CS 201 Homework 6

This assignment is focused on *Namespaces* and the *Standard Template Library (STL)*.

Note: If you encounter difficulties, please discuss on Discord #cs201.

Note: You will find Chapter 21 from Bjarne Stroustrup's book to be very useful.

Grading

- (Pass/Fail) Programs compile and run
- (Pass/Fail) GitHub Project
- (Pass/Fail) Overleaf Report written in LaTeX
- (Pass/Fail) Report submitted on Blackboard as a PDF (lastname.firstname-hwN.pdf)
- (Pass/Fail) Report contains link to GitHub repository
- (50 pts) Report contains *design* paragraph (~100 words)
- (50 pts) Report contains *post mortem* paragraph (~100 words)
- (50 pts) Report questions
- (50 pts) Report contains *sample run*
- (50 pts) Source Code has *Makefile* or Visual Studio Solution *.sln*
- (50 pts) Source Code is neat and documented
  - Name, Class, Date, and Description at top
  - All functions and classes have documentation (besides `main()`)
- (100 pts) 2 of the *additional programs* are completed
- (100 pts) GitHub frequent commits (ensure you use the program name in your text)

General Requirements

Programs that do not compile will not be graded; there is no point in turning in code that does not compile. For full credit, turn in only what was requested—no project files, executables, etc. As always:

- Code should compile, execute and work correctly.
- It should be clear to the user what the program is doing and what the user is expected to do.
- The source code should begin with comments indicating the source file's filename, author, date, and purpose.
- The source code should be neat and readable.
- Functions should be commented with their purpose.
- Other comments should be used anywhere things might not be completely clear.

Questions

- What is the difference between a container, an iterator, and an algorithm?

- What is namespace pollution?
- vector member function end returns an iterator. This iterator does not point to the last item in the vector. What does it point to?
- Give an example of a Standard Template Library algorithm: name it, and briefly explain what it does.
- What is an easy way to access a value in a map?
- How do we remove a key from a map?
- What does it mean for a sorting algorithm to be stable?
- What is a lambda function?
- In the context of Standard Template Library algorithms, lambda functions can be very useful. Explain.
- What happens if we seed a pseudorandom number generator with the same value (seed) each time a program is run? Why is this behavior sometimes desirable?

Main Program (random-map.cpp)

Write a C++ program that uses the `<random>` header to test both a uniform distribution of random numbers and a normal distribution of random numbers using a **std::map<int,int>** to simulate a histogram.

- Consult the documentation (https://en.cppreference.com/w/cpp/numeric/random) and use their example to start out with:

```
#include <iostream>
#include <iomanip>
#include <string>
#include <map>
#include <random>
#include <cmath>

int main()
{
    // Seed with a real random value, if available
    std::random_device r;

    // Choose a random mean between 1 and 6
    std::default_random_engine e1(r());
    std::uniform_int_distribution<int> uniform_dist(1, 6);
    int mean = uniform_dist(e1);
    std::cout << "Randomly-chosen mean: " << mean << '\n';

    // Generate a normal distribution around that mean
    std::seed_seq seed2{r(), r(), r(), r(), r(), r(), r(), r()};
    std::mt19937 e2(seed2);
    std::normal_distribution<> normal_dist(mean, 2);

    std::map<int, int> hist;
    for (int n = 0; n < 10000; ++n) {
        ++hist[std::round(normal_dist(e2))];
    }
    std::cout << "Normal distribution around " << mean << ":\n";
    for (auto p : hist) {
        std::cout << std::fixed << std::setprecision(1) << std::setw(2)
                  << p.first << ' ' << std::string(p.second/200, '*') << '\n';
    }
}
```

- Modify this program so that it includes the following new functions:
    - **int RandomBetweenU(int first, int last)** which returns a uniform random number between first and last, inclusively (e.g. 1 and 6 returns numbers between 1 and 6)
    - **int RandomBetweenN(int first, int last)** which returns a normally distributed random number between first and last, inclusively (e.g. 1 and 6 returns numbers between 1 and 6)
    - **int RandomBetween(int first, int last)** which returns numbers using the **rand()** function from the C standard library **<stdlib.h>**.
    - **void PrintDistribution(const std::map<int, int> &numbers)** which prints a list (similar to the code above) of the random numbers clearly showing they are normally or uniformly distributed.
- Compare all three random number generators and print their histograms.
- *Include sample output in your report.*

Additional Program 1 (namespace.cpp)

Design a C++ program that uses **namespace**s. Use a program you created in Homework 5 and put all the FLTK callbacks and data structures inside of their own namespace. Be sure to move these into a separate **.hpp** and **.cpp** file so that your main.cpp file only contains the main program. *Include a sample screenshot output in your project report.*

Additional Program 2 (hangman.cpp)

Design a C++ program that allows a user to play the game Hangman. You are free to choose the words that the program will use in the game. Make sure that they get ten chances to guess the correct word corresponding to head, body, right and left arms, right and left legs, right and left hands, and right and left feet. Use a C++ **map** to keep track of all the letters currently used. Use the C++ **<algorithms>** header to search for appropriate tools to implement your program. Do not use a while or for loop in your program (except to accept user input). Rather utilize the **STL** and **lambda functions** to implement your program. *Include sample output in your project report showing a game won and a game lost.*

Additional Program 3 (shopping.cpp)

Design a C++ program that represents a store front. Use a plain old data (POD) **struct** to represent the items in the store. At minimum it should include **unitPrice** and **units** members as follows:

```
struct Record {
    double unitPrice;
    int units;
};
```

Come up with a list of items (at least five) to sell to the user and store them in a **map<string, Record>** object. Allow the user to add or remove items to or from their shopping cart. Allow them to add more than one item in the cart. Give the user the option to see their current shopping cart. Calculate the total using the **accumulate()** algorithm. *Include sample output in your project report showing the user adding and removing items from the cart and a final cart showing at least three items, one of which has more than one unit.*

Additional Program 4 (math-catch.cpp)

Use Catch2.hpp (from https://github.com/catchorg/Catch2) to test the following STL functions:

- sin(x)
- atan2(x)
- accumulate(b,e,i)
- inner_product(b,e,b2,i)

Additional Program 5 (calculator.cpp)

Continue to develop the tokenizer program from the previous homework assignment. This time add the ability to convert the order of the tokens from infix notation to postfix notation. For example, **3 + 4 * 5** is infix notation because the operators are between the numbers. In postfix notation, the order is **3 4 5 + *** because the operators come after the numbers. We will push these in order on two stacks as we encounter them, one for numbers (accumulator stack) and one for operators (operator stack). Then when we pop the operators off the stack we will push the result back onto the accumulator stack. So, in the example given, we will pop the * and multiply 4 and 5, pushing 20 on the accumulator stack (now 3 20). Then we pop the + and add 3 and 20, pushing 23 on the accumulator stack. *Include sample output showing the calculation of several different expressions (at least 3).* Optionally, use a **map<string,double>** to support variable assignment so that “x = 3 + 5” will cause the map to store the pair {“x”, 8} and a later expression “y = 6 + x” will result in adding a second pair {“y”, 14}.