2018-4-16 16006128 Juliana Bowles

Introduction:

For this practical, we were tasked with creating a program to execute a rudimentary magic trick, intended to give the illusion of reading the user's mind.

Design:

My main design was to represent the deck of cards as much as possible as a linked-list, with pointers, and consistently think of the pointers as an actual deck of cards.

The cards themselves, are treated as nodes in a linked list, with an icon, for the unicode character of that card, a rank, and a suit. I chose this since I felt it would be easier to start with pointers if I planned on using them, and while I wanted to implement unicode characters, I felt it was safer to have the ranks and suits individually to fall back on and check. Another one of the most notable design choices to make was generating the 21 random cards, and printing the cards to the user. How do I get the list in the first place? Do I generate 21 random cards, or 52 and take a segment of 21? Do I iterate through the list, printing values, and printing a new line at every third value? Or do I instead sort the list into 3 shorter lists?

For the random cards, I first generated a full deck of cards, similar to store-bought cards, Ace of Spades through to the King of Diamonds, no Jokers. I then shuffled this, using a shuffle method I will touch upon later in greater detail. From this, I then selected the top 21 cards, placing the other 31 to a side, in another object, so I could re-use cards (without generating new cards), and so I could free the cards at the end of the program.

From here, I split the cards into 3 piles, or decks as I called them, starting with the left hand deck, then the central one, then the right hand deck, as per the specification. To do this, I utilised a pointer pointing to the base of each deck, so as not to lose them, and a pointer pointing to the top of each deck. Once I had these 3 decks, I printed them to the user. Upon them selecting the deck with the correct card, I then collected them up as per the specification. The leftmost deck the user didn't choose, the return pointer was set to this head. Then this deck's pointer was set to the foot of the deck the user had chosen, who's pointer was then set to the head of the remaining deck. This was made simple by the fact that I had already split up the lists beforehand.

Finally I repeated this process twice more.

For the shuffle method, since I was using linked lists, I could not simply randomly throw values around, as this would be incredibly difficult and inefficient, but instead this was where my treating the linked lists like actual decks of cards came in use, since to "shuffle" the linked list, I simply split it in half, and then created a new linked list, and for randomly selected one list, and took the head of that list, and moved it along by one. In practicality, this ended up being some form of corrupt merge-sort, and it seemed to be very efficient.

Testing:

For my testing, I didn't implement the actual specification until the last ¼ of my workload, instead I simply kept running through with custom cards, lists, trying to break each method to its extreme. Finding odd quirks with my program, fixing memory leaks, and making sure that every previous method worked before attempting to add a new method. Upon finishing the specification, I attempted to break my own program, giving invalid answers for user input, and constantly checking that the correct card was guessed. I believe I ran it seriously 20 times or so.

I also utilised the valgrind command to ensure that no memory leaks existed once I was sufficiently far through.

Difficulties:

Slightly humorously, for a ling time, my program failed to run without a printf() statement at the beginning of the main method. While I could not seem to fix this, it was not present if a simple printf statement was present.

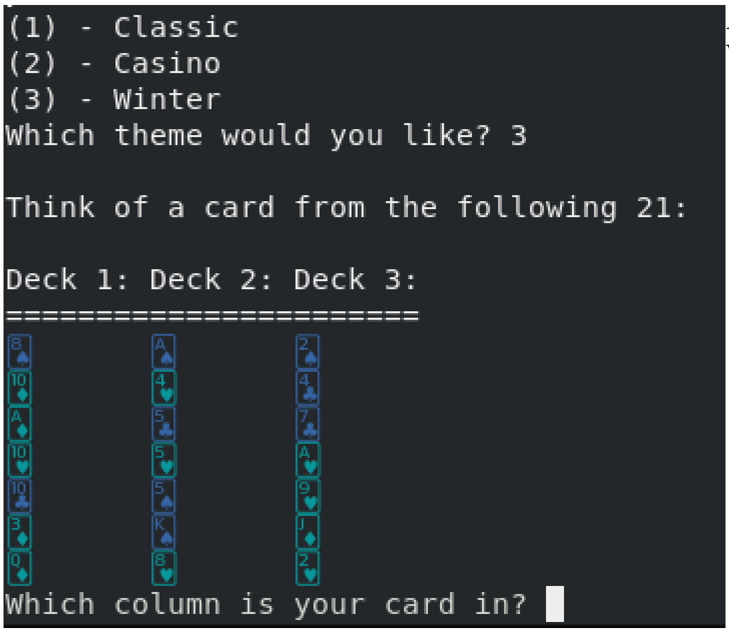
Other than that, I encountered some issues with my use of unicode characters, most notably, upon compile 52 warnings are generated, and this can be very tiring, but the program also refused to compile in gcc, but compiles absolutely fine in clang.

Conclusion:

While the program throws 52 warnings upon compile, the program actually appears to be fairly robust, and I believe that the code, while containing a large amount of methods, no methods are needlessly confusing, I attempted to keep them as small and concise as possible, with the exception of the giant switch statement in printCardInfo(). However I feel that is excusable, since it is only called once. I am pleased with the way the program seems to handle memory, and how it treats the linked-list constantly as an actual deck of cards.

Extension:

For my extensions, I vastly improved the user interface, adding colour, selectable themes, and implementing the Unicode playing card characters. However I felt that there was the possibility that this could arguably be similar to using arrays, so the card struct also contains values for rank and suit independent of this icon. The program also runs as many times as the user wants, using exactly the same "deck" of cards. Never re-generating the deck, and re-shuffling upon each cycle, so an entirely new set of cards could appear.

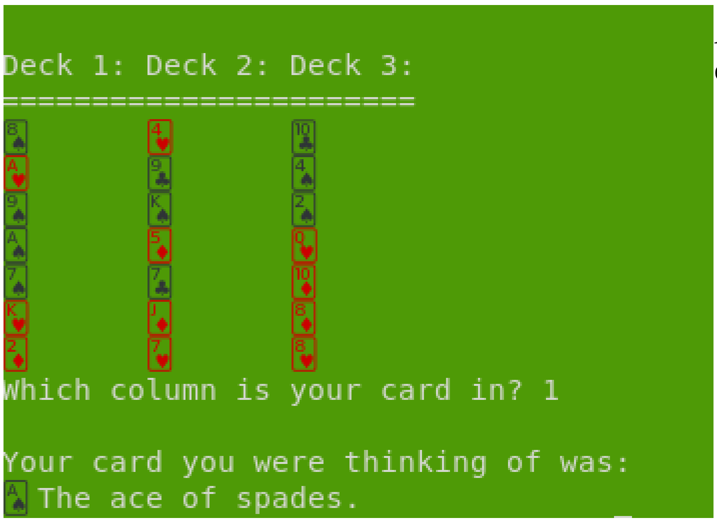


```
(1) - Classic
(2) - Casino
(3) - Winter
Which theme would you like? 3

Think of a card from the following 21:

Deck 1: Deck 2: Deck 3:
=======================
🂨       🂡       🂢
🃊       🂴       🃔
🃁       🃕       🃗
🂺       🂵       🂱
🃚       🂥       🂹
🃃       🂮       🃋
🃍       🂸       🂲
Which column is your card in?
```

An example of selecting theme: The Winter theme.



```
Deck 1: Deck 2: Deck 3:
=======================
🂨       🂴       🃚
🂱       🃙       🂤
🂩       🂮       🂢
🂡       🃅       🂽
🂧       🃗       🃊
🂾       🃋       🃈
🃂       🂷       🂸
Which column is your card in? 1

Your card you were thinking of was:
🂡 The ace of spades.
```

An example of the program guessing the card, in the Casino theme.