**CSCI 1730 - Programming Assignment 2 - 100 pts.**

**Due Date: Feb 27, 2019**

1. Write a C++ program that can be used to calculate and display the area and the perimeter of three different two-dimensional geometric shapes: a circle, a rectangle, and a triangle.

Your program should be modular and must contain the following functions:

* Function main will be the program driver – it will repeatedly display a menu of shapes to the user asking the user to select one and then call the appropriate functions to prompt the user for input of dimension(s), read the dimension(s), calculate the area and perimeter, and display the results. The repetition should continue until the user selects to exit the program.
* Three overloaded getDim functions that will ask the user to enter the needed shape dimension(s) from the user and return the dimension(s) to main.
* Three overloaded area functions that will take in the shape dimension(s) and calculate and return the shape area.
* Three overloaded perimeter functions that will take in the shape dimension(s) and calculate and return the shape perimeter.
* One display function that will take a shape indicator value and the shape area and perimeter values and display the labeled results.

**Note:** Your program should use Heron’s formula (given below) for finding the area of a triangle (the formula uses the triangle’s side lengths instead of the base and the height to find the area).

**Note:** Your program should also include a global named constant PI initialized to 3.141592654 that will be needed by your circle area and perimeter functions.

Area and Perimeter Formulas:

Circle:   where *r* is the circle radius.

Rectangle:   where *l* and *w* are the rectangle length and width.

Triangle:  

where *a*, *b*, and *c* are the lengths of the sides of the triangle.

Here is the output from a sample run of the program (user input in **bold**):

AREA/PERIMETER CALCULATOR

Select a shape:

Circle (1) Rectangle (2) Triangle (3) Exit (4)

Enter selection => **1**

Enter circle radius: **10**

Circle area = 314.159 and perimeter = 62.8319

AREA/PERIMETER CALCULATOR

Select a shape:

Circle (1) Rectangle (2) Triangle (3) Exit (4)

Enter selection => **2**

Enter rectangle length: **3.5**

Enter rectangle width: **4.5**

Rectangle area = 15.75 and perimeter = 16

AREA/PERIMETER CALCULATOR

Select a shape:

Circle (1) Rectangle (2) Triangle (3) Exit (4)

Enter selection => **3**

Enter 1st side of triangle: **3**

Enter 2nd side of triangle: **4**

Enter 3rd side of triangle: **5**

Triangle area = 6 and perimeter = 12

AREA/PERIMETER CALCULATOR

Select a shape:

Circle (1) Rectangle (2) Triangle (3) Exit (4)

Enter selection => **4**

Bye...

2. Write a C++ program that lets the user play the game of Rock, Paper, Scissors against the computer. The program should work as follows.

1. The program first determines the computer’s play in the game, as follows: A random number in the range of 1 through 3 is generated. If the number is 1, then the computer has chosen “rock”. If the number is 2, then the computer has chosen “paper”. If the number is 3, then the computer has chosen “scissors”.
2. The program then prompts the user to enter her “rock”, “paper”, or “scissors” choice and then reads it, using the same number scheme as above.
3. The program then displays both choices (in words) and the result of the play. A winner is selected according to the following game rules:
   * If one player chooses rock and the other player chooses scissors, then rock wins. (The rock smashes the scissors.)
   * If one player chooses scissors and the other player chooses paper, then scissors wins. (Scissors cuts paper.)
   * If one player chooses paper and the other player chooses rock, then paper wins. (Paper covers rock.)
   * If both players make the same choice, that play is a tie.

The program should repeatedly ask the user if she wants to play again and continues play (i.e., repeats steps 1-3, above) until she selects to end the game. After the user decides to end play, the program should display the overall results of the game – that is, how many times the user won, how many times the computer won, and how many ties there were.

Your program should be modular and must contain the following functions:

* Function main will be the program driver – it will repeatedly call the following functions to get the user’s play selection, randomly generate the computer’s play selection, determine the winner of a play, and tally the results. The repetition should continue until the user selects to exit the program, at which time it will call a function to display the final results.
* getCompSelection – this function will use the rand function to determine the computer’s play selection and return it.
* getUserSelection – this function will prompt for and read the user’s play selection and return it.
* playResults – this function will take the computer’s and user’s play selections, as well as counters for determining the total play results, and will determine the play result and display it, and will also increment the counters appropriately.
* finalResults – this function will take the total play results counters and generate the final display.

**Extra code needed:**

1. You need to add these to your #includes list:

#include <cstdlib>

#include <ctime>

1. You need to include this call to the srand function after your variable declarations in main in order to “seed” the rand function:

srand(time(NULL));

Here is the output from a sample run of the program (user input in **bold**):

It's time to play ROCK, PAPER, SCISSORS!

What is your play?

Rock (1), Paper (2), Scissors (3): **1**

You played rock and I played scissors

You win.

Play again? (y or n): **y**

What is your play?

Rock (1), Paper (2), Scissors (3): **3**

You played scissors and I played scissors

The play was a tie.

Play again? (y or n): **y**

What is your play?

Rock (1), Paper (2), Scissors (3): **2**

You played paper and I played scissors

I win.

Play again? (y or n): **y**

What is your play?

Rock (1), Paper (2), Scissors (3): **2**

You played paper and I played rock

You win.

Play again? (y or n): **n**

Final results of our match:

You won 2 time.

I won 1 time.

There were 1 ties.

Bye, bye...

3. The Fibonacci sequence is 0, 1, 1, 2, 3, 5, 8, 13, 21, … where the first two terms are 0 and 1, and each term thereafter is the sum of the two preceding terms. Write a C++ program that repeatedly prompts for and reads a positive value *n*, calls a **recursive** function fibonnaci to calculate the *n*th number in the Fibonacci sequence, and then displays the number. For example, if *n* = 9, then the program would display 21.

Here is the output from a sample run of the program (user input in **bold**):

Fibonnaci number generator

Which one do you want (negative to exit)? **4**

The 4th Fibbonaci number is 2

Which one do you want (negative to exit)? **5**

The 5th Fibbonaci number is 3

Which one do you want (negative to exit)? **6**

The 6th Fibbonaci number is 5

Which one do you want (negative to exit)? **7**

The 7th Fibbonaci number is 8

Which one do you want (negative to exit)? **-1**

4. When tossing one die, results of 1 through 6 are possible results. So, when tossing a pair of dice, sums of 2 through 12 are possible results. If the dice are fair, it can be shown that the probability of tossing each of these sums is:

2 or 12: ; 3 or 11: ; 4 or 10: ;

5 or 9: ; 6 or 8:; 7: 

Write a **C++ program** that will repeatedly simulate tossing a fair pair of dice a number of times specified by the user (any number between 1 and 100,000), and calculate and display the total number of each sum produced along with its percentage of the total tosses (the relative frequency approximation of probability).

Your program should be modular and must contain these functions:

* rollDie – this function will receive an array and the number of tosses and will simulate tossing one die the specified number of times, storing the toss results in the array.
* findSum – this function will receive the number of tosses, two arrays holding the toss results for two dice, and a third array, and will calculate the toss sums, storing these in the third array.
* tossCount – this function will receive the number of tosses, an array of toss sums, and an array of counters for the possible sums, and will examine the toss sums array and count the toss sum totals.
* display – this function will receive the number of tosses and an array of counters for the possible sums, and display the total number of tosses, and the total number of each of the possible sums, along with the probability of each possible sum.

Your function main should repeatedly obtain a number of tosses from the user and then call the other functions to handle the simulations, until the user elects to exit the program.

Notes and suggestions:

* In rollDie, use the rand function to simulate each die toss. Remember to include an appropriate call to srand in main.
* Functions tossCount and display both receive “an array of counters for the possible sums”. As noted above, the toss sums will be values 2 through 12. So, use an integer array of size 13, with all elements initialized to 0, for the toss sums counter array. Then, to count a sum of value *k* (*k* between 2 and 12), increment element *k* of the toss sums counter array.
* In display, the probability of a given sum is the percentage of tosses that produced that sum.
* In display, use the tab escape sequence \t to produce evenly spaced columns of numbers.

Here is output from a sample run of the program (user input in **bold**):

Enter number of tosses : **5000**

Total number of tosses = 5000

Toss Count Probability

2 131 2.62

3 270 5.4

4 427 8.54

5 549 10.98

6 708 14.16

7 826 16.52

8 692 13.84

9 575 11.5

10 422 8.44

11 264 5.28

12 136 2.72

Do another simulation? (y or n): **y**

Enter number of tosses: **25000**

Total number of tosses = 25000

Toss Count Probability

2 707 2.828

3 1411 5.644

4 2041 8.164

5 2760 11.04

6 3497 13.988

7 4122 16.488

8 3545 14.18

9 2861 11.444

10 2025 8.1

11 1371 5.484

12 660 2.64

Do another simulation? (y or n): **n**

5. Create a Point **structure** for modeling points in the *xy*-plane. Then, write a menu-driven **C++ program** that uses variables of type Point to perform a variety of tasks involving points in the *xy* plane.

Your program should be modular and must contain these functions:

* dist – this function will receive two Points and calculate and return the distance between the Points.
* slope – this function will receive two Points and calculate and return the slope of the line connecting two Points, if it exists. This function should have a bool reference parameter that will return a value to the user indicating whether the slope is defined or not.
* midPoint – this function will receive two Points and calculate and return the midpoint of the line segment between two Points.
* equation – this function will receive two Points and display the equation of the line passing through two Points.
* collinear – this function will receive three Points and determine if three Points are collinear. If the Points are collinear, this function should return true; otherwise, it should return false.
* readPt – this function will read a single point entered by the user in standard point “format” (for example “(4,3)”) into a Point variable and then return the Point.
* showPt – this function will receive a Point and then display the Point in standard point “format” (for example “(4,3)”).

Your function main should repeatedly display a menu of the above point manipulation tasks, read the user’s task selection, and then call the appropriate function(s) from above to perform the selected task and display the result, until the user selects to exit the program.

Notes and suggestions:

* When the slope is not defined, function slope can return any value for the slope – the bool parameter will be set appropriately so that the calling function will know not to use the returned value.
* In function collinear, use function slope to determine its result. When checking for equality of two slopes, do not use the “==” operator on the float values - instead, determine if the absolute value of the float values are smaller than a threshold value (something small, like 0.000001).

Algebra Topics Review:

* Slope of the line between (*a*, *b*) and (*c*, *d*),  provided ; slope undefined for vertical lines.
* Distance between (*a*, *b*) and (*c*, *d*): 
* Midpoint between (*a*, *b*) and (*c*, *d*): 
* The equation of a non-vertical line passing throughand is given by  where *m* is the slope of the line.
* The equation of a vertical line passing through and is given by  where the slope of the line is undefined.

Here is output from a sample run of the program (user input in **bold**):

POINTLAND

What do you want to do?

1 - Find distance between two points

2 - Find slope

3 - Find a midpoint

4 - Find an equation of a line

5 - Determine if three points are collinear

6 - Exit

Selection => **1**

Enter point 1: **(1,1)**

Enter point 2: **(4,5)**

Distance = 5

POINTLAND

What do you want to do?

1 - Find distance between two points

2 - Find slope

3 - Find a midpoint

4 - Find an equation of a line

5 - Determine if three points are collinear

6 - Exit

Selection => **2**

Enter point 1: **(2,3)**

Enter point 2: **(3,5)**

Slope = 2

POINTLAND

What do you want to do?

1 - Find distance between two points

2 - Find slope

3 - Find a midpoint

4 - Find an equation of a line

5 - Determine if three points are collinear

6 - Exit

Selection => **3**

Enter point 1: **(1,2)**

Enter point 2: **(3,4)**

MidPoint = (2,3)

POINTLAND

What do you want to do?

1 - Find distance between two points

2 - Find slope

3 - Find a midpoint

4 - Find an equation of a line

5 - Determine if three points are collinear

6 - Exit

Selection => **4**

Enter point 1: **(0,2)**

Enter point 2: **(1,5)**

Equation: y = 3x+2

POINTLAND

What do you want to do?

1 - Find distance between two points

2 - Find slope

3 - Find a midpoint

4 - Find an equation of a line

5 - Determine if three points are collinear

6 - Exit

Selection => **5**

Enter point 1: **(0,0)**

Enter point 2: **(1,1)**

Enter point 3: **(2,3)**

Points are not collinear

POINTLAND

What do you want to do?

1 - Find distance between two points

2 - Find slope

3 - Find a midpoint

4 - Find an equation of a line

5 - Determine if three points are collinear

6 - Exit

Selection => **5**

Enter point 1: **(0,2)**

Enter point 2: **(3,4)**

Enter point 3: **(6,6)**

Points are collinear

POINTLAND

What do you want to do?

1 - Find distance between two points

2 - Find slope

3 - Find a midpoint

4 - Find an equation of a line

5 - Determine if three points are collinear

6 - Exit

Selection => **6**

6. Create a **class** Elevator that will simulate the operation of an elevator. Here are the details:

* Assume that for a given elevator, it must keep track of its current floor position.
* Assume that an elevator will initially start on the first floor.
* An elevator should be able to service a request to move to a specified floor. In servicing a request, the elevator should display a summary of its floor movement.
* An elevator should also be able to return its current floor position to other program modules.

Using your Elevator class, write a **C++ program** that will use the Elevator class to simulate the operation of three elevators traveling between the 1st and 10th floors of a building, from the perspective of a person waiting for an elevator on the first floor. Use an array to model the three elevators. The program should repeatedly do the following tasks:

* Display the current positions of each of the three elevators.
* Ask the user to choose one of the elevators to use (i.e., like pushing the “up” button for one of the elevators – we assume here that each elevator has its own “up” button).
* If the chosen elevator is not on the first floor, the driver should give the elevator a request to come to the first floor.
* Ask the user which floor she wants (i.e., like pushing the “floor number” button inside the elevator).
* Send the chosen elevator a request to move to the selected floor.

Here is output from a sample run of the program (user input in **bold**):

Elevator Status

A B C

1 1 1

Which elevator do you want (1=A, 2=B, 3=C, or other to exit)? **1**

Which floor do you want? **3**

Starting at floor 1

Going up - now at floor 2

Going up - now at floor 3

Stopping at floor 3

Elevator Status

A B C

3 1 1

Which elevator do you want (1=A, 2=B, 3=C, or other to exit)? **3**

Which floor do you want? **5**

Starting at floor 1

Going up - now at floor 2

Going up - now at floor 3

Going up - now at floor 4

Going up - now at floor 5

Stopping at floor 5

Elevator Status

A B C

3 1 5

Which elevator do you want (1=A, 2=B, 3=C, or other to exit)? **2**

Which floor do you want? **4**

Starting at floor 1

Going up - now at floor 2

Going up - now at floor 3

Going up - now at floor 4

Stopping at floor 4

Elevator Status

A B C

3 4 5

Which elevator do you want (1=A, 2=B, 3=C, or other to exit)? **3**

Starting at floor 5

Going down - now at floor 4

Going down - now at floor 3

Going down - now at floor 2

Going down - now at floor 1

Stopping at floor 1

Which floor do you want? **6**

Starting at floor 1

Going up - now at floor 2

Going up - now at floor 3

Going up - now at floor 4

Going up - now at floor 5

Going up - now at floor 6

Stopping at floor 6

Elevator Status

A B C

3 4 6

Which elevator do you want (1=A, 2=B, 3=C, or other to exit)? **9**

**What you need to turn in:**

1. **Source code listing:** A printed copy of the source code for each problem. Remember to include the name of each group member in a comment at the top of your source code. Be sure to follow the “Code Style Guidelines” specified in class.
2. **Source code files:** E-mail me your source code as attachments.
3. **Working in Groups:** Every student will work with two other students in our class on this assignment; all members of the group must contribute to the solution. **Turn in only one copy of the solution** – **clearly identify the name of each member on everything that you turn in.**
4. **Late Assignments:** Assignments are due before class on the specified due date (both the paper copies and the e-mail copies). If you wish to turn in the paper copy of an assignment after class, place them under my office door. Assignments turned in late will be assessed a 20% penalty per class day late.