Lab 7: Numerical Conversions and Structures

Assigned: *Tuesday, November 17th, 2020*
Due: *Monday, November 23rd, 2020*
Points: *100*

- There is NO MAKEUP for missed assignments.
- We are strict about enforcing the LATE POLICY for all assignments (see syllabus).

Introduction

The assignment for this week will utilize everything you've learned so far in the context of doing number conversions and working with C++ structures.

It is HIGHLY RECOMMENDED that you develop the algorithms for each program FIRST and THEN develop the C++ code for it.

Step 1: Getting Ready

First, open a terminal window and log into the correct machine. Change into your CS 16 directory, create a **lab07** directory and change into it. There are no skeleton files provided for you.

Step 2: Create and Edit Your C++ Files

This week, you will need to create **3 files: converter.cpp** (for the 1st Exercise) and **headers.h** & **functions.cpp** (for the 2nd Exercise). These correspond to one of the 2 exercises listed below, which make up this lab. Each exercise is worth 50 points and should be solved in its own file and everything must be submitted for full assignment credit.

IMPORTANT NOTE: This lab will be also graded for use of comments and style. See below for details. In addition, we will take *major* points OFF if you use C++ instructions/libraries/code that either (a) was not covered in class, or (b) was found to be copied from outside sources (or each other) without proper citation.

converter.cpp

This program will ask the user for a positive, non-zero, decimal number (cannot be larger than what a C++ integer will allow, that is 2^{31} - 1) and it returns the value in binary, octal, and hexadecimal.

REQUIREMENTS, ASSUMPTIONS:

1. Your program must have a function called **`convertFromDec`** that takes in 2 arguments: an **`unsigned int`** representing a decimal number, and an **`unsigned int`** representing a converting base. It will return a string of the converted number. The declaration must be:

```
string convertFromDec(unsigned int decnum, unsigned int base);
```

2. The function **`convertFromDec`** will convert the decimal number into a number in the converting base and return the converted number as a string. When you design it, you do NOT need to have it convert **`decnum`** into ANY base: ONLY into binary, octal, and hexadecimal.

3. The main function will ask the user, over and over again, to enter a positive decimal number or anything else to quit. If what the user enters is NOT a positive number, the program should quit with no other output given (remember, you cannot use **`exit()`** or **`break`** here!)

4. You can have other functions in the program, if you like.

5. Be sure to utilize ONLY techniques we've covered in lecture. Do not use "special" arrays or pointers, do not use vectors, special functions that do conversion automatically for you, etc… You will get zero points if you do. You are expected to do the conversions algorithmically (see point 1 in the "Hints" below).

6. If you use the **`stoi()`** function, you can only use it how it's been used in class!

HINTS:

1. The best way to design the function **`convertFromDec`** is to implement the algorithm we discussed in lecture that can convert a decimal number into any base by dividing and using/updating the remainders and quotients.
2. A good and easy way to make sure the user input is valid is to get input as a string then convert it into the appropriate integer.
3. Dealing with strings will also help you with making conversions into hexadecimal numbers (because of the letter digits in hexadecimal).

Here is a sample run of the program. Make sure your outputs look like this exactly (no trailing whitespaces, newlines between each iteration of the question asked.

```
$ ./converter
Enter positive decimal number (anything else quits): 7
7 in binary is: 111
7 in octal is: 7
7 in hex is: 7

Enter positive decimal number (anything else quits): 13
13 in binary is: 1101
13 in octal is: 15
13 in hex is: D

Enter positive decimal number (anything else quits): 882
882 in binary is: 1101110010
882 in octal is: 1562
882 in hex is: 372

Enter positive decimal number (anything else quits): 156780
156780 in binary is: 100110010001101100
156780 in octal is: 462154
156780 in hex is: 2646C

Enter positive decimal number (anything else quits): 0
$ ./converter
Enter positive decimal number (anything else quits): -22
$ ./converter
Enter positive decimal number (anything else quits): whaaat?
$
```

records.cpp

This program will ask the user for inputs on (fictitious) student information, will store these in an array of a certain structure, and present the data, sorted, in an output file.

I will give you 3 files (on our webpage): 1 main program file (**records.cpp**) that you will use as-is (no changes allowed to it) and 2 skeleton header and function definitions files (**headers_skeleton.h** and **functions_skeleton.cpp**, both of which should be renamed **headers.h** and **functions.cpp** and then submitted to Gradescope).

REQUIREMENTS, ASSUMPTIONS, and HINTS:

1. This program has to define and use a structure called **`UndergradStudents`**. This structure should contain student ID numbers, first and last names (2 separate variables), major, and GPA scores for each undergraduate year (first step is to realize what data types they need to be).
    a. Only first and last names can have whitespaces in them.
    b. This structure has to be defined in **headers.h**

2. In the **`main()`** function (which, again, you will not write), the program declares an array of 20 objects of this structure, called **`ObjArray`**. This is to be used as a partially-filled array: in other words, it can contain fewer than, but not more than, 20 objects.

3. In the **`main()`** function, the program calls the function **`InitializeStructures`**, which is already declared for you. You cannot change its declaration. This function has to initialize all the values of the array of **`UndergradStudents`** by **user input**. This must be the ONLY way you do your initializations.

4. This function should ask the user to enter the first name, last name, major, and each year's GPA, in a loop (over and over again) and assign these values accordingly. The user can quit the input loop by entering a "X" character for the "Enter first name for student *N*" question. The loop has to quit if it reaches 20 repeated query sets (see sample run example).

    Example (see sample run for a fuller example with multiple entries):

```
Enter first name for student 1 (or X to quit): Jimmy
Enter last name for student 1: Manicotti
Enter major for student 1: EE
Enter GPA Year 1 for student 1: 3.2
Enter GPA Year 2 for student 1: 3.5
Enter GPA Year 3 for student 1: 3.7
Enter GPA Year 4 for student 1: 4.0
```

5. In the **`main()`** function, the program calls the function **`WriteResults`** which writes sorted results in an output file that has to be called "**Results.txt**". The order of the information written in this file is specific – look at the sample run for details. **In addition**, these results (a) must be presented **sorted in ascending alphabetical order of the last name** (NOTE: do not sort by first name if 2 last names are the same), and (b) must show each student's **average GPA score**, calculated with a precision of **2** decimal places.

    Example (see sample run for a fuller example with multiple entries):

```
These are the sorted results:
ID# 1, Manicotti, Jimmy, EE, Average GPA: 3.60
```

6. **HINT:** You are going to need at least 3 functions in this program: the 2 mentioned above (**`InitializeStructures`** and **`WriteResults`**) and a function that can sort an array of **`UndergradStudents`**. This can be designed based on Bubble Sort without much trouble.

7. You will NOT be submitting the **records.cpp** file. We will use our own, which will look exactly like the one I've given you, and printed below:

```
// USE THIS FILE AS-IS - DO NOT CHANGE ANY OF THIS
//
#include <iostream>
#include <string>
#include <fstream>
using namespace std;

void InitializeStructures(UndergradStudents us[], int &size);
void WriteResults(ofstream &outf, UndergradStudents us[], int size);

// YOU HAVE TO COMPLETE THESE 2 FILES:
#include "headers.h"
#include "functions.cpp"

int main() {
    UndergradStudents ObjArray[20];
    int ObjArraySize;
    ofstream outf;

    InitializeStructures(ObjArray, ObjArraySize);
    WriteResults(outf, ObjArray, ObjArraySize);

    return 0;
}
```

8. A sample run would look like this. Note the empty line between each batch of queries and how the user quits the run of questions.

```
$./records
Enter first name for student 1 (or X to quit): Jimmy
Enter last name for student 1: Manicotti
Enter major for student 1: EE
Enter GPA Year 1 for student 1: 3.2
Enter GPA Year 2 for student 1: 3.5
Enter GPA Year 3 for student 1: 3.7
Enter GPA Year 4 for student 1: 4.0

Enter first name for student 2 (or X to quit): Rula
Enter last name for student 2: Abadai
Enter major for student 2: CS
Enter GPA Year 1 for student 2: 4
Enter GPA Year 2 for student 2: 3.7
Enter GPA Year 3 for student 2: 3.8
Enter GPA Year 4 for student 2: 4

Enter first name for student 3 (or X to quit): Xavier "Bobby"
Enter last name for student 3: Xenophenoius
Enter major for student 3: PHIL
Enter GPA Year 1 for student 3: 2.2
Enter GPA Year 2 for student 3: 1.3
Enter GPA Year 3 for student 3: 3.9
Enter GPA Year 4 for student 3: 4.0

Enter first name for student 4 (or X to quit): Adam
Enter last name for student 4: Aardvark
Enter major for student 4: CS
Enter GPA Year 1 for student 4: 3
Enter GPA Year 2 for student 4: 3
Enter GPA Year 3 for student 4: 3
Enter GPA Year 4 for student 4: 3.1

Enter first name for student 5 (or X to quit): Paula Nancy
Enter last name for student 5: Manitoba-Smith-McFugazi
Enter major for student 5: COMM
Enter GPA Year 1 for student 5: 4
Enter GPA Year 2 for student 5: 4
Enter GPA Year 3 for student 5: 3.1
Enter GPA Year 4 for student 5: 2.7

Enter first name for student 6 (or X to quit): X
$ cat Results.txt
These are the sorted results:
ID# 4, Aardvark, Adam, CS, Average GPA: 3.02
ID# 2, Abadai, Rula, CS, Average GPA: 3.88
ID# 1, Manicotti, Jimmy, EE, Average GPA: 3.60
ID# 5, Manitoba-Smith-McFugazi, Paula Nancy, COMM, Average GPA: 3.45
ID# 3, Xenophenoius, Xavier "Bobby", PHIL, Average GPA: 2.85
$
```

The file contents should look exactly like the example given (I'm only showing you an example with 5 entries – the program can process anywhere between 1 and 20 entries). **As you can see, (a) the list is presented sorted in ascending alphabetical order of the last name, and (b) each student's *average* GPA score is shown (calculated with a precision of 2 decimal places).**

Step 3: Create a makefile and Compile the Codes with the make Command

In order to learn another way to manage our source codes and their compilations, we will first create a **makefile** and put in the usual g++ commands in it. Afterwards, whenever we want to compile our programs, the Linux command is a lot shorter – so this is a convenience.

Using your text editor, create a new file called **makefile** and enter the following into it:

```
all: converter records

converter: converter.cpp
        g++ -o converter converter.cpp -Wall -std=c++11

records: records.cpp
        g++ -o records records.cpp -Wall -std=c++11
```

Then from the Linux prompt, you can do one of two things: either issue separate compile commands for each project, like so:

```
$ make records
```

Or, you can issue one command that will compile all the projects mentioned in the **makefile**, like so:

```
$ make
```

If the compilation is successful, you will not see any output from the compiler. You can then run your programs, for example:

```
$ ./records
```

If you encounter an error, use the compiler hints and examine the line in question. If the compiler message is not sufficient to identify the error, you can search online to see when the error occurs in general.

Remember to re-compile the relevant files after you make any changes to the C++ code.

Step 4: Submit using GRADESCOPE

BEFORE YOU SUBMIT YOUR FINAL VERSION FILES: Make sure that you:

a) **STYLIZE** your program appropriately to additionally make it easier on others reading your code. Apply ALL the style pointers I have posted on Piazza. This includes the use of comments.

This will be graded for an *extra 20 points* beyond the automatic grade you get from Gradescope.

b) **MEET THE REQUIREMENTS OF THIS LAB**. This means, if I said certain features or program aspects were *required* to be in your final code, then they must be there.

If these requirements are not met, you will lose MAJOR points on the exercise. This simulates a real-world situation (e.g. your customer wants certain features, so you must provide it – while I am not your customer, per se, I *am* giving you points for your efforts!)

Log into your account on **https://www.gradescope.com/** and navigate to our course site. Select this assignment. Then click on the "Submit" button on the bottom right corner to make a submission. You will be given the option of uploading files from your local machine or submitting the code that is in a GitHub repo. Choose the first option and follow the steps to **upload ALL 3 files** to Gradescope.

You may submit this lab multiple times. You should submit only after you test it on your computer (and CSIL) and are satisfied that the programs run correctly. The score of the last submission uploaded before the deadline will be used as your assignment grade.

Step 5: Done!

Once your submission receives a score of 100/100 on the autograder, you are done with this assignment. REMEMBER that there are 20 additional points that will be scored according to your proper use of comments and styling. The total will then be normalized to 100 points again (i.e. 120 score will be 100%) to grade this lab.

WE WILL BE CHECKING FOR PLAGIARISM – DO NOT COPY FROM OTHER STUDENTS OR FROM SOURCES ONLINE! USE PROPER CITATION OF CODE THAT YOU DID NOT WRITE! THE CONSEQUENCES WILL - AT MINIMUM - BE A ZERO ON THIS LAB AND POSSIBLY A ZERO IN THIS COURSE AND YOU WILL BE REPORTED TO THE UNIVERSITY.