CMPT 115 – Assignment #4

2014-2015 WT2

Due: Friday, February 6, 2015 at 10:00pm (2300)

Total Marks: 55

Version History

Written Questions (30 pts)

Reminder: whenever we ask you to "write an algorithm", we want a complete algorithm, including a header, written in pseudocode. We will deduct 25% if your algorithms are in C++. Remember, pseudocode is a tool for you to focus on larger issues, rather than syntax and minutiae related to coding. This 25% penalty is an attempt to get you to practice designing an algorithm before you implement it in some language.

Use a single text file/document to contain all written questions (exercises 1-3). Microsoft Word documents (doc,docx) should not be handed in, because of incompatibilities between versions. Use text-only format (.txt) or PDF format (.pdf). If you are not sure about this, ask your TA. If the marker cannot open your file (for whatever reason), you will receive a grade of 0 for the written part.

Name your submission file **a4-written.txt**. Ensure that each answer is clearly marked for the question it relates to; and ensure that your file/document is clearly marked with your name, student number and NSID at the top. For written questions and algorithms, you can use <- instead of ←.

Exercise 1 Student ADT (20 pts)

In this question, we want you to design an ADT called Student similar to that from the last assignment, to represent some of the information that a class list of students might need. It won't be completely realistic, but it will help you put into practice the concepts of ADTs.

Using pseudo-code, complete the following steps:

1. In the previous assignment you were to develop a student record. In this assignment we will build a more complete student record. A Student record should include
    a. a student's name (first, and last),
    b. a student number,
    c. an NSID (aaannn format)
    d. an array of 10 assignment grades,
    e. a midterm exam grade and a final exam grade.

    Design a record type to contain all this data. Add any additional fields that you think might be necessary to make this record type complete or manage it more effectively.
2. Design and implement the following operations (modify those from the previous assignment if that helps (remember designing software should include adaptability to changing needs):
    - **createStudentRecord()** – creates a new blank student record on the heap and returns a reference to the record
    - **destroyStudentRecord(s)** – de-allocates the student record **s**
    - **displayStudentRecord(s)** – displays the student record **s** on the console
    - **readStudentRecordFromConsole()** – obtains data for a student record from the input stream and returns a reference to the new populated student record
    - **changeAssignmentGradeForStudent(s,a,g)** – adds a given number **g** to the grade of a given assignment number **a** for a given student record **s**

- **changeExamGradeForStudent(s,x,e) –** adds a given number **e** to the **x** ('M' = midterm, 'F' = final) exam grade of a given student record **s**

The parameters and details of these operations are not precisely stated, so make some reasonable assumptions to assist your design. Your records should be allocated on the heap, and you should use references to records as parameters.

Every operation should have a full algorithm header, including pre-, post- and return conditions. Every function should have a body that describes the function in enough detail that you can simply translate it to C++ when the time comes.

What to hand in:

In your file a4-written.txt, indicate the question (exercise) number very clearly, so it can be seen easily by a marker. Your algorithms (with pre and post-conditions and return criteria) should be in this file.

Grading

• 18 marks, 3 marks per function. Your algorithms have pre- post- and return criteria. Your algorithm bodies are reasonable for a pseudo-code design. You've paid attention to the use of dynamic memory allocations, and have the basics correct.

• 2 marks for modifying any existing solutions effectively for this requirement

Exercise 2 A Student ADT test application (10 pts)

Design an algorithm to test your Student ADT. Your answer should be in the form of pseudo-code, like main(), in which you call all your ADT operations, and check that they worked. Since this is pseudo-code, and not C++, you obviously cannot run this application, but a careful design of a test-application can make your development work a lot easier, so you should get in the habit of it.

For this algorithm, the pre-and post-conditions are trivial, since this is essentially a "main" program. Feel free to omit them. **You may draw on the testing algorithms from your previous assignment but with the proper modifications made necessary by the changed requirements of this assignment.**

What to hand in:

In your file a4-written.txt, indicate the question (exercise) number very clearly, so it can be seen easily by a marker. Your algorithm along with pre and post conditions should be in this file.

Grading

- 3 marks: Your algorithm creates and destroys at least one student record.
- 4 marks: Your algorithm calls each function at least once.
- 3 marks: Your algorithm would present evidence that your functions are working correctly or not.

Programming Questions (25 pts)

All C++ source code files must contain your name, student number, and NSID; and the course, assignment number, and question number as comments at the top of the file. All C++ programs must compile under **g++** without errors or warnings, using the standard flags described in tutorial, namely **-Wall -pedantic**. Use of **String** class or other object-oriented code (except for **cin** or **cout** and related file-stream objects, which are allowed) will result in a flat deduction of 25%. (That means: don' t use advanced techniques to avoid learning the concepts we are attempting to teach you!)

Note: All students are encouraged to bring assignments for a 5-min code review. Simply bring your graded assignment to one of the instructors or TAs during their help desk hours, within two weeks after it had been graded, and we will give you personalized suggestions based on your own code, and a bonus mark of 10% of this assignment value (ex., 10 bonus marks for an assignment with a total mark of 100).

Exercise 4 Tutorial 4 Exercise (5 pts)

Submit your tutorial 4 exercises along with the assignment. The lab exercises worth 4% of your final mark.

What to hand in:

Your C++ code for Lab 4: **lab4exercise1.cc, myTime.cc, myTime.h, testTime.cc**

File-redirection Tutorial

This part is a preparation for doing the next question.

So far in the course, we have been using the console to communicate between us and our running programs. This gets a bit tedious. Fortunately, the UNIX command-line provides a tool that we can use to speed up our interactions, so that we don' t have to type. Note that the technique we describe is available on the command-line, but not in Eclipse. So you must use the command line for this!

The tool is called " file-redirection." What happens is that we can tell UNIX to connect **cin** to a file, so that every use of **cin** takes data from the file (in the order it appears in the file). The redirection is indicated on the command-line, and not in your program. From the point of view of your program, it looks like **cin** is taking input from the console, as usual. Your program doesn' t need to know that file-redirection is happening at all. In UNIX, the file-redirection operator looks like "<" and can be pronounced "from."

Example 1. This is an example that will read **cin** input from a text file given as an argument when running its executable.

The input data file, named `input.txt`, contains four integer numbers, one floating point number, one string (with no space in it), and one character separated by spaces, as follows:

12 3.3 8 hello 50 c 70

The example source code that reads in the data is (the source code is in file redirect.cc)

```
// File-redirection tutorial
//
#include <iostream>
using namespace s td ;
int main ( ) {
        int i1, i2, i3, i4;
        char c;
        char s[20];
        float f ;

        cin >> i1 >> f >> i2 >> s >> i3 >> c >> i4 ;
        cout << "The integers read in from file are : "
                << i1 << " " << i2 << " " << i3 << " " << i4 << endl
                << "The float number read in from file is : " << f << endl
                << "The character read in from file is : " << c << endl
                << "The string read in from file is : " << s << endl ;
        return 0 ;
}
```

Then you can compile your program using the following command:

```
g++ -Wall -pedantic -o myProg redirection.cc
```

And run your program like this:

```
./myProg < input.txt
```

Provided that the file input.txt is in the same folder as your program, the console will then output:

```
The integers read in from file are: 12 8 50 70
The float number read in from file is: 3.3
The character read in from file is: c
The string read in from file is: hello
```

Note: The program itself knows nothing about the file; using file-redirection, UNIX simply responds to each call to **cin** with the next piece of data from the file to the program as if it were typed in by a user. Again, this is not something you can do (easily) with Eclipse. If you are using Visual Studio, simply launch a COMMAND window in Windows, change directory (cd) to the directory where the executable is, then enter the executable name, in full, and the redirect. The standard **cout** will route to the command window while the **cin** will redirect to the input file.

We will show you how to read files within a program, but we'll leave that for another time.

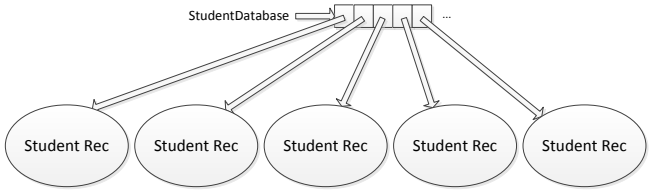
Note: When you use file redirection, every *cin* you use in your program will be connected to the file provided on the command-line, until your program ends. If your program consumes all the data in the file, all uses of *cin* after that will silently fail, as no further data is in the file. This will look like your program is simply skipping over the *cin* statements.

Exercise 5 Dynamic Memory, References (20 pts)

We will work on an array of student records and input data for the students. The student record type is defined as:

```
struct Student {
  char *name;  // array of characters making up the name
  int score;
};
```

Your array will contain *references* to Student records, not the actual records themselves. Thus the name of your array is a reference to a reference of student data, shown here pictorially:



The database is provided in a file called *database.txt*, where the first line is the number of students in the file, and each of the remaining lines represents one student record, where the string before the space is the student's name, and the number after the space is the student's score. For example, the first three lines of the file are:

```
200
Aaliyah 54
Aaron 73
```

It means that there are 200 students in this database and the first student's name is Aaliyah, with a score of 54.

Your task is to implement three functionalities:

1. Dynamically allocate an array of references to Student records, the size of which is determined by the number of students read in from the first line of the file. Each element in the array is a reference to exactly one dynamically allocated Student record.
2. The **getData** algorithm reads the students' data from the database file.

Algorithm getData (arrStu, sSize)
Read the student information from console and save to array
Pre: **arrStu :: An array of references to Student records**
sSize :: Integer, the number of students read in from the database file, which is also the size of the array arrStu.
Post: each student's name and score read in from the database file is saved into the corresponding attributes of a dynamically allocated structure and a reference to that structure is stored in the array elements in arrStu.
Return: nothing

3. The prtData algorithm displays the student data, located by index, on the console.

```
Algorithm prtData (arrStu, sSize, iS)
Output the data associated with the student in the student-array at index
   iS
Pre:
    arrStu :: An array of references to Student records
    sSize :: Integer, the size of the array arrStu.
    iS :: Integer, the location within the student-array of the target student;
     where 0 <= iS < aSize
Post: console displays the student data for arrStu[iS].
Return: nothing
```

What to hand in:

- Hand in your design document in a file called a4q5design.txt, including all pseudo-code, with pre- and post-conditions, and return criteria. Demonstrate that you did the design work for this exercise.
- Hand in your C++ program in a file called a4q5.cc or a4q5.cpp. It should reflect your design document.
- A text-document called a4q5testing.txt that shows your program working on a few test cases. You may copy/paste from the console.

Grading

- 10 marks: Your design document is complete, with algorithms describing the task at several levels of abstraction. Your algorithms have pre- and post-conditions, etc. The lowest level of abstraction is suitable for translation into C++.
- 5 marks: Your implementation is correct, and follows the design you submitted. Your program describes the task using functions that have pre- and post- conditions, etc, as internal documentation (comments).
- 4 marks: You demonstrated your program working with at least 3 test cases.
- 1 mark: Your name, student number and NSID on all algorithms and in source code.

The List of Files for Assignment #4

1. a4-written.txt
2. lab4exercise1.cc, myTime.cc, myTime.h, testTime.cc
3. a4q5design.txt
4. a4q5.cc (or a4q5.cpp)
5. a4q5testing.txt