CS2002 Practical 3 - C Programming 2

Matriculation Number: 160001362

16/03/2018

Contents

1 Overview **3**

2 Design and Implementation **3**
2.0.1 Files and Libraries 3
2.1 Index Based 5
2.1.1 Summary 5
2.1.2 Data Structures 6
2.2 Pointer Based 6
2.2.1 Summary 6
2.2.2 Data Structures 7

3 Testing **7**
3.1 Input Validation 7
3.2 Gathering Columns 10
3.3 Dealing Cards 11

4 Conclusion and Evaluation **12**

5 Extensions **12**

1 Overview

This practical specified the development of a system to perform a card trick that finds a user selected card based on the column in which their card resides among twenty-one random cards. The practical further specified that the implementation should utilise an index (array) based solution and/or a pointer based solution. For this submission both an index based solution has been developed as well as a non-global pointer based solution which makes use of linked lists.

For the pointer based implementation I challenged myself to not use a single set of square brackets to reference an index (even for displaying card suits/ranks) which proved challenging but achievable.

To track changes in the project I used git in conjunction with GitHub with my own private git repository. This proved useful to keep a library of working versions of the program that I could roll back to if local changes caused the program to mysteriously break (such as causing a segmentation fault).

The provided README.md file specifies execution instructions.**TALK ABOUT EXTENSIONS AND MEMORY LEAKS**

2 Design and Implementation

The index based implementation is contained with a folder 'Index_Based' and the pointer based implementation is contained within a folder 'Pointer_Based', both of which are within the 'ReadMyMind' directory. The code is heavily commented to make it clearn how each of the implementations work.

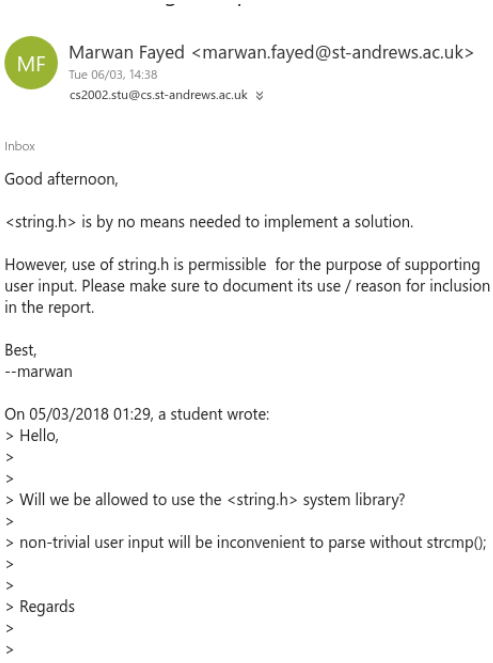
Both implementations of the program have the same following files within their respective directories:

2.0.1 Files and Libraries

- **readmymind.c** - This file contains the main function of the program. It is responsible for handling the control flow of the program and declaring the necessary structs and variables required.
- **cards.c** - This file is responsible for initialising the fundamental data structures required for use in the program including the deck, the cards, as well as the columns used to store the 21 cards (using getDeck, getCard, and fill respectively).
- **io.c** - Handles the input and output of the solution. The file contains functions to get the user's column selection, validate this input, and convert this input into an integer format more easily utilised in the program. In addition the file contains a function to print the cards of a columns

structure to the terminal as well as print the centre card of a columns structure.

- **actions.c** - Contains functions relating to the actions performed during the card trick. The 'gather' function creates a new columns struct and initialises it with the previous columns rearranged so that the column selected by the user is in the centre. The 'deal' function iterates through the cards of each column in the columns struct returned from 'gather' and deals the cards into the new struct from left to right across the columns.
- **readmymind.h** - Contains definitions for the size of data structure dimensions (e.g. COLUMN_SIZE, SUIT_SIZE), the structs representing the data structures, as well as for the signatures of functions used throughout the program.
- **stdbool.h** - Used to allow true and false values to be returned when validating the user's input.
- **stdlib.h** - Used to set appropriate values to NULL as well as enabling the use of malloc and free.
- **stdio.h** - Used for input and output.
- **assert.h** - Used to terminate program if branches of code are reached that should be unreachable.
- **string.h** - Used for the strlen function to check if the size of the user's input is greater than a single character. Although not explicitly stated as one of the permitted libraries in the specification, Dr Fayed stated in a query that its use is permissible:



Marwan Fayed <marwan.fayed@st-andrews.ac.uk>
Tue 06/03, 14:38
cs2002.stu@cs.st-andrews.ac.uk

Inbox

Good afternoon,

<string.h> is by no means needed to implement a solution.

However, use of string.h is permissible for the purpose of supporting user input. Please make sure to document its use / reason for inclusion in the report.

Best,
--marwan

On 05/03/2018 01:29, a student wrote:
> Hello,
>
>
> Will we be allowed to use the <string.h> system library?
>
> non-trivial user input will be inconvenient to parse without strcmp();
>
>
> Regards
>
>

2.1 Index Based

2.1.1 Summary

Declares and initialises a deck array of 52 cards and uses it to fill a Columns struct of 21 cards (of 3 column structs of 7). The columns are filled by randomly generating a number (signifying a card to select from the deck) and checking an array of selected values to see if the card has already been picked. It does this until 21 cards have been selected.

The user then inputs their chosen column (this data is validated using an array of valid inputs) and this value is converted to an integer format and is then used to gather the columns. The columns are gathered by assigning the selected column of the previous Columns struct to the middle Column struct of the new Columns struct. The other columns are then iterated over and assigned.

Finally the Columns struct with the selected column in the middle is dealt to a new set of columns by iterating over each element in the new Columns struct and calculating which old element should be dealt.

The above process of getting user input, gathering the cards, and dealing the cards is then repeated three times with the user's card being displayed at the end.

2.1.2 Data Structures

- **Card** - Has two integer attributes:
  - **suit** - a number from 0 - 3 representing either spade, heart, diamond, or club.
  - **rank** - a number from 0 - 12 representing and Ace, 2, 3, 4, 5, 6, 7, 8, 9, Jack, Queen, or King respectively.
- **Deck** - Has one attribute 'cards' which is an array of type Card which has the size of the constant DECK_SIZE (in the general case 52).
- **Column** - Like the Deck struct - has one attribute 'cards' which is an array of type Card which has the size of constant COLUMN_SIZE (typically 7).
- **Columns** - Has an attribute 'column' which is an array of columns of size NUM_COLUMNS (typically 3).

2.2 Pointer Based

2.2.1 Summary

The pointer based implementation has an additional C file '**linkedlist.c**' which has functions to interface with the linkedlist data structure declared in readmymind.h. These include functions to create and add nodes to a linked list as well as retrieve nodes from a linked list. In addition there is a function to free the memory occupied by nodes of a linked list to avoid memory leakage and a function to update the indices of a given linked list.

A linked list implementation by HackerEarth[1] was used to give an idea of how to implement a basic linked list data structure in C.

The pointer version uses almost the exact same process as described in section 2.1.1. Aside from the way the data structure is accessed during the trick, the main difference between the two implementations is the way that the user input and output works. In the pointer based implementation instead of accessing card icons to be displayed using a constant array, a switch statement containing print statements is used.

Similarly, instead of storing a constant array of valid inputs and accessing it to validate the user's input, a switch statement is used for input validation. A switch statement rather than iteration is also used in the gather stage. While the implementation of these features is more elegant in the array based implementation, this alternate implementation demonstrates that arrays are not strictly necessary.

[1] https://www.hackerearth.com/practice/data-structures/linked-list/singly-linked-list/tutorial/

The pointer implementation shares a common Card struct with the index based implementation, however the other following structures differ in implementation:

2.2.2 Data Structures

- **Card** - Has two integer attributes:
  - **suit** - a number from 0 - 3 representing either spade, heart, diamond, or club.
  - **rank** - a number from 0 - 12 representing and Ace, 2, 3, 4, 5, 6, 7, 8, 9, Jack, Queen, or King respectively.
- **LinkedList** - Has the following attributes:
  - **card** - Stores a Card attribute that can be thought of as the 'value' of the node (a card in the deck or columns).
  - **next** - A linkedlist pointer to the next node the in the list.
  - **index** - An integer signifying a node's position in its linkedlist - manually set and used so that it would be easier to translate the functions in the index/array based implementation into the pointer implementation.
  - **chosen** - A boolean used when selecting the random 21 cards from the deck of cards to signify whether a particular card has already been selected.
- **Node** - A typedef that allows a LinkedList pointer to be declared using the keyword 'Node'.
- **Deck** - A struct that has one attribute 'node' which is a LinkedList Node. Used to allow 52 nodes to be added, each of which stores a unique playing card.
- **Columns** - A struct that contains three Node attributes called 'first', 'second' and 'third'. Each attribute stores the heads of their respective linkedlists which represent the three columns of the card trick.

3 Testing

The following is a sample of output generated from testing the implementations. The function scanf was used to get the input from the user and proved surprisingly robust given the limited scope of this program.

3.1 Input Validation

Figure 1: Invalid inputs being validated. Includes multiple character input, character input larger than allocated memory for the input char*, invalid single character input, as well as some miscellaneous input.

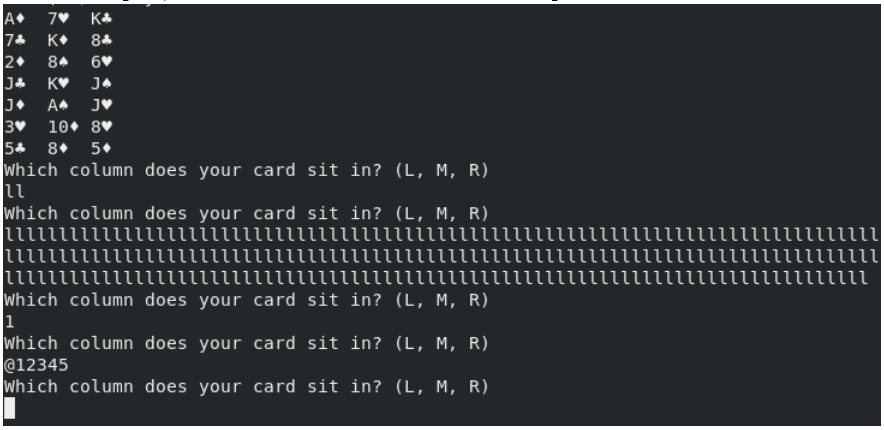


Figure 2: Example of valid inputs being accepted. Valid inputs include the characters l, L, r, R, m, and M.

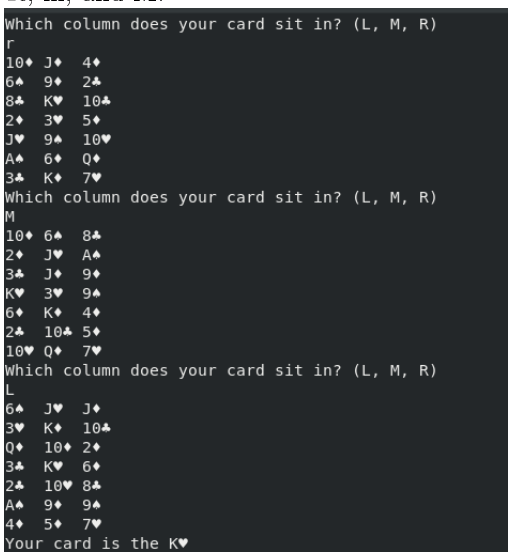
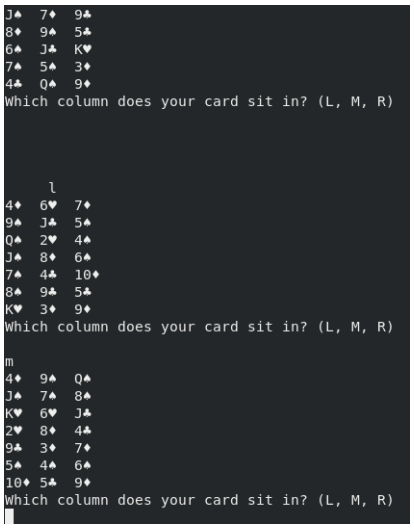
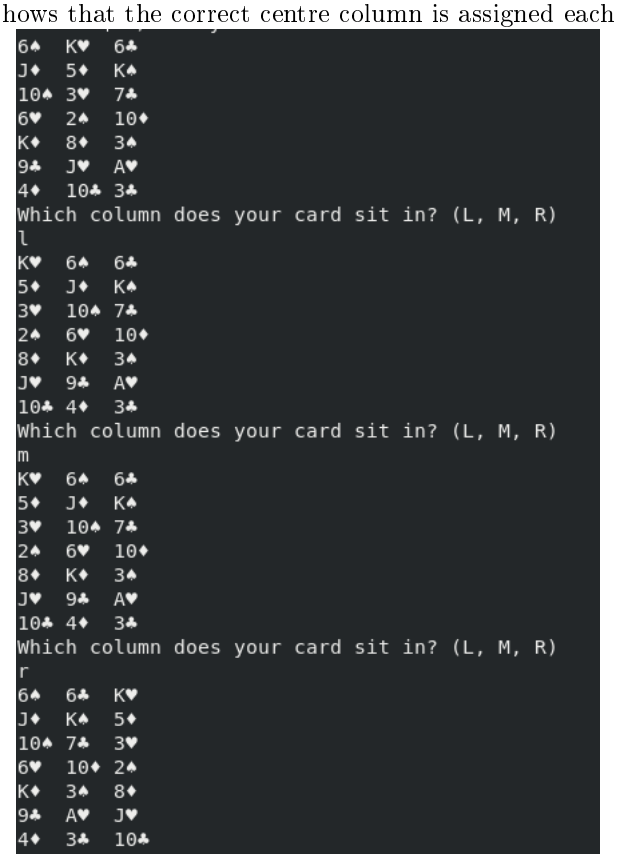


Figure 3: Example of valid inputs being accepted with preceding white space (white space and new line characters do not effect the single character input).



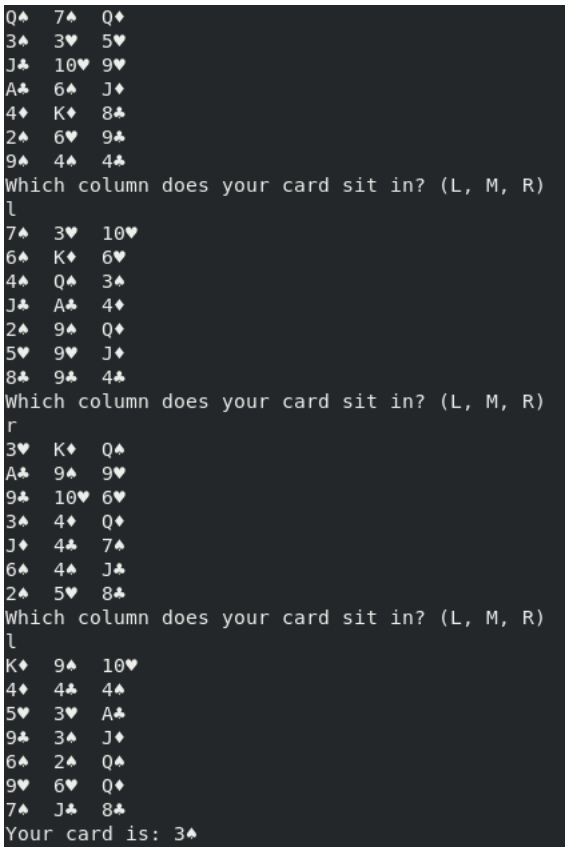
3.2 Gathering Columns

Figure 4: Output of the gather stage when the deal stage is commented out. The output shows that the correct centre column is assigned each time.



3.3 Dealing Cards

Figure 5: Output of multiple deal stages when the user's card is the 3 of spades. This output clearly shows that the cards are correctly dealt left-to-right, top-to-bottom.



4 Conclusion and Evaluation

In conclusion the requirements specified in the practical document to develop a card trick program have been fully implemented using both an index based implementation as well as non-global pointer based solution utilising a linked list structure. The solutions also both follow the constraints specified in regard to the allowed libraries as well as both only using local instances of data structures.

To improve upon my solution I would re-implement the way the newColumns struct is initialised in the pointer implementation when the cards are gathered and dealt. The current implementation of the program leaks 1.5MB of memory due to unused allocated memory that was once pointed to by a Columns struct. Without the frees at the end of the main function the program initially had over 3MB of leaked memory. I attempted to free the original columns struct when newColumns is assigned in both the gather and deal functions to reduce the amount from 1.7MB to 0MB however this resulted in a segmentation fault and so I decided to permit some memory not being freed (this would be more of a problem if the trick is repeated many times).

Figure 6: Output of 'valgrind ./readmymind'

```
==23702== HEAP SUMMARY:
==23702==     in use at exit: 1,728 bytes in 72 blocks
==23702==   total heap usage: 153 allocs, 81 frees, 5,612 bytes allocated
==23702==
==23702== LEAK SUMMARY:
==23702==    definitely lost: 216 bytes in 9 blocks
==23702==    indirectly lost: 1,512 bytes in 63 blocks
==23702==      possibly lost: 0 bytes in 0 blocks
==23702==    still reachable: 0 bytes in 0 blocks
==23702==         suppressed: 0 bytes in 0 blocks
==23702== Rerun with --leak-check=full to see details of leaked memory
==23702==
```

5 Extensions

- Letting user repeat trick
- Colours for cards using ansi codes **DONE**
- Compare complexity of each solution and show time comparison (also discuss memory leakage).

References

[1] Mohd Sanad, Zaki Rizvi, *"Singly Linked List Tutorial"*, HackerEarth, https://www.hackerearth.com/practice/data-structures/linked-list/singly-linked-list/tutorial/, (2016).