Lab 3 – Exercises

EXERCISE 1. Consider the provided folder `DList_template`, which contains an incomplete implementation of doubly linked lists.

(i) Implement the missing member functions for insertion and deletion of list elements and for sublist checking. You need to rename `dlist_template.cpp`.

The completed implementation is provided in the folder `DList` after the lab.

(ii) Test your implementation by using either the provided program `dlistmgt.cpp` or by creating your own test program.

(iii) Further explore the functions on doubly linked lists and compare the performance with the implementation of unbounded arrays.

EXERCISE 2. Define a modified class SLIST, which represents singly linked lists with an additional pointer from the dummy node to the last node of the list.

Reuse as much as possible the code for doubly linked lists including your own extensions from Exercise 1.

EXERCISE 3.

(i) Explore a list-of-blocks representation for sequences, a simple data structure for sequences that combines the advantages of linked lists and unbounded arrays and is more space-efficient. Use a linked list, where each item stores an array of K elements for some large constant K.

(ii) Implement such a data structure. Reuse as much as possible the implementations of `AList` and `DList` (or `SList`). Exploit that each list index is given by a block index plus an offset into the array representing the block.

EXERCISE 4.

(i) Implement the *insertion sort* algorithm by adding a new member function `isort` to either `AList` or `DList`.

(ii) Investigate for `AList` and `DList`, if *insertion sort* can be implemented in place, i.e. by using just the space occupied by the list plus a few additional memory with total space not depending on the length of the list.

(iii) Sketch an in-place implementation of *insertion sort*, if possible.