There will be a handmarking component in this assignment, whose purpose is to ensure that you are following an appropriate standard of documentation and style, and to verify any assignment requirements not directly checked by Marmoset. Please code to a standard that you would expect from someone else if you had to maintain their code. Further comments on coding guidelines can be found here: [https://www.student.cs.uwaterloo.ca/~cs247/current/assignment.shtml](https://www.student.cs.uwaterloo.ca/~cs247/current/assignment.shtml). Please be aware that if handmarking shows you have not followed the requirements of the question, correctness marks from test cases can partially or totally be taken away.

Some or all of these programs will ask you to develop C++ modules in order to make a provided test harness file which includes those modules work. Since you must use the main provided in the test harness to test your program, it is suggested that you first begin by providing empty implementations of all of the necessary functions. Then you will at least be able to compile, and as you implement each function you can recompile and test them immediately.

Your programs will be tested against the provided sample executables, so you can do the same! Use bash input/output redirection to generate files and compare them with the `diff` command to check if you are producing the same output as the sample executable. For example, if your executable is named `my_a1q1` and is in the same directory as the provided sample executable `a1q1` then you could do the following bash commands to test (assuming you’re in that current working directory):

```bash
./a1q1 < sample.in > sample.out
./my_a1q1 < sample.in > my_sample.out
diff sample.out my_sample.out
```

It is suggested you create many input files for your specific test cases, and use this procedure to test your program. Even better, a bash script could be written to automate checking all of your tests! These assignments are in testing as much as they are in coding! Without thinking of the appropriate test cases and checking them it will be hard to get full marks! If you didn’t think about a test case, even if it is not explicitly pointed out to you in the specification, that doesn’t mean you deserve those marks.

**Note:** All of your program must not leak resources. If your program produces the correct result but in doing so leaks resources then test case will not be considered as having been passed. We will use the tool valgrind to test your programs (which is available on the student servers) so you can too. Your code must compile and run correctly on the student server.

**Allowed headers for this assignment:** For this assignment you may only use the headers `<iostream>`, `<utility>`, and `<cmath>`. Inclusion of any other headers (not created by you) may result in a loss of some or all of your marks.

0. This course will be using a public git repository to help me with disseminating resources to you. This question is to help you access those materials and make sure you have done so. The process is very simple, but it must be done from your student.linux.cs.uwaterloo.ca server.
For instructions on accessing your student account, see the following website: https://uwaterloo.ca/math-faculty-computing-facility/accessing-student-linux-servers.

**Important:** the instructions on that page are for the math student servers. The process to connect is the same for the CS student servers, but the server is different. While those pages will tell you to use UWuserid@linux.student.math.uwaterloo.ca you should use UWuserid@linux.student.cs.uwaterloo.ca. Note the difference (cs instead of math)!

Once you have connected to the server you will be on a Linux environment (specifically Ubuntu). This is where you should be testing all your assignments, as we will be testing them on these same servers. Now that you are on the student server run the following command to create a copy of the course repository:

```
git clone ssh://linux.student.cs.uwaterloo.ca/u/cs247/pubrepo/1195/.git
```

Then you should have a directory named 1195 in your current working directory. In that directory you should find (currently) two directories, lectures and a1. lectures is the directory that stores code examples from lectures, and a1 contains the provided files for this assignment.

In the future more examples will be added to the lectures directory, as well future assignment files will be provided here as well. Whenever you want to get the most up to date files in this directory all you have to do is navigate to it as your current working directory, and run the command `git pull`. It is advised that you do this everytime you go to work on CS247!

**Deliverables:** In the a1 directory you will find a file named q0.txt. Submit that exact text file as your answer to this question.

1. In this question you will be developing some very basic ADTs and operations to simulate very simplistic forces in a two-dimensional plane. You will have to develop the classes Point, Force, and accompanying operations in order to work with the provided test harness a1q1.cc.

   The Point class will model a point in a two-dimensional plane with a given x and y float value. The Force class will model a force in two-dimensional plane, with a given angle (in degrees), and magnitude. Angles in degrees start from the line out in the positive x direction of the cartesian plane, and wraps back around at 360 degrees. See below for an example:
You have been provided a header 2DMotion.h, which you may change as you like. However, you must implement all the functionality used by the provided test harness file a1q1.cc, as that file (or a similar one in how it uses the Point and Force classes) will be used to test your program. You may not change the test harness, as we will test you with our own copy of it, so any changes you make won’t be reflected in how your solution is tested! You must implement the following functions for this question:

- **Point** default constructor. Should initialize both x and y to 0.
- **Force** default constructor. Should initialize both angle and magnitude to 0.
- An overloaded input operator for **Point** objects. Should read in first the x field, then the y field.
- An overloaded input operator for **Force** objects. Should read in first the angle, then the magnitude.
- An overloaded output operator for **Point** objects. Should print them out in the format: "(x, y)" (Note <variable> is used to denote that the variables value should go there, so if the point has x value 4, and y value 5, you should print out “(4, 5)”.
- An overloaded output operator for **Force** objects. Should print them out in the format: "<degrees> degrees with magnitude of <magnitude>".
- An overloaded addition operator between a **Point** object and a **Force** object. This should effectively create a new **Point** that is the result of “moving” the original by that **Force**. This requires a bit of trigonometry! You will require the `<cmath>` header, and should use the PI constant defined in the provided 2DMotion.h file. In order move a **Point** by a given **Force** you must determine the horizontal and vertical components of the given **Force**. Doing so is simple trigonometry. Consider that the magnitude of a **Force** is simply the hypotenuse of a right-angle triangle. Given the hypotenuse and angle of a right-angle triangle you can easily find out the length of the other sides (the horizontal and vertical components) using sin and cos provided in the `<cmath>` library - but be wary, those operations work on radians!
- An overloaded multiplication operator between a **Force** and a float scalar. This should simply produce a new **Force** which has a magnitude scaled by the given scalar.
- **int Point::quadrant()** - A member function that returns the quadrant (1, 2, 3, or 4) that the given point is in. See quadrants in diagram above.

**Deliverables:** You must include in your submission a1q1.zip your files 2DMotion.h and 2DMotion.cc, no other files other than our own test harness will be used to compile your solution - so do not create any additional files.

2. For this question you may not use any STL container, nor may you use any STL smart pointers. You must implement the class in question by managing memory yourself. These headers are already banned from the assignment, but this is to remind you. You may create any helper classes you want yourself to help you manage memory.

Connect Four is a two-player game in which players take turns dropping pieces into a columns of a grid. The pieces, of course, fall to the lowest unoccupied spot in that column because gravity. The most famous iteration of Connect Four is played on a 6 × 7 grid (6 rows, 7 columns).

The goal of Connect Four is to be the first player to have a contiguous line of four pieces all of your colour either horizontally, vertically, or diagonally.

In this assignment you will create a game ConnectX, which is played on an arbitrary n × m grid, and the goal is to be the first player to connect X pieces all of your colour, either
horizontally, vertically, or diagonally. You will do so by implementing the \texttt{ConnectX} class. In your game the two players will be referred to as blue and red, and the characters B and R represent their pieces on the board respectively, while a space character represents an empty space. Blue will always play first.

You have been provided a header for the \texttt{ConnectX} class, which you may change as you like. However, you must implement all the functionality used by the provided test harness file \texttt{a1q2.cc}, as that file (or a similar one in how it uses the \texttt{ConnectX} class) will be used to test your program. You may not change the test harness, as we will test you with our own copy of it, so any changes you make won’t be reflected in how your solution is tested!

The required functions are listed below, for their exact behaviour refer to the sample executable:

- \texttt{ConnectX(int, int, int)} - A basic constructor that takes in three positive integer parameters, the number of rows, the number of columns, and the target goal X.
- A copy constructor, which must be a deep copy.
- A move constructor, which must be a constant time operation.
- A copy assignment operator, which must be a deep copy.
- A move assignment operator, which must be a constant time operation.
- A destructor, which must free all resources the object holds.
- \texttt{void ConnectX::playTurn()} - A member function that plays the turn for whichever players turn it currently is. For full details of its behaviour refer to the sample executable. At a high level it should:
  (a) Print out a prompt to \texttt{cout} for the current player to enter the column they’d like to play “Blue/Red player enter a column to play” (where Blue/Red is whichever one is appropriate for current player).
  (b) Read in an integer from \texttt{cin}.
  (c) If the integer read in is not within the bounds of the game board, or is a column that is already entirely full, the message “Invalid move! Try again” should be printed, and the player should be prompted again for an input.
  (d) After reading a valid column to play, the spot played in should be updated appropriately, and if that would end the game then the message “Blue/Red player wins” should be printed, where again Blue/Red is whichever one is appropriate.
- An overloaded output operator that prints out the board, row by row, one row per line. It should use pipe characters ‘|—|’ at both the left and right edges of each row, as well as inbetween spots. A spot with a blue piece should be printed as B, a spot with a red piece should be printed as R, and an open spot should be printed as a space character. Each row should end with a new line, including the last one.

\textbf{Memory Management Requirements:} You must manage your own memory in this assignment, using \texttt{new} and \texttt{delete}. All copies of objects must be deep copies (can test this behavior in sample executable). Additionally your move constructor and move assignment operator must be constant time! Your code will be handmarked for these requirements, if you fail to meet them then you will lose some or all of your correctness marks.

\textbf{Deliverables:} You must include in your submission \texttt{a1q2.zip} your files \texttt{ConnectX.h} and \texttt{ConnectX.cc}, no other files other than our own test harness will be used to compile your solution - so do not create any additional files.
3. **Thought Question** - For this question you are to come up with a question about something you’d like to know more about from the lectures or the assigned readings. Be creative, as long as it is tangentially related to C++ or the OOP concepts discussed in class it is acceptable. I will do my best to answer all of your questions in a reasonable amount of time (but make no promises).

**Deliverables:** For this question you must send me an e-mail at r2hackma@uwaterloo.ca. The subject of the e-mail **MUST be exactly CS247-A1Q3** or it will not be auto-filtered into my appropriate folder and I will not see it. You **must** send this e-mail from your UWaterloo account, and tell me in it your name and Quest ID. Also it must, of course, include your question :)