The University of Queensland
School of Information Technology and Electrical Engineering
Semester Two, 2013
CSSE2310 / CSSE7231 - Assignment 4
Due: 11:10pm 25 October, 2013
Marks: 50
Weighting: 25% of your overall assignment mark (CSSE2310)
Revision 4.2

Introduction

Your task is to write a networked card game in c99. This will require two programs: a client (called `client499`) and a server (called `serv499`).

This assignment will require the use of threads, tcp networking and thread-safety. Your assignment submission must comply with the C style guide available on the course website. As with Assignment 3, tabs are allowed but all source code will be run through `expand` before marking.

This assignment is designed to be completed in stages. Good design and thinking about the final target would help but it is important to *design*, implement and *test* each stage before moving on to the next one. Do not attempt the whole thing in one go.

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`

In this course 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.**

The Game — 499

A game requires four players in positions P1, P2, P3 and P4 (respectively). The positions will be assigned by giving P1 to the player with the earliest name in lexicographic order. For example, if Zack, Henri, André and Walter connect to the game, then André will be P1, Henri will be P2, Walter will be P3 and Zach will be P4. Player names are not required to be unique, but if multiple players with the same name are in a game, the order they are placed in is up to you.

The game is played in teams, so P1 and P3 form one team, while P2 and P4 form the other.

The deck for the game consists of four suits **S**, **D**, **C**, **H**. Each suit has 13 ranks **A, K, Q, J, T, 9, 8, 7, 6, 5, 4, 3, 2** (highest to lowest). Individual cards are written rank first. For example: **AD, 9S, JH**.

Some definitions:

Play a trick where each player (in order) contributes one card.

Win a trick to play a trick and contribute the highest card in it.

Lead to play the first card in a trick. The suit of the card determines which suit the other players must play if they can. To have the lead means that you will play first.

Hand a set of 13 cards which are then played in 13 tricks.

Trumps one of the suits which beats any card not of that suit. For example: if **C** is the trump suit, then **3C** would beat **9H**, **AD** and **2C** but not **4C**. The trump suit is determined by bidding (see later).

Following suit To play a card which has the same suit as the card which was lead. A card which does not follow suit and does not belong to the trump suit can not win the trick.

Bowers this game doesn't use them. If you don't already know what they are, it isn't important.

Misere are you making up words?

Game flow

The game has the following stages:

1. Both teams start with a score of 0.
2. Play a hand:
   (a) Each player is dealt a hand of 13 cards.
   (b) The players bid to determine trumps.
   (c) (Tricks) Each player plays one card and the player with the highest card wins the trick.
   (d) Repeat 2c until all cards have been played.
   (e) If the team that made the highest bid, succeeded in that bid, then the points for the bid are added to their score. If they failed in their bid, the points for the bid are subtracted from their score.
3. If at least one of the teams has more than 499 points, they win. If at least one of the teams has less than −499 points, the other team wins. Otherwise, go to 2.

Bidding

This is the process of determining the trump suit. A bid consists of a number and a suit (or the special bid 'PP' which indicates pass). So a bid of **6H** means that if `H` was trumps, the team thinks they will win at least six tricks. Each bid has points attached to it. See table below.

Suit	4	5	6	7	8	9
S	20	70	120	170	220	270
C	30	80	130	180	230	280
D	40	90	140	190	240	290
H	50	100	150	200	250	300

Each player in turn will be asked for their bid (the first player is not allowed to pass). They must then make a bid which is worth more points than the previous one or they must pass. Once they have passed, they take no further part in the bidding for this hand. Bidding continues in order (amongst the players who haven't passed) until only one player is left or until a player bids **9H**. That player's last bid is the winner.
For example,

```
P1 : 4H
P2 : 4C   Not a legal bid since it is lower value than 2H
P2 : 5C
P3 : PP   They have passed so don't take part anymore
P4 : 6C
P1 : 6H
P2 : PP
P4 : 7S   Skipped P3 since they had already passed
P1 : PP
```

P4 made the last bid so **7S** is the winning bid and **S** is trumps.

Protocol

All protocol messages from the server to the client consist of single lines of text. The first character in the line determines the type of the message.

Character	Purpose	Format	Action
M	Send an "info" message from server to client	`M%s`	Display `Info: %s`
H	Give player their hand	`H%s`	
B	Make a bid	`B%c%c` or `B`	Client ask user for a bid (higher than the one specified in the message). The second form is for the first player
L	Lead a card	`L`	Client ask user for a card to play
P	Play a card following suit if possible	`P%c`	Client ask user for a card. (The %c indicates the suit)
A	Informs client that their lead or play was accepted	`A`	client removes card from their hand
T	Informs clients of winning bid	`T%c%c`	
O	Game is over	`O`	

Messages from the client to the server are `%c%c\n` indicating the card to play or the bid to make. If the client sends an invalid message to the server, the server will treat that message like it would an invalid bid or play (that is it will ask the client again).

Client Interface

The client ("`client499`" from now on) takes the following arguments.

1. The name of the player.
2. The name of the game to connect to.
3. The port the game server is listening on.
4. The hostname to connect to. This parameter is optional and if not given, it should default to the local machine.

There is no requirement that player names be unique. Two clients could connect to a game using the same name. All ports must be integers 1 <=port<= 65535. Error messages and exit stati are given below:

Condition	**Status**	**Message to stderr**
Incorrect number of arguments	1	Usage: client499 name game port [host]
Invalid port# or empty string for name or game	4	Invalid Arguments.
Can't connect to server	2	Bad Server.
Server doesn't follow protocol	6	Protocol Error.
User eof	7	User Quit.
System error	8	System Error.
It's all good	0	

Unexpected disconnects by from the server will be treated as the server not following protocol.

At the beginning of each hand, the client will display the new hand. Whenever the client displays the current hand, it should do so as follows:

- `S:` followed by the ranks of any **S** cards (in decreasing order) separated by spaces.
- `C:` followed by the ranks of any **C** cards (in decreasing order) separated by spaces.
- `D:` followed by the ranks of any **D** cards (in decreasing order) separated by spaces.
- `H:` followed by the ranks of any **H** cards (in decreasing order) separated by spaces.

So a hand might be shown as:

```
S: A K Q J T
C: 9 8 7 6 2
D:
H: 8 7 6
```

In successive tricks, cards that have been played should be removed. Do not remove a card until the server has sent an "A" message.
Next the prompt:

```
[XY] - Bid (or pass)>
```

should be displayed (where XY is the previous bid) or

```
Bid>
```

If you are making the first bid. Note that the first player is not allowed to pass. If the bid the user enters is not valid, then redisplay the prompt and wait for a new bid.

Once the bidding is complete, the client will wait for the serve to ask it to play. If it has the lead, then the hand should be displayed followed by the prompt:

```
Lead>
```

If the client does not have the lead, then it will display the hand followed by the prompt:

```
[Y] play>
```

where Y is the suit that was lead. Keep displaying the hand and prompt until a valid card is chosen. *Note that for a card to be valid, it must be in your hand and belong to the lead suit (if possible).*

This sequence repeats until the server advises the game is over. At that point, the client should exit with status 0.

There is one other important aspect of the client interface: Messages. At various times, the client will expect a message from the server. For example, a new hand. At any point where a message is expected, the server can send an info message. When this happens, display the message

```
Info: text of message
```

and continue to wait for another message. This mechanism will be used by the server for various things including: to advise the user of scores and the results of bidding.

Server

The server ("`serv499`" from now on) takes the following arguments:

1. port number to listen on
2. Greeting message
3. the name of a text file storing a shuffled deck of cards

All ports must be integers 1 <=port<= 65535. Error messages and exit stati are given below.

Condition	**Status**	**Message to stderr**
Incorrect number of arguments	1	Usage: serv499 port greeting deck
Invalid port#	4	Invalid Port
Error listening on port	5	Port Error
Error reading deck file (file contains bad data) or can't open it at all	6	Deck Error
System error	8	System Error
It's all good	0	

The deckfile will consist of decks each represented by a line of 104 characters. It will indicate the order the cards are to be dealt. The first card will be dealt to P1, the second to P2, the third to P3, the forth to P4, the fifth to P1 etc. If there are multiple lines in the file, then the first line will be used for the first hand and successive lines will be used for successive hands. When there are no more lines in the file, the server should start with the first deck again. Note that the server must check to see that the cards it is reading for a deck are valid cards (and that there are enough of them). It does not however need to check for duplicate/missing cards. Also, all decks must be loaded before the server listens for connections.

The server will wait for clients to connect. It sends a greeting and waits for the client to send their name and the name of the game they wish to play in. When a game fills up, the server should start a thread to run that game and continue listening for more connections. See Figure 1. For the sequence of actions of a game thread, See Figure 2.

At the beginning of each hand, **P1** will be the first player to bid. The client who wins the bid gets the lead in the initial trick of that hand. For successive tricks of a hand, the player that won the previous trick gets to lead.

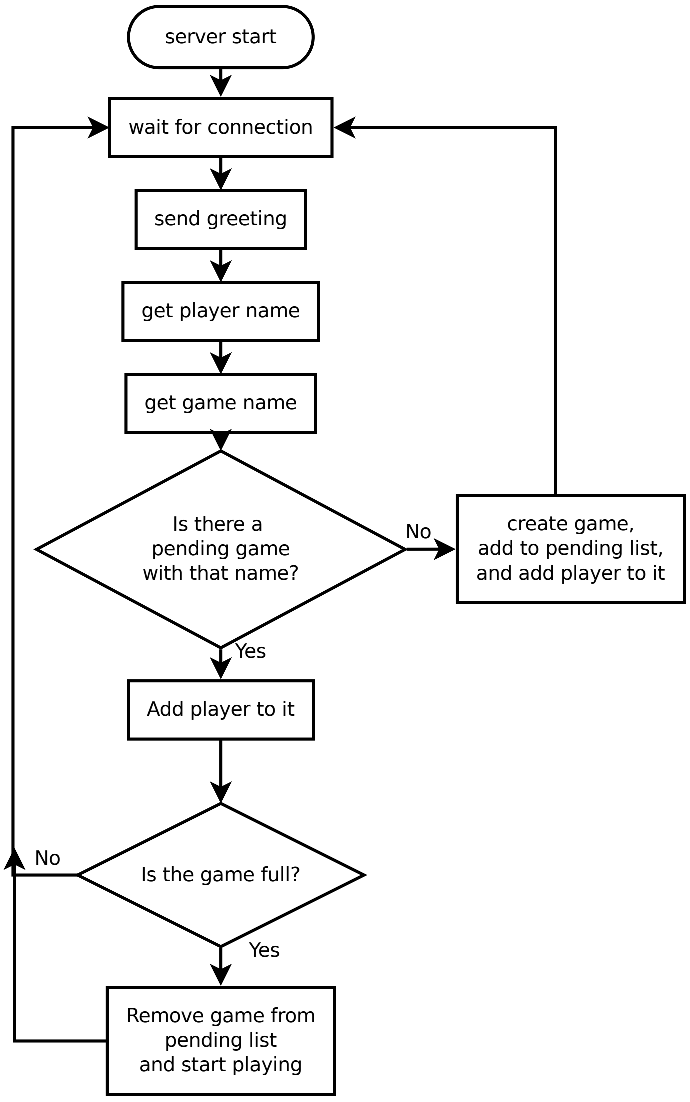
server start
wait for connection
send greeting
get player name
get game name
Is there a pending game with that name?
No
create game, add to pending list, and add player to it
Yes
Add player to it
Is the game full?
No
Yes
Remove game from pending list and start playing

Figure 1: Server activities on connection

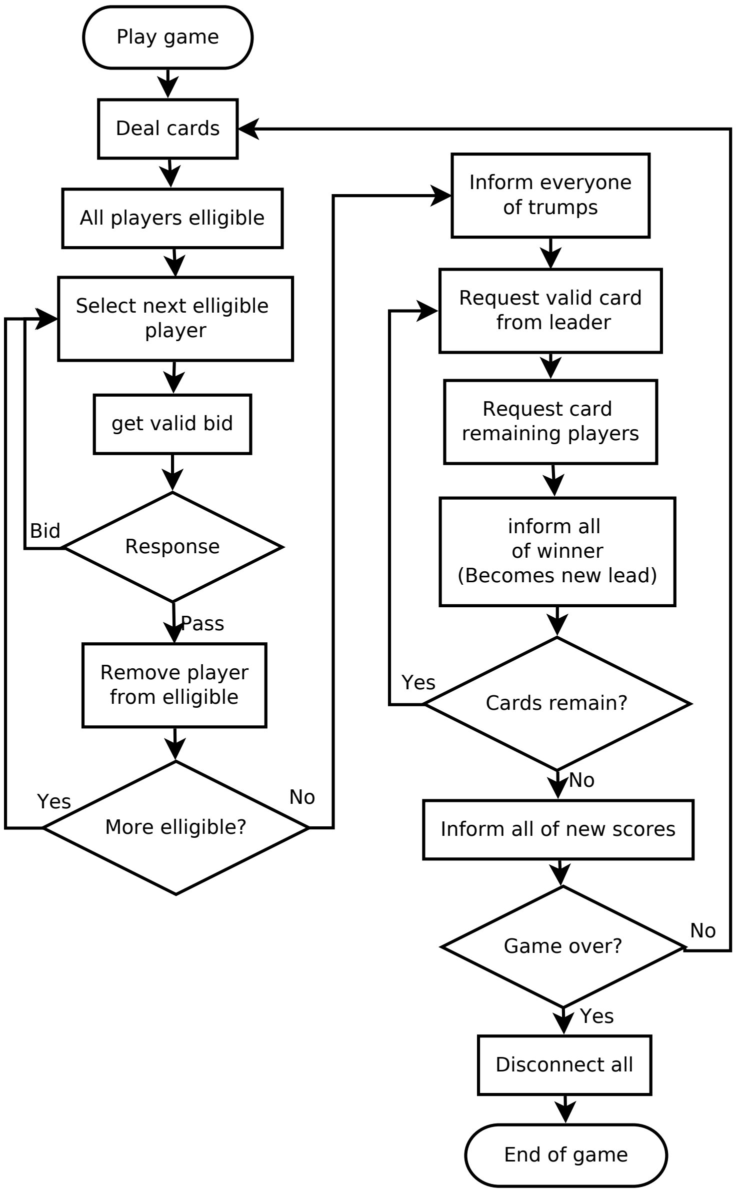
Play game
Deal cards
All players elligible
Select next elligible player
get valid bid
Bid
Response
Pass
Remove player from elligible
Yes
More elligible?
No
Inform everyone of trumps
Request valid card from leader
Request card remaining players
inform all of winner (Becomes new lead)
Yes
Cards remain?
No
Inform all of new scores
Game over?
No
Yes
Disconnect all
End of game

Figure 2: Flow chart for server thread managing a single game.

The server sends informational messages to clients using `M` in the following circumstances:

- Send the welcome message to a client when it connects.
- The game begins (Send two messages in order to all clients)
  - First message text is: `Team1:  %s, %s` (substitute P1, P3)
  - Second message text is: `Team2:  %s, %s` (substitute P2, P4)
- When another player (legally) passes instead of bidding. `%s passes`.
- When another player makes a legal bid. `%s bids %c%c` for example: Tom bids 6H
- When a player wins a trick. `%s won`
- At the end of a hand. `Team 1=%d, Team 2=%d` for example: Team 1=50, Team 2=140
- When one team wins. `Winner is Team %c` for example: Winner is Team 2
- When another player disconnects early. `%s` disconnected early for example: Jim disconnected early
- When another player (legally) plays a card. Message text is: `%s plays %c%c` (substitute player name and card).

Information messages should be sent to clients before the next move/lead/bid message. No messages should be sent to clients after they have been sent a over message (**O**). On recieving **O**, the client will disconnect.

Client disconnections will be handled as follows. Regardless of when you detect the disconnection, it should not be acted upon until the server needs to read from that client. That is, when a read call involving that client fails. So if the server is waiting for P3 to send back a move and P1 disconnects first, then P3 disconnects, P3 will be regarded as the disconnecting player. Other clients will be sent the message given above, then a game over message. After that, all clients in the game will be disconnected. Note that this happens even if clients disconnect before the game starts. Their positions do not get filled by later connections.

Compilation

Your code (both programs) must compile with the command:
`make`

Each individual file must compile with at least `-Wall -pedantic -std=gnu99`. You may of course use additional flags (eg `-pthread`) 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 one of the executables are not created by make, the you will receive 0 marks for functionality involving that executable. Any code without academic merit will be removed from your program before compilation is attempted (and if compilation fails, you will receive 0 marks for functionality). If your code produces warnings (as opposed to errors), then you will lose style marks (see later).

Your solution must not invoke other programs. Your solution must not use non-standard headers/libraries.

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.

- client - startup errors handled correctly. [4 marks]
- server - startup errors handled correctly. [4 marks]
- client - plays a single game. [6 marks]
- play a single game with one deck. [4 marks]
- play a single game with one deck with invalid bids. [2 marks]
- play a single game with one deck with invalid plays. [2 marks]
- correctly handles early disconnect. [2 marks]
- play mutiple games sequentially with one deck. [5 marks]
- play multiple games sequentially with multiple decks. [3 marks]
- play a number of concurrent games with one deck. [6 marks]
- play a number of concurrent games with mulitple decks. [4 marks]

Note that later categories may contain earlier categories. For example, "multiple games" categories assume that you can handle early disconnects.

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 version 1.7 of the 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 suggestions

1. Work out what text a hypothetical server would send, then use `netcat` as a server and pass that text to your client.
2. Begin your server to run a single game at a time but put as much code/data in functions and structs as possible to that you can do multiple games later.
3. Only use a single deck initially.
4. when you come to work on your server, I suggest an order like this:
   - Argument checking
   - Thread function to manage one game.
   - Multi-tasking.

Changes from 4.2 → 4.1

(a) `M` Messages relating to bids and lead/play should be sent to all players **except** the one that caused the message. That is, you do not hear messages about your own plays. Announcements about winning tricks, still go to all players.

(b) The client must receive the inital greeting `M` message before any other messages. Apart from this, no M message should be "required" by the client. However, it should handle any M message correctly regardless of when it arrives.

(c) All client prompts with >should follow it with a space. eg: `Lead> ` is followed by a space.

(d) If the client receives an 'O' message at any time, it should end the game successfully.

Changes from $4.0 \rightarrow 4.1$

(a) Made text involving “A” messages a bit clearer.

(b) Clarify when hands are supposed to be displayed.

(c) Q: When should input such as bids and plays be verified?
A: Ideally, both the server and the client should do their own checks.

(d) Your client should send game and name info as soon as you connect. Similarly, the server should send the greeting as soon as each client connects. That is, both sides should send their initial info before attempting to read info.

(e) Capitalisation for server error message fixed.

Changes from $\beta \rightarrow 4.0$

(a) Changed hand display to not repeat suits.

(b) Added marking scheme

(c) Note: The client informs the server of its name and requested game name by sending them to the server on individual lines. Eg: Barry\nEndersGame\n

Changes from $\alpha \rightarrow \beta$

(a) Removed remaining references to clients being able to send messages to other clients. ie clients never send M messages. Chat messages have been renamed to Info messages to avoid confusion.

(b) Correct diagrams so player name is sent before game name.

(c) Clarified what verification the server needs to do when loading decks.

(d) If the hostname lookup fails, that should be reported in the same way a connection failure would be.

(e) The game only takes one deckfile, but it can contain more than one deck (one per line).

(f) Modified client error table.

(g) Note that all players receive the **T** message.

(h) Corrected a number of errors in example bidding.

(i) Changed error message for being unable to open the deck file for reading.