Übungen zur Vorlesung Informatik II (D-BAUG) FS 2017

D. Sidler, F. Friedrich

http://lec.inf.ethz.ch/baug/informatik2/2017

Solution to exercise sheet # 7 **3.4.2017 – 11.4.2017**

Problem 7.1. FIFO List

In this exercise we adapt the Stack (Stapel) implementation which was presented during class with the functionality of a First-in-First-out List. This means we insert elements at one end, while removing them from the other end. You can find the code template of the assignment here: https://codeboard.ethz.ch/inf2baugex07t01

As shown in class, the list consists of two classes.

```
public class ListNode {
    private int data;
    private ListNode next;
    ... // Constructors + Access methods + toString
}
```

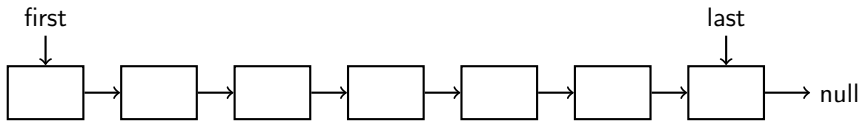
represents a single element of the list and its data.

The class `FifoList` uses objects of `ListNode`, in order to store a list of `integer`-numbers. It has the following structure:

```
public class FifoList {
    private ListNode first; // reference of the first element
    private ListNode last; // reference of the last element

    public FifoList() // constructor
    public String toString() // human readable form
    public void insert(int data) // insert an element
    public int remove() // get "oldest" element and delete it from List
}
```

Your first task is to implement the two methods `toString` and `insert`. Before you start coding have a look at the following figure



and think about the following: Given that you can only reach an element by following the directed arrows starting from either first or last

- Which side is the easiest to enter a new element?
- Which side is the easiest to delete an element?

Keep in mind that for both operations you might also need to change arrows of predecessors in the chain. (There should be a clear preferred side to insert, and the opposite side should clearly be the better to delete)

To test your implementation we provide `main` function which reads in the following commands:

```
push 12
pop
```

The `push` command inserts the given value, in this example 12. The `pop` command removes an element and returns it.

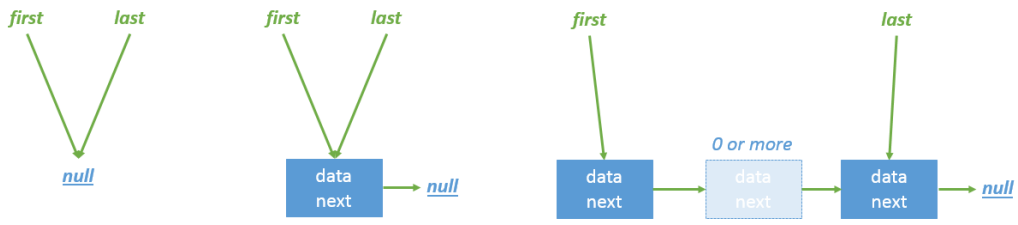


Figure 1: different states a FiFo list can be in (from left to right)

1. *An empty FiFo list - first and last are both null*
2. *An FiFo list with only a single element - first and last are equal*
3. *A FiFo list with at least two or more elements*

Insert

```
//pre: the list with m elements,
//       where m can be any positive integer including 0 (empty list)
//post: the list with m+1 elements, with one element added by you
//          (ListNode with data "n")
public void insert(int n)
```

Take care that you cover all cases of states the list can be in as sketched in 1, such that inserting an element works for each of those cases. Also note that inserting

- in state 1 should bring you to state 2
- in state 2 should bring you to state 3
- in state 3 should keep you in state 3 but with one more element

Converting the list into a printable string

Your next task is to fill the method:

```
//pre: -
//post: return a string that is simply the concatenation of calling
//            the toString method of all list elements.
//            make sure that you print in insertion order.
public String toString()
```

To implement this method you will have to traverse the list. You start by creating an empty string and then, while traversing the list, call the `toString` method of each element. After each call you concatenate the string with your result so far. Pay attention that you print your list in the insertion order (first inserted first, last inserted last).

You can validate your implementation of `toString` and `insert` by un-commenting the annotation `@RunTests`. You should already pass the two tests `Insert 1` and `Insert 2`. To pass the remaining tests you have to implement the remove functionality described below.

Remove (removing oldest element)

Your final task is to implement the routine Remove which will remove the oldest element.

```
//pre: the list with m elements,
//    where m can be any positive integer including 0 (empty list)
//post: if the list is empty return 0 and leave the list empty,
//     otherwise return the "data" value of the oldest element
//     and remove it from the list
public int remove()
```

Your Remove routine has to remove on the opposite end of where your Insert method inserts elements. Also in this case make sure that you cover each case your list can be in. Make sure that you re-adjust the pointers first/last accordingly after removing an element.
You can check your complete implementation (insert, toString and remove in combination) by uncommenting the annotation `@RunTests` again. Now you should be able to pass all tests.

Solution of Problem 7.1.

```
class FiFoList {
	private ListNode first; // reference of the first element
	private ListNode last; // reference of the last element (aids fast
	    adding)

	public FiFoList() {
		first = null; //
		last = first; // indicates an empty list (both first and last are
		    null)
	}

	//pre: -
	//post: return a string that is simply the concatenation of calling
	//		the toString method of all list elements.
	//		make sure that you print in insertion order
	public String toString() {
		ListNode le = first; // starting point
		String str = "";
		while (le != null) {
			str += le.toString(); // adds the Strings
			le = le.next(); // go from one element to the next one
		}
		return str;
	}

	//pre: the list with m elements, where m can be any positive integer
	    including 0 (empty list)
	//post: the list with m+1 elements, with one element added by you
	    (ListNode with data "n")
	//		you can choose on which side you add the element, as long
	    you remove from the other and traverse correctly
	//		when printing
	public void insert(int n) {
		ListNode le = new ListNode(n,null); // create new element
		//this assumes that both have to null at the same time always
		if (last == null) {
			first = le;
			last = le;
		}
		else {
```

```
                last.setNext(le);
                last = le;
        }
    }


    //pre: the list with m elements, where m can be any positive integer
        including 0 (empty list)
    //post: if the list is empty return 0 and leave the list empty,
    //            otherwise return the "data" value of the oldest element
        and remove it from the list
    public int remove() {
        if(first == null) {
                return 0;
        }
        else {
                ListNode le = first;
                // in case we only have one element
                if (last == first) {
                        first = last = le.next();
                }
                else {
                        first = le.next();
                }
                return le.getData();
        }
    }
}
```