

Homework 6

You have to submit your solutions as announced in the lecture.

Unless mentioned otherwise, all problems are due 2017-03-23, 11:00.

There will be no deadline extensions unless mentioned otherwise in the lecture.

As before, you are not allowed to use any existing implementations in your programming language's libraries. Exception: You may use libraries for concepts we have learned previously, e.g., for linked lists.

Problem 6.1 *Stacks*

Points: 8

In any object-oriented programming language, implement the data structure of mutable stacks. Make sure that all stack operations take constant time.

This should be a class with a mutable field for an immutable linked list. We say that the stack is *backed* by an immutable list.

Write a test program that

- creates a new stack of integers
- pushes some values onto the stack
- pops all values and prints them

Depending on your programming language, this might look as follows:

```
class Stack[A]()  
  private elements := Nil[A]           the immutable linked list backing the stack, initially empty  
  
  fun push[A](x : A) : unit =  
    ...  
  fun pop[A]() : Option[A] =  
    ...  
  fun top[A]() : Option[A] =  
    ...  
  
test := new Stack[int]()  
test.push(4)  
test.push(5)  
test.push(2)  
while test.top().isDefined  
  print test.pop()                     prints 2,5,4
```

Problem 6.2 *Queues*

Points: 6

Implement the data structure of mutable queues. Make sure that all queue operations take constant time.

For example, this could be a class like in the first problem but backed by a doubly-linked list.

Write a test program that

- creates a new queue of integers
- enqueues some values in the queue
- dequeues all values and prints them

Problem 6.3 *Queues backed by Two Singly-Linked Lists*

Points: 6

Doubly-linked lists can be tricky to implement. As an alternative, give an implementation of *Queue*[A] that is backed by *two* immutable singly-linked lists.

This can be done in such a way that all queue operations take constant time with the exception that *dequeue* sometimes takes linear time.

Problem 6.4 *Buffers*

Points: 8

Implement a fixed-size circular buffer as follows: *Buffer*[*A*] consists of

- an *Array*[*A*] named *elements* of some fixed size *n*
- an integer $0 \leq \textit{begin} < n$ indicating the first valid entry in *elements*
- an integer $0 \leq \textit{size} \leq n$ indicating the number of valid entries in *elements*

Enqueueing writes to *elements*[(*begin* + *size*) mod *n*] and increments *begin* and *size* modulo *n*. Dequeueing reads from *elements*[*begin*], increments *begin*, and decrements *size* modulo *n*.

Because addition, incrementing, and decrementing are computed modulo *n*, the buffer becomes circular: when reaching index *n* (which is outside the buffer), we start from 0 again.

Your implementation should detect *buffer underflow* (trying to dequeue from an empty buffer) and *buffer overflow* (trying to enqueue into a full buffer) and raise appropriate exceptions.

Write a test program that

- creates a new buffer of integers
- enqueues some values in it
- dequeues all values and prints them