PWr 2013-03-07
Algorithms and Data Structures, summer semester 2012/2013
Laboratory – **List 3**

1. Write a program with below operations for a two-way **unordered** cycled list (without sentinel):
    a. **void** `init(List2W& l)` – which initialize the list `l`.
    b. **void** `insertHead(List2W & l, int value)`- insert an element with `value` as a head (first element) in a list `l`.
    c. **int** `deleteHead(List2W & l)`- remove a head (first element) from a list `l`, and return a value of the head. If the head does not exist, return -9999
    d. **void** `insertTail(List2W & l, int value)`- insert an element with `value` as a tail (last element) in a list `l`.
    e. **int** `deleteTail(List2W & l)` - remove a tail (last element) from a list `l` and return a value of the tail. If the tail does not exist, return -9999
    f. **int** `findValue(List2W & l, int value)` - find first element in list `l` with value and return its position (starting from 0). If there is no such element, return -1;
    g. **void** `removeAllValue(List2W & l, int value)` – remove from list `l` all elements which are equal to `value`. If there is no such element, do nothing.
    h. **void** `showListFromHead(List2W & l)` - show elements of list `l` starting from the head. The values are written in one line, separated by one space. If a list is empty – the line is empty. The line ends with newline character.
    i. **void** `showListFromTail(List2W & l)` - show elements of list `l` starting from the tail. The values are written in one line, separated by one space. If a list is empty – the line is empty. The line ends with newline character.
    j. **void** `clearList(List2W & l)` - remove all elements from list l.
    k. **void** `addList(List2W & l1, List2W & l2)` - move all elements from list `l2` to list `l1`. The order of elements does not change, elements from `l2` are after elements from `l1`. After this operation the list `l2` is empty. If `l1` and `l2` are the same list – do nothing.

    Format of a stream on judgment system is presented in appendix 1. Prepare 2-3 interesting tests using this format.

For **10 points** present solutions for this list till **2013-03-19**.
For **7 points** present solutions for this list till **2013-03-26**.
After 2013-03-26 the list is closed.

Appendix 1

The solution will be automated tested with tests from console of presented below format. The test assumes, that there are up to X different lists, which there are created as the first operation in the test. Each list can be initialized separately.

If a line starts from '`#`' sign, the line have to be ignored.

If a line has a format:
`GO` *n*
your program has to create *n* lists (without initialization). The lists are numbered from 0 like an array of lists. Default current list is a list with number 0.

If a line has a format:
`CH` *n*
your program has to choose a list of a number `n`, and all next functions will operate on this list. There is *n>=0*.

If a line has a format:
`IN`
your program has to call `init(l)` for current list `l`. For any list this operation will be called once.

If a line has a format:
`IH` *x*
your program has to call `insertHead(l,x)` for current list `l`.

If a line has a format:
`DH`
your program has to call `deleteHead(l,x)` for current list `l` and write the return value on the console.

If a line has a format:
`IT` *x*
your program has to call `insertTail(l,x)` for current list `l`.

If a line has a format:
`DT`
your program has to call `deleteTail(l,x)` for current list `l` and write the return value on the console.

If a line has a format:
`FV` *x*
your program has to call `findValue(l,x)` for current list `l`, and write on output returned value.

If a line has a format:
`RV` *x*
your program has to call `removeAllValue(l,x)` for current list `l`.

If a line has a format:
`SH`
your program has to call `showListFromHead(l)` for current list `l`.

If a line has a format:
`ST`
your program has to call `showListFromTail(l)` for current list `l`.

If a line has a format:
`CL`
your program has to call `clearList(l)` for current list `l`.

If a line has a format:
`AD n`
your program has to call `addList(l, l2)` for current list `l` and for list l2 which is the *n*'th list in the array of lists

If a line has a format:
`HA`
your program has to end the execution, writing as the last line "END OF EXECUTION". Every test ends with this line.

For example for input test:

```
GO 2
IN
IH 1
IH 2
CH 1
IN
IH 4
AD 0
SH
CH 0
ST
IT 1
ST
HA
```

The output have to be:

```
4 2 1

1
END OF EXECUTION
```