CS224 Object Oriented Programming and Design Methodologies Lab Manual

Lab 05 - Classes

Department of Computer Science

Dhanani School of Science and Engineering

Habib University

1 Objects and Classes

Often working on large programming projects we run into scenarios where the program keeps getting more and more complex, as the project keeps getting larger the complexities increase. Code becomes less and less reusable and readable. And if there are multiple programmers working on the project than it gets even harder to manage.

We have looked into one way to make our lives easier and thats breaking our program into smaller functions, but at sometimes even that approach doesn't help us much.

To solve this problem, we have the notion of *object-oriented programming*, a programming style in which tasks are solved by collaborating objects. The fundamental idea behind object-oriented programming is to create singular units that combine both *data* and the *functions* that operate on that data. Such units are called an *objects*. Each object has its own set of data, and it's own set of functions that can act upon the data. These functions are called *member functions* or *methods* and these data items are called *attributes* or *instance variables*.

Ideally we wouldn't just want to implement a single object that would defeats the purpose of this entire exercise. Instead we implement a *class*. A class describes a set of objects with the same behavior. For example the string class describes the behavior of all strings. The class specifies how a string stores its characters, which member functions can be used with strings, and how the member functions are implemented. Similarly a class for mammals would describe the general characteristics for mammals, a class for cars would do this for cars. We can even have a class for something like a cash register, this class would hold the data items a cash register has and its functionality. The code listing 1 shows the cash register class.

```
class CashRegister
{
    public:
        void clear();
        void add_item(double price);
        double get_total() const;
        int get_count() const;
    private:
        int item_count;
        double total_price;
};
```

Listing 1: sample code for a Cash Register class

Once we have created a such a class we can also create objects (instances of the class). These objects can hold data as in the attributes and also call member functions for other functionality. Listing 2 shows a sample code where an object of the CashRegister class is created in the main function and member functions are called on it.

```

class CashRegister
{
    public:
        void clear();
        void add_item(double price);
        double get_total() const;
        int get_count() const;
    private:
        int item_count;
        double total_price;
};


int main()
{
```

```
    CashRegister register_1;
    register_1.add_item(42.0);
    register_1.get_count();
    return 0;
}
```

Listing 2: sample code for a Cash Register class with a sample main function

Among member functions of a class we there is a special member function called a *constructor*. A constructor is a member function that initializes the data members of an object. The constructor is automatically called whenever an object is created. By supplying a constructor, you can ensure that all data members are properly set before any member functions act on an object.

Now lets look at an example of a class that would be more useful in our programming context. A *counter* is a variable that counts things. Each time some event takes place, the counter is incremented. The counter can also be accessed to find the current count. Let's assume that this counter is important in the program and must be accessed by many different functions. In procedural languages such as C, a counter would probably be implemented as a global variable. However, global variables complicate the program's design and may be modified accidentally. In this example, counter, provides a counter variable that can be modified only through its member functions. Listing 3 shows a sample code for such a counter class.

```
// counter.cpp
// object represents a counter variable

#include <iostream>
using namespace std;

////////////////////////////////////////////////////////////////

class Counter
{
    private:
        unsigned int count;
    public:
        Counter() : count(0)
            { /*empty body*/ }
        void inc_count()
            { count++; }
        int get_count()
            { return count; }
//count
//constructor
//increment count
//return count
};
////////////////////////////////////////////////////////////////

int main()
{
    Counter c1, c2;                     //define and initialize
    cout << "\nc1=" << c1.get_count();  //display
    cout << "\nc2=" << c2.get_count();
    c1.inc_count();
    c2.inc_count();
    c2.inc_count();
    //increment c1
    //increment c2
    //increment c2
    cout << "\nc1=" << c1.get_count();  //display again
    cout << "\nc2=" << c2.get_count();

    cout << endl;
```

```
42      return 0;
43  }
```

Listing 3: Sample code for the counter class

2 Problem

The lab objectives of this lab is to get familiar with concept of classes and objects in C++. The following problems all tests this.

1. Get rich quick scheme

After taking CS224 from friend university (the best liberal arts college in pakistan with a world class campus and infrastructure) you realize CS might not be as fun as they made it out to be. Feeling fooled and betrayed you decide to focus your efforts on more important matters; *getting rich quick*. You along with your "gang" of narcissistic misfits, came up with a scheme. You decide to install an illegal tollbooth on the Pehlwan Goth road.

Your friend Frank who is supposedly the "money guy" suggests that each passing car should pay 50 Rupees toll. As you have no legal jurisdiction to collect toll some cars may pass by without paying anything. However, you are taking "intelligence pills" which happen to make you a master programmer.

You decide to program this tollbooth with these new found programming skills and the trauma from CS224.

Model this tollbooth with a class called *TollBooth*. The class should have the following:

- The two data items are a type *unsigned int* to hold the total number of cars, and to hold the total amount of money collected.
- A constructor that initializes total number of cars, and to hold the total amount of money collected to 0.
- A member function called *payingCar()* that increments the car total and adds 50 to the cash total.
- A member function called *nopayCar()*, increments the car total but adds nothing to the cash total.
- A member function called *display()* displays the two totals.

Make appropriate member functions const.

Write a main function that creates and object of class *TollBooth*. Then you take input in a loop, if you have input "p" that means a paying car had passed by, if you have input "n" that means a non-paying car had passed by. On input "q" the loop should terminate and print the number of cars that passed by and the total amount of money earned.

Input Format For Custom Testing
Each line consist of p or n. p indicates when paying car passes, n indicates a non-paying car passes. The last input is q, when loop terminates, and totals are shown.

Sample Case 0
Sample Input For Custom Testing
p p p n n p q
Sample Output
Total cars passed: 6 Total toll collected: Rs. 200
Explanation
Total 6 cars passed, 4 paid the toll hence total collected toll is Rs. 200

2. Stack Class (using Arrays)

Stacks are an important abstract data type that is used in many programming applications. Often times you would have to implement your own stack class to create stack objects to use in your code.

In this problem, you have to implement Stack ADT by creating a class that provides interfaces for the following basic operations. Create a class called *MyStack*. The class should not be initialized without specifying the size of the stack. Your code must take care of memory leaks and release all memory resources.

You have to implement the following member functions in the MyStack class:

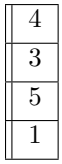
- **void push(int);** This function adds an element to the top of the stack.
- **int pop();** This function remove and return element from the top of the stack.
- **int top();** This function return the element at the top of the stack without removing it. Return -1 if stack is empty.
- **bool isempty();** This function return true if stack is empty, false otherwise.

The details of Stack Operations have been described below.

- **initialize** $\longrightarrow$ the constructor initializes an empty stack.

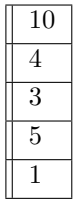
  

  Then items may be "pushed" into the stack to add elements to it. Imagine after certain operations the stack looks like the following:

  The next operations would describe how the stack can be further modified.

- **Push** $\longrightarrow$ The push operation adds a new element to the stack. As stated above, any element added to the stack goes at the top, so push adds an element at the top of a stack. So if we push 10 to the stack shown above we get the stack in the following state:



- **Pop** $\longrightarrow$ The pop operation removes and also returns the top-most (or most recent element) from the stack. So if we pop the stack shown above the stack would be in the following state:

- **Top** $\longrightarrow$ The Top operations only returns (doesn't remove) the top-most element of a stack.

TOP() = 4

- **isEmpty** $\longrightarrow$ This operation checks whether a stack is empty or not i.e., if there is any element present in the stack or not.

IS_EMPTY() = FALSE

3. Complex Calculator

Complex numbers are important number system in mathematics with applications in various areas of computer science such in computer graphics and quantum computing.

In this problem we are going to define a class of complex numbers. As in previous labs we have implemented complex numbers without using classes or using any sort of object orientation, this exercise will illustrate some advantages of using classes and having and object orientated approach.

Define a class Complex, the class will have two private attributes: *real* and *imag* both of type *double*. Define the necessary constructors to create objects of complex type: A default constructor that initializes the attributes with zero, and an overloaded constructor that initializes the attributes with supplied values.

The class will also have methods to *add*, *subtract*, and *multiply* complex numbers. Since complex numbers can be added/multiplied/subtracted with a real number as well, so the add, subtract

and multiply functions should be overloaded: one where operand is Complex numbers, other where operand is real.

The class will also have a method called *show*, that display the complex number in usual way i.e. $x + yi$

Main function is already given in listing 4, you have to make it working.

Refer to this page for multiplication of complex numbers:
https://www.mathsisfun.com/algebra/complex-number-multiply.html

Sample Input For Custom Testing:
2 5 5 3 10

Sample Output:
c1: $2 + 5i$ c2: $5 + 3i$ d1: 10 c1+c2: $7 + 8i$ c1-c2: $-3 + 2i$ c1*c2: $-5 + 31i$ c1+d1: $12 + 5i$ c1-d1: $-8 + 5i$ c1*d1: $20 + 50i$

Explanation
First input line is c1 = 2+5i, second line is c2 = 5+3i, third line is a real number d1 = 10. The result of addition/subtraction/multiplication are given as output.

```
int main()
{
    double real, imag;
    cin>>real>>imag;
    Complex c1 = {real, imag};
    cin>>real>>imag;
    Complex c2 = {real, imag};
    double d1;
    cin>>d1;
    Complex result;
    //showing the numbers:
    cout<<"c1: "; c1.show();
    cout<<"c2: "; c2.show();
    cout<<"d1: "<<d1<<endl;
    // Check the opertions where both operands are complex
    result = c1.add(c2);
    cout<<"c1+c2: "; result.show();
    result = c1.subtract(c2);
    cout<<"c1-c2: "; result.show();
    result = c1.multiply(c2);
    cout<<"c1*c2: "; result.show();
    // Check the opertions where one operator is complex, other is double
    result = c1.add(d1);
    cout<<"c1+d1: "; result.show();
    result = c1.subtract(d1);
    cout<<"c1-d1: "; result.show();
    result = c1.multiply(d1);
```

```
28      cout<<"c1*d1: "; result.show();
29      return 0;
30  }
```

Listing 4: "Main function for complex calculator"