The University of Queensland
School of Information Technology and Electrical Engineering
Semester Two, 2014
CSSE2310 / CSSE7231 - Assignment 3
Due: 11:10.0pm 3 October, 2014
Marks: 50
Weighting: 25% of your overall assignment mark (CSSE2310)
Revision 3.4

Introduction

In this assignment, you will write two programs to play a card game. The first program (`player`) 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 (`hub`) will start a number of processes 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) 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: `./player 3 B`

The parameters to start the hub are: the name of a file containing the decks of cards to use, the names of programs to run as players (one for each player). For example: `./hub ex.decks ./player ./player ./player` Would start a game with 3 players (each running `./player`) and using the decks contained in `ex.decks`. There must be at least 2 players and no more than 4.

The game

The game is played with a deck of 16 cards, each labelled with a number between 1 and 8 (inclusive). The nunber of each card type in the deck is as follows:

Label	1	2	3	4	5	6	7	8
Quantity	5	2	2	2	2	1	1	1

The game is played in a series of rounds. The winner(s) of each round gets one point, the game ends when one or more players reach four points.

A round is played as follows. At the beginning of each round, the first card is taken from the deck and set aside. Next, one card is given to each player in turn. Beginning with Player A, each player takes their turn:

1. They are given an additional card from the deck.
2. They then choose one of the two cards they are holding to discard, they keep the other card.
3. This might result in one of the players being eliminated from the round (they can play again at the start of the next round).

The round ends when there are no cards left in the deck or when there is only one player left.

What the cards do

Most cards have an effect when discarded.

8. If a player ever discards this card, they are out of the round immediately.
7. This card doesn't do anything when discarded. However, it must be discarded in preference to 6 or 5. That is, you can't keep 7 and throw out 6 or 5.
6. When this card is discarded, by you, *during your turn*, you pick (target) another player. The two of you swap cards (each of you only have one card at this point).
5. When this card is discarded [during your turn], you pick (target) a player (you are allowed to target yourself). The targeted player must discard the card they are holding and get a new one from the deck. If the deck is empty, they are given the card which was put aside at the beginning of the round.
4. When this card is discarded, the player can not be targeted until the start of their next turn.
3. When this card is discarded [during your turn], you target another player. The two of you compare cards, if one of you has a lower card, they are out of the round.
2. This card has no effect when discarded.
1. When this card is discarded [during your turn], target another player and guess what they are holding. If you guess correctly, they are out of the round. If not, nothing happens. You are not permitted to guess that they are holding 1.

Note, only 4 and 8 have any effect if discarded when it is not your turn (for example if you are forced to discard because someone else played a 5). Also, 5 is the only card needing a target which allows you to target yourself. It is possible that you play a card which needs a target but no target is available. For example if there are only two players still in the game, and the other player just discarded 4. In this case you can still play the card but you need to specify that there is no target.

How your player will work

Your player will choose what to play based on the following rules (in order until one tells you what to do):

1. If you hold 7 and either 5 or 6, then discard 7.
2. Choose the lower of the two cards.
3. If the card needs a target, choose the first unprotected player starting with the player after you. If there is no unprotected target, you must target yourself if the card is a 5 or target nobody for any other card.
4. If the card is a 1 (and hence needs a guess), choose the highest card which has not been played yet. So for example if 8, 7, 6, 5, 4, 4, 3 have been played, then you would choose 5 (since there is still one 5 out there somewhere). If you are holding the other 5, this does not alter the logic.

If the player receives a message it does not understand from the hub, it should close down with the relevant exit status.

Messages

Players will be be represented by upper case letters. So, in a four player game, the players will be (in order) A,B,C,D. Where it is necessary to indicate "no player"/"noone", use '-'.

Cards are represented by the characters '1'...'8'. Where it is necessary to indicate "no card", use '-'.

Message from player to hub

The player will only send one type of message to the hub.

c_1pc_2

Where c_1 is the card being played, p is the target player and c_2 is the card being guessed. For example:

```
1B5
```

Messages from hub to player

Message	**Params**	**Meaning**
newround c	c is a card ie: $c \in \{'1' \ldots '8'\}$	Start a new round. Your card is c. eg: `newround 4`
yourturn c	c is a card	It is your turn, your additional card is c. `yourturn 3`

thishappened $p_1 c_1 p_2 c_2 / p_3 c_3 p_4$	p_1 is a player
	c_1 is a card
	p_2, p_3, p_4 are either a player or '-' for nobody
	c_2, c_3 are either a card or '-' for no card
		Reporting a move in the game.
		Player p_1 played c_1 targeted at player p_2. Player p_1 guessed that player p_2 holds card c_2. This resulted in player p_3 dropping card c_3. It also resulted in player p_4 being eliminated.
		Eg: `thishappened A3C-/C5C`
		Player A played card 3 targeted at Player C (there was no guess since card 3 doesn't use one). As a result of this, Player 3 dropped card 5. Player 3 was out of the round.
		`thishappened B4--/---`
		Player B played card 4 (There was no target or guess. Noone dropped anything and noone was eliminated).
		`thishappened D1B3/B3B`
		Player D played card 1 targetting player B and guessing 3. Player B dropped card 3 and was eliminated.

replace c	c is a card	Replace the card you are holding with c. `replace 3`
scores i j ...	All params are non-negative integers (space separated).	Gives the updated scores for each player. (Also indicates the end of a round). `scores 3 3 2 1`
gameover		Indicates the end of the game. No more contact with the hub should be expected. The player should close down. `gameover`

Player output

The first thing `player` does when it starts up is to print a single `-` character to `stdout`. The hub will read and discard this `-` character. Everything after that which is sent to `stdout` will be interpreted as messages to the hub. Do not send any extra output to `stdout`.

`stderr` should be used in the following way:

- If there are errors processing commandline arguments, then print the message to `stderr` and exit with the relevant status.
- When a message (eg XYZ) is received from the hub, print the following to `stderr`

`From hub:XYZ`
Only print the first 21 characters of the message (followed by a newline). Then print status information to `stderr`. If the message is processed successfully, then print another set of status information. If the message was not processed successfully, end the game with the relevant exit status. If the message is a valid `yourturn` message, then an additional set of status information should be printed after the new card is added to your hand but before any card is played.

- When a message (eg XYZ) is sent to the hub, print the following to `stderr`
  `To hub:XYZ`

The status information will be as follows. A line for each player:
`LP:CARDS`
where

- L — is the letter for the player.
- P — is '-' if the player is out of the round, '*' if the player is protected (ie they last discarded a 4) or ' ' otherwise.
- CARDS — are the cards they have played

After this, print
`You are holding:XY`
where X and Y are the cards you are holding. If you are only holding one card, print `-` as the second card. If you have been eliminated, then you will not be holding any cards (so print `--` instead). If it is your turn, the second card will be the one you have just received. For example (in a three player game):

```
A :1346
B*:1124
C-:1278
You are holding:31
```

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

When to update state

Typically, in a system like this, you would wait until the hub approves your move before you update your state information. However, this specification does not include any mechanism for the hub to inform the player that it doesn't like a move. So in this case, update your hand (and the cards **you** have played) when you send each move to the hub. Update information about other players when you receive a `thishappened` message from the hub.

If you learn from a `thishappened` message that you have been knocked out of the round, you should set your hand to `--`. That is, once you have been knocked out, your player should not be holding any cards.

Deck file

The deck file will consist of a number of '`\n`' terminated lines. Each line stores an ordering for the cards (the same cards will be present in each line, just rearranged). eg:

```
1234567811112345
5432154321678111
```

The decks will be used in turn (one per round). If more decks are required than were in the file, then go back to the first deck.

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`.

- A single line when `thishappened` messages are sent to players. Parts in [ ] are optional and should only be output if relevant. ? should be replaced with the relevant information:

  ```
  Player ? discarded ?[ aimed at ?[ guessing ?]].[ This forced ? to discard ?.][ ? was out.]
  ```

  For example:

  ```
  Player A discarded 3 aimed at B. This forced A to discard 6.  A was out.
  Player B discarded 1 aimed at C guessing 8.
  Player C discarded 5 aimed at B. This forced B to discard 7.
  Player D discarded 4.
  ```

- At the end of each round, display a message:

  ```
  Round winner(s) holding ?: ?
  ```

  Giving the highest card left and the player(s) holding it. For example:

  ```
  Round winner(s) holding 8:  D
  Round winner(s) holding 4:  C D
  ```

- At the end of the game (assuming no early ends). 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: hub deckfile prog1 prog2 [prog3 [prog4]]`
Unable to open decks file for reading	2	`Unable to access deckfile`
Contents of the decks file are invalid	3	`Error reading deck`
There was an error starting and piping to a player process	4	`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.	5	`Player quit`
One of the players has sent an invalid message	6	`Invalid message received from player`
The hub recieved SIGINT	7	`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.

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 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.

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 game over message (2 marks)
- (Player) Correct choice for inital 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 of exec (2 marks)
- Correctly handle players which close early (4 marks)
- Correctly handle games using a single deck (6 marks)
- Correctly handle invalid messages (2 marks)
- Play complete games with 2 players (5 marks)
- Play complete games with 3 and four players (5 marks)
- Correctly cleanup subprocesses on SIGINT (2 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. 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.

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.

6. Remember that there are two ways a round can end.

6b. You are not permitted to use any of the following functions in this assignment.

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

7. You may assume that only 8bit characters are in use[no unicode].
8. When hub exits (under program control or in response to `SIGINT`), it should not leave any child processes running (you can use `SIGKILL`).
9. All tab characters will be treated as being (up to 8 spaces).
10. You may not use any `#pragma` in this assignment.
11. 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 "`
12. 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.
13. You will need to do something with SIGPIPE.
14. The `-` character in the client should be printed before any argument checking and error messages.
15. Valid messages contain no leading, trailing or embedded spaces. Messages must be exactly the correct length.
16. All decks must be read into memory before the game starts.
17. All messages except the initial `-` are followed by newlines.
18. Version 2.0 of the style guide will be released soon. When it is, you will be expected to follow it.

Changes 3 → 3.1

- Changed hub→player message output so that only the first 21 characters (followed by a newline).
- Note: the player won't actually do anything with scores messages other than validating them and printing them out.
- What is a valid message? A valid message (from hub→player) is one which has the correct format and doesn't refer to cards or players which do not exist. Eg. player D in a two player game.

  Valid messages do not require the player to check whether what the hub has sent them is sane or consistent apart from that. So for example, the player does not need to check if the hub played five instances of #3 when there are only supposed to be two #3s.
- *What order should the messages from the hub be sent in?* When the hub receives a valid move from a player, it should send any specific messages related to that move first. After that, it should send the `thishappened` to all players. So a #5 will result in `replace`, then `thishappened`.

Changes 3.1 → 3.2

- Modified compilation instructions not to refer to concepts from last year.
- Separated Item 6 into two points for clarity.
- Modified the wording of Item 9 slightly.
- Added actions for SIGINT.
- Modified player output in the case where the message is `yourturn`.

Changes 3.2 → 3.3

- Updated change state instructions to include clearing hand when you are eliminated from a round.
- Valid messages:
  - you are not required to check that messages are in a correct sequence.
  - You are expected to check that scores are integers between 0 and 4 inclusive.
- Minor modification to marking scheme.

Changes 3.3 → 3.4

- Some minor gramatical fixes.
- Removed hub reporting of player exit statuses. (Players still need to exit with correct status and messages).
- tweak marking scheme
- Modified Point 15 in notes. Messages should not have trailing chars.