

25

Procedural programming assignment

supporting documentation

Darren Butler | Software Development | 8th January 8, 2019

Table of Contents

[Data & Structures 2](#_Toc534736835)

[Cards 2](#_Toc534736836)

[Deck 3](#_Toc534736837)

[Played Cards 3](#_Toc534736838)

[Trump 3](#_Toc534736839)

[Lead 3](#_Toc534736840)

[Players 3](#_Toc534736841)

[Conclusions 4](#_Toc534736842)

[Functions 4](#_Toc534736843)

[calculate rank 4](#_Toc534736844)

[Calculate suit 5](#_Toc534736845)

[Get suit character 6](#_Toc534736846)

[move card 7](#_Toc534736847)

[copy card 7](#_Toc534736848)

[print card 8](#_Toc534736849)

[calculate ranking 9](#_Toc534736850)

[save game 11](#_Toc534736851)

[Gameplay Code Synopsis 12](#_Toc534736852)

[Players Setup (1.1) 12](#_Toc534736853)

[Hand (2.0) 12](#_Toc534736854)

[initialize deck (2.1) 12](#_Toc534736855)

[Deal (2.2) 13](#_Toc534736856)

[Trump (2.3) 13](#_Toc534736857)

[Trick (3.0) 13](#_Toc534736858)

[Initialize Trick (3.1) 13](#_Toc534736859)

[Previous trick winner leads (3.2) 13](#_Toc534736860)

[play (4.0) 14](#_Toc534736861)

[display (4.1) 14](#_Toc534736862)

[Determine rob legality (4.2) 14](#_Toc534736863)

[card selection (4.3) 15](#_Toc534736864)

[legal renege validation (4.4) 15](#_Toc534736865)

[Check card ranking (4.5) 16](#_Toc534736866)

[award points to winner (4.6) 16](#_Toc534736867)

[save game 17](#_Toc534736868)

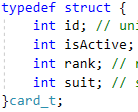
[load game 17](#_Toc534736869)

# Data & Structures

## Cards

For my project, I elected to use structs when dealing with cards. Each card in the game has 4 integers to represent important information about the card

* Id: Each card as a unique id, which denotes its position within the deck (0 – 51). This is important in game like 25 where the ace of hearts (id: 13) has a special role.
* Rank: The rank of the card (1-13) where 1 = Ace, 2 – 10 = numeric values, 11 = Jack, 12 = Queen and 13 = King. It is important to be able to compare the ranks of cards.
* Suit: Each card as a suit, where 0 = Spades, 1 = Hearts, 2 = Clubs, 3 = Diamonds. Like with rank, in the game 25 it is important to be able to compare the suit of cards.
* Active Status: Each card has a 1 or 0 to represent whether the card is active, for example, if I draw the ace 3 of hearts, I need to de-activate it in the deck (card\_t array) and then activate the card in the players hand (card\_t array). This is essential to dealing out unique cards.



### Deck

My program uses a struct array of cards (card\_t) to track the deck. At the start of a deal, the deck is initialized, and all cards are set to be active. As cards are drawn their active status is toggled.

### Played Cards

I used an struct array of cards (card\_t) to track cards played each round for display and comparison purposes.

### Trump

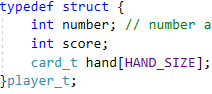
I use a card\_t type for storing the trump card. This is essential for later determining which player wins a trick, or if the dealer can rob or not.

### Lead

Like the trump, I used a card\_t type to store the lead card played each trick for later display and comparison.

## Players

A player is represented by a struct, player\_t, that contains integers to track the players number and score. Each player struct also contains and array of cards(card\_t) which is used to represent the players hand.

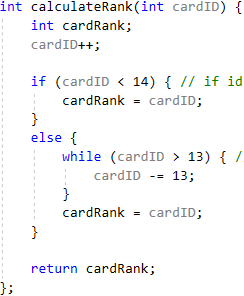


## Conclusions

Overall, I’m pleased with the data types & structs used in my application. Structs have proven to be a versatile approach when compared with other possible solutions (2-Dimensional Arrays or single integers).

# Functions

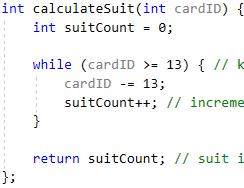
## calculate rank



This function takes in the id of a card and returns the card’s rank based on the id. It works by:

1. Incrementing the card id initially // so that for example card 0, the Ace of Spades will get a rank of 1
2. If the cards id is less than 14, then assign it’s rank to id // this is for the first suit of cards, spades
3. While the card id is greater than 13, keep decrementing the id, until you have a number less than 13 // this number is now the cards rank

## Calculate suit



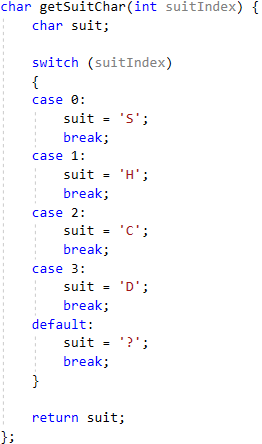
This functions in the id of a card and returns the card’s suit as an integer (0 = Spades, 1 = Hearts, 2 = Clubs, 3 = Diamonds)

1. Initialize suit count to 0
2. While the card id is greater or equal to 13, decrement the card id by 13, and increment suit count

The while loop will execute until the card id is less than or equal to 13, therefore its suit is the number of times you needed to decrement 13 to get to less than or equal to 13. A Spades card will be decremented 0 times, a diamond card id will need to be decremented 3 times.

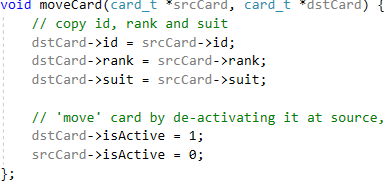
I have included a spreadsheet “Card Overview” which gives a better visual representation of how I managed card ids ranks and suits.

## Get suit character



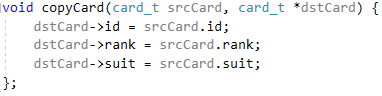
This function takes a suit index, as calculated using the previous functions, and returns a character to represent that suit. S = Spades, H = Hearts, C = Clubs, D = Diamonds.

## move card



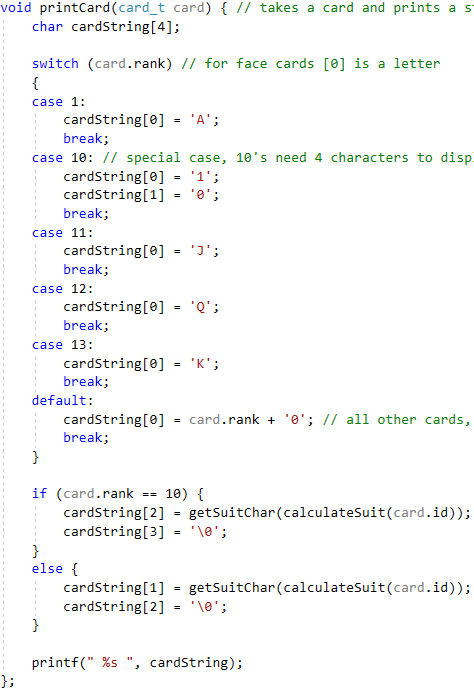
This function takes in the pointers to two cards, a source and destination card. It then copies all the values from source to the destination card and toggles the active status of both. This function requires the use of pointers to work. If you only passed two cards through, you would only change the values of the cards within the scope of the function and would have no effect on main. I engaged in independent research to get this function working (primarily YouTube tutorials) but doing so was well worth the time spent. As it allowed me to clean-up my code and reduce code re-use dramatically.

## copy card



This function is a simplified version of the move card function, where I did not want to alter the active status of the source or destination card. This was useful for cases like the lead card, where I wanted to copy the values of the played card to the lead, but not alter its active status.

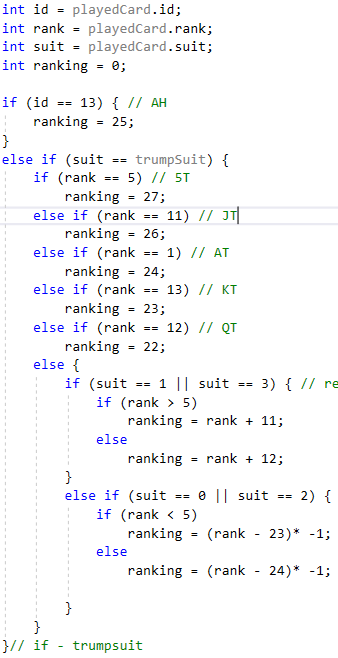
## print card



This function reads in a type card\_t and prints a string to represent that card. For example, a card, id=36, rank=11, suit=2 will be printed as “ JC “ – the jack of clubs. This function was invaluable as printing cards is frequently required.

## calculate ranking

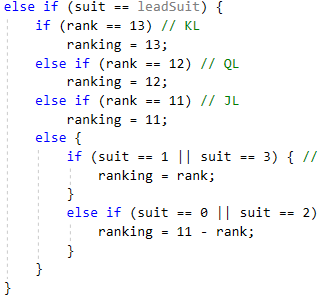
This function takes in a type card\_t, an int for the trump suit, and an int for the lead suit. It then returns and absolute ranking for that card in a trick. It is a longer than my other functions so I will break it down to two parts. In my supporting spreadsheet I have included a sheet “Absolute Rankings” which gives a visual representation of my ranking method. Essentially the best card in the game, the 5 of trump, has the highest possible ranking of 27. Then all other cards are less.



Firstly, I set the integers id, rank and suit to the values of the card passed to the function. I then initialize ranking to 0. I then enter an if statement.

1. If the card’s id is 13, assign it the rank of 25 // card id13 is the ace of hearts, the third best card in the game regardless of what the trump or lead suit are.
2. If the card played is of the same suit as the trump.
   1. If its rank is 5 assign it the ranking of 27 // 5 of trump
   2. If its rank is 11 assign it the ranking of 26 // Jack of trump
   3. So on for all the face cards
   4. If the trump suit is red, higher numeric cards get higher ranking
   5. If the trump suit is black, lower numeric cards get higher rankings

The second section is used when the card played is not of the trump suit. I rank it based on the lead suit.



As before assign ranking based on the face value or numeric value, “high in red, low in black”.

If the card is not of the trump or lead suit (and not an ace of hearts) it’s ranking is a 0. Such a card can never win. Because the first card played is the lead card, it will always start out with the highest ranking, but will be compared with subsequently played cards rankings.

## save game

This is a straight forward function that takes in all essential data from a game and writes it to txt file “savegame.txt” which can be later read in order to load a game. I believe it to be a self-explanatory function so I will not go into further detail here.

# Gameplay Code Synopsis

In this section I will briefly describe how each gameplay related code block works. The number preceding the title of a subheading refers to the block of code in my program, i.e. 1.1 – Players Setup is labeled in my code with the same title, for easy reference.

## Players Setup (1.1)

This code block is responsible for setting up the players aspect of the game. It works as follows:

* Prompt the user for number of players (within the range 2-6)
* For each player, set their number to be i+1, and their score to be 0

## Hand (2.0)

This section contains all gameplay aspects once the number of players has been determined. Before entering this section, I initialize high score to 0 and trick count to 1.

The whole section is enclosed in a do while loop that will keep running until either

1. Any player reaches a score of 25
2. The players decide to not play the next trick and a new deal is required

## initialize deck (2.1)

This block uses a for loop to cycle through each card in the card\_t array deck[]

1. Set id of card to i
2. Set the active status to 1
3. Call the calculate rank function and set its rank
4. Call the calculate suit function and set its suit

## Deal (2.2)

This block contains a set of nested for loops, that run for each player, 5 times (5 cards in a hand) to deal 5 cards to each player.

A do while loop is used to get a random number from 0 – 51, the active status of the card in deck[r] at the random position is checked.

If the card is inactive (i.e. it has already been drawn), then repeat until an undrawn card is picked. The move card function is then called to move the card from deck to hand.

## Trump (2.3)

This section works exactly like the above, picking a random number until a card at deck[r] which has not been drawn is found, then call the move card function to move it to the trump card.

## Trick (3.0)

Before entering this section, i, inGameOption and trumpRobbed are initialized.

This section is enclosed in a do while loop that runs 5 times(for 5 cards in hand), and if the players decide to play the next trick.

If all 5 cards have been played, and no player has reached 25 points, the code will loop back to section 2.1 and start a new deal, trump etc.

## Initialize Trick (3.1)

This line of code resets the trick winning rank to 0. This is so that the lead card ranking will be greater than 0, and replace it, then any other cards will have to compete with the lead card ranking.

## Previous trick winner leads (3.2)

This line of code checks if it is not the first trick to be played, if it is the case, the player to start this trick (currentPlayer) is the winner of the last.

## play (4.0)

The code blocks 4.x all execute for each player. The 4.0 section loops n times, where n is the number of players.

I have an if statement that checks if the currentPlayer is equal to the numOfPlayers, if this is the case, set currentPlayer to 0. This was done so that if the game chooses to start with player 3 (out of 6) the play order should be 3, 4, 5, 0, 1, 2.

## display (4.1)

* 4.1.1 – If the trump was robbed, output a message to the player
* If it is the first play of the trick
  + 4.1.2 Display the trump card
  + 4.1.5 Display the players hand
* If it is not the first play
  + 4.1.2 Display the trump card
  + 4.1.3 Display the lead
  + 4.1.4 Display all cards played this trick
* 4.1.5 Display the players hand // while cycling through the players hand at this time, check if they hold the Ace of trump to determine if they can rob later

## Determine rob legality (4.2)

I have two types of robs, dealer rob and rob. Dealer rob being if the trump card is ace, a normal rob being if any player was dealt the ace of trumps.

The first check in this section is if the trump card is an Ace, and the last player to play is up, and it’s not the first trick. Because my game chooses the first to play at random, I have assumed that the player to their right was the dealer, i.e. if player 3 deals, player 4 starts, then player 3 is the last to play that trick.

If the trump card is an ace, the last to play in the first trick (the dealer) should be able to rob.

I have already checked if any player holds the ace of trump, at the end of the display section.

Next enter an if statement if either legalDealerRob or legalRob are true, then prompt the user whether they would like to rob or not.

If they elect to rob, prompt the player to choose a card to swap. The next block of code is input validation.

If the player selected a valid card, call the copy card function to copy the values of the trump card to the card at the position they indicated. Then re-display their hand and prompt them to play a card that trick

## card selection (4.3)

This section is identical to the code at the end of the legal robbing section. Prompt player to pick a valid card // active and within range 1-5.

## legal renege validation (4.4)

This section determines when a player can or cannot renege and stops them playing a card if doing so would be an illegal renege.

First check if the card fits the below descriptions

* Check if it is the first play of the trick // any card is valid to play for the first player
* Check if a trump is led // check if the played card is of the same suit as trump
* Else non-trump is led // check if played card matches either trump or lead suit

Then handle the exceptions // 5T, JT, AH

A for loop is used to cycle through the players hand and check if they hold and of the special cards and if a card of higher rank has not already been played.

If this is the case the player may legal renege.

Finally cycle through the players hand to check if they hold no cards of the trump or lead suit, if this is the case, they may legally renege as they cannot possibly follow suit.

If the player attempts to illegally renege, print an error message and prompt them to pick another card.

After a valid card has been selected, it is then moved to the playedCards array using the move card function.

If the first to play, their selected card values are copied to the lead.

## Check card ranking (4.5)

This block of code uses the absolute ranking system I laid out when describing the calculate ranking function. I have included a spreadsheet which gives a visual representation of how card rankings work.

Using the played card, trump and lead suits, calculate the ranking of the card. If the ranking is higher than the current winner, then assign the played card to be the winning card and assign the current player to the winning player.

## award points to winner (4.6)

Once ever player has played a card, give 5 points to the winning player

Clear the played cards array

Check if any player has reached 25, if they have break; the outer do while to end the game.

Otherwise print the in-game menu and let players decide if they want to play the next trick, save, output the scores or quit without saving.

## save game

If the save game option is selected in the in-game menu. The application will call the save game function, which then writes the following data to “savegame.txt”

* Card\_t array deck[]
* Card\_t trump
* Player\_t array players[]
* Int numOfPlayers
* Int trickWinPlayer
* Ing highScore
* Bool trumpRobbed
* Int i, trickCount

## load game

If this option is selected from the main menu, the data stored in the savegame.txt file is read and assigned to its respective variable. The game then continues play for the next trick. The load game section of code is nearly identical to the normal gameplay section, excluding a Boolean (which is used to skip the phases before a deal on first load, then later runs will deal as expected). This is a bad practice as I have nearly doubled the lines of code in my project. If I was starting a new project with a save/load game function was expected, I would only have one section for gameplay, and maybe a Boolean check at the start to determine if it is a new game or a loaded game. Seeing as saving and loading a game are only worth 8% together, I decided to leave the code as is.