Hacettepe University
Department of Computer Science & Engineering
BiL203 Programming Laboratory
1st Experiment

Subject : Arrays
Submission Date : 10.10.2013
Final Deadline : 31.10.2013 - **17:00**
Programming Language : C++
Development Environment : Visual Studio 2010 - 2012
Advisors : Dr. Mustafa EGE, Dr. Sevil ŞEN, R.A. Cumhur Yiğit ÖZCAN

Introduction

Subject of this experiment is usage of arrays as data structure. Since you are not allowed to use vectors or any other dynamic data structure, you will need to use arrays effectively. Theme of the experiment is something that you are, hopefully, very familiar with : ***Solitaire*** (http://en.wikipedia.org/wiki/Solitaire). Solitaire is one of the default card games that are shipped with most of the Windows versions.

In this experiment, you are expected to implement an application that is a Solitaire game played via file I/O.

Problem

There are a number of different Solitaire games each of which comes with a different set of rules. This experiment is based on a classic version of the game, **Klondike Solitaire**, which also is the default version included in Windows (http://en.wikipedia.org/wiki/Klondike_solitaire). This sheet will focus on defining how to implement the game using file I/O, therefore please refer to the link provided above if you are not familiar with the game rules. Nevertheless, some terms that are used to define different sections of the game board are given below (see Fig. 1).

Terminology

Tableau Area: This is the largest part of the game board which contains 7 slots for piles. At the starting of the game, a total of 28 cards are dealt on the piles located on this area.
Pile (Tableau Pile): One of the 7 groups of cards that are stacked on each other downwards. Initially, a pile contains a number of ([0 - 6]) closed cards and an opened card at the topmost of the pile.
Stock: After the initial dealing on the tableau area, 24 remaining cards (closed) are placed on this section that is located on the upper left corner of the board.
Waste: The waste consists of 3 slots that will hold the cards that are taken from the stock. Initially the slots in this section are left blank.
Suit: A suit is one of the four categories into which the cards are divided. There are four different suits; Hearts (♥), Diamonds (♦), Spades (♣) and Clubs (♠).

Foundation: A foundation is a slot for holding cards from a particular suit. The cards stored on a foundation should be stacked on each other in a way that every card should be located on the card whose number is exactly one lower than itself. And of course, the initial card to be stored on the foundation should be the ace (A) of the suit.
Foundation Area: The upper right corner of the board which consists of 4 foundations.

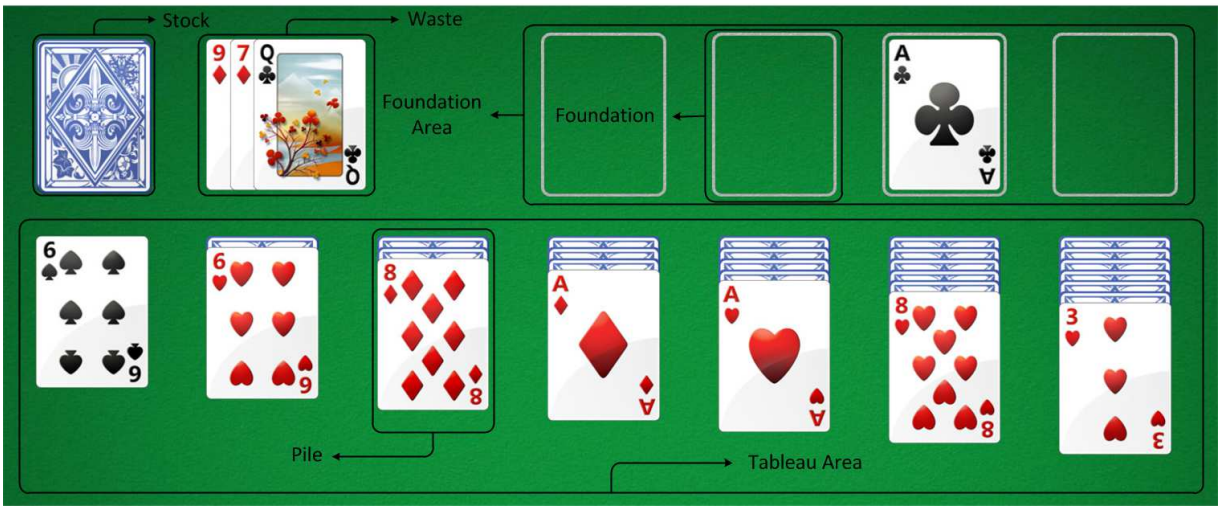

Fig. 1 : Klondike Solitaire Terminology

Representation of the Game Board

As the game will be played via file I/O, you will need to print an output that represents the current state of the game board repeatedly. The cards will be represented by a capital letter; H for Hearts, D for Diamonds, S for Spades, C for Clubs, and a 2-digit-number from 1 (for the ace) to 13 (for the King). Thus, a card representation will hold 3 cells. On the other hand, a closed card on the board will be represented by 3 'at' (@) characters. Finally, an empty space on the

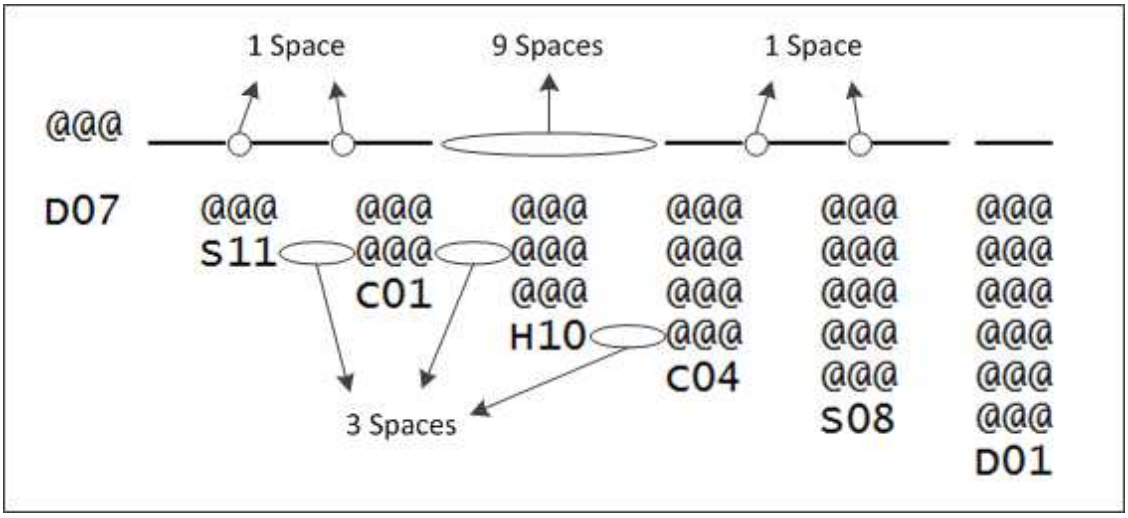

Fig. 2 : Representation of the game board

board will be represented by 3 'underscore' (_) characters. In the picture below, you can see the representation of the board at the starting of a game. There will be 1 space between stock and each waste slot, 9 spaces between waste area and foundation area, 1 space between each foundation slots and 3 spaces between piles (see Fig. 2).

Execution of the Program and I/O Files

The program will need to take 3 arguments from the command line to work properly that are the paths of; deck, commands and output files.

Deck: The deck file will contain the initial state of the cards (one for each line) in the deck. You should use an array of 52 elements to store the data from deck file. The **last** card in the deck file will be the topmost card of the deck, assuming that you hold the deck with the front sides of the cards look **downwards** (see Fig. 3).

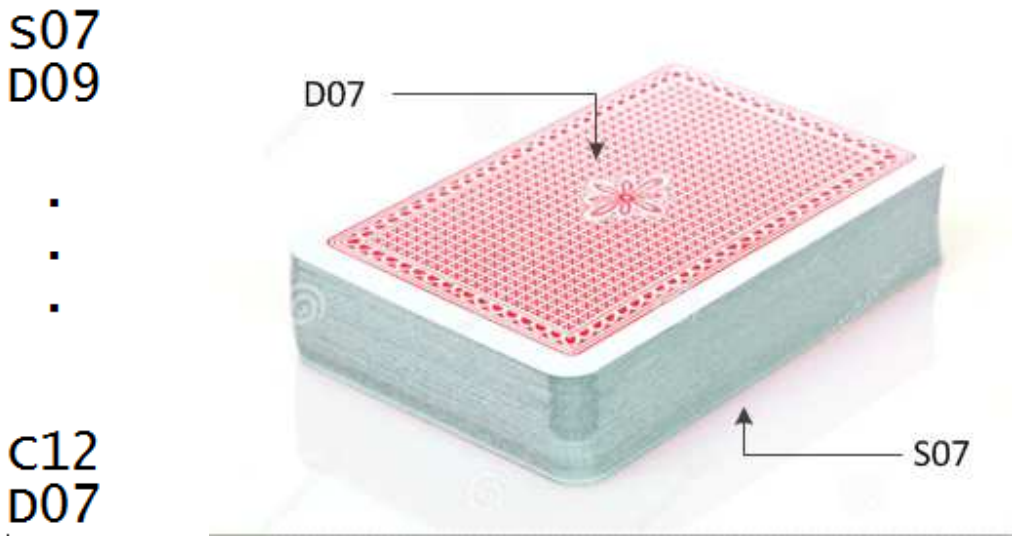

Fig. 3 : Deck

Commands: The commands file will contain the commands (one for each line) each of which represents a move to be made on the game. There are 5 main command types and some of these types may have different variations depending on the parameters. Please see the 'Command List' section for details.

Output: The output file will be created by your program each time it starts. You should print an output of the board for each step of the game (including the initial state after dealing the cards). Furthermore, you should also print the command given at each step and -if required- an error message. If the user successfully completes the game (by stacking all of the 52 cards on their corresponding foundation), you will need to print "You Win!" and then "Game Over!" to the output.

Command List

1) open from stock

This command will open 3 (or less if there are less than 3 cards left in the stock) cards from the stock and place them over the waste. If there are no more cards in the stock (0 cards), this command will gather the cards from the waste area and place them back on the stock. Make sure you place the cards back into the stock in the correct order (check the example I/O files given), and check a few samples from the Internet if you are not familiar with this rule since the order of the cards in the stock is extremely crucial. This part is actually the place where you will show that you are able to use the arrays the way you need, which is the main goal of this experiment.

2) move to foundation [pile <pile_num>] [waste]

This command will move one card either from the top of the given pile or from the waste to the corresponding foundation. Each of the foundations in the foundation area are specifically reserved for one suit. First one is for hearts, second one is for diamonds, third one is for spades and finally the fourth one is for clubs. Make sure you follow this criteria while developing the game.

3) move [pile <source_pile_num> <source_pile_card_index>] <destination_pile_num>
[waste] <destination_pile_num>
[foundation <source_foundation_num>] <destination_pile_num>

The move command has 3 different versions each of which has a different source but all of them finally places the card(s) on a pile. The 'pile' version is for moving a card or a building of cards from a pile to another pile. Source pile number represents the source pile and the destination pile number represents the destination pile. Since more than one card can be taken from a pile to be placed on another pile, a source pile card index parameter is required. For this parameter, 0 indicates only the topmost card in the pile will be moved, 1 indicates first 2 cards and so on...

'Waste' version of the command indicates that the topmost (or rightmost) card of the waste will be moved on the destination pile whose number will be given by the destination pile number parameter.

'Foundation' version of the command is for moving the topmost card from the foundation whose number is given by the source foundation number parameter and placing it on the destination pile.

4) open <pile_num>

The open command is used to open a closed card on the pile. To open a card on the pile, it should be the topmost card of that pile (and of course, it should be closed).

5) exit

Prints "Game Over!" and terminates the execution of the program.

Error Handling

The commands in the command file will fit the command definitions. So you will not need to check for errors in the commands (such as misspelled words or missing parameters etc.)

However, the intended move may not be executable at all times. The command may try to open a card on a pile where the topmost card is already opened, try to move a card to foundation while its predecessor is not the topmost card of the foundation, try to move a card on the piles where the move does not fit the building rules of the solitaire etc. For any of the commands that you cannot execute successfully, you should print an error message ("Invalid Move!") and keep the board as unaffected by the command.

Design Notes

* The file that contains your main method should be named "Main.cpp".

* You are not allowed to use vectors or any other types of dynamic data structures. You should implement the software only by using arrays.

* Although this is not an "Object Oriented Programming" experiment (i.e. you will not need to make use of OOP paradigms such as inheritance, polymorphism, method overriding/overloading etc.) you should still implement a meaningful class diagram and provide it in your report.

* The input and output file paths are NOT constant. They will be given as program arguments from the command line. Argument ordering will be as follows:

<deck_file> <commands_file> <output_file>

* Samples of input and output files can be found on ftp. Examine them carefully before you start coding and ask any question you have via the communication channels mentioned at the *notes and restrictions* section.

Report

Your final report should contain these topics:

1. Cover Page
2. Software Design Notes
    - 2.1. Problem: Re-define the experiment in your own words. Keep the definition short by mentioning only the important parts. **DO NOT COPY PASTE FROM THIS DOCUMENT**.
    - 2.2. Solution:
        - 2.2.1. Explain how you approached the problem, what was the most important part of the problem from your perspective and how it effected your design. And then add anything else that you think will further explain your design.
        - 2.2.2. Provide a class diagram at the solution section. Below the diagram, give an explanation for each class's role in the design. Use only 1 - 2 sentences for each class. If you have more to say, you should say it at the first part of the solution section (2.2.1). If your class diagram is too big to be put in the report file, submit it as a *jpg* file with the report.
        - 2.2.3. Explain in what parts of the program you have used the arrays as data structure. Where did you use arrays and for what purpose? What is the cost for adding or removing an element from those arrays? If you do any other operation than adding or removing an element, explain those operations.

Notes and Restrictions

1) The output of your program will be graded automatically. Therefore, any difference of the output (even a smallest difference) from the sample output will cause an error and you will get 0 from execution. Keep in mind that a program that does not work %100 right is a program that works wrong.

2) Do not submit any file via e-mail. You should upload your files via "Online Experiment Submission System" which is located at http://submit.cs.hacettepe.edu.tr

3) Final Submission Format:

```
<student_number>.zip
        |---  src
              |--- Main.cpp
              |--- *.h
              |--- *.cpp
        |--- report
              |--- report.pdf
              |--- *.jpg (optional)
```

4) You should submit only source files (*.cpp and *.h); not the compiled executable.

5) Your work should obey general software engineering principles and conventions (Design, Comments, Indentation, Variable/Class/Function Naming etc.).

6) Save all your work until the assignment is graded.

7) This is not a team project. You are expected to provide an INDIVIDUAL work.

8) You should follow announcements on "Dersler.Bil203" newsgroup which is located at http://news.cs.hacettepe.edu.tr. You will be held responsible for any announcement and explanation made by the advisors. (Will be replaced by Moodle that is located at http://dersler.cs.hacettepe.edu.tr)

9) The facebook group located at http://www.facebook.com/groups/bbmnewsgrup is an alternate communication channel.

10) Late submissions will be graded over 85 for the first 24 hours and over 70 for submissions between 24 - 48 hours late.