CMPS 101
Algorithms and Abstract Data Types
Winter 2006

Programming Assignment 2

Due Wednesday January 25, 10:00 pm

In this assignment you will create a program with the same functionality as **pa1**, but now in C. Our purpose is again threefold: to make sure everyone is up to speed with C (especially pointers and structures), to practice modularity and ADTs, and to build an ADT implementation which will be used (with modifications) in future assignments.

Again you are to write a program that takes as input a single positive integer *n* from the command line, and performs a *perfect shuffle* on *n* cards (in order) until the original sorted order is reproduced, then print out the number of shuffles performed. The program operation will be as described in the **pa1** handout. As before your List ADT will be a double ended queue, with a current-position marker. Read the handout entitled "ADTs and Modules in Java and ANSI C", paying special attention to the section on ANSI C. Also read the handout entitled "Some Additional Remarks on ADTs and Modules in ANSI C". Both handouts are posted on the webpage. Your List module will export a `ListRef` type, and support the following operations.

```
/***** Constructors-Destructors *****/
ListRef newList(void);
void freeList(ListRef* pL);

/***** Access functions *************/
int isEmpty(ListRef L);
int atFirst(ListRef L);
int atLast(ListRef L);
int offEnd(ListRef L);
int getFirst(ListRef L);
int getLast(ListRef L);
int getCurrent(ListRef L);
int getLength(ListRef L);
int equals(ListRef A, ListRef B);

/***** Manipulation procedures ******/
void moveFirst(ListRef L);
void moveLast(ListRef L);
void movePrev(ListRef L);
void moveNext(ListRef L);
void insertBeforeFirst(ListRef L, int data);
void insertAfterLast(ListRef L, int data);
void insertBeforeCurrent(ListRef L, int data);
void insertAfterCurrent(ListRef L, int data);
void deleteFirst(ListRef L);
void deleteLast(ListRef L);
void deleteCurrent(ListRef L);

/***** Other operations **********/
ListRef copy(ListRef L);
void printList(ListRef L);
```

Function `newList` returns a `ListRef` which points to a new empty list. Function `freeList` frees all heap memory associated with its `ListRef*` argument, and sets `*pL` to `NULL`. Function `printList()` prints out the current state of a List object. This function plays roughly the same role as the "toString" function in Java, and will be invaluable when debugging your list module. The other functions are described in the **pa1** specifications. You may also wish define a type called `ListElement`, which is just a synonym for `int`. `ListElement` would then be the return type of several access functions, and input parameter type for several manipulation procedures, making it easy to recast your integer List as a "list-of-something-else" if necessary.

All of the above functions are required for full credit, but you may add additional operations if you like such as the following, whose operation is described in **pa1**.

```
ListRef cat(ListRef A, ListRef B);
void makeEmpty(ListRef L);
```

Your program should be structured in three files: a client program (Shuffle.c), a List implementation file (List.c), and a List header file (List.h). You must also turn in three other files: a Makefile, a driver program (ListClient.c) whose purpose is to test your List module (i.e. taking the place of function main in the Java List class), and a README file describing the files created for this assignment, their purposes, and relationships. Note that the above file names are *not* optional. In addition, your Makefile must create an executable called **Shuffle** for proper grading. Each file you turn in must begin with your name, and cats user ID.

A simple Makefile for this assignment might look like:

```
# Makefile for Programming Assignment 2

Shuffle : List.o Shuffle.o
	gcc -o Shuffle Shuffle.o List.o

Shuffle.o : List.h Shuffle.c
	gcc -c -ansi -Wall Shuffle.c

ListClient: List.o ListClient.o
	gcc -o ListClient ListClient.o List.o

ListClient.o : List.h ListClient.c
	gcc -c -ansi -Wall ListClient.c

List.o : List.h List.c
	gcc -c -ansi -Wall List.c

clean :
	rm -f Shuffle ListClient Shuffle.o ListClient.o List.o
```

The first line is a comment, as are all lines starting with "#". The rest of the file is organized into blocks of the form

```
Target : Dependencies
	Operation
```

separated by blank lines. **Important:** the white space before the “operation” is a tab, not spaces! *Target* is a file to be created, and the *dependency list* for that target consists of those files on which the target depends. If one of the files in the list changes, the target will be recompiled. *Operation* is the command which creates the target. The targets are listed in “top down” order, since make occasionally gets confused if they are listed in another order. Once you have a Makefile, your entire program can be compiled (or re-compiled) simply by typing the unix command “gmake”. This is efficient since only the changed modules will be re-compiled. The target “clean” is known as a phony target. Nothing is created, but an operation is performed. By typing “gmake clean” you remove all old targets. An excellent on-line manual for makefiles is mentioned on the webpage. You can also go to my CMPS 12B Spring 2005 webpage, follow the link to 12M lab assignments, then read lab assignment 1:

```
http://www.soe.ucsc.edu/classes/cmps012b/Spring05/lab.html
```

That lab assignment has a section on Makefiles. Note that the compile operations mentioned in the above Makefile call the gcc compiler. It is a requirement of this and all other assignments in C that your program compile without warnings or errors under gcc, and run properly in the IC Solaris computing environment provided by ITS (Information and Technology Services). In particular you should not use the cc compiler. Information on how to turn in your program is posted on the webpage.