

Project #3
CpSc 8270: Language Translation
Computer Science Division, Clemson University
Evaluation and Visualization of an Expression Tree
Brian Malloy, PhD
September 15, 2016

Due Date:

In order to receive credit for this assignment, your submission must be submitted, using the `web handin` command, by 8 AM, Tuesday, October 4th of 2016. If you are unable to complete the project by the first due date, you may submit the project within three days after the due date with a ten point deduction.

Project Specification:

You will find a directory, `astCalc`, in the course repo at:

`8270Assets-2016/projects/3/astCalc`

`astCalc` contains a scanner and parser, `scan.l` and `parse.y`, that adapts the routines found in “flex & bison”, `fb3-1`, by John Levine. Also, `astCalc` has a class, `Ast`, that builds an *abstract syntax tree* for the expressions evaluated in the parser, `alltest.py`, and `cases`. You can use all of `astCalc` as a starting point to accommodate the specification of this project, or begin fresh if you wish, but I suggest you study it in either case.

The specification of the project is that you must use `flex` and `bison` to build an *abstract syntax tree* or *expression tree* to evaluate arithmetic expressions. The code in `astCalc` already evaluates some expressions but, for the project, you must extend this code to include additional operators, and add code that uses `dot` to build a graphical representation of the expression tree built by the `bison` generated parser. However, don't use the factored grammar in `astCalc` but rather set operator precedence in your `bison` specification.

Your solution should allow the operators in expressions of the form: $\{x + y, x - y, x * y, x/y, x^{**}e, (x), -x\}$; which are *addition*, *subtraction*, *multiplication*, *division*, *exponentiation*, *parentheses*, and *unary minus*, respectively.

We will discuss the `astCalc` example, and the `dot` tool, during lecture. `dot` is a graph drawing tool, included as part of the `graphviz` suite of drawing tools, and is considered by many to be the default tool to use to visualize graphs. I have placed a `dot` reference in the repo and you will probably find there all of the information you will need about `dot`. Sample expression tree solutions can also be found on the next page.

Your code should be properly organized, indented, and free of memory leaks in both your `flex/bison` specifications as well as the code that builds the expression tree itself. If you complete these specifications you can receive a grade of 90%. Additional credit may also be earned through interesting extensions of the project. For one example, the code in `astCalc` is patterned after the code written by Levine and is not object oriented. An object oriented implementation would therefore be an example of an interesting extension. Another alternative: *variables*. Also, provide for two types: `int` and `float`.

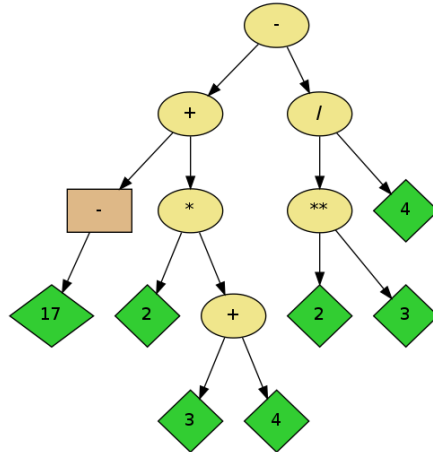
Submission:

Your submission must be a compressed directory, use `tar` or `zip`, and should include a `README` file. Your `README` must be an `ascii` file. **Do not** submit a `README` in `rtf`, `word`, `pdf`, or **any other** format. Submit only an `ascii` file consisting of digits or upper/lower case letters. You must name your file `README`.

Your project will be tested on an Ubuntu platform running Linux 14.04, and compiled with gcc C++ version 4.9.3, or clang++ 3.3. You must submit your project using the web handin command.

input: $-17 + 2*(3+4) - 2*3/4$

output: -5



input: $-17 + 2*(3+4) - 2*3/4$

