Compiler Construction - Assignment 1

LIACS, Leiden University

Fall 2008

Introduction

In this exercise, you will learn how to use the lexical analyzer generator "`flex`" and the parser generator "`bison`". Both of these tools are widely used in compiler front-end development. For this exercise, a framework of a simple calculator written using `flex` and `bison` is already given. Your task is to extend this framework, thereby adding functionality to the calculator.

Assignment

In your CVS repository, a module named "`assignment1`" is available. This module contains a framework for a simple integer calculator that is implemented using `flex` (the `calc.l` file) and `bison` (the `calc.y` file). A makefile is provided; after performing a checkout of the CVS module, first execute `make first` once. In order to build the calculator, simply execute `make`.

Currently, the calculator only supports the addition and unary minus operators. Your task is to add the following functionality to the calculator:

- Extend the `calc.l` file such that the sequence "`PI`" is recognized as a single token. When the `PI`-token is encountered during the parsing phase, the value `3` has to be filled in. E.g., the expression `4+PI` should evaluate to `7`.
- New mathematical operators:

Substraction	`x - y`
Multiplication	`x * y`
Division	`x / y`
Modulo	`x % y`
Exponentiation	`x ** y`
Factorial	`x!`

- Support for parentheses, such that expressions like `(x - (y + z)) + 1` can be parsed and computed properly.
- Absolute value of an expression, e.g., `|2| = 2` and `|-2| = 2`.
- Take care of the following precedence rules for the binary operators, given from highest to lowest. Operators on the same line take the same precedence:

  ```
  **
  !
  *   /   %
  +   -
  ```

  Enforce these precedence rules without changing the grammar! Instead, use dedicated bison/yacc declarations to achieve your goal.

A sample input file (`input8`) is provided. Your calculator should be able to parse and correctly compute all of the expressions in this file. Some additional remarks:

- The exponentiation operator is right associative: `2**1**3` should be evaluated as `2**(1**3)`.
- The exponentiation and factorial operators should take precedence over the unary minus operator. E.g., `-2**2` should evaluate to `-4` (and not to `+4`).
- Assume `0! = 1`. For negative values, the factorial operator is not defined. In such a case, print an error message.
- The grammar of the framework contains four global non-terminals: `line`, `stmt`, `expr` and `number`. Together with other bison/yacc functionality, this is enough to accomplish all goals. Therefore, you are not allowed to add any new non-terminals to the grammar.

Submission & Grading

Submit your work using CVS. Do not send anything by email. Make sure the latest version of your work has been committed to the CVS repository. Then, tag that version using the command:

```
cvs tag DEADLINE1
```

This tag command has to be issued *before* October 2nd, 2008 at 23:59.

This assignment is marked either as sufficient or insufficient. When you receive an insufficient mark, you have exactly ONE opportunity to improve your work. If your improved version satisfies the requirements, you will receive a sufficient mark. Note that this assignment has to be sufficient in order to receive a final grade for the practical part of the Compiler Construction course. Also note that submitting your work too late or submitting work that is not of your own will result in an insufficient mark.

Assistance will be given by Michiel Helvensteijn in room 306/308. A schedule for this can be found on BlackBoard. Additionally, you can go to Sven van Haastregt in room 1.22.