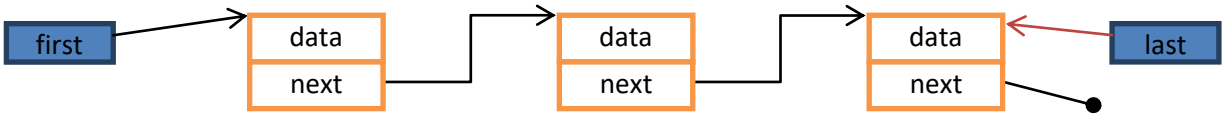
Algorithms and computational complexity

Laboratory 4: Linked lists.

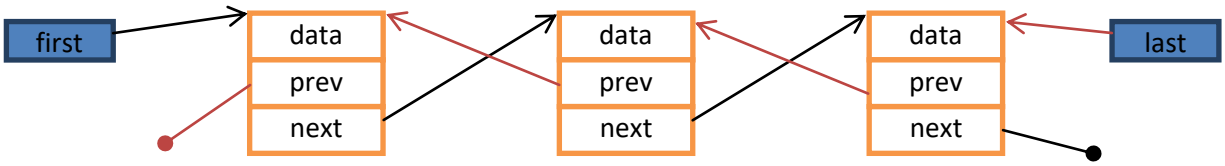
1. **One-way linked list concept.**

   A one-way linked list is a data structure that allows you to store data in an orderly form, as well as to quickly insert and delete items to and from the list. The list consists of elements containing data and a pointer (or reference) to the next element. This way, the class representing a list stores only the reference to the first element of the list (optionally also the last one) in order to remember the entire list. The structure of the one-way list is illustrated in the figure below:



2. **Two-way linked lists.**

   Two-way lists are ordered, linear data structures. They differ from one-way lists only in that the navigation between elements of the list can be performed in two directions. A typical bi-directional list consists of elements that, in addition to the data stored in the list, contain information about the previous and next elements of the list. The structure of the two-way list is illustrated in the figure below:



3. **Generic classes.**

   Generic classes are classes with parameterized data types. Generic classes have a complete implementation, but they do not define the data types used in this implementation. For example:

```
public class Element<T> {
    private T data;
    private Element<T> next, prev;

. . .
}
```

   The parameter of the Element class is some type T. We can now use this class instance to store different types of data. E.g. (after defining the appropriate constructor):

```
Element<Integer> e = new Element<Integer>(5);
Element<String> e2 = new Element<String>("Ala ma kota");
```

   Since Java 7, the Java compiler has a built-in type inference mechanism for deduction of generic types. Thus, we do not have to repeat parameters (types) each time. Therefore, the following expression is valid:

```
Element<Integer> n = new Element<>(5);
```

   The generic programming mechanisms available in Java should be used.

   Attention! The types given as parameters must be reference types! E.g. use *Integer*, not *int*.

4. Tasks.

1. Create a new Java project using NetBeans, Eclipse or InteliJ IDE. Create the package ***aiz.list***. Download the *ElemOne* and *ListException* classes and the *IList* interface from the website and put them in the aiz.list package. In this package, create the ListOne class that implements the IList interface, and then use the "generate abstract methods" (or similar) option to generate an empty implementation of all interface methods. **[1p]**

2. Implement all the methods of the *ListOne* class that implement the *IList* interface. This class is intended to be a generic one-way list using ElemOne objects as list items.
   Hint: in the *ListOne* class, declare the fields *first* and *last* - references to the first and last element of the list, as well as the field *count* to keep the number of elements. **[3p]**

3. Write a program that uses all the methods of the ListOne class to demonstrate their operation. **[1p]**

4. Create a class that represents the bidirectional list item (e.g. *ElemTwo*). Implement a bidirectional list that implements the IList interface. Write a program to show the operation of a bidirectional list. **[3p]**

5. Implement the ***IList <T> join (IList <T> second)*** method that joins two lists and returns a new list. Make a deep copy of the items so that the resulting list elements are completely independent of those from input lists. **[1p]**

6. Implement the method: ***boolean similar (IList <T>)*** that checks if the given list is similar (to the one for which the method is called, *this*). The method is supposed to return true only if one list is a permutation of the other (both contain the same number of the same values but in any order). **[1p]**

Homework:

1. Write a program that reads only even numbers from a text file to a list.
2. Write a program that reads numbers from a text file into a bidirectional list and sums every third number.
3. Write a program that reads integers into a list and calculates:
   a.maximum and minimum value,
   b. sum,
   c. arithmetic mean,
   d. standard deviation.
4. Write a program that reads numbers from a file or from the keyboard in such a way that the list is sorted at every moment of the program's operation.
5. Implement the ***iterator*** () method returning an object implementing ***Iterator <T>***.
   https://docs.oracle.com/javase/7/docs/api/java/util/Iterator.html

Next week:

Graphs. Graph representations. Graph algorithms.