Jordan Lian

Engineering Student

Jordan Lian
10 Forsyth Street
Boston, MA 02115
610.425.8865
lian.jo@husky.neu.edu

20 NOVEMBER 2017

Richard Whalen
Professor, Northeastern University
360 Huntington Avenue
Boston, MA 02115

Dear Professor Whalen,

This recent coding assignment went really well. Firstly, I had to create a program that would "roll the dice" twice, take the sum, and then record the percentage of the number of each sum (2-12). For my program I declared variables for each roll (1 and 2) the sum of the two rolls, variables for the percentages asked for at the end, and variables for the range (r1 and r2, 1 and 6 for dice). Then I asked the user to enter the range (1 to 6) and the number of rolls before applying the srand() and rand() functions. Then I used a switch (case) logic statement to add 1 for each sum number (2-12, n2++ when the sum of the rolls was 2). Then I stated the results (each toss (2 rolls) and its sum) and the percentages of each sum. For 1000 tosses, I got 2.6% of the tosses had a sum of 2, 5.1% had a sum of 3, 9.1% had a sum of 4, 12.3% had a sum of 5, 13.8% had a sum of 6, 17% had a sum of 7, 12% had a sum of 8, 9.3% had a sum of 9, 9.6% had a sum of 10, 7% had a sum of 11, and 2.2% had a sum of 12. Since 1000 tosses is a lot of tosses, I simply commented out the output results of each toss (with each roll number and the total sum) for the sake of space. Otherwise, the results could all be printed out.

For the second part of the assignment, I first wrote all the subfunctions above the main function with a semicolon. Then below the main function, I wrote those same sub-functions below with no semicolon, and wrote the formulas in a loop for each separate sub-function. The three functions I had to test were the buckling load, compressive stress, and the slenderness limit functions. The maximum load had to exceed the entered load based on the width (starts at 2 inches, increases by increments of 2 inches), and the entered length for the buckling load and compressive stress functions. For the slenderness limit function, the length to width ratio had to be less than or equal to 50. I returned those values to the main functions. If those three conditions were met in the main function, then the test passes. If not, then the test fails. I used an if statement that said that the test would fail if any of the three conditions were not met (I used or syntax) followed by a break. Then I put an else statement after that saying that else the test passed (if all three conditions were met). To wrap that, I used a for loop to increase the width by 2 inches if the test failed up until the width was as great as the entered length. After I just used ofstream to outfile the results in a text document. For an 18,000 pound load and a 72 inch width, the program calculated that an 8 inch width would be best for the square timber column. For a 2 inch, 4 inch, and 6 inch width, the tests failed, but the 8 inch width worked, and the program explicitly stated that.

Below is the pseudocode, source code, and output file results for the 2 separate problems. Thank you for reading.

Sincerely,

Jordan Lian

Program 1: Roll the Dice -- Winning at Craps

Pseudocode:

//library files, include cstdlib for srand() and rand()

//declare variables, roll1, roll2, sum, percentage variables, r1 and r2 for range (1 to 6)

// enter range and number of rolls

//apply srand() and rand()

//show outputs, and sum of each output, (1 roll = 2 tosses > 2 sums)

//use switch, case logic statement to add one to each number (n2++, n3++) when the sum corresponds with whatever number

//use percentage variables accordingly - n2 / n_rolls (total rolls)

//end main

//outfile the results to a .txt document using ofstream

Source code:

//This program "rolls the dice" twice, takes the sum of the two numbers generated, and records the percent of sums of 2-12 amongst the total number of rolls

//declare library files

```
#include <iostream> //input output

#include <cstdlib> //cstandard library - srand() rand()

#include <ctime>

#include <fstream>


using namespace std; //cin cout
```

```
int main(void)

{

//declare variables
        int n_rolls; //user input for number of rolls
        int i, roll_1, roll_2, sum, r1, r2;
        int n2 = 0, n3 = 0, n4 = 0, n5 = 0, n6 = 0, n7 = 0, n8 = 0, n9 = 0, n10 = 0, n11 = 0, n12 = 0;
        double perc_2s, perc_3s, perc_4s, perc_5s, perc_6s, perc_7s, perc_8s, perc_9s, perc_10s, perc_11s, perc_12s;

        ofstream outfile;
        outfile.open("roll_dice.txt");

        //set the seed value
        srand(time(NULL));

        cout << "Please enter the lower bound of the range for each roll (1 please because we're dealing with dice)
--> ";
        cin >> r1;

        cout << "Please enter the upper bound of the range for each roll (6 please because we're dealing with dice)
--> ";
        cin >> r2;

        cout << "Please enter the number of rolls --> ";
        cin >> n_rolls; //get user input

                //for loop for number of tosses
        for(i=0 ; i < n_rolls; i++)
        {
```

```
        roll_1 = rand() %r2 + r1; //do mod 6 + 1 for the dice program
        roll_2 = rand() %r2 + r1;
        sum = roll_1 + roll_2;

        cout << "\nthe number is " << roll_1; //output toss value
        outfile << "\nthe number is " << roll_1;
        cout << "\nthe number is " << roll_2;
        outfile << "\nthe number is " << roll_2;
        cout << "\nthe sum of the two numbers is " << sum;
        outfile << "\nthe sum of the two numbers is " << sum;
        cout << "\n----------";
        outfile << "\n----------";

switch (sum)
        {
                case 2:
                        n2++;
                        break;
                case 3:
                        n3++;
                        break;
                case 4:
                        n4++;
                        break;
                case 5:
                        n5++;
                        break;
                case 6:
                        n6++;
                        break;
```

```
                case 7:
                        n7++;
                        break;
                case 8:
                        n8++;
                        break;
                case 9:
                        n9++;
                        break;
                case 10:
                        n10++;
                        break;
                case 11:
                        n11++;
                        break;
                case 12:
                        n12++;
                        break;
        } // close switch

} //end for loop

perc_2s = (double) 1.0 * n2 / n_rolls * 100; //cast for integer division
perc_3s = (double) 1.0 * n3 / n_rolls * 100;
perc_4s = (double) 1.0 * n4 / n_rolls * 100;
perc_5s = (double) 1.0 * n5 / n_rolls * 100;
perc_6s = (double) 1.0 * n6 / n_rolls * 100;
perc_7s = (double) 1.0 * n7 / n_rolls * 100;
perc_8s = (double) 1.0 * n8 / n_rolls * 100;
perc_9s = (double) 1.0 * n9 / n_rolls * 100;
```

```
        perc_10s = (double) 1.0 * n10 / n_rolls * 100;
        perc_11s = (double) 1.0 * n11 / n_rolls * 100;
        perc_12s = (double) 1.0 * n12 / n_rolls * 100;



        cout << "\n\nPercent of 2s = " << perc_2s << "%";
        outfile << "\n\nPercent of 2s = " << perc_2s << "%";
        cout << "\nPercent of 3s = " << perc_3s << "%";
        outfile << "\nPercent of 3s = " << perc_3s << "%";
        cout << "\nPercent of 4s = " << perc_4s << "%";
        outfile << "\nPercent of 4s = " << perc_4s << "%";
        cout << "\nPercent of 5s = " << perc_5s << "%";
        outfile << "\nPercent of 5s = " << perc_5s << "%";
        cout << "\nPercent of 6s = " << perc_6s << "%";
        outfile << "\nPercent of 6s = " << perc_6s << "%";
        cout << "\nPercent of 7s = " << perc_7s << "%";
        outfile << "\nPercent of 7s = " << perc_7s << "%";
        cout << "\nPercent of 8s = " << perc_8s << "%";
        outfile << "\nPercent of 8s = " << perc_8s << "%";
        cout << "\nPercent of 9s = " << perc_9s << "%";
        outfile << "\nPercent of 9s = " << perc_9s << "%";
        cout << "\nPercent of 10s = " << perc_10s << "%";
        outfile << "\nPercent of 10s = " << perc_10s << "%";
        cout << "\nPercent of 11s = " << perc_11s << "%";
        outfile << "\nPercent of 11s = " << perc_11s << "%";
        cout << "\nPercent of 12s = " << perc_12s << "%\n\n";
        outfile << "\nPercent of 12s = " << perc_12s << "%\n\n";

system ("pause"); //for dev only to hold dos window
        return 0;  //assign value zero to main
```

} //end of main

Outfile: roll_dice.txt - for 1000 rolls

Percent of 2s = 2.6%

Percent of 3s = 5.1%

Percent of 4s = 9.1%

Percent of 5s = 12.3%

Percent of 6s = 13.8%

Percent of 7s = 17%

Percent of 8s = 12%

Percent of 9s = 9.3%

Percent of 10s = 9.6%

Percent of 11s = 7%

Percent of 12s = 2.2%

Program 2: Hoping it does not collapse

Pseudocode:

//write all the subfunctions above main with a semicolon

//write int main (void)

//write sub-functions below with no semicolon, and write the corresponding formulas in those sub-function loops

//return those values to the main function

//use a for loop in the main function to test different width values, increments of 2 inches, i = 2, i = i+2

//put an if statement in the for loop, saying that if any one of the 3 conditions aren't met (sub-function constraints, use "or" syntax)), then the test fails

//print out the results of the failed and the passed tests, plus the recommended width

//else the test passes (else means all of the conditions are met)

//end for loop, then end main

//outfile the results to a .txt document using ofstream

Source Code:

/* This program tests three functions to see if a square timber column can be designed. The constraints of those functions must be met. The user will enter a load that must be exceeded (in those functions) and a length will be applied in the three functions. The recommended width will determined based on the results. */

```
//library files
#include <iostream>
#include <cmath>
#include <fstream> //for outfile

using namespace std;

//constants
#define E 1700000 // modulus of elasticity (lb/in^2)
#define max_com_str 445 //maximum comopressive strength for a Douglas Fir tree (lb/in^2)

//repeat of all function prototypes ending with a semicolon!
	double buck_load (double max_load_BL, double column_length, double i); //buckling load
	double comp_stress (double max_load_CS, double i); //compressive stress
	double slend_limit (double SL, double column_length, double i); // slenderness limit

int main (void)

{
	//declare variables
	double column_load;
	double column_length;
	double max_load_BL;
	double max_load_CS;
	double SL;

	ofstream outfile;
	outfile.open("structure_test.txt");

	cout << "Please enter the expected load on the column in pounds --> ";
```

```
	cin >> column_load;


	cout << "\nPlease enter the length of the column in inches --> ";
	cin >> column_length;


	cout << "\n";


	for (int i = 2; i <= column_length; i = i+2) //for loop to add 2 inches to the width to test the function outputs
	{
		if (buck_load(max_load_BL, column_length, i) < column_load //if any of the conditions aren't met, use "or" statements
			|| comp_stress(max_load_CS, i) < column_load
			|| slend_limit (SL, column_length, i) >= 50) {
			cout << "Testing a beam with Area of " << i << " by " << i << " inches --> Test Failed\n";
			outfile << "Testing a beam with Area of " << i << " by " << i << " inches --> Test Failed\n";


		}
		else { //all of the conditions are met
			cout << "Testing a beam with Area of " << i << " by " << i << " inches --> Test Passed\n";
			outfile << "Testing a beam with Area of " << i << " by " << i << " inches --> Test Passed\n";
			cout << "\nFor a load of " << column_load << " pounds and a length of " << column_length << " inches, the recommended square beam has sides of " << i << " inches\n\n";
			outfile << "\nFor a load of " << column_load << " pounds and a length of " << column_length << " inches, the recommended square beam has sides of " << i << " inches\n\n";
			break;
		}
}
	system ("pause");
	return 0;
} //end of main
```

```
//list all subfunctions below main ...
        double buck_load (double max_load_BL, double column_length, double i) //buckling load
{
        //local variables
        max_load_BL = (0.3 * E * i * i) / pow(column_length / i, 2);
        return max_load_BL;
} // end of subfunction buck_load

double comp_stress (double max_load_CS, double i) //compressive stress
{
        max_load_CS = i * i * max_com_str;
        return max_load_CS;
} // end of subfunction comp_stress

double slend_limit (double SL, double column_length, double i) // slenderness limit
{
        SL = column_length / i;
        return SL;
} // end of subfunction slenderness limit
```

Output File: structure_test.txt

Testing a beam with Area of 2 by 2 inches --> Test Failed

Testing a beam with Area of 4 by 4 inches --> Test Failed

Testing a beam with Area of 6 by 6 inches --> Test Failed

Testing a beam with Area of 8 by 8 inches --> Test Passed

For a load of 18000 pounds and a length of 72 inches, the recommended square beam has sides of 8 inches