The University of Queensland
School of Information Technology and Electrical Engineering
Semester Two, 2015
CSSE2310 / CSSE7231 - Assignment 3
Due: 11:10pm 2nd October, 2015
Marks: 50
Weighting: 25% of your overall assignment mark (CSSE2310)
Revision 0.9.1

Introduction

In this assignment, you will write two programs to play a card game ("clubs"). A separate document describing the rules of "clubs" will be provided. The first program (`clubber`) will listen on its `stdin` for information about the game and give its moves to `stdout`.The program will send information about the state of the game to `stderr`.The second program (`clubhub`) will start a number of processes (eg `clubber`) to be players and communicate with them via pipes. The hub will be responsible for running the game (sending information to players; processing their moves and determining the score). The hub will also ensure that, information sent to `stderr` by the players, is discarded.

Your programs must not create any files on disk not mentioned in this specification or in command line arguments. Your assignment submission must comply with the C style guide (version 2.0.2) available on the course blackboard area. 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. You should read and understand the statements on student misconduct in the course profile and on the school web-site: `http://www.itee.uq.edu.au/itee-student-misconduct-including-plagiarism`

As with Assignment 1, we will use the subversion (svn) system to deal with assignment submissions. 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.**

Invocation

The parameters to start a player are: the number of players (in total) and the label of this particular player (starting at A). For example:

```
./clubber 3 B
```

The parameters to start the hub are:

- The name of a file containing the decks of cards to use.
- The number of points to play to.
- The names of programs to run as players (one for each player). There must be at least two and no more than four players. the names of programs to run as players (one for each player).

For example: `./clubhub ex.decks 10 ./clubber ./clubber ./clubber`
Would start a game with 3 players (each running `./clubber`) and using the decks contained in `ex.decks`. The game will end when at least one player has 10 or more points.

Representations

Whenever cards are represented as strings (in input files, output files, messages to user, etc) they will use two characters. The first will be one of {S,C,D,H} indicating the suit of the card (Spades, Clubs, Diamonds, Hearts respectively). The second will be one of {2,3,4,5,6,7,8,9,T,J,Q,K,A} indicating the rank of the card.

Decks format

A decks file will consist of a sequence of decks **separated** by lines containing a single dot (**.**). An individual deck will be stored as a number of lines containing comma separated cards. Empty lines and lines beginning with # should be skipped over when processing.

For example:

```
#File with a single deck
2S,3S,4S,5S,6S,7S,8S,9S,TS,JS,QS,KS,AS
2C,3C,4C,5C,6C,7C,8C,9C,TC,JC,QC,KC,AC
2D,3D,4D,5D,6D,7D,8D,9D,TD,JD,QD,KD,AD
2H,3H,4H,5H,6H,7H,8H,9H,TH,JH,QH,KH,AH
#There is no terminating .
```

A file with additional decks would look like:

```
2S,3S,4S,5S,6C,7S,8C
9S,TS,JS,QS,KS,AS
2C,3C,4C
5C,6C,7C,8C
9C,TC,JC,QC,KC,AC
2D,3D,4D,5D,6D
7D,8D,9D,TD,JD,QD,KD,AD
2H
3H
4H,5H,6H,7H,8H,9H
TH,JH,QH,KH,AH
.


#Suits in order are boring
2H,3H
2S,3S,4S,5S,6C,7S,8C
TH,JH,QH,KH,AH
2C,3C,4C
9S,TS,JS,QS,KS,AS
5C,6C,7C,8C
9C,TC,JC,QC,KC,AC
2D,3D,4D,5D,6D
7D,8D,9D,TD,JD,QD,KD,AD
```

```
4H,5H,6H,7H,8H,9H
```

How your player will work

1. If the player is leading:
   (a) If the player has the lowest known club, play it. That is, any clubs which are lower than it have already been played.
   (b) Play the lowest available card from the suits in this order:
      - Diamonds
      - Hearts
      - Spades
      - Clubs
2. If the player is not leading:
   (a) If the player can "follow suit", then play the lowest available card of the lead suit.
   (b) If the player is last to play in this trick, the play the highest ranked available card from the suits in this order:
      - Hearts
      - Diamonds
      - Clubs
      - Spades
   (c) Otherwise, play the highest ranked available card from the suits in this order:
      - Clubs
      - Diamonds
      - Hearts
      - Spades

Messages

Messages from hub to player

Message	**Params**	**Meaning**
newround *list of cards*		Start a new round and this your hand.
newtrick		It is your turn to lead
trickover		The current trick is over
yourturn		It is your turn to play
played *c*	*c* is a card	*c* was played (possibly by you)
scores *list of scores*		Scores of each player
end		The hub won't be communicating any more. Shutdown normally

For example Player C might see:

```
newround 4H,7D,AC,3C,5C,7H,9D,TC,2S,4S,8S,JS,3D
played A2C
played BAD
yourturn
```

Messages from player to hub
The player will send two types of message to the hub:

- As soon as the player has started successfully (no errors in argv), send a single dash character followed by a newline.
- In response to a `newtrick` or `yourturn`, the card they wish to play (followed by a newline).

Player output to stderr

When a message (eg XYZ) arrives from the hub, print the following to `stderr`
`From hub:XYZ`
Only print the first 20 characters of the message (followed by a newline if one was not present already). Then print the following information after the message has been processed:
Hand: *The player's current hand*
Played (S): *comma separated list of spades (ranks only) which have been played this round*
Played (C): (as above for Clubs)
Played (D): (as above for Diamonds)
played (H): (as above for Hearts)
Scores: *comma separated list of scores in player order*

The cards in the "Hand" lines must be arranged by suits (S,C,D,H), then by rank within each suit. The cards on the "Played" lines must be arranged in increasing rank order.

For example:

```
From hub:played2H
Hand: 5S,8S,9S,3C,AC,4D,JD,KD,8H,JH
Played (S):
Played (C): 5,T
Played (D): 5,7,8,9
Played (H): 2,T,Q,K,A
Scores: 0,0,0,0
```

The player program will also generate some error messages which are to be sent to `stderr`.

Hub output

The hub should discard any output sent to the `stderr` of player processes. As well as errors outlined below(which will be sent to `stderr`), the hub should output the following to `stdout`.

- When a new round begins display the hand of each player in order.
- When a round ends, display the scores message sent to players.
- When a player leads a card, display:

  ```
  Player ? led ??
  ```

- When a player plays a card, display:

  ```
  Player ? played ??
  ```

With the ?s filled in with player and card.

- At the end of the game (assuming no player exits early). Display a message giving the winners. For example:
  `Winner(s):  D`
  `Winner(s):  A B`

Exits

Exit status for player

All messages to be printed to `stderr`.

Condition	Exit	Message
Normal exit due to game over	0	
Wrong number of arguments	1	`Usage: player number_of_players myid`
Invalid number of players	2	`Invalid player count`
Invalid player ID	3	`Invalid player ID`
Pipe from hub closed unexpectedly (i.e. before a gameover message was received)	4	`Unexpected loss of hub`
Invalid message from hub	5	`Bad message from hub`

Exit status for hub

All messages to be printed to `stderr`.

Condition	Exit	Message
Normal exit due to game over	0	
Wrong number of arguments	1	`Usage: clubhub deckfile winscore prog1 prog2 [prog3 [prog4]]`
Winscore is not a positive integer	2	`Invalid score`
Unable to open decks file for reading	3	`Unable to access deckfile`
Contents of the decks file are invalid	4	`Error reading deck`
There was an error starting and piping to a player process	5	`Unable to start subprocess`
A player process ends unexpectedly. ie Reading from the pipe from that process fails before a gameover message has been sent to it. [only applies once all child processes have started successfully]	6	`Player quit`
One of the players has sent an invalid message	7	`Invalid message received from player`
One of the players sent a properly formed message but it was not a legal play	8	`Invalid play by player`
The hub received SIGINT	9	`SIGINT caught`

Note that both sides detect the loss of the other by a read failure. Write calls could also fail, but your program should ignore these failures and wait for the associated read failure. Such write failures must not cause your program to crash.

Shutting down the Hub

Whether the hub shuts down in response to a SIGINT or of its own volition, it should follow the same procedure.It should ensure that all child processes have terminated. If they have not terminated within 2 seconds of the hub sending the "end" message, then the hub should terminate them with SIGKILL.For each player (in order), do one of the following:

1. If the process terminated normally with exit status 0, then don't print anything.
2. If the process terminated normally with a non-zero exit status, then print (to `stderr`):

   `Player ? exited with status ?`
3. If the process terminated due to a signal, then print (to `stderr`):

   `Player ? terminated due to signal ?`

(Fill in ? with the appropriate values)

Compilation

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

Each individual 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.

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

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`. Code checked in to any other part of your repository will not be marked.

The due date for this assignment is given on the front page of this specification. Note that no submissions can be made more than 96 hours past the deadline under any circumstances.

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 not linked by the makefile*. 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.

- (Player) argument checking (2 marks)
- (Player) correctly handles early hub loss and “end” message (2 marks)

- (Player) Correct choice for initial move (2 marks)
- (Player) Correctly handle one round (4 marks)
- (Player) Correctly handle complete game (2 marks)
- (Player) Detect invalid messages (2 marks)
- Hub argument checking (2 marks)
- Detect failure to start players (2 marks)
- Correctly handle players which close early (3 marks)
- Correctly handle 2 player games using a single deck (5 marks)
- Correctly handle invalid messages / invalid plays (2 marks)
- Play complete games with 2 players (5 marks)
- Play complete games with 3 and four players (6 marks)
- Correctly cleanup still running subprocesses on exit (3 marks)

The items marked (Player) indicate that the player will be tested independently of the hub (so the hub does not need to be working in order to get these marks). Keep in mind that tests of complete games may use varying numbers of decks in the deckfile.

Style (8 marks)

If g is the number of style guide violations and w is the number of compilation warnings, your style mark will be the minimum of your functionality mark and:

$$8 \times 0.9^{g+w}$$

The number of compilation warnings will be the total number of distinct warning lines reported during the compilation process described above. The number of style guide violations refers to the number of violations of the current C Programming Style Guide. 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)` and `expand(1)` tools. 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.

Late Penalties

Late penalties will apply as outlined in the course profile.

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.

Test Data

Test data and scripts for this assignment will be made available. (`testa3.sh`, `reptesta3.sh`) 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.* Testing that your assignment complies with this specification is still *your* responsibility.

Example session

The following example is for a two player game to 10 points. It finishes in a single round.

The hub would output (long lines wrapped):

```
Player (A): 2S,4S,6S,8S,TS,QS,AS,3C,5C,7C,9C,
JC,KC,2D,4D,6D,8D,TD,QD,AD,3H,5H,7H,9H,JH,KH
Player (B): 3S,5S,7S,9S,JS,KS,2C,4C,6C,8C,TC,
QC,AC,3D,5D,7D,9D,JD,KD,2H,4H,6H,8H,TH,QH,AH
Player A led 2D
Player B played 3D
Player B led 2C
Player A played 3C
Player A led 4D
Player B played 5D
Player B led 4C
Player A played 5C
Player A led 6D
Player B played 7D
Player B led 6C
Player A played 7C
Player A led 8D
Player B played 9D
Player B led 8C
Player A played 9C
Player A led TD
Player B played JD
Player B led TC
Player A played JC
Player A led QD
Player B played KD
Player B led QC
Player A played KC
Player A led AD
Player B played AH
Player A led 3H
Player B played 2H
Player A led 5H
Player B played 4H
Player A led 7H
Player B played 6H
Player A led 9H
Player B played 8H
Player A led JH
Player B played TH
Player A led KH
Player B played QH
Player A led 2S
Player B played 3S
Player B led AC
Player A played AS
Player B led 5S
Player A played 4S
Player B led 7S
Player A played 6S
Player B led 9S
Player A played 8S
Player B led JS
Player A played TS
Player B led KS
Player A played QS
scores 12,1
```

```
Winner(s): B
```

Player A would see the following on its `stdin`:

```
newround 2S,4S,6S,8S,TS,QS,AS,3C,5C,7C,9C,JC,
KC,2D,4D,6D,8D,TD,QD,AD,3H,5H,7H,9H,JH,KH
newtrick
played 2D
played 3D
trickover
played 2C
yourturn
played 3C
trickover
newtrick
played 4D
played 5D
trickover
played 4C
yourturn
played 5C
trickover
newtrick
played 6D
played 7D
trickover
played 6C
yourturn
played 7C
trickover
newtrick
played 8D
played 9D
trickover
played 8C
yourturn
played 9C
trickover
newtrick
played TD
played JD
trickover
played TC
yourturn
played JC
trickover
newtrick
played QD
played KD
trickover
played QC
yourturn
played KC
trickover
newtrick
played AD
played AH
trickover
newtrick
played 3H
played 2H
trickover
newtrick
played 5H
played 4H
trickover
newtrick
played 7H
played 6H
trickover
newtrick
played 9H
played 8H
trickover
newtrick
played JH
played TH
trickover
newtrick
played KH
played QH
trickover
newtrick
played 2S
played 3S
trickover
played AC
yourturn
played AS
trickover
played 5S
yourturn
played 4S
trickover
played 7S
yourturn
played 6S
trickover
played 9S
```

```
yourturn
played 8S
trickover
played JS
yourturn
played TS
trickover
played KS
yourturn
played QS
trickover
scores 12,1
end
```

Player A would send the following on its `stdout`.

```
-2D
3C
4D
5C
6D
7C
8D
9C
TD
JC
QD
KC
AD
3H
5H
7H
9H
JH
KH
2S
AS
4S
6S
8S
TS
QS
```

Notes and Addenda:

1. This assignment implicitly tests your ability to recognise repeated operations/steps and move them into functions to be reused. If you rewrite everything each time it is used, then writing and debugging your code will take much longer.
2. Start early.
3. Write simple programs to try out fork(), exec() and pipe().
4. Be sure to test on moss.
5. You should not assume that system calls always succeed.

5b. You are not permitted to use any of the following functions in this assignment.

- `system()`
- `popen()`
- `prctl()`

6. You may assume that only 8bit characters are in use[no unicode].
7. When hub exits (under program control or in response to `SIGINT`), it should not leave any child processes running (you can use `SIGKILL`).
8. All tab characters will be treated as being (up to 8 spaces).
9. You may not use any `#pragma` in this assignment.

10. Where your program needs to parse numbers (as opposed to characters) from strings, the whole string must be a valid number. e.g. `"3biscuit"` is not a valid number, nor is `"3 "`

11. Neither program should assume that the other will be well behaved. That is, if the hub sends a valid message, you may believe it. However, if the hub sends an invalid message, your player should not crash.

12. You will need to do something with SIGPIPE.

13. Valid messages contain no leading, trailing or embedded spaces. Messages must be exactly the correct length.

14. All decks must be read into memory before the game starts.

15. All messages except the initial `-` are followed by newlines.

16. You should only report on the exit statuses of the players if all players started successfully.