MIDDLE EAST TECHNICAL UNIVERSITY, NORTHERN CYPRUS CAMPUS
CNG140 C Programming Assignment 3

Date handed out: **13 April 2015 Monday**
Date submission due: **28 April 2015 Tuesday 23:55**

Programming Assignment 3: Snakes and Ladders

Purpose:
The main purpose of this programming assignment is to revise the topics that we have covered so far including arrays, pointers, functions, repetitive statements, conditional statements and fundamentals of C programming.

In this assignment, you are going to implement a game of **Snakes and Ladders** based on the given specifications. This game is an ancient Indian board game and now it is a worldwide classic. It is played between two or more players on a grid board. A number of "ladders" and "snakes" are shown on the board, each connecting two specific board squares. The objective of this game is to navigate one's game piece, according to die rolls, from the start (Bottom Square) to the finish (Top Square), helped or hindered by ladders and snakes respectively[1].

In your implementation you will keep the coordinates of the snakes, ladders and players inside a two dimensional array. This will help you to practice with arrays, pointers and all of the three control constructs (sequence, selection and repetition). Do not try to compile your entire program in one "big bang". Compile it piece by piece. Test each piece that you have compiled to make sure it works correctly before you add the next piece.

Game Specifications:
In this assignment, you are going to implement a game of Snakes and Ladders which will be played between a player (#) and a computer (@). All ladders will be vertical where each step is represented by underscore (_). In addition, there will be two types of snakes: Curly and Straight. These shapes of these snakes are shown below where the upper case letters show their heads and the lower case letters show their tails. Each snake on the board should be represented with a different alphabetic character. For example, if there are two snakes on the board, the first one is represented with A/a and another one is represented with B/b.

Curly Snake	Straight Snake
`Aaa`	`A`
`  a`	`a`
`  aaa`	`a`
	`a`
	`a`
	`a`
	`a`
	`a`

[1] `http://en.wikipedia.org/wiki/Snakes_and_Ladders`

The player should be able to select the size of the game board at the beginning of the game. The board size can be 10x10 and 15x15. The number of "ladders" and "snakes" for each board size is given below:

Board Size	**Ladders**	**Snakes**
10x10	2 ladders with 3 steps 1 ladder with 5 steps	2 curly snakes
15x15	2 ladders with 3 steps 1 ladder with 5 steps 1 ladder with 6 steps	2 curly snakes 1 straight snake

When the player selects the board size, your program should dynamically allocate a memory for the board in the main and then randomly locate the snakes and ladders on the board. See an example board below for the size 10x10. As can also be seen from this example, <u>there should be no overlap between the ladders and the snakes.</u>

100	99	98	97	96	95	94	93	92	91
81	_	83	84	85	_	87	88	89	90
80	_	78	77	76	_	74	73	72	71
61	_	A	a	a	_	67	68	69	70
60	_	58	57	a	55	54	53	52	51
41	_	43	44	a	a	a	48	49	50
40	39	_	37	36	35	34	33	32	31
21	22	_	24	B	b	b	28	29	30
20	19	_	17	16	15	b	13	12	11
1	2	3	4	5	6	b	b	b	10

Player Position: 0 = Computer Position: 0

When the board is ready to play, both the player and the computer will roll one dice and the higher roll goes first. Before each round, the player should decide to whether continue to play or not.

```
Do you want to continue to play (Y/y: Yes, N/n: No):
```

Once the round is started, two dice are rolled for the player and the computer and their positions on the board are changed according to the sum of the dice values. However, if they have the same value on both of the dice, they have to go back based on the dice value. For example, if they have 3 on both of the dice, they have to go 3 steps back. Moreover, if they have 5 and 6 on the two dice, they have extra 3 more steps to go forward (5+6+3=14 steps). <u>Do no forget to show the dice values of the player and the computer at the end of the round.</u> The player and the computer cannot be in the same square. In case of this situation, the one arriving later should go to the previous square.

During their travel on the board:

- If they meet the heads of the snakes, they need go back to the end points of the tails of the snakes. For example, if they meet the Snake A in the given board above, they have to go back to the square 47.
- If they meet the beginning of the ladders, they need go forward to the end points of the ladders. If they meet the ladder starting in the square 18, they should go to the square 43.

For each round you need to clear your screen, and show the current board again with the current position of the player and the computer. See examples below.

100	99	98	97	96	95	94	93	92	91
81	_	83	84	85	_	87	88	89	90
80	_	78	77	76	_	74	73	72	71
61	_	A	a	a	_	67	68	69	70
60	_	58	57	a	55	54	53	52	51
41	_	#	44	a	a	@	48	49	50
40	39	_	37	36	35	34	33	32	31
21	22	_	24	B	b	b	28	29	30
20	19	_	17	16	15	b	13	12	11
1	2	3	4	5	6	b	b	b	10

Player Position: 43 < Computer Position: 47

100	99	98	97	96	95	94	93	92	91
81	_	83	84	85	_	87	88	89	90
80	_	78	77	76	_	74	73	72	71
61	_	A	a	a	_	67	68	69	70
60	_	58	57	a	55	54	53	52	51
#	_	43	44	a	a	a	48	49	@
40	39	_	37	36	35	34	33	32	31
21	22	_	24	B	b	b	28	29	30
20	19	_	17	16	15	b	13	12	11
1	2	3	4	5	6	b	b	b	10

Player Position: 41 < Computer Position: 50

When one of the players complete the board, your program should show the number of rounds and who win the board. See an example below.

```
Congratulations! The player has won ☺ The board is completed in 20 rounds.
```

Programming Requirements:
In order to implement this game you will need to write at least the following functions, but if you need more functions you can add them.

Function	**Explanation**
`initializeBoard`	This function will set the values of each square of the board (1, 2, 3 … so on.)
`initializeSnakesLadders`	This function will randomly locate snakes and ladders on the game board based on the size of the board.
`move`	This function will move a player/a computer on the board based on the dice values.
`step_size`	This function will calculate the step size for a player/a computer based on the dice values. Positive step numbers are for going forward and negative step numbers are for going back.
`locate_Ladder`	This function will be used by the `initializeSnakesLadders` function to locate a ladder.
`locate_CurlySnake`	This function will be used by the `initializeSnakesLadders` function to locate a curly snake.
`locate_StaightSnake`	This function will be used by the `initializeSnakesLadders` function to locate a straight snake.
`check_snakes`	This function will check the square after the move to see whether there is a snake or not. If there is a snake, it returns a step size to change the player/the computer position.
`check_ladders`	This function will check the square after the move to see whether there is a ladder or not. If there is a ladder, it returns a step size to change the player/the computer position.
`display_board`	This function will clear screen and display the current status of the board with the current positions of the player.
`display_results`	This function will display the final results.

Grading Policy:
Your program will be graded as follows:

Grading Point	**Mark (100)**
Main	10
initializeBoard	10
initializeSnakesLadders	10
move	10
step_size	5
locate_Ladder	5
locate_CurlySnake	5
locate_StaightSnake	5
check_snakes	10
check_ladders	10
display_board	5
display_results	5
Code quality (Appropriate comments, variable names, formulation of selection statements and loops, reusability, extensibility etc.)	10

Please make sure that you follow the restrictions for the assignment as follows.

- **You are not allowed to use global variables**.
- Strictly obey the input output format. Do not print extra things.
- You are not allowed to use `goto` statement.
- Name your source file "snakesladders.c"
- Upload only source file. Do not compress it (zip, rar, …)