Lab 5: Structs; Linked Lists and Their Relationship to Arrays

1) Structs

In addition to the built-in variable-length and fixed-length types provided by C (for example, strings and the primitive types we discussed in Lab 3), the user can create special composite types called *structs*. Struct variables are bundles of multiple variables. They can be declared in either the .h or the .c files.

Here is an example of a struct:

```
struct shoppingListItem {
        char *itemName;
        int aisle;
        double price;
};
```

This struct represents a shopping list item. The item has a name and a price, and is found in an aisle. These characteristics are represented by a char *, a double and an int, all in the same struct variable. As can be shown in this example, you want to use structs when several variables are associated together. Here, all these attributes (which you could also call *fields*) are characteristics of the shopping list item.

http://en.wikipedia.org/wiki/Struct_(C_programming_language) contains more information about structs and their syntax.

Problem 1: Create a shopping list item "potatoes", found in aisle 3 with price of $4.99. Print out how many bytes of memory are used to store this shoppingListItem. Also, make sure you allocate and free a memory chunk of the right size for this struct.

2) Linked Lists

The type of variable you can use for a struct field is not only limited to the types C provides. Types you create can also be used in struct fields. User-created types are used in a special way in a data structure called a *linked list.* In a C linked list, each struct contains a pointer to the next struct. For example:

```
struct shoppingListItem {
        char *itemName;
        int aisle;
        double price;
        struct shoppingListItem *nextItem;
};
```

This allows the program to efficiently "read" through the items in the shopping list, if several shopping list items are connected to one another by having each item point to the next item.

Problem 2: Write a program that creates a shopping list of at least 5 items, connected through a linked list. Shopping list items should contain at least 3 fields, and the struct should not contain the same combination of data types as the example. Then have the program read (iterate) through the linked list, printing out the item names and all the other relevant information about the items.

Now insert an item 3rd on the list, and iterate through and print out the list as before. Make sure you allocate and free memory appropriately when you write this program.

3) Linked Lists are an Alternative to Arrays

Like arrays, linked lists can store large numbers of the same data type sequentially. Compared to arrays, linked lists have several advantages and disadvantages, so they are used for different purposes.

It is relatively quick to change the size of a linked list or add an element to the beginning--simply create a new element and connect it to the beginning of an old linked list. However, adding an element to the beginning of an array would require you to move all the elements after it one place over. Changing the size of the array requires you to re-initialize it. Assuming the array is not filled and the linked list's iterator has not passed to the end of the list, it is easier to add an element to the end of the array--none of the elements already in the array would need to move, whereas for the linked list, the iterator would need to step through each element of the list until it reached the end. Only one linked list element at a time is accessible to the iterator.

It is easier to read elements in the middle of arrays, as these elements can simply be accessed by their index. In the linked list, the iterator must step through each previous element.

Problem 3: Create a program called barn, that stores information about, for each pen, whether that pen does or does not contain a cow. Create methods for taking a cow out of a pen, and putting a cow into an empty pen. Use the data structure (linked list or array) you think is more appropriate.

Problem 4: Create a program called CV, in which jobs are listed in reverse-chronological order. The program should contain a method that allows the user to update the CV with the latest job and at least 2 other relevant methods: one that allows the CV to be printed chronologically, and one that allows the CV to be printed reverse-chronologically. The program should be written so chronological and reverse-chronological printing require the computer to perform the same number of steps. Use the data structure (linked list or array) you think is more appropriate. ***WARNING:* This problem is tricky! You need to think very critically about these data structures and what they can and can't do.**

Problem 5: Think of another example of data that lends itself to being stored sequentially. Write a program based on that example, manipulating the data in at least 3 ways. Use either a linked list or an array, depending on what is more appropriate for the example you have chosen.