CPS235 Object Oriented Programming in C++

LAB2 BESE-15A *8th March 2010*

Objects and Classes in C++

Objectives

By the end of this lab, you should be able to

- Create classes with public and private members
- Create constructors with/without arguments
- Use default parameters in constructors
- Create member functions of class
- Pass objects as arguments to functions and return objects from functions
- Use const and static variables and functions
- Make your own copy constructor
- Use destructors
- Use enumerations

Instructions

- This is a graded lab.
- If you have any problems, you are encouraged to consult with me.
- Complete the lab within the lab hours and submit it to the following folder
  \\csdept\data\Assignments\Lec Aisha Khalid\OOP\15A\Lab2
- Make sure that you show your programs to me and answer any questions that I may have in order to get full credit for the lab.

Question 1:

The class declaration for a Stock class is given below. Implement the class functions.

```
class Stock
{
      private:
            char company[30];
             int shares;
             double share_val;
             double total_val;
              void set_tot()       //sets total_val equal to the product
                                   of shares and share_val

      public:
             Stock();
             //constructor sets company to “no name”, shares to 0,
             share_val and total_val to 0.0. Use the strcpy function to
             copy character arrays
```

```
        Stock(const char co[], int n, double pr);
        //constructor sets company to co,shares to n and share_val
        to pr, then calls the set_tot() function


       void buy(int num, double price);
       //This function is called when a company buys some shares.
       In this case, the value in num is added to the shares
       variable of the object, and the price of the shares is
       stored in the share_val variable. The total_val variable is
       updated using the set_tot function


        void sell(int num, double price);
       //This function is called when a company sells some shares.
       In this case, the value in num is subtracted from the
       shares variable of the object, and the price of the shares
       is stored in the share_val variable. The total_val variable
       is updated using the set_tot function


        void update (double price);
       //this function is called to update the price of a share,
       it must also call the set_tot() function to update
       total_val


   void show();
   //displays all data members of the invoking object

};
```

1. Which functions in the above class declaration can be made constant? Make them constant in your code.
2. Write a function called total_count() which gives the value of the total number of objects created at any time.
3. Create a copy constructor for the above class which also updates the count of objects created at any time. Print a line in the copy constructor which says “Copy Constructor called for object:” and then call the show function to display which object has been copied.
4. Create a destructor for the class, which decrements the value of your count variable. Print a line in the destructor which says “Destructor called for object:” and then call the show function to display which object has been destroyed.
5. To show when the copy constructor and destructor have been used, you will have to create a member function, call it test() which takes as argument a Stock Object passed to it by value. Make a function which accomplishes this task and test it in main.
6. In the main function, create an array of Stock Objects which can hold 4 Stock Objects initialized to

   “PTCL”, 12, 20.0

"Engro Foods", 200, 22.0
"ACBL", 120, 43.25
"Attock Oil", 50, 345.45

The syntax for making an array of objects is given below:
You can call a different constructor for different elements of the array. You may declare a Stock array of 10 elements, but assign values to only the first 3. In this case, the remaining 7 elements will be initialized using the default constructor.

```
const int SIZE = 10;
Stock stocks[SIZE] = {Stock("PTCL",12,20.0),Stock(),
                      Stock("Engro Foods", 3, 45.6)};
```

Do the following in main:

- Display the stock information of all 4 objects
- Perform operations on the objects to implement the following:
  - PTCL buys 4 shares priced at Rs 45.5 each
  - Engro Foods sells 10 shares priced at Rs. 40 each
  - Attock Oil also sells 30 shares each priced at Rs. 300.0
  - Call the update function on the respective objects after each buying/selling operation.
- Display the stock information of all updated objects
- Call a function which you have made previously to test the copy constructor and destructor.

Question 2:

Create a class called **`time`.** The time class has three private data members of type **int**, named **`hours`, `minutes`** and **`seconds`.**

- Provide a no-argument constructor for the class which initializes the data members to 0. Use the initializer list for initialization.
- Provide another constructor which initializes the data members to values provided in the main function. It should also validate whether the values are correct i.e., **`hours`** should be between 0 and 23 (inclusive), **`minutes`** should be between 0 and 60 (inclusive) and **`seconds`** should also be between 0 and 60 (inclusive). If an invalid value is entered, the constructor should automatically change it to 0.
- Provide a 3-arg constructor to initialize the data members of the class with default arguments set to 0. It performs the same tasks being performed by the previous two constructors. While testing this constructor in main, you should make sure that you comment out the other two constructors.
- Write a function **`getTime()`** which asks the user to input the values for hours, minutes and seconds. This must also validate the input as in the above case.
- Write a function **`displayTime()`** which displays the time in the format 11:39:45 i.e., **`hours:minutes:seconds.`**
- Write a function that adds two objects of type **`time`** passed as arguments. The time should be added in such a way that if the sum of the **`seconds`** exceeds 60, that should subtract 60 from the **`seconds`** and add 1 to the **`minutes`**. Similarly if the sum of the **`minutes`** exceeds 60, subtract 60 from **`minutes`** and add 1 to **`hours`**. If the **`hours`** exceed 23, then it should be set to **`hours % 24`**. So the sum of 12:59:54 and 12:40:44 would be 1:40:38
- Create a function **`time_to_secs(),`** which takes as argument an object of type **`time`** and returns the equivalent in seconds (type **`long`**) where

```
long totalsecs = t1.hours*3600 + t1.minutes*60+ t1.seconds;
```

- Create another function, **`secs_to_time()`** that takes as argument a time in seconds (type **`long`**) and returns an Object of type **`time`**.
- Write a main program that defines objects in different ways, providing none, one, two or three arguments. Write statements to test all member functions of the class.
- Create an array of 10 **`time`** objects. Prompt the user to enter a **`time`** value, store it in the array, then ask him if he wants to enter any more. If the answer is 'y', continue to ask for input but if the answer is 'n', then stop taking any further input. Display all the **`time`** values entered by the user.

Question3:

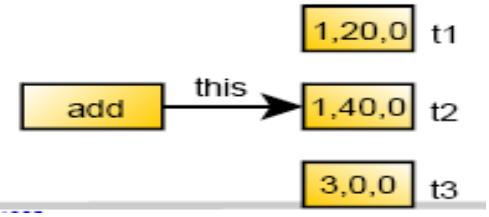
The **`this`** pointer is a pointer accessible only within the nonstatic member functions of a class. It points to the object for which the member function is called.

```
class time
{  int      hours;
   int      minutes;
   int      seconds;

            time( );
            time(h, m);
   time     add(time t1, time t2);
};

int main( )
{  time    t1(1, 2, 0);
   time    t2(1, 4, 0);
   time    t3;

   t3 = t2.add(t1);
}
```



You can use the this pointer in one of the following two ways:

```
(*this).nameOfMemberVariable
this->nameOfMemberVariable
```

Rewrite the Distance Class as given in lecture slides. In the following function, use the **`this`** pointer explicitly to refer to the invoking object.

```
Distance Distance :: add_dist(Distance d2){
      Distance temp;
      temp.inches = (*this).inches + d2.inches;

      if (temp.inches >= 12.0)
      {     temp.inches -= 12.0;
            temp.feet =1;
      }
      temp.feet += (*this).feet + d2.feet;
      return temp;}
```

Write a main program to test the functions of your class.

Question4: Enumerations

An enumeration is a user-defined type that consists of a set of named constants known as enumerators. A declaration gives the type a name and specifies the permissible values (*the enumerators)*. Definitions can then create variables of this type. Internally enumeration variables are treated as integers

```
enum  Colour  { eRED, eBLUE, eYELLOW, eGREEN, eSILVERGREY,eBURGUNDY };
Colour auto_colour;
…
auto_colour = eBURGUNDY;
```

An enum is a form of integer. So,

```
cout << auto_colour;      //Would print 5
```

Enumerators are stored by compiler as integers. By default, first enumerator is 0, next enumerator value is previous enumerator value + 1.
When defining enumeration it is possible to specify integer constant for every enumerator

```
enum e_acomany {
      Audi=4,
      BMW=5,
      Cadillac=11,
      Ford=44,
      Jaguar=45,
      Lexus,
      Maybach=55,
      RollsRoyce=65,
      Saab=111
};
```

By the rule "next enumerator value is previous + 1", the value of "Lexus" enumerator is 46.
By default, enumerated values start from 0 but you can make them start from a different number e.g.,

```
enum Suit {clubs = 1, diamonds, spades, hearts};
```

This would give a value of 1 to clubs, 2 to diamonds and so on…
You can perform arithmetic and relational operations on enumerated types since they are stored as integers
The following assignment may only generate a warning but it will still compile

```
auto_color = 5;
```

Other examples of possible uses of enumerations include:

```
enum week { Mon=1, Tue, Wed, Thu, Fri Sat, Sun} ;

enum escapes { BELL = '\a', BACKSPACE = '\b', HTAB = '\t', RETURN =
'\r', NEWLINE = '\n', VTAB = '\v' };

enum boolean { FALSE = 0, TRUE };
```

Implement the program on page 261 of Robert Lafore's book under the heading Arrays of Cards. See if you can understand how the enumerators are being used.