Project 2
<Snakes and Ladders>

CIS-5
Name: David Seitz
Date: 5/3/20

Introduction

Title: Snakes and Ladders

Snakes and Ladders is an ancient game dating back to at least 200 BC, I selected it because you requested a well known game that has been played for a long time, and dating back to 200 BC, it is perhaps the oldest analog game in the world that has widespread use. Its played on a board with a ten by ten grid. On the board, there are Snakes that connect to tiles lower on the board and Ladders that connect to a higher point on the board. It is a player with up to four players who take turns rolling a six-sided dice to move forward that many spaces. If they land on a Snake they will go back to the previous space that the Snake connects to. If they land on a Ladder they will progress to the higher space that the ladder connects to. The goal is to reach the 100 space first. If a player's role would put them on a space greater than 100, they do not move; they must land exactly on 100.

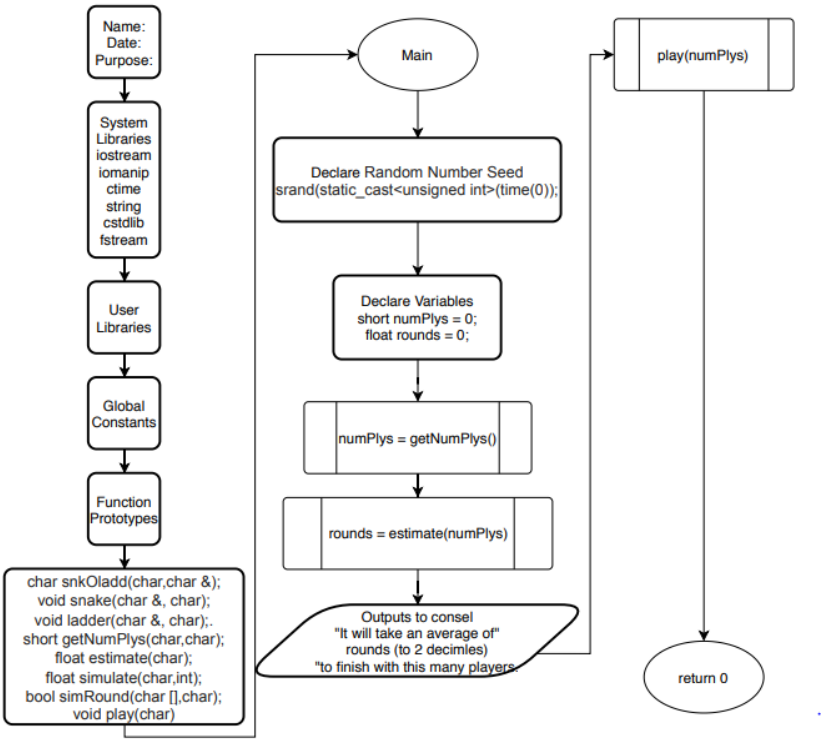
My Approach

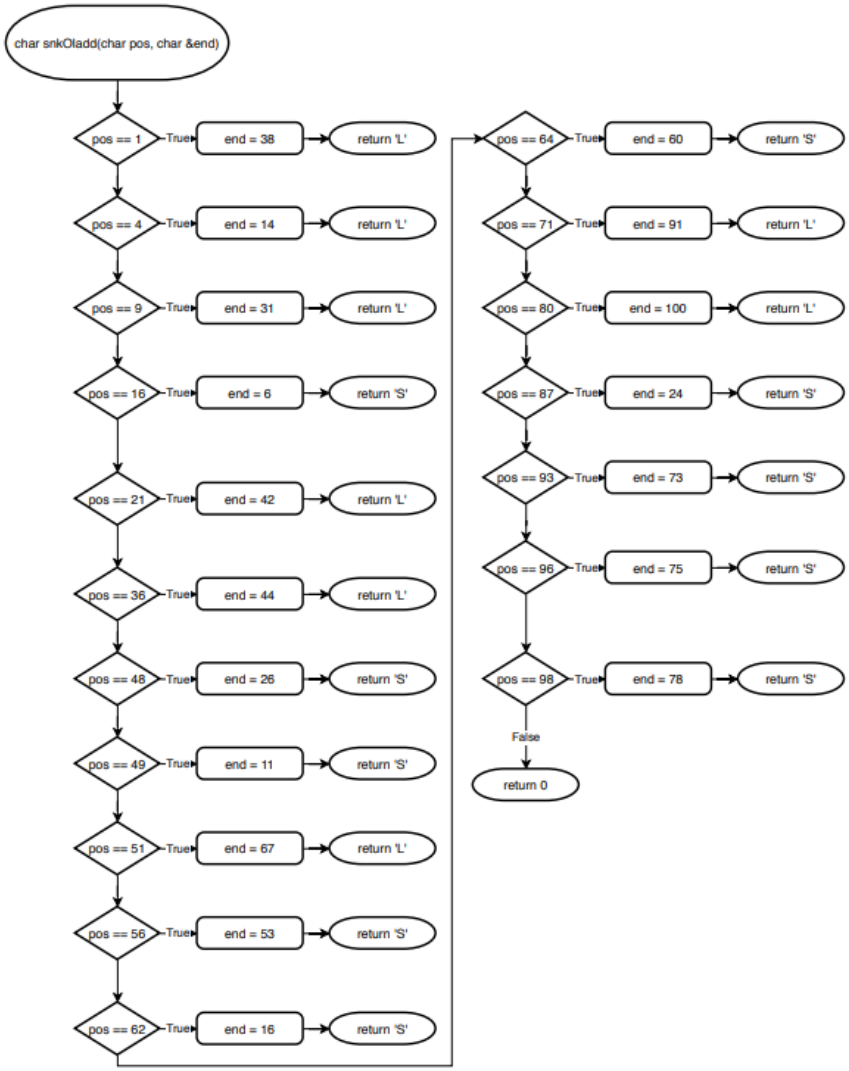
I started this project without use of arrays or functions in the middle of the semester. The midterm version was written to be as compact as possible within the restrictions, and at first it was fitting in in 200 lines because I was already approximating some space saving advantages of arrays using pointers. My problem was that given the simplicity of snakes and ladders, it had no real use of math so I created a game length estimate system as an excuse to use math and floats in the program.

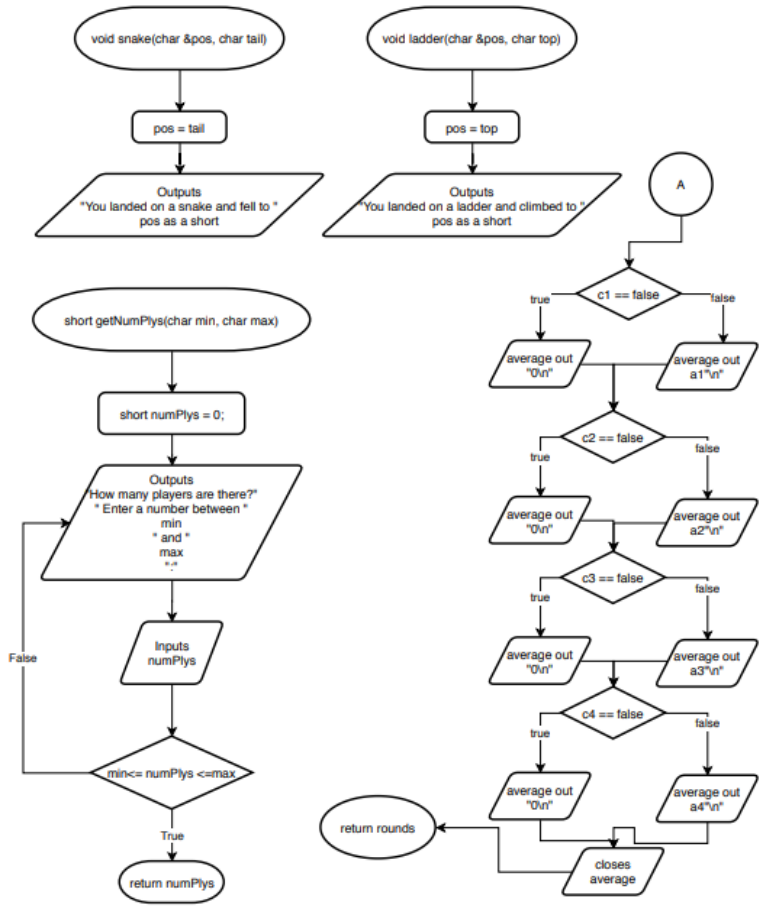
Initially I was possible without an array initially, because the board is more or less irrelevant, the game only needs to know where the players are and if they are on a position that connects to another one. All of the logic for determining if the players did land on a snake would have actually been more complicated if I was having the computer store a board.

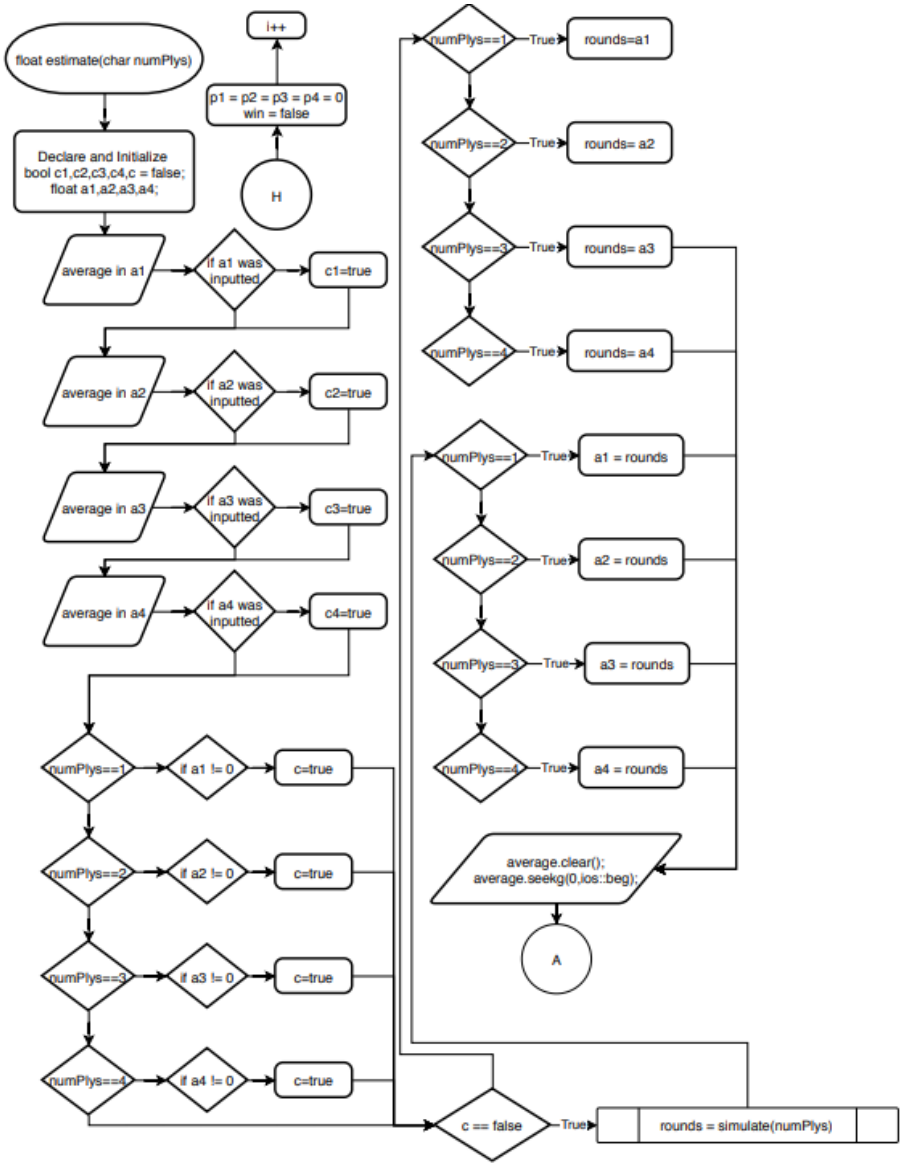
For Project 2, I started by converting my pointer and switch statement pseudo arrays to actual arrays in order to make the code more manageable and shave off the 30 or so lines of code that they added. My next step was to move everything into functions in order to both clean things up, and shave lines off of repeated sections of code. I started this step by dragging the 3 main parts of the program into 3 separate functions (getNumPlys, simulation, play), quickly after that I moved the most egregious portion of repeated code into functions as well (checking if someone landed on a snake or a ladder). After that I made everything that should be a function a function to make things more organized; finally I did a lot of debugging.

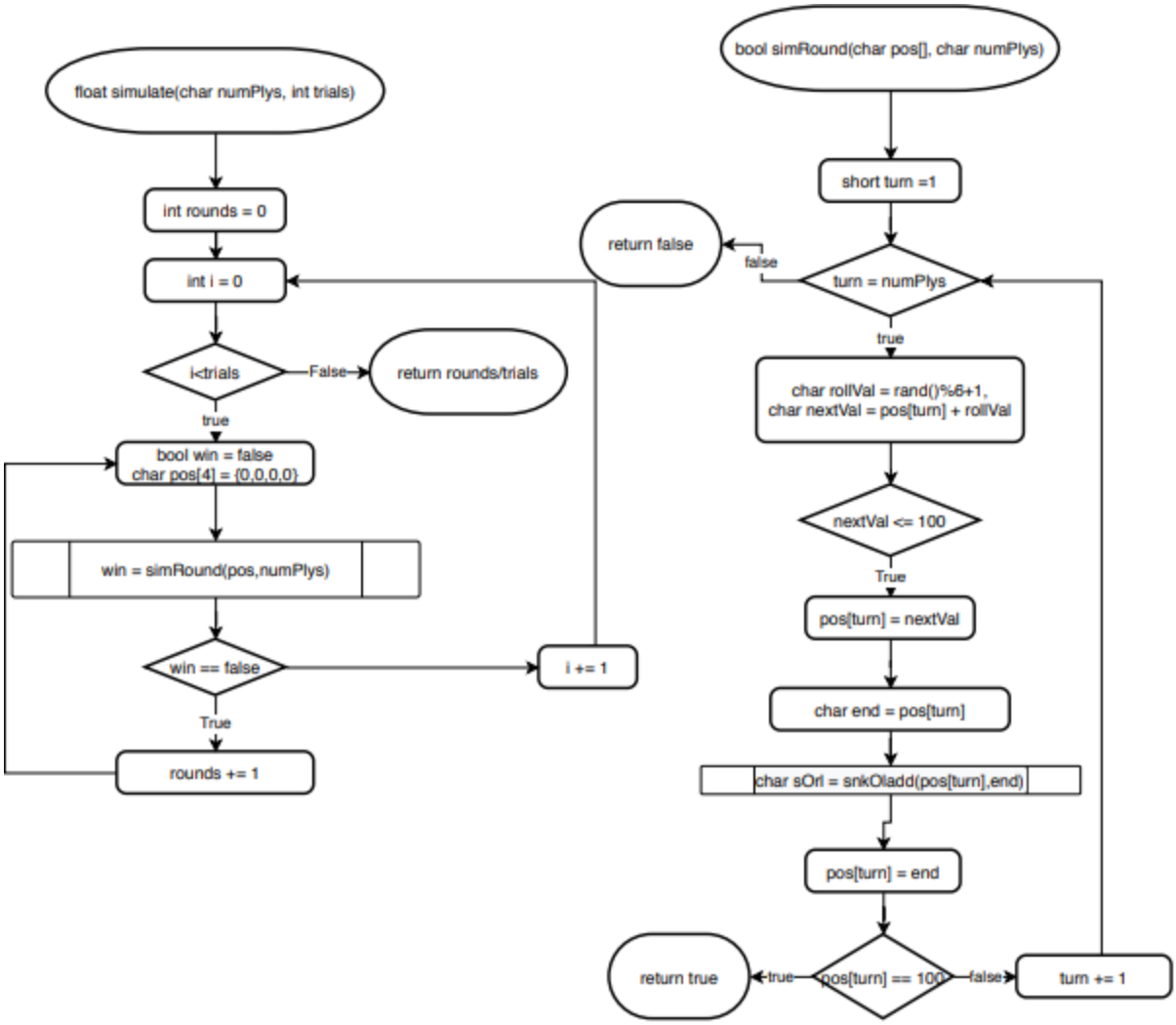
Flowchart

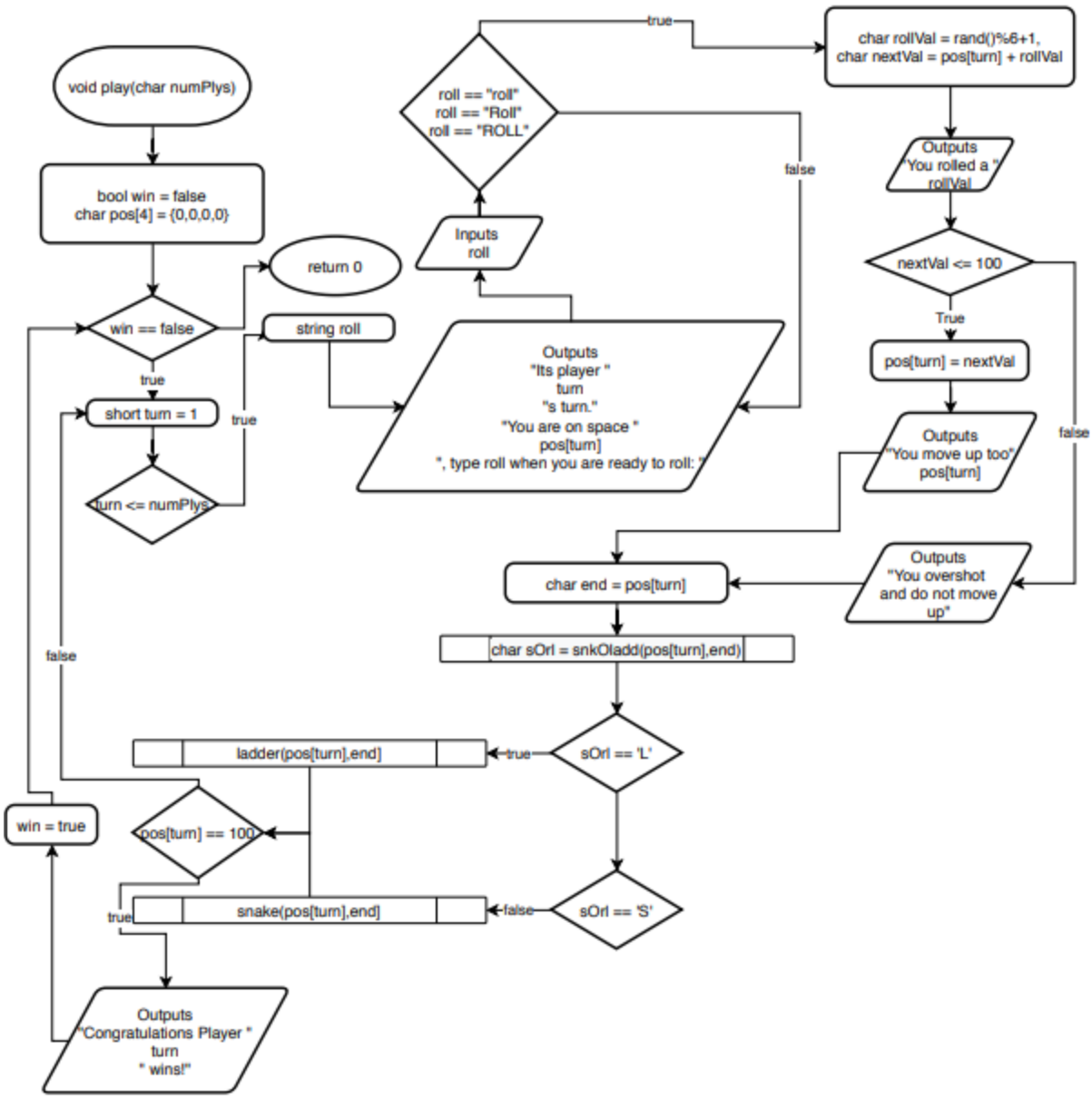

Name:
Date:
Purpose:
System Libraries
iostream
iomanip
ctime
string
cstdlib
fstream
User Libraries
Global Constants
Function Prototypes
char snkOladd(char,char &);
void snake(char &, char);
void ladder(char &, char);.
short getNumPlys(char,char);
float estimate(char);
float simulate(char,int);
bool simRound(char [],char);
void play(char)
Main
Declare Random Number Seed
srand(static_cast<unsigned int>(time(0));
Declare Variables
short numPlys = 0;
float rounds = 0;
numPlys = getNumPlys()
rounds = estimate(numPlys)
Outputs to consel
"It will take an average of"
rounds (to 2 decimles)
"to finish with this many players.
play(numPlys)
return 0


char snkOladd(char pos, char &end)
pos == 1
True
end = 38
return 'L'
pos == 4
True
end = 14
return 'L'
pos == 9
True
end = 31
return 'L'
pos == 16
True
end = 6
return 'S'
pos == 21
True
end = 42
return 'L'
pos == 36
True
end = 44
return 'L'
pos == 48
True
end = 26
return 'S'
pos == 49
True
end = 11
return 'S'
pos == 51
True
end = 67
return 'L'
pos == 56
True
end = 53
return 'S'
pos == 62
True
end = 16
return 'S'
pos == 64
True
end = 60
return 'S'
pos == 71
True
end = 91
return 'L'
pos == 80
True
end = 100
return 'L'
pos == 87
True
end = 24
return 'S'
pos == 93
True
end = 73
return 'S'
pos == 96
True
end = 75
return 'S'
pos == 98
True
end = 78
return 'S'
False
return 0


void snake(char &pos, char tail)
pos = tail
Outputs
"You landed on a snake and fell to "
pos as a short
void ladder(char &pos, char top)
pos = top
Outputs
"You landed on a ladder and climbed to "
pos as a short
A
c1 == false
true
false
average out
"0\n"
average out
a1"\n"
c2 == false
true
false
average out
"0\n"
average out
a2"\n"
c3 == false
true
false
average out
"0\n"
average out
a3"\n"
c4 == false
true
false
average out
"0\n"
average out
a4"\n"
closes
average
return rounds
short getNumPlys(char min, char max)
short numPlys = 0;
Outputs
"How many players are there?"
" Enter a number between "
min
" and "
max
Inputs
numPlys
min<= numPlys <=max
False
True
return numPlys


float estimate(char numPlys)
Declare and Initialize
bool c1,c2,c3,c4,c = false;
float a1,a2,a3,a4;
average in a1
If a1 was inputted
c1=true
average in a2
If a2 was inputted
c2=true
average in a3
If a3 was inputted
c3=true
average in a4
If a4 was inputted
c4=true
numPlys==1
if a1 != 0
c=true
numPlys==2
if a2 != 0
c=true
numPlys==3
if a3 != 0
c=true
numPlys==4
if a4 != 0
c=true
c == false
True
rounds = simulate(numPlys)
p1 = p2 = p3 = p4 = 0
win = false
i++
H
numPlys==1
True
rounds=a1
numPlys==2
True
rounds= a2
numPlys==3
True
rounds= a3
numPlys==4
True
rounds= a4
numPlys==1
True
a1 = rounds
numPlys==2
True
a2 = rounds
numPlys==3
True
a3 = rounds
numPlys==4
True
a4 = rounds
average.clear();
average.seekg(0,ios::beg);
A


float simulate(char numPlys, int trials)
int rounds = 0
int i = 0
i<trials
False
return rounds/trials
true
bool win = false
char pos[4] = {0,0,0,0}
win = simRound(pos,numPlys)
win == false
i += 1
True
rounds += 1
bool simRound(char pos[], char numPlys)
short turn =1
return false
false
turn = numPlys
true
char rollVal = rand()%6+1,
char nextVal = pos[turn] + rollVal
nextVal <= 100
True
pos[turn] = nextVal
char end = pos[turn]
char sOrl = snkOladd(pos[turn],end)
pos[turn] = end
return true
true
pos[turn] == 100
false
turn += 1


void play(char numPlys)
bool win = false
char pos[4] = {0,0,0,0}
win == false
true
short turn = 1
turn <= numPlys
true
return 0
string roll
Outputs
"Its player "
turn
"s turn."
"You are on space "
pos[turn]
", type roll when you are ready to roll: "
Inputs
roll
roll == "roll"
roll == "Roll"
roll == "ROLL"
true
false
char rollVal = rand()%6+1,
char nextVal = pos[turn] + rollVal
Outputs
"You rolled a "
rollVal
nextVal <= 100
True
false
pos[turn] = nextVal
Outputs
"You move up too"
pos[turn]
Outputs
"You overshot
and do not move
up"
char end = pos[turn]
char sOrl = snkOladd(pos[turn],end)
sOrl == 'L'
true
ladder(pos[turn],end)
sOrl == 'S'
false
snake(pos[turn],end)
pos[turn] == 100
true
win = true
false
Outputs
"Congratulations Player "
turn
" wins!"

Checklist

Chapter	Section	Topic	Line
2			
	2	cout	156
	3	libraries	15-20
	6	Integers	46
	7	Characters	242
	8	Strings	286
	9	Floats	47
	10	Bools	165
	15	Comments	56
3			
	5	Type Casting	44
	7	Formatting outputs	52
	8	Strings	286
4			
	2	if	168
	4	else-if	304
	5	nesting	292
	6	if-else-if	304
	8	Logic Operators	174
	11	Validating User Input	155
	13	Conditional Operators	158
	14	Switch	172

5			
	1	Increment/Decrement	240
	2	While	280
	5	Do while	243
	6	For Loop	281
	11	Files I/O Both	163-171, 222-233
6		Functions	
	3	Function Prototypes	30-40
	5	Pass By Value	162
	8	Return	236
	9	Return Boolean	274
	11	Static Variables	
	12	Default Arguments	34
	13	Pass by reference	143
	14	overloading	
	15	exit() function	312
7		Arrays	
	1 to 6	Single Dimensioned Arrays	242
	8	Single Dimensioned as Function argument	254
		Passing Arrays to and from Functions	254

Sample console:

```
How many players are there? Enter a number between 1 and 4: 3
It will take an average of 23.53 turns to finish with this many players.



Its player 1's turn.
You are on space 0, type roll when you are ready to roll: roll
You rolled a 2
You move up too 2


Its player 2's turn.
You are on space 0, type roll when you are ready to roll:
```

This is the first turn. The user inputed that there are 3 players. The first player took their turn by typing roll. The second player's turn started but they did not roll yet.

Program

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */

/*
 * File:   main.cpp
 * Author: David Seitz
 * Created on June 6, 2020, 11:12 AM
 * Purpose: Snake and ladders
 */

//System Level Libraries
#include <iostream>
#include <ctime>
#include <string>
#include <cstdlib>
#include <iomanip>
#include <fstream>

using namespace std; // Libraries compiled under std

//User Level Libraries

//Global Constants - Science/Math Related
//Conversions, Higher Dimensions

//Function Prototypes
char snkOladd(char,char &);//Figures out if the player landed on a snake or a ladder.
void snake(char &, char);//Moves the player down a snake.
void ladder(char &, char);//Moves the player up a ladder.

short getNumPlys(char =1,char=4);//Gets the number of players from the user. Inputs min and
max players.

float estimate(char);//estimates how long the game will last, pulls from a file if availbe.
float simulate(char,int = 100);//Simulates a bunch of games to estimate calculate an estimate.
```

```
bool simRound(char [],char);//Simulates a round.

void play(char);
//Execution Begins Here!
int main(int argc, char** argv) {
   //Random Number Seed Set Here
   srand(static_cast<unsigned int>(30));
   //Variable Declarations
   short numPlys=0;//The number of players in this game.
   float rounds=0;//The average amount of turns the game lasts with that amount of players.
   //Figures out how many players their are
   numPlys = getNumPlys();
   //Figures out the game lenght
   //rounds = estimate(numPlys);
   cout<<setprecision(2)<<fixed<<"It will take an average of "<<rounds<<" turns to finish with
this many players."<<endl<<endl;

   //Gets names

   //The actual game.
   play(numPlys);
   //Clean Up
   //Exit stage right!
   return 0;
}
char snkOladd(char pos, char &end){
   switch (pos){//Figures out if they landed on a shoot or ladder
      case 1:{
         end = 38;
         return 'L';
      }
      case 4:{
         end = 14;
         return 'L';
      }
      case 9:{
         end = 31;
         return 'L';
      }
```

```
case 16:{
    end = 6;
    return 'S';
}
case 21:{
    end = 42;
    return 'L';
}
case 28:{
    end = 84;
    return 'L';
}
case 36:{
    end = 44;
    return 'L';
}
case 48:{
    end = 26;
    return 'S';
}
case 49:{
    end = 11;
    return 'S';
}
case 51:{
    end = 67;
    return 'L';
}
case 56:{
    end = 53;
    return 'S';
}
case 62:{
    end = 16;
    return 'S';
}
case 64:{
    end = 60;
    return 'S';
```

```
        }
        case 71:{
            end = 91;
            return 'L';
        }
        case 80:{
            end = 100;
            return 'L';
        }
        case 87:{
            end = 24;
            return 'S';
        }
        case 93:{
            end = 73;
            return 'S';
        }
        case 96:{
            end = 75;
            return 'S';
        }
        case 98:{
            end = 78;
            return 'S';
        }
    }
    return 0;
}
void snake(char &pos, char tail){
    pos = tail;
    cout<<"You landed on a snake and fell to "<<(short)pos<<endl;
}
void ladder(char &pos, char top){
    pos = top;
    cout<<"You landed on a ladder and climbed to "<<(short)pos<<endl;
}


short getNumPlys(char min , char max ){
```

```
    short numPlys =  0;
    do{//Asks for how many players their are until the user gives a valid input.
        cout<<"How many players are there? Enter a number between "<<(short)min<<" and
"<<(short)max<<": ";
        cin>>numPlys;
    }while(numPlys < min || numPlys > max);
    return numPlys;
}

float estimate(char numPlys){
    fstream average("average.txt");
    float rounds=0;//The average amount of turns the game lasts with that amount of players.
    bool c1,c2,c3,c4,c = false;//If the file reading failed
    c1 = c2 = c3 = c4 = false;
    float a1,a2,a3,a4;//The value in the file
    if(average>>a1)c1=true;//Reads the file for the first value
    if(average>>a2)c2=true;//Reads the file for the secound value
    if(average>>a3)c3=true;//Reads the file for the third value
    if(average>>a4)c4=true;//Reads the file for the forth value
    switch(numPlys){//Figures out which place is
        case 1:
            if (a1 != 0) c=true;
            break;
        case 2:
            if (a2 != 0) c=true;
            break;
        case 3:
            if (a3 != 0) c=true;
            break;
        case 4:
            if (a4 != 0) c=true;
            break;
    }
    if (c==false){//If the number of players does not have an average
        rounds = simulate(numPlys);
        switch(numPlys){//Sets the propper a to rounds
            case 1:
                a1 = rounds;
                break;
```

```
        case 2:
            a2 = rounds;
            break;
        case 3:
            a3 = rounds;
            break;
        case 4:
            a4 = rounds;
            break;
    }
}
else{//If there is a value for that number of players
    switch(numPlys){//Sets rounds to the propper a
        case 1:
            rounds=a1;
            break;
        case 2:
            rounds=a2;
            break;
        case 3:
            rounds=a3;
            break;
        case 4:
            rounds=a4;
            break;
    }
}
average.clear();//Goes back to the start of the file
average.seekg(0, ios::beg);
//Refills the file
c1==false?
    average<<"0"<<endl:
    average<<a1<<endl;
c2==false?
    average<<"0"<<endl:
    average<<a2<<endl;
c3==false?
    average<<"0"<<endl:
    average<<a3<<endl;
```

```
    c4==false?
       average<<"0":
       average<<a4;

    average.close();//Closes the file.
    return rounds;
}
float simulate(char numPlys,int trials){
    int rounds = 0;
    for(int i=0;i<trials;i++){//Plays the game 100 times
       bool win = false;//Wether the game has been won yet.
       char pos[4] = {0,0,0,0};//The location of players
       do{//Goes until someone has won.
          win = simRound(pos,numPlys);//Simulates a round
          rounds ++;//adds to the round counter
       }while(win==false);//if the test game is over
       //Resets the board
       for (char q = 0; q <= 4; q++){pos[q]=0;}
       win = false;
    }
    //Once it has looped through 100 games
    return (float)rounds /trials;//Finds the average amount of turns
}
bool  simRound(char pos[],char numPlys){
    for (short turn = 1; turn<=numPlys;turn++){//Does a turn for each player.
       //Rolls the dice
       char rollVal (rand()%6 +1),//The roll they got
            nextVal = pos[turn] + rollVal ;//The place that roll will put them.
       //Moves the player
       if( nextVal <=100){//If their roll will not put them over 100
          pos[turn] = nextVal;//Moves the player up
       }

       //Checks for a snake or a ladder
       char end = pos[turn];
       char sOrl = snkOladd(pos[turn],end);
       pos[turn] = end;

       //Check if the player wins
```

```
      if (pos[turn] ==100){//Checks if the player landed on 100
         return true;//And sets win to true, ending the loop.
      }
   }
   return false;
}

void play(char numPlys){
   bool win = false;//Wether the game has been won yet.
   char pos[4] = {0,0,0,0};//The location of players
   while(win==false){//Goes until someone has won.
      for (short turn = 1; turn<=numPlys;turn++){//Does a turn for each player.
         cout<<endl<<endl;
         cout<<"Its player "<<turn<<"'s turn."<<endl//Tells people who's turn it is,
            <<"You are on space "<<(short)(pos[turn])<<", type roll when you are ready to roll:
";//where they are, and prompts them to roll.
         {
            string roll;//stores their roll input.
            cin>>roll;
            if (roll == "roll" ||roll == "Roll"||roll=="ROLL"){//If they typed roll
               char rollVal (rand()%6 +1),//The roll they got
                    nextVal = pos[turn] + rollVal ;//The place that roll will put them.
               cout<<"You rolled a "<<(short)rollVal<<endl;//Tells them their roll.
               if( nextVal <=100){//If their roll will not put them over 100
                  pos[turn] = nextVal;//Moves the player up
                  cout<<"You move up too "<<(short)pos[turn]<<endl;//Tells them their new
position.
               }
               else{//If the roll will put the player over 100
                  cout<<"You overshot so you do not move"<<endl;//Tells them they overshot and
they do not advance.
               }
            }
         }
            char end;
            char sOrl = snkOladd(pos[turn],end);
            if (sOrl == 'L') ladder(pos[turn],end);
            else if (sOrl == 'S') snake(pos[turn],end);
```

```
        if (pos[turn] ==100){//Checks if the player landed on 100
            cout<<"Congradulations Player "<<turn<<" wins!!";//If they did tells them they one
            win = true;//And sets win to true, ending the loop.
        }
      }
   };
   exit(0);
}
```