Exercises: Lists (Theory)

- Exercises about linked lists: 1 to 7, 12 to 14
- Exercises about doubly linked, circular and ordered lists: 8-11
- Throughout these exercises you can use list operations to implement your programs

(1) Implement the methods *retrieve*, *next*, *previous* and *first* of a linked list
   - *retrieve* - returns the element at a position p on a list L. The result is undefined if p = END(L) or if L has no position p.
   - *next* and *previous* - return the positions following and preceding a position p on a list L. If p is the last position on L, then next will return END(L). next is undefined if p is END(L). previous is undefined if p is 1. Both return undefined if L has no position p.
   - *makenull* - causes L to become an empty list and returns position END(L).
   - *first* - returns the first position on list L. If L is empty, the position returned is END(L).

   Determine the running time of each method.

(2) Implement the method that returns the lenght of a linked list. Determine its running time. Devise and describe a strategy to improve the running time and implement it.

(3) Extend the basic List ADT to include the operations *concatenateLists* and *splitList*. *concatenateLists* is a function to concatenate two lists. *splitList* is a function to split a list in two. *splitList* must receive the position at which split must occur as an argument. Implement both methods on linked lists. Compute the running time of both.

(4) An alternative specification of the basic list ADT can be:

```
spec LIST[ITEM]
     genres list, item, position
     operations
          insert:item natural list->list
          delete:natural list->list
          locate:item list->natural
          retrieve:natural list->item
          next:item list->item
          previous:item list->item
          makenull:list->list
          first:list->item
endspec
```

   Notice that the functions *insert*, *delete* and *retrieve* now receive a natural number (instead of a position) as an argument, and that the function *locate* now returns a natural too. Provide implementations for these functions on a linked list, then compute their running time and compare each with the given specification.

(5) Provide a linked list implementation for the methods *end*, *rest*, *last*, *modify* and *onList* for the extended specification of a list given.
   - *rest* returns a list containing all the elements on the list except for the first one.
   - *last* returns the last element on the list.
   - *modify* gets a position p and an element e an assigns e to the data element at position p on the list
   - *onList* gets an element and returns true if that element is on the list or false otherwise.

   Indicate the running time of each method.

(6) Write a method to interchange the elements at positions p and *next(p)* in a singly linked list. What is it running time?

(7) Another array representation of lists is to delete by simply replacing the deleted element by a special value "deleted," which we assume does not appear on lists otherwise. Rewrite the list operations to implement this strategy. What are the advantages and disadvantages of the approach compared with our original array representation of lists?

(8) Write the methods of the basic specification of the list ADT to operate on doubly-linked list. Are there any differences with the running time of the singly linked list?

(9) Implement the last operation on: (a) a singly linked list, (b) a singly circular linked list, and (c) a doubly linked circular list. Compare their running time.

(10) Write programs to insert, delete, and locate an element on a sorted list using array and pointer implementations of lists. What is the running time of each of your programs? Justify your answer.

(11) Write a program to merge
a) two sorted linked lists,
b) *n* sorted linked lists.
Calculate the running time of each program. Extend the list ADT to include these functions.

(12) A list of lists is a list of elements of the type list. Create a specification for this ADT that includes the following operations: *insertlist*, *deletelist*, *retrievelist*, *nextlist*, *previouslist*, *firstlist* and *concatenateNLists*. *ConcatenateNLists* will concatenate the first N lists of the data structure in a single list that will be located in the first list of the resulting data structure. Implement a method for the function *concatenateNLists* and compute its running time. Write also a method to print all data of a list of lists elements and compute its running time. Tip: The operations of the basic list ADT can be called.

(13) Implement the operations of the stack ADT in terms of the list ADT. That is, call list operations to implement stack operations whenever possible. Analyze the running time of each operation and compare it with the standard stack.

(14) Implement the operations of the queue ADT in terms of the list ADT. That is, call list operations to implement queue operations whenever possible. Analyze the running time of each operation and compare it with the standard queue.