CSE 373 -- Autumn 1999 -- Assignment #2

(*due Monday, October 18*)

1. **For each of the following questions, assume that we want to write a routine called `DeleteMax()`. This routine should search the data structure in question for the largest integer value that it contains, delete that value from the data structure, and return it to the user. You may assume that the data structures will not contain duplicate values.**

   (a) Write `DeleteMax()` for a List ADT, assuming that you do not have access to the List ADT's implementation -- you may only use the standard List operations as described in lecture (or the book). Your routine should have the following prototype:

   ```
   int DeleteMax(List<int>& myList);
   ```

   (b) Repeat part (a), but assume that you are writing `DeleteMax()` as a new method for an integer array-based List ADT. You should write the method by directly accessing the ADT's fields -- not by using its operations. Here's the class definition:

   ```
   class IntList {
     private:
       int *data;
       int length;
     public:
       int DeleteMax();
       ...
   };
   ```

   (c) Repeat part (b), but now for a singly link-based List implementation with a dummy node at its head. Here are the relevant definitions:

   ```
   class IntListNode {
     int data;
     IntListNode *next;
   }

   class IntList {
     private:
       IntListNode *head;
     public:
       int DeleteMax();
       ...
   };
   ```

   (d) Repeat part (a), but assume that your data is stored in a Queue rather than a List. You may only use the standard operations supported by the Queue ADT. If it helps, you may assume that the Queue only contains non-negative numbers. Here's the prototype:

   ```
   int DeleteMax(Queue<int>& myQueue);
   ```

   (e) Repeat part (a), but assume that your data is stored in a Stack. Use only the standard Stack operations. (*Hint:* you may use additional stacks as necessary). Here's the prototype:

```
int DeleteMax(Stack<int>& myStack);
```

(f) Give the asymptotic running time and space required for each of your implementations above.

2. **Exercise 3.17 (a)**

3. **One of the disadvantages of `float` and `double` variables in C or C++ is that they can only represent values with a finite amount of decimal precision (this is due to the fact that they use a fixed amount of storage per value). For example, there is no way to represent the exact value 1/3 as a `float`, or even to represent more than a certain number of decimal places.**

   (a) Propose a List-based implementation of floating point values that can store values with a user-specified precision. Imagine that users construct these values by supplying a value `v` (using some arithmetic expression) and a precision `p`. For example, for `v = 100/3`, `p = 3` would cause `33.333` to be stored, whereas `p = 6` would cause `33.333333` to be stored.

   (b) Would you use an array-based or linked list-based List implementation for your floating point representation? Why?

   (c) Show what the values 12.3456 and 1234.56 would look like in your implementation (sketch out a picture that displays what the data structure would look like)

   (d) How much space is required by your representation, asymptotically speaking?

   (e) Give code for an `Add()` operation for your floating point representation. The operation should add two values stored in your representation and return the sum as a new value in your representation. You may assume that both values are positive, and you may use standard List operations or access the internal representation directly.

   (f) What is the asymptotic running time for your `Add()` operation?