Prof. Dr.-Ing. Ralf Schenkel
Tobias Zeimetz
Trier University

Wintersemester 22/23

Exercises for the Class

# Elements of Computer Science: Programming

Assignment 05

Submission of solutions until 3:30 p.m. at 05.12.2022
at `moodle.uni-trier.de`

- Every task needs to be edited in a meaningful way in order to get a point!
- Please comment your solutions, so that we can easy understand your ideas!
- If you have questions about programming or the homeworks, just ask you teachers!
- **Submission that can't be compiled are rated with 0 points!**

## Exercise 1 (Evaluation: Numbers)

The task is based on the scavanger hunt exercises: Write a program to solve the following modified version of the scavenger hunt.

- As with previous tasks, there is a size specification `size` for the field first, followed by the entries for the field itself.
- However, there is now another number as input, which represents the start position for the movement through the array (instead of the fixed value $0$).
- Test (up to) $15$ values including the start position contained in the array (i.e. up to $14$ times the setting `index = array[index]`).
- If a value is found in `array[index]` which cannot be used as an index for the given field (because it is at least as big as `size` or $< 0$, i.e. negative), the scavenger hunt should end immediately, even if no $14$ steps have been taken yet. The value found is then the "treasure" found during the scavenger hunt (if $\geq$`size`) or a "blank" (if $< 0$).
- The output should be the positions of the (up to) $15$ visited fields during the scavenger hunt and also the value of the treasure or the blank (or an additional $0$ if no end was found on the $15$ positions and therefore the search was aborted).
- In addition to the number values, you can also output text such as “treasure”, “blank” or “abort”, but only the numbers are evaluated.

Examples:

1. Input: `9 0 2 1 4 5 7 -10 10 6 3`
   (size is 9, startposition is at position 3, treasure (value 10) at position 7, blank at (value -10) at position 6)

   Output: `3 4 5 7 Treasure 10`

2. Input: `9 0 2 1 4 5 7 -10 10 6 1`
   (like the first example, but startposition is 1)

   Output: `1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 Abort 0`

3. Input: `9 0 2 1 4 5 7 -10 10 6 7`
   (like the first example, but startposition is 7)

   Output: `7 Treasure 10`

4. Input: `9 0 2 1 4 5 7 -10 10 6 8`
   (like the first example, but startposition is 8)

   Output: `8 6 Blank -10`

5. Input: `16 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 99 0`
   (Treasure (value 99) at position 15, but startposition at 0)

   Output: `0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 Abort 0`

6. Input: `16 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 99 1`
   (like the previous example, but startposition is 1)

   Output: `1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 Treasure 99`

You will need a loop again when programming. However, there are several reasons why the loop is exited (treasure, rivet abort). Therefore it makes sense to use a loop with a Boolean variable as condition or to use **`break`**.

## Exercise 2 (Evaluation: Text)

In the above scavenger hunt, there were two different reasons to end the treasure hunt with `Abort`: (a) because the number of steps with 15 was simply too small (example 5 above) or because you are caught in a loop and therefore a treasure or a blank can never be found (see example 2).

There are several ways to recognize these loops. The basic idea is to somehow determine when positions in the field are reached repeatedly. This recognition can be implemented in different ways: (a) In a second field you store which positions have already been reached or (b) you think about how long the scavenger hunt can last if you are not caught in a loop. (b) is much easier to implement.

Now modify your solution to the previous task (i.e. with additional start value) so that the output is limited to the following three texts(!): `Treasure`, `Blank` or `InfinityLoop`. Positions, values or a prompt should no longer be displayed.

1. Input: `9 0 2 1 4 5 7 -1 10 6 3`

   Output: `Treasure`
2. Input: `9 0 2 1 4 5 7 -1 10 6 1`

   Output: `InfinityLoop`
3. Input: `9 0 2 1 4 5 7 -1 10 6 7` (like the above exercise, but with startposition 7)

   Output: `Treasure`
4. Input: `9 0 2 1 4 5 7 -1 10 6 8` (like the above exercise, but with startposition 8)

   Output: `Blank`
5. Input: `16 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 99 0`

   Output: `Treasure`

**Exercise 3 (Evaluation: Text)**

Write a Java program that reads a `int` number $n > 1$ and determines whether it is a prime number:

Test in a `for` loop for all values $k$ between 2 and $n-1$ whether $n$ can be divided by $k$ without remainder. If there is such a $k$, then $n$ is not a prime number and the program should output the smallest such divider $k$. Otherwise (there is no such $k$), `This number is a prime number` should be output, e.g. as follows:

```
77
This number has 7 as the smallest divisor
```

or

```
79
This number is a prime number
```

For the evaluation, the output must correspond exactly to the texts cited in these examples. If $n < 2$ is entered, the program may produce any output.

**Exercise 4 (Evaluation: Numbers/Text)**

Write (e.g. based on the previous assignment) a Java program that reads a `int` number $m > 1$ and determines how many prime numbers there are between 2 and $m$ (both inclusive). (If $m = 1$ is entered, the program may again produce any output; the input and reading of $m$ has already been performed in the specified program).

```
79
The number of prime numbers up to this input is 22
fast enough
```

**Watch out:** For this task, the computing time required for the evaluation is also measured. A fast solution requires less than one second per case for all evaluation examples. Slow solutions take much longer. The faster your program is, the more cases can be successfully evaluated within one second. For example, consider whether the prime number test for a given $n \leq m$ really needs to test all numbers between 2 and $n - 1$.

For slow but otherwise correct solutions there is a maximum of 10 points, the remaining 5 points are awarded on the basis of speed. The time measurement incl. the output **fast enough** resp. **too slow** is given in the program and must not be modified.

**Advise:** A nested loop solves the problem, but will unfortunately be too slow to get 15 points. Look at the *Sieve of Eratosthenes* for a quick solution.