PWr Week 2
Data Structures and Algorithms,
Laboratory – **List 2**

1. Implement **finite** queue with **only one** organizing index (inefficient implementation). Implement operation:
   a. **void** `init(Queue& q, int size)` – which initialize the queue `q`.
   b. **bool** `enqueue(Queue& q, int value)` – which put a `value` on the end of queue `q and return true`. If there is no place – only return false;
   c. **bool** `dequeue(Queue& q, int &value)` – which remove and return under `value` an element from the front of a queue `q`. In this case the function return true. If there is no element – only return false;
   d. **bool** `isEmpty(Queue q)` – return **true** if queue `q` is empty, otherwise - **false**;
   e. **bool** `isFull(Queue q)` – return **true** if queue `q` is full, otherwise - **false**;
   f. **void** `show(Queue q)` – show elements of a queue q starting from the front. The values are written in one line, after each value write one comma (e.g. "1,2,3,"). If a queue is empty – the line is empty. The line ends with newline character.

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

2. Write a program with below operation for a **one-way unsorted straight linked** list:
   a. **void** `init(List& l)` – which initialize the list `l`.
   b. **void** `insertHead(List& l, int elem)`- insert an element `elem` as a head (first element) in a list `l`.
   c. **bool** `deleteHead(List& l, int &oldhead)`- remove a head (first element) from a list `l`. Return **true** if operation was successful and under parameter `oldHead` function has to return value of the head. Otherwise return **false**.
   d. **void** `insertTail(List& l, int elem)`- insert an element `elem` as a tail (last element) in a list `l`.
   e. `bool deleteTail(List& l, int &oldTail)` - remove a tail (last element) from a list `l`. Return **true** if operation was successful and under parameter `oldTail`, function has to return value of the head. Otherwise return **false**.
   f. **int** `findPosOfValue(List& 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. **bool** `deleteValue(List& l, int value)` – remove from list `l` first element which is equal to `value` and return **true**. If there is no such element, do nothing and return **false**.
   h. **bool** `atPosition(List& l, int pos, int &value)` – find in list `l` an element on specified position `pos`. Return **true** and a value on this position. If a position does not exist, only return **false**;
   i. **void** `showListFromHead(List& l)` - show elements of list `l` starting from the head. The values are written in one line, after each value write one comma (e.g. "1,2,3,". If a list is empty – the line is empty. The line ends with newline character.
   j. **void** `clearList(List& l)` - remove all elements from list l.

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

For 10 points present solutions for this list till Week 3.
For 8 points present solutions for this list till Week 4.
For 5 points present solutions for this list till Week 5.
After Week 5 the list is closed.

Appendix 1 (for a queue).

The solution will be automated tested with tests from console of presented below format. The test assumes, that there are up to X different queues, which there are created as the first operation in the test. Each queue can be initialized with different size, and it will be done in consecutive operations.

If a line is empty or starts from `'#'` sign, the line have to be ignored.
In any other case, your program should print an exclamation mark and write (copy) introduced a line and then, depending on the command follow the correct procedure / function.

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

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

If a line has a format:
`IN` *n*
your program has to call `init(q, n)` for current queue `q`. There is $n>=1$. For any queue this operation will be called once.

If a line has a format:
EN *x*
your program has to call `enqueue(q,x)` for current queue `q`. Write on output one line with a returned boolean value.

If a line has a format:
DE
your program has to call `dequeue(q,x)` for current queue `q` and write on output one line with value `x`. If there are no elements in the queue – write one line with "false".

If a line has a format:
EM
your program has to call `isEmpty(q)` for current queue `q`, and then depending on return value, write on output one line with text "true" or "false".

If a line has a format:
FU
your program has to call `isFull(q)` for current queue `q`, and then depending on return value, write on output one line with text "true" or "false".

If a line has a format:
SH
your program has to call `show(q)` for current queue `q`, which write one line on output with values after each write one coma (without spaces), e.a. string "4,6,1,"

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 3
EN 1
EM
EN 3
EN 4
FU
EN 5
DE
SH
CH 1
IN 5
EM
FU
CH 0
EN 6
SH
HA
```

The output have to be:

```
START
!GO 2
!IN 3
!EN 1
true
!EM
false
!EN 3
true
!EN 4
true
!FU
true
!EN 5
false
!DE
1
!SH
3,4,
!CH 1
!IN 5
!EM
true
!FU
```

```
false
!CH 0
!EN 6
true
!SH
3,4,6,
!HA
END OF EXECUTION
```

Appendix 2 (for a linked list).

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 is empty or starts from `'#'` sign, the line have to be ignored.
In any other case, your program should print an exclamation mark and write (copy) introduced a line and then, depending on the command follow the correct procedure / function.

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 on output one line with value `x`. If there are no elements in the list – write one line with "false".

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 on output one line with value `x`. If there are no elements in the list – write one line with "false".

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

If a line has a format:
`DV x`
your program has to call `deleteValue(l,x)` for current list `l`, , and write on output returned value (true or false).

If a line has a format:
`AT x`
your program has to call `atPosition(l,x)` for current list `l`, and write on output returned value or "false" if there is no such position.

If a line has a format:
`SH`
your program has to call `showListFromHead(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:
`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
IT 3
SH
FV 2
AT 0
DH
DT
FV 2
AT 2
HA
```

The output have to be:

```
START
!GO 2
!IN
!IH 1
!IH 2
!IT 3
!SH
```

```
2,1,3,
!FV 2
0
!AT 0
2
!DH
2
!DT
3
!FV 2
-1
!AT 2
false
!HA
END OF EXECUTION
```