Assignment #4

Computer Science I – COP 3502

Objectives

1. To learn how to use stacks and queues
2. To practice implementing known algorithms

Problem: WAR: The Card Game(war.c)

"This is a children's game played in many parts of the world. No strategy is involved - simply the ability to recognize which of two cards is higher in rank, and to follow the procedure of the game. … The object of the game is to win all the cards."

At the beginning of the game, the deck is split evenly between two players. For our program, these two hands of 26 will be read in from a file.

Each player turns over a card. For the purposes of the program, player one lays down their card first followed by player two. The player with the highest card wins both cards.

Card rankings: A K Q J 10 9 8 7 6 5 4 3 2

If two players play a card of the same rank, there is a WAR! "The tied cards stay on the table and both players play the next card of their pile face down and then another card face-up. Whoever has the higher of the new face-up cards wins the war and adds all six cards face-down to the bottom of their packet. If the new face-up cards are equal as well, the war continues: each player puts another card face-down and one face-up. The war goes on like this as long as the face-up cards continue to be equal. As soon as they are different the player of the higher card wins all the cards in the war."

"If you don't have enough cards to complete the war, you lose. If neither player has enough cards, the one who runs out first loses. If both run out simultaneously, it's a draw. Example: Players A and B both play sevens, so there is a war. Each player plays a card face down, but this is player B's last card. Player A wins, since player B does not have enough cards to fight the war."

Quoted information is from http://www.pagat.com/war/war.html. You can read their instructions on the card game for more information.

Program Instructions

The starting hands for both players are predetermined by the input file "games.txt". These should be read into a queue for each player. Players will play from the top of their hand and cards obtained in the game should be added to the bottom of their hand.

During play, player one sets their card down first so this will be the new bottommost card for whomever wins that hand. The player with the highest cards wins, unless one of the cards is an Ace (represented by the number 1).

In the event of a war, the program must maintain a stack for the war cards. See the pictures below for more information.

10H
AD
6C
3C

JC
AS
5H
7S

AD
6C
3C

AS
5H
7S
JC
10H

AD
6C
3C

AS
5H
7S
JC
10H

7S
3C
5H
6C
AS
AD

JC
10H
7S
3C
5H
6C
AS
AD

Recommendations

The following structures are recommended, but not required, for your program. You may choose to write different structures for your stacks and queues.

```
struct card {
   int value;
   char suit;
};

struct queue {
   struct card playercards[52];
   int front;
   int numElements;
   int size;
};

struct stack {
   struct card tablecards[52];
   int top;
};
```

Specifications

These specifications must be followed exactly.

Input:

1. The input location will be an input file titles “games.txt”.
2. The first line of the file will be an integer, n, representing the number of games to be played.
3. Following will be 2n lines – each pair of lines representing the hands for the two players. Player one’s hand will be listed first.
4. In each line there will be 26 cards represented by their value and suit
    a. Values: 1 2 3 4 5 6 7 8 9 10 11 12 13
    b. Suits: H D C S

Sample Input File:

```
2
7 D 11 D 13 S 6 H 6 D 13 H 7 H 10 C 12 S 12 H 1 H 12 C 1 C 2 C 4 H 6 S 1 S 9 C 4 C 7 C 8 S 5 S 2 D
11 H 3 H 5 H
9 H 3 D 11 S 10 S 8 D 5 D 4 S 2 H 4 D 3 S 8 H 13 C 8 C 9 D 5 C 13 D 1 D 10 D 3 C 7 S 10 H 6 C 9 S 11
C 2 S 12 D
4 D 2 C 13 S 1 H 3 C 7 D 7 S 13 D 6 D 5 H 9 H 12 H 8 D 5 D 6 C 12 S 10 S 1 C 3 D 9 C 13 C 12 C 2 H 1
D 11 H 5 C
2 S 11 D 9 D 2 D 8 H 5 S 4 C 10 C 8 S 8 C 7 H 11 S 1 S 6 H 13 H 4 S 9 S 10 D 10 H 11 C 4 H 12 D 3 H
7 C 6 S 3 S
```

You can create additional sample files using the provided handgenerator.c.

Output:

1. Output location must be an output file titled “war.out”
2. Print the current game
    a. Game #<num>
3. Print the following game events:
    a. Normal round:
        i. <Player 1’s card> <Player 2’s card>
    b. War:
        i. <Player 1’s card> <Player 2’s card>
        ii. WAR
        iii. <Player 1’s discard> <Player 2’s discard>
        iv. <Player 1’s card> <Player 2’s card>
    c. A player runs out of cards:
        i. Player <1/2> wins!
    d. Both players run out of cards simultaneously:
        i. There was a draw!

Program:

1. You must use two separate queues: one for each players hand
2. You must implement a stack for the cards in the event of a war
3. For each abstract data type, you must write the separate functions that describe its behavior
    a. Add elements
    b. Remove elements
    c. Initialize
    d. isEmpty

You may choose to set the size of the abstract data types at 52 or use dynamic memory allocation as necessary.

Sample Run

Please view the webcourse assignment for the output file that matches the sample input file above.

Deliverables

One source file – *war.c* – is to be submitted over WebCourses.

Restrictions

Your program must be able to be compiled and executed in a standard C Development Environment. Yourprogram should include a header comment with the following information: your name, course number, section number, assignment title, and date. Also, make sure youinclude comments throughout your code describing the major steps in solving the problem.

Grading Details

Your programs will be graded upon the following criteria:

1) Your correctness

2) Your programming styleand use of white space. Even if you have a plan and your program works perfectly, if your programming style is poor or your use of white space is poor, you could get 10% or 15% deducted from your grade.

3) Your comments. Each function must have appropriate pre and post conditions. Internal comments should be used liberally – both in main and in any functions.

4) Compatibility – You must submit C source files that can be compiled and executed in a standard C Development Environment. If your program does not compile, you will get a sizable deduction from your grade.