C++ Programming

Ch. 10 Objects and Classes

Spring 2014

Myung-Il Roh

Department of Naval Architecture and Ocean Engineering
Seoul National University

Ch. 10 Objects and Classes

Contents

- ☑ Abstraction and Classes
- ☑ Class Declaration
- ☑ Definition of Class Methods
- ☑ Using Classes
- ☑ Constructor
- ☑ Destructor
- ☑ The ‘this’ Pointer
- ☑ Creating Arrays of Objects
- ☑ Example of Using Class
- ☑ Summary
- ☑ Practice

Abstraction and Classes (1/2)

: To express a set of complex data to the user identification unit (by an identifier)

- To implement the abstract interface to the user-defined type in C++
- Class = Set of " " or Set of " "

Abstraction and Classes (2/2)

☑ Composition of a Class

(*.h)

- To describe the data component, in terms of
- To describe the public interface, in terms of (termed methods)

(*.cpp)

- To describe how member functions are implemented

☑ Features of the Class

- Binding of data and methods into a single unit
- ‘ ’ and ‘ ’ keywords: To describe access control for class members
- Data hiding: To define normal data with ‘ ’ and member functions with ‘ ’

Class Declaration

❑ " " keyword (private identification)
- It can (data hiding).
- A private member can be accessed only through the public member functions or friend functions.
- Default access control

Describing data by class members

❑ " " keyword (public identification)
- It identifies class members that constitute the public interface for the class (can be access from outside).
- It represents abstract components.
- In general, .

Describing public interface by member functions

* Member functions are also called 'methods'.

```
class Stock
{
private:
        char company[30];
        int shares;
        double share_val;
        double total_val;
        void set_tot() { ... }
public:
        void acquire(...);
        void buy(...);
        void sell(...);
        void update(...);
        void show();
};
```

Definition of Class Methods

☑ Implementation of Class Methods (Member Functions)

- The function header for a member function uses
  ➡ To indicate to which class the function belongs
  - void Stock::update(double price)
  - void bufoon::update()
- Class methods can .

☑ Application of Class Methods to an Object

(Declaration of class variables)

- Stock kate, joe;

with declared objects

- kate.show();
- joe.show();

- Each object we create contains storage for its own internal variables, i.e., class members.
- All objects of the same class share the same set of class methods, with just copy of each method.

Using Classes

☑ Procedures for Applying a Class to a Program

- **Step 1:** (member functions)
- **Step 2:**
- **Step 3:**

- **Definition of class and methods**
  - **Input of data values**
    - Company: NanoSmart
    - Shares: 20
    - Share Price: $12.50
    - Total Worth: $250.00
  - **Declaration of data output method**
    - void show();

Using Classes
- Step 1: Declaration and Definition of Class and Methods

```
#include <iostream>
using namespace std;
#include <cstring>

class Stock
{
private:
    char company[30];
    int shares;
    double share_val;
    double total_val;
    void set_tot() { total_val = shares * share_val; }

public:
    void acquire(const char* co, int n, double pr);
    void show();
};
```

```
// for using strncpy() function

// Company name
// Number of shares
// Share price
// Total worth

// Input of data by accessing
   to private members
```

Using Classes
- Step 2: Implementation of Methods

```
void Stock::acquire(const char * co, int n, double pr)
{
    strncpy(company, co, 29);
    company[29] = '₩0';
    shares = n;
    share_val = pr;
    set_tot();
}

void Stock::show( )
{
    cout << "Company name: " << company
         << " Number of shares: " << shares << '₩n'
         << " Price: $" << share_val
         << " Total worth: $" << total_val << '₩n';
}
```

Access to the each private member and input data

* 'private' member
- company[]: Company name
- shares: Number of shares
- share_val: Share price
- set_tot(): Total worth

Using Classes
- Step 3: Creation of Class Objects

```
int main()
{
  Stock stock1;

  stock1.acquire("Smart",  20,  2.50);

  stock1.show();

  return 0;
}
```

Creation of an object "stock1"

Calling of the object's class method
(Access of private members through the class method)

Constructors (1/3)

☑ Constructors

- **Prototype for constructor**
  - Ex.
    ```
    Stock(const char* co, int n = 0, ...);
    ```
- **Constructor definition**
  - Ex.
    ```
    Stock::Stock(const char* co, int n, double pr)
    {
        ...
    }
    ```
  - The constructor has no declared type ( ).
- **Features of constructors**
  - Timing when constructors are called: When a class object has defined
  - They can define and initialize data members at once.
  - Objects doesn't call constructors, but constructors are being used to create objects.

Constructors (2/3)

☑ Using Constructors (Way to Initialize an Object)

- Method 1: Calling of the constructor
  - Ex.
    Stock food = Stock("world cabbage", 250, 1.25);

- Method 2: Calling of the constructor
  - Ex.
    Stock food("world cabbage", 250, 1.25);

- Method 3: Use of
  - Ex.
    Stock *food = Stock("world cabbage", 250, 1.25);

Constructors (3/3)

☑ Default Constructors

- If the program doesn't provide a constructor, the compiler automatically supplies a default constructor.
- Providing a non-default constructor without providing a default constructor is error!!

- Way to define default constructors
  - Method 1: Providing default values for all the arguments to the existing constructor
    - Ex. Stock(const char* co = "Error", int n = 0, double pr = 0.0);
  - Method 2: Using function overloading to define the second constructor which has no arguments
    - Ex. Stock();
- Declaration of object variables without initializing them explicitly
  - Stock Stock1 = Stock(); // Call default constructor explicitly.
  - Stock Stock1; // Call default constructor implicitly.

Destructors (1/2)

☑ Destructors

- A program automatically calls a do-nothing destructor if we don't provide a destructor. If our constructor uses 'new' to allocate memory, the destructor should be use 'delete' to free that memory.

- Prototype for destructor
  - Ex.
    ~Stock();

- Destructor definition
  - Ex.

```
Stock::~Stock()
{
    ...
}
```

Destructors (2/2)

☑ Calling Destructors

- Our code shouldn't explicitly call a destructor (called automatically by the compiler).
- If we create a static storage class object, the destructor is called automatically when the program terminates.
- If we create an automatic storage class object, the destructor is called automatically when the program exits the block of code in which the object is defined.
- If the object is created by using 'new', the destructor is called automatically when we use 'delete' to free the memory.

The ‘this’ pointer (1/4)

☑ A pointer that points to the object itself used to invoke a member function

☑ Ex. A method that returns a reference of larger value between two ‘Stock’ objects

- Method for prototype declaration
- Ex.

Stock&) ;

: Passing the object to the function ‘topval’ by calling by reference

: The function won’t modify the implicitly accessed object.

: The function returns a reference to one of two constant objects, the return type also has to be a constant reference.

The 'this' pointer (2/4)

- Method prototype declaration (continued)
- Ex.

```
class Stock
{
private:
    char company[30];
    int shares;
    double share_val;
    double total_val;
    void set_tot() { total_val = shares * share_val; }
public:
    Stock();                        // default constructor
    Stock(const char * co, int n, double pr);
    ~Stock() {}                     // do-nothing destructor
    void show() const;
    const Stock& topval(const Stock &s) const;
};
```

The 'this' pointer (3/4)

- Method definition and calling
- Ex.

```
const Stock& Stock::topval(const Stock &s) const
{
    if (s.total_val > total_val)
        return s;
    else
        return itself?
}
```

Cf. Is it possible that 'total_val' which is private can access to 'total_val' in 's'?

Accessing control is done by the class level. Thus, objects in the same class can access to the private components of each other. Therefore, it is possible that a method of one object can access to the private component ('s.total_val') of another object ('s') in the same class.

The ‘this’ pointer (4/4)

Stock kate(“W.Inc”, 100, 63);

Stock joe(“P.Inc”, 120, 30);

```
const Stock& Stock::topval(const Stock &s) const
{
    if (s.total_val > total_val)
        return s;
    else
        return *this;
}
```

kate.topval(joe);

This invokes ‘topval()’ with ‘kate’, so ‘s’ is ‘joe’, this points to ‘kate’, and ‘*this’ is ‘kate’.

joe.topval(kate);

This invokes ‘topval()’ with ‘joe’, so ‘s’ is ‘kate’, this points to ‘joe’, and ‘*this’ is ‘joe’.

An Array of Objects

☑ We can create multiple objects which are in the same class.

☑ Array of Objects and Its Uses

- Ex.

```
Stock mystuff[4];          // Create an array of 4 Stock objects.
mystuff[3].show();         // Apply 'show()' method to 4th element.
```

☑ Initialization of the array

- Ex. Use a constructor to initialize the array elements.

```
Stock stocks[STKS] = {
    Stock("NanoSmart", 12, 20.0),
    Stock("Boffo Objects", 200, 2.0),
    Stock("Monolithic Obelisks", 130, 3.25),
    Stock("Fleep Enterprises", 60, 6.5)
};
```

Example of Using Classes

☑ An example of using constructor, destructor, this pointer, and array of objects

- Output the most valuable shares you have

- Input data

Company name	No.of Shares	Price
NanoSmart	12	$20.0
Boffo	200	$2.0
Fleep	60	$6.5

- Programming procedures
  - Definition of a class
  - Implementation of class methods
  - Implementation of the 'main' function

Example of Using Classes
- Class Definition

* Private (data members)
- Company name: 'company'
- Number of shares: 'shares'
- Price: 'share_val'

```
#ifndef _STOCK_H_
#define _STOCK_H_

class Stock
{
private:
    char company[30];
    int shares;
    double share_val;
    double total_val;
    void set_tot()
    { total_val = shares * share_val;}
```

* Public (methods)
- Output function of members: 'show()'
- Comparing function of two objects: 'topval( )'

```
public:
    Stock();                    // Default constructor

    Stock(const char * co, int n, double pr);
    ~Stock() {}
    void show() const;
    const Stock& topval(const Stock &s) const;
};

#endif
```

Example of Using Classes
- Implementation of Class Methods

* Cf.
- Definition of the method
- Using 'this' pointer

```
void Stock::show() const
{
    cout << "Company name : " << company
    << "  Number of shares : " << shares << '\n'
    << "  Price : $" << share_val
    << "  Total worth : $" << total_val << '\n';
}

const Stock& Stock::topval(const Stock &s) const
{
    if (                              )
            return   ;
    else
            return       ;  //
}
```

Example of Using Classes
- Implementation of the 'main' Function

* **Cf.**
- **Initialization of an array of objects**
- **Calling function of the array of objects:**
  stock[st].show( );
  top.topval(stocks[st]);

```
const int STKS = 3;

int main()
{
    Stock stocks[STKS] = {
        Stock("NanoSmart", 12, 20.0),
        Stock("Boffo Objects", 200, 2.0),
        Stock("Fleep Enterprises", 60, 6.5)
    };

    int st;
    for (st = 0; st < STKS; st++)
        stocks[st].show();
    Stock top = stocks[0];
    for (st = 1; st < STKS; st++)
        top = top.topval(stocks[st]);
    cout << "₩nMost valuable shares:₩n";
    top.show();

    return 0;
}
```

Summary

, and

.

, whereas

, also called methods,

. The class combines , and the private aspect accomplishes data hiding.

☑ If we want a member function to act on more than one object, we can pass additional objects to the method as arguments.

☑ If a method needs to refer explicitly to the object that evoked it, it can use the this pointer. The this pointer is set to the address of the evoking object, so .

Practice

- ☑ Make an illustrated animal book program using ‘animal’ class. Define class and its methods and using them on your program.
  - The ‘animal’ class has the kind (int), the height (average height, double), and the weight (average weight, double) as class members.
  - A default constructor should be “0” (unknown), height = 0.0, weight = 0.0 when there are no arguments. If there are three arguments, the first parameter is the kind, and the second and third parameters should be the height and the weight.
  - Define a public method that prints out the average height and the average weight of the animal when we input the kind.