Practical 2 Report

Design and Implementation

Overview

The general structure and design of this program was to only make declarations and function calls from the main method and the define a set of functions to execute the spec. The main method calls the relevant methods to first validate input (to the extent required by the spec). The next step is to create the card structure and convert K to binary. The final step is to shuffle the card structure and print the results. The functions are split across 3 .c files and 2 header files, there is a makefile to handle linking and compiling.

Data Structure

The spec delineates the different ways of approaching this practical in terms of types of data structure. I opted to go for the heap/dynamic memory implementation that doesn't make use of or benefit from contiguous memory. To implement it this way I created a strut called 'Card', with variables {char val; char suit; struct Card* next;}. This allowed me to create a linked list with each node representing a Card. Furthermore, each Card is memory allocated so that the memory is dynamic and only the number of cards entered will be created. This means that the program doesn't have wasted space and it can only be navigated using list pointers and C pointers (this is much more difficult than using an array). At the end of each run each node is freed back to the compiler.

List Creation

I implemented a function to create a list of cards given a space separated input stream. The method requires one Card pointer to hold the top of the list, one to create new cards and another to hold the end of the list at any point so new cards may be appended. The function will create and append cards until the end of the stream is reached.

Print Methods

To print the shuffles I implemented three functions, one that holds a case switch for the 'stringplace' enum. This means data passed in will be Stacscheck compliant when output. The second function takes a list of cards and the type of shuffle, it then makes use of the switch case to print out all the cards as well as the correct prefixes and suffixes. The final method takes a reverse-ordered binary number and calls each shuffle methods before passing the shuffled list and type of shuffle into the second print function.

Binary Conversion

To achieve this the relevant function takes the kth position desired and returns the binary representation of the number as an array of integers, the binary number however is in reverse. For example, if k = 37, then the array created is {1,0,1,0,0,1} (The binary representation for 37 being 100101). To achieve this, we take (int) log10(k)/log10(2) to get the position of the first '1'. Continuing with k = 37, we would get 6. A '1' is placed at the 6th index, we subtract 2^6 from 37 and repeat the process till 0 or 1 is reached. If the last result is 1, we must deal with this separately.

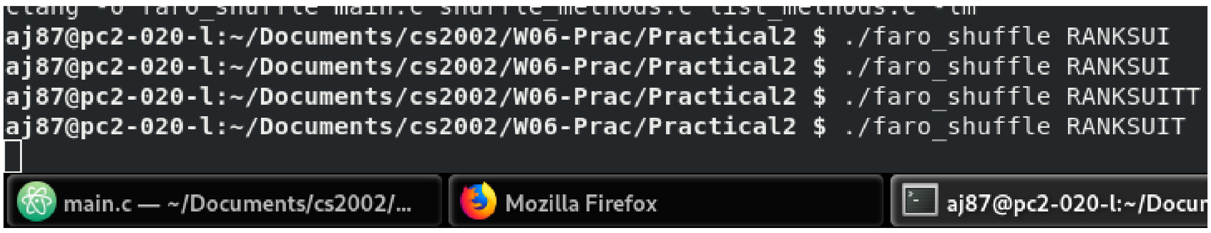
Shuffle Methods

The effect achieved by the shuffle methods is outlined in the spec, so specifically with my data structure you need to break it into two sub-lists, and then re-arrange the pointers in each list so that they're interweaved.

Testing

Paramater

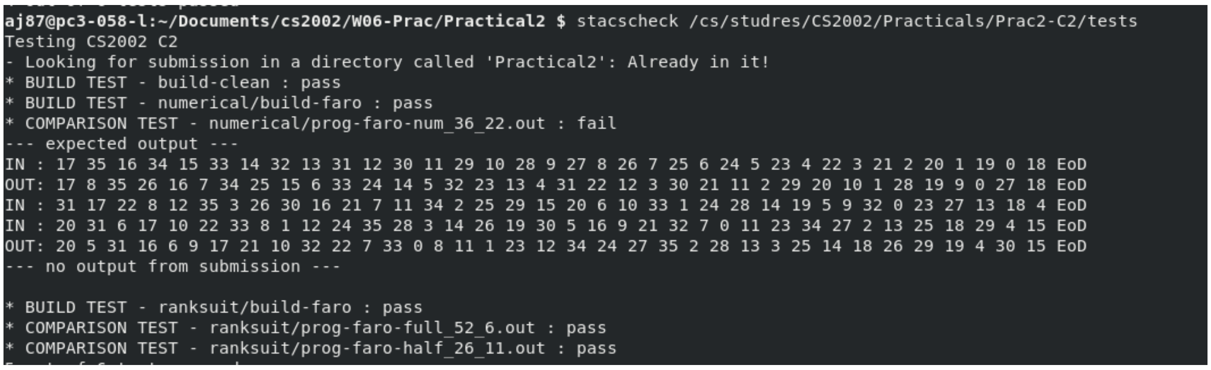
This test is to ensure only 'RANKSUIT' will run the full program, as seen below only RANKSUIT gets to the input data stage.



```
clang -o faro_shuffle main.c shuffle_methods.c list_methods.c -lm
aj87@pc2-020-l:~/Documents/cs2002/W06-Prac/Practical2 $ ./faro_shuffle RANKSUI
aj87@pc2-020-l:~/Documents/cs2002/W06-Prac/Practical2 $ ./faro_shuffle RANKSUI
aj87@pc2-020-l:~/Documents/cs2002/W06-Prac/Practical2 $ ./faro_shuffle RANKSUITT
aj87@pc2-020-l:~/Documents/cs2002/W06-Prac/Practical2 $ ./faro_shuffle RANKSUIT
```

Stacscheck

For this test I checked that my program satisfied the automated checker, because I did no extensions the numerical test doesn't pass.

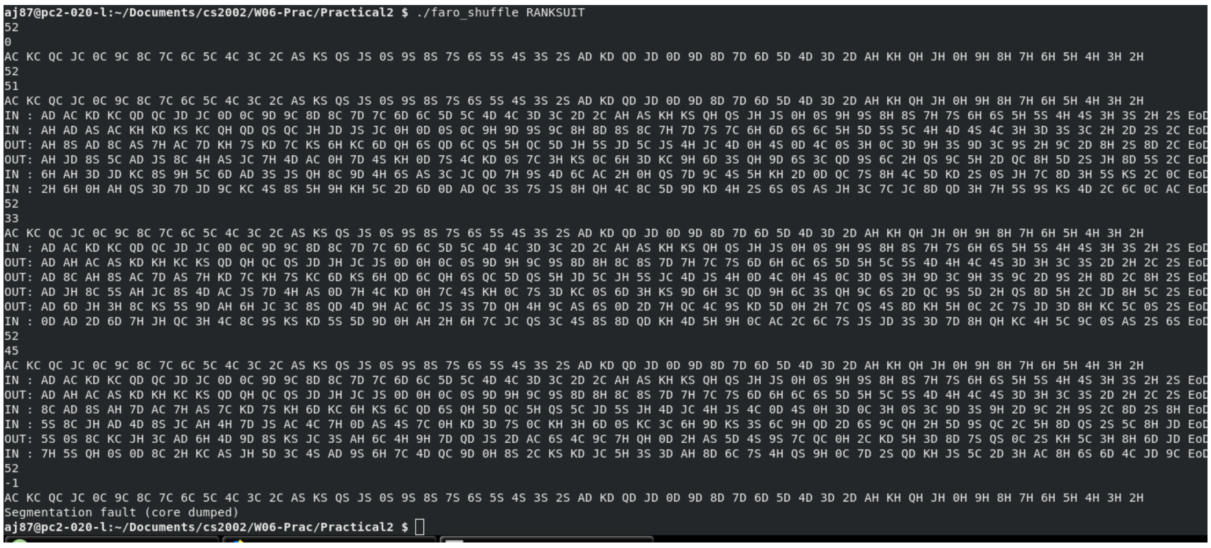


```
aj87@pc3-058-l:~/Documents/cs2002/W06-Prac/Practical2 $ stacscheck /cs/studres/CS2002/Practicals/Prac2-C2/tests
Testing CS2002 C2
- Looking for submission in a directory called 'Practical2': Already in it!
* BUILD TEST - build-clean : pass
* BUILD TEST - numerical/build-faro : pass
* COMPARISON TEST - numerical/prog-faro-num_36_22.out : fail
--- expected output ---
IN : 17 35 16 34 15 33 14 32 13 31 12 30 11 29 10 28 9 27 8 26 7 25 6 24 5 23 4 22 3 21 2 20 1 19 0 18 EoD
OUT: 17 8 35 26 16 7 34 25 15 6 33 24 14 5 32 23 13 4 31 22 12 3 30 21 11 2 29 20 10 1 28 19 9 0 27 18 EoD
IN : 31 17 22 8 12 35 3 26 30 16 21 7 11 34 2 25 29 15 20 6 10 33 1 24 28 14 19 5 9 32 0 23 27 13 18 4 EoD
IN : 20 31 6 17 10 22 33 8 1 12 24 35 28 3 14 26 19 30 5 16 9 21 32 7 0 11 23 34 27 2 13 25 18 29 4 15 EoD
OUT: 20 5 31 16 6 9 17 21 10 32 22 7 33 0 8 11 1 23 12 34 24 27 35 2 28 13 3 25 14 18 26 29 19 4 30 15 EoD
--- no output from submission ---

* BUILD TEST - ranksuit/build-faro : pass
* COMPARISON TEST - ranksuit/prog-faro-full_52_6.out : pass
* COMPARISON TEST - ranksuit/prog-faro-half_26_11.out : pass
```

Generic

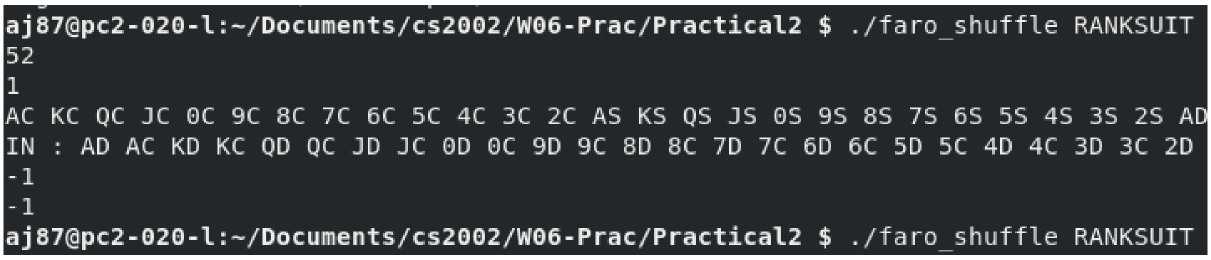
On top of Stacscheck I thought it'd be good to do some extra testing. I tested 0, 51, 33, 45 and -1. The first 2 are edge cases and the following 2 are arbitrary choices. As you can see they AC card is in the correct place for each test. Finally, the -1 shows that the program crashes as it shouldn't expect a -1 there.



```
aj87@pc2-020-l:~/Documents/cs2002/W06-Prac/Practical2 $ ./faro_shuffle RANKSUIT
52
0
AC KC QC JC 0C 9C 8C 7C 6C 5C 4C 3C 2C AS KS QS JS 0S 9S 8S 7S 6S 5S 4S 3S 2S AD KD QD JD 0D 9D 8D 7D 6D 5D 4D 3D 2D AH KH QH JH 0H 9H 8H 7H 6H 5H 4H 3H 2H
52
51
AC KC QC JC 0C 9C 8C 7C 6C 5C 4C 3C 2C AS KS QS JS 0S 9S 8S 7S 6S 5S 4S 3S 2S AD KD QD JD 0D 9D 8D 7D 6D 5D 4D 3D 2D AH KH QH JH 0H 9H 8H 7H 6H 5H 4H 3H 2H
IN : AD AC KD KC QD QC JD JC 0D 0C 9D 9C 8D 8C 7D 7C 6D 6C 5D 5C 4D 4C 3D 3C 2D 2C AH AS KH KS QH QS JH JS 0H 0S 9H 9S 8H 8S 7H 7S 6H 6S 5H 5S 4H 4S 3H 3S 2H 2S EoD
IN : AH AD AS AC KH KD KS KC QH QD QS QC JH JD JS JC 0H 0D 0S 0C 9H 9D 9S 9C 8H 8D 8S 8C 7H 7D 7S 7C 6H 6D 6S 6C 5H 5D 5S 5C 4H 4D 4S 4C 3H 3D 3S 3C 2H 2D 2S 2C EoD
OUT: AH 8S AD 8C AS 7H AC 7D KH 7S KD 7C KS 6H KC 6D QH 6S QD 6C QS 5H QC 5D JH 5S JD 5C JS 4H JC 4D 0H 4S 0D 4C 0S 3H 0C 3D 9H 3S 9D 3C 9S 2H 9C 2D 8H 2S 8D 2C EoD
OUT: AH JD 8S 5C AD JS 8C 4H AS JC 7H 4D AC 0H 7D 4S KH 0D 7S 4C KD 0S 7C 3H KS 0C 6H 3D KC 9H 6D 3S QH 9D 6S 3C QD 9S 6C 2H QS 9C 5H 2D QC 8H 5D 2S JH 8D 5S 2C EoD
IN : 6H AH 3D JD KC 8S 9H 5C 6D AD 3S JS QH 8C 9D 4H 6S AS 3C JC QD 7H 9S 4D 6C AC 2H 0H QS 7D 9C 4S 5H KH 2D 0D QC 7S 8H 4C 5D KD 2S 0S JH 7C 8D 3H 5S KS 2C 0C EoD
IN : 2H 6H 0H AH QS 3D 7D JD 9C KC 4S 8S 5H 9H KH 5C 2D 6D 0D AD QC 3S 7S JS 8H QH 4C 8C 5D 9D KD 4H 2S 6S 0S AS JH 3C 7C JC 8D QD 3H 7H 5S 9S KS 4D 2C 6C 0C AC EoD
52
33
AC KC QC JC 0C 9C 8C 7C 6C 5C 4C 3C 2C AS KS QS JS 0S 9S 8S 7S 6S 5S 4S 3S 2S AD KD QD JD 0D 9D 8D 7D 6D 5D 4D 3D 2D AH KH QH JH 0H 9H 8H 7H 6H 5H 4H 3H 2H
IN : AD AC KD KC QD QC JD JC 0D 0C 9D 9C 8D 8C 7D 7C 6D 6C 5D 5C 4D 4C 3D 3C 2D 2C AH AS KH KS QH QS JH JS 0H 0S 9H 9S 8H 8S 7H 7S 6H 6S 5H 5S 4H 4S 3H 3S 2H 2S EoD
OUT: AD AH AC AS KD KH KC KS QD QH QC QS JD JH JC JS 0D 0H 0C 0S 9D 9H 9C 9S 8D 8H 8C 8S 7D 7H 7C 7S 6D 6H 6C 6S 5D 5H 5C 5S 4D 4H 4C 4S 3D 3H 3C 3S 2D 2H 2C 2S EoD
OUT: AD 8C AH 8S AC 7D AS 7H KD 7C KH 7S KC 6D KS 6H QD 6C QH 6S QC 5D QS 5H JD 5C JH 5S JC 4D JS 4H 0D 4C 0H 4S 0C 3D 0S 3H 9D 3C 9H 3S 9C 2D 9S 2H 8D 2C 8H 2S EoD
OUT: AD JH 8C 5S AH JC 8S 4D AC JS 7D 4H AS 0D 7H 4C KD 0H 7C 4S KH 0C 7S 3D KC 0S 6D 3H KS 9D 6H 3C QD 9H 6C 3S QH 9C 6S 2D QC 9S 5D 2H QS 8D 5H 2C JD 8H 5C 2S EoD
OUT: AD 6D JH 3H 8C KS 5S 9D AH 6H JC 3C 8S QD 4D 9H AC 6C JS 3S 7D QH 4H 9C AS 6S 0D 2D 7H QC 4C 9S KD 5D 0H 2H 7C QS 4S 8D KH 5H 0C 2C 7S JD 3D 8H KC 5C 0S 2S EoD
IN : 0D AD 2D 6D 7H JH QC 3H 4C 8C 9S KS KD 5S 5D 9D 0H AH 2H 6H 7C JC QS 3C 4S 8S 8D QD KH 4D 5H 9H 0C AC 2C 6C 7S JS JD 3S 3D 7D 8H QH KC 4H 5C 9C 0S AS 2S 6S EoD
52
45
AC KC QC JC 0C 9C 8C 7C 6C 5C 4C 3C 2C AS KS QS JS 0S 9S 8S 7S 6S 5S 4S 3S 2S AD KD QD JD 0D 9D 8D 7D 6D 5D 4D 3D 2D AH KH QH JH 0H 9H 8H 7H 6H 5H 4H 3H 2H
IN : AD AC KD KC QD QC JD JC 0D 0C 9D 9C 8D 8C 7D 7C 6D 6C 5D 5C 4D 4C 3D 3C 2D 2C AH AS KH KS QH QS JH JS 0H 0S 9H 9S 8H 8S 7H 7S 6H 6S 5H 5S 4H 4S 3H 3S 2H 2S EoD
OUT: AD AH AC AS KD KH KC KS QD QH QC QS JD JH JC JS 0D 0H 0C 0S 9D 9H 9C 9S 8D 8H 8C 8S 7D 7H 7C 7S 6D 6H 6C 6S 5D 5H 5C 5S 4D 4H 4C 4S 3D 3H 3C 3S 2D 2H 2C 2S EoD
IN : 8C AD 8S AH 7D AC 7H AS 7C KD 7S KH 6D KC 6H KS 6C QD 6S QH 5D QC 5H QS 5C JD 5S JH 4D JC 4H JS 4C 0D 4S 0H 3D 0C 3H 0S 3C 9D 3S 9H 2D 9C 2H 9S 2C 8D 2S 8H EoD
IN : 5S 8C JH AD 4D 8S JC AH 4H 7D JS AC 4C 7H 0D AS 4S 7C 0H KD 3D 7S 0C KH 3H 6D 0S KC 3C 6H 9D KS 3S 6C 9H QD 2D 6S 9C QH 2H 5D 9S QC 2C 5H 8D QS 2S 5C 8H JD EoD
OUT: 5S 0S 8C KC JH 3C AD 6H 4D 9D 8S KS JC 3S AH 6C 4H 9H 7D QD JS 2D AC 6S 4C 9C 7H QH 0D 2H AS 5D 4S 9S 7C QC 0H 2C KD 5H 3D 8D 7S QS 0C 2S KH 5C 3H 8H 6D JD EoD
IN : 7H 5S QH 0S 0D 8C 2H KC AS JH 5D 3C 4S AD 9S 6H 7C 4D QC 9D 0H 8S 2C KS KD JC 5H 3S 3D AH 8D 6C 7S 4H QS 9H 0C 7D 2S QD KH JS 5C 2D 3H AC 8H 6S 6D 4C JD 9C EoD
52
-1
AC KC QC JC 0C 9C 8C 7C 6C 5C 4C 3C 2C AS KS QS JS 0S 9S 8S 7S 6S 5S 4S 3S 2S AD KD QD JD 0D 9D 8D 7D 6D 5D 4D 3D 2D AH KH QH JH 0H 9H 8H 7H 6H 5H 4H 3H 2H
Segmentation fault (core dumped)
aj87@pc2-020-l:~/Documents/cs2002/W06-Prac/Practical2 $ 
```

Exit Clause

This was to test that when a -1 is given t the program for the card number it exits. This works for Stackscheck, however when I run the program, I find that after -1 something else must be input before the program is input as seen below.



```
aj87@pc2-020-l:~/Documents/cs2002/W06-Prac/Practical2 $ ./faro_shuffle RANKSUIT
52
1
AC KC QC JC 0C 9C 8C 7C 6C 5C 4C 3C 2C AS KS QS JS 0S 9S 8S 7S 6S 5S 4S 3S 2S AD
IN : AD AC KD KC QD QC JD JC 0D 0C 9D 9C 8D 8C 7D 7C 6D 6C 5D 5C 4D 4C 3D 3C 2D
-1
-1
aj87@pc2-020-l:~/Documents/cs2002/W06-Prac/Practical2 $ ./faro_shuffle RANKSUIT
```

Problems Encountered

The first problem was to do with structs, I had never had to implement them before and the syntax to use them was rather different than I had originally understood, it is however quite intuitive and easy to use. The second problem I encountered, and the most difficult one, was using pointers to navigate arrays and linked lists. As the data structure I used is based on linked nodes and they're referred to using pointers it quickly became confusing and hard to remember what things meant and what. It also became very easy to accidentally lose data and thus it took a while to get the manipulation of the structure correct. The final problem was finding an efficient way of converting to binary and secondly an intuitive way to store it. This was hard because it is process that when performed by a human, they use logic that would be very hard to program, thus I had to find a way that suited a program. Ironically, the shuffle logic was one of the easier parts of the program.

Evaluation

Overall, I'm proud of the code I have produced. It is quite efficient and streamlined and makes a good use of memory. There is probably a better way of converting base 10 to base 2 however for the most part I think the solution I came up with is easily understood and the design structure choice was very suited to the problem. I do realise that the program isn't completely robust and several inputs could crash it however I decided that implementing full validation wasn't necessary due to Stackscheck promises, which leads me to believe that we can expect good input.