apl. Prof. Dr. Thomas Noll
Sebastian Junges, Christoph Matheja, Matthias Volk

Compiler Construction 2017
— Programming Exercise 6 —

Upload in L2P until July 15th before the exercise class.

Programming Exercise 1 (6 Points)

We now finish the implementation of our `i2Compiler` by generating the corresponding `Jasmin` code for our parsed program.

As stated on the Jasmin Webpage:
"*Jasmin is an assembler for the Java Virtual Machine. It takes ASCII descriptions of Java classes, written in a simple assembler-like syntax using the Java Virtual Machine instruction set. It converts them into binary Java class files, suitable for loading by a Java runtime system.*"

Our compiler generates code for the Jasmin language from the parsed *While* language. As a recap the grammar for the *While* language is given as follows:

start	→	program EOF
program	→	statement program | statement
statement	→	declaration SEM | assignment SEM | branch | loop | out SEM
declaration	→	INT ID
assignment	→	ID ASSIGN expr | ID ASSIGN READ LBRAC RBRAC
out	→	WRITE LBRAC expr RBRAC | WRITE LBRAC STRING RBRAC
branch	→	IF LBRAC guard RBRAC LCBRAC program RCBRAC | IF LBRAC guard RBRAC LCBRAC program RCBRAC ELSE LCBRAC program RCBRAC
loop	→	WHILE LBRAC guard RBRAC LCBRAC program RCBRAC
expr	→	NUMBER | ID | subexpr
subexpr	→	expr PLUS expr | expr MINUS expr | expr TIMES expr | expr DIV expr | expr MOD expr
guard	→	relation | subguard | NOT LBRAC guard RBRAC
subguard	→	guard AND guard | guard OR guard
relation	→	expr LT expr | expr LEQ expr | expr EQ expr | expr NEQ expr | expr GEQ expr | expr GT expr

Implement `generator.JasminGenerator.translateProgram(ASTNode)` which given the *program* node in an abstract syntax tree returns the generated Jasmin Code as a string.

Hint: It is a good approach to write methods for every language construct and call them recursively (similar to a recursive descent parser). Once you get the idea, it is actually less effort than you might think!

Methods for generating `Jasmin` code for the main class, for writing to and reading from the console are already provided. You should also implement the method `translateExpr(ASTNode)` which translates an *expression* into `Jasmin` code and is used in the method for writing to the console.

Here are some helpful resources for the `Jasmin` language:

- Jasmin main page: http://jasmin.sourceforge.net/
- Jasmin user guide: http://jasmin.sourceforge.net/guide.html
- List of machine instructions for Jasmin: http://jasmin.sourceforge.net/instructions.html

- Explanation of the machine instructions: https://docs.oracle.com/javase/specs/jvms/se8/html/jvms-6.html

To test your implementation you can write code in the *WHILE* language and run it through our compiler with:

```
$java -cp bin Main tests/gcd.txt --out gcd.j
```

The generated code is written to the given filename (in this example `gcd.j`). Next you can use `Jasmin` to build an executable Java class file:

```
$java -jar lib/jasmin.jar gcd.j
```

You can execute the generated class file as a Java program and observe its behavior:

```
$java gcd
42
27
GCD:
3
```

After finishing this exercise you now have your own simple compiler for Java code!