CS2002 Practical 3 - Read My Mind

By: 150013565 Tutor: Juliana Bowles

Date: 16th March 2018

Abstract

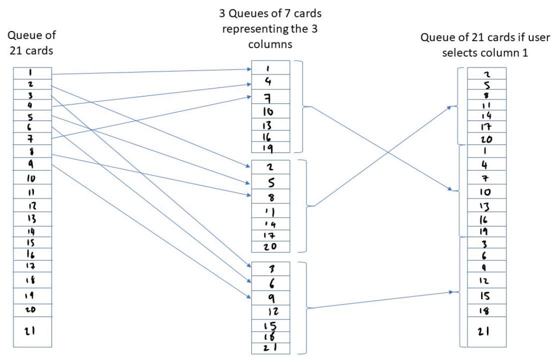
The twenty one card trick also known as the 11th card trick or the three column trick, is a simple card trick that utilizes mathematics to find the user's card.

A set of twenty cards are randomly selected from a standard deck of fifty two playing cards. The cards are then dealt face up into three columns. The user then picks one of the visible cards and tells the performer which column the card is in. The cards are then picked up and re-dealt in the same manner. This process is repeated three times. Upon the third dealing of the cards, the user's selected card will be in the middle of the second column or, alternatively, be the eleventh card in the deck.

Design

To begin this practical, a struct to represent a card and a deck needs to be constructed. The card struct contains two integer variables to represent its rank and suit. The struct also contains a pointer to the card beside it. This is done so that when cards are put in the deck, all cards are aware of what cards are next to it. The deck is implemented as a queue, where the deck maintains references to the head and rear of the deck. Every card that is added to this queue is appended onto the rear so that the new card is the new rear of the queue. The deck also has an integer variable to keep track of the size of the deck.

As mentioned above, the deck is implemented as a queue. A queue was chosen because by using this structure, the implementation becomes a lot simpler. In more detail, the initial twenty one cards are enqueued into a queue. Then to split them into three columns, pop each element in the queue into three smaller queues modulo three. If the result is zero, the element is placed into the first queue, if the result is one, the element is placed into the second queue and if the result is two, the element is placed into the third queue. For example, the fifth element in the big queue is placed into queue three since five modulo three is two. Once three smaller queues are obtained requeue each element in the three smaller queues back into the big queue making sure that the user selected column goes into the queue second. For example if the user selects column one, queue two would be requeued first followed by queue one and then by queue three. A graphical representation of this process is shown below.


Queue of 21 cards
3 Queues of 7 cards representing the 3 columns
Queue of 21 cards if user selects column 1

Implementation

card.h

Inside this file is the struct definition of a card. I have implemented the card struct to contain three fields - rank, suit and a pointer reference to the card next to it. The rank is an integer variable representing the number on the card i.e. 12 represents king, 0 represents ace, and 1 represents two etc. The suit is another integer variable representing the suit of the card - 0,1,2,3 represents clubs, suits, hearts and diamonds respectively. The pointer reference to the card next to it is to help with the queue implementation since a queue contains all the elements sequentially linked together.

deck.h

Inside this file is the struct definition for my queue which I have called a deck. The struct contains three fields - a pointer to the first card in the deck, a pointer to the last card in the deck and a size variable. The pointer to the first card in the deck is to keep control and track of the head. This assists with the implementation of queue methods such as dequeue. The pointer to the last card is to keep control and track of the tail. This assists with the implementation of queue methods such as enqueue. The size field is an integer variable which does as its name suggests - keeps a reference to how big the queue is.

This file also contains the function prototypes for a queue. Methods such as enqueue, dequeue, queueContains and empty are defined in this file. These methods are implemented in the file *deckofcards.c*.

deckofcards.c

This file contains the implementation for the queue. After covering stacks and queues last semester, this implementation of a queue is fairly standard. The only method to note is the getCard method which is used to obtain the person's card. Since user's card will be in the middle position in the middle column, it will be the 11th card in the deck. This method is used to obtain that 11th card by iterating through the queue.

readmymind.c

createDeck

This method generates a random card to fill the deck of cards. The card is generated as follows; a random number is generated from the *'time.h'* library. Two modulo operations are performed on that number, modulo thirteen and modulo four this will generate a random number between zero and thirteen and zero and four, zero inclusive. Once two numbers are generated, these two numbers represent a card, for example, if the method generated nine and three, this would represent the ten of diamonds. These numbers are assigned to a malloc'd card variable and then gets passed to a method queueContains. The queueContains method checks if the card is already in the deck or not, if it is the card is free, else another card is generated. This process is repeated until twenty one cards are created.

printDeck and printCard

The printDeck method works by first printing out the column headers *1,2,3*, displaying the to the user how to interact with the program. Afterwards, the method prints out the cards by row. To print a card, the method printCard is used. The print card first prints out the rank and then prints out the suit. It does this by a simple switch statement. It switches the number stored inside the card's struct and prints out what that number represents. The suit is printed out as ASCII characters to help differentiating between the different cards. After the card is printed, a gap is printed to separate each card.

getUserInput

To obtain the user input, a statement is printed instructing what to do. The *scanf* function is then used to take the user's input and put it in an array of chars. The array of chars is then checked using *sscanf* to determine whether the input is an integer or not. If it is not an integer an error message is printed out prompting the user to try again. The input is only valid if it is 1,2 or 3 representing the three columns. An exit condition is also added in case the user wants to quit.

collectDecks

Once the cards have been printed out and the user enters the column number their card is in, the cards must be collected. This is done by malloc'ing three new decks. Each column is then put it in a deck by enqueing the dequeued card from the original deck. Once three decks are obtained, the user's input is then put in a switch statement and *combineDecks* is used to merge all three decks back into one big deck. The three decks are then emptied and the pointers to the cards inside the deck are freed. The deck pointers themselves are also freed.

combineDecks

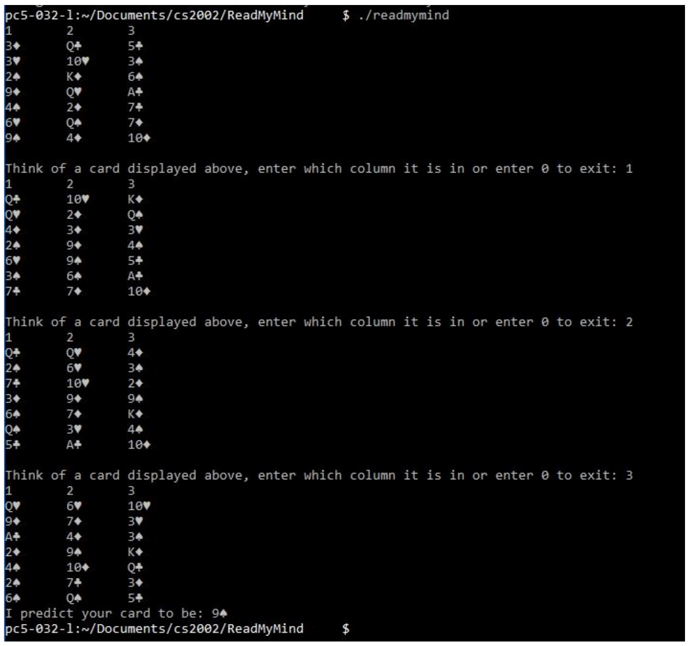
This method works by dequeuing the elements three smaller decks and enqueing them back into the big deck and making sure that the user's deck is dequeued and enqueued to the middle position.

main

The main method contains lots of local variables for the conditions of the trick. For example, there are variables to keep control of how many rows, columns, total number of cards and how many ranks and suits there are. A deck is malloc'd so that twenty one cards can be created and enqueued into the deck. The deck is then printed using the *printDecks* method. A for loop is initiated to obtain the user's input, collect the decks and print the decks. Once the loop has been executed for a total of three times, a statement is printed out 'prediciting' what card the user selected. The deck is then emptied and the cards are freed. The pointer to the deck is then also freed making sure of that there are no memory leaks.

Testing

To run the program, use the command *'make main'* in terminal and run the executable *'readmymind'* afterwards. Testing was conducted manually a sample screenshot of a test is as follows:



```
pc5-032-l:~/Documents/cs2002/ReadMyMind     $ ./readmymind
1       2       3
3♦      Q♣      5♣
3♥      10♥     3♠
2♠      K♦      6♠
9♦      Q♥      A♣
4♠      2♦      7♣
6♥      Q♠      7♦
9♠      4♦      10♦

Think of a card displayed above, enter which column it is in or enter 0 to exit: 1
1       2       3
Q♣      10♥     K♦
Q♥      2♦      Q♠
4♦      3♦      3♥
2♠      9♦      4♠
6♥      9♠      5♣
3♠      6♠      A♣
7♣      7♦      10♦

Think of a card displayed above, enter which column it is in or enter 0 to exit: 2
1       2       3
Q♣      Q♥      4♦
2♠      6♥      3♠
7♣      10♥     2♦
3♦      9♦      9♠
6♠      7♦      K♦
Q♠      3♥      4♠
5♣      A♣      10♦

Think of a card displayed above, enter which column it is in or enter 0 to exit: 3
1       2       3
Q♥      6♥      10♥
9♦      7♦      3♥
A♣      4♦      3♠
2♦      9♠      K♦
4♠      10♦     Q♣
2♠      7♣      3♦
6♠      Q♠      5♣
I predict your card to be: 9♠
pc5-032-l:~/Documents/cs2002/ReadMyMind     $
```

Test No.	Input	Expected Output	Actual Output
1	1,2,3	The last card in the first column to be in the middle of the second column	Correctly guessed card
2	3,1,3	The second last card in the third column to be in the middle of the second column	Correctly guessed card
3	2,2,1	The top card in the second column to be in the middle of the second column	Correctly guessed card
4	4	Invalid input message	Invalid input message
5	abc	Invalid input message	Invalid input message
6	-1	Invalid input message	Invalid input message
7	0	Exit message	Exit message
8	0.1	Exit message	Exit message
9		Nothing program waits for input	Nothing program waits for input

Evaluation

There are a couple of problems with my implementation. One such problem is that my solution is not scalable. If the trick could be expanded to say three columns and nine cards or five columns and seven cards, the code will need to be adapted to suit the new circumstances. The most complicated aspect that will need to change is when it comes to switching the user's selected column into the middle.

Another problem with my implementation is printing the cards. My solution prints the hard coded representation of a card's rank and value directly to terminal. If a deck had a non-standard set for example a deck with five suits and sixteen cards, they would have to be hard coded into the print functions. This could be addressed by using two arrays filled with suits and ranks of the cards and changing the struct definition of card to let rank and suit point to locations in those arrays.

Although I have validated the user's input, if the user enters a float, the program will always round the float down and accept the rounded down number as the user's input. Ideally, it would be better to reject the input and return an error message. Given my current implementation for my input validation, I am not sure how to check for floats as an input.

An alternative way to implement this practical would be to use arrays. In comparison to a queue based solution, arrays would be able to access cards a lot quicker since accessing elements by index is $O\{1\}$ where as with a queue, it is $O\{n\}$ depending where the element is in the queue. However, time complexity is a trade-off with simplicity. With arrays splitting the deck into three columns and merging them all up again can easily go wrong. Whereas with a queue based structure, if implemented correctly, is guaranteed to have a first in first out structure. This is much more organised and has a lower margin of error.

Conclusion

Overall this practical was quite challenging. The queue based solution took me a while to come up with. Implementing this structure using a custom Node class in Java really helped me understand what needed to be done before implementing it in C in order to get it working successfully.

One aspect of this practical I struggled with was memory leaks and undefined behaviour. During development, I had not yet freed cards after they had been used which lead to some strange, undefined behaviour. However, after properly freeing cards, the program started to work as expected.

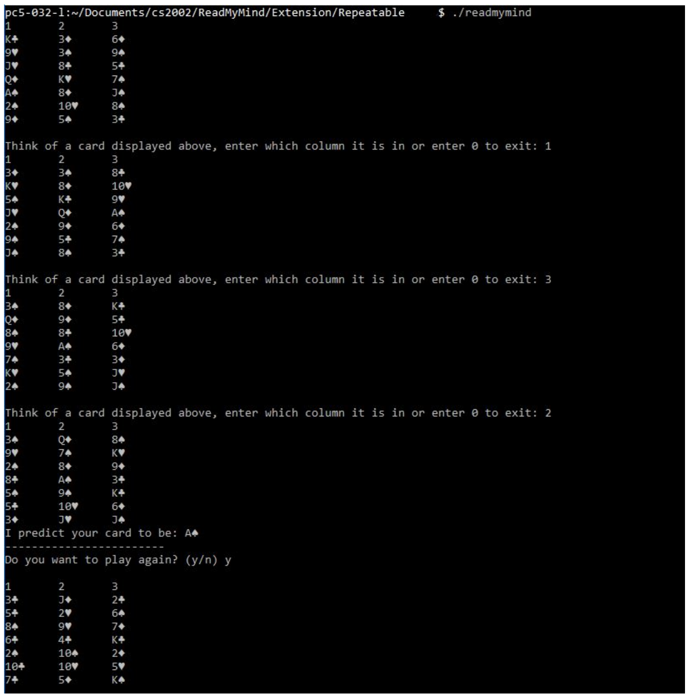
Extensions

Looping The Program

This extension is located in ExtensionsRepeatable. The program can be compiled by running *'make main'* in terminal and running the produced executable called *'readmymind'*.

This extension was fairly trivial. To do this I added a while loop into my main method and added the possibility of exiting after each time the trick has been performed so that the program won't infinitely loop. Input validation was also added to prevent unexpected input. I made sure only to malloc the deck once, and not every time the program ran as this could lead to the program using up too much memory. Each time the trick has been fully performed, the deck queue is completely emptied so that a new, random set of cards can populate the queue.

Here is an image of this extension working in terminal:



```
pc5-032-l:~/Documents/cs2002/ReadMyMind/Extension/Repeatable     $ ./readmymind
1       2       3
K♣      3♦      6♦
9♥      3♠      9♠
J♥      8♣      5♣
Q♦      K♥      7♠
A♠      8♦      J♠
2♠      10♥     8♠
9♦      5♠      3♣

Think of a card displayed above, enter which column it is in or enter 0 to exit: 1
1       2       3
3♦      3♠      8♣
K♥      8♦      10♥
5♠      K♣      9♥
J♥      Q♦      A♠
2♠      9♦      6♦
9♠      5♣      7♠
J♠      8♠      3♣

Think of a card displayed above, enter which column it is in or enter 0 to exit: 3
1       2       3
3♠      8♦      K♣
Q♦      9♦      5♣
8♠      8♣      10♥
9♥      A♠      6♦
7♠      3♣      3♦
K♥      5♠      J♥
2♠      9♠      J♠

Think of a card displayed above, enter which column it is in or enter 0 to exit: 2
1       2       3
3♠      Q♦      8♠
9♥      7♠      K♥
2♠      8♦      9♦
8♣      A♠      3♣
5♠      9♠      K♣
5♣      10♥     6♦
3♦      J♥      J♠
I predict your card to be: A♠
------------------------
Do you want to play again? (y/n) y

1       2       3
3♣      J♦      2♣
5♣      2♥      6♠
8♠      9♥      7♦
6♣      4♣      K♣
2♠      10♠     2♦
10♣     10♥     5♥
7♣      5♦      K♠
```

Mathematical Analysis

This trick works because of the mathematics behind it and I was curious as to how this works.

The mathematics behind the trick is as follows [2]; the person's card is at position $1 \leq x \leq 7$ in any given column. When the cards are picked up the new position of the card is $7 + x$. When the cards are displayed again, the position of the card is now at $p_1 = \lceil \frac{7+x}{3} \rceil$ in a column. The next time the cards are picked up and displayed, the position of the card is now at $p_2 = \lceil \frac{7+p_1}{3} \rceil$ in a column. The final time the cards are picked up the card is now at $7 + p_2$. Simplifying this last expression down, we can obtain:

$$7 + p_2 = 7 + \lceil \frac{7+p_1}{3} \rceil \tag{1}$$

$$7 + \lceil \frac{7+p_1}{3} \rceil = 7 + \lceil \frac{7+\lceil \frac{7+x}{3} \rceil}{3} \rceil \tag{2}$$

$$7 + \lceil \frac{7+\lceil \frac{7+x}{3} \rceil}{3} \rceil = 7 + \lceil \frac{\frac{21+7+x}{3}}{3} \rceil \tag{3}$$

$$7 + \lceil \frac{\frac{21+7+x}{3}}{3} \rceil = 7 + \lceil \frac{28+x}{9} \rceil \tag{4}$$

$$7 + \lceil \frac{28+x}{9} \rceil = 7 + \lceil 3 + \frac{1+x}{9} \rceil \tag{5}$$

$$7 + \lceil 3 + \frac{1+x}{9} \rceil = 10 + \lceil \frac{1+x}{9} \rceil \tag{6}$$

Since $1 \leq x \leq 7$, $10 + \lceil \frac{1+x}{9} \rceil$ is always going to equal 11. Hence, regardless of which card the person chooses, the card will always in the 11th position after three iterations.

A generalization for a n columns by m rows, where n and m are odd integers, is possible to obtain as shown by Champanerkar [1]. This paper shows that for a person's card to be in the middle of the middle column after 3 iterations this equality must hold;

$$n + 1 \leq m \leq 3n$$

If we let $n = 3$ and $m = 7$, we can see that this equality holds true hence the trick works for 3 columns and 7 rows.

A possible extension would be to expand the program to work with a user defined row and column length. Perform a simple analysis on the user's input to determine if the trick would work or not. The program should print out the cards if it will work or provide feedback to the user letting them know that with their defined row and column lengths, the trick won't work.

Another Mathematical Card Trick

`https://www.youtube.com/watch?v=l7lP9y7Bb5g&t`

The above is a YouTube link which links to a video produced by Numberphile, a channel dedicated to mathematics. The aforementioned video inspired this extension exercise.

The outline of the trick is as follows:

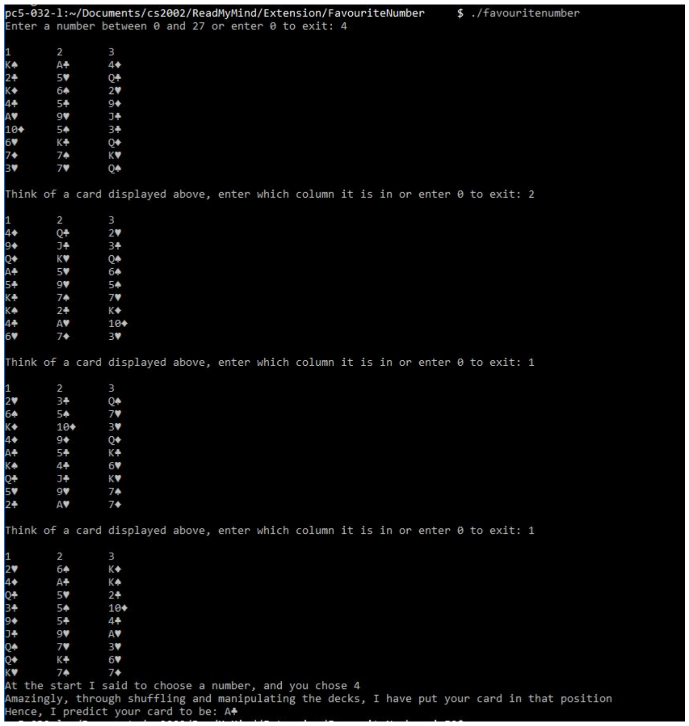
1. Get a random set of 27 cards from a standard deck of cards
2. Ask the person you are performing this trick to, to select a number between 0 and 27
3. Once they've told you a number, convert it to base 3 and reverse it. For example if the person chooses the number 11, 11 in base 3 is 102 and if you reverse that, it becomes 201
4. Display the 27 cards into 3 columns of 9 cards
5. Ask the person you are performing this trick to to pick a visible card on the table and tell you the column the card is in
6. Once they've told you which column it is in, use the first digit in the base three number to put that column in that position when all the columns are collected. Continuing with our example of 11, the first digit in the base 3 number represents the first collection, 2 means that the user's column must be placed at the bottom. The second digit represents the second collection, 0 means that the user's column must placed at the top. The third digit represents the third collection, 1 means that the users column must placed into the middle

7. Re-deal the cards by placing them row first and do this until no more re-deals have to be done

8. Performed correctly, the person's card will be the in the position the person chose at the beginning of the trick i.e. if the person chose 11, the card will be in the 11th position.

This trick was designed in much the same way as the trick the specification outlined. There was one big queue and three smaller queues. Rather than always putting the person's column in the middle, the column would be placed at either the top, middle or bottom depending on what the person's number was. Additional methods such as getUserNumber, convertToBaseThree, positionDecks and switchDecks were implemented to perform this trick. The getUserNumber method is used to obtain the user's number at the beginning of the trick. It has input validation so that the user does not enter an invalid number. The convertToBaseThree method is used to convert the user's number into base three. It contains a static integer array of size three to hold the three integers representing the user's number in base three. A pointer is returned from this method to point to this static array. The positionDecks method takes the user's input and organises the decks so that the program knows which deck the user selected. The switchDecks method, uses the pointer to the static array mentioned above to determine which position the deck should be placed in.

To run the program, use the command *'make main'* in terminal and run the produced executable *'favouritenumber'*. Testing was conducted manually a sample screenshot of a test is as follows:



```
pc5-032-l:~/Documents/cs2002/ReadMyMind/Extension/FavouriteNumber     $ ./favouritenumber
Enter a number between 0 and 27 or enter 0 to exit: 4

1       2       3
K♠      A♣      4♦
2♣      5♥      Q♣
K♦      6♠      2♥
4♣      5♣      9♦
A♥      9♥      J♣
10♦     5♠      3♣
6♥      K♣      Q♦
7♦      7♠      K♥
3♥      7♥      Q♠

Think of a card displayed above, enter which column it is in or enter 0 to exit: 2

1       2       3
4♦      Q♣      2♥
9♦      J♣      3♣
Q♦      K♥      Q♠
A♣      5♥      6♠
5♣      9♥      5♠
K♣      7♠      7♥
K♠      2♣      K♦
4♣      A♥      10♦
6♥      7♦      3♥

Think of a card displayed above, enter which column it is in or enter 0 to exit: 1

1       2       3
2♥      3♣      Q♠
6♠      5♠      7♥
K♦      10♦     3♥
4♦      9♦      Q♦
A♣      5♣      K♣
K♠      4♣      6♥
Q♣      J♣      K♥
5♥      9♥      7♠
2♣      A♥      7♦

Think of a card displayed above, enter which column it is in or enter 0 to exit: 1

1       2       3
2♥      6♠      K♦
4♦      A♣      K♠
Q♣      5♥      2♣
3♣      5♠      10♦
9♦      5♣      4♣
J♣      9♥      A♥
Q♠      7♥      3♥
Q♦      K♣      6♥
K♥      7♠      7♦
At the start I said to choose a number, and you chose 4
Amazingly, through shuffling and manipulating the decks, I have put your card in that position
Hence, I predict your card to be: A♣
```

Further testing was done as follows (counting the cards start at 0):

Test No.	Input	Expected Output	Actual Output
1	4,2,1,1	The first card in the second column to be sequentially be the fourth card in the deck	The chosen card is the fourth card in the deck
2	20,3,3,2	The sixth card in the third column to be sequentially be the twentieth card in the deck	The chosen card is the twentieth card in the deck
3	4,4	Invalid input message	Invalid input message
4	abc	Invalid input message	Invalid input message
5	-1	Invalid input message	Invalid input message
6	0	Exit message	Exit message
7	0.1	Exit message	Exit message
8		Nothing program waits for input	Nothing program waits for input

The card trick is definitely more impressive in person than it is via terminal. The 'wow' effect isn't present through a computer as it may seem that the card is just placed in that position after the third re-deal. However, the trick was implemented correctly and works as intended.

Given more item I would have liked to animate the trick or at least find out how to delay the printing of the cards so that the trick would appear more authentic. However, I am still very pleased with how this extension turned out.

References

[1] Champanerkar, J., and Jani, M. Stable fixed points of card trick functions. *arXiv preprint arXiv:1308.3396* (2013).

[2] (https://math.stackexchange.com/users/393611/evangelos bampas), E. B. Mathematics behind this card trick. Mathematics Stack Exchange. URL:https://math.stackexchange.com/q/2055423 (version: 2017-04-13).