INTRODUCTION TO PROGRAMMING
BERKELEY CITY COLLEGE
FALL 2019
LAB 4

FOR MULTIPART QUESTIONS, UPLOAD EACH PART AS YOU COMPLETE IT. Each part builds on the previous part: i.e. if you did not complete 1.a correctly, you'll have a hard time implementing 1.b

1.a) Points

Write a class called Point with the following features:

- Private variables called xCoord, yCoord, and zCoord. These should be stored as floats
- A public constructor that takes three floats and assign those floats to the relevant private variables.
- A public constructor that takes no parameters and assigns the value 0 to xCoord, yCoord, and zCoord
- Public functions called getX(), getY(), and getZ() that return (as a float) the value of the relevant variables.
- Public functions called setX(), setY(), and setZ(), all of which take a float as a parameter and sets the relevant variable to equal that float.

1.b) Spheres

Write a class called Sphere with the following features:

- A private variable called center. This should be stored as a Point
- A private variable called radius. This should be stored as a float.
- A public constructor that takes a Point and a float and creates a Sphere object with Point as the center and the float as the radius
- A public constructor that takes a float and creates a Sphere object with the float as the radius and with a Point at (0,0,0) as the center.
- A public function called getCenter(), that returns a Point representing the center of the Sphere, and a public function called getRadius(), which returns as a float the radius of the sphere.
- A public function called getVolume(), which returns (as a float) the volume of the sphere. To find a sphere's volume, use this formula: volume = $4\Pi r^3 / 3$
- A public function called getSurfaceArea(), which returns (as a float) the surface area of the sphere. Use this formula: surface area = $4\Pi r^2$

2.a) Karel objects

Write a class called Karel with the following properties:

- private variables called row and column that represent Karel's current location. These should be stored as ints.
- a private variable called direction. Define a enum of named Direction, with NORTH, WEST, SOUTH, and EAST as the valid values for an object of the Direction type.
- A constructor that takes three parameters: a row, a column, and a direction. This constructor should set the relevant private variables for this Karel object
- A constructor that takes no parameters and sets Karel's row and column to 0, facing EAST
- functions named getRow(), getColumn(), and getDirection() that return the relevant private variables.
- note: there should not be functions (other than the ones below) to change Karel's private variables.
- A function called move(), which moves Karel one space forward in the direction that they're facing. If a move() results in Karel being off the board or inside a wall, the program should crash.
- A function called turnLeft(), which changes Karel's direction.
- A function called safeMove(), which only moves if Karel can safely move in that direction.
- A function called putBeeper(), which puts down a Beeper object at Karel's current square
- A function called pickBeeper(), which picks up a Beeper object at Karel's current square. If there are no beepers present, the program should crash.
- A function called safePickBeeper(), which only tries to pick up a beeper if there is a beeper present.

Write a class called Beeper with the following properties:

- Private variables called row, and column. These should both be ints. row and column represent the location of a Beeper object.
- A public constructor that takes two variables (row and column) and creates a new Beeper object with its private variables set appropriately.
- A public constructor that takes no variables and creates a new Beeper object with row == 0 and column == 0
- Functions to both get and set the private variables.

2.b) Karel world

Write a class called GameWorld with the following properties:

- A private variable representing the state of the game board. This could be represented by a 2d array, although there are other valid ways to represent it.
- Private constants called DEFAULT_ROWS and DEFAULT_COLUMNS, that represent the default size of the game board
- A constructor that takes two ints as parameters and sets the size of the game board based on those ints.
- A constructor that takes no parameters and creates a board of default size
- A function called inBounds() that takes a row and a column as parameters. inBounds() should return true if that row and column pair are inbounds, and false otherwise. There can be walls on the board. If the row and column pair are inside a wall, inBounds() should return false.
- A function called beepersPresent(), which takes two parameters and returns true if there are one or more Beeper objects at the row/column represented by the parameters.
- A function called printWorld(), which prints the world. Represent squares containing Karel as an O, squares containing any nonzero number of beepers as an x, and Karel standing on a beeper or beepers as @. Represent a wall as #. Print - for unoccupied squares.

2.c) Karel filereading

The following functions should be added to your GameWorld class:

- A public function called boardFromFile(), which should read in a file (filename passed in as a parameter) and convert it into a GameWorld object. The format of the file will be the same as the format of the text produced by printWorld(). The following file would create a world with three rows and seven columns, with Karel standing on a beeper at row 2, column 0, with walls at (row 0, column 3), (row 1, column 3), (row 1, column 5), and (row 2, column 5), and a beeper at (row 2, column 6).

```
---#---
---#-#-
@----#x
```

- A public function called boardToFile(), which writes the board to a file. Use the format above. The filename should be passed in as a parameter.

Note: On many compilers, the function for opening files wants a C-style string instead of a C++ string. use the function c_str() to convert a C++ string to a C-string:

```
string myFile = "hello.txt";
file.open(myFile.c_str());
```

2.d) Play Karel

Create a main() function that uses the GameWorld, Karel, and Beeper objects to allow the user to repeatedly input Karel commands. The following are the valid inputs from the user -- if they give you anything else, tell them their input is invalid and request another input. User input must EXACTLY follow the format below: the name of the command, a pair of empty parentheses, and a semicolon.

move();
safeMove();
turnLeft();
readFile();
writeFile();
putBeeper();
pickBeeper();
safePickBeeper();
quit();

NOTE: THIS NEXT ONE REQUIRES SOME THINKING. WRITE OUT YOUR ALGORITHM ON PAPER AND CONVINCE YOURSELF THAT IT'S RIGHT BEFORE YOU TRY TO IMPLEMENT THE FULL PROGRAM. THERE ARE MANY, MANY DIFFERENT WAYS TO DO THIS. I'LL ACCEPT ANYTHING THAT PRODUCES THE RIGHT ANSWER.

3.a) uwu part 1

Let's call a string containing the characters "uwu" an *adorable string.* When testing a string for adorability, it does not matter whether or not the characters u, w, and u appear side-by-side, so long as they appear in that order. "uwu" is an adorable string, as is "uwuwu", but "wuu" and "uuw" are not adorable strings. Also, though, "uxwxu" is considered an adorable string, since although the characters u, w, and u are separated by the character 'x', they still appear in the string in the proper order. Strings count as adorable regardless of the case of the characters: "UwU" is adorable, as are "uWu" and "UWu".

Write a function that takes a string and returns whether or not that string is adorable. Your function must have this prototype:

```
bool isAdorable(string);
```

3.b) uwu part 2

Define the *adorability count* of a string as the number of occurrences of u, w, and u in that order in the string. The string "uwu" has an adorability count of 1, the string "OwO" has an adorability count of 0, and the string "did yo**u** **w**atch nar**u**to?" has an adorability count of 1, because the letters u, w, and u appear in that order. However, the string "do yo**u** **w**ant to r**u**n like nar**u**to?" has an adorability count of 2: the first occurrence of uwu consists of the u in "you", the w in "want", and the u in "run", while the second occurrence of uwu consists of the u in "you", the w in "want", and the u in "naruto". The string "uwu uwu" has an adorability count of 6. I've indicated each adorable substring with underlines:

uwu uwu
uwu uwu
uwu uwu
uwu uwu
uwu uwu
uwu uwu

Write a function that takes a single string as input, then returns its adorability count. Your function must have the following prototype:

```
int adorableCount(string);
```