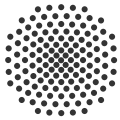

University of Stuttgart

Programming Paradigms Prof. Dr. Michael Pradel

University of Stuttgart, Summer 2022

Exercise 3: Control Flow

(Deadline for uploading solutions: May 25, 2022, 11:59pm Stuttgart time)

The materials provided for this homework are:

- a pdf file with the text of the homework (this document);
- a zip file with **the templates file and folder structure you must use for the submission**.

```
exercise3.zip
├── task_1
│   └── task_1.csv (Substitute this file with your solution)
├── task_2
│   ├── src
│   │   ├── main/scala
│   │   │   └── Main.scala (Substitute this file with your solution)
│   │   └── test/scala
│   │       └── MySuite.scala
│   └── .build.sbt
└── task_3
    ├── Java
    │   ├── src/iterator
    │   │   ├── Course.java
    │   │   ├── University.java (Substitute this file with your solution)
    │   │   └── IteratorTest.java
    │   ├── .project
    │   └── .classpath
    └── Python
        ├── course.py
        ├── university.py (Substitute this file with your solution)
        └── test_iterator.py
```

Figure 1: Folder structure to use for submission.

The submission must be compressed in a zip file (not rar, 7z, gz, or anything else) using exactly the folder structure and names in Figure 1. To ensure that your submission has the expected structure, check it with the validator.jar tool provided on the course page, using the following command (where exercise3.zip is the original file provided to you, and your_exercise3.zip is your modified version of it):

java -jar validator.jar exercise3.zip your_exercise3.zip

Note: Submissions that do not comply with the exact folder structure and names in Figure 1, or that do not work with the indicated versions of Java, Scala, and Python, cannot be graded.

1 Task 1 (25% of total points of the exercise)

For this task you have to manually evaluate the given expressions under different sets of rules for precedence and associativity. Figures 2a and2b define two sets of rules.

Operator	Rules 1		Type	
	Assoc.	Prec.	Operands	Result
`**`	R	2	numeric	numeric
`*`	L	3	numeric	numeric
`/`	L	3	numeric	numeric
`+`	L	4	numeric	numeric
`-`	L	4	numeric	numeric
`>>`	L	5	numeric	numeric
`<<`	L	5	numeric	numeric
`>`	L	6	numeric	boolean
`<`	L	6	numeric	boolean
`!=`	L	7	numeric	boolean
`&&`	L	8	boolean	boolean
`||`	L	8	boolean	boolean

(a) Ruleset 1.

Operator	Rules 2		Type	
	Assoc.	Prec.	Operands	Result
`**`	L	2	numeric	numeric
`+`	L	2	numeric	numeric
`-`	R	3	numeric	numeric
`*`	L	3	numeric	numeric
`!=`	L	4	numeric	boolean
`&&`	L	5	boolean	boolean
`/`	L	5	numeric	numeric
`>>`	R	5	numeric	numeric
`<<`	R	5	numeric	numeric
`>`	R	6	numeric	boolean
`<`	R	6	numeric	boolean
`||`	L	8	boolean	boolean

(b) Ruleset 2.

Figure 2: Sets of rules for precedence and associativity. A lower number in the "Prec." column means they should be applied first. The "Assoc." column indicates whether the operator is left-associative ("L") or right-associative ("R").

Task: Using the rules in Figure 2a, evaluate the following expressions:

1. `3 > 40 / 2 && 256 != 2 ** 8`
2. `10 << 1 != 5 || 8 > 2 ** 3`
3. `3 > 4 * 2 / 4 + 18 && 3 > 2 + 2`

Using the rules in Figures 2b, evaluate the following expressions:

4. `971 + 56 - 3 / 4 + 2 + 2 ** 3`
5. `true || 45 / 3 * 4 + 1 > 18`
6. `2 << 3 + 1 << 3 < 625 / 5 * 5`

To submit your answer, please fill in the file *exercise3/task_1/task_1.csv*. Each line in the file correspond to one expression, i.e., there should be six lines in total. In each line, enter only the result of evaluating the expression. The meaning of individual operators is the same as in Java. Additionally, '`**`' represents the exponential (power) operation (e.g., `2 ** 3` is `8`). To indicate the result of evaluating an expression, use the syntax specified by Java literals (e.g., the number five is `5` and the Boolean values are `true` and `false`). Furthermore, all literals are integer and as such consider integer division. Note that the result can either be an integer or a Boolean. Assume to work with 64-bit integers.
Tip: If you want to check what is the meaning of an individual operation in Java you can just run it here: `https://www.w3schools.com/java/tryjava.asp?filename=demo_compiler`.
Evaluation Criteria: Your solution will be compared against the correct result of evaluating each expression under each set of rules.

2 Task 2 (45% of total points of the exercise)

For this task you have to implement three tail-recursive functions in Scala:

1. `arraySum`, which computes the sum of an array of integers.
2. `isPalindrome`, which checks whether the string is a palindrome.
3. `arraySort`, which sorts a list of integers in ascending order.

For this task you must use Scala version 3.1.2, which you can download from here (`https://www.scala-lang.org/download/`). You must implement the functions in such a way that they are tail-recursive, as described in the lecture. As a brief recap, a function is tail-recursive when it calls itself as its last action so that its stack frame can be used for the next call.
Writing a tail-recursive function helps the compiler performing some optimizations, and Scala has a special annotation to tell the compiler that a function should be treated as a tail-recursive function. This feature helps you detect errors early, since a function that is annotated as tail-recursive but but cannot be recognized as such by the compiler will raise a compile-time error.

2.1 Starting Point

In the provided zip file you will find the template Scala project to use for this task in the */task_2* folder. The 'src' folder contains the only source files you must use for this task, together with 'build.sbt' files to help you configure the project with the correct Scala version. Your solution must be entirely implemented following this template, using a single source file (`Main.scala`) and no other files. Please keep the names of the three functions called in the test cases unchanged. You are free to use the IDE of your choice; be sure to configure the correct Scala version used for evaluation. We have tested it with VSCode and the Metals extension.

Examples of correct input-output pairs for the three functions:

Input	Output
`arraySum(Array(5, 10, 15), 0)`	`30`
`isPalindrome("helloolleh")`	`true`
`isPalindrome("hellohello")`	`false`
`arraySort(ListBuffer(4, 3, 5, 2, 1), ListBuffer())`	`ListBuffer(1,2,3,4,5)`

Note that the annotation `@tailrec` must be kept in the same position also in the submitted solution.

Your task is to implement the function body of the three functions, as indicated by the IMPLEMENT HERE comments in `Main.scala`. You are allowed to use only Scala standard libraries.

2.2 Testing

You find some MUnit tests in file `MySuite.scala`. Use them to check whether your functions work as expected. We will use additional tests to evaluate your solution, and you are advised to also add further tests for your own testing.

2.3 Evaluation Criteria

Your code will be evaluated against a held-out set of test cases that exercise your code.

3 Task 3 (30% of total points of the exercise)

This task is about different kinds of iterators available in popular languages. You are given two implementations of a university setting, in Java and Python. Your task is to implement an iterator in each language that returns the list of all the marks ever given in a university. A `University` object refers to `Course` objects, which store for each student id the corresponding mark as an integer. The order of iteration is up to you.

3.1 Starting Point

1. For Java, please extend the `University` class to enable iterating over the marks of the various courses by implementing the `Iterable<Integer>` interface.
2. For Python, please extend the given skeleton of a "true" iterator, i.e., the `iterate_marks()` method of the `University` class.

Note: The implementation of the iterator must be yours, i.e., you are not allowed to call into any third-party library.
For Java, you can import the Eclipse project provided under */task_3/Java*. For Python, you can find the Python code under */task_3/Python*.

3.2 Testing

To execute the test cases you have to implement the iterator first. The Python test case requires *pytest*, which you can install, e.g., via *pip install pytest*

3.3 Evaluation Criteria

Your code will be executed with different social graphs, and we will check whether the iterators enumerate the correct set of people. During the evaluation, we will use Java 8 and Python 3.7.