CSCI 104 – Homework 1

Data Structures and Object Oriented Design

HW1

- Directory name in your github repository for this homework (case sensitive): `hw1`
  - Once you have cloned your `hw_usc-username` repo, create this `hw1` folder underneath it (i.e. `hw_usc-username/hw1`)
  - If your `hw_usc-username` repo has not been created yet, please do your work in a separate folder and you can copy over relevant files before submitting

A Few Notes on Repositories

1. Never clone one repo into another. If you have a folder `cs104` on your VM and you clone your personal repo `hw_usc-username` under it (i.e. `cs104/hw_usc-username`) then whenever you want to clone some other repo, you need to do it back up in the `cs104` folder or other location, NOT in the `hw_usc-username` folder.
2. Your repos may not be ready immediately but be sure to create your GitHub account and fill out the GitHub information form linked to at the end of Lab 01.

Skeleton Code

On many occasions we will want to distribute skeleton code, tests, and other pertinent files. To do this we have made a separate repository, `homework-resources`, under our class GitHub site. You should clone this repository to your laptop and do a `git pull` regularly to check for updates.

```
$ git clone git@github.com:usc-csci104-spring2016/homework-resources.git
```

Again, be sure you don't clone this repo into your `hw_usc-username` repo but at some higher up point like in a `cs104` folder on your laptop.

Note: If you can't access the repository, you can find the files here and here

Problem 1 (Course Policies, 10%)

Carefully study the information on the course web site, then answer the following questions about course policies:

Place your answers to this question in a file name `hw1.txt`

Part (a):

Which of the following are acceptable behaviors in solving homeworks/projects?

1. Looking up information relevant to the course online.
2. Looking up or asking for sample solutions online.
3. Copying code from my classmates, and then editing it significantly.
4. Asking the course staff for help.
5. Sitting next to my classmate and coding together as a team or with significant conversation about approach.
6. Sharing my code with a classmate, if he/she just wants to read over it and learn from it

Part (b):

Which of the following are recommended ways of writing code?

1. gedit
2. emacs
3. Eclipse
4. sublime
5. Microsoft Visual Studio
6. notepad

Part(c):

What is the late submission policy?

1. Each assignment can be submitted up to two days late for 50% credit.
2. Each student has 3 late days of which only 1 can be used per HW.
3. Students need to get an approval before submitting an assignment late.

Part(d):

After making a late submission by pushing your code to Gihub you should...

1. Do nothing! Sit back and enjoy.
2. Complete the online late submission form.
3. Email your instructor.

Problem 2 (Git, 10%)

Carefully review and implement the steps discussed in Lab1. Then, answer the following questions:

Continue your answers to this question in the file name `hw1.txt`

Part (a):

Which of the following git user interfaces are accepted and supported in this course?

1. Git Bash (Windows)
2. GitHub Desktop Client
3. Terminal (Mac or Linux)
4. Eclipse eGit
5. Tower Git Client

Part (b):

Provide the appropriate git command to perform the following operations:

1. Stage an untracked file to be committed. The file is called 'hw1q2b.cpp'.
2. Display the details of the last three commits in the repository.

Part (c)

Let's say you staged three files to be committed. Then, you ran the following command:

`git commit`
What will git do?

Problem 3 (Review Material, and Programming Advice)

Carefully review recursion and dynamic memory management from your CSCI 103 notes and textbook. You may also find Chapters 2 and 5 from the textbook, Chapters 2 and 3 from the lecture notes, and the C++ Interlude 2, helpful.

You will lose points if you have memory leaks, so be sure to run `valgrind` once you think your code is working.

```
$ valgrind --tool=memcheck --leak-check=yes ./sum_pairs input.txt output.txt
```

Hint: In order to read parameters as command line arguments in C++, you need to use a slightly different syntax for your `main` function: `int main (int argc, char * argv[])`. Here, `argc` is the total number of arguments that the program was given, and `argv` is an array of strings, the parameters the program was passed. `argv[0]` is always the name of your program, and `argv[1]` is the first argument. The operating system will assign the values of `argc` and `argv`, and you can just access them inside your program.

Problem 4 (Recursion and Streams, 15%)

Remember that a string being a palindrome can be characterized as follows:

- The empty string is a palindrome.
- Any single character is a palindrome.
- If *c* is a character, and *p* is a palindrome, then the string *cpc* is also a palindrome.

Consider the program palindrome.cpp, which we have provided for you in the `homework_resources/hw1` folder/repo. Copy the file to your `hw_usc-username/hw1` folder. The program is intended to receive a string as input, and output "Palindrome" indicating the string is a palindrome, or "Not a Palindrome" indicating it isn't. However, the program currently outputs "Palindrome" for all strings. Copy, compile, and run the program with several different input to verify this.

Your job is to fix the code so that it gives the correct output for all possible inputs. Only make changes where the code indicates they should be made: you should not change the main function, nor the start of the helper function.

Problem 5 (Streams, 20%)

Write a program that reads in a text file and outputs the words of that file in reverse order, one per line. To achieve this, you will either want to use recursion or dynamic memory allocation (your choice). The program should receive the filename as a command line parameter.

For the text file, the first line will consist only of an integer representing the number of words in the file; let's call it `n`. This will be followed by `n` words. Each word will consist only of letters (uppercase or lowercase), and the words will be separated by some amount of whitespace (space, tab, newline).

The following is a sample input and output of the program.

- Input text file: hw1q5.txt

```
11
Hello World
saying this   like Yoda
But soylent green   are  people
```

- Compiling the program

```
> g++ -g hw1q5.cpp -o hw1q5
```

- Running the program and the output

```
> ./hw1q5 hw1q5.txt
people
are
green
```

```
soylent
But
Yoda
like
this
saying
World
Hello
```

Problem 6 (Strings and Dynamic Memory, 45%)

The new residential college Tommy Trojan Tower is about to be constructed, and the designers are running simulations to verify there will be enough space to cover the demand for student housing. You will be writing the infrastructure that the designers will use in their simulations. Here are the required specifications for your program:

- You will create an array, where each index in the array represents a floor of Tommy Trojan Tower (if there are `n` floors, they are labeled from `0` to `n-1`). You will not know how many floors there will be until runtime, so you must allocate this array dynamically.
- Each floor of Tommy Trojan Tower will contain another array, where each index represents a student living on this floor of Tommy Trojan Tower (if there are `n` students, they are labeled from `0` to `n-1`). Note that each floor may have a vastly different number of students, which you will only find out at runtime.
- Each student in Tommy Trojan Tower will contain an array of strings, indicating all of the possessions of that student.

You can store this information in a 3-dimensional array `string ***trojans`.

The program should receive two command line parameters. The first one will be the input file name, and the second will be the output file name. The first line of the input file will consist of a single integer, which is the number of floors in Tommy Trojan Tower. You may assume the first line is formatted correctly.

Each successive line in the file will be one of the following commands. If a line is not properly formatted, you should output `Error - incorrect command` to the output file and move on to the next line in the input file. You should execute the commands in order.

- `MOVEIN i k`: This command will indicate that `k` students are going to live on floor `i`, and should allocate the appropriate space. If there are already students living on floor `i`, you should write `Error - floor i is not empty` to the output file. If floor `i` does not exist, you should write `Error - floor i does not exist`.
- `MOVEOUT i`: This command will clear all students out from floor `i`, and deallocate the appropriate memory (all students and their possessions from that floor). If there are no students living on floor `i` or

if floor `i` does not exist, you should output an error.

- `OBTAIN i j k n1 n2 n3 ... nk`: This command sets the string sequence `n1 n2 ... nk` as the possessions of student `j`, living on floor `i`. If there is no student `j` on floor `i`, you should output an error. If the student already has possessions, you should output an error.
- `OUTPUT i j`: This command outputs the possessions of student `j` living on floor `i`, one per line, to the output file. If there is no student `j` on floor `i`, or if the student has no possessions, the user should get an error.

Once the last command has been executed, the program should correctly deallocate all memory and terminate.

In your solution, you cannot use `vector`, `deque`, or `list` data types from the STL. Instead you will be dynamically allocating and deallocating all of your memory via arrays of strings. The most important part of this problem is to make sure that your solution does not leak memory. You will lose significant points for memory leaks.

We have provided a skeleton for you in the `homework_resources/hw1` folder/repo. Copy the skeleton file `tommytrojan.cpp` to your `hw_usc-username/hw1` folder to get started.

Commit then Re-clone your Repository

Be sure to add, commit, and push your code in your `hw1` directory to your `hw_usc-username` repository. Now double-check what you've committed, by following the directions below (failure to do so may result in point deductions):

1. Go to your home directory: `$ cd ~`
2. Create a `verify` directory: `$ mkdir verify`
3. Go into that directory: `$ cd verify`
4. Clone your hw_username repo: `$ git clone git@github.com:usc-csci104-spring2016/hw_usc-username.git`
5. Go into your `hw1` folder `$ cd hw_username/hw1`
6. Recompile and rerun your programs and tests to ensure that what you submitted works.

-