



## ASSIGNMENT 10

### GENERAL INFORMATION

This assignment is made public on Friday, April 11, 2003. The assignment is due on

Friday, April 25, 1 PM (after the Easter break).

Hand in your assignment to the teaching assistant running your lab session.

The first page of your (written) assignment has to contain at least the following information:

- the course name (Introduction to Programming - Concepts and Tools)
- your name and your student number
- name and student number of the fellow student if you submit in pairs
- assignment number

*Please staple your assignment!*

You will get back the graded assignment one week after submission deadline.

### WARM-UP AND SUGGESTIONS FOR FURTHER EXERCISES

As warm-up I suggest that you have a look at the review questions of Chapter 12 of the book (pp. 426–427 of my copy).

As to further exercises I suggest the following from Chapter 12 of the textbook (pp. 427–430 of my copy):

- Exercises 1–7 are all minor additions to the class `IntList`, each dealing with one method or enhancement.
- Exercise 8 is also about list of integers, but here you are asked to represent that list not as a linked list, but as an array of integers.
- Exercise 16 asks you to use class `Stack` (of which the code is provided in the book) to solve a simple problem, namely reversing the words in a sentence.
- The exercises 25–28 are interesting, but you may find them difficult. The questions deal with Polish notation, and how to use stacks to evaluate expressions in Polish notation. From the perspective of computer science Polish notation for expressions is interesting since you do not need parenthesis to uniquely determine the order of evaluation.

For this Chapter I find most exercises suitable for further practising, so just pick some.

It is up to you whether or not you want to work on these exercises. Those exercises will not be marked.

## QUESTIONS

### 1. (*Abstract Data Types*)

LEARNING OBJECTIVES: Thinking about abstract data types and finding useful applications. In class we have mentioned a number of different abstract data types, most notably

- (a) lists (here understood with variable length, where each cell can be accessed *directly*),
- (b) queues (with FIFO discipline, first in first out), and
- (c) stacks (with LIFO discipline, last in first out).

For each of those three ADTs you are supposed to find an example where you would use them, and explain why the specific characteristics suit that example/application best. Write about one longer paragraph for each.

Examples I have in mind are the following (the examples doesn't work well on purpose): To store the customer data of a car-rental company one should use an array of size 1000. Using an array of that size will ensure that there is always enough space to add new customers, and there is no car-rental company in the world that has more than 1000 customers. ... (This is, indeed, only a short paragraph.)

WHAT TO SUBMIT: Submit one paragraph for each part of the question.

### 2. (*Linked Lists*)

LEARNING OBJECTIVES: Working with a simple dynamic data type (here linked list), enhancing its features, reasoning about references.

In class we discussed the example of a linked list. The corresponding files

- `IntListElement.java`
- `IntList.java`

can be downloaded from the course homepage. You are supposed to add the following methods to class `IntList`:

- (a) `boolean moveForward()`  
This is similar to method `next()`, except that if we are already at the last item of the list then we stay at that, (whereas `next()` attempts to move forward and `current` is set to `null`). The method should return `true` if `current` actually points to a different element than before, i.e., if `current` got moved, and `false` otherwise.
- (b) `int howMany()`  
The method returns the number of elements in the linked list.
- (c) `void moveToTail()`  
The method moves `current` to the tail of the linked list, pointing to the last list element. If there is no list element then `current` is set to `null`.
- (d) `void append(IntList)`  
The method takes as argument another object of class `IntList`, and appends that list to the list that called the method.
- (e) `IntListElement find(int)`  
The method takes an integer element and searches through the list to find whether that integer value is stored in one of the list elements. If so, it returns a pointer to that particular element. If the integer number isn't found in one of the list elements then the method returns `null`.

Write a main program that will test you methods, and submit the code of your main program and of class `IntList`.

For each of these questions think carefully about how you walk through the list, and what you have to do with your pointers (references).

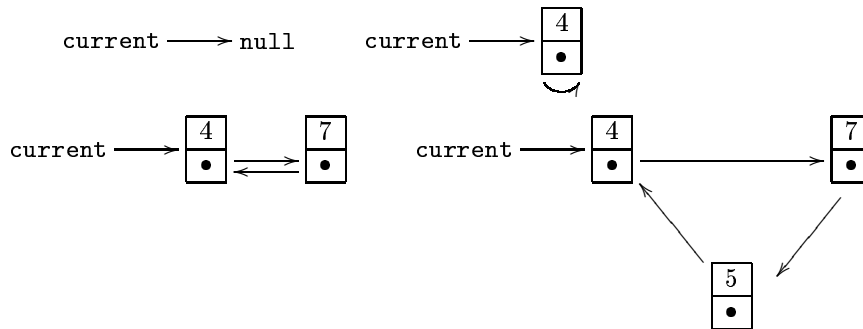
WHAT TO SUBMIT: Submit the code for the methods, plus the code of your main class where you test your implementation.

### 3. (Abstract Data Type Revolver)

LEARNING OBJECTIVES: Design and implementation of a dynamic data type.

*Below you should make a serious attempt on at least half of the questions.*

In this question we will implement another abstract data type, namely a circular structure allowing us to store integers. We will call this data type `IntRevolver`. The elements of such a data type are `IntListElements`, and the only difference to a linked list is that the pointer of the 'last' element in the list actually points to the head. As a consequence we don't really need the head explicitly, and it is enough to have `current`, pointing to the current cell of this circular structure. The pictures below show such structures with 0, 1, 2, and 3 elements, where the dot represents the cell holding the pointer, and `current` points to null respectively a cell containing 4.



For the following questions be very careful about the re-direction of pointers.

- (a) Create the class `IntRevolver`. Its only private attribute should be a pointer `current` of class `IntListElement`.

- (b) Add a method

```
int getCurrent()
```

that retrieves the information of the current cell. If the circular structure is empty then 0 is returned.

- (c) Add a method

```
void insert(int)
```

that allows to add a new item to the circular list. (Think carefully about the special situations when the list is empty, or when the list contains just one element. Draw diagrams to see what happens.)

- (d) Add a method

```
IntListElement remove()
```

that allows to remove the item 'after' `current` from the circular list. (Think carefully about the special situations when the list is empty, or when the list contains just one element. Draw diagrams to see what happens.)

- (e) Add a method

```
void next()
```

which will move `current` to the next item in the circular list.

- (f) Add a method

```
int count()
```

that counts the elements in the circular list. (Think about how you can ensure that you go around the circular structure just once.)

- (g) Add a method

```
IntListElement find(int)
```

that returns a pointer to a list element that contains the passed parameter of type `int`. If no cell contains that number then the `null` is returned. (Again think about how you ensure that you go around the circular structure just once.)

(h) Add a method

```
public String toString()
```

that will return a string holding the contents of all cells (one cell at a line). The first entry should be that of cell `current`.

(i) Think of at least one more method that might be a useful addition to this class, describe it, and implement it.

Test your class using a suitable main program, and submit the code of your program and of your class `IntRevolver`.

WHAT TO SUBMIT: Submit a printout of your implementation of the class `IntRevolver`, plus the code of the class that you use to test your dynamic data type.