EECS 280 Lab 06: Abstract Data Types

Due Sunday, 21 February 2016, 8:00pm

In this lab, you will practice building and using Abstract Data Types (ADTs). We'll juxtapose both the C style (structs) and the C++ style (classes) of implementing ADTs here in order to highlight the differences between the two, but keep in mind it would probably be better to pick and use one consistent approach in real programming projects.

This lab covers material from these lectures:

- 09 Structs and Classes
- 10 Abstract Data Types

Requirements

You may work on this lab either individually or in groups of 2-3. Include your name(s) in the comments at the top of the file. Submit the files below on CTools. You do not need to turn in any other files. If you work in a group, each person must submit a copy in order to receive credit.

Files to submit:

- `lab06.cpp`

Completion Criteria:

To pass this lab, you must finish tasks 1-3. There are no optional tasks in this lab.
This checklist will give you an idea of what we look for when grading for completion:

- ✓ (Task 1) Implement the functions `EmployeeRecord_init`, `EmployeeRecord_promote` and `EmployeeRecord_demote` in lab06.cpp.
- ✓ (Task 2) Implement the constructor for `Employee` in `lab06.cpp`.
- ✓ (Task 2) Implement the member functions promote and demote for `Employee` in `lab06.cpp.`
- ✓ (Task 3) Create 2 employees using `EmployeeRecord` and add them to the `eecsSoft` vector
- ✓ (Task 3) Create 2 employees using `Employee` and add them to the `umichWorks` vector
- ✓ (Task 3) Promote and demote an employee of each type.

Task 0 - Preliminaries

The Files

We have provided starter files for this lab. If you have a terminal open, the following command will automatically download it from the EECS 280 Google Drive repository to your current directory:

```
    $ wget goo.gl/V4k0Cp -O - | tar xzk
```

In case you are working locally and want to manually download the files, they are also attached to the CTools assignment and available on the course Google Drive Repository (see link in header).

Here's a brief summary of the files included in this lab. Files you need to turn in are shown with a **red** background.

`lab06.h`	Contains declarations for the `EmployeeRecord` struct and accompanying functions, as well as the class definition for `Employee.`
`lab06.cpp`	Contains definitions for the `EmployeeRecord` functions and `Employee`'s member functions. Includes the `main` function and testing code.

Testing Code

`lab06.cpp` contains a `main` function with testing code we've written for you. Compile it with:

```
    g++ -Wall -Werror -O1 -pedantic lab06.cpp -o lab06
```

The testing code in `main` will print out employee info for eecsSoft and umichWorks. You must implement tasks 1 and 2 before going to task 3.

Introduction

In this lab, we will develop two ways to represent an employee of a company; one using structs, and the other using classes. We will then represent two companies, eecsSoft and umichWorks, who must save employee records, and be able to promote/demote employees. eecsSoft prefers coding C Style, so they want to use structs to represent employees and write other functions to set/update employee records. umichWorks prefers coding C++ Style, so they will use classes to represent employees and use member functions to set/update employee records.

Task 1 - EmployeeRecord (C Style, Structs)

In the C style, we use structs to group together individual member variables (data, but not operations) into an aggregate type. Here, we represent the employee with their name, gender, age and rank.

```
struct EmployeeRecord

{

    string name;

    string gender;

    int age;

    int rank;

};
```

We would also like to perform operations on EmployeeRecords, and the C style way to do this is by defining functions that take those objects as parameters **by pointer** (why?). The following are already declared in `lab06.h` and implemented in `lab06.cpp`.

```
// EFFECTS: Prints Name, Gender, Age and Rank of employee
void EmployeeRecord_printInfo( const EmployeeRecord *rec );
```

We also need a function for promote, demote and set record - **to complete task 1**, fill in the empty implementations for the `promote`, `demote` and `setRecord` functions in `lab06.cpp`.

```
// MODIFIES: EmployeeRecord rec
// EFFECTS: Sets all fields in EmplyeeRecord rec
void EmployeeRecord_init( EmployeeRecord *rec, string name, string
gender, int age, int rank );

// MODIFIES: rank in EmployeeRecord rec
// EFFECTS: Increases rank by one
void EmployeeRecord_promote( EmployeeRecord *rec );

// MODIFIES: rank in EmployeeRecord rec
// EFFECTS: Decreases rank by one
void EmployeeRecord_demote( EmployeeRecord *rec );
```

Task 2 - Employee Objects (C++ Style, Classes)

We will now implement objects for representing employees, as well as promoting and demoting them. Following the C++ style, we declare a class that specifies the member variables and functions (data and operations) an object will have. `lab06.h` already contains a partial declaration of the `Projectile` class.

```
class Employee
{
      // OVERVIEW: A representation of an employee,
      //           as well as actions we can take on them
```

The data members, which are declared as `private`, are name, gender, age and rank information on the employee. They are represented as two strings and two integers, respectively.

```
private:
            string name;
            string gender;
            int age;
            int rank;
```

In the C++ style, when objects are instantiated (created) for the first time, a *constructor* is called that allows initial set up of the object's internal state. You need to implement this as part of Task 2.

```
public:
  // Constructor
  // EFFECTS: Sets employee with name, gender, age and rank information
  Employee(string in_name, string in_gender, int in_age,
           int in_rank );
```

The change you need to make for task 2 is to implement the Constructor for Employee, as well as implementing promote and demote in this way:

- `promote` will raise the rank member variable by 1
- `demote` will lower the rank member variable by 1

Task 3 - Being the boss at eecsSoft and umichWorks

Now, we will finally be able to use our new ADTs to represent records for two Silicon Valley start-ups: eecsSoft and umichWorks.

As mentioned earlier, eecsSoft loves using C Style programming. So we will represent the company using a `vector` of `EmployeeRecord`.

```
vector<EmployeeRecord> eecsSoft;
```

We also need to represent umichWorks. We know they prefer C++ Style programming, so we will represent the company using a `vector` of `Employee.`

```
vector<Employee> umichWorks;
```

The first part of Task 3 is to create **two** employees at eecsSoft and add them to the `eecsSoft` vector then create **two more** employees at umichWorks and add them to the `umichWorks` vector (Hint: to add an item to a vector, use the `push_back` method).

It turns out some of our employees are performing above expectations, while other are slacking. Therefore, for **promote one employee and demote another at eecsSoft** here:

```
cout << "Welcome to eecsSoft! Here are our loyal employees: " << endl;
      for( int index = 0; index < eecsSoftSize; ++index )
      {
            EmployeeRecord_printInfo( &eecsSoft[index] );
      }

      // TASK 3: Promote one employee and demote the other at eecsSoft 
here

      cout << "Here is the new info for the employees of eecsSoft: " << 
endl;
      for( int index = 0; index < eecsSoftSize; ++index )
      {
            EmployeeRecord_printInfo( &eecsSoft[index] );
      }
```

(**Note:** When promoting and demoting employees, please make sure to call `EmployeeRecord_promote` and `EmployeeRecord_demote` on the EmployeeRecords you have inside the vector. You can access Employee 1 with eecsSoft[0], and Employee 2 with eecsSoft[1])

Finally, do the same as umichWorks. **Promote one employee and demote another at umichWorks** here:

```
cout << endl << "Welcome to umichWorks! Here are our loyal employees: " <<
endl;
      for( int index = 0; index < umichWorksSize; ++index )
      {
            umichWorks[index].printInfo();
      }

      // TASK 3: Promote one employee and demote the other at umichWorks
here

      cout << "Here is the new info for the employees of umichWorks: " <<
endl;
      for( int index = 0; index < umichWorksSize; ++index )
      {
            umichWorks[index].printInfo();
      }
```

(**Note:** When promoting and demoting employees, please make sure to call `promote` and `demote` on the Employees you have inside the vector. You can access Employee 1 with umichWorks[0], and Employee 2 with umichWorks[1])