

Project: An LALR(1)-Parser Generator for Java

Problem Specification

The goal of this project is to develop an LALR(1)-parser generator for Java. Like JavaCC, the LALR(1)-parser generator has to provide features to specify the structure of lexical tokens (i.e., regular expressions) and grammar productions. Both the lexical tokens and grammar productions should also allow for the specification of semantic actions. In case of lexical tokens, for example, this means one should be able to convert a string representation of a number into an Integer value. Semantic actions in productions may occur anywhere of the right-hand side of a production. That is, semantic actions should be treated like ϵ -symbols. Finally, in the specification of both lexical tokens and grammar productions in should be possible to define some form of error-handling mechanisms. In particular, the generated parser should recover from a syntax error using some user-defined synchronization points.

The generated scanner/parser should be encoded as a Java class. There is not limit on the number of generated Java classes. However, in order to increase productivity, the LALR(1)-parser generator has to compile the generated Java classes while processing a grammar specification. That is, unlike JavaCC, which generates only Java files, the LALR(1)-parser generator has to produce Java class files, if the grammar specification does not contains any errors.

Application Requirements

The parser generator should be implemented in Java and JavaCC. You can use any Java implementation platform (e.g., Java SDK, J# .NET, eclipse). However, the grammar specification should be 100% Java-compliant.

Policies, Rules, and Constraints

You can use everything that has been shown in class or was part of an assignment. Every student has to work on his/her own project, but the discussion of ideas is allowed. Note, however, that your solution must be **original**. That is, it **must reflect your own ideas and work**.

You have to document your project. That is, you have to write a report that will contain:

- A requirement specification,
- A design specification,
- A documentation of the grammar specification file format,
- Discussion of special features of your application, and
- Open problems and possible future work.

What Is Important?

General Issues:

- Submit you project on time!
- Be ready to explain parts of the source code.
- Check for an appropriate structure of the code, libraries, and executables.
- Are there any extra features? Be ready to explain them.

Report Checklist:

- Are the basic requirements covered?
- What are the additional requirements that you have identified?
- What are the requirements that you have changed?
- Is the specification complete with respect to the required functionality?
- Is the presentation well structured?
- Have you included prototypes, screen shots, UML diagrams, flow charts, sequence diagrams, etc.?
- Have you highlighted the special features of your application?
- Have you explained the basic design principles (that is, main program entities, classes, libraries, etc.)?
- What are the open problems?
- What are the possible future extensions?
- Have you added any extra features?

Milestones

- March 30, initial design of specification file grammar.

Deliveries

- Milestone results
- A package (ZIP-file) that contains both the applications and the source code.
- A written report.

Important Dates

- Tuesday, April 25, 2006: Submission of the project package.
- Friday, April 28, 2006: Submission of the written report.