apl. Prof. Dr. Thomas Noll
Philipp Berger, Matthias Volk

Compiler Construction 2018/19
— Programming Exercise 5 —

Upload in L2P until December 17th before the exercise class.

General Remarks

- *Important*: You must not touch methods/functionality outside of the `checkDeclaredBeforeUsed` function. You may implement your own additional methods and classes, but not change any existing ones.
- Please make sure that as a default, no GUI or similar blocking interface is opened when using your implementation.
- Submit any files necessary and do not change file layouts if not strictly required for achieving the task at hand. **Never** make changes to generated files (i.e., visitors generated for a grammar by ANTLR, etc), always use subclassing or similar techniques to keep your functionality separate. Keep in mind that your code needs to be tested in an automated fashion!
- Make sure that your zip file contains the "i2Compiler" folder, with that folder containing at least the "src" folder with all necessary files for compiling your code.
  Example: "/i2Compiler/src/Main.java" should be a valid path in your archive.

Programming Exercise 1 (6 Points)

In this exercise we implement a semantic check. In our *WHILE* language we require that every variable identifier is *declared* before the variable is used (read or set). Additionally, a variable defined inside a scope like an `if` statement or a `while` loop is not visible outside this scope. We do not care whether a variable has been *initialised* before it is read. Examples:

This is valid:

```
int x; int y;
if (x <= y) {
    write("Hello world.");
}
write(x);
```

This is not valid (`y` is undefined and `z` is undefined outside the `if`-statement):

```
int x;
if (x <= y) {
    int z;
}
write(z);
```

Implement `checker.DeclarationChecker.checkDeclaredBeforeUsed()`.

Hint: for this you do not need to implement any attributed grammars and their evaluation. Instead simply walking the abstract syntax tree once and checking the required property suffices.

To check the type of a given non-terminal or token, you can use the methods in `util.WhileAlphabet`.

An example call of the program is:

```
$java -cp bin Main tests/valid.txt
...
Every variable was declared before use: true
```

You can also generate a visualization of the abstract syntax tree obtained by the parser. With the optional commandline argument `--dot output.dot` the abstract syntax tree is written as dot output to the given file. Then the following command generates a PDF depicting the tree:

```
$dot -Tpdf output.dot -o output.pdf
```