
Bonus-Exercise

This is a bonus-exercise. Points earned in this exercise will benefit you for your final grade of the "Basispruefung". Maximal achievable bonus is 0.25 on top of your grade and it will be added **after** the correction of the exam, but before averaging with the "Informatik I" grade. (Meaning it is a real bonus).

Academic integrity

Only submit a solution you did yourself and you understand. To ensure this we reserve the right to invite people for an oral debriefing of the exercise. We select people both randomly and those that got suspiciously similar code to other people. (This will be checked by automated tools).

Asking questions

Before sending a mail with a question to this exercise check out the homepage Inf II Homepage. We might update the exercise in case it contains bugs and collect all frequently asked questions with their answers on the website. In case your question was not answered by the information given on the homepage please send your question to the email

`Informatik2_baug@lists.inf.ethz.ch`

Using the judge

We will use the same system, as we already used for the first five exercises. Those that successfully submitted a homework so far can ignore this section. If you are one of the few people that did not use the judge system so far follow the instructions below to create an account within the system:

- Open https://challenge.inf.ethz.ch/team/?cid=9 in a Web Browser and login with your student account.
- On the next site you will be asked for an “Enrollment Key“ - enter “bonus“ as the key.
- A detailed instruction on how to use the judge can be found here: Assignment 1

Please note that you can, as it was the case before, resubmit as often as you want. The judge will refuse submissions only after the due date 24th of May, 23:59:59.

1 Write a function that ... (12 pts)

In the first and hopefully easiest part of the bonus exercise you are asked to complete functions according to a specification.

1.1 ... computes the greatest common divisor of two integers

```
//pre: two positive non-zero integers n and m
//post: return the greatest common divisor of n and m
// i.e. the largest positive integer that divides n and m without a remainder
public static int gcd (int n, int m)
```

1.2 ... computes the median of three doubles

```
//pre: non-null array a of length 3
//post: return the median of {a[i]: 0 <= i < 3}
public static double median3(double a[])
```

1.3 ... returns the product of all integers between two integers

```
//pre: two integers a and b
//post: return the product of all integers in the generalized interval
// [a,b] = {integer x: a <= x <= b or b <= x <= a}
public static int prod(int a, int b)
```

1.4 ... computes the maximal length of all subsequences of ones

```
//pre: a sequence of numbers stored in an integer array a
//post: return length L of longest contiguous subsequence of ones
//       L = max {(b-a+1): 0 <= a <= b < len(a), a[i] = 1 for all a <= i <= b}
//       0 if the passed reference is null
//       0 if its empty
//example: the sequence 1 1 1 2 3 5 6 1 1 1 1 1 2 2 should evaluate to 5
public static int lensOfOnes (int [] a)
```

Files and Submission

Skeleton link for 1.1: http://lec.inf.ethz.ch/baug/informatik2/2016/ex/ex10/function/gcd/Main.java
Submission link for 1.1: https://challenge.inf.ethz.ch/team/websubmit.php?cid=9&problem=IB16BFGc

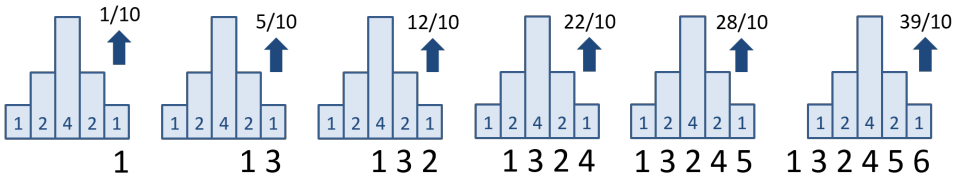
Skeleton link for 1.2: http://lec.inf.ethz.ch/baug/informatik2/2016/ex/ex10/function/median3/Main.java
Submission link for 1.2: https://challenge.inf.ethz.ch/team/websubmit.php?cid=9&problem=IB16FMed

Skeleton link for 1.3: http://lec.inf.ethz.ch/baug/informatik2/2016/ex/ex10/function/prod/Main.java
Submission link for 1.3: https://challenge.inf.ethz.ch/team/websubmit.php?cid=9&problem=IB16FPro

Skeleton link for 1.4: http://lec.inf.ethz.ch/baug/informatik2/2016/ex/ex10/function/ones/Main.java
Submission link for 1.4: https://challenge.inf.ethz.ch/team/websubmit.php?cid=9&problem=IB16FOne

2 Weighted Mean (11 pts)

Goal of this part of the exercise is to write an object that can be used in order to compute a weighted mean on a stream of numbers.[1] Before we formally describe the exercise, consider the following scenario in order to understand the task. Assume you get a sequence of numbers, say `1 3 2 4 5 6 . . .` Now, already while the numbers are provided you want to compute a weighted sum by multiplying the most recent available numbers with the values of a so called *kernel* providing the weights for the sum, here: `1,2,4,2,1`



In order to normalize the weighted sum, you have to divide by the sum of all values of the kernel. In the example above, the kernel values add up to $1+2+4+2+1=10$, so the results from the object are computed as follows

$$\frac{1\cdot 1}{10}=0.1 \qquad \frac{2\cdot 1+1\cdot 3}{10}=0.5 \qquad \frac{4\cdot 1+2\cdot 3+1\cdot 2}{10}=1.2 \qquad \ldots \qquad \frac{1\cdot 3+2\cdot 2+4\cdot 4+2\cdot 5+1\cdot 6}{10}=3.9$$

Note that in the beginning we do not have enough data for the whole width of the kernel, which is why we only use a part of the kernel or – here equivalently – fill up the unavailable positions with zeroes. Later, we have more data than the kernel width and therefore forget about old data: having received 6 values, only 5 values are taken into account in the example above (right most figure).

It might help to have a concrete example in mind in order to understand this: Assume you measure the rainfall in the mountains. Now you want to model the amount of water that flows down the river at the bottom of the mountains: first you get the water from the small hills close by (little), then you get the water from the main part (a lot) and eventually the rest for the parts far away. Feeding the rainfall measures int our weighted sum object might result into a reasonable model for river flow.

In the following, we describe phenomenologically and with example code what the object is supposed to do. Your task is to implement the corresponding class such that the object's behavior is as expected.

Task Description

Provide a class `WeightedMean` with the following properties:

- A **WeightedMean** `f` can be instantiated with an array of doubles (a "kernel") in the following way:

```
double[] a = {1,2,4,2,1};
// a: non-empty array of positive non-zero values
WeightedMean w = new WeightedMean(a);
```

- `WeightedMean` provides an **input method** `Put` that accepts integer numbers and has no return value

```
int value;
w.Put(value);
```

[1] Such objects are used a lot in digital signal or image processing for filtering streams of data. The mathematical term for this is a *convolution*.

- `WeightedMean` provides an **output method** `Get` that returns the current weighted sum as a double. Assuming that the weighted mean `w` was initialized with kernel `a` of length `k`, and assuming we have provided ("Put") data $d_0, \ldots, d_{n-1}$, the object will return the following value

$$\frac{\sum_{i=0}^{m-1} d_{n-1-i} \cdot a_i}{\sum_{i=0}^{n-1} a_i}$$

  where $m := min(k, n)$.

- Example:

```
double[] kernel = [1,4,1];    // kernel of width 3
WeightedMean w = new WeightedMean(kernel);
w.Put(1); // now w.Get() would return 1*1/6=0.167
w.Put(2); // now w.Get() would return (1*2+4*1)/6 = 1
w.Put(3); // now w.Get() would return (1*3+4*2+1*1)/6 = 2.0
w.Put(4); // now w.Get() would return (1*4+4*3+1*2)/6 = 3.0
w.Put(8); // now w.Get() would return (1*8+4*4+1*3)/6 = 4.5
...
```

To test your implementation, enter the kernel size followed by the kernel weights followed by list of numbers followed by `end`. For example, input (kernel size, kernel data, data) `3 1.0 4.0 1.0 1 2 3 4 1 end` should lead to this output

```
0.16666666666666666
1.0
2.0
3.0
3.3333333333333335
```

Files and Submission

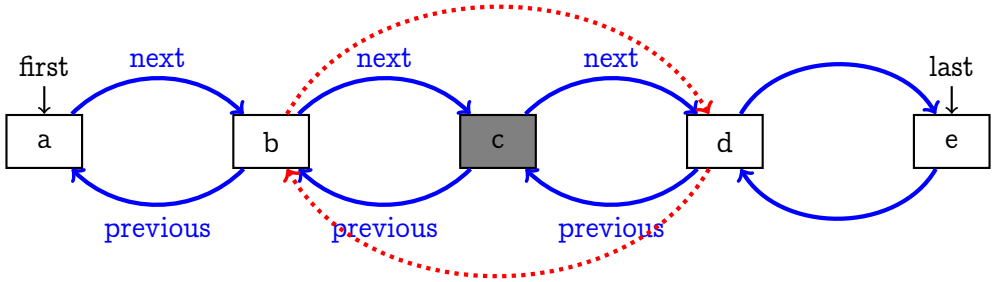
Skeleton link for 2: `http://lec.inf.ethz.ch/baug/informatik2/2016/ex/ex10/objects/Main.java`
Submission link for 2: `https://challenge.inf.ethz.ch/team/websubmit.php?cid=9&problem=IB16BObj`

3 Remove an Element from a Double Linked List. (11 pts)

In this exercise your task is to remove an element from a double linked list. A double linked list is a list where each element has a reference to the next element and the previous element of the list.

```
class Node{
    private int data;          // holds the value of the node
    private Node next;         // references to next
    private Node previous;     // and previous node
    ...
}
```

Double linked lists are known for particular efficient removal and insertion of elements. In the figure below, the marked element is removed by resetting the next pointer of its previous element and the previous pointer of its next element accordingly.



Task

Implement the method `remove(Node n)` of the class `List`.

Hint: Before implementing think of invariants and special cases.

To test your implementation use `add` to build up a list, `out` for output and `remove` to remove a list element. For example, input

```
add 1 2 3 4 5
out
remove 3
out
```

should lead to this output:

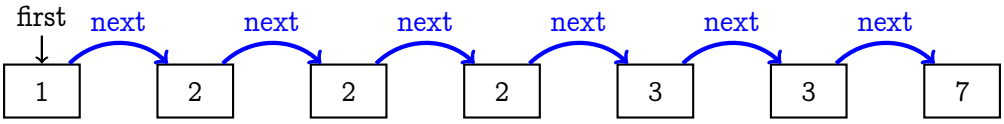
```
F-> 1 <-> 2 <-> 3 <-> 4 <-> 5 <-L
F-> 1 <-> 2 <-> 4 <-> 5 <-L
```

Files and Submission

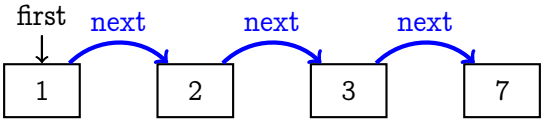
Skeleton link for 3: `http://lec.inf.ethz.ch/baug/informatik2/2016/ex/ex10/doubly/Main.java`
Submission link for 3: `https://challenge.inf.ethz.ch/team/websubmit.php?cid=9&problem=IB16BDLL`

4 Remove Duplicates (11 pts)

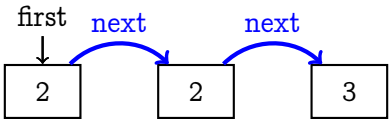
Your task in this exercise is to remove duplicates from a sorted list. An intial list might look like this:



After removal of the duplicates, the list look should look like this:



and the removal operation itself should return a list that contains the duplicates, keeping further duplication, if any.



Task

Implement the method `removeDuplicates` of the class `List`.

The following test input

```
add 1 2 8 3 2 3 2 4 4 2 2 2 2 3
out
duplicates
out
```

should lead to this output:

```
1 -> 2 -> 2 -> 2 -> 2 -> 2 -> 2 -> 2 -> 3 -> 3 -> 3 -> 4 -> 4 -> 8 ->
2 -> 2 -> 2 -> 2 -> 2 -> 2 -> 3 -> 3 -> 4 ->
1 -> 2 -> 3 -> 4 -> 8 ->
```

Files and Submission

Skeleton link for 4: `http://lec.inf.ethz.ch/baug/informatik2/2016/ex/ex10/sorted/Main.java`
Submission link for 4: `https://challenge.inf.ethz.ch/team/websubmit.php?cid=9&problem=IB16BLis`

5 Words (11 pts)

This exercise is about a hypothetical game where the players are asked to take words out of a sentence. The following rule makes the game interesting: the characters of a word have to be available in the sentence *in the correct order but not necessarily in a contiguous way*. If a word is contained in the sentence, its characters are removed one by one from the sentence in the order they appear in the text. Each character of a word is removed once. The objective of the game is simple: the first player who cannot build a word from the remaining characters any more, looses the game. Note that the human players are not allowed to check for remaining characters. They have to memorize this.

Consider the following **example**:
Start sentence: "viel erfolg bei dieser uebung."
Example word: "vers"
Remaining characters: "il efolg bei dieer uebung." (A player does not see this)
Example word: "leber"
Remaining characters: "i folg i diee uebung."
...

Implement in this exercise a class Text with the following interface:

```
class Text {
    // Initial text to start with
    Text(String t);

    // pre: non-null string s
    // post: return if text contains the characters of the string
    // characters have to be found in the right order but not
    // necessarily contiguously
    // in case the string is found, the corresponding characters
    // have been removed one by one from the text.
    boolean HasString(String s);

}
```

In order to keep it simple, we only feed lowercase characters, whitespaces and punctuation marks.

To test your implementation, enter a sentence followed by `<enter>` and a sequence of words with command `word`. For example, input

```
maus und elefant.
word mund
word aus
word elf
```

should lead to this output:

```
contained
not contained
contained
```

Files and Submission

Skeleton link for 5: `http://lec.inf.ethz.ch/baug/informatik2/2016/ex/ex10/words/Main.java`
Submission link for 5: `https://challenge.inf.ethz.ch/team/websubmit.php?cid=9&problem=IB16BWor`

6 Stack-based Calculator (11 pts)

This task is about implementation of a calculator on a stack. Basically the idea is that values and operations can be deposited on a stack for evaluation. However, focus of this exercise is not on the stack itself. It only serves as the storage container to keep intermediate results. Focus of this exercise is on the implementation of the operations (add, multiply, minus) on the stack, for which OOP techniques will be used.

The idea is to complement the already existing implementation such that the following operations would be, for example, working:

```
Op left = new Number(10);
Op right = new Number(20);
Op op = new Adder(left, right);
Op op = new Minus(op);
System.out.println(op.ToString());  // output is -(10+20)
System.out.println(op.Evaluate());  // output is -30
```

The provided source code file contains classes `Number`, `Adder`, `Multiplier`, `Minus`, `Stack` and `Main`. The content of all classes except the `Minus` class is already finished.

Your tasks:

1. Complement the code such that it *compiles* by only using the mechanism of inheritance. The main function should now behave as expected when you use commands `value`, `add`, `mult` and `out`.
2. Complement the code such that the functionality of `minus` also works: provided that `o` is an Op item, a `Minus` item created as

   ```
   Op m = Minus(o);
   ```

   should evaluate to the negative of `o`, i.e. it should hold that `m.evaluate()==-o.evaluate()`. Moreover, the string created by `m.ToString()` should prefix the string of `o.ToString()` with a single Minus-sign (and no further whitespaces).

To test your implementation, use commands like `value` (followed by an arbitrary number of integers), `add`, `mult`, `out` or `minus`. For example:

```
value 20 30 add value 30 40 add
mult
out
```

should lead to this output:

```
((20 + 30) * (30 + 40)) = 3500
```

Hint: In order to accomplish this task and understand the mechanism, first follow task (1) and make sure the code simply compiles. Then play with the calculator starting from some provided examples (in the code) to learn how it works and what it does. Only then implement the `Minus` object accordingly.

Files and Submission

Skeleton link for 6: `http://lec.inf.ethz.ch/baug/informatik2/2016/ex/ex10/oop/Main.java`
Submission link for 6: `https://challenge.inf.ethz.ch/team/websubmit.php?cid=9&problem=IB16BOop`

7 Longest Path (Recursion) (11 pts)

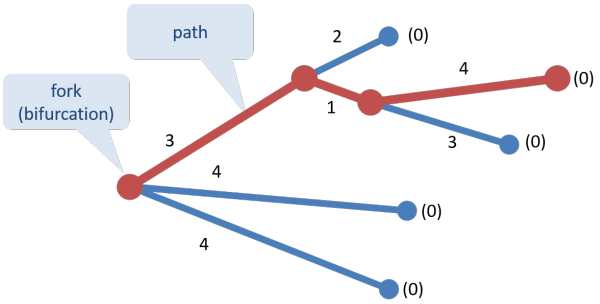
This exercise is about (recursively) computing the length of the longest path reachable from the root of a tree. We consider the following setup: We start at some fork. A fork can be either the end of a path (i.e. a bifurcation without outgoing paths) or the beginning of one ore more paths that each lead to the next fork. We assume that each path ends somewhere and paths do not join, i.e. that the underlying structure is in fact a tree.

We describe a set of paths by a sequence of integers, starting with a fork. A fork (and, recursively, the set of forks reachable from it) is specified as follows:

1. An integer n that provides the number of outgoing paths. $n = 0$ means, there is no outgoing path, i.e. the fork provides the end of a path.
2. If $n > 0$ then the sequence provides n times the following:
   (a) A number describing the length of the outgoing path.
   (b) Followed by the specifcation of the next fork on that path

Example: The paths in the figure to the right are described with input sequence 3 3 2 2 0 1 2 4 0 3 0 4 0 4 0, or layouted more intuitively:

3
3 *2*
2 *0*
1 *2*
4 *0*
3 *0*
4 *0*
4 *0*



The longest path in this example has length 8. The corresponding path lenghts are marked **boldface red**. The *blue italics* numbers correspond to the number of outgoing paths for each fork.

Implement the following function that reads the decription of the paths starting at a fork and returns the length of the longest path.

```
// pre: integers at scanner describing a fork
// post: return longest path described by input
public static int longestPath(java.util.Scanner scanner){
```

Use `scanner.nextInt()` in order to retreive the next integer. You can assume that inputs are well-formed and thus a correct program would terminate correctly with our inputs.

Hint: This exercise is not about trees: you do not have to build up a tree in memory. Use recursion instead. Without recursion, this task is much more difficult to solve.

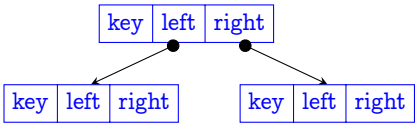
Files and Submission

Skeleton link for 7: `http://lec.inf.ethz.ch/baug/informatik2/2016/ex/ex10/paths/Main.java`
Submission link for 7: `https://challenge.inf.ethz.ch/team/websubmit.php?cid=9&problem=IB16BRec`

8 Trees (11 pts)

In a tree nodes have more than one successor. Recall that a *binary search tree* is a tree of order two with $key_{x.left} < key_x < key_{x.right}$ for each node x

```
public class Node {
    int key;
    SearchNode left;
    SearchNode right;
    ...
}
```



Your task in this exercise is to count occurences of nodes that provide a key in a given interval:

```
// non-balancing binary search tree
class Tree{
    // insert k sorted into the tree
    public Node Insert (int k){...}

    // pre: binary search tree
    // post: return number of elements n in the tree
    //       with from <= n.key <= to
    public int Elements(int from, int to){

        IMPLEMENT THIS

    }

    // print the tree
    public void print(){...}
}
```

Files and Submission

Skeleton link for 8: http://lec.inf.ethz.ch/baug/informatik2/2016/ex/ex10/trees/Main.java
Submission link for 8: https://challenge.inf.ethz.ch/team/websubmit.php?cid=9&problem=IB16BTre

9 Online Median with Heaps (11 pts)

Your task in this exercise is to implement the online median as discussed in the lectures.

In the template we provide a fully working implementation of a Min-/Max-Heap. Moreover, we provide a template for the online median class that has the following interface:

```
class OnlineMedian {

    // constructor
    OnlineMedian() {...}

    // post: insert a new value to this online median
    //       such that the computation of the median
    //       can be performed in O(1);
    public void Insert(double value){
      // implement this
     }

    // pre:  at least one value has been entered into this
    //       data structure via method Insert
    // post: Return the median of all values that have been
    //       entered in this data structure via method Insert.
    public double Get()
    {
      // implement this
    }
}
```

Implement functions `Insert` and `Get`. Everything else is already implemented. Do not be surprised if your solution for this method is pretty short.

To test your implementation, you can add numbers to your online median using `add`, the median is output via `out`:

```
add 1 8 2 4 7 6 6 3 2 9 10
out
add 10 10 10
out
add 10
out
```

should lead to this output:

```
6.0
6.5
7.0
```

Files and Submission

Skeleton link for 9: `http://lec.inf.ethz.ch/baug/informatik2/2016/ex/ex10/median/Main.java`
Submission link for 9: `https://challenge.inf.ethz.ch/team/websubmit.php?cid=9&problem=IB16BMed`