Assignment 7

Radityo Eko Prasojo, Werner Nutt,
Rafael Penaloza, Guohui Xiao

List Operations and Sorting with Lists

1. Operations on Linked Lists

Implement a data type `List` that realizes linked lists consisting of nodes with integer values, as discussed in the lecture. Remember that:

- an object of type `Node` has two fields, an integer `val` and (a pointer to) a Node `next`;
- an object of type `List` has one field, (a pointer to) a Node `head`;

The type List must have the following methods:

1. `boolean isEmpty();`
2. `int length();`
   returns the number of nodes in the list, which is 0 for the empty list;
3. `void print()`
   print the content of all nodes;
4. `void addAsHead(int i)`
   creates a new node with the integer and adds it to the beginning of the list;
5. `void addAsTail(int i)`
   creates a new node with the integer and adds it to the end of the list;
6. `void addSorted(int i)`
   creates a new node with the integer and adds it behind all nodes with a `val` less or equal the `val` of the node, possibly at the end of the list;
7. `Node find(int i)`
   returns the first node with `val i`;

8. `void reverse()`
   reverses the list;

9. `int popHead()`
   returns the value of the head of the list and removes the node, if the list is nonempty, otherweise returns `NULL`;

10. `void removeFirst(int i)`
    removes the first node with `val i`;

11. `void removeAll(int i)`
    removes all nodes with `val i`;

12. `void addAll(List l)`
    appends the list `l` to the last element of the current list, if the current list is nonempty, or lets the `head` of the current list point to the first element of `l` if the current list is empty.

(12 Points)

2. Operations on Head-Tail Lists

Implement a data type `HTList` that realizes head-tail lists consisting of nodes with integer values, as also discussed in the lecture. Remember that:

- an object of type `Node` looks as before;
- an object of type `HTList` has two fields, (a pointer to) a Node `head` and (a pointer to) a Node `tail`;

Modify the methods for the type `List` from the previous exercise so that they work for head-tail lists. You will find that some methods need not be changed at all, while for others you also have to manage the `tail` pointer.

(6 Points)

3. Sorting with Lists

In this exercise, we want to realize list versions of sorting algorithms that we know already for arrays.

1. Develop a version of Insertion Sort for linked lists. Realize it as a method

   ```
   void insertionSort()
   ```

that turns the current list into a sorted list.

Hint: The idea of Insertion Sort is to repeatedly insert nodes into a sorted list such that the order is preserved. Check out which of the methods defined for your linked list type contain ideas that can be used for implementing `insertionSort`.

2. Develop a version of Quicksort for head-tail lists. Realize it as a method

   ```
   void quickSort()
   ```

   that turns the current list into a sorted list.

   Hint: The idea of Quicksort is to repeatedly choose an element of the current list as pivot and to partition the current list into two lists, one with elements less or equal than the pivot value and another one with elements greater or equal than the pivot value. Then, the two new lists are each sorted recursively and appended.

   Check which of the methods defined for your head-tail list type can be used to implement `quickSort`.

(12 Points)

Deliverables.

1. Your implementation of the classes in Exercises 1-3

Combine all deliverables into one zip file, which you submit via the OLE website of the course. Please, follow the "Instructions for Submitting Course Work" on the Web page with the assignments, when preparing your coursework.
Also, include name, student ID, code of your lab group (A, B, or C), and email address in your submission.

Submission: Until Mon, 16 May 2015, 11:55 pm, to

Lab A / Lab B / Lab C