C++ Class Exercise
Foundation Course SS 2018

Dharmin B.

March 28, 2018

1 Pizza counting bot

Write a C++ program that asks the user to enter the number of pizza slices eaten in a party by 10 different people (P1, P2, ..., P10)
Once the data has been entered the bot must analyze the data and output which person ate the most number of pizza slices.
(HINT : Use array to store the input provided by the user.)

Modifications

- Modify the program so that it also outputs which person ate the least number of pizza slices in the party.
- Modify the program so that it outputs a list in order of number of slices eaten by all 10 people. For example

```
P4:  ate 10 slices
P3:  ate 7 slices
P8:  ate 4 slices
...
P5:  ate 0 slices
```

2 Area calculator Bot

Make a OOP C++ program which calculates area of rectangles and triangles. Ask the user for two numbers. Print the area of rectangle(length * height) and triangle(0.5 * length * height).

OOP Conditions

- Make 3 classes, one of type shape, one for rectangle and one for triangle.
- In main, create an object for rectangle and one for triangle.
- The rectangle and triangle class should inherit from shape class. Height and length should not be public.

Example

input	Output
Length : 5 Height : 3	Triangle : 7.5 Rectangle : 15.0
Length : 4 Height : 6	Triangle : 12.0 Rectangle : 24.0

3 Guessing Bot

Write a C++ program that calculates a random number 1 through 100. The program then asks the user to guess the number. If the user guesses too high or too low then the program should output "guess higher" or "guess lower" accordingly. The program must let the user continue to guess until the user correctly guesses the number.

Modification

- Modify the program to output how many guesses it took the user to correctly guess the right number.
- Modify the program so that instead of the user guessing a number the bot came up with, the bot guesses the number that the user has secretly decided. The user must tell the bot whether it should guess higher or lower. The bot should randomly but not blindly. The bot should improve its guesses based on the feedback provided by the user.
- Modify the program so that no matter what number the user thinks between 1 and 100, the bot can guess it in 7 or less guesses. (HINT : Here the bot will not be guessing randomly.)

4 Treasure Game Simulator

Make a OOP C++ program that outputs a simple grid based game board (10 x 10) to the screen.
One example is shown below.

```
.  .  .  .  .  .  .  .  .  .
.  P  .  .  .  .  .  .  .  .
.  .  .  .  .  .  T  .  .  .
.  .  .  .  .  .  .  .  .  .
.  .  .  .  .  .  .  .  .  .
.  .  .  .  T  .  .  .  .  .
.  .  .  .  .  .  T  .  .  .
.  .  .  .  .  .  .  .  .  X
.  .  .  .  .  .  .  .  .  .
.  .  .  .  .  .  .  .  .  .
```

The player (marked with P) should be able to move either up, down, left, or right each turn. If the player steps on a trap(marked with T) then they lose. If the player makes it to the treasure(marked with X) then they win.
If the player goes outside then they lose.

OOP Conditions

- Create a class for board, a class called MovingObject and a class for player. Player class should inherit from MovingObject class.
- Traps for modification should inherit from MovingObject.
- Position of MovingObject should not be public.
- Call deconstructor of traps when they move outside(for modification).
- Position of treasure should be private in board class with getter methods. The position of treasure should be set in constructor of board.

Modification

- Allow the traps to move randomly in any direction(up, down, left or right) once per turn. If traps go outside the board, they get deleted. The traps cannot move into each other. The traps cannot move into treasure position.

5 Wolf Colony Simulator

Write a OOP C++ program that creates an array(size 100) of wolf objects. Each wolf object must have properties

- sex: Male, Female (random at creation 50/50)
- color: white, brown, black, mixed
- age : 0-10 (years old)
- name : randomly chosen at creation from a list of wolf names (at least 10 different names).
- radioactiveWolf: true/false (decided at time of wolf creation 2% chance of true)

At program initialization 5 wolves must be created with random colors.
Each turn afterwards the wolves age 1 year.
So long as there is at least one male age 2 or older, for each female wolf in the list age 2 or older; a new wolf is created each turn. (i.e. if there was 1 adult male and 3 adult female wolves, three new wolves would be born each turn)
New wolves born should be the same color as their mother.
If a wolf becomes older than 10 years old, it dies.
If a radioactive wolf is born then each turn it will change exactly one non radioactive wolf into a radioactive wolf.(if there are two radioactive wolves two wolves will be changed each turn and so on...)
Radioactive wolves are excluded from regular breeding and do not count as adult wolves.
Radioactive wolves do not die until they reach age 20.
The program should print information about wolves in the colony each turn in a file as follows

```
Iteration 23
Adult Male :  5
Adult Female :  8
Kid Male :  2
```

```
kid Female :  3
Radioactive :  1
```

The program should also output(on screen) each turns events such as

```
wolf Henry was born!
wolf Elisa was born!
Radioactive wolf DarthVader was born!
wolf Wolverine died!
```

When all the wolves have died the program terminates.
If the wolf population exceeds 90 a food shortage must occur killing exactly 90% of the wolves (randomly chosen)

OOP Conditions

- There should be a wolf class. The property variables of wolf class should be private and only accessible by getter and setter method.
- Create functions like `isAdult` which returns boolean.
- Create functions for writing into text files.
- **For grid modification**
  - Create a new class for board. Create one object of this class from main. This class should contain the objects of all wolf objects.
  - The position of wolf should be a parameter of wolf object.
  - Wolves cannot move outside the board.
  - Create function called `isEmpty` to check if a location in grid is empty or not.

Modifications

- Allow the user to hit the 'k' key to initiate a mass wolf cull! which causes half of all the wolves to be killed (randomly chosen).

- Modify the program to place the wolves in an 10 x 10 grid. Have the wolves move one space each turn randomly. Represent young males with m, adult males with M, young females with f, adult females with F and radioactive wolves with X
  - Modify the program so that radioactive wolves only convert wolves that end a turn on an adjacent square.
  - Modify the program so that new baby wolves are born in an empty random adjacent square next to the mother wolf. (if no empty square exits then the baby wolf isn't born)