Assignment 3

CSCI 4020U

1 About the assignment

In this assignment, you are to implement a simple, but complete programming language with features:

- Integer and string data
- Assignment and dereference of variables
- Arithmetics of integers
- Concatenation of strings and integers
- Iteration over integer ranges
- Function declaration
- Function invocation
- Recursion

2 Sample programs

program 1

```
x = "Hello";
y = "World";

print(x ++ " " ++ y);
```

Expected output:

```
Hello World
```

program 2

```
x = "woof ";
y = "Dog goes " ++ (x * 2);

print(y);
```

Expected output:

```
Dogs goes woof woof
```

program 3

```
sum = 0
for(i in 10..20) {
  sum = sum + i;
}

print(sum)
```

This iterates over the range 10, 11, … 20. Note, the end 20 is included in the range.
Expected output:

```
165
```

program 4

```
function greeting(name, message) {
  x = "Hi,";
  x = x ++ " my name is " ++ name ++ ".";
  print(x);
  print(message);
}

greeting("Albert", "How are you?");
```

Expected output:

```
Hi, my name is Albert.
How are you?
```

program 5

```
function factorial(n) {
  if(n < 2) {
    1;
  } else {
    n * factorial(n-1);
  }
}

print(factorial(10));
```

Expected output:

```
3628800
```

3 Your tasks

You are given the following files:

```
worksheet.ipynb

./src:
- backend
- PL.g4

./src/backend:
- data.kt
- expr.kt
- runtime.kt
```

3.1 Parsing

You need to complete the grammar file `src/PL.g4` so that it can parse all the programs.

You are provided a template:

```
grammar PL;

@header {
import backend.*;
}

@members {
}

program returns [Expr expr]
    : { $expr = new NoneExpr(); };


WHITESPACE : [ \t\r\n] -> skip;
```

The `program` symbol is the start symbol, and it only returns an empty expression `NoneExpr` which always evaluates to `None` data.

3.2 Backend implementation in Kotlin

You are to complete the implementations:

1. `data.kt`: implement the data classes for

    - integer
    - string
    - boolean
    - function

   You are free to make your own choices.

2. `expr.kt`: implement the `Expr` subclasses that corresponds to the constructs of your programming language.

3. No changes to `runtime.kt` is required, but you are free to modify it if you prefer.

3.3 SDD

Connect the backend Kotlin implementation to your grammar using actions.

> ⚠ Be aware:
>
> SDD actions are expressed in *Java*, and thus you must conform to the Java programming language. For example, you need to end the lines with `;`.

4 Testing your code (iteratively)

We provided a workflow to help you iteratively test your implementation:

- `make build`
- Rerun the notebook `worksheet.ipynb`

4.1 Make

A `Makefile` with the default target `build` that helps you to compile:

- The kotlin backend classes in `data.kt`, `expr.kt` and `runtime.kt`.
- grammar file

- Java class files

4.2 Jupyter notebook

A Kotlin notebook `worksheet.ipynb` is provided to test your implementation.

⚠ Be aware:

1. You must `make` after any change to the `.g4` or `.kt` files.
2. You must restart and rerun the notebook after recompiling the files. This is because Kotlin kernel caches old compiled classes.