CPSC 4660 PL Compiler

Steven Deutekom
Ricky Bueckert
University of Lethbridge

March 30, 2020

1 Contributions

The entire class contributed equally on this project. At each phase we split the work up evenly. Until the current work from home pandemic response we collaborated in the lab whenever we worked. This involved deciding what to work on next when a portion of the project was finished. It also involved debugging and design decisions. It is always helpful to have another person to bounce ideas off of and a fresh pair of eyes when code is not working. We feel the team aspect was valuable both for finishing the project and for learning about team work for our future jobs and research.

2 Files Included

2.1 Header Files

- Symbol.h
- SymbolTable.h
- Token.h
- Scanner.h
- Administration.h
- Parser.h
- Grammar.h
- Types.h
- BlockTable.h
- Assembler.h - Provided by professor

2.2 Source Files

- plc.cc
- SymbolTable.cc
- Token.cc
- Scanner.cc

- Administration.cc
- Parser.cc
- BlockTable.cc
- Assembler-2.cc - Provided by professor

2.3 Test Files

- search.pl - Search a list of integers for a given integer
- fact.pl - Recursivley calculate factorials
- recursion.pl - Simple recursion to make a given number 0
- arith.pl - Tests the basic arithmetic operators given 2 numbers
- bool.pl - Tests the boolean operators
- table.pl - Input a table and it is printed out again
- maxmin.pl - Finds the min and max number in a given list of 5 integers
- errorTest.pl - Tests a number of different errors. Tries to ensure that the parser catches all errors and recovers to a stable state.
- typeErrors.pl - Check a few of the possible type errors.
- p2.pl, simple_proc.pl - Provided by professor to ensure proper output of parser
- driver.cc, interp.h, interp.cc - Provided by professor for the interpreter

3 Compiling and Running

1. Navigate into the project folder and type the following:
   ```
   $ make
   ```
2. This will create a binary called compiler that can be run with the following command:
   ```
   $ compiler <testfile-path> -o <output-file>
   ```
3. If you omit the `-o <output-file>` flag the tokens will be output to a file called `pl.out`
4. If you add the `-v` flag some debugging info will be printed. This will include each function that is called and matched tokens. This will not affect the output of tokens.
5. In addition to the compiler an interpreter will be created as well. It can be run as follows:
   ```
   $ interpret <compiled-file-path>
   ```
6. When running programs `Input:` allows you to enter a single integer.
7. Output from programs will be displayed with `Output:`

4 Bugs

No bugs. However, while testing in the final phase we discovered our grammar does not allow for negative constants. The unary minus is part of the simpleExpression rule in the grammar. Constant definitions do not use this rule, but use the constant rule. The constant rule only deals with numbers, bools, and identifiers. So without some changes to the grammar for now there are no negative constants. Though it is possible to use negative numbers everywhere else.

5 Time Spent

Hours spent on this project combine all group member contributions. Scanner 30 hrs. Parser 40 hours. Rule additions and Scope and Type checking 30 hours. Code generation 20 hours.