Design of Optimizing Compilers CS5214: Assignment #1

Due on Friday, Feb 5th, 2016

Weng-Fai Wong 18:30

Pan An(A0134556A)

Contents

Design of Calculator

This calculator is designed and implemented in Mac OS X system, any unix compatible system can be able to install and run the program. The calculator is designed with the utilization of FLex and Bison. Basic functionalities in the original design is listed below:

- Numerical system. The calculator should be able to detect different data types including: integers, floating point numbers and hexadecimals.
- Operations. The Calculator should be able to conduct basic BODMAS rules. Also it should support basic scientific calculations like trigonometric and logrithmic operations.
- Batch processing. The calculator should be able to batch process many calculations from a specific file.
- Symbolic computation. This feature is not implemented due to the lack of time. But the calculator should be able to integrate with symbolic engines and conduct complex mathmatical operations.

The detailed design and introduction about the calculator is listed in the following sections. The tarball name of my project is:

$$PanAnassignment1.tar.gz$$

Installation

The installation of the calculator can be easily done with a customized **makefile**. Open terminal and switch to the calculator folder:

$\$make$
$\$./calculator$

The calculator should be running. You can type in any tasks, if the task expression is wrong, the calculator will report error and exit.
You can also put multiple tasks to **test.txt**, then enter the calculator and type:

$\$test$

the results of all the tasks(before the first wrong task expression) should be processed and displayed.

Numbering System and Basic Calculation

The numbering system of the calculator supports basic integer, scientific floating number and hexadecimals. Noticing that hexadecimals in this case is all considered and will be transformed to 32-bit integers in any task.
Some correct case of scientific floating expression is listed below:

- .1 (0.1)
- 1e3 (1000)
- 1e-2 (0.01)
- .1e (0.1)
- -0.1e-4 (-0.00001)

[Numbering System and Basic Calculation] continued on next page...

The calcolator supports BODMAS rules expressions. Basic operations includes: +, -, *, / and ** (power). Here is an example of task that contains different operations:

$$(2 + 3 * *3) * *(3 * (4 + 6) - 12)$$

and the result is: 210457279823473787946401792.000000.

Scientific Calculation

Supported scientific operations are listed below:

- Constant number pi
- Trigonometric and hyperbolic: sin, cos, tan, sinh, cosh, tanh
- Standard functions: mod, ceil, floor, abs, sqrt
- Logrithmics: log2 and log10

Here is an example of task that contains different operations:

$$(sin(.5 * pi)) + (cos(pi)) * *2$$

Noticing that all of the scientific operation expressions should have extra '()' around it.

Challenges and Solutions

During implementation of this calculator, one of the challanges is that when designing the calculator the designer should be completely clear about each data types and operations. For example: floating numbers it self have a lot of cases that should be taken into consideration. By using better designed regular expression I was able to make sure that it does not go wrong. By splitting up integer number and floating number into different expression definition block, I was able to manipulate the whole calculation structure without going wrong.
Also the utilization of third class libraries greatly reduced the amount of work done my myself.
Flex itself controls the whole process of the script, one of the main difficulties to make the calculator better accepted by everybody. The parsing and loop controlling is controlled by FLex itself. The only way to make this work is to override different functions. I used some function pointer tricks to make it look better when the program goes wrong. Like when the calculator finished the test.txt batch processing the program is supposed to give an error saying invalid syntax because EOF is not gonna be processed correctly. I override the flex function and pointed out come to it, so what you get is:

$$AllResultsfromTestFile!!$$

One thing that I did not achieve in my original design is the symbolic engine integration. Since Python has provided strong and mature packages for symbolic calculation, I was able to model a better version of calculator by having some more expression definitions and having the calculator interact with python scripts. However the time is not enough for me to implement all these features.
Another difficulties is to consider the operating system compatibility of the calculator. Some of flex libraries have different compilation rules in different operating systems. Luckily I solved with a simple Makefile.

continued on next page...