CSC 362 Programming Assignment #3
Due date: Thursday, October 15

This assignment will test your ability to handle pointers into an array of chars (string). The program itself will implement a variation of the game of chutes and ladders. In this game, two players take turns rolling a 6-sided die and moving their piece on the game board. If they land on a chute, they slide "down" to the end of the chute (thus moving backward) and if they land on a ladder, they climb up the ladder (thus moving forward). In this game, you will have a few differences though as follows.

1. After a player lands on a chute or ladder, it is removed from the board so that it is not used again.
2. Some squares are denoted as "safe havens". Some squares have the player move back to the previous safe haven and some squares have the player move forward to the next safe haven. Once a haven has been landed on, it is removed from the board. So for instance if player 1 moves backward to a haven and then player 2 lands on the same square to "move backward to a safe haven", player 2 moves further back because the nearest haven behind player 2 was removed when player 1 landed on it. If a player is to move backward to the last haven and there are none left, the player moves back to square 0. If the player is to move forward to the next haven and there are none left, the player stays where they are.
3. If a player lands on the other player, the player who just moved is moved back 1 square. Note that in moving back 1 square, if the player lands on a chute, ladder or move forward/backward, that is ignored.

The game board is set up as an array of characters. The characters are as follows:

' ': a normal square, a player who lands here does not move again in this turn
'B': move backward to the nearest preceding safe haven
'F': move forward to the next safe haven
'H': a haven which a player might move to when landing on a 'B' or 'F', once landed on by a 'B' or 'F', remove the 'H' (do not remove the 'H' if landed on normally)
'a' – 'm': a chute, move backward, change this square to ' '
'o' – 'z': a ladder, move forward, change this square to ' '
'n' – is not used

Here is the code to create the board (this is also placed separately on the website as a textfile):

```
char board[100]="  mHk nH l B He Flq p H  hByHlho H B  jr HFB ir j H  F ku gd
H pjB mH x  BF i H  m oB HlHFBhoH BB ";
```

For a chute or ladder, to determine the distance to move, take the lower case letter and subtract 110 from it. For instance, 'm' is stored in ASCII as 109. 109-110 = -1 so 'm' means "move back 1 square". Landing on 'p' would move you forward 2 squares (112 – 110). 'n' is not used because it is ASCII 110.

Now the tricky part of this program. You cannot use array accessing at all. You *must* use pointers to access into board and to change any character in board. The board itself is a string (char[100]) as shown above. You will have two char pointers, *p1 and *p2 which will point to some location within the board. In order to move a player, use pointer arithmetic, for instance `p1=p1+move`. To test where the player has landed, use *p1 as in `if(*p1=='B')…` Note that to avoid a run-time error, make sure board <= p1/p2 < board+1000 before dereferencing (*p1/*p2). p1/p2 can equal board (board[0]) but not board+100 (this would be board[100] which is beyond the bounds of the array).

This assignment will require the following functions (at a minimum):

- main – to declare and initialize the players (p1, p2) and the board, to seed the random number generator, and to loop through the game until a player wins. The loop will iterate while neither player has won the game. A player wins once they reach or exceed the end of the board (board+100). In each move, both players take a turn (see the move function below). After the

turn ends, output the current board (see the output function). After the loop, determine and output who won the game.

- output – this function outputs the current game board setup to a file. Iterate through each location in the board and if a player is at that position, output a 1 or 2 (the player's number), otherwise output what is at that location of the board. Use putc. You must access the board through a pointer. You can control the loop by iterating until you reach the '\0' character at the end of the board string.
- move – this function receives the player pointers (p1 and p2) and the end of the board. To move a player, generate a random number from 1-6 and add this to the pointer. If, after moving, the player is still within the board, test the square that the player is on to see if the player needs to move elsewhere (lands on a chute, ladder, move forward/backward square, or on the other player) and if so, move the player appropriately. Output the result of the player's turn (what the player rolled, if the player landed on a chute, ladder and how far they moved, or a 'B' or 'F', whether there was a collision or not), and where the player is now (the numeric distance from the beginning of the board). NOTE: a player who lands on a chute, ladder, 'B' or 'F' is moved and then can collide with the other player which then causes the player to be moved again. If there is a collision and the player moves back 1 square, do not move the player again even if the player lands on a chute/ladder/B/F. All output from this function goes to the console window using printf. Move should move player 1, output the results, move player 2 and output those results.
- find haven – this function is called if the player lands on a 'B' or 'F'. This function determines the nearest haven to move to, returning the new location (an address) and removing the haven (change the 'H' to '_').
- chute_ladder – this function returns the new location of the player after sliding down a chute or climbing up a ladder. The distance moved is `p1+(int)(*p1-110)`. (or p2 for player 2). This resets the pointer to be the current location plus the distance to move. The (int) ensures that (*p-110) is cast as an int rather than a char. For instance, if p1 is at board+2 then p1 is on an 'm'. This operation computes p1 + (109-110) which is p1 – 1. Return this new address to main so that you can reassign the given pointer to it, as in (for player 1's turn) `p1=move(p1, …);` Before moving p, reset *p to '*' to show that a chute or ladder has been removed.

The following is a partial sample output from my version for the console window. We see players landing on a chute (26), ladder (19) and a 'B' causing the player to move back to square 8.

```
You:  rolled 3  now at 4
Me:   rolled 3  collision! ... moving back one square ... now at 3
You:  rolled 5  now at 9
Me:   rolled 3  now at 6
You:  rolled 4  now at 13
Me:   rolled 3  now at 9
You:  rolled 6 landed on a ladder...moving 3... now at 22
Me:   rolled 2  now at 11
You:  rolled 4 landed on a chute...moving -6... now at 20
Me:   rolled 1 moving backward to haven ...  now at 8
```

The following is an excerpt of the text file created for the moves shown above.

```
 21k nH l B He Flq p H  hByHlho H B  jr HFB ir j H  F ku gd  H pjB mH x  BF i H  m oB HlHFBhoH BB
 mHk2nH1l B He Flq p H  hByHlho H B  jr HFB ir j H  F ku gd  H pjB mH x  BF i H  m oB HlHFBhoH BB
 mHk nH2l B1He Flq p H  hByHlho H B  jr HFB ir j H  F ku gd  H pjB mH x  BF i H  m oB HlHFBhoH BB
 mHk nH l2B He Fl* p1H  hByHlho H B  jr HFB ir j H  F ku gd  H pjB mH x  BF i H  m oB HlHFBhoH BB
 mHk n2 l B He Fl*1p H  *ByHlho H B  jr HFB ir j H  F ku gd  H pjB mH x  BF i H  m oB HlHFBhoH BB
 m2k n_ l B He Fl* p H  1ByHlho H B  jr HFB ir j H  F ku gd  H pjB mH x  BF i H  m oB HlHFBhoH BB
```

Hand in your program and a complete sample output of both the console window and the file for one run.