**Assignment 6**

**Mouna Kacimi, Werner Nutt, Camilo Thorne**

# List Operations and Sorting with Lists

**Instructions:** Your assignment should represent your own effort. However, you are not expected to work alone. It is fine to discuss the exercises and try to find solutions together, but each student shall write down and submit his/her solutions separately. It is good academic standard to acknowledge collaborators, so if you worked together with other students, please list their names.
You can write up your answers by hand (provided your handwriting is legible) or use a word processing system like Latex or Word. Experience shows that Word is in general difficult to use for this kind of task.
For a programming task, your solution must contain (i) an explanation of your solution to the problem, (ii) the Java code, in a form that we can run it, (iii) instructions how to run it. Also put the source code into your solution document. For all programming tasks, it is not allowed to use any external libraries ("import") if not stated otherwise.
Please, include name and email address in your submission.

## 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. `void print()`
   print the content of all nodes;

3. `void addHead(Node n)`
   adds the node to the beginning of the list;
4. `void addTail(Node n)`
   adds the node to the end of the list;
5. `void addSorted(Node n)`
   adds the node behind all nodes with a `val` less or equal the `val` end of the list;
6. `Node find(int i)`
   returns the first node with `val i`;
7. `void reverse()`
   reverses the list;
8. `Node popHead()`
   returns the head of the list and removes it, if the list is nonempty, and returns `NULL` otherwise;
9. `void removeFirst(int i)`
   removes the first node with `val i`;
10. `void removeAll(int i)`
    removes all nodes with `val i`;
11. `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.

(8 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 which of the methods defined for your linked list type can be used to implement `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 of 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`.

(10 Points)

**Instructions:** Write a short report where you say for every method how you realized it.
Submit code for a package `MyLists` that contains the three classes `Node`, `List` and `HTList`, each class with the methods specified above.

Submission: Until Thu, 8 May 2014, 8:30 pm, to

`dsa-submissions AT inf DOT unibz DOT it.`