ComS 327 Project Part 1

Solitaire: read input files

Spring 2020

1 Summary

The semester-long project will be based on the card game Solitaire, also known as Klondike. For the first part, you must read an input file that describes a game configuration (possibly in the middle of a game), make sure the input file is valid, and display summary information about the input file. In later parts of the project, you will be required to do *much* more processing of the input file, and will likely need to read the game configuration into an appropriate data structure, so plan accordingly. For this part of the project, all source code must be C.

Specifically, your `Makefile` must build an executable named `check`. When run, `check` should read an input file and write summary information to standard output. If the input file is *invalid*, then display an appropriate error message to standard error, that indicates why the input file is invalid. Otherwise, if the input file is *valid*, then the following should be written to standard output:

```
Input file is valid
C covered cards
S stock cards
W waste cards
```

In the above, `C` indicates the total number of hidden cards (needed to be turned over before the game can be won), `S` indicates the number of remaining stock cards, and `W` indicates the number of waste cards (see Section 2 below). Example input files and their associated outputs are given in Section 6.

Your `check` executable should accept a filename as the first and only command-line argument. If no filename is specified, then `check` should read from standard input. Thus, the following commands should be equivalent, assuming file `infile.txt` is present:

```
./check infile.txt
./check < infile.txt
cat infile.txt | ./check
```

2 Game overview

Figure 1 shows a screenshot of an online Solitaire game (this is Google's, which you can play for free). The game uses a standard 52-card deck, and the game uses the following terminology.

- The *tableau* is the main playing area, comprising seven columns of cards. In the project specs, these will be numbered from left to right as columns 1 through 7. Each column contains zero or more covered cards (face down), followed by zero or more face up cards that must form a "pile", where cards in a tableau pile are subject to the rule: a card may only be placed on a card with rank one higher, and opposite color. Additionally, if there are any covered cards, then they must be followed by at least one uncovered card. Kings may be played on any empty column. In Figure 1, column 1 contains a pile of four cards, column 2 is empty, column 3 contains two covered cards and a pile of three cards, and so on. Note there are a total of 15 covered cards in the tableau for this example.

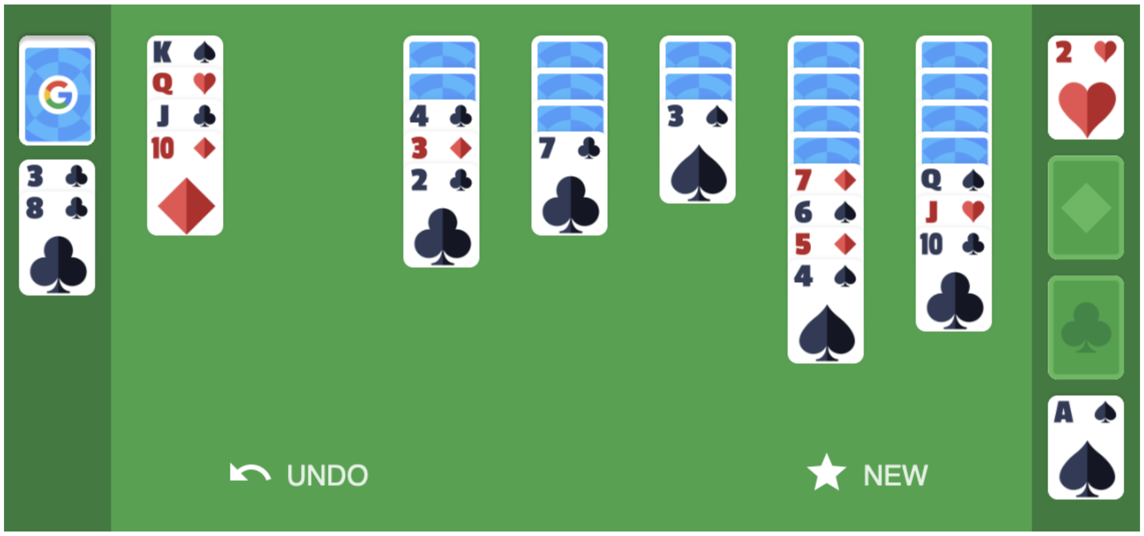

Figure 1: Google's Klondike Solitaire (search in Google for "solitaire")

- There are four *foundations*, one for each suit, that are built up in rank order starting from Aces (rank 1). In Figure 1, the hearts foundation contains the Ace and two of hearts, the spades foundation contains the Ace of spades, and the remaining foundations are empty.
- The *stock* contains cards which may be played. These are turned over, either one at a time or three at a time (depending on the variation of Klondike being played).
- The *waste* contains turned over cards from the stock. The top of the waste may be played onto the tableau or one of the foundations, if such a move is legal. In Figure 1, the waste contains currently the three of clubs, and the eight of clubs (on top). There are no legal moves for the eight of clubs.

For a more detailed overview of Klondike Solitaire, see the wikipedia page at `https://en.wikipedia.org/wiki/Klondike_(solitaire)`.

3 Input file format

The input is a (mostly) free-form text file, meaning an arbitrary amount of whitespace (`' '`, `'\t'`, `'\r'`, and `'\n'` characters) may separate elements, unless otherwise specified. Comments, which start with a `'#'` character, and end with a `'\n'` character, should be treated the same as a single `'\n'` character. The input file has the following sections, which start with the specified keyword, and must appear exactly in this order.

1. The `RULES:` section indicates which variant of Solitaire is being played (this will be significant in later parts of the project). This section contains, in order:
   - "`turn 1`" or "`turn 3`". This indicates if 1 or 3 cards should be turned over at a time from the stock to the waste.
   - "`unlimited`" or "`limit N`" where `N` is one of $\{0, 1, 2, \ldots, 9\}$. This indicates the number of times that the waste can be "reset" back into the stock.
2. The `FOUNDATIONS:` section indicates the current top card of each of the four foundations, in the order of **clubs, diamonds, hearts, spades**. Cards are always specified in the order of **rank** and then **suit**, which are single characters each with no space in between.
   - The rank is one of { `'A'`, `'2'`, `'3'`, `'4'`, `'5'`, `'6'`, `'7'`, `'8'`, `'9'`, `'T'`, `'J'`, `'Q'`, `'K'` }.

- The suit is one of { `'c'`, `'d'`, `'h'`, `'s'` }, short for clubs, diamonds, hearts, and spades. Clubs and spades are black in color, while diamonds and hearts are red in color.

   For the *foundations only*, the rank may also be `'_'`, indicating that the foundation is empty.

3. The `TABLEAU:` section indicates the current status of the seven tableau columns. A column is specified on a single line, with one or more space characters between elements. A comment may appear at the end of a column line. A column line contains the following, in order:

   - Zero or more covered cards.
   - The `'|'` character, which serves to separate the covered cards from the uncovered ones.
   - Zero or more uncovered cards.

   Columns are specified in reverse order: column 7 first, then column 6, and so on, down to column 1. Blank lines and comment lines may appear between column lines.

4. The `STOCK:` section, indicating both the waste and the remaining stock. This section contains, in order:

   - Zero or more cards in the waste. The last card in the waste is the top card.
   - The `'|'` character, which serves to separate the waste from the stock.
   - Zero or more cards in the stock. The first card in the stock is the top of the stock. Playing the next card(s) in the stock can be simulated by moving the `'|'` character to the right.

   Any amount of whitespace (including comments and newlines) may separate elements in this section.

5. The `MOVES:` section. This will be specified in part 2.

Example input files are given in Section 6.

4 Invalid input files: what to check for

When the input file is *invalid* (see discussion below), your `check` executable should write a descriptive and appropriate error message to standard error. If the error is due to an improperly-formatted input file, the error message should include the line number of the formatting error. The program may terminate after displaying the first error message.

4.1 Missing or duplicate cards

For extra points, check the initial game configuration for missing cards or duplicate cards. You may assume that all the appropriate cards are under the top card in each foundation. For example, if the 4 of hearts is the top of the hearts foundation, then this means the Ace, 2, and 3 of hearts are below it. If the Ace of hearts appears also in (say) the stock, then this would be considered a duplicated card. Similarly, if the 5 of hearts does not appear in the tableau, the stock, the waste, or the foundation, then this would be considered a missing card.

4.2 Valid piles

For extra points, check that the tableau piles are valid. This means, for each face-up card in a column except the first, it must have rank exactly one lower than the card before it, and opposite color.

4.3 The MOVES section

Your `check` executable does not need to verify the correctness of anything after the `MOVES:` keyword. Your `check` executable may stop processing the input file once the `MOVES:` keyword has been seen.

4.4 Formatting errors

Input files with formatting errors, such as incorrect or out of order keywords, or invalid cards, should cause an appropriate error message to be generated. These messages should include the line number of the formatting error, to help users debug the input file. For example:

```
Error near line 7: expecting 'TABLEAU:'
```

5 Grading

The project is worth 100 points for individuals, and 110 points for groups of 2. The points breakdown for various features of the project are listed below. You may implement more features than required, for extra credit.

10 pts	`README` file, that describes implemented features
10 pts	`DEVELOPERS` file, that gives an overview of your implementation, including a breakdown of source files and functions, and who authored each function.
10 pts	Working `Makefile`. Typing "`make`" should build your `check` executable.
7 pts	Reads from `stdin` if no filename argument
7 pts	Reads from filename passed as argument
4 pts	Normal output to `stdout`
4 pts	Correctly formatted output on valid inputs
6 pts	Reports correct number of covered cards on valid inputs
6 pts	Reports correct number of stock cards on valid inputs
6 pts	Reports correct number of waste cards on valid inputs
5 pts	No error messages on valid inputs
5 pts	Error messages to `stderr`
5 pts	Formatting error messages include line numbers
10 pts	Appropriate error messages for formatting errors
5 pts	Stress tests
10 pts	Check that the tableau piles are legal
5 pts	Check for duplicate cards
5 pts	Check for missing cards

6 Examples

6.1 Screenshot example

Input file

```
# Comments. Mostly ignored by the parser.
#
# Input file that could (depending on the hidden cards)
# correspond to the screenshot of Google's Solitaire.
#

# The input file is designed to be human readable
# and editable with minimal pain, which makes it
# easier for everyone to create test inputs.
# However it makes parsing a little tricky.

RULES:      # This section must be first
```

```
# The following must appear in this order

  turn 1     # flip over one card at a time
  unlimited  # Easiest possible version of Klondike

FOUNDATIONS: # Alpha order: c,d,h,s
  _c    # nothing on clubs foundation
  _d    # nor on the diamonds foundation
  2h    # hearts foundation has 2h and Ah
  As    # spades foundation has As

TABLEAU:
  #
  # Columns in reverse order. This is because, if you
  # tilt your head 90 degrees to the left,
  # you will get the tableau as you would see it while playing.
  #

  8d 5c 7h Jd | Qs Jh Tc   # Column 7

  Ad 3h 4d 5s | 7d 6s 5d 4s # Column 6

  7s Kd | 3s               # Column 5

  6h Qc 4h | 7c            # Column 4

  8s 2s | 4c 3d 2c         # Column 3

  |                        # Column 2 (empty)

  | Ks Qh Jc Td            # Column 1 (nothing covered)


STOCK:
  3c 8c | Th Kh 8h # These don't have to appear all on one line
  Qd 9s 6c Kc Ac Ts
  Js          2d  # Lots of spaces, just because
  9h 6d 9c 5h 9d

MOVES:
  # For part 1, ignore everything after the MOVES: keyword.
  # For later parts, there may be things after this to process.
```

Corresponding output

```
Input file is valid
15 covered cards
16 stock cards
2 waste cards
```

6.2 Almost over game

Input file

```
#
# An almost won game
#
RULES:
turn 1
limit 1  # one more waste pile reset

FOUNDATIONS:
Ac 2d Jh Ts

TABLEAU:
Td | 8d
| 6d 5c 4d 3c
| Kd Qs
| 9c
| Ks Qd Js
| Kc Qh Jc
| Kh Qc Jd Tc 9d 8c 7d 6c 5d 4c 3d 2c

STOCK:
7c |

MOVES:
```

Corresponding output

```
Input file is valid
1 covered cards
0 stock cards
1 waste cards
```

6.3 Stuck game

```
RULES: turn 1 unlimited FOUNDATIONS: Jc Td Th Ts
TABLEAU:
  |
  | Kc Qh
  |
  | Kh Qs
  | Kd Qc
  Jh Jd Js Ks | Qd
  |
STOCK: | MOVES:
```

Corresponding output

```
Input file is valid
4 covered cards
0 stock cards
0 waste cards
```

6.4 Example with duplicated cards

```
RULES: turn 1 unlimited FOUNDATIONS: Jc Qd Qh Ks
TABLEAU:
  |
  | Kc Qh Jc
  |
  | Kh Qs
  | Kd Qc
  |
  |
STOCK: | MOVES:
```

Corresponding error message

```
Duplicated cards: Qs Qh Jc
```

6.5 Example with missing cards

```
RULES: turn 1 unlimited
FOUNDATIONS: Jc Jd Jh Js
TABLEAU:
  |
  | Kc Qh
  |
  | Kh
  |
  |
  |
STOCK:
  Qs |
MOVES:
```

Corresponding error message

```
Missing cards: Ks Qd Kd Qc
```

7 What to submit

7.1 In your git repository

Remember to include the following in your git repository.

- All TAs and the instructor as "reporters" (with read access).
- C Source code.
- A `Makefile`, so your executable can be built by simply typing "`make`". The TAs will **build and test your code on `pyrite`**.
- A `README` file, to indicate which features are implemented.
- A `DEVELOPERS` file, with necessary information about the source code for other developers. For group submissions, this file also should indicate which student(s) implemented which function(s).
- A tag with an appropriate name (e.g., "Part1", or "Part1a", "Part1b" in case you need to create more than one tag) as a frozen snapshot for the TAs to grade.

7.2 In Canvas

Please submit under the appropriate assignment in Canvas: either as an individual submission (you worked by yourself) or as a group submission (you worked in a group of 2 students). In the text submission box, indicate:

- The webpage URL for your (group's) git repository. Please indicate if this is a different repository than the one you used for homework 1 (if you are working in a group of 2 students, it should be).
- The name of the git tag for the TAs to grade.
- For group submissions, indicate which students are part of the group.