CSCI 1730 - Programming Assignment 3 - 50 pts.
Due Date: Thursday, March 19, 2015

What you need to turn in:

- **Code listing:** A printed copy of your C++ code solution for each problem. Remember to include your name on every page you turn in. Be sure to follow the "Code Style Guidelines" specified in the "Assignment Information and Guidelines" handout that was distributed on the first day of class (these guidelines are also available at the class web page).
- **Code files:** E-mail me a copy of your C++ code. Copy/paste all code into one e-mail message – please, do not send attachments. **Enter your name in the subject line of your email.**
- **Working in Pairs:** If you want to work with one other student in our class on this assignment, this is acceptable provided that both members of the pair make a contribution to the solution. If you decide to work in a pair, turn in only one copy of the solution – clearly identify the name of each pair member on everything that you turn in.
- **Late Assignments:** Assignments are due **by the end of class** on the specified due date (both the paper copies and the e-mail copies). Assignments turned in late will be assessed a 20% penalty per day late.

1. (10 pts.) When tossing one die, results of 1 through 6 are possible. So, when tossing a pair of dice, sums of the individual die results of 2 through 12 are possible. If the dice are fair, it can be shown that the probability of tossing each of these sums is:

$$\text{2 or 12: } \frac{1}{36} \approx 2.8\% \text{ ; 3 or 11: } \frac{1}{18} \approx 5.6\% \text{ ; 4 or 10: } \frac{1}{12} \approx 8.3\% \text{ ;}$$

$$\text{5 or 9: } \frac{1}{9} \approx 11.1\% \text{ ; 6 or 8: } \frac{5}{36} \approx 13.9\% \text{ ; 7: } \frac{1}{6} \approx 16.7\%$$

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).

Program requirements:

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
```

2. (15 pts.) Create a `point` structure that can be used to represent a point 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.

Program requirements:
Your program should be modular and must contain these functions:

- `dist` – this function will receive two points (i.e., two `point` variables) and calculate and return the distance between the points. **Note:** The function should **return** the distance (a `float` value), **not display** the distance.
- `slope` – this function will receive two points (i.e., two `point` variables) and calculate and return the slope of the line connecting two points, if it exists. **Note:** The function should **return** the slope (a `float` value), **not display** the slope. This function should also 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 (i.e., two `point` variables) and calculate and return the midpoint (i.e., a `point` value) of the line segment between two points. **Note:** The function should **return** the midpoint (a `point` value), **not display** the midpoint.
- `equation` – this function will receive two points (i.e., two `point` variables) and display the equation of the line passing through two points.
- `collinear` – this function will receive three points (i.e., three `point` variables) 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 (i.e., one `point` variable) 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 (see above description of `slope` function) will be set appropriately so that the calling function will know not to use the returned value.
- Three points are collinear if the slope between one pair of the points equals the slope between a different pair of the points. So, make use of the `slope` function in `collinear` to determine its result.

Point Topics Review:

- Slope of the line between (a,b) and (c,d), $m = \dfrac{d-b}{c-a}$ provided $a \neq c$; slope is undefined for vertical lines.
- Distance between (a,b) and (c,d): $D = \sqrt{(c-a)^2 + (d-b)^2}$
- Midpoint between (a,b) and (c,d): $M = \left(\dfrac{a+c}{2}, \dfrac{b+d}{2} \right)$

- The equation of a non-vertical line passing through (a,b) and (c,d) is given by $y = mx + (b - ma)$ where *m* is the slope of the line.
- The equation of a vertical line passing through (a,b) and (a,d) is given by $x = a$ 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 => 6
```

3. (10 pts.) 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.

Your `elevator` class should have appropriate member data, constructors, and member functions to fulfill the above requirements.

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.

Program requirements:

- 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
```

4. (15 pts.) Create a class, called `tictactoe`, which will allow you to write a complete program to play the game of tic-tac-toe. The `tictactoe` class should allow the play of a game of tic-tac-toe by two human players. The class should allow moves only to empty positions and should determine when someone wins the game or when the game is a draw. Here is what should be included in this class:

- Member data should include a private 3-by-3 array of integers to model the board
- A no-argument constructor that initializes an empty board to all zeros
- Public member functions to perform these actions:
    - `clearboard` – resets the board to empty
    - `showboard` – displays the board
    - `getXmove` – asks player X for a position and makes that move; you can assume the user will enter integers when asked for a position, but this function should check to make sure the position is valid (row and column numbers between 1 and 3 inclusive) and that the position has not already been played – appropriate error messages should be displayed in either case and the user should be asked to re-enter the position
    - `getOmove` – asks player O for a position and makes that move; you can assume the user will enter integers when asked for a position, but this function should check to make sure the position is valid (row and column numbers between 1 and 3 inclusive) and that the position has not already been played – appropriate error messages should be displayed in either case and the user should be asked to re-enter the position
    - `checkwin` – determines if someone has won the game or if the game is a draw or if the game is still in progress; the function should return an integer flag that indicates the result of the check (i.e., player X wins, player O wins, the game is a draw, or the game is still in progress)

Hint: The class array models the board and, as noted above, the initial board should contain all zeros. One way to store moves of the game is to store a +1 in the appropriate array position for a player X move and a -1 in the appropriate array position for a player O move. Be careful when handling the array and the user row/column entries – array indexing is not the same as row/column numbering.

After you have written your class, write a `main` driver program that will create a game of tic-tac-toe, and then repeatedly play tic-tac-toe games until the user chooses to stop.

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

```
Time for tic tac toe...

        -       -       -

        -       -       -

        -       -       -


Enter X play position (row# column#): 1 1

        X       -       -
```

```
        -       -       -
        -       -       -
Enter O play position (row# column#): 2 2

        X       -       -
        -       O       -
        -       -       -

Enter X play position (row# column#): 1 3

        X       -       X
        -       O       -
        -       -       -
Enter O play position (row# column#): 1 2

        X       O       X
        -       O       -
        -       -       -

Enter X play position (row# column#): 3 2

        X       O       X
        -       O       -
        -       X       -
Enter O play position (row# column#): 2 1

        X       O       X
        O       O       -
        -       X       -

Enter X play position (row# column#): 2 3

        X       O       X
        O       O       X
        -       X       -
Enter O play position (row# column#): 3 3

        X       O       X
```

```
        O       O       X

        -       X       O


Enter X play position (row# column#): 3 1


        X       O       X

        O       O       X

        X       X       O

Game a draw...
Play again? (y or n): n
```

For those that feel ambitious, add another member function, `compOmove`, which will allow the computer to select the player O moves. If you attempt to do this, plan out your strategy for player O carefully before writing any code. Then, modify your main to allow a choice of human versus human play or human versus computer play. (Note: If anyone would like to see a version of the program that allows this option, ask me for an .exe file of such a program.)