❑ The data elements are also called as *static properties* and the data values are called as *dynamic properties*.
- ❑ Both the static and dynamic properties together defines the *state of the object*.

Class

- Class is a collection of some properties, behavior
- BOX class:

Properties:- *length* , *breadth*, *width*. (data elements)

Behavior:- *open*, *close*. (functions)



A Box class

- Class is a logical structure or prototype where as Object has physical existence.
- Object is an instance of a Class.
- Class is a collection of similar types of objects where the data values may not be the same.
- Class does not posses dynamic properties.

General Structure of a CLASS

```
class class_name
{
    access specifier:
            member variables;
            member functions;
            class variables;
            class functions;
};
```

- The access specifier of a class provides its outside view i.e. it defines the way, the identifiers (variables and functions) are accessed from outside.
- Access specifiers are of 3 types. ***public***, ***private*** and ***protected***.

- public: The identifiers can be accessed directly outside the class.
- private: The identifiers can only be accessed inside the class.
- protected: same as private but it can be inherited but private identifiers can't be.

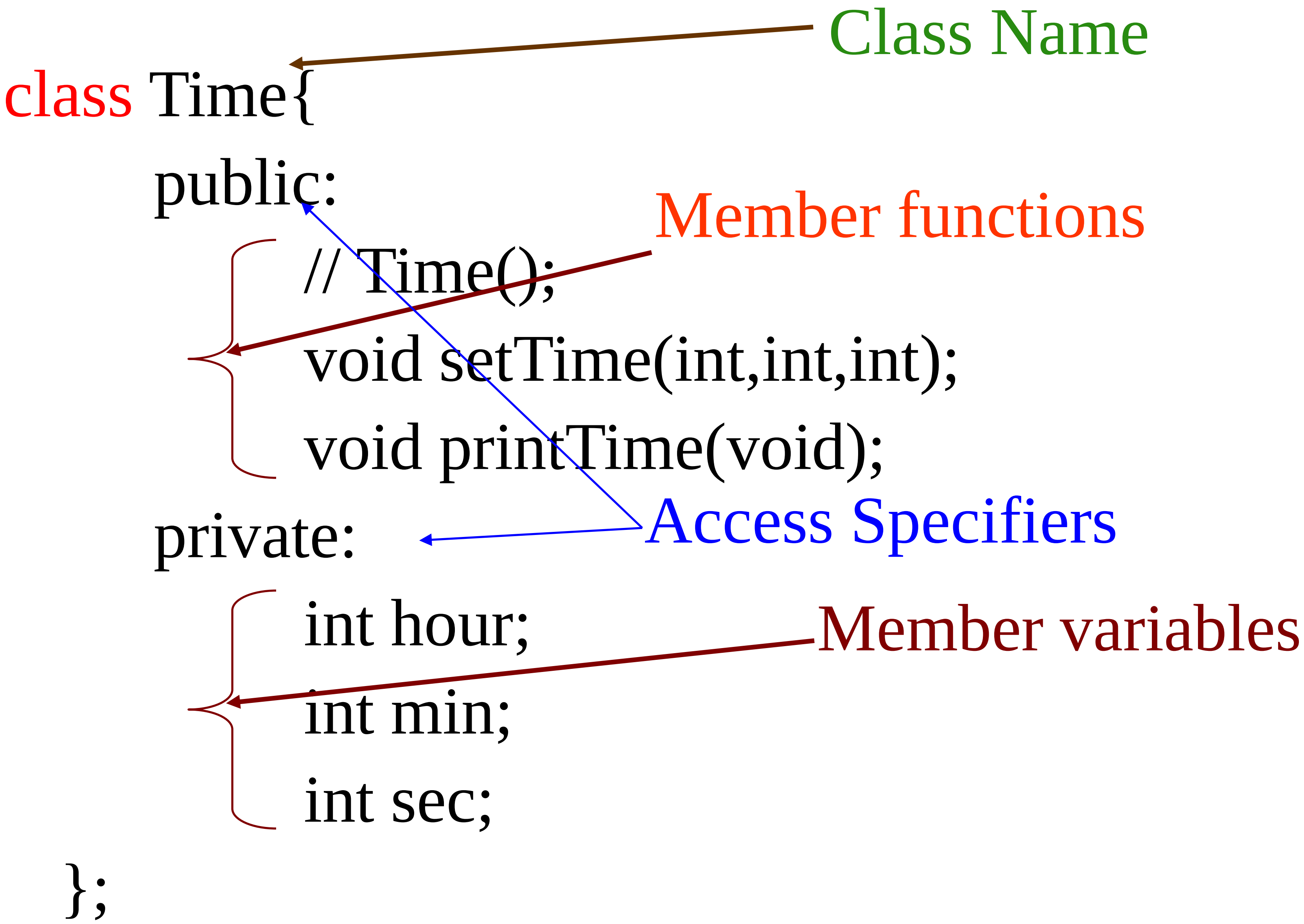
The default access specifier in a class is private.

❑ Any variable or function specified with keyword ***static*** are called as class variable or class functions.

NOTE !!!

Each class definition must be ended with a semicolon mark (;)

An Example



```
class Time{
      public:
            // Time();
            void setTime(int,int,int);
            void printTime(void);
      private:
            int hour;
            int min;
            int sec;
};
```

Class Name

Member functions

Access Specifiers

Member variables

Structure vs. Class in C++

- A structure is simply a class whose members are public by default.

Example:

```
struct Time{
        int hour;
        int min;
        int sec;
        public:
        void setTime(int,int,int);
        void printTime(void)
};
```

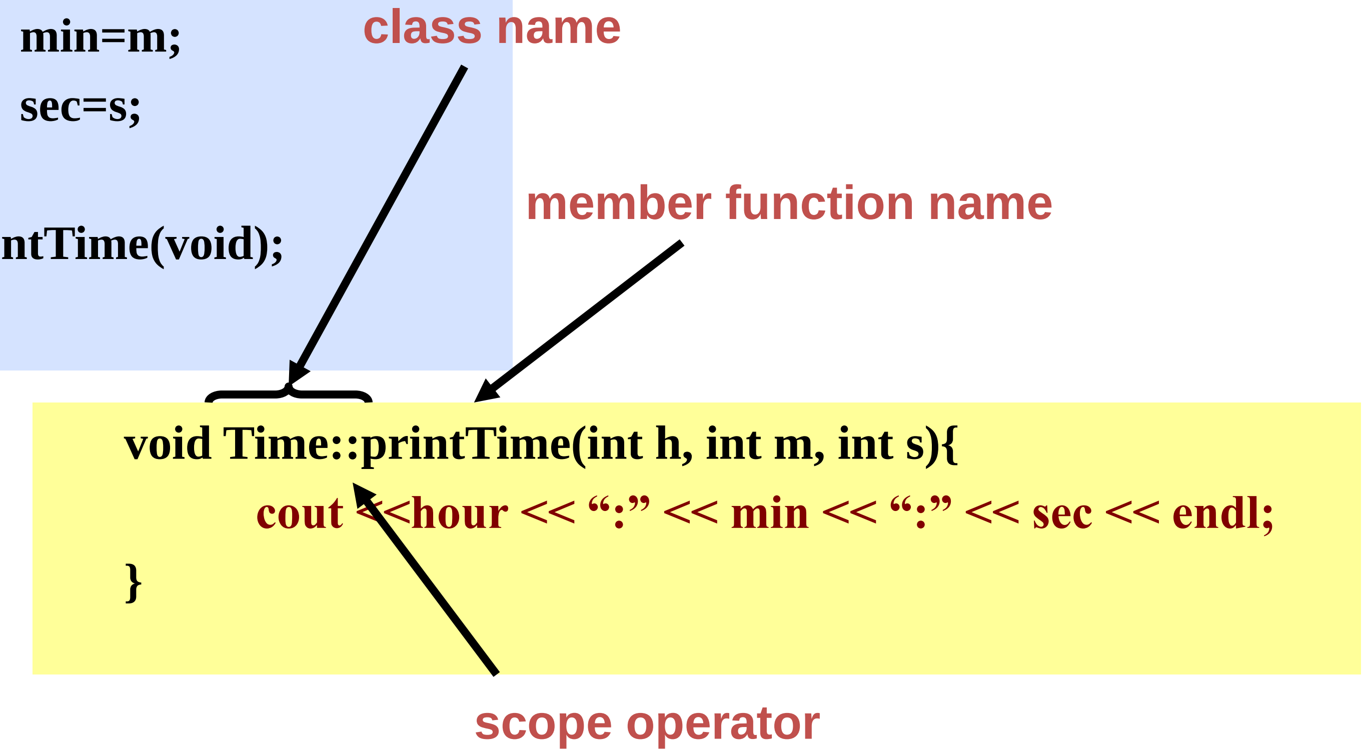
Public by default

```
class Time{
        int hour;
        int min;
        int sec;
        public:
        void setTime(int,int,int);
        void printTime(void);
};
```

Private by default

Define a Member Function (non-static)

```
class Time{
    private:
            int hour;
            int min;
            int sec;
    public:
             void setTime(int h,int m,int s){
                        hour=h;
                        min=m;
                        sec=s;
            }
             void printTime(void);
};
```



class name

member function name

```
void Time::printTime(int h, int m, int s){
        cout <<hour << “:” << min << “:” << sec << endl;
}
```

scope operator

Declaration of an Object

Similar to declaration of a structure variable in C++.

```
struct emp e1;
Struct emp e2,e3,e4;
```

```
emp e1;
emp e2,e3,e4;
```

```
class Time t1;
class Time t2,t3,t4;
```

```
Time t1;
Time t2,t3,t4;
```

```
class Time{
    private:
            int hour,min,sec;
    public:
             void setTime(int h,int m,int s){
                 hour=h;   min=m;   sec=s;
            }
             void printTime(void);
};
void Time::printTime(int h, int m, int s){
    cout <<hour << “:” << min << “:” <<
    sec << endl;
    }
```

```
main(){
    Time t;
    t.setTime(13,27,6);
 ➡ t.printTime();
    }
```

t

```
hour = 13
min = 27
sec = 6
```

```
13:27:6
```

Understanding private and public

```
class X{
    public:
        int a;
};
```

```
main(){
    X x1;
    x1.a=5;
    cout<<a    //Error
    cout<<x1.a    // 5
}
```

```
class X{
        int a;
};
```

```
main(){
    X x1;
    x1.a=5;    //Error
    cout<<a    //Error
    cout<<x1.a    //Error
}
```

```
class X{
        int a;
    public:
        void set(int b){
            a=b;
        }
        void get(){
            cout<<a;
        }
};
```

```
main(){
    X x1;
    x1.set(5);
    x1.get();    // 5
}
```

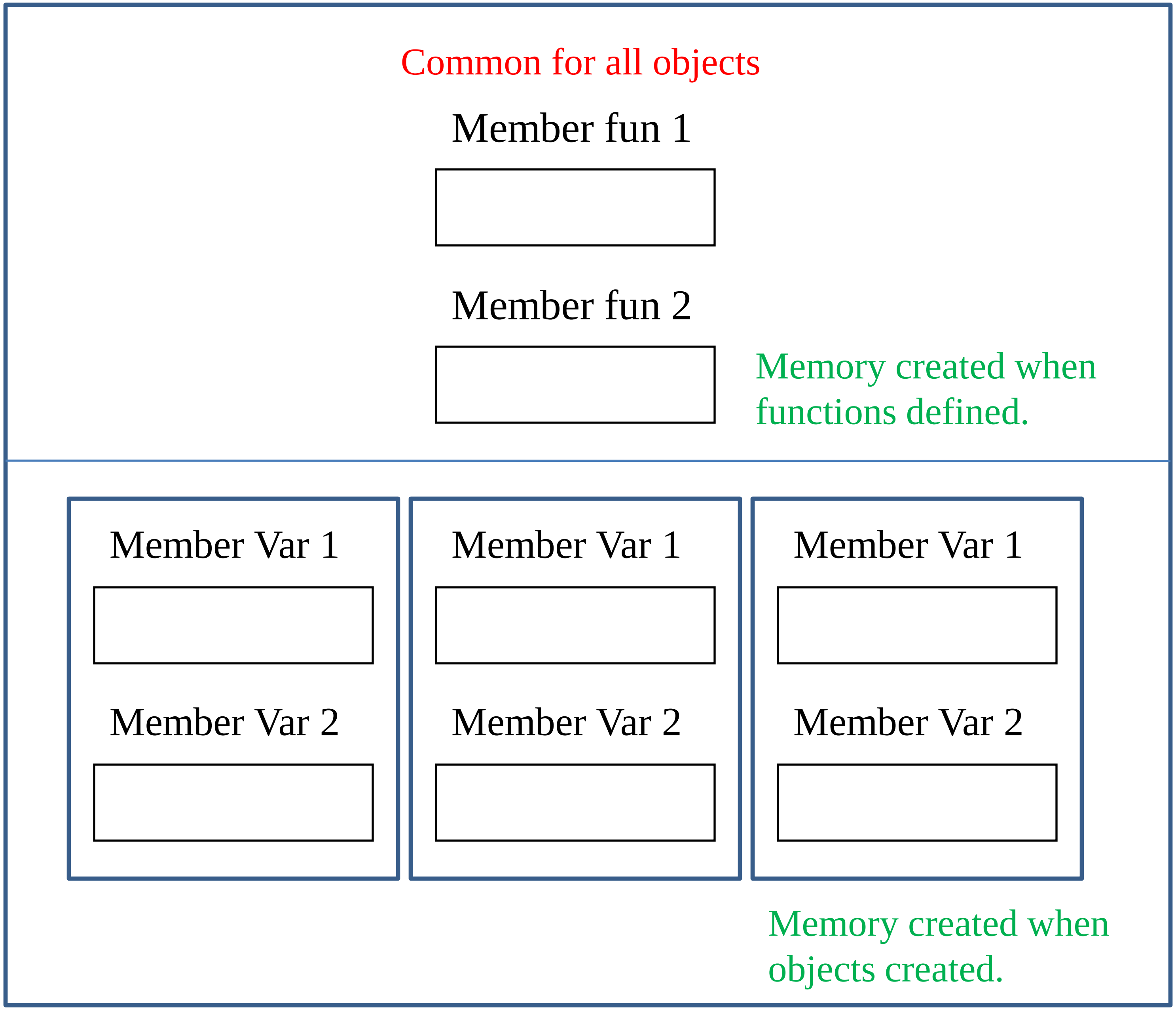
Properties of Member Function

- Several classes can have member function with same name.
- Member function can access all the data members inside the class irrespective of their access specifiers.
- One member function can call another member function of same class directly without using dot operator. Such a mechanism is called ***nested member function***.
- Member functions defined inside the class are inline by default, but the outside defined member functions are to be made inline explicitly if needed

```
class X{
            int a;
            void disp(int b){
                        a=b;
                        cout<<a;
            }
    public:
            void call_disp(int c){
                        disp(c);
            }
};
```

```
main(){
    X x1;
    x1.disp(5);  // Error
    x1.call_disp(5);
}
```

Memory Allocation for Objects



Common for all objects

Member fun 1

Member fun 2

Memory created when functions defined.

Member Var 1

Member Var 2

Member Var 1

Member Var 2

Member Var 1

Member Var 2

Memory created when objects created.

- When class is defined memory is allocated for member functions but not for member variables.
- When we declare an object then member variables get memory allocated.
- Member functions get memory only once.
- All the objects share common member functions.

Static member variable

- It retains its value through out the program.
- It gets its memory allocated at the time of class definition.
- Only one copy of the static variable is created and is shared by all the objects.
- These variables must be initialized by the programmer outside the class only.
- It can be accessed by class name and :: operator.

```
class X{
            int a;
    public:
            static int b;
            void incr(){
                        a=10;
                        b++;
                        cout<<b;
            }
};
int X::b=5;    // default ininitalization
               // value is zero
```

```
main(){
    X x1,x2,x3;
    x1.incr();     // 6
    x2.incr();     // 7
    x3.incr();     // 8
   cout<<X::b; // 8
}
```

Static variables are called as ***class variables*.**

Initialization syntax

<data-type> <class name> :: <var-name> = <value>;

Static member function

- It gets its memory allocated at the time of class definition as like a non-static member function.
- It can access other static members only (static member variable and other static member function)
- Like static member variable it can be accessed by class name and **::** operator, hence are called as ***class function***.

```
class X{
        int a;
         static int b;
    public:
        static void incr(){
                a=10;
                b++;
                cout<<b;
        }
};
int X::b;
```

Static member function can’t access a non-static member directly

```
main(){
    X x1,x2,x3;
    x1.incr();  // 1
    x2.incr();  // 2
    x3.incr();  // 3
   X :: incr();   // 4
}  cout<<X::b;
```

ERROR: ‘b’ is private member.

A non-static member function can access both static and non-static members.

```
class X{
            int a;
             static int b;
      public:
            static void incr(X ob){
                        ob.a=10;
                        b++;
                        cout<<b;
            }
};
int X::b;
```

```
main(){
     X x1;
     X :: incr(x1);  // 1
}
```

A static member function can access non-static members through object of that class.

Member function overloading

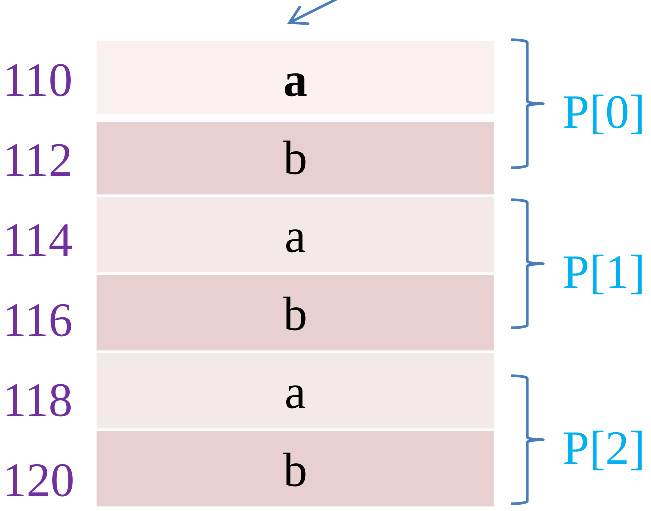
```
class X{
            int a,b;
      public:
            void set(){
                        a=b=0;
            }
            void set(int m){
                        a=b=m;
            }
            void set(int m,int n){
                        a=m;
                        b=n;
            }
            void disp(){
                        cout<<a<<b;
            }
};
// Member function set() is overloaded
```

```
main(){
      X x1,x2,x3;
      x1.set();
      x1.disp();        // 0 0
      x2.set(5);
                        // 5 5
      x2.disp();
      x3.set(10,20);
      x3.disp();    // 10 20
}
```

Array of objects

```
class X{
      public:
            int a,b;
};
```

```
main(){
      X p[3];
      int i;
      for(i=0;i<3;i++){
            cin>>p[i].a>>p[i].b;
      }
      for(i=0;i<3;i++){
            cout<<p[i].a<<p[i].b;
      }
}
```



110	**a**	P[0]
112	b	
114	a	P[1]
116	b	
118	a	P[2]
120	b	

```
      p[2].a=50;
      p[2].b=60;
      cout<<p[2].a<<p[2].b;
}
```

Const Member Function

- ❑ Does not modify the state of the object

```
class  Time
{
  private :
      int     hrs, mins, secs ;

  public :
      void       printTime( )  const ;
} ;
```

function declaration

function definition

```
void Time :: printTime( ) const
{
     cout <<hrs << “:” << mins << “:” << secs << endl;
}
```

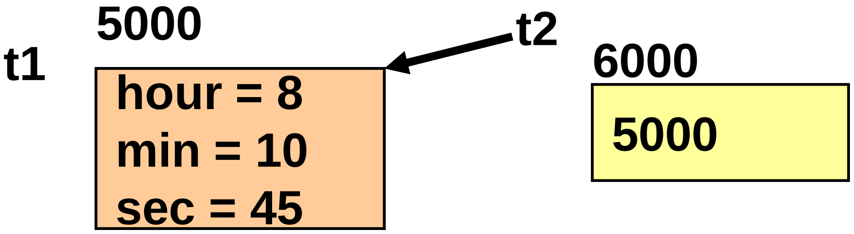
Pointer to an Object

```
class Time{
    private:
            int hour,min,sec;
    public:
             void setTime(int h,int m,int s){
                  hour=h;   min=m;   sec=s;
            }
             void printTime(void);
};
void Time::printTime(int h, int m, int s){
    cout <<hour << “:” << min << “:” <<
    sec << endl;
    }
```

t2 is a pointer to a Time object

```
main()
{
 Time t1;
  t1.set(13,27,6);      //dot notation

Time *t2;
 t2 = &t1;
 t2->set(8,10,45);      //arrow notation
}
```



Assignments

1. Implement a structure Rectangle with following operation
   - printValue(): print value of sides rectangle
   - setValue(l,w): set the value of sides of rectangle
   - area(l,w)
   - perimeter(l,w)
2. Create a class COMPLEX to implement the following operations

   setNum();
   printNum();
   add();
   subtract();
   multiply();
3. Implement Q1 using class by initializing 3 objects in different methods
   - Static
   - By pointer object
   - Dynamic
4. Repeat Q1 and Q3 for Time class with following operations
   - printTime()
   - setTime(h,m,s)
   - inMinutes(): print the time in term of minutes
   - inSecond(): print the time interm of seconds