Translated from French to English - www.onlinedoctranslator.com

Course: C++ Programming

Chapter 2: C++ Classes

Directed by:
Dr. Sakka Rouis Taoufik

https://github.com/srtaoufik/coursCpp

1

I.Introduction

The class mechanism in C++ allows programmers to define new types specific to their applications.

A class is a very strong extension of the C language's notion of data structure. Within a class, we can define data and methods. These entities constitute the members of the class.

2

II.Definition of a class

In C++ the syntax of a class is as follows:

```
[struct | class | union] ClassName {
// Modifier d'acces for the dataeits members
//Donneits members
// Modifier d'acces for member functions
//Member functions
} ;
```

Remark: The code for a member function can be done In or outside of the class. In the second case, the method name will be prefixed by the class name followed by the operator **::**

3

II.Definition of a class

Example:

```
struct Point {
private :
  // Dedeclaration of 2 dataer-type memberse the
          float abs ;
          float ord ;
public:
  void initialize (float a, float b) {
           abs=a;
           ord=b;
 }
 void move (float dx, float dy);
} ;

void Point::move (float dx, float dy) {
          abs+=dx;
          ord+=dy;
}
```

4

III.Object and instance of a class

An object corresponds to the physical location of a copy (or instance) of the class. Each instance of the class (or object) has its own copies of data (non-static) of the class.
Member data (non-static) of a class will generally be called the components of an object.

Example :

```
class Point {
        static int num ;
        int abs ;
        int ord ;
/* nb is a static data member while abs and ord are not static*/…
};
```

Remark:The member (non-static) functions of a class, on the other hand, are defined to be applied to instances of the class. For example: Point P1; … P1.display() ;

5

III.Object and instance of a class

Creating an instance involves a memory allocation mechanism. Each memory allocation has a specific lifespan.

- **Static lifetime**: Static objects are those created by a declaration located outside of any function.
- **Automatic lifetime**: these objects are allocated from the memory stack with a lifetime corresponding to the execution of the enclosing block. Automatic objects are those created by a declaration in a function or in a block.
- **Temporal lifespan**: temporal objects are created for the purpose of evaluating an instruction, they are destroyed at the end of the expression evaluation.
- **Explicit lifespan**: these objects are created dynamically by an explicit request (using malloc and new). Their destruction is the responsibility of the programmer.

6

III.Object and instance of a class

Example:

```
Point M ;
//global object (hardee static)

void main(){
        Point A, B ; //automatic object ;
        Point *P = new Point ; //creexplicit stack ation
        static Point C ; //static object
        ....
        delete p ; //destruction  explicit
}
```


IV.Protection of members of a class

It is possible to prevent access to fields or certain methods by any function other than those of the class. This operation is called encapsulation to do this, you need to use the following keywords:

- *public*: access is free;
- *private*: accesses are allowed in the class functions only;
- *protected*: access is allowed in the functions of the class and its descendants (see the inheritance chapter) only.

Remark: The keyword *protected* is only used in class inheritance.

IV.Protection of members of a class

Example :

```
struct client {
private:   // Private data:
    char Name [20], First Name [20];
    unsigned int Date_Entry;
    int Balance;
    // There is no private method.
public:    // Public data and methods:
    // There is no public data.
    bool in_the_red (void);
    bool good_client (void);
};
```


IV.Protection of members of a class

RQ:By default, classes built with ***struct*** have all their members public. It is possible to declare a class whose elements are all private by default. To do this, simply use the keyword ***class*** instead of the keyword ***struct***.

Example :

```
class Client{
    // private is now useless.
    char Name [21], First Name [21];
    unsigned int Date_Entry;
    int Balance;
public: // Public data and methods.
    bool in_the_red(void);
    bool good_client(void);
};
//…
Client A, B;
int s= A.Balance; //error because Balance is private
bool b= A.dans_le_rouge() ; //ok because this method is public
```

V. The functions objects

Object functions are member functions of a class that are not declared static.

V.1 Declaration of a member function

Declaring an object function is the same as the usual definition, however, declared in the class declaration block, it is bound to that particular class.

Example :

```
class Account {
        int amount;
public :
        void init(int); // object function
        int balance(); // object function
        void depot(int); // object function
} ;
void display (Account); //usual function
```


V. The functions objects

The difference between an object function and a regular function is that the member function has an **implicit argument** which is the object on which it is applied. The function will be applied to an object, class instance.

These functions are used with the classic "." or "." notation.→» depending on whether the function is applied to an object or a pointer to an object.

Example :

```
void main() {
        Account C1;
        Account *P=&C1 ;
        C1.initialize(30);
        P→deposit (500); // C1.depot(500);
}
```

V. The functions objects

V.2 Definition of a member function

The definition of an object function can be done outside the definition block of its belonging class.
When defining an object function, its name is prefixed by the name of the class it belongs to followed by the scope resolution operator**::**

Example :

```
void Account :: initialize(int m ) {
  amount =m ;
}

void Point :: display(){
   cout<< "this point has coordinates "<<x<< "and " <<y <<endl ;
}
```


V. The functions objects

V.3 Defining an inline member function

The inline function mechanism allows you to eliminate the cost of calling a function by substituting the code for that function at the level of the function's call instruction. Similarly, a member function can be defined inline.

Examples:

Either explicitly using the inline keyword	**Either implicitly in the body of the class definition**

class Account {
  int amount;
public :
  int balance() ;
} ;
inline int Account :: balance ( ) {
  return amount;
}
	class Account {
	  int amount;
	public :
	  int balance ( ) {
	  return amount ;}
	} ;

VI.Static data and member functions of a class

In this paragraph we will see the use of the keyword***static***in classes.
This keyword is used to characterize:
-static data members of classes,
-static member functions of classes,
-static data of member functions.

VI.Static data and member functions of a class

VI.1. Static Member Data

A static data member (class variable) is a global variable of the class.
It is not an object component.
The definition of a class variable is used to store information about all instances of the class.

Example :

```
class Point {
  //x and y are 2 private non-static data members
  int x, y ;
  //nb is a static data member
  static int num ;
} ;
int Point :: nb=0 ; //explicit initialization
```

Remark: The variable Point::nb will be shared by all objects of class Point, and its initial value is zero.

VI.Static data and member functions of a class

VI.1. Static Member Data

Non-static data: linked to an object

Static data: class-related (provides information about the class)

Remark: Defining static data members follows the same rules as defining global variables. That is, they behave like variables declared externally.

The static variables of member functions must be initialized inside the member functions. They also belong to the class, not to objects. In addition, their scope is reduced to that of the block in which they were declared.

VI.Static data and member functions of a class

VI.1. Static Member Data

Example :

```
#include <stdio.h>
class Test {
public:
   int fn(void);
};
int Test::fn (void){
   static int count=0;   // static variable
   return count++;
}
int main (void){
   Test object1, object2;
   printf("%d", object1.fn() );   // Displays 0
   printf ("%d \n ", object2.fn());  // Poster 1
   return 0;
}
```

Will display *0* and *1*, because the static variable *account* is the same for both objects.

VI.Static data and member functions of a class

VI.2. Static member functions

Classes can also contain static member functions.

This may seem surprising at first glance, since member functions already belong to the class, i.e. to all objects.

In fact, this means that these member functions will not receive the pointer to the *this* object, as is the case for other member functions.

➔ they will only be able to access the static data of the object.

VI.Static data and member functions of a class

VI.2. Static member functions

Example :

```
class Integer {
    int i ;
    static int j ;
public:
    static int get_value (void);
} ;
int Integer::j=0 ;
int Integer::get_value (void) {
    j=1; // Legal.
    return i; // ERROR! get_value cannot access i.
}
```

The function get_value from the above example cannot access the non-static data member i, because it does not work on any object. Its scope of action is only the Integer class. On the other hand, it can modify the static variable j, since this belongs to the class Integer and not to objects of this class.

VI.Static data and member functions of a class

VI.2. Static member functions

Calling static member functions is done exactly like calling non-static member functions, by specifying the identifier of one of the objects of the class and the name of the member function, separated by a period.

We can therefore simply call a static function by qualifying its name with the name of the class to which it belongs using the scope resolution operator**::**

VI.Static data and member functions of a class

VI.2. Static member functions

Example :

```
class Integer {
    static int i;
public:
    static int get_value (void);
};
int Integer::i=3;
int Integer::get_value(void){
    return i;
}
void main(){
    // Calls the static function get_value
    int result;
    result =Integer::get_value();
}
```

VII.Friendly functions and classes

Sometimes it's necessary to have functions that have unlimited access to a class's fields. Usually, using such functions reflects a lack of analysis in the class hierarchy, but not always. So they're still necessary regardless.

Such functions are called friend functions. For a function to be a friend of a class, it must be declared in the class with the keyword *friend*.

It is also possible to make a class a friend of another class, but in this case, this class should perhaps be a child class. Using friend classes may reflect a design flaw.

VII.Friendly functions and classes

VII.1. Friendly functions

Friend functions are declared by preceding the classic function declaration with the keyword *friend*inside the target class declaration. Friend functions are not methods of the class, however (this would not make sense since methods already have access to class members).

Example :

```
class Point {
    int x,y ;                    // Private data.
    friend void init(int, int);// The init function is a friend function.
};
Point A;
void init (int abs, int ord) {
   Ax=abs ;// accesses private data of object A
   Ay=ord ;
}
```

VII.Friendly functions and classes

It is possible to declare a function of another class as friend, by specifying its full name using the scope resolution operator.

Example :

```
class Point {
    int x, y ; // A private data.
    friend void Segment::init(int, int, init, int); // A friend function.
};
class Segment{
  Point G, D;
  public :
    void init (int abs1, int ord1, int abs2, int ord2 ){
            Gx=abs1 ;// accesses private data of object A
            Gy=ord1 ;
            Dx=abs2 ;// accesses private data of object B
            Dy=ord2 ;
   }
} ;// end class B
```

➔ The init method of the Segment class can use the private members of any instance object of the Point class.

VII.Friendly functions and classes

VII.2. Friendly classes

To make all methods of a class that is friends with another class, you just need to declare the entire class as a friend. To do this, you must once again use the keyword *friend* before the class declaration, inside the target class.

Attention:This time again, the declared friend class will not be a subclass of the target class, but rather a globally scoped class.

VII.Friendly functions and classes

VII.2. Friendly classes

Example :

```
#include <iostream.h>
class A {
   friend class B; // All methods of B are friends.
   int i; // Private data of class A.
public:
   void A (void) {
      i=0;
     }
};
A A1;
class B {
public:
   void print_A() {
      cout<< A1.i <<endl; // Accesses A's private data.
      }
};
void main(void){
   B b1;
   b1.print_A();
  }
```