Specification

Assignment 1: Left, Right Center

Files needed:

lrc.c: the C file that contains my left, right, center program.

philos.h: a list of up to 14 philosophers who are the players of the games

Makefile: a make file to easily run the program by prompting `make` into the Linux terminal on our Ubuntu virtual machine.

README.md: describes how the program works and runs in addition to how to use the makefile

Description of the Program: In this game of Left, Right and Center you will need at least 2 players and at most 14 players. Each player will have a person on the left and a person on their right side. Each player also starts with 3$ which represents how many dice they roll up to 3 dice. (e.g. if you have x > 3$, you will roll 3 dice). If you have $0, you do not roll any dice and are out of the game. The dice however are unique and have 4 symbols; L,R,C, and 3 × •. If you roll "L" you pass a $1 to the left player, if you roll "R" you pass a $1 to the right player, and if you roll "C" you put a $1 in the center pot. Lastly if you roll "•" you ignore and proceed with the game. Last man standing wins.

Functions involved:

For this particular game I intend to use 3 functions to build this program: main(), left(), and right(),

main():

```
Input a seed:
Input how many players are playing:
  Assign money = 3
  While loop:
  Checking array for amount of money
  If $ > 3
     How many times = roll(3)
  Else
     Roll = how much money you have
    for (dice =1, dice < roll, dice++)
    Switch:
    case = left
       -1 from current player
       +1 to left player
    case roll = right
       -1 from current player
       +1 to right player
    case roll = center
       -1 from current player
       +1 to center pot
    case:skip
      Skip(ignore)
```

```
Check how many players have more than 0$
For
  If Check money > 0
  One active player

If active player = 1
print(winner)
End

Else
Increment to next player
Back to top of while loop
```

You will first be prompted to enter in a random seed. The reason is because randomness doesn't completely occur in computers so to obtain a fair game we need a random seed. Next they will ask us to prompt a number of players 1 < players ≤ 14. Next we need to simulate the game in which player 1 will begin rolling. To do so we put it through a while loop because we need it to run until there is only 1 player left with a minimum of $1. In this while loop the **roll()** will also be called. But before we can tell the player to roll we must check how much money they have to determine how many times they roll. These conditional statements will simply be 1 if statements that say if you have more than $3 you will roll a fixed number 3. In the else statement you roll however much money you have. (e.g. if you have 2$ you roll twice, if you have $0 you roll 0 times). The rolling system is quite intuitive in that it does exactly what it's supposed to do. As described in the program description L means pass left, R means pass right and C means put in center. These conditions will be checked through a switch statement. We will perform calculations using the **right_person() and left_person() functions** in order to see who will get +1$. We will also be sure to incorporate a subtraction to the current person who rolled since that is part of the deal too. Roll will only be called if the player still has at least $1. Lastly there is a checker after every round that shows how many players are still in after a round. If there is one player left you will get a winner message. Else we will increment one and repeat the process till a winner is declared

left_person():

CREDIT: PROFESSOR // SOURCE: Assignment 1 Lab Document

```
1 //
2 //  Returns  the  position  of the  player  to the  left.
3 //
4 // pos:      The  position  of the  current  player.
5 // players: The  number  of  players  in the  game.
6 //
7  static  inline  uint8_t  left(uint8_t pos , uint8_t  players) {
8    return  ((pos + players  - 1) % players);
9 }
```

As described above this function contains a simple algorithm to help determine whose players are of the left of them. It is important to note that the philosophers are in a fixed order so whoever is of the left of Them will always stand true no matter what.

right_person():
CREDIT: PROFESSOR // SOURCE: Assignment 1 Lab Document

```
1 //
2 //  Returns  the  position  of the  player  to the  right.
3 //
4 // pos:      The  position  of the  current  player.
5 //  players: The  number  of  players  in the  game.
6 //
7  static  inline  uint8_t  right(uint8_t pos , uint8_t  players) {
8    return  ((pos + 1) % players);
9 }
```

Similar to the left() function, the right function contains basically an identical algorithm with right in place of left unlike last time. This will determine who's on the right of them and if there is no one on the right of them it will loop back to start of the list because the first member does not have a person to the left of them. (e.g. 13->14->1)

Bonus mention:
Dice:
CREDIT: PROFESSOR // SOURCE: Assignment 1 Lab Document

```
1  typedef  enum  faciem { PASS , LEFT , RIGHT , CENTER } faces;
2 faces  die[] = { LEFT , RIGHT , CENTER , PASS , PASS , PASS };
```

These contain all the possible outcomes for the dice. By calling this inside the roll and by utilizing the random() we should get a playable random roll.

WRITE UP 4/9:

- Initially I wanted to have 4 separate functions but it did not work because it required global variables. Global variables I was told by TA Eugene and Tutor Tristan were not needed and would likely get messed up in a program where variables are constantly being changed.
- I also added later on a winner checker to check after every round to see who was still technically in the game.
- I also simplified my code quite a bit thanks to the TA sections.
- What worked well was that I understood the program logically and my only real issue was having to learn how to program in C before the deadline.