CS 262, Fall 2014, All sections

Project #2 (100p)

Due date: Wednesday Oct. 29 at 11:59pm

In your second project you will implement *singly linked lists* in C. You will the use your implementation to write a program that creates a representation of a deck of cards and makes a random shuffle of the deck. The algorithms for implementing linked lists will be discussed in your classes. Therefore, this description does not list implementation details of the required functions.

Detailed requirements for linked lists implementation:
You will need to write several functions.

0. To represent a node in a linked list you can use a structure

   ```
   struct Node {
       SomeDataType data;
       struct Node *next;
   }
   ```

   where *SomeDataType* can be any C data type.

1. Write a function to make a new list. Your function should create a dummy head node to represent an empty list.

   ```
   struct Node *newList(void) // returns a head of a new empty list
   ```

2. Write functions to insert elements into a list and to delete elements from a list

   ```
   struct Node *delete(struct Node * prev); // deletes the node after prev
   struct Node *insert(struct Node *prev, SomeDataType data)
                   // inserts a new node with data field data after prev
   ```

3. Write functions to count the number of elements in a list without the head and to print the list

   ```
   int length(struct Node *head)        // number of elements in the list
   void printList(struct Node *head)    // print the data fields for the entire list
   ```

   Note that *printList* will print the linked list implemented to support the second part of your project.

In the second part of your project you will write functions to manipulate a deck of cards. You should assume that a full deck contains 52 cards: card values (A,2,3,4,5,6,7,8,9,10,J,Q,K) in four suits (Spades,Diamonds,Hearts,Clubs). You can use either integers or *enum*s to represent cards and suits, but you should print card values using their symbols ('A','2',. . . ,"10",'J','Q','K') and the suits using first letter of their name. Examples: *(10,S),(A,C),(7,D)* to stand for 10 of spades, ace of clubs, and 7 of diamonds.

Detailed requirements for deck of cards manipulation:
You will write a function to make a deck and a function to shuffle a deck of cards. To make a deck you loop through card values and suits and create pairs of values to represent the cards; those pairs are then inserted into the list representing the deck. To shuffle the deck you will write a random shuffle method that works as follows.

```
struct Node *randomShuffle(struct Node *olddeck) // returns new deck
{
  int len = length(olddeck)
  Make a new linked list newdeck
  for (int i=len-1; i>=0; i--)
  {
    j = random()%(i+1)
    move the jth card from olddeck to the front of newdeck
  }
  return newdeck
}
```

You can use *delete* and *insert* functions to move the cards from deck to another. If you know how to do it you can just move a node while setting all links properly.

Instructions for submission:

0. To test your linked list code you should generate 10 random numbers in the range [0..1000] and insert them in a ascending order into an empty list. Every time you generate a number you need to traverse the linked list and find a proper place for the number to be inserted. **This means that you should not sort numbers before insertion.** You should calculate the list length and print it together with the list after each number insertion. Note that for this part to work your data field should be of type *int*.
1. When you are sure that your linked list functions work properly you should modify the code to perform the second part of the project. For this you should create new files and keep the basic working linked list version around. Note that for this part to work your data field should be a structure or array with two values, one for a card value and one for its suit.
2. Use script do show your session in which you compile and run your code for both part #1 and part #2 of the project.
3. Use *tar* command to create an archive of your script file, your fully commented source code, and your Makefile. You should follow these naming rules:

(a) Name your c source file as
<your email account>_<your class section #>_project2.c
For example, *cliu6_section001_project2.c*

(b) Name your script as
<your email account>_<your class section>_project2
For example, *cliu6_section001_project2*
If you used other name for the script, change it by using unix command *mv*.

(c) Your Makefile should include *clean* as well. Makefile name is still *Makefile*.

(d) Use *tar cvf* to compress all files including source code, Makefile and script to
<your email account>_<your section>_project2.tar.
NOTE: DO NOT compress the folder. PLEASE compress files directly.
For example, if all your files are in directory *./Project2*. You do the following steps:

```
> cd ./Project2
> tar cvf cliu6_section001_project2.tar <related file names> or
    > tar cvf cliu6_section001_project2.tar * (* means all files in this directory)
```

(e) Double check that you followed the above procedures before you submit the compressed tar file.

4. Submit your *tar* file on Blackboard.