TCSS421A – Spring 2016
Programming project

Your homework in this course consists of programming assignments (the theoretical aspects are covered in the quizzes). Make sure to turn in the source files for the *whole* project completed so far when turning in each individual part: your project is incremental, and all previously completed parts must continue to work when the new parts are added. Also, include all test input files you use to test your compiler, and a short report briefly describing your solution for each part.

The project is worth 80 points, and is divided in four parts indicated below. There are 20 bonus points as indicated; these may be transferred to the midterm exam (up to 10 points or until the resulting grade reaches 30 points, whichever happens first) and to the final exam (up to 10 of the remaining points or until the resulting grade reaches 30 points, whichever happens first).

It is recommended to do Exercises 1.1 and 1.2 of the Textbook before starting the project itself, since they are meant to familiarize the student with the existing *j--* compiler that will be modified/extended in the project.

You must present your project in the last lecture (~5 min per group). *Your presentation slides must be uploaded to Canvas with the last part of your project*. Always identify your work. All members of a group must upload their own copy of the project material, clearly identified.

1. *(20+4 points, 4 points each, 2 weeks) Scanning: Textbook Exercises 2.10 – 2.15* ***– due April 11, 2016****:*

   Exercise 2.10. Modify Scanner in the *j--* compiler to scan (and ignore) Java multi-line comments.

   Exercise 2.11. Modify Scanner in the *j--* compiler to recognize and return all Java operators that are not reserved words.

   Exercise 2.12. Modify Scanner in the *j--* compiler to recognize and return all Java reserved words.

   Exercise 2.13. Modify Scanner in the *j--* compiler to recognize and return Java double-precision literals (returned as `DOUBLE_LITERAL`).

   Exercise 2.14. Modify Scanner in the *j--* compiler to recognize and return all other literals in Java, for example, `FLOAT_LITERAL`, `LONG_LITERAL`, etc.

   (Bonus) Exercise 2.15. Modify Scanner in the *j--* compiler to recognize and return all other representations of integers (hexadecimal, octal, etc.).

2. *(36+8 points, 4 points each, 3 weeks) Parsing: Textbook Exercises 3.21 – 3.30 and 3.34 –* ***due May 2, 2016****:*

Exercise 3.21. Modify the Parser to parse and return nodes for the `double` literal and the `float` literal.

Exercise 3.22. Modify the Parser to parse and return nodes for the `long` literal.

Exercise 3.23. Modify the Parser to parse and return nodes for all the additional operators that are defined in Java but not yet in *j--*.

Exercise 3.24. Modify the Parser to parse and return nodes for conditional expressions, for example, `(a > b) ? a : b`.

Exercise 3.25. Modify the Parser to parse and return nodes for the `for`-statement, including both the basic `for`-statement and the enhanced `for`-statement.

Exercise 3.26. Modify the Parser to parse and return nodes for the `switch`-statement.

Exercise 3.27. Modify the Parser to parse and return nodes for the `try-catch-finally` statement.

Exercise 3.28. Modify the Parser to parse and return nodes for the `throw`-statement.

Exercise 3.29. Modify the Parser to deal with a `throws`-clause in method declarations.

(Bonus) Exercise 3.30. Modify the Parser to deal with methods and constructors having variable arity, that is, a variable number of arguments.

(Bonus) Exercise 3.34. Say we wish to add a `do-until` statement to *j--*. For example,

```
do {
      x = x * x;
} until (x > 1000);
```

Write a grammar rule for defining the context-free syntax for a new `do-until` statement. Modify the Scanner to deal with any necessary new token(s). Modify the Parser to parse and return nodes for the `do-until` statement.

3. *(8 points, 4 points each, 1 week) Type checking (Textbook Exercises 4.1 through 4.2) –* ***due May 9, 2016****:*

   Exercise 4.1. The compiler does not enforce the Java rule that only one type declaration (that is, class declaration in *j--*) be declared public. Repair this in one of the `analyze()` methods.

   Exercise 4.2. The `analyze()` method in `JVariable` assigns the type `Type.ANY` to a `JVariable` (for example, `x`) in the AST that has not been declared in the symbol table. This prevents a cascading of multiple errors from the report of a single undeclared variable. But we could go further by declaring the variable (as `Type.ANY`) in the symbol table so as to suppress superfluous error reports in subsequent encounters of the variable. Make this improvement to analysis.

4. *(16+8 points, 4 points each, 3 weeks) Code generation (Textbook Exercises 5.6, 5.7, 5.11, 5.12, 5.15, and 5.21) –* ***due May 30, 2016****:*

   Exercise 5.6. Add the `do-while` statement to *j--*, adding it to your compiler and testing it thoroughly.

   Exercise 5.7. Add the classic `for`-statement to *j--*, adding it to your compiler and testing it thoroughly.

   Exercise 5.11. Add conditional expressions to *j--*, adding them to your compiler and testing them thoroughly. Conditional expressions are compiled in a manner identical to `if-else` statements. The only difference is that in this case, both the consequent and the alternative are expressions.

   Exercise 5.12. Add the conditional-or operator `||` to *j--*, adding it to your compiler and testing it thoroughly. Make sure to avoid unnecessary branches to branches. The conditional `||`, like the conditional `&&`, is short-circuited.

   (Bonus) Exercise 5.15. Add the `throw`-statement to *j--*, adding it to your compiler and testing it thoroughly. The `throw`-statement is straightforwardly compiled to JVM code using the `athrow` instruction;

   (Bonus) Exercise 5.21. Add the primitive type `long` to *j--*, adding it to your compiler and testing it thoroughly.