CSE 214 - Data Structures
Fall 1997
Program #1

Due: Monday, September 15, 1997

Anyone who has played poker or any other card game is familiar with the idea of shuffling the deck to mix it up after the game. A perfect shuffle splits the deck into two equal halves and then interleaves the cards, one from each half. Perfect shuffles are good for mixing up the deck, or are they?

To be precise about what we mean by perfect shuffle, let us define what happens to the ith card in a deck of $2n$ cards. We will only consider the case where the deck has an even number of cards, to avoid ambiguity about where to split it. If $1 \leq i \leq n$, the card will go to the $2i-1$st position after shuffling, since the first half of the deck all goes to odd positions. If $n+1 \leq i \leq 2n$, the card will go to the $2(i-n)$th position. For example, consider the case of $n = 4$, with a deck of eight distinct cards represented by the numbers 1 to 8. What happens when we repeatedly shuffle the deck:

1	1	1	1
2	5	3	2
3	2	5	3
4	6	7	4
5	3	2	5
6	7	4	6
7	4	6	7
8	8	8	8

After only three shuffles, we are back to where we started from, despite the fact that there are $8! = 40,320$ permutations of the cards. What about a normal deck of 52 cards, which corresponds to $n = 26$? The amazing fact is that eight consecutive perfect shuffles always take you back to the starting point.

Your first program in CSE 214 will compute and print a table of the number of perfect shuffles necessary for a deck of $2n$ cards to return to its initial configuration, for $1 \leq n \leq 100$. A deck of cards can be represented by an array of size $2n$ whose values are the integers between 1 and $2n$. The shuffle operation is described above. You should have a Modula-3 procedure "shuffle" which accepts a deck consisting of an array and a count of the number of cards in the array and which returns the shuffled array.

You are to hand-in the electronic copy of your program files using *handin*. You are also responsible to turn in a printed copy of your program, including the output of the program, which will be a table of the number of shuffles as a function of n, for $1 \leq n \leq 100$. As you can verify by hand, this table begins:

deck size	shuffles
2	1
4	2
6	4
8	3

Note that your programs will be graded for style and clarity as well as correctness. Document your program to make sure the algorithm is understandable.

Challenge Problems

Each assignment will contain one or more extra credit problems for students who want a challenge, or to pursue the subject in greater depth. NOTE THAT THESE PROBLEMS ARE NOT REQUIRED. Sometimes they will require mathematical or programming expertise which go well beyond the class prerequisites. In this case, important ideas can be found by studying modular arithmetic and the cycle structure of permutations.

(1) Can you develop and implement an algorithm for determining the repetition number without actually simulating the effect of shuffling? How does the time your program take for $n = 500$ compare with the two programs?

(2) Can you devise a fairly simple shuffling-like strategy which takes significantly longer to repeat than just repeated perfect shuffles? How long does it take to repeat for the conventional deck of 52 cards?

Rules of the Game

(1) The main purpose of this assignment is to establish that each of you are proficient enough with Modula-3 and the CS UG environment to ensure that you will be ready to meet the challenge of the more demanding programs to come.

(2) Make sure you understand the exact definition of the perfect shuffle operation. If you don't, there is not the slightest chance that your program will work. Work through the example above carefully.

(3) Debugging a program such as this is tricky without knowledge that shuffle is working correctly. Don't be afraid to print out the contents of the deck in an intermediate program. Also, don't be afraid to test it on small values of n before running the complete table.

(4) The extra credit problems are just that - extra credit. The amount of credit you will receive for working on these problems will be fairly small, so interest and curiousity should be the motivation instead of greed. It is not required or expected that you work on the extra credit problems in order to received an A or any other grade.

(5) Include the line (without quotes) 'source ~cse214/login' in your .cshrc file on the UG machines This should set up you account so you can use the handin program.

(6) Don't forget to write out and sign the pledge. "On my honor as a student I have neither given nor recieved aid on this program." Your program will not be graded without it.