02322 Machine Oriented programming

Project 2

Card game implemented using linked lists

Project 2 (just like project 1) is a deliverable in this course and is going to contribute to your grade for the course. The course is evaluated as a whole.

In this project we are going to implement in C (using linked lists) a card game called Yukon shown in Figure 1. You'll implement the game using text displayed in the terminal window, but you'll also have the possibility, if the time allows, to create a graphical user interface for the game. The game will be implemented in steps where you'll gradually implement the game, starting with functionality such as shuffling the cards and ending up with a full implementation of the game. In case you want to try out the game and get familiar with the game rules and gameplay, you can see an online implementation of the game here: https://www.icardgames.com/yukon.html

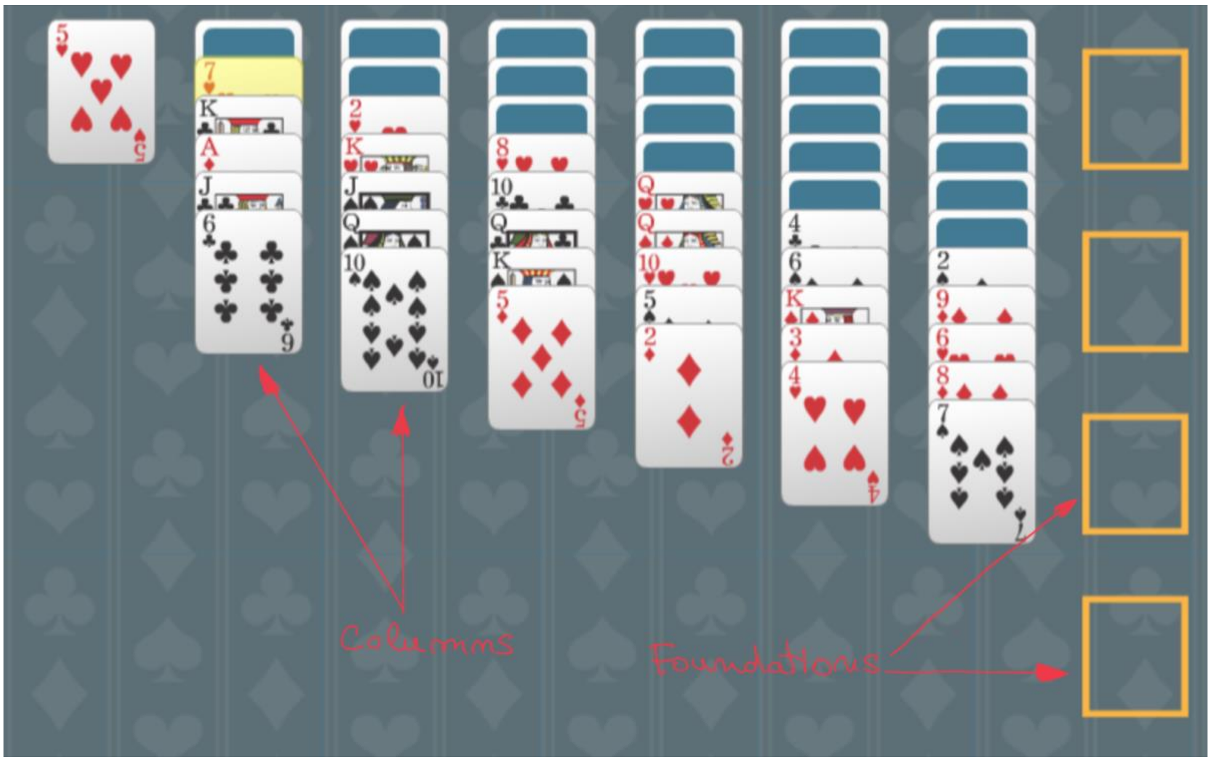


Figure 1 Layout of cards at the start of the Yukon game

Here is a short description of the rules for this game:
To play Yukon Solitaire, the entire deck is dealt into 7 columns of 1, 6, 7, 8, 9, 10, and 11 cards from left to right. 21 of these cards are dealt face down and the rest are dealt face up as shown in Figure 1. Cards must be moved into the four foundations arranged by suit and in order from Ace - King. Cards can be moved between the columns by placing a card (and the eventual cards below it) from a column to a new column, below the bottom card from that column, if that card that has a value one greater than the card that we want to move but is not of the same suit (it can also have the same color but from a different suit). If there are no cards below a hidden card on a column, then the card becomes visible. Once all cards are moved to the foundations, the player wins.

Requirements

The cards from a deck of cards and from columns are stored in linked lists. You are not supposed to use arrays to store the value of the cards.

For displaying the cards as text on the terminal window, each card is represented with the help of 2 characters:

- First character represents the value of the card: A – Ace, numbered cards from 2 up to 9, T - 10, J – Jack, Q – Queen, K-King
- The second character represents the suit of the card: C - clubs (♣), D - diamonds (♦), H - hearts (♥), S – spades (♠)
- If the card is not visible, then we just use parenthesis: []

Here it's shown how the cards from Figure 1 should be represented on the terminal window:

```
C1      C2      C3      C4      C5      C6      C7

5H      []      []      []      []      []      []              []      F1
        7H      []      []      []      []      []
        KC      2H      []      []      []      []              []      F2
        AD      KH      8H      []      []      []
        JC      JS      TC      QH      []      []              []      F3
        6C      QS      QC      QD      4C      []
                TS      KS      TH      6S      2S              []      F4
                        5D      5S      KD      9D
                                2D      3D      6H
                                        4H      8D
                                                7S
LAST Command:
Message:
INPUT >
```

Figure 2 View of the game in the terminal window

Hint: Use tabs to separate the columns. For example: `printf("\tKC\t2H\n")`

Before displaying the cards, we first have a raw with the name of the columns: C1 – C7. The foundations are also labeled with names from F1 to F4.

At the bottom, after all the columns are displayed there should be 3 lines containing the following:

- Last Command – Displays the last command that was inserted at the input prompt
- Message: The message we've got from running the last command. It could be an error message if there was a problem running the last command or OK if the command was executed without error.
- a prompt called "Input >" where we can input new commands.

In the start of the game there is no deck of cards loaded so we'll have the following view:

```
C1      C2      C3      C4      C5      C6      C7

                                                                []      F1

                                                                []      F2

                                                                []      F3

                                                                []      F4
LAST Command:
Message:
INPUT >
```

Figure 3 Initial view

Here are the commands that your game should support:

First, we look at commands that are used before starting the game (let's call it STARTUP phase) which allow us to get a deck of cards, shuffle it, start playing the game and quitting the program.

1. LD <filename>

It allows us to load a deck of cards. <filename> is the name of the file containing a deck of cards. If no filename is given then we are going to load a new unshuffled deck of cards (first we get all the clubs from Ace (A) to King (K), then we get all the diamonds from A to K, then all the hearts from A to K and finally all the spades from A to K). If no filename is specified, then the message should be OK. If the filename is not valid then we should get an error message saying that the file does not exist. If the file name is valid then the program should also validate the input from the file (it should check that the values in the file correspond to correct cards and that we have the right number of cards: 13 cards per suit from A to K and 4 suits summing up to 52 cards). If the file content is OK then the message returned should be OK else there should be a message detailing the first error found in the file (with line number). The file used to load the deck of cards stores one card per line as in this example:

```
5D
5S
KD
9D
....
```

```
C1    C2    C3    C4    C5    C6    C7

[]    []    []    []    []    []    []          []    F1
[]    []    []    []    []    []    []
[]    []    []    []    []    []    []          []    F2
[]    []    []    []    []    []    []
[]    []    []    []    []    []    []          []    F3
[]    []    []    []    []    []    []
[]    []    []    []    []    []    []          []    F4
[]    []    []

LAST Command:LD
Message:OK
INPUT >
```

Figure 4 Game view after a new deck of cards was loaded (value of the cards is hidden)

2. SW

It shows all the cards on the terminal in the order they are placed in the deck. For example, if we have a new unshuffled deck of cards is should show:

```
C1    C2    C3    C4    C5    C6    C7

AC    2C    3C    4C    5C    6C    7C          []    F1
8C    9C    TC    JC    QC    KC    AD
2D    3D    4D    5D    6D    7D    8D          []    F2
9D    TD    JD    QD    KD    AH    2H
3H    4H    5H    6H    7H    8H    9H          []    F3
TH    JH    QH    KH    AS    2S    3S
4S    5S    6S    7S    8S    9S    TS          []    F4
JS    QS    KS

LAST Command:SW
Message:OK
INPUT >
```

Figure 5 Showing an unshuffled deck of cards

It should return an error message if no deck of cards is loaded and OK otherwise.

3. SI <split>

This command shuffles the cards in an interleaved manner. It first splits the deck in 2 piles of cards. If the optional parameter <split> is specified, then we put the number of cards from the top of the deck, specified by this parameter, into one pile and the rest into the second pile. The parameter <split> should be a positive integer smaller than the number of cards in the deck. If the parameter <split> is not specified, then we use a random value (smaller than the number of cards in the deck) for this parameter and use it to split the deck in two.
Then we make a third pile, let's call it the shuffled pile, where we first add the top card from the first pile, next we add the top card from the second pile, next the top card from the first pile, next the top card from the second pile and keep repeating until we finish all the cards from one of the two piles. The remaining cards from the other pile we add to the bottom of the shuffled pile. The shuffled pile becomes now our current card deck.

4. SR

This command shuffles the cards in a random manner. Let's consider our deck of cards as the unshuffled pile. We remove the top card from the unshuffled pile, and we add it in a random position in a new pile called the shuffled pile. We keep doing this until we finish all the cards from our pile. Now the shuffled pile becomes our current card deck.

Note: At first there is only one position to put the card in the new pile, as the new pile is empty at the beginning. Second time, there will be two possible positions, before or after the first card. Third time, there are three possible position; on top of the pile, in the bottom of the pile or in between. Every time there will be one more position to add the card. This continues until original pile is empty.

5. SD <filename>

Saves the cards from the current card deck to a file specified by the parameter <filename>. If parameter <filename> is not specified, then use the default filename "cards.txt".
The file stores the cards, one card per line as in this example:

```
5D
5S
KD
```
….

6. QQ

Command that quits the program. The program exits.

7. P

This command starts a game using the current card deck. It puts the game in the PLAY phase. The cards are dealt from left to right (from column 1 to column 7), one row at a time. All the previous commands that are available for the STARTUP phase are not available in the PLAY phase and the user should get an error message “Command not available in the PLAY phase”. In Figure 6 you can see what we get at the start of a game when using a new deck with unshuffled cards (the card deck shown in Figure 5).

```
C1    C2    C3    C4    C5    C6    C7

AC    []    []    []    []    []    []          []    F1
      8C    []    []    []    []    []
      AD    2D    []    []    []    []          []    F2
      7D    8D    9D    []    []    []
      KD    AH    2H    3H    []    []          []    F3
      6H    7H    8H    9H    TH    []
            QH    KH    AS    2S    3S          []    F4
                  4S    5S    6S    7S
                        8S    9S    TS
                              JS    QS
                                    KS

LAST Command:P
Message: OK
INPUT >
```

Figure 6 Cards at the start of the game if an unshuffled card deck is used

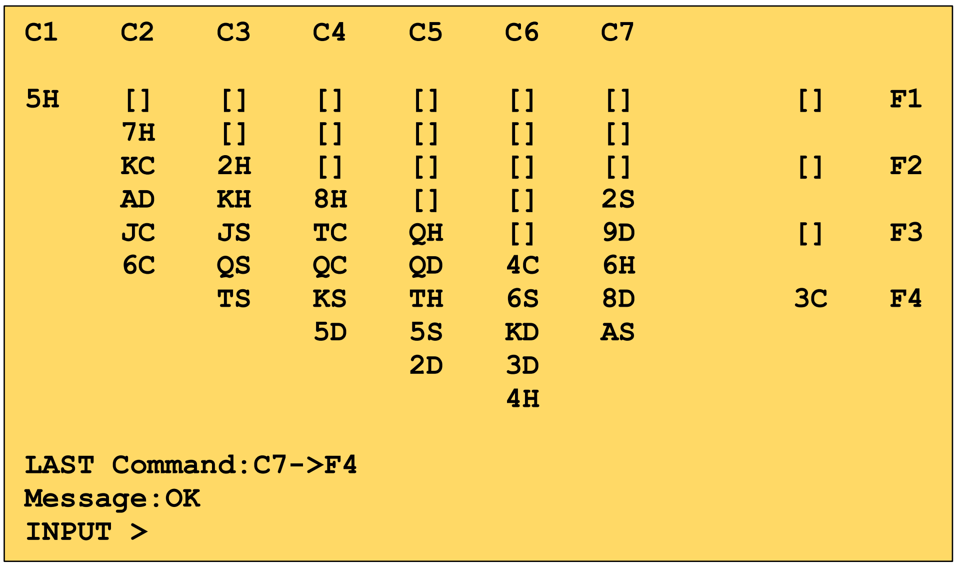
8. Q

Command that quits the current game and goes back to the STARTUP phase. The memory still contains the deck of cards used to play the game that we are quitting. So, if we use the command P again after Q, we basically restart the last game.

9. <Game Moves>

The general format for move commands is: <from> -> <to> where:

- <from> is the card that should be moved. It can be either:
    - a card from a column which is identified first with column number and then with the card number with a “:” between them <from>=<column>:<card>. For example, if we are referring to the card 4 of Hearts from the column C6, then <from> will have the value C6:4H.
    - the bottom card from a column. If we only mention the column number, then the bottom card from that column is selected. In this case <from>=<column>
    - the top card from a foundation. It’s identified by the foundation number. For example, if we need to move the top car from foundation 3 then <from> will have the value F3
- <to> is the destination of the card. It can be either:
    - A column. The card will be placed in the bottom of the column. If we want to place the card in the bottom of column 4 then <to> will have the value C4.
    - A foundation. The card will be placed on a foundation. If we want to place the card on the top of foundation 2 then <to> will have the value F2.



```
C1    C2    C3    C4    C5    C6    C7

5H    []    []    []    []    []    []                []    F1
      7H    []    []    []    []    []
      KC    2H    []    []    []    []                []    F2
      AD    KH    8H    []    []    2S
      JC    JS    TC    QH    []    9D                []    F3
      6C    QS    QC    QD    4C    6H
            TS    KS    TH    6S    8D                3C    F4
                  5D    5S    KD    AS
                        2D    3D
                              4H

LAST Command:C7->F4
Message:OK
INPUT >
```

Figure 7 Game view while playing

We refer to Figure 7 as the start point for each one of these few move examples:

- C6:4H->C4 moves the 4 of hearts from column 6 to the bottom of column 4. Alternatively, we could have used the command C6->C4. After this the column 4 will have the following cards: [],[],[], 8H, TC, QC, KS, 5D, 4H
- C6:4C->C4 moves the 4 of clubs and all the cards below it from column 6 to the bottom of column 4. After this the column 4 will have the following cards: [],[],[], 8H, TC, QC, KS, 5D, 4C, 6S, KD, 3D, 4H
- C7:AS->F2 moves the ace of spades to the foundation 2. Alternatively, we could have used the command C7->F2
- F4->C6 moves the card on the top of foundation 4, in our case the 3 of clubs, to column 6.

The moves should not be performed unless they are valid. Here are a few rules to check for validity:

- The card that we are trying to move should actually exist. For example, if there is no 4 of hearts in column 2, we cannot use C2:4H as the <from> parameter. If we do, we should get an error message. Likewise, we should get an error message if we try to move a card from an empty foundation or column.
- You can move a card to a column only if the card on the bottom of that column is 1 higher than the card we want to move and from a different suit (it's OK if they are the same color as long as they are from a different suit as for example hearts and diamonds).
- If we want to put a card on a foundation, then it should come from the bottom of a column (it cannot have other cards below it). It can be placed on a foundation only if the existing card on top of the foundation is 1 smaller than the card we want to place and from the same suit.
- Only the card from the top of a foundation can be moved. It can be placed on a column if the value on the bottom of that column is 1 bigger and from a different suit.

If the move is valid then the message should be OK otherwise you should return a message that says the move is not valid.

Extensions

If the time allows it, you are welcome to implement one or more of the following extensions for extra credit:

- Validate the input. Make the program robust enough to handle any kind of input you use for the INPUT prompt. The program should not crash even if you write incorrect commands. In that case it should just return error messages.

- Implement extra instructions:

10. U

Undo the last move. You should be able to undo as many moves as you want until the start of the game.

11. R

Redo the previous move. It only works if you've run an undo command before.

12. S <filename>

Save the current state of the game to a file specified by parameter <filename>. It's up to you to choose the file format of the file.

13. L <filename>

Load a game from a file. Restores the game to a previous state that it had when it was saved with the S command.

- Another possible extension is to create a graphical user interface (GUI) for the game with the help of the SDL library. This is a rather advanced implementation, and you should only attempt it if you are ready with the other requirements.

  These links should get you started with SDL:
  - Introductory video for using SDL in C: https://youtu.be/yFLa3ln16w0
  - The official website for SDL: http://libsdl.org
  - SDL tutorials: https://lazyfoo.net/tutorials/SDL/