Snakes & Ladders

Cian Herlihy – R00205604

I unfortunately could not complete the command line arguments because I was unaware of how to do it through visual studio code, and I tried through command line too, but I must have been doing it wrong. If I knew how to run the code with arguments, then it would be a simple fix of changing the input of snakes and ladders and getting the value from scanf to argv[1] and argv[2]. Then insert them the same way in the linked list.

I decided to update this game to change a Singly Linked List to a Doubly Linked List. This is to improve the player landing on the Head of a Snake and traverse backwards in the List rather than my previous methods of going back to the start and work my way up to the correct position. This should make the code slightly faster too inherently.

# snakes-and-ladders.c

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Cian Herlihy - R00205604 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Snakes & Ladders Assignment \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <stdlib.h>

#include <stdio.h>      // Import Libraries for Functionality of Code

#include <time.h>

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Structs for Player and nodes in Doubly Linked List \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Structs for Node which stores value (Numbered Square on board) and the Next Value for a linked list

struct node {

    struct node \*prev;          // Pointer to Previous Node (Node Address)

    int value;                  // Nodes Value to Indicate Number on Board

    int player;                 // Player value to Indicate Players Presence

    struct node \*snake;         // Pointer to Tail of Snake (Node Address)

    struct node \*ladder;        // Pointer to Top of Ladder (Node Address)

    struct node \*next;          // Pointer to Next Node (Node Address)

};

typedef struct node node;       // Alternative Variable Names (e.g. node)

struct player                   // Struct not needed if honest but kept this in from an earlier version

{

    int position;               // Update Player Position as the Player Traverses through the Board

};

typedef struct player player\_1; // Alternative Variable Names (e.g. player\_1)

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Random Number Generator \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int random\_gen(int MAX)

{

    int result;

    // Generate Random Number with the Max number being passed through as a parameter

    Random:

    result=rand()%MAX;

    // Jumps back to the label 'Random' when the result is 0. This generates a new number

    if(result==0)

        goto Random;

    else

        return result; // Return Result of Random Number Generator

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Main \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int main(int argc, char\* argv[])

{

    /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

    \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Generate Nodes \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

    \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

                                    Start of Creating Nodes for the Board

                                            --Notes--

                                    '->' are struct pointers

                                    '\*' are regular pointers

                                    '&' is the address of variables

    \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

    \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Generate Nodes \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

    \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

    // Variables for Nodes

    int board\_size, i;

    struct node \*head, \*tail, \*thisNode, \*tempNode;

    srand(time(NULL)); // randomize seed (Better Randomisation in generator. Does not have same effect when in function of the generator)

    // Generate Board Size randomly

    Board:

    board\_size = random\_gen(64);

    // Error Check Board Size to be Between 32 - 64

    if (board\_size > 64 || board\_size < 32) { goto Board; }

    for (i=1; i<=board\_size; i++)   // Loops the Creation of Nodes to the Boards Size that was Randomly Generated Between 32-64

    {

        // Allocate Memory for each node using malloc

        thisNode = (node \*)malloc(sizeof(node));

        thisNode->value = i;                // Assign Value to i for number of square on the board it represents

        thisNode->ladder = 0;               // Assign Default Values to Fields

        thisNode->snake = 0;                // Assign Default Values to Fields

        thisNode->player = 0;               // Assign Default Values to Fields

        thisNode->ladder = thisNode;        // Assign Default Address to Itself

        thisNode->snake = thisNode;         // Assign Default Address to Itself

        // Stores address of first node to head and tempNode

        if (i == 1)

        {

            head = tempNode = thisNode;     // Sets Address of head to First Node

            thisNode->player = 1;           // Sets Player to start on First Node

            thisNode->prev = NULL;          // Sets First Node to have no Previous Node Address

        }

        else

        {

            // Sets Value of thisNode to the &next of tempNode

            tempNode->next = thisNode;      // Sets Next address for the Current Node

            thisNode->prev = tempNode;      // Sets the Previous Address to the Previous Node Address

            tempNode = thisNode;            // Moves on to Next Node Before Returning to Start of Loop

        }

    }

    tempNode->next = NULL;  // Assigns last node \*next pointer to NULL to finish the Linked List

    tail = tempNode;        // Assigns Tail to list for starting at end of list

    tempNode = head;        // Set value of Head to TempNode

    /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

    \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* End Generate Nodes \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

    \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

    // Variables and Arrays

    int snakes, ladders, count=0;

    int snake\_positions[20];

    int ladder\_positions[20];

    // Set all array positions to 100 to avoid mixup. (100 because the board size only goes up to 64)

    for (int i=0; i<20; i++)

    {

        snake\_positions[i] = 100;

        ladder\_positions[i] = 100;

    }

    //Input Snakes and Ladders Count to input to board

    Snakes\_Input:

    printf("How many Snakes would you like on the board? \nMax = 10 >>> ");

    scanf("%d", &snakes);   // I think the input code could be changed to allow for command line arguments by setting the value of snakes to argv[1] and ladders to argv[2]

    // Error Check so User inputs number Between 0 - 10

    if (snakes > 10 || snakes < 0) { goto Snakes\_Input; }   // goto Jumps to Label at Line 133

    //Input Snakes and Ladders Count to input to board

    Ladders\_Input:

    printf("How many Ladders would you like on the board? \nMax = 10 >>> ");

    scanf("%d", &ladders);  // I think the input code could be changed to allow for command line arguments by setting the value of snakes to argv[1] and ladders to argv[2]

    // Error Check so User inputs number Between 0 - 10

    if (ladders > 10 || ladders < 0) { goto Ladders\_Input; }    // goto Jumps to Label at Line 142

    /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

    \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Generate Snakes \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

    \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

    for (int i=0; i<snakes; i++)                        // Limit loop to amount of snakes selected

    {

        int snake\_head, snake\_tail;                     // Variable Names

        Head:

        snake\_head = random\_gen(board\_size-1);          // Generate Snakes Head Position

        Tail:

        snake\_tail = snake\_head - random\_gen(10);       // Generate Tail Position (No More than 10 Spaces away from Snakes Head)

        if (snake\_head < 2){goto Head;}                 // Snakes Head landed on First Node so Jump back to Head Label

        if (snake\_tail < 2){goto Tail;}                 // Snakes Tail landed on First Node so Jump back to Tail Label

        int exists = 0;                                 // Setting Variable to check if it already exists

        for (int x=0; x<20; x++)                        // Loop through array of Snake positions. (Default: Line 122-127)

        {

            if (snake\_positions[x] == snake\_head || snake\_positions[x] == snake\_tail) // Checking Array for the head and tail

            {

                exists = 1; // Found a Snake in the Position Already

                i = i - 1;  // Reduce I so it will keep looping until it successfully generates a Ladder that doesnt exist

                break;      // break loop when it finds a match and go generate a new 1

            }

        }

        if (exists == 0)                            // Checks if it found a match or not for the Snakes Positions

        {

            snake\_positions[count] = snake\_head;    // Adds Snake Head

            snake\_positions[count+1] = snake\_tail;  // Adds Snake Tail

            count = count + 2;                      // Count is for position in array to add snake to

            continue;                               // Continue Loop to Generate next Snake or Exit Loop

        }

    }

    count = 0; // Re-use Variable for Ladders Array

    /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

    \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Generate Ladders \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

    \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

    for (int i=0; i<ladders; i++)                       // Limit loop to amount of snakes selected

    {

        int ladder\_top, ladder\_bottom;                  // Variable Names

        Bottom:

        ladder\_bottom = random\_gen(board\_size-1);       // Generate Ladders Bottom Position

        Top:

        ladder\_top = random\_gen(10) + ladder\_bottom;    // Generate Ladders Top Position (No More than 10 Spaces ahead of Ladders Bottom)

        if (ladder\_top < 2 && ladder\_top >= board\_size){goto Top;}              // Generate Ladder Top Again if on First Node

        if (ladder\_bottom < 2 && ladder\_bottom >= board\_size){goto Bottom;}     // Generate Ladder Bottom Again if on First Node

        int exists = 0;                                 // Setting Variable to check if it already exists

        for (int x=0; x<20; x++)                        // Loop through array of Ladder positions. (Default: Line 122-127)

        {

            if (snake\_positions[x] == ladder\_top || snake\_positions[x] == ladder\_bottom || ladder\_positions[x] == ladder\_top || ladder\_positions[x] == ladder\_bottom)

            {

                exists = 1; // Found a Ladder in the Position Already

                i = i - 1;  // Reduce I so it will keep looping until it successfully generates a Ladder that doesnt exist

                break;      // break loop when it finds a match and go generate a new 1

            }

        }

        if (exists == 0)                                // Checks if it found a match or not for the Snakes Positions

        {

            ladder\_positions[count] = ladder\_bottom;    // Adds Ladder Bottom

            ladder\_positions[count+1] = ladder\_top;     // Adds Ladder Top

            count = count + 2;                          // Count is for position in array to add snake to

            continue;                                   // Continue Loop to Generate next Snake or Exit Loop

        }

    }

    count = 0; // Resetting to 0 for good measure but not necessary

    /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

    \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Place Snakes & Ladders on Board (Nodes) \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

    \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

    struct node \*index, \*climbLadder, \*slippySnake;     // Create variable of type struct node

    index = head;           // Set variable equal to head for navigation of Doubly Linked List

    while (index != NULL)   // Loop through Doubly Linked List

    {

        for (int i=0; i<snakes\*2; i+=2)                 // Loop through list and step up by 2 each iteration

        {

            if (snake\_positions[i] == index->value)     // Find Index in List that matches the Snakes Head position

            {

                slippySnake = index;                    // Copy Address of Head of Snake (Create New Temporary Index)

                int move = index->value - snake\_positions[i+1]; // Calculate Positions to Move Backward

                for (int z=0; z<move; z++)

                {

                    slippySnake = slippySnake->prev;    // Go Backward Through List

                }

                index->snake = slippySnake;             // Insert Address of Snake Tail into the Node

            }

        }

        for (int i=0; i<ladders\*2; i+=2)                // Loop through list and step up by 2 each iteration

        {

            if (ladder\_positions[i] == index->value)    // Find Index in List that matches the Ladder Bottom position

            {

                climbLadder = index;                    // Copy Address of Bottom of Ladder (Create New Temporary Index)

                int move = ladder\_positions[i+1] - index->value; // Calculate Positions to Move Forward

                for (int z=0; z<move; z++)

                {

                    climbLadder = climbLadder->next;    // Go Forward Through List

                }

                index->ladder = climbLadder;            // Set Pointer for Bottom of Ladder to Top of Ladder

            }

        }

        index = index->next;                            // Traverse Doubly Linked List

    }

    /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

    \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Printing Details to Start of File \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

    \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

    player\_1 p1;    //Create player\_1 as p1

    p1.position=0;  //Starting Position set to 0

    // Opening File and printing Board Details before the Game Commences

    FILE \*file\_write = fopen("Snakes\_and\_Ladders--Results--.txt", "w");     // Open File and Overwrite Contents

    fprintf(file\_write, "The Board size is %d.\n", board\_size);             // Write to file the Board Size Generated

    for (int i=0; i<110; i++)

    {

        fprintf(file\_write, "=");               // Print to File 110 \* '=' to form a double line in code

    }

    fprintf(file\_write, "\n");

    fprintf(file\_write, "Snake(s)  |");     // Write Label for Snakes and follow with positions

    for (int i=0; i<snakes\*2; i+=2)             // Loop through all Snakes and Write their Positions to the File

    {

        fprintf(file\_write, " %d - %d |", snake\_positions[i], snake\_positions[i+1]);    // Write Positions to File

    }

    fprintf(file\_write, "\n");                  // Skip to New Line once all Snakes have been added

    fprintf(file\_write, "Ladder(s) |");     // Write Label for Ladders and follow with positions

    for (int i=0; i<ladders\*2; i+=2)            // Loop through all Ladders and Write their Positions to the File

    {

        fprintf(file\_write, " %d - %d |", ladder\_positions[i], ladder\_positions[i+1]);  // Write Positions to File

    }

    fprintf(file\_write, "\n");                  // Skip to New Line once all Laddersg have been added

    for (int i=0; i<110; i++)

    {

        fprintf(file\_write, "=");               // Print to File 110 \* '=' to form a double line in code

    }

    fprintf(file\_write, "\n");                  // Skip to New Line once all Laddersg have been added

    fprintf(file\_write, "Player 1 Starts off at %d.\n", head->value);   // Write Player Position to File

    fclose(file\_write);                                                 // Close File Connection

    /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

    \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* While Loop to start the Game \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

    \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

    while(p1.position != board\_size) // Loop through until Player Struct Reaches Final Node. Could easily be a variable

    {

        // Opens File to write player moves to and Append Text to text file for results of game

        FILE \*file\_write = fopen("Snakes\_and\_Ladders--Results--.txt", "a+");    // Append to the File

        struct node \*index, \*end;                                               // Variable Names for Structs

        index = head;                                                           // Set Index to the start of Doubly Linked List

        /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

        \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Move Player on the Board \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

        \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

        while (index->value != board\_size)                              // Loop through list until end of list

        {

            // Rolling Dice for Player randomly and print to file what the Player Rolled

            int rolled = random\_gen(6);                                 // Randomly Roll Dice

            fprintf(file\_write, "Player 1 Rolled a %d.\n", rolled);     // Write to File what Player just Rolled

            if(index->player == 1)                          // Found Players Position in Linked List

            {

                // Checks if Moving player forward the roll amount will be on the board

                if (index->value + rolled > board\_size)

                {

                    rolled = board\_size - index->value;     // Replace Dice Roll with Max Amount before Player Goes off Board

                }

                index->player = 0;                          // Set to 0 because Player will no longer be there

                for (int i=0; i<rolled; i++)                // Loop through the amount shown on dice Roll

                {

                    index = index->next;                    //Move Forward to Next address in Doubly Linked List

                }

                /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

                \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Check Node if it is at the bottom of a Ladder \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

                \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

                if (index->ladder!=index)

                {

                    fprintf(file\_write, "Landed on Bottom of a Ladder! You Climbed Up from %d ", index->value);

                    index = index->ladder;                          // Goes to Address at Top of the Ladder

                    index->player = 1;                              // Landed on Bottom of Ladder So Player Climbed up to Top of Ladder

                    p1.position = index->value;                     // Updates Player Position in Player Struct

                    fprintf(file\_write, "to %d.\n", index->value);  // Write to File where the Player Move to

                }

                /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

                \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Check Node if it is at the Head of a Snake \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

                \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

                if (index->snake!=index)                                // Checks if Node has a Snakes Head on it

                {

                    fprintf(file\_write, "Landed on the Head of a Snake! You fell from %d ", index->value);

                    index = index->snake;                           // Goes to Address at Tail of Snake

                    index->player = 1;                              // Place Player 1 on Tail of the Snake

                    p1.position = index->value;                     // Updates Player Position in Player Struct

                    fprintf(file\_write, "to %d.\n", index->value);  // Write to the File with New Player Position

                }

                /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

                \*\*\*\*\*\*\* Move Player if No Ladder or Snake. Check if Player is on Final Node \*\*\*\*\*\*\*

                \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

                index->player = 1;              // No Snake or Ladder on this Square

                p1.position = index->value;     // Update Player Struct Position

                if (index->next == NULL)        // Check \*next if NULL for Final Node

                {

                    fprintf(file\_write, "Player Moved to %d\n=========================\nCongratulations! You Won!\n=========================\n\n", index->value);

                    printf("Game Complete\n");  // Print to Console that the Game is Complete

                    fclose(file\_write);         // Close File Connection

                    exit(0);                    //Exits Program before rest of code is run

                }

                else

                {

                    fprintf(file\_write, "Player 1 Position: %d\n\n", index->value); //Print Players new Position

                }

            }

            else

            {

                index = index->next;            // Traverse Nodes in Doubly Linked List

            }

        }

    }

    fclose(file\_write);                         // Close File Connection

    return 0;

}