CSC 362 Programming Assignment #3
Due date: Wednesday, October 19

This assignment will test your ability to handle pointers used to index into a string (array of chars). 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 as follows.

1. After a player lands on a chute or ladder, the chute/ladder is removed so that it is not used again.
2. Some squares are denoted as "safe havens". Some squares indicate that the player will move back to the previous safe haven and move forward to the next safe haven. Once a haven has been landed on, that haven is removed from the board. For instance, if player 1 lands on a square to move back to the previous haven and then player 2 lands on the same square to move backward to the previous 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 a 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 does not move.
3. If a player lands on a square that the other player is currently on, 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 forward/backward move, that is additional move ignored.

The game board is set up as an array of characters. The variable is declared as follows (note that this is placed separately on the website so that you don't have to copy and paste it from this pdf).

```
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 ";
```

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 moved to because of a 'B' or 'F', remove the 'H' (do not remove the 'H' if landed on normally)
'a' – 'm': a chute, move backward (see below), change this square to ' '
'o' – 'z': a ladder, move forward (see below), change this square to ' '
'n' – is not used

To determine the distance to move on a chute or ladder ('a' – 'm', 'o' – 'z'), 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 to access into the board array. You *must* use pointers to access into the board array to determine what a player lands on, and change the board array when you are to remove a chute/ladder ('a' – 'm', 'o' – 'z') or haven

('H'). The board itself is a string (char[100]) as shown above. Use two char pointers, say *p1 and *p2, to store the location of where the two players currently reside on the board. In order to move a player, use pointer arithmetic, for instance if `move` is the amount to move forward, then reset the player's location using `p1=p1+move`. To test the square where the player has landed, use *p1 as in `if(*p1=='B')…` Note that to avoid a run-time error, make sure that your pointer (p1 and p2) is `>= board` and `< board+100`. If you attempt to dereference your pointer which is not pointing within the array's bounds, you get a run-time error.

You are required to break this program into functions. The following are the minimum functions to write. You may have other functions if you find them needed or useful.

- main – declare and initialize the players (the char pointers, say p1, p2) and the board, seed the random number generator; loop through the game until a player wins (loop while neither player has won, or hasn't yet reached the end of the board at board + 100), in the loop: main calls the function move (see below), once for each player, move will output the result of the roll and if the player moves forward or backward and to where, this output goes to the console window; after the turn ends, output the current board to a disk file (see the output function). After the loop (after the game ends), determine and output who won the game.
- output – outputs the current game board setup to a file, iterating through the board string using a loop and putc (**do not** use printf("%s", board);). Output what is at each location of the board, using '1' and '2' for the two players. This will require as you iterate through the array that you test that board location to each player and if the two pointers are equal (the board location and the player), output that player's number. This function should be passed the two players and a pointer to the board plus the FILE variable. Control the loop by testing the pointer into the board to '\0'.
- move – this function receives the player pointers, the player whose turn it is to move, and the end of the board and returns the new pointer value of where the player is moved to. Generate a random number from 1-6 and add this to the proper player's 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. To move based on a chute/ladder or haven, there are separate functions to call as described below. Output the result of the player's turn (the player's number (1 or 2), what the player rolled, if the player landed on a chute, ladder, 'B' or 'F', whether there was a collision or not and where they moved to. 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 but if the player lands on the other play which results in landing on a chute/ladder/'B'/'F', do not move the player again. All output from this function goes to the console window using printf. main will call move twice, once for each player, using notation like this:

```
p1 = move(p1, p2, board, 1);   // player 1's turn
p2 = move(p2, p1, board, 2);   // player 2's turn
```

- findHaven – 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 '_'). This function will be called from move, not main.

- chuteLadder – this function returns the new location of the player moving from a chute or ladder. The distance moved is `p1+(int)(*p1-110);` (or p2 for player 2). 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 returning from this function, set the current character (the square of the chute or ladder) to ‘*’ to indicate it no longer is a chute/ladder. This function will be called from move, not main.

The following is a partial sample output from my version of the program 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.