The University of Queensland
School of Information Technology and Electrical Engineering
Semester Two, 2019
CSSE2310 / CSSE7231 - Assignment 1

Due: 20:00pm 15th August, 2019
Marks: 50
Weighting: 25% of your overall assignment mark (CSSE2310)
Revision: 1.1

Introduction

Your task is to write a C99 program (called `bark`) which allows the user to play a game(described later). This will require I/O from both the user and from files. Your assignment submission must comply with the C style guide (v2.0.4) available on the course website.

This assignment is to be your own individual work. Using code which you did not write[1] is against course rules and may lead to a misconduct charge.

Game instructions

The game consists of a deck of cards and a board (a rectangular grid with spaces to place cards). The deck consists of cards each having a number between 1 and 9 (inclusive) and a suit (represented by a capital letter).

A new game will end immediately if there are not at least 11 cards in the deck.

Each player draws a starting hand of 5 cards from the deck[2]. At the beginning of each player's turn, they draw one card from the deck and then play one card onto the board in an unoccupied space. Each card played (except for the first one) must be played immediately adjacent (horizontally or vertically) to a card already on the board. The game ends at the end of the turn when either

- the last card has been drawn from the deck.
- there is no free space on the board.

For the purposes of adjacency, the opposite edges of the board wrap. For example, in the following 5×4 board:

```
..X..
DBBBE
..B..
..Z..
```

X and Z are adjacent. D and E are also adjacent.

Scoring

For each suit, find the longest path between two cards of the same suit. Each step in the path:

- must be from a smaller number to a bigger number.

[1] See the first lecture for exceptions
[2] ie. The first player draws five cards, then the second player draws five cards

- AND must horizontal or vertical.

For example (space added for clarity):

```
7A      4A      5B      5C
2C      2B      2A      4C
..      1D      9R      6D
..      1A      ..      2D
..      3D      ..      5A
```

The longest path for A would be:
1A → 3D → 4A → 7A [4 points]
The path for B would be:
2B → **4A** → 5B [3 points]
Note that paths can have different suits in the middle of the path, just not at the ends.

The first player scores "odd" letters (A, C, E ...), the second player scores "even" letters (B, D, ...). The scoring player gains points equal to the number of cards in their longest path. For example if the longest paths for each suit were:

A	5
B	2
C	3
D	6
E	2

then the first player would get $5 = \max\{5, 3, 2\}$ and the second player would get $6 = \max\{6, 2\}$.

Invocation

To start a new game:
`bark` *deckfile width height p1type p2type*

To load a saved game:
`bark` *savefile p1type p2type*

deckfile/savefile are the names of / paths to the files.
The acceptable player types are: `h` or `a`

`h` denotes a player whose moves will come via `stdin`.

`a` denotes a player whose moves will be chosen by the program.

The dimensions of the board must each be between 3 and 100 inclusive.

Deck file format

```
N
Card1
Card2
...
CardN
```

Where

- N is the number of cards in the deck
- $Card1 \ldots CardN$ cards in the deck.

Lines are separated[3] by '\n' (this applies to all input files).

Cards in files (deck files and save game files) are written as rank followed by suit.

Save file format

```
w h n v
deckname
ccccccccccc
ddddddddddd
GGGGGG
GGGGGG
GGGGGG
GGGGGG
```

Where

- w and h are the width and height of the board respectively (single space separated). Reminder, dimensions are between 3 and 100.
- n is the number of cards which have been drawn from the deck already (including dealing).
- v is 1 or 2 indicating which player should have their turn now.
- *deckname* is the name of the deck file.
- *ccccccccccc* is the current hand for Player 1.
- *ddddddddddd* is the current hand for Player 2. (The number of cards in each player's hand will vary depending on whose turn it is).
- The player indicated in v should have six cards, the other player should have five cards.
- *GGGGGG* is the board with blank places indicated by **

Example save file

```
7 6 14 2
d1.deck
2A3A5A6C6D
2B3B4B5B7D7C
1A1B**********
**4A**********
**************
**************
**************
**************
```

[3] Be careful of the difference between "terminated by" and "separated by". "Separated" implies that the last line of input might not end in a newline.

Interaction

After any valid move is made, redisplay the board. Also display the board at the beginning of the game or after a successful load.

Whenever a hand needs to be displayed, the cards should be listed in the order dealt / drawn. That is, newly drawn cards are added as the last card in the hand. The cards should be space separated.

Human Player

Human player initial prompt (replace # with 1 or 2 depending on which player is asked for input):

`Hand(#):` *C1 C2... C6*

`Move?`

Repeat prompt (used when the previously entered move was invalid for some reason, or if the user attempted to save the game):

`Move?`

In both of the above, `Move?` is followed by a space.

To place a card from hand:

`card c r`

where card is a number between 1 and 6 (inclusive) indicating the card from hand to play; and c and r are the column and row respectively (beginning with 1). If the move is not valid (including if the place indicated by (c, r) is already filled), prompt for a new move.

To save a game:

`SAVE`*zzzzz*

Where zzzzz is the name of the file to save the game to. This name must contain at least one character.

If the name is too short or there is some other problem saving the game, then print:

`Unable to save`

to standard out.

Automated Players

At the start of their turn, they will display their hand:

`Hand:` *C1 C2 ...C6*

Automated players will not prompt for moves. Instead, after they have chosen a move, they will print the following to `stdout`:

`Player ? plays ?? in column ? row ?`

followed by a newline. With `?` replaced with (in order)

- the player number
- the card (number then suit)
- the column number
- the row number.

The player chooses the first card in their hand and places it in the first space which is adjacent to an already played card.

Player 1 starts searching in the top left corner of the board $(0, 0)$, look across the row (from left to right). If no suitable space ıs found, move down to the next row an continue searching from the lefthand side.

If there are no cards on the board, then Player 1 will play the first card in their hand in the middle of the board. So for a 10×7 board, the "middle" would be at $(5, 4)$.

Player 2 acts in a similar way but starts at the bottom right corner and starts at the right hand side of each row (right to left). The player will move up a row if it doesn't find a suitable space.

End of Game

When a game ends normally(not due to an error/ end of input), prınt the scores to standard out:
`Player 1=? Player 2=?`

substituting the scores for `?`.

Exit status

Check for the error conditions in this table in the order given. For example, the named deckfile is does not exist and the board dimensions are not numbers, then use error 2 not error 3.

Code	**Reason**	**Message** (to `stderr`)
0	Normal game over	
1	Incorrect number of args	`Usage: bark savefile p1type p2type` `bark deck width height p1type p2type`
2	Player types are not `a` or `h` or the board dimensions are not integers in the correct range	`Incorrect arg types`
3	Can't read/process deckfile (in either mode) This includes errors in the deckfile format	`Unable to parse deckfile`
4	Can't read/process savefile	`Unable to parse savefile`
5	Not enough cards left in deck for a new game	`Short deck`
6	Board full on load	`Board full`
7	End of input for human player	`End of input`

Submission

Submission must be made electronically by committing using subversion. In order to mark your assignment the markers will check out `https://source.eait.uq.edu.au/svn/csse2310-s???????/trunk/ass1`. Code checked in to any other part of your repository will not be marked.

Note: No late submissions will be accepted for this assignment (see ECP). The markers will evaluate the last commit in your repository before the deadline. Late by 1 minute (or less) is still late.

Compilation

Your code must compile with command:
`make`
When you compile, you must use at least the following flags: `-Wall -pedantic -std=c99`. You may add additional flags if you wish (eg `-g`) but you must not use flags or pragmas to try to disable or hide warnings.

Specification Updates

It is possible that this specification contains errors or inconsistencies or missing information. Clarifications may be issued via the the course discussion forum. Any such clarifications posted 5 days (120 hours) or more before the due date will form part of the assignment specification. If you find any inconsistencies or omissions, please notify the teaching staff.

Test Data

Testing that your assignment complies with this specification is your responsibility. Some test data and scripts for this assignment will be made available.
`testa1.sh` will test the code in the current directory. `reptesta1.sh` will test the code which you have in the repository. The idea is to help clarify some areas of the specification and to provide a basic sanity check of code which you have committed. They are not guaranteed to check all possible problems nor are they guaranteed to resemble the tests which will be used to mark your assignments.

Marks

Marks will be awarded for both functionality and style.

Functionality (42 marks)

Provided that your code compiles (see above), you will earn functionality marks based on the number of features that your program correctly implements (as determined by automated testing), as outlined below. Partial marks may be awarded for partially meeting the functionality requirements[4]. Not all features are of equal difficulty. If your program does not allow a feature to be tested then you will receive 0 marks for that feature, even if you claim to have implemented it. For example, if your program can never open a file, we can not determine if your program would have loaded input from it. The markers will make no alterations to your code (other than to remove code without academic merit). Your programs should not crash or lock up/loop indefinitely. Your programs should not take a long time to run.

Please note that some features referred to in the following scheme may be tested in other parts of the scheme. For example, if you can not display the initial board, you will fail a lot of tests, not just the one which refers to "initial board". Students are advised to pay close attention to their handing of end of input situations.

- Command args — correct response to
  - incorrect number of args (1 mark)
  - incorrect types of args (3 marks)
  - problem with deck file (3 marks)

[4] ie passing some but not all of the tests of a feature

  - problems with save file (3 marks)

- Human players
  - Correctly display initial board (2 marks)
  - Correctly handle first move (2 marks)
  - Correctly handle second move (2 marks)
  - Correctly save game (after unspecified number of moves) (2 marks)
- New games — Automated players
  - Handle first move (2 marks)
  - Handle second move (3 marks)
- Load saved game
  - Display board (2 marks)
  - Correctly handle first move (on loaded board) (2 marks)
- Complete Games
  - Human only (3 marks)
  - Human / Type a (6 marks)
  - Any combination (6 marks)

Style (8 marks)

Style marks will be calculated as follows:
Let A be the number of style violations detected by simpatico plus the number of build warnings. Let H be the number of style violations detected by human markers. Let F be the functionality mark for your assignment. Let M_A be the automated style mark and M_H be the human mark.

- If $A > 10$, then your style mark will be zero and M_H will not be calculated.
- Otherwise, let $M_A = 4 \times 0.8^A$ and $M_H = M_A - 0.5 \times H$ your style mark S will be $M_A + \max\{0, M_H\}$.

Your total mark for the assignment will be $F + \min\{F, S\}$.

A maximum of 5 violations will be penalised for each broad guideline area. The broad guideline areas are Naming, Comments, Braces, Whitespace, Indentation, Line Length and Overall. For naming violations, the penalty will be one violation per offending name (not per use of the name) up to the maximum of five. You should pay particular attention to commenting so that others can understand your code. The marker's decision with respect to commenting violations is final — it is the marker who has to understand your code. To satisfy layout related guidelines, you may wish to consider the `indent(1)` tool. Your style mark can never be more than your functionality mark — this prevents the submission of well styled programs which don't meet at least a minimum level of required functionality.

Tips

1. No valid line of input will contain more than 80 characters.

Example Sessions

```
$ ./bark d1.deck 3 4 a a
......
......
......
......
Hand: 1A 2A 3A 4A 5A 6C
Player 1 plays 1A in column 2 row 2
......
..1A..
......
......
Hand: 1B 2B 3B 4B 5B 7D
Player 2 plays 1B in column 2 row 3
......
..1A..
..1B..
......
Hand: 2A 3A 4A 5A 6C 6D
Player 1 plays 2A in column 2 row 1
..2A..
..1A..
..1B..
......
Hand: 2B 3B 4B 5B 7D 7C
Player 2 plays 2B in column 2 row 4
..2A..
..1A..
..1B..
..2B..
Hand: 3A 4A 5A 6C 6D 8C
Player 1 plays 3A in column 1 row 1
3A2A..
..1A..
..1B..
..2B..
Hand: 3B 4B 5B 7D 7C 9C
Player 2 plays 3B in column 3 row 4
3A2A..
..1A..
..1B..
..2B3B
Hand: 4A 5A 6C 6D 8C 7D
Player 1 plays 4A in column 3 row 1
3A2A4A
..1A..
..1B..
..2B3B
Hand: 4B 5B 7D 7C 9C 7E
Player 2 plays 4B in column 1 row 4
3A2A4A
..1A..
..1B..
4B2B3B
Hand: 5A 6C 6D 8C 7D 7F
Player 1 plays 5A in column 1 row 2
3A2A4A
5A1A..
..1B..
4B2B3B
Hand: 5B 7D 7C 9C 7E 9D
Player 2 plays 5B in column 3 row 3
3A2A4A
5A1A..
..1B5B
4B2B3B
Player 1=4 Player 2=4
```

```
$ ./bark d1.deck 3 3 h h
......
......
......
Hand(1): 1A 2A 3A 4A 5A 6C
Move? 1 1 1
1A....
......
......
Hand(2): 1B 2B 3B 4B 5B 7D
Move? 3 1 3
1A....
......
3B....
Hand(1): 2A 3A 4A 5A 6C 6D
Move? 5 1 1
Move? 5 1 2
1A....
6C....
3B....
Hand(2): 1B 2B 4B 5B 7D 7C
Move? 3 2 2
1A....
6C4B..
```

```
3B....
Hand(1): 2A 3A 4A 5A 6D 8C
Move? 1 2 3
1A....
6C4B..
3B2A..
Hand(2): 1B 2B 5B 7D 7C 9C
Move? 2 3 2
1A....
6C4B2B
3B2A..
Hand(1): 3A 4A 5A 6D 8C 7D
Move? 2 3 3
1A....
6C4B2B
3B2A4A
Hand(2): 1B 5B 7D 7C 9C 7E
Move? 1 3 3
Move? 1 1 3
Move? 1 2 1
1A1B..
6C4B2B
3B2A4A
Hand(1): 3A 5A 6D 8C 7D 7F
Move? 1 3 1
1A1B3A
6C4B2B
3B2A4A
Player 1=3 Player 2=3
```

Updates

1.0 → 1.1

- Correct "middle" of board example for 1-based coordinates. middle=$\lfloor \frac{D+1}{2} \rfloor$.
- Used # (instead of ?) as a placeholder in example prompt to avoid confusion with the real ? after Move. (The Human player section).

0.9 → 1.0

1. Corrected duplicate card in the board example.
2. Formatting fixes and minor clarifications
3. Modifed deckfile description to use "CardN" rather than CN.
4. Added reminder about range of board dimensions.
5. Added an example savefile.
6. The move? prompt is followed by a space.
7. Emphasised the direction of search for automated players.
8. Exit status and error messages to be checked in the order they appear in the table.
9. Corrected M to M_H in style mark section.
10. Corrected the description of the save format (width and height were backwards).
11. Emphasised that while the endpoints of a path must be the same suit, you can have other suits in the middle.