ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Lecturer: B. Gärtner

Informatik für Mathematiker und Physiker HS12

Exercise Sheet 11

Submission deadline: 3.15pm - Tuesday 4th December, 2012

`Course URL`: http://www.ti.inf.ethz.ch/ew/Lehre/Info1_12/

Note: You should test your implementations thoroughly. For each class you altered, write a program that tests all of its functionality.

Assignment 1 - (4 points)

From the course web page, download the following files:

- Node.h - header file of the `Node` class
- Node.cpp - implementation of the `Node` class
- List.h - header file of the `List` class
- ListSkeleton.cpp - implementation of the `List` class

Inspect the provided code. Your task is to implement the method `reverse()` in the `List` class that is partially implemented in the ListSkeleton.cpp file. You should implement `reverse()` so that the reversal is done *in-place*. This means that you do *not* create any temporary nodes or lists. Also, the number of operations needed by your `reverse()` should be linear in the size of your list. What this means is that if n is the size of the list, then there exist $k, c \in \mathbb{N}$ such that the number of operations needed by `reverse()` is less than $kn + c$ for any list size n.

Assignment 2 - (4+4+4 points)

From the course web page, download the following files:

- DLNode.h - header file of the `DLNode` class (i.e. class representing a doubly linked node)
- DLNode.cpp - implementation of the `DLNode` class
- SortedDLList.h - header file of the `SortedDLList` class
- SortedDLListSkeleton.cpp - implementation of the `SortedDLList` class

Inspect the provided code. The class `SortedDLList` represents a Doubly Linked List that is sorted the ascending order. Your task is to implement the following methods in the `SortedDLList` class that is partially implemented in the SortedDLListSkeleton.cpp file:

1. `insert(int value)` - insert a new node with the given `value` into a correct position in a list (i.e. the list is sorted afterwards)

2. `remove (const DLNode* n)` - removes the given node n from the list. Your implementation should remove the given node using *constant* number of operations for any given n.

3. `merge(const SortedDLList& l)` - merges `*this` list with the given sorted doubly linked list l. Your implementation should use a liner number of operations in the length of the lists. I.e., let s_1 and s_2 be the size of `*this` and `l`, respectively. Then there exists $k, c \in N$ such that the number of operations executed by merge(const SortedDLList& l) is less than $k(s_1 + s_2) + c$.

Challenge - (8 points)

Implement the method `sort()` in the class `List` from **Assignment 1**. After this method is called, `*this` is sorted in ascending order.