Compiler Project 2012: *Mini-Java compiler*

Implement a compiler for the Mini-Java programming language. The work has two parts: firstly, a front end making a full syntactic and semantic analysis of Mini-Java; secondly, a back end that generates low-level bytecode. The language analyzer must correctly recognize and process all valid (and invalid) Mini-Java programs. It should report syntactic errors, and then continue analyzing the rest of the source program. It must construct an AST and do a semantic analysis of this internal representation. If the given program was found free from errors, the back-end generates bytecode that can be subsequently executed. Note that you cannot use any automatic tools to generate bytecode: the task of this project is to *implement* the algorithms for such a tool.

Implementation requirements and grading criteria

You are expected to properly use and apply the compiler techniques taught and discussed in the course lecture materials and exercises. C# is used as the implementation language, and the low-level target language is CIL bytecode (Microsoft Common Intermediate Language). You must yourself make sure that your system can be run on the development tools available at the CS department.

The emphasis of the grading is the quality of the implementation: its overall architecture, clarity, and modularity. Pay attention to programming style and commenting. Grading of the assignment will consider (undocumented) bugs, level of completion, and its overall success (solves the problem correctly). The evaluation will particularly cover technical advice and techniques given by the course.

Documentation

Write a report on the assignment, as a document in PDF format. The title page of the document must show appropriate identifications: the name of the student, the name of the course, the name of the project, and the date and time of delivery.

Describe the overall architecture of your language processor with UML diagrams. Explain the diagrams. Clearly describe the testing process, and the design of test data. Tell about possible shortcomings of your program (if well documented and explained they may be partly forgiven). Give instructions how to build and run your compiler. The report must include the following parts

1. The Mini-Java token patterns as *regular expressions* or, alternatively, as *regular definitions*.
2. A *modified context-free grammar* that is more suitable for recursive-descent parsing; techniques used to resolve any remaining syntactic problems; these modifications must not affect the language that is accepted.
3. Specify *abstract syntax trees* (AST), i.e., the internal representation for Mini-Java programs; you can use UML diagrams or alternatively give a syntax-based definition of the abstract syntax.
4. *Error handling* strategies and solutions used in your Mini-Java implementation (in its scanner, parser, semantic analyzer, and code generator).

For completeness, include this original project definition and the Mini-Java specification as appendices of your document; you can refer to them when explaining your solutions.

Delivery of the work

The project must be stored in a zip form. This zip should include all relevant files, contained in a

directory that is named according to your unique department user name. The deliverable zip file must contain (at least) the following subfolders.

```
<username>
  ./doc
  ./src
```

Compress (zip) the whole folder and deliver its *address* to the project supervisor.