Assignment 2: Parser

CSI4104 Compiler Design, Fall 2019
Document version 1.4

Due date: Wednesday Nov 20, 2019, 23:00.

Introduction

We will implement a recursive descent parser (RD parser) for MiniC. With this assignment, we only check the syntactic correctness of the input program. In Assignment 3, we will complete this parser to build an Abstract Syntax Tree (AST).

Specification

Like C, C++ and Java, MiniC also has the dangling-else problem (see the lecture slides). Your parser must match an **else** branch with the closest previous unmatched **then** branch.

Your parser will call the scanner that you developed in Assignment 1 to obtain the token sequence from the input program. If your scanner does not work properly, you can use the scanner provided to you with this assignment (see Section "Scanner" below).

At this stage, your compiler consists of a scanner and a parser. The compiler is required to work as follows. If the input program is syntactically legal, then your compiler should print "`Compilation was successful.`" as the last message. If the input program is syntactically illegal, then your compiler should print "`Compilation was unsuccessful.`" as the last message. Before this message, your parser is required to print an error message.

Implementing Your Parser

We provide you with an updated version of the skeleton compiler that you should extend to create your MiniC parser. You will find the new skeleton on the server in the archive file /opt/ccugrad/Assignment2/CSI4100_Assignment_2.tgz. Please set up your programming environment as specified in Assignment 1.

In the following, you find a description of the skeleton-files that have been added or changed compared to Assignment 1. All directories are **relative** to the top-level directory `CSI4100_Assignment_2`:

Package MiniC (directory ``MiniC'')

ErrorReporter.java: counting and issuing of syntax errors.
MiniC.java: the new version of the compiler driver.

Package Scanner (directory MiniC/Scanner)

Scanner.java: empty scanner implementation which you should **replace** by **your own scanner** from **Assignment 1**. In case your scanner from Assignment 1 does not work properly, you can use scanner classfiles provided with this assignment instead (see Section ``Scanner'' below).

Package Parser (directory MiniC/Parser)

Parser.java: the skeleton parser for you to complete.
SyntaxError.java: the SyntaxError exception class.
Parser/tst/base/testcases/: the testcases for this assignment.
Parser/tst/base/solutions/: the solutions for this assignment.

Parser testscript (directory MiniC/scripts)

parsertest.sh: this is a shell-script that you can use to automatically test your parser. The script will run your parser on every testcase in the Parser/tst/base/testcases directory and compare the output to the corresponding solutions in Parser/tst/base/solutions. To run this script, change to the MiniC directory and type ``*./parsertest.sh*'' (please note the ``./'' before the script's filename ``parsertest.sh'').

Please note: the parser testscript is provided to help you debugging your parser. The second purpose is to show you how large testing tasks can be automated by scripts. For the remaining assignments, you will have to create such scripts by yourself (e.g., by adapting the scanner or parser testscripts). **This script is provided as is, without guarantees about its functionality and correctness. Please use at your own risk, and do make sure that you also test your parser manually, to have sufficient confidence that your parser works correctly.**

Gradle buildscript (top-level directory CSI4100_Assignment_2)

gradle.build: updated build-script with two new build-tasks. For details, please refer to Section ``Building your parser'' below.

The parser provided to you compiles immediately, but it parses only a very limited subset of the MiniC language. Please **do not** expect any testcase to be parsed correctly at this stage. You should extend this parser to parse the complete MiniC language. This will take about 400 lines of Java code.
The testcases c25-c27 and c66-c74 are illegal according to the MiniC syntax. The solution files for these testcases contain error messages. You are **not** required to produce the same error messages. The only messages that count for marking are "`Compilation was (un)?successful.`". Note that the skeleton parser does not have an elaborate error handling strategy yet (for a production compiler more pointed error messages would be required).

In the MiniC grammar, several productions are left-recursive. You should replace left-recursive productions by their equivalent EBNF productions (as discussed in the lectures). The MiniC grammar is not LL(1) because of productions like

```
program          ::=   (variable-def | function-def)*
```

Both variable-def and function-def begin with the same symbols "typespecifier ID". Left-factoring eliminates this parsing conflict. You should transform the MiniC grammar to make it LL(1). The lecture slides contain hints on how various types of productions can be encoded with a recursive descent parser. Note that in order to parse X*, we need to look at all tokens that possibly start a sentence derived from X; these tokens are referred to as FIRST[X]. Given the first-set FIRST[X], X* is parsed as

```
while (currentToken.kind is in  FIRST [X]) {
  parse X();
}
```

The lecture slides on ``Syntax Analysis'' contain examples for this case.

Building Your Parser

Like with the previous assignment, we use the Gradle build automation system to compile our MiniC compiler. File gradle.build defines two build tasks:

- jar: to build the MiniC compiler using your scanner from Assignment 1. To do so, you should copy your Scanner.java file from Assignment 1 to directory MiniC/Scanner/.

- jarNoScanner: to build the MiniC compiler using the scanner classfiles provided with Assignment 2 (see Section ``Scanner'' below).

To build the MiniC compiler, navigate to the top-level directory of the Assignment 2 source tree, which is `CSI4100_Assignment_2`, and use either of the two build tasks, i.e., type:

```
./gradlew jar
```

or

```
./gradlew jarNoScanner
```

Note: the jarNoScanner build task relies on the dummy Scanner.java implementation provided with Assignment 2. If you change the dummy scanner file, e.g., by replacing it with your scanner from Assignment 1, then the "jarNoScanner" build may break. If you experience build problems with ``jarNoScanner'', restore the dummy Scanner.java implementation in directory MiniC/Scanner.

Scanner

In case your scanner from Assignment 1 does not work properly, you can use the scanner classfiles provided in the ``resources'' sub-directory. Compile using the "jarNoScanner" build task. This build task will compile the MiniC framework and copy the provided scanner's classfiles to the directory where gradle keeps all classfiles of a build, i.e., in directory `build/classes/`.

The ``jarNoScanner" build task depends on the dummy Scanner.java implementation provided with Assignment 2. This dependency is only to keep the MiniC framework compile-able. After compilation, the Scanner.class file generated from the dummy scanner is replaced by the provided scanner's classfile from the ``resources'' sub-directory (see file build.gradle and the copyClassfilesNoScanner build task for further details).

Running and Testing Your Parser

Like with Assignment 1, gradle creates a jar-file in subdirectory `build/libs`. As with Assignment 1, you can run the compiled MiniC compiler straight away:

```
java -jar build/libs/MiniC-Parser.jar MiniC/Parser/tst/base/testcases/c1.txt
```

will produce the output

```
********** MiniC Compiler **********
Syntax Analysis ...
Compilation was (un)successful.
```

You are encouraged to find additional testcases to the ones already provided to you. This will ensure that your parser can cope with all MiniC syntax issues.

Syntax Errors

Your parser is not required to recover from a syntax error. On discovering the first syntax error, your parser is expected to print an error message. Your parser can then stop processing the remaining input. The skeleton compiler already provides the infrastructure to support this limited form of error handling. On encountering a syntax error, a parse method should call method syntaxErrror() from the parser, which will call the ErrorReporter object and throw a SyntaxError exception. This exception will be propagated to the nearest enclosing exception handler (parse() in our case, see also the lecture slides on "Exceptions in Java"). Method compileProgram() in MiniC.java examines the error count from the ErrorReporter object to find out if a compilation was successful or not.

Marking

Your parser will be assessed by how it handles correct and incorrect inputs. We will use a large number of test-cases for marking. The test-cases provided to you are a subset of the test-cases that we will use during marking. Testing will be done on the server, so please make sure that your parser compiles and executes there.

Deliverables and Submission

You should extend the parser in file Parser.java so that it covers the complete MiniC specification. Note that for this assignment it is only necessary to modify file Parser.java. Submission will therefore be restricted to this single file. If you need to submit additional files, you must let me know in advance. *You are not allowed to modify other files, e.g., Token.java and SourcePos.java, otherwise automated testing of your parser might no longer work.*

To submit your file Parser.java, please log in on our server, create a submission directory, e.g., ~/myassignment2, and copy your modified Parser.java to this directory.

After copying file Parser.java to your submission directory, you must cd to the submission directory and type 'submit'. This will collect your file and copy it to our repository.

```
[you@elc1 ~] cd ~/myassignment2
[you@elc1 myassignment2] submit
```

Please note: There is no submission unless you issue the 'submit' command! Failing to issue the 'submit' command will result in 0 points. The timestamp of your submission will be recorded by the 'submit' command. Please note also that if you re-submit, then a previous submission will be overwritten and a new timestamp will be recorded.

If you have questions, don't hesitate to contact us. ☺

Good luck!