Parser

This assignment is based on the project described in *Modern Compiler Implementation in ML*, section **PROGRAM**, pages 81-82 and pages 100-101. The source code mentioned in this section (as being available from `$Tiger/chap3` and `$Tiger/chap4`) should be taken from the Documents (https://bb.au.dk/webapps/blackboard/content/listContent.jsp?course_id=_43271_1&content_id=_355480_1&mode=reset) part of this website, stored in the tar archive `part3.tgz`. This archive should be unpacked and used for an initial test as follows (where `myPrompt` stands for your command line prompt, such as `dovs@linux-en0j:~`, and it is assumed that you downloaded `part3.tgz` to your home directory, `/home/dovs`):

```
myPrompt> cd ~/dovs
myPrompt> tar xzf ~/part3.tgz
myPrompt> cd part3
myPrompt> make sa
...
myPrompt> ./compiletests
...
```

In order to obtain a working lexer you need to copy the contents of your `tiger.lex` from part2 to `part3/tiger.lex`. You cannot simply copy the file `tiger.lex` into part3, because you need to use a `type svalue`, `type token`, `type lexresult`, and `%header` directive as specified in `part3/tiger.lex`, but the rest should be directly reusable in part3. The file `rescue-tiger.lex.sml` contains the ML-Lex output from a working lexer; you may use this in case your own lexer does not fully work.

The `Makefile` contains several goals that you may use (to use goal `X`, execute `make X`). Compared to part2, the goal `sa` has been added to the `Makefile` as a shorthand for the goal `standalone`, which will generate a stand-alone binary compiler that is executed by the script `tigerc`. For an interactive session ('REPL'), use the goal `ia`, or the long form `interact`. A new goal `rs`, short for `rescue`, has been added. This goal will generate a binary compiler using the lexer implementation in `rescue-tiger.lex.sml`. You may inspect the `Makefile` and the script `forcetigerc` to see how this is done; the main point is that `touch` is used on `tiger.lex.sml` to ensure that it is used as-is, rather than being overwritten (if it is older than `tiger.lex` then the compilation manager will run ML-Lex and it will then generate a new `tiger.lex.sml` based on your `tiger.lex`, which will prevent the use of the rescue lexer).

Parser requirements

The parser should parse the Tiger language as specified in the appendix, extended with the exponentiation expression *exp* ^ *exp*. Exponentiation operator has the second-highest precedence after unary minus, i.e., it binds tighter than * or /, but weaker than unary minus.

The parser should use the semantic actions to construct an abstract syntax tree for the program. Note that the initial source code provided for this project is ready for the construction of a parser that does not return anything, corresponding to a set of empty semantic actions. It is recommended that you construct such a "check only" parser first, and only implement the semantic actions when the grammar works correctly. Note that the parser return type must then be adjusted in several locations in the source code.

Parsing conflicts

The parser should not have reduce/reduce conflicts or shift/reduce conflicts. If you have a small number of shift/reduce conflicts, then they should be described in your report, explaining why they are benign. It is possible to create the parser without shift/reduce conflicts so this should be your goal.

Test cases

You should also create 5 small Tiger programs containing syntax errors, and describe how your parser reacts to them in your report. Make sure that your implementation works on the provided test files `../testcases/test{0,1,2}?.tig` , which should all parse without errors. You should deliver the source code in an archive:

```
myPrompt> cd ~/dovsproject
myPrompt> zip -r ~/part3.zip part3
```

Report

The project should be documented by a short report, only a handful of pages. It may be formatted using LaTeX or other document processing systems and delivered as a PDF file, or it may be a plain ASCII file. The report must briefly describe your approach, especially insofar as it differs from the project description due to, e.g., problems making it work, or interesting experiments. Use the report template provided for this hand-in as a starting point.

Deliverables

- `part3.zip` This is a zip archive file that will contain your solution. This file can be obtained by running the command

  ```
  myPrompt> cd ~/dovsproject
  myPrompt> zip -r ~/part3.zip part3
  ```

  This file should include the source code to your solution.
- `part3-tests.zip` This is a zip file (created in a similar manner as above) that should contain the five test cases that you created.
- *Project report* as a PDF file. See the section of the site for a template, and description above for what to include in the report.

Submission instructions

Upload three file: `part3.zip` , `part3-tests.zip` , and `part3-report.pdf` using the course submission system (http://dovs.cs.au.dk/).

Deadline

Wednesday, September 23rd, 14:00.

Content on this page is maintained by Aslan Askarov.