Project #3
CpSc 8270: Language Translation
Computer Science Division, Clemson University
Evaluation and Visualization of an Expression Tree
Brian Malloy, PhD
October 4, 2016

Due Date:

In order to receive credit for this assignment, your project must be submitted, using the `web handin` command, by 8 AM, Saturday, October 8^{th} 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, scan.l, and a parser, parse.y, that adapts the routines found in "flex & bison", fb3-1, by J. Levine. Also, astCalc has a class, Ast, that builds an *abstract syntax tree* for the expressions evaluated in the parser. Finally, astCalc contains a test harness, test.py, and a directory containing test cases and their output, cases. You can use all of astCalc as a starting point to accomodate the specification of this project, or begin fresh if you wish, but I suggest you study astCalc in either case.

The current state of astCalc is that it uses flex and bison to build an *abstract syntax tree* or *expression tree* that evaluates arithmetic expressions. For this project, modify or extend astCalc as follows:

1. Do not use the factored grammar in astCalc but rather set operator precedence in the bison spec.
2. The current scanner, scan.l, returns literal character tokens; modify the scanner to return enum tokens, generated by Bison.
3. Extend astCalc to include additional operations 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.
4. Extend astCalc so that it uses dot to build a graphical representation of the expression tree built by the bison generated parser.
5. Extend the test cases contained in the directory cases to adequately test your project. Beware of a space at the end of the $>$.

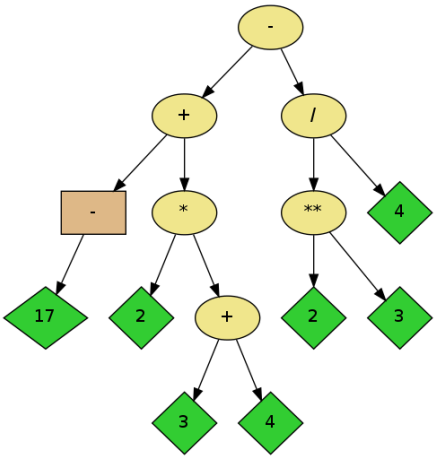
If you complete these specifications you can receive a grade of 90%. Additional credit may be earned through interesting extensions of the project. As an example, the code in astCalc is patterned after the code written by Levine and is not object oriented. An object oriented implementation would be an interesting extension. Another alternative: *variables*. Also, provide for two types: int and float.

Submission:

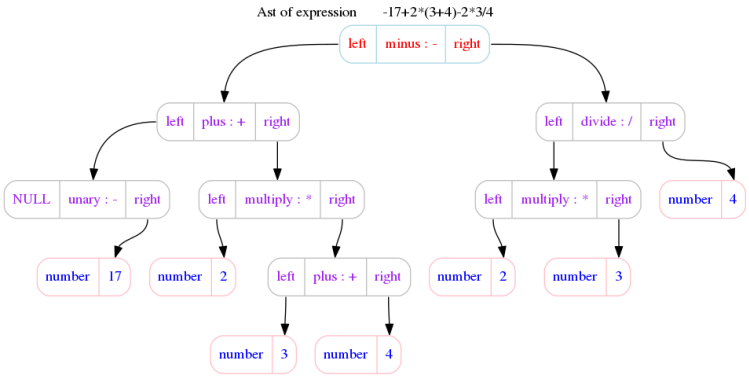
Your submission must be a compressed directory, use tar or zip, and should include a README file. Your README must be an ascii file. **Do not** submit a README in rtf, word, pdf, or **any other** format. Submit only an ascii file consisting of digits or upper/lower case letters. You must name your file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

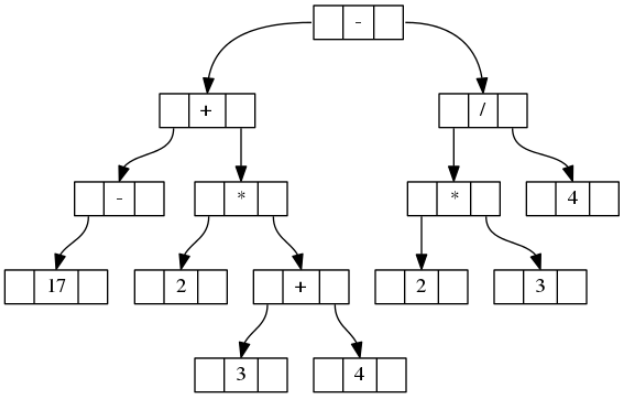
Your project will be tested on an Ubuntu platform running Linux 14.04, and compiled with gcc C^{++}version 4.9.3, or clang++ 3.3. You must submit your project using the web handin command.

```
input:  -17 + 2*(3+4) - 2**3/4
output: -5
```




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




-
+
/
-
*
*
4
17
2
+
2
3
3
4