CS 201 Homework 06

Bryan Beus

November 18, 2019

Source Code Link: `https://github.com/siddhartha-crypto/cs201/tree/master/hw6`

1 Design

1.1 Histogram

With the example provided in the homework and in the CPP reference files, there is little left to design. I intend to keep the same structure, simply ensuring that all values stay within the bounds of 1 and 6, as instructed.

1.2 Hangman

For this additional program, I intend to keep the GUI elements simple. While I could spend time making ASCII art that visualizes the hangman's progress towards the grave, I intend to keep a simple running list of body parts captured. The sentence the user is guessing will be a simple quote that I pick, and will be static throughout the duration of the project.

1.3 Catch

I've used Ruby testing source in JavaScript a long time ago, and I would like to have the ability to write tests into my programs here. Therefore, I'll be experimenting with this test software. There are provided instructions in the tutorial section of the Github account, and I intend to simply follow them for the duration of the project.

2 Post Mortem

2.1 Histogram

The challenge I found in this project was to keep the normal distribution within the values of 1 and 6. I tried setting the mean value to `3.5`, but this produced undesirable results. Also, a normal distribution should only have a standard deviation. By design, the distribution should not be limited in its potential endpoints. I tried setting very small standard deviations, such as `0.8`, but occasionally I would still receive outlier values outside the expected range.

The only way, in the end, that I could ensure that any returned values were inclusively between 1 and 6 was to set the standard deviation to `1` and ignore any extreme outliers. I also set the mean to `4`, as this produced the most stable results.

2.2 Hangman

This was an enjoyable, yet challenging project. In all, I spent the most time here. Learning to use the STL and lambda functions was a worthwhile experience. I learned how to include variables from outside the scope of the lambda function, to use `auto` when declaring iterators, and how to access different elements of a pair within a lambda.

I still have questions about updating values via iterators when accessing the values via pointers. I will try to ask during class.

Otherwise, the rest of the assignment was fairly straightforward.

2.3 Catch

This turned out to be much easier to use than the work I've done previously with JavaScript. The default template provided in the tutorial showed how to use the basic elements of the test suite, and I inserted the four functions provided in the homework assignment with little trouble.

In the future, I would like to try splitting this into a different type of project tree. Testing is very useful, and I like to use it, whenever possible.

3 Answers to Questions

- A container contains raw elements, such as a collection of `int` values, or a collection of `char` values to together create a STL `string`. An iterator is a handy type of pointer that points to the values in a container, and this iterator has several built-in C++ functionality that make usage easier. An algorithm is a statement of logic that performs a computation. An algorithm can acts on an iterator and on a container.

- Namespace pollution is when values taken from multiple namespaces are identical in syntax, and therefore the compiler becomes logically confused about what the user intended by the conflated namespace values.

- The `vector.end()` iterator points to a space directly after the last value in the vector. This can also be called the `EOF` or end of file value.

- One STL algorithm is the `std::for_each()` method. This method iterates over each item in a container, starting at the provided starting point and moving to the end. The user can provide a function, such as a lambda function, that can perform an action according to any of the values in the container at each space.

- One easy way to access a value in a map is as follows: `myMap [myKeyValueHereByType]`. This would return the second value of the map at the provided key pair.
- Erasing an element pair from a map can be done as follows: `myMap.erase("valueToBeErased")`.
- A sorting algorithm is said to be stable if values that are of identical sorting weight remain in their position, once sorted. Unstable sorting algorithms will move around values of identical sorting weight within their local region, and this can be a waste of resources.
- Most simply put, a lamda function is a function that has no name.
- We can pass lambda functions into STL algorithms easily. This is a clean way to perform a series of computations on a container, without having to create a fully defined function that would otherwise be used only once.
- If we reuse the same seed for the randomization generator, we will receive the same sequence of random numbers. This can be useful if we are in a testing scenario and want to quickly and easily suspend randomization between separate trials.

4 Sample Output

4.1 Histogram

```
1 ********
2 ********
3 ********
4 ********
5 ********
6 ********

1 *
2 ******
3 ****************
```

```
4 *****************
5 ******
6 *

1 ********
2 ********
3 ********
4 ********
5 ********
6 ********
```

4.2 Hangman

```
---Losing Result---
Current Progress:
________ ______ ____ __ ____ _ ____ ______ ________ _ ____ ___
 ↪  ____ ________ _______ _____, _ _____ __ _____ ____ ___ ____
 ↪  ___, _______ _ __ _ _______ ______.

The hangedman's state of affairs:

head
body
right arm
left arm
right hand
left hand
right leg
left leg
right foot
left foot
Mistakes: 10/10

You lose. Try again!

---Winning Result---
Current Progress:
Whenever people tell me that I will regret whatever I just did
 ↪  when tomorrow morning comes, I sleep in until noon the next
 ↪  day, because I am a problem solver.


You win. Congrats!
```

4.3 Catch

```
===================================
All tests passed (4 assertions in 1 test case)
```

5 My Programs

5.1 Histogram

```cpp
/*
 * main.cpp
 * CS 201
 * November 15, 2019
 * Bryan Beus
 * Main file for the histogram project in hw6
 */

#include <iostream>
#include <iomanip>
#include <string>
#include <map>
#include <random>
#include <cmath>
#include <stdlib.h>

using std::cout;
using std::cin;
using std::endl;

// Return a random number according to a uniform distribution
int RandomBetweenU(int first, int last, std::mt19937 &e1);

// Return a random number according to a normal distribution
int RandomBetweenN(int first, int last, std::mt19937 &e1);

// Return a random number using the standard library rand()
 ↪  function
int RandomBetween(int first, int last);

// Print the provided distribution to the console
void PrintDistribution(const std::map<int, int> &numbers);

int main()
{
  // Declare endpoints of the range for the distributions
  int first = 1;
  int last = 6;

  // Declare pseudo-random device for creating seeds
  std::random_device r;

  // Create a seed sequence to feed to the generator
  std::seed_seq seedObj{r(), r(), r(), r(), r(), r(), r(), r()};

  // Declare random-number generator and provide with seedObj
   ↪  sequence
  std::mt19937 e1(seedObj);

  // Create uniform historgram
  std::map<int, int> uniformHistogram;
  for (int i = 0; i < 10000; i++) {
    ++uniformHistogram[std::round(RandomBetweenU(first, last,
     ↪  e1))];
  }

  // Print the uniform histogram
  PrintDistribution(uniformHistogram);

  // Create normal histogram
  std::map<int, int> normalHistogram;
  for (int i = 0; i < 10000; i++) {
    int val;
    // Technically, a randomized normal distribution should never
     ↪  have defined endpoints
    // However, because the assignment lists 1 and 6 as the
     ↪  required range,
    // Only accept values that fit within the range
    while (true) {
      val = std::round(RandomBetweenN(first, last, e1));
      if (val <= last && val >= first)
        break;
    }
    ++normalHistogram[val];
  }

  // Print the normal histogram
  PrintDistribution(normalHistogram);

  // Create standard rand() histogram
  std::map<int, int> standardHistogram;
  for (int i = 0; i < 10000; i++) {
    ++standardHistogram[std::round(RandomBetween(first, last))];
  }

  // Print the standard histogram
  PrintDistribution(standardHistogram);

  return 0;
}

```

```cpp
// Return a random value according to a uniform distribution
int RandomBetweenU(int first, int last, std::mt19937 &e1) {

  std::uniform_int_distribution<int> uniform_dist(first, last);
  int val = uniform_dist(e1);

  return val;

}

// Return a random value according to a normal distribution
int RandomBetweenN(int first, int last, std::mt19937 &e1) {

  std::normal_distribution<> normal_dist(4, 1), min(first),
   ↪  max(last);
  int val = normal_dist(e1);

  return val;

}

// Return a random value according to the default standard library
int RandomBetween(int first, int last) {
  // Acquire a random int value
  int val = (std::round(std::rand()));

  // Ensure the value is within six digits
  val = val % 6;

  // Add 1 to the value, to account for the 0 - 5 default range
  val++;

  // Return value
  return val;

}

// Print the provided distribution to the console
void PrintDistribution(const std::map<int, int> &numbers) {

  for (auto p: numbers) {
    cout << std::fixed << std::setprecision(1) << std::setw(2) <<
     ↪  p.first << ' ' << std::string(p.second/200, '*') << endl;
  }

  cout << endl;
}
```

5.2 Hangman

```cpp
/*
 * hangman_main.cpp
 * CS 201
 * November 15, 2019
 * Bryan Beus
 * Main file for hangman project in hw6
 */

#include <cstdlib>
#include <iostream>
#include <iomanip>
#include <algorithm>
#include <iterator>
#include <utility>
#include <vector>
#include <string>
#include <map>

#include "hangman.hpp"

using std::vector;
using std::pair;
using std::string;
using std::cout;
using std::cin;
using std::endl;
using std::make_pair;
using std::for_each;
using std::getline;
using std::istringstream;
using std::ostringstream;
using std::to_string;
using std::find;
using std::map;
using std::setw;
using std::left;

int main() {

  // Initialize bool value to test whether user is finished with the game
  bool isFinished = false;

  // Initialize vector of body parts to capture for the hanged man
  vector<string> hangmanItems {"head", "body", "right arm", "left arm", "right hand", "left hand", "right leg", "left leg", "right foot", "left foot"};
  vector<pair<string, bool>> hangmanState;

  // For each body part, add an assumed value of "false" to indicate that the body part is not yet captured
  for_each(hangmanItems.begin(), hangmanItems.end(), [&hangmanState](string item) {
    hangmanState.push_back(make_pair(item, false));
  });

  // Initialize the quote for the game
  string quote = "Whenever people tell me that I will regret whatever I just did when tomorrow morning comes, I sleep in until noon the next day, because I am a problem solver.";

  // Initialize a vector to hold each element of the sentence to be guess
  vector<pair<char, bool>> quoteStatus;

  // For each element, pair it with a value of false to indicate that it should remain hidden
  for_each(quote.begin(), quote.end(), [&quoteStatus](char n) {
    bool statusBool = false;

    // Values that are not part of the guessing game, including periods, whitespaces, and commas, do not need to be hidden
    if (n == '.' || n == ' ' || n == ',') {
      statusBool = true;
    }
    quoteStatus.push_back(make_pair(n, statusBool));
  });

  // Initialize the number of turns (or mistakes) the user has made
  int turns = 0;

  // Initialize the variable to test whether the user has found a new variable
  // Assume true for now
  bool charFound = true;

  // Initiate while loop to continue game so long as user has neither won nor lost
  // Prompt user input for each turn of the game
  while (!isFinished) {
    isFinished = promptUserInput(quoteStatus, hangmanState, isFinished, turns, charFound);
  }

  return 0;
}
```

5.3 Catch

```cpp
/*
 * main.cpp
 * CS 201
 * November 15, 2019
 * Bryan Beus
 * Main file for catch project in hw6
 */

#include <iostream>
#include <vector>
#include <functional>
#include <iomanip>
#include <string>
#include <cmath>
#include <math.h>
#include <stdio.h>
#include <numeric>

// Make M_PI available for calls to the value of pi
#define _USING_MATH_DEFINES

// Initialize and include Catch test suite
#define CATCH_CONFIG_MAIN
#include "catch.hpp"

using std::cout;
using std::cin;
using std::endl;
using std::sin;
using std::atan2;
using std::accumulate;
using std::inner_product;
using std::vector;

// Declare primary TEST_CASE
TEST_CASE( "Common math functions perform", "[math]" ) {

  // Initialize intended value of sin
  double sinVal = sin(M_PI);

  // Initialize intended value of call to atan2()
  double x, y, atan2Val;
  atan2Val = atan2 (x, y) * 180 / M_PI;

  // Initialize intended value of call to accumulate()
  int init = 100;
  vector<int> accPreVals{10, 20, 30, 40};
  int accVal = accumulate(accPreVals.begin(), accPreVals.end(),
   ↪  init);


  // Initialize intended value of call to inner_product()
  vector<int> a {0, 1, 2, 3, 4};
  vector<int> b {5, 6, 7, 8, 9};
  int innProVal = inner_product(a.begin(), a.end(), b.begin(),
   ↪  0);


  // Use Catch test suite to test the expected values against the
   ↪  values in practice
  REQUIRE( sin(M_PI) == sinVal );
  REQUIRE( atan2 (x, y) * 180 / M_PI == atan2Val );
  REQUIRE( accumulate(accPreVals.begin(), accPreVals.end(), init)
   ↪  == accVal );
  REQUIRE( inner_product(a.begin(), a.end(), b.begin(), 0) ==
   ↪  innProVal );
}
```