CIS*1500: ASSIGNMENT 1

Weighing: 10%
Due: October 18, 2015 - 11:55 PM

Choose **1** of the **2** options.

1. THE GAME OF PIG

Pig is a simple dice game first described by John Scarne in 1945[1]. It is a form of jeopardy dice game, where a players decisions are between protecting current gains, or risking current gains to make greater gains.

In *Pig*, each player takes a turn rolling a 6-sided dice, playing to reach a certain total (typically 100). If a player rolls a value between 2 and 6 it is added to their score, and they can roll again. If they roll a 1 (a "pig"), they lose their turn and add nothing to their score. At any point after rolling a 2-6, the player may choose to to *hold*, which ends their turn, and adds their turn total to their score. The winner is the first player to get a score greater than or equal to the total.

When looking at the simple rules of the game, one can surmise that the game involves some form of repetition to deal with the different turns of the players, and decision statements to deal with the choice of roll or hold. The game also involves a random number generator to simulate the dice. The game has two players, so an algorithm may perform like this (starting with one player):

1. Ask the player whether they wish to roll or hold?
    1.1. If **roll** is chosen, use the random number generator to generate a number between 1 and 6.
        1.1.1. If the number "rolled" is 1, go to step **2**.
        1.1.2. If the number "rolled" is 2-6, the value is added to the players *roll total*. Go to step **1**.
    1.2. If **hold** is chosen, transfer the *roll total* to the players total. Go to Step **2**.
2. Is one of the players total greater than or equal to the total?
    2.1. Yes - the game ends.

---

[1] Scarne, J., Scarne on Dice. Harrisburg, Pennsylvania: Military Service Publishing Co. (1945)

2.2. No - go to step **3**.

3. Control is passed to the other player. Go to step **1**.

Task

Design and implement a C program to play the *Game of Pig*. The game will be played with two human players taking alternate turns. The total for the game (which is typically 100), should be input by the user before the game begins. The program should display the dice rolled, for example:

```
+-------+
| o   o |
|   o   |
| o   o |
+-------+
```

Below is a skeleton for a function called **showDice()**. This function has one parameter, the number rolled by the "dice" (random number). It displays that appropriate graphic. This is a basic function and does not require any special knowledge.

```
void showDice(int num)
{

}
```

Your program should include the following: decision statements, loops, character input, random number generation.

See *Program Guidelines* at the end of this assignment.

Test Program Execution

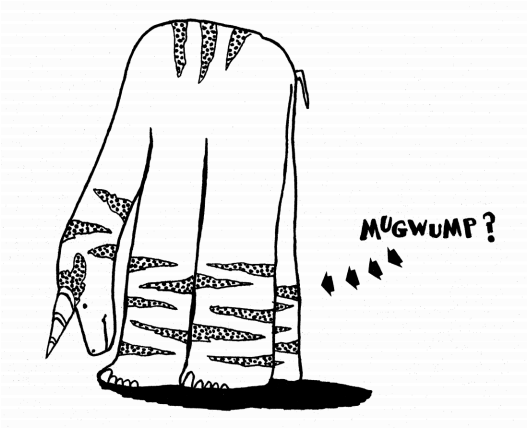
An example of a program with these features is shown running below (user inputs are **bolded**):

```
Welcome to the Game of Pig
==========================
What is the game total? 20
--------------------------
Player 1 (score = 0)
--------------------------
```

```
(r)oll or (h)old?
> r
dice rolls a

     +-------+
     | O   O |
     |       |
     | O   O |
     +-------+

Roll total: 4
(r)oll or (h)old?
> r
dice rolls a

     +-------+
     | O     |
     |       |
     |     O |
     +-------+

Roll total: 6
(r)oll or (h)old?
> h
Player 1 score = 6

Player 2 (score = 0)
-------------------------
(r)oll or (h)old?
> r
dice rolls a

     +-------+
     |       |
     |   O   |
     |       |
     +-------+

Sorry, loose a turn!
Player 2 score = 0

Player 1 (score = 6)
-------------------------
...
```

2. FIND THE MUGWUMP

Find the Mugwump was published as a game written in Basic, and is attributed to students of Bud Valenti of Project SOLO in Pittsburgh Pennsylvania.



The objective in this game is to find the Mugwump hiding on a square of a 10 by 10 grid. Homebase (upper left) is position (0,0) and a guess is a pair of whole numbers (0 to 9), separated by a space. The first number is the number of units to the below of homebase and the second nurnber is the distance to the right of homebase. You get ten guesses to locate the Mugwump ~ after each guess, the computer tells you how close you are to the Mugwump. Playing the game with the aid of graph paper and a compass should allow you to find all the Mugwumps in six or seven moves using triangulation.

Task

Design and implement a C program to play the *Find the Mugwump*. The game will be played with two human players taking alternate turns. The game has the following constraints:

- The board will be restricted to 10 ×10 squares in size. User input will be in the form 0 to 9, where 0 represents the first row or column, and 9 represents the tenth row or column.
- User input will be in the form of x-coordinate y-coordinate
- The position of the mugwump will be randomly generated.

The program should display the users guess graphically, for example:

```
  0 1 2 3 4 5 6 7 8 9
0 . . . . . . . . . .
1 . . . . . . . . . .
2 . . . . . . . . . .
3 . . . . W . . . . .
4 . . . . . . . . . .
5 . . . . . . . . . .
6 . . . . . . . . . .
7 . . . . . . . . . .
8 . . . . . . . . . .
9 . . . . . . . . . .
```

Below is skeleton for a function called **showBoard()**. This function has two parameters representing the x and y coordinates of their guess. It displays that position on a board. This is a basic function and does not require any special knowledge.

```
void showBoard(int x, int y)
{

}
```

Your program should include the following: decision statements, loops, character input, random number generation.

See *Program Guideline*s at the end of this assignment.

Test Program Execution

An example of a program with these features is shown running below (user inputs are **bolded**):

```
What is your guess (x y)? 3 4
You are 5.0 units from the Mugwump.
  0 1 2 3 4 5 6 7 8 9
0 . . . . . . . . . .
1 . . . . . . . . . .
2 . . . . . . . . . .
3 . . . . W . . . . .
4 . . . . . . . . . .
5 . . . . . . . . . .
6 . . . . . . . . . .
7 . . . . . . . . . .
8 . . . . . . . . . .
9 . . . . . . . . . .
Turn number 2
What is your guess (x y)? 3 9
You are 2.0 units from the Mugwump.
  0 1 2 3 4 5 6 7 8 9
0 . . . . . . . . . .
1 . . . . . . . . . .
2 . . . . . . . . . .
3 . . . . . . . . . W
4 . . . . . . . . . .
5 . . . . . . . . . .
6 . . . . . . . . . .
7 . . . . . . . . . .
8 . . . . . . . . . .
9 . . . . . . . . . .
Turn number 3
What is your guess (x y)? 5 9
```

ASSIGNMENT GUIDELINES

Program expectations

- All files are inside your A1 folder.
- If you choose Topic 1 the filename is called **a1PIG.c**. If you choose Topic 2 the filename is called **a1MUG.c**.
- Source code has the proper style (e.g. indentation).
- A plain text file called README is in the root folder. It contains information about running and using your program as well as any known limitations of the program. (see the policies document)
- You may not use any global variables, or goto statements in this program!

Musts: How to get more than ZERO

- Submit your assignment before the due date!
- Your code must compile on a Raspberry Pi that is running the CIS*1500 customized operating system.
- Your code must compile, with no errors or warnings, with gcc using the flags **-std=c99 -Wall -pedantic**.
- Your program must run without crashing or causing a "segmentation fault".
- Your program must play the game, and show the outcome.

NOTE that code that fails to compile will receive an automatic grade of zero.

Deliverables:

The required **.c** file, and the README file.

Marks breakdown

Refer to the rubric for this assignment.

Grading

This assignment is graded by a TA in person. You must arrange a booking to grade your assignment. At the time of the appointment you must present your assignment and it will be compiled.

- Failure to show at the first appointment will result in a 10% reduction in the assignment grade. You will have to re-book the appointment.
- Failure to show at the second appointment will result in another 10% reduction in the assignment grade. Failure to show for the third appointment will result in a grade of zero.
- Failure to make an appointment in the allotted time period for grading will result in an automatic grade of zero.

PLEASE NOTE:

Assignments that compile with errors or warnings get a grade of zero.
Assignments that do not run will get a grade of zero.

Late Assignments:

- Late assignments will be assigned a grade of **zero**. There are no makeup assignments.

Regrades:

- Requests made within 5 business days of receiving your mark.
- Requests must be emailed to cis1500@socs.uoguelph.ca
- Requests must have the word **regrade** and the name of the assignment or exam in the subject line
- Requests must contain a detailed description of why you feel the assignment should be regraded
- A regrade is not a chance to redo the assignment. The original submission will be graded.