Prepared for s4536324. Do not distribute.

The University of Queensland
School of Information Technology and Electrical Engineering
Semester Two, 2019
CSSE2310 / CSSE7231 - Assignment 3
Due: 21:00pm 26th September, 2019
Marks: 50
Weighting: 25% of your overall assignment mark (CSSE2310)
Revision 3.3

Purpose

In this assignment, you will write a group of three C99 programs to play a game (described later). One of the programs will be called `2310hub` it will control the game. The other two programs (`2310alice` and `2310bob` will be players.

The hub will be responsible for running the player processes and communicating with them via pipes (created via `pipe()`). These pipes will be connected to the players' standard ins and outs so from their point of view communication will be via `stdin` and `stdout`.

If some of the required programs compile and some do not, there will still be some marks allocated to each component working independently (see the marking section for more information).

Other than the hub starting player processes, there should not be any other parallelism used in this assignment. For example, no multi-threading nor non-blocking operations. Your programs are not permitted to start any additional processes[1] nor are your programs permitted to create any files *during their execution*.

Your assignment submission must comply with the C style guide (version 2.0.4) available on the course blackboard area. It must also not use banned functions or commands listed in the same place.

This is an individual assignment. You should feel free to discuss aspects of C programming and the assignment specification with fellow students. You should not actively help (or seek help from) other students with the actual coding of your assignment solution. It is cheating to look at another student's code and it is cheating to allow your code to be seen or shared in printed or electronic form. You should note that all submitted code may be subject to automated checks for plagiarism and collusion. If we detect plagiarism or collusion, formal misconduct proceedings will be initiated against you. A likely penalty for a first offence would be a mark of 0 for the assignment. Don't risk it! If you're having trouble, seek help from a member of the teaching staff. Don't be tempted to copy another student's code.

Do not commit any code to your repository unless it is your own work or it was given to you by teaching staff. **If you have questions about this, please ask.**

While you are permitted to use sample code supplied by teaching staff this year in this course. Code supplied for other courses or other offerings of this course is off limits — it may be deemed to be without academic merit and removed from your program before testing.

Game

What follows is an abstract description of the game. Precisely what your programs need to output will be described separately.

The game consists of:

- Players (P of them): who are positioned in a circle. One player is labelled 0 and the remaining players are numbered in clockwise order.

[1] other than those specified on the hub commandline arguments

- A deck of cards (d of them) containing no duplicates. Each card in the deck has:
  - A suit from $\{S, C, D, H\}$
  - A rank (in hexadecimal) from $1 \ldots 9, a \ldots f$.
- A threshold (T)

At the start of the game, each player is given a "hand" of $H = \lfloor \frac{d}{P} \rfloor$ cards (ie rounded down). That is, the first H cards in the deck are given to Player 0, the next H cards are given to Player 1, etc. Each player tracks:

- Their points (S)
- The number of cards with the 'D' suit, played in rounds which they won (V).

D cards are worth -1 points at the end of the game unless the player won at least T of them. In whıch case, they are worth 1 point each.

1. The game is played in H rounds (one card per round).
2. The lead player for each round is:
   - Player 0 in the first round.
   - The player who won the previous round for all other rounds.
3. The lead player plays one of the cards from their hand. The *suit* of this card is the "lead suit".
4. The other players, in clockwise order from the lead player, play one card from their hands:
   (a) If they have any cards belonging to the *lead suit*, then they must play one of those.
   (b) Otherwise play any card
5. Find the highest ranked card of the *lead* suit. The player who played that card:
   - gets 1 point/ (ie increases S)
   - is the lead player for the next round.
   - records how many **D** cards were played that round. (ie increases that player's V value)
6. When all rounds are complete, each player's score is
   - $S - V$ if $V < T$
   - $S + V$ if $V \geq T$

Deck file format

The first line of the deck file will contain a single int indicating how many cards are in the deck. The remaining lines (each ended by '\n') contain two chars, the suit and the rank of the card.

2310hub

The hub will take the following commandline arguments (in order):

- the name of the file to read to get the initial deck of cards
- the number of D cards needed in order to count them as positive.
- a list of player programs to start

Eg: `./2310hub d1.deck 11 ./2310alice ./2310bob ./2310bob`
Would start a game with one `2310alice` player and two `2310bob` players. A single player would need to win 11 or more D cards to have them count positive.

When running player processes, the hub must ensure that any output to `stderr` by players is supressed.

Output

The following are output to the hub's `stdout` followed by newline.

- At the start of a round, output the number of the lead player:
  `Lead player=?`

- At the end of a completed round output all cards played (in order starting with the lead player):
  `Cards=?.? ?.?`
  There should be a `.` between suit and rank.

- At the end of the game, output the scores (starting with Player 0):
  `0:??? 1:??? 2:???`

Exits and Messages

The following conditions should be checked in the order shown in this table. All messages are to `stderr` followed by newline.

Exit	Condition	Message
0	Normal exit	
1	Less than 4 commandline arguments	`Usage: 2310hub deck threshold player0 {player1}`
2	Threshold < 2 or not a number	`Invalid threshold`
3	Problem reading / parsing the deck	`Deck error`
4	Less than P cards in the deck	`Not enough cards`
5	Unable to start one of the players	`Player error`
6	Unexpected EOF from a player	`Player EOF`
7	Invalid message from a player	`Invalid message`
8	Player chooses card they don't have or don't follow suit	`Invalid card choice`
9	Received `SIGHUP`	`Ended due to signal`

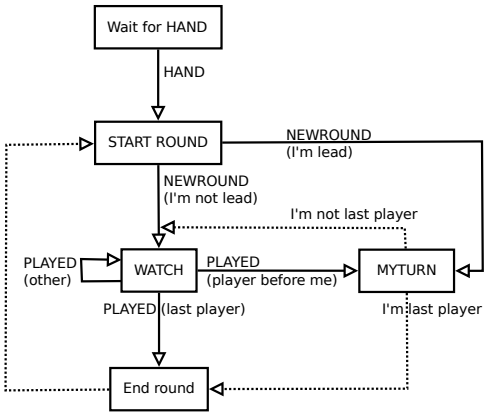
For a player to be considered "started", the hub must have read a @ from the corresponding player→hub pipe (this must be sent before any other output). For example, if the hub gets EOF while looking for the @, that would be error 5.

On recieving `SIGHUP`, the hub should kill the player processes before closing down.

Messages

When cards are sent though a pipe, they are encoded as the suit immediately followed by the rank eg `C5` or `Hc`.

Direction	Format	Detail
hub → player	`HAND`*n*`,`*SR*`,`*SR*`,`*SR*	Send a player their hand *n* is the number of cards in the hand
hub → player	`NEWROUND`*L*	Send to all players to indicate the start of a new round. *L* is the number of the lead player
hub → player	`PLAYED`*w*`,`*SR*	Another player played a card. *w*: number of the player, *SR*: the card
hub → player	`GAMEOVER`	Instruct players to close normally
player → hub	`PLAY`*SR*	Inform hub which card they will play



The figure above is intended to illustrate the order messages and actions should occur in. It does not constrain how you implement it. Solid lines indicate which messages should cause the statemachine to move between states. Dotted lines indicate transitions which are followed without needing a message. A `GAMEOVER` message could arrive at any time and will cause the player program to stop normally. Any other unexpected messages should cause an exit due to a bad message.

Validating messages

Player programs need to check that a message is properly formatted. eg Messages involving cards must contain only correctly formatted cards.

Player programs need to recognise when a message has arrived out of order. For example: a `PLAYED` message arriving before the `HAND` message, would be out of order.

Your players do not generally need to track game history. So players don't need to recognise:

- Another player has played the same card multiple times.
- Another player has played a card which you are holding.

- We will not test with decks containing duplicate cards.
- We will not test sending any messages (other than `GAMEOVER` when the game is supposed to be over).

The hub is still expected to check that players are holding the card they chose to play.

Players

Both player programs ("players") will have the same types of output, only their individual strategies will vary.

Players will read messages from the hub from their `stdin` and send instructions back to the hub via `stdout`. All extra output will be sent to `stderr`.

Players take the following commandline arguments:

- The number of players in the game
- This player's number (0 based).
- Threshold for **D** to count positive
- Initial hand size

The following messages are output to `stderr` and followed by newline.

Exit	Condition	Message
0	Normal exit	
1	Incorrect number of arguments	`Usage: player players myid threshold handsize`
2	Number of players < 2 or not a number	`Invalid players`
3	Invalid position for number of players	`Invalid position`
4	Threshold < 2 or not a number	`Invalid threshold`
5	Hand size < 1 or not a number	`Invalid hand size`
6	Invalid message from hub	`Invalid message`
7	Unexpected EOF from hub	`EOF`

The above conditions should be checked in order. *Conditions 1–5 should only be checked at startup.* Once the initial startup checks have successfully passed, the player should send a single '@' character (no newline) to `stdout`.

Extra output from players (to `stderr`)

At the end of each round, print the id of the lead player followed by the list of cards played in that round. eg: `Lead player=1: C.5 C.1 C.e H.3`

Player "alice" strategy

In the following, stop once a decision is reached.

1. If we are the lead player:
    - Check each of the suits in this order **S, C, D, H**. If we have at least one card in the suit, play the highest ranked one.
2. If we have a card in the lead suit:

- Play the lowest card in the lead suit

3. Check the suits in the following order **D, H, S, C**
   - If we have a card for that suit, play the highest one.

Player "bob" strategy

In the following, stop once a decision is reached.

1. If we are the lead player:
   - Check each of the suits in this order **D, H, S, C**. If we have at least one card in the suit, play the lowest ranked one.
2. If at least one other player (including us) has won at least (*threshold* − 2) **D** cards and some **D** cards have been played this round:
   - If we have a card in the lead suit
     - Play the highest card in the lead suit
   - Check each of the suits in this order **S, C, H, D**
     - If you have a card of that suit, play the lowest one
3. If we have a card in the lead suit:
   - Play the lowest card in the lead suit
4. Check the suits in this order **S, C, D, H**
   - If you have at least one card in the suit, play the highest.

Compilation

Your code must compile (on a clean checkout) with the command:
`make`

Each individual `.c` file must compile with at least `-Wall -pedantic -std=gnu99`. You may of course use additional flags but you must not use them to try to disable or hide warnings. You must also not use pragmas to achieve the same goal. Your code must be compiled with the `gcc` compiler.

If the make command does not produce one or more of the required programs, then those programs will not be marked. If none of the required programs are produced, then you will receive 0 marks for functionality. Any code without academic merit will be removed from your program before compilation is attempted [This will be done even if it prevents the code from compiling]. If your code produces warnings (as opposed to errors), then you will lose style marks (see later).

Your solution must not invoke other programs apart from those listed in the command line arguments for the hub. Your solution must not use non-standard headers/libraries.

Submission

No late submissions will be marked for this assignment under any circumstances. Submission must be made electronically by committing using subversion. In order to mark your assignment, the markers will check out `/trunk/ass3/` from your repository on `source.eait.uq.edu.au`. *Do not create subdirectories under `/trunk/ass3/`.* The marking may delete any such directories before attempting to compile. Code checked in to any other part of your repository will not be marked.

Test scripts will be provided to test the code on the trunk. Students are *strongly advised* to make use of this facility after committing.

Note: Any .h or .c files in your `trunk/ass3` directory will be marked for style *even if they are neither linked by the makefile nor #included by some other file.* If you need help moving/removing files in svn, then ask. Consult the style guide for other restrictions.

You must submit a `Makefile` or we will not be able to compile your assignment. Remember that your assignment will be marked electronically and strict adherance to the specification is critical.

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 your program correctly implements, as outlined below. Partial marks may be awarded for partially meeting the functionality requirements. 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 run for unreasonably long times.

- 2310alice player
  - Arg checking (2 marks)
  - Correctly handle first move as lead (2 marks)
  - Correctly handle first move as non-lead (2 marks)
  - Correctly play a single round game (2 marks)
  - Bad hub messages (2 marks)
- 2310bob player
  - First play as lead (1 mark)
  - First play as non-lead (2 marks)
  - End of first round (2 mark)
- For 2310hub
  - Argument checking (5 marks)
  - Correctly handle `SIGHUP` (2 marks)

- Play partial game with 3 alices (3 marks)
- Play complete game with 3 alices (4 marks)
- Play complete game with more alices (3 marks)
- Play complete game with a mixture of player types (10 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.

- If $A > 10$, then your style mark will be zero and M 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 $F + \min\{F, S\}$.

Specification Updates

It is possible that this specification contains errors or inconsistencies or missing information. It is possible that clarifications will be issued via the course website. 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.

Sample session

Execution: `./2310hub d1.deck 2 ./2310alice ./2310alice`
Deckfile:

```
7
C4
D2
D3
D4
C2
C3
Hf
```

hub output

```
Lead player=0
Cards=C.4 C.2
Lead player=0
Cards=D.3 D.4
Lead player=1
Cards=C.3 D.2
0:1 1:5
```

Player 0

Input:

```
HAND3,C4,D2,D3
NEWROUND0
PLAYED1,C2
NEWROUND0
PLAYED1,D4
NEWROUND1
PLAYED1,C3
GAMEOVER
```

Ouput:

```
@PLAYC4
PLAYD3
PLAYD2
```

Player 1

Input:

```
HAND3,D4,C2,C3
NEWROUND0
PLAYED0,C4
NEWROUND0
PLAYED0,D3
NEWROUND1
PLAYED0,D2
GAMEOVER
```

Output:

```
@PLAYC2
PLAYD4
PLAYC3
```

Tips and Fixes

1. Your work will be made much easier if you place code used in multiple places into a source file shared by multiple programs. eg:
   - Encoding and decoding messages
   - Do I have a card of suit X?
   - What is my highest card of suit X?
   - What is my lowest card of suit X?
2. Write some test code to check shared functions *before* using them in the programs.
3. Remember that there is no '\n' after the initial '@'
4. Do not make your players' behaviour depend on argv[0]. There is no guarantee that programs will be named the way you think they will.
5. Use `tee` to capture the input and output to your players for debugging purposes.
6. Messages not terminated by '\n' (ie EOF happens), should still be processed.
7. You are not permitted to use non-blocking I/O in this assignement.
8. You are not permitted to use multi-threading in this assignment.
9. We strongly recommend not using `read()` or `write()` in this assignment `fread()` and `fwrite()` are better but still not as useful as you might think.

Updates

3.2→3.3

1. Corrected misspelling of GAMEOVER in text.
2. Extra output in player section is supposed to be to player's `stderr` not the hub's.
3. Noted that exits 1–5 for players are only checked at startup.

0.1→3.2

1. Corrected misspellings
2. Fixed non-hex example of card rank.
3. Fixed high card rank not being in hex.
4. Fixed Alice strategy in example session.
5. Empahised that V is a per player quantity.
6. Clarify what about messages needs to be validated
7. Added more tips
8. Added state machine diagram and explanation.