Prepared for s4392687. Do not distribute.

The University of Queensland
School of Information Technology and Electrical Engineering
Semester Two, 2016
CSSE2310 / CSSE7231 - Assignment 3
**Due: 11:10pm 30th September, 2016**
Marks: 50
Weighting: 25% of your overall assignment mark (CSSE2310)
Revision 3.1 (fixed due date from 3.0).

# Introduction

In this assignment, you will write two types of programs `C99` programs to play a game "King of St Lucia". A separate document describing the rules will be provided. The first type of program (players) will listen on their `stdin` for information about the game and give their moves to `stdout`. These types of program will send information about the state of the game to `stderr`. The second program (`stlucia` — also known as the *hub*) will start a number of processes to be players and communicate with them via pipes. The stlucia will be responsible for running the game (sending information to players; processing their moves and determining the score). The stlucia will also ensure that, information sent to `stderr` by the players, is discarded.

The idea is that you will produce a number of player programs (with different names) which implement different playing strategies. However all players will probably share a large amount of common code (with the other players *you* write).

*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 2.0.3) 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 - players

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 maximum number of players permitted in a game will be 26.

# Invocation - stlucia

The parameters to start the hub are:

- The name of a file containing a list of dice rolls 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 players.

For example: `./stlucia ex.rolls 15 ./typeA ./typeA ./typeA`
Would start a game with 3 players (each running `./typeA`) and using the rolls contained in `ex.rolls`.The game will end when at least one player has 15 or more points.

## Representations

Whenever rolls are represented as strings, use the following symbols: 1, 2, 3, H, A, P. For ordering purposes: $1 < 2 < 3 < H < A < P$. Roll files will contain the above characters and newlines in any combination (skip over newlines). Any other character in a roll file is an error.Your program should read the entire roll file into memory when it starts rather than trying to detect problems later.When all rolls are exhausted, go back to the beginning.

## How your players will work

EAIT

Rerolls: If they have more than 2 of the same number (1,2 or 3), they will keep them. If they have less than 6 health, they will keep any $H$ they roll. Apart from that, they will reroll as many dice as possible as many times as possible.

Retreat: This player will retreat from StLucia if they have less than 5 health, otherwise they will stay.

SCIENCE

Rerolls: If they have less than 5 health, they will keep any $H$ they get and reroll everything else. If they have 5 or more health, they will keep any A they get and reroll everthing else.

Retreat: This player will retreat from StLucia immediately.

HABS

Rerolls: If they have less than 5 health, then they will reroll any As they have. Apart from that, they will not reroll.

Retreat: This player will not retreat unless they have less than 4 health. If there is only one other player left, they will never retreat.

HASS

Rerolls: If they are in StLucia, they will never reroll Ps. They will reroll everything else. If they are not in StLucia, they will reroll As unless the As they have rolled are enough to eliminate the player in StLucia. They will reroll everything else.

Retreat: This player will not retreat from StLucia.

MABS

Rerolls: If they are not in StLucia, they will always keep Hs and 3s. Apart from this, they will reroll everything else. If they are in St Lucia, they will keep 3s and As. Apart from this, they will reroll everything else.

Retreat: This player will retreat from StLucia immediately.

# Messages

**Messages from stlucia to player**

| Message | Params | Meaning |
|---|---|---|
| turn $d_1d_2d_3d_4d_5d_6$ | $d_i$ are dice values arranged smallest to largest. | It is your turn and here is your roll |
| rerolled $d_1d_2d_3d_4d_5d_6$ | arranged in increasing order. | Used to communicate the result of a reroll |
| rolled p $d_1d_2d_3d_4d_5d_6$ | $p$ is the letter of the player | used to inform other players of Player $p$'s final dice |
| points p v | $p$ is the letter of the player $v$ is a number of points | inform all players when a player is awarded points |
| attacks p v in/out | | inform all players that Player $p$ is attacking for $v$ points of damage either **in**to or **out** of StLucia. |
| eliminated p | | Inform all players that Player $p$ is out of the game.<br>Player $p$ should close down at this stage. |
| claim p | | Inform all players that Player $p$ has claimed StLucia. |
| stay? | | Ask the player in StLucia who has just been attacked if they wish to leave |
| winner p | | Inform all remaining players that Player $p$ has won |
| shutdown | | Inform all players that it is shutting down (used if stlucia is shutting down due to a signal) |

For example Player D might see:

```
rolled A 123HHH
rolled B 122AAA
claim B
points B 1
rolled C 23HAA
attacks C 2 in
claim C
points C 1
turn 112HPP
```

**Messages from player to stlucia**

| Message | Params | Meaning |
|---|---|---|
| keepall | | inform stlucia that you do not wish to reroll any of your dice |
| reroll $d_1 \ldots d_k$ | Inform server which dice you wish to reroll. You would expect a `rerolled` in response. | |
| stay | | inform the server you wish to remain in StLucia (in response to a `stay?` message. |
| go | | inform the server you wish to leave StLucia (in response to a `stay?` message. |

## Player output to stderr

When a message (eg XYZ) arrives from the hub, print the following to `stderr`
`From hub:XYZ`
After the message has been processed, print any player specific output described in the strategy section.
The player program will also generate some error messages which are to be sent to `stderr`.

## Order of processing during a turn

After a player has decided on their dice, communications and checks should happen in the following order:

1. Inform other players what was rolled.
2. Healing (stderr reports healing amount as 0 if there were H dice but the player is already at maximum health).
3. Attacks are processed and damage reported.
4. New player claims StLucia
5. Points for the turn are reported
6. Player eliminations reported
7. Test for game over

## Hub output

The stlucia 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 player's dice are finalised, print:
  `Player ? rolled ??????`
- When a player scores points, print:
  `Player ? scored ? for a total of ?`

- When a player takes damage (this amount can not be larger than their remaining health), print:
  `Player ? took ? damage, health is now ?`
- When a player heals (this amount could be zero if they are already at 10 health) print:
  `Player ? healed ?, health is now ?`
- When a player enters StLucia, print:
  `Player ? claimed StLucia`
- When a winner is declared, print:
  `Player ? wins`

# 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 my_id` |
| Invalid number of players | 2 | `Invalid player count` |
| Invalid player ID | 3 | `Invalid player ID` |
| Pipe from stlucia closed unexpectedly (i.e. before a winner message was received) | 4 | `Unexpectedly lost contact with StLucia` |
| Invalid message from stlucia | 5 | `Bad message from StLucia` |

## Exit status for stlucia

All messages to be printed to `stderr`.

| Condition | Exit | Message |
|---|---|---|
| Normal exit due to game over (recieved `winner` or `eliminated` | 0 | |
| Wrong number of arguments | 1 | `Usage: stlucia rollfile winscore prog1 prog2 [prog3 [prog4]]` |
| Winscore is not a positive integer | 2 | `Invalid score` |
| Unable to open rolls file for reading | 3 | `Unable to access rollfile` |
| Contents of the rolls file are invalid | 4 | `Error reading rolls` |
| 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 action | 8 | `Invalid request by player` |
| The stlucia 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 stlucia

Whether the stlucia 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 all terminated within 2 seconds of the hub sending the `winner` / `eliminated` / `shutdown` 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` [1]. 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,

[1] That is, `https://source.eait.uq.edu.au/svn/csse2310-$USER/trunk/ass3`

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.

- For *EAIT* player
  - argument checking (2 marks)
  - correctly handles early loss of stlucia and ending messages (2 marks)
  - respond correctly to their first turn (2 marks)
  - (Player) Correctly handle complete game (4 marks)
  - (Player) Detect invalid messages (2 marks)
- For StLucia
  - argument checking (2 marks)
  - Detect failure to start players (2 marks)
  - Correctly handle players which close early (3 marks)
  - Correctly handle 2 player games with *EAIT* players (4 marks)
- play complete games with 3 *science* players. (2 marks)
- play complete games with 4 *HABS* players. (2 marks)
- play complete games with 5 *MABS* players. (2 marks)
- play complete games with a mixture of player types. (12 marks)

### 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.

## Notes and Addenda:

1. Please note that this assignment requires compliance with an updated style guide.
2. 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.
3. Start early.
4. Write simple programs to try out fork(), exec() and pipe().
5. Be sure to test on moss.
6. 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()`

7. You may not use any `#pragma` in this assignment.
8. 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 "`
9. 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.
10. You will need to do something with SIGPIPE.
11. Valid messages contain no leading, trailing or embedded spaces. Messages must be exactly the correct length.
12. Make a separate player to test stlucia against badly behaved players rather than breaking an existing one.
13. You should only report on the exit statuses of the players if all players started successfully.

14. `struct`s and `enum`s are your friends use them.

15. Just because communication is required to be in a particular format, does not mean that you must store information internally in that form. Instead, convert from the external format to the internal format as soon as possible and convert to the external form as late as possible.

16. There will be a number of different communication channels in a complete system. Make sure that you can monitor any of the channels.