CHARUSAT
CHAROTAR UNIVERSITY OF SCIENCE AND TECHNOLOGY

CE142: OBJECT ORIENTED PROGRAMMING WITH C++

December – April 2019

Chapter – 6

Classes and objects

Devang Patel Institute of Advance Technology and Research

DEPSTAR

Objectives

- To learn about specifying Classes and Object in C++
- To Learn accessing class member using object
- To learn about private member function accessibility

Contents

- Structures in C and its limitations
- Specifying a Class
- Creating Objects
- Accessing Class Members
- Defining Member Functions
- Making an outside Function Inline
- Nesting of Member Functions
- Private Member Functions

Introduction

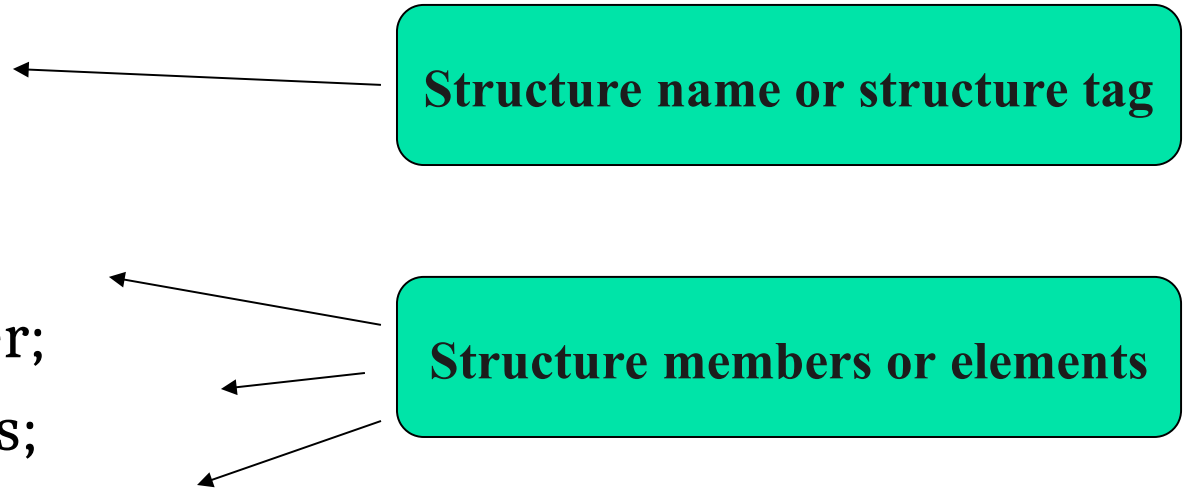
- Classes is an extension of the idea of structure used in C.
- It is a new way of creating and implementing a user-defined data type.

Structures in C

- A structure is a convenient tool for handling a group of logically related data items.
- It is a user defined data type with a template.
- Once the structure type has been defined, we can create variables of that type using declarations, that are similar to the built-in type declarations.

Structures in C

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



The keyword struct declares student as a new data type that can hold three fields of different data types.

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

Limitations of Structures in C

- The standard C does not allow the struct data type to be treated like built-in types.

- They do not permit data hiding.
- Structure members can be directly accessed by the structure variables by any function anywhere in their scope.

Structures and Classes in C++

- C++ supports all the features of structures as defined in C.
- In C++, a structure can have both variables and functions as members.
- It can declare some of its members as 'private'.
- In C++, the structure names are stand-alone and can be used like any other type names.

  student A; // C++ declaration

Continue…

- By default the members of a class are private, while, by default, the members of a structure are public.

CLASS

- A class is a way to bind the data and its associated functions together.
- It allows the data(and functions) to be hidden, if necessary, from external use.
- A CLASS specification has two parts:
  - Class Declaration
  - Class Function Definitions

Describes the type and scope of its members

Describes how the class functions are implemented

Class Declaration

```
class class_name
{
  private :
            variable declarations;
            function declarations;
  public :
            variable declarations;
            function declarations;
};
```

The **class** declaration is similar to a **struct** declaration.

Continue…

- The body of a class is enclosed within braces and terminated by a semicolon.
- The class body contains the declaration of variables and functions.
- These functions and variables collectively called class members.

```
class class_name
{
  private :
                    variable
    declarations;
          function
    declarations;
  public :
          variable
    declarations;
          function
    declarations;
}
```

Continue…

- Members grouped into two sections :
  - Private - visibility labels
  - Public
- The keywords are followed **by colon**.

```
class class_name
{
  private :
            variable
    declarations;
            function
    declarations;
  public :
            variable
    declarations;
            function
    declarations;
```

Continue…

- The class members that have been declared as private can be accessed only from within the class.
- Public members can be accessed from outside the class also.
- Keyword **private** is optional. By default, the members of a class are **private**.

```
class class_name
{
  private :
            variable
   declarations;
            function
   declarations;
  public :
            variable
   declarations;
            function
   declarations;
```

Continue…

- The variables declared inside the class are known as **data members**.
- and the functions are known as **member functions**.
- Only the member functions can have access to the private data members and private functions.

```
class class_name
{
  private :
    variable declarations;
    function declarations;
  public :
    variable declarations;
    function declarations;
};
```

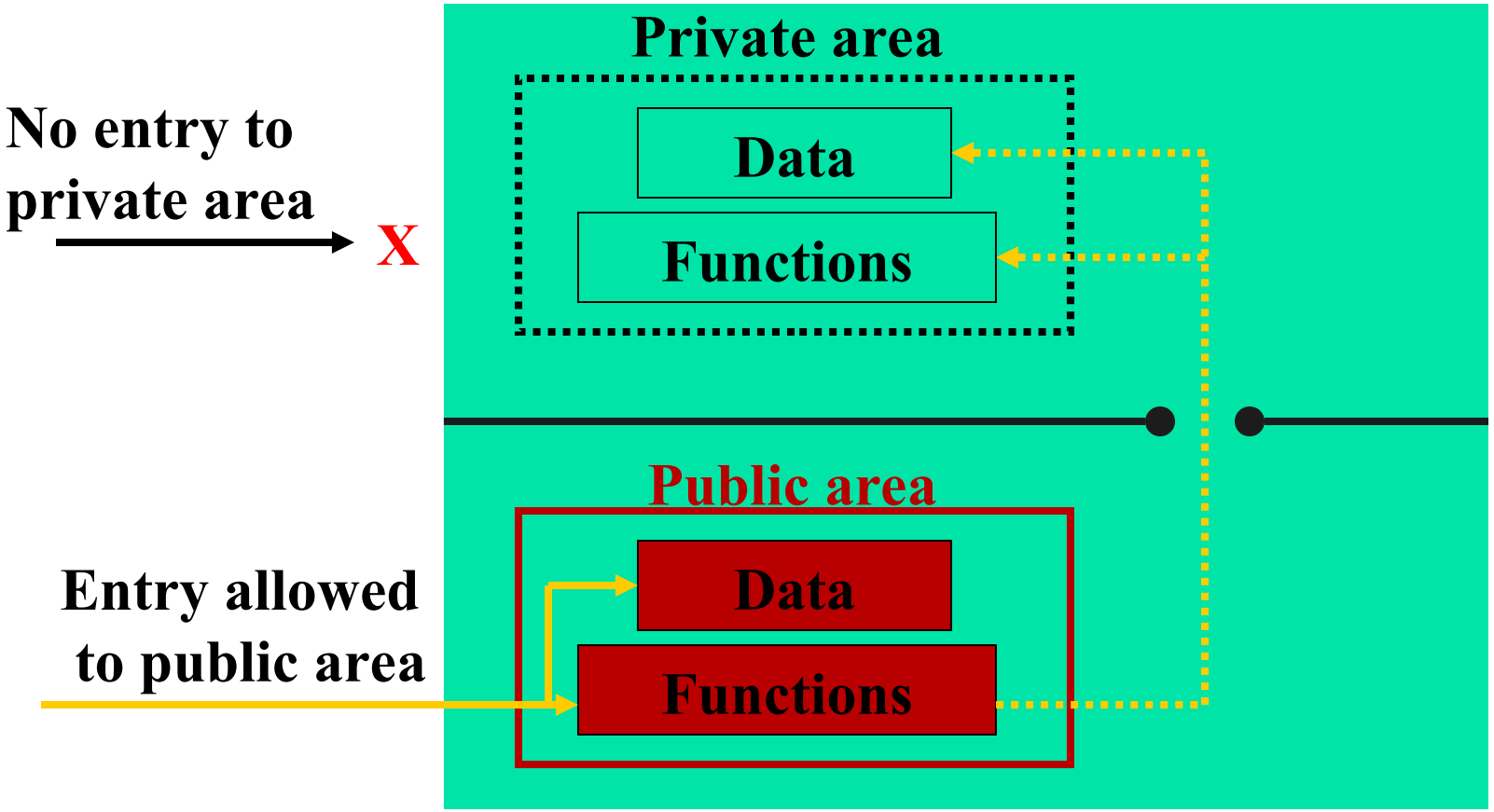
Continue…

- The public members (both functions and data) can be accessed from outside the class.

- The binding of data and functions together into a single class-type variable is referred to as **encapsulation**.

```
class class_name
{
  private :
    variable declarations;
    function declarations;
  public :
    variable declarations;
    function declarations;
};
```

Continue…



Data hiding in CLASS

Class Example

```
class item
{
        int     number;                 //  variable
    declaration
        float   cost;                   //  private  by
    default
  public :
        void    getdata( int a, float b);   //function declaration
        void    putdata( void );            // using prototype
};
```

Class Example

- Give meaningful names to classes.
- Names become the new type identifier that can be used to declare instances of that class type.
- The class **item** contains two data members and two member functions.

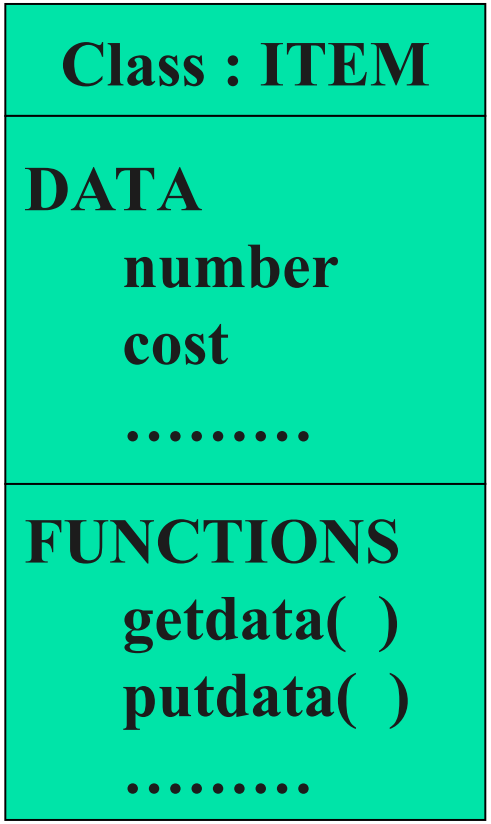
```
class item
{
    int     number;
    float   cost;
 public :
    void    getdata(int a, float
    b);
    void    putdata(void);
};
```

Class Example

- The data members are private by default
- While both the functions are public by declaration.
- The functions are declared but not defined.
- Actual function definition will appear later in the program.

```
class item
{
    int     number;
    float   cost;
 public :
    void    getdata(int a, float
    b);
    void    putdata(void);
};
```

Class Example



Representation of a class

```
class item
{
    int     number;
    float   cost;
 public :
    void    getdata(int a, float
    b);
    void    putdata(void);
};
```

Creating Objects

Once a class has been declared, we can create variables of that type by using the class name.

item x ; // create a variable **x** of type **item**.

In C++, the class variables are known as **objects**.

item x, y, z ; // declare more than one objects in one statement

Continue…

- The declaration of an object is similar to that of any basic type.
- The necessary memory space is allocated to an object at this stage.
- Class specification, like a structure, provides only a template and does not create any memory space for the objects.

Continue…

- Object can also be created when a class is defined by placing their names immediately after the closing brace.

```
class item
{
    ……
    ……
    ……
} x, y, z ;
```

Accessing Class Members

- The private data of a class can be accessed only through the member functions of that class.

object-name . function-name (actual-arguments);

In our example, although **x** is an object of the type **item** to which number belongs, the number can be accessed only through a member function and not by the object directly.

Defining Member Functions

- Member functions can be defined in two places:
  - Outside the class definition.
  - Inside the class definition.

Defining Member Functions

- **Outside the Class Definition**
  - Member functions that are declared inside a class have to be defined separately outside the class.
  - Their definitions are very much like the normal functions.
  - They should have a function header and a function body.
  - An important difference between a member function and a normal function is that a member function incorporates a membership "identity label" in the header.

This label tells the compiler which class the function belongs to.

Defining Member Functions

- **Outside the Class Definition**

```
return-type class-name : : function-name (argument declaration)
{
    Function body
}
```

  - The membership label class-name : : tells the compiler that the function function-name belongs to the class class-name.
  - The scope of the function is restricted to the class-name specified in the header line.

Defining Member Functions

- **Inside the Class Definition**
  - Replace the function declaration with the definition of the function inside the class.
  - When a function is defined inside a class, it is treated as an inline function.
  - All the restrictions and limitations that apply to an inline function are also applicable to the functions defined inside a class.

Making an Outside Functions Inline

- The member functions defined outside a class can be made inline by using the qualifier **inline** in the header line of function definition.

```
class item
{
                .....
                .....
    public :
                void getdata (int a, float b);
};
inline void item : : getdata (int a, float b)
{
    number = a ;
    cost = b ;
}
```

Nesting of Member Functions

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

- But a member function can be called by using its name inside another member function of the same class.

- This is known as nesting of member functions.

Private Member Functions

- Private member functions can be created for making them to be hidden.
- A private member function can only be called by another function that is a member of its class.
- Even an object cannot invoke a private function using the dot operator.

Private Member Functions

```
class product
{
    int       code ;
    float stock ;
    void read ( void ) ;
 public :
    void      update( void ) ;
    void      display( void ) ;
};
```

If p1 is an object, then p1.read ( ) is illegal.

However, the function read( ) can be called by any of the public functions of this class.

```
void product : : update
    ( void)
{
      read ( ) ;
};
```

Arrays within a CLASS

The arrays can be used as member variables in a class.

```
const int size = 10;
class matrix
{
        int mat [ size ] ;
    public:
        void getval ( ) ;
        void putval ( ) ;
};
```

Memory Allocation for Objects

- The member functions are created and placed in the memory space only once when they are defined.
- Since all the objects belongs to that class use the same member functions, no separate space is allocated for member functions when the objects are created.
- Only space for member variables is allocated separately for each object.
- Separate memory locations for the objects are essential, because the member variables hold different data values for different objects.

Static Data Members

- A data member of a class can be qualified as static.
- Characteristics of static member variables:
  - Static data members are declared within the class and must be defined outside of the class using scope resolution operator.
  - It is initialized to zero when the first object of its class is created. No other initialization is permitted.
  - 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.
  - It is visible only within the class, but its lifetime is the entire program.

Static Data Members

- Static variables are normally used to maintain values shared among all the objects.
- The type and scope of each static member variable must be defined outside the class definition.
- This is because the static data members are stored separately rather than as a part of an object.
- Since they are associated with class itself rather than with any class object, they are also known as class variables.

Static Data Members

- Static variables are like non-inline member functions as they are declared in a class declaration and defined in the source file.
- While defining a static variable, some initial value can also be assigned to the variable.
- type class-name : : static-variable = initial value;

Static Member Functions

- Like static member variable, we can also have static member functions.
- Properties of member functions:
  - A static function can have access to only other static members (functions or variables).
  - A static member function can be called using the class name (instead of its objects) as:
  - ***class-name : : function-name;***

Example

```
#include <iostream>
using namespace std;
class Demo
{
      public:
            static int X;
       public:
            static void fun()
            {
                cout <<"Value of X: “;
                cout << X << endl;
            }
};
```

```
int Demo :: X =10;//Definition

int main()
{
      Demo::fun();
      cout<<Demo:: X;
      return 0;
}
```

Output:

```
Value of X: 10
10
```

Arrays of Objects

- Arrays of variables that are of type class are called ***arrays of objects***.

- **The array manager contains five objects, viz manager[0], manager[1], manager[2], manager[3] & manager[4].**
- **Array of objects behave like any other array.**
- **manager [i]. putdata( ); to execute the putdata( ) member function of the i^{th} element of the array manager.**

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

employee manager [5];
employee worker [25];
```

Objects as Function Arguments

- An object can be used as a function argument like any other data type.
- Two ways:
  - A copy of the entire object is passed to the function. ***(Pass-by-Value)***
  - Only the address of the object is transferred to the function. ***(Pass-by-Reference)***
- The pass-by-reference method is more efficient since it requires to pass only the address of the object and not the entire object.

Objects as Function Arguments

- An object can also be passed as an argument to a non-member function.

- Such functions can have access to the public member functions only through the objects passed as arguments to it.

- These functions cannot have access to the private data members.

Example

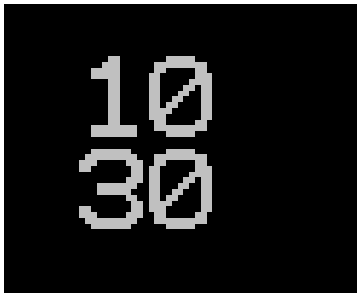
```
class Demo_reference
{
   public:
            int aa;
   public:
            void hello(Demo_reference db)
            {
            db.aa=20;
            }
            void hello1(Demo_reference& db)
            {
            db.aa=30;
            }
};
```

```
main()
{
   Demo_reference a1,a2;

   a1.aa=10;
   a2.aa=10;
   a1.hello(a1);
   a2.hello1(a2);

   cout<<a1.aa<<endl;
   cout<<a2.aa;
}
```

Output:



Friendly Functions

- The private members can not be accessed from outside the class.
- A non-member function can not have an access to the private data of a class.
- However ……. ?

Friendly Functions

- C++ allows a common function to be made friendly with more than one classes, thereby allowing the function to have access to the private data of these classes.
- Such a function need not be a member of these classes.
- To make an outside function friendly to a class, we have to simply declare this function as a friend of the class.

Friendly Functions

- The function declaration should be preceded by the keyword ***friend***.
- The function is defined elsewhere in the program like a normal C++ function.
- The function definition does not use either the keyword friend or the scope operator : :

```
class employee
{
    ---
    ---
  public :
    ---
    ---
    friend void it_cal
   (void);
}
```

Friendly Functions

- The functions that are declared with the keyword friend are known as ***friend function***.
- A function can be declared as a friend in any number of classes.
- A friend function, although not a member function, has full access right to the private members of the class.

Friendly Functions

Special Characteristics:

- It is not in the scope of the class to which it has been declared as friend.
- Since it is not in the scope of the class, it cannot be called using the object of the class.
- It can be invoked like a normal function without the help of any object.

Friendly Functions

Special Characteristics:

- Unlike member functions, it cannot access the member names directly and has to use an object name and dot membership operator with each member name.
- It can be declared either in the public or private part of a class without affecting its meaning.
- Usually, it has objects as arguments.

Friendly Functions

- Member function of one class can be friend functions of another class.
- In such cases, they are defined using the **scope resolution operator** as:

```
class X
{
    ...
    ...
    int fun1 ( );
    ...
};
```

```
class Y
{
    ...
    ...
    friend int X : : fun1 ( );
    ...
};
```

Friend class

- We can also declare all the member functions of one class as the friend functions of another class.
- In such cases, the class is called a ***friend class***.

```
class Z
{
    ...
    ...
    friend class X ;
    ...
};
```


CHARUSAT
Chapter – 6 : Classes And Objects

Returning Objects

Like a function can receive objects as arguments, it can also return objects.

Const Member Functions

- If a member function does not alter any data in the class, then it is called a const member function.

void mul (int, int) const ;

void get_balance( ) const ;

- The qualifier const is appended to the function prototypes (in both declaration and definition). The compiler will generate an error message if such functions try to alter the data values.

Pointer To Members

- It is possible to take the address of a member of a class and assign it to a pointer.
- The address of a member can be obtained by applying the operator & to a fully qualified class member name.
- A class member pointer can be declared using the operator : : * with the class name.

Pointer To Members

- We can define a pointer to the member m as follows:

int A : : * pm = &A : : m;

A : : * "pointer-to-member of A class".

&A : : m means "address of the m member of A class".

```
class A
{
  private :
    int m ;
  public :
    void show( ) ;
} ;
```

Pointer To Members

- The dereferencing operator .* is used when the object itself is used with the member pointer.
- The dereferencing operator ->* is used to access a member when we use pointers to both the object and the member.

```
class A
{
    int m ;
  public :
    void show( ) ;
} ;
A a ;
int A : : * pm = & A : : m ;
A * pa = & a ;
```

Pointer To Members

- The dereferencing operator

.* is used when the object itself is used with the member pointer.

- The dereferencing operator

->* is used to access a member when we use pointers to both the object and the member.

```
class A
{
    int m ;
  public :
    void show( ) ;
} ;
A a ;
int A : : * pm = & A : : m ;
A * pa = & a ;
```

To refer the member m a .* pm

To refer the member m pa -> * pm

Pointer To Members

- We can also design pointers to member functions which, then, can be invoked using the dereferencing operators in the main.

(object-name . * pointer-to-member function) ( )

(pointer-to-object -> * pointer-to-member function) ( )

The precedence of ( ) is higher than that of . * and -> * , so the parentheses are necessary.

Local Classes

- Classes can be defined and used inside a function or a block. Such classes are called local classes.
- Local classes can be used global variables and static variables but can not use automatic variables. The global variables should be used with the scope operator (: :)
- They cannot have static data members and member functions must be defined inside the local classes.

Review Questions

Answer the following questions :

1. What is a friend function?
2. What is the difference between a C structure and a C++ class?
3. What is the significance of classes in OOPs?
4. How do you define Static Data member ?
5. What do you understand by Memory Allocation of an Object?

Presentation Prepared By:

Ms. Krishna Patel

Ms. Khushi Patel

Contact us:

dweepnagarg.ce@charusat.ac.in
parthgoel.ce@charusat.ac.in
hardikjayswal.it@charusat.ac.in
dipakramoliya.ce@charusat.ac.in
krishnapatel.ce@charusat.ac.in
khushipatel.ce@charusat.ac.in

Subject Teachers:

Ms. Dweepna Garg
Subject Coordinator

Mr. Parth Goel
https://parthgoelblog.wordpress.com

Mr. Hardik Jayswal