**CS5214 –DESIGN OF OPTIMISING COMPILERS**

**Programming Assignment 1**

|  |  |
| --- | --- |
| Submitted By | VICTOR DOMINGO DELA CRUZ |
| Matriculation ID | A0120578E |
| Submitted File Name | victor-delacruz-assignment-1.tar.gz |
| Submission Date | 05 Feb 2016 |

Table of Contents

[1 Description of Assignment 1 2](#_Toc442434575)

[2 Solution Package Overview 2](#_Toc442434576)

[3 Solution Description 2](#_Toc442434577)

[4 Scanner 4](#_Toc442434578)

[4.1 Supported Symbols and Operations 4](#_Toc442434579)

[4.2 Supported Numbers 4](#_Toc442434580)

[5 Parser 5](#_Toc442434581)

[5.1 Grammar Structure 5](#_Toc442434582)

[6 Conclusion 5](#_Toc442434583)

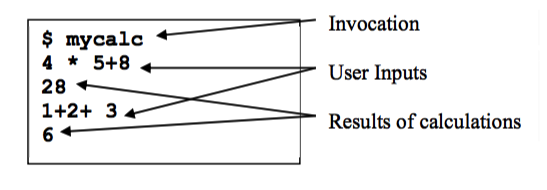
[6.1 Key Features 5](#_Toc442434584)

[6.2 Retrospection 5](#_Toc442434585)

[6.3 Enhancement Opportunities 5](#_Toc442434586)

# Description of Assignment 1

Using JLex (or JFlex) and CUP, implement a simple calculator that will parse an arithmetic expression and return the result. Here is an example of the behaviour of the program:



# Solution Package Overview

The following tree view shows the key structure of the submitted implementation solution for the programming assignment.

.

├── Calculator /\* main folder \*/

│   ├── Calculator /\* package contains the compiled Calculator classes \*/

│   ├── Calculator.cup /\* CUP input specification \*/

│   ├── Calculator.lex /\* JLex input specification \*/

│   ├── JLex /\* folder contains JLex source and classes \*/

│   │   ├── Main.java /\* JLex source code

│   ├── JavaCup /\* contains all CUP related files and binaries \*/

│   │   ├── java\_cup /\* package contains the java\_cup classes \*/

│   ├── README /\* file describing compilation and invocation \*/

│   ├── Yylex.java /\* scanner file, renamed output from JLex \*/

│   ├── mycalc /\* bash shell script to compile and run everything in one go \*/

│   ├── parser.java /\* parser code, output from CUP \*/

│   ├── setenv /\* adds Calculator dir to ${PATH} in bash \*/

│   └── sym.java /\* symbol code, output from CUP \*/

# Solution Description

The solution is broken down into two main process components, the **Scanner** and the **Parser**. The specifications for each are described in details in the later sections.

* The ***scanner*** is implemented using the ***JLex*** lexical analyzer framework.

|  |  |  |
| --- | --- | --- |
| **Process Steps** | **Command** | **Remarks** |
| 1. For platform compatibility, the JLex source code is compiled prior to initial use | ***> javac -target 1.8 JLex/Main.java*** | where: -target 1.8 option corresponds to the java –version installed on the machine (Java 8 in this case) |
| 1. A file **Calculator.lex** is provided as an input specification |  | Refer to Scanner section for details |
| 1. The specification is then fed to the JLex scanner generator program to generate the customize scanner program | ***> java JLex.Main Calculator.lex*** | This step generates the java scanner program named Calculator.lex.java |
| 1. Rename the output file from last step to match class name later use with the parser program | ***> mv -f Calculator.lex.java Yylex.java*** | Where “-target 1.8” option corresponds to the java –version installed on the machine (Java 8 in this case) |

* The ***parser*** is implemented using the Java-based ***Constructor of Useful Parsers (CUP)***.

|  |  |  |
| --- | --- | --- |
| **Process Steps** | **Command** | **Remarks** |
| 1. A file Calculator.cup is provided as an input specification to CUP |  | Refer to Parser section for details |
| 1. The .cup file needs to be compiled using the .jar file provided in CUP framework | ***> java -cp JavaCup java\_cup.Main Calculator.cup*** | This step generates the java parser programs named *parser.java* and *sym.java* |
| 1. Compile the generated files together with the renamed scanner file from scanner section above | ***> javac -cp .:JavaCup -d . parser.java sym.java Yylex.java*** | This step generates the java scanner program |
| 1. The program is now ready to use and may be invoked using this command | ***> java -cp .:JavaCup Calculator.parser*** | Where “-target 1.8” option corresponds to the java –version installed on the machine (Java 8 in this case) |

* A script is provided as an alternative method for compilation and invocation for UNIX bash:

|  |  |  |
| --- | --- | --- |
| **Process Steps** | **Command** | **Remarks** |
| 1. Include “.” in $PATH environment var | ***> source setenv*** | Run from the root Calculator/ folder |
| 1. Compile and run | ***> mycalc*** | Modify the target –version accordingly |

# Scanner

The scanner module directly interacts with the user and is implemented using the **Jlex** framework. It generates tokens for the parser module. The download link for the source code is given below:

[*http://www.cs.princeton.edu/~appel/modern/java/JLex/current/Main.java*](http://www.cs.princeton.edu/~appel/modern/java/JLex/current/Main.java)

## Supported Symbols and Operations

The following table list downs the supported symbols, their meaning and example usages

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Symbol** | **Functionality** | **Examples** |  | **Symbol** | **Functionality** | **Examples** |
| + | Addition | 1+1e-3; 0xde + 2.5; |  | ~ | Bitwise negation | ~1; ~0xdead; |
| - | Subtraction | 1-1e-3; 0xde - 2.5; |  | << | Shift left | 1<<0xde; |
| \* | Multiplication | 1\*1e-3; 0xde \* 2.5; |  | >> | Shift right | 1>>0xde; |
| / | Division | 1/1e-3; 0xde / 2.5; |  | & | Bitwise AND | 1&0xde; |
| % | Modulo | 0xdeadbeef % 21; |  | ^ | Bitwise XOR | 1^0xde; |
| sqrt | Square root | sqrt(2); sqrt1e5; |  | | | Bitwise OR | 1|0xde; |
| log | Logarithmic | log(2); log1e5; |  | ^^ | Exponentation | 1^^1e-3; 0xde^^2.5; |
| sin | Sine | sin(2); sin1e5; |  | cot | Cotangent | cot(2); cot1e5; |
| cos | Cosine | cos(2); cos1e5; |  | sec | Secant | sec(2); sec1e5; |
| tan | Tangent | tan(2); tan1e5; |  | csc | Cosecant | csc(2); csc1e5; |
| ( ) | Precedence | (1+2)/3-4\*5; |  | ; | End-of-line | See above |

## Supported Numbers

The following table list downs the supported number types

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Notation** | **Types** |  | **Notation** | **Types** |
| 0-9 | Decimal digits |  | 0-9a-f | Hexadecimal digits |
| …,-1,0,1,… | Long integers |  | 1.23 | Floating numbers |
| 0x | Hexadecimal Notation |  | 1.23e-5 | Scientific notation |

# Parser

The parser module takes in the scanner module as an input and directly performs parsing of the tokens generated by the scanner during runtime, performing Shift-Reduce operations. The library is provided in the link below.

*http://www2.cs.tum.edu/projects/cup/*

## Grammar Structure

There are two main grammar branches separating integer (type: Long) and float (type: Double) actions. The diagram below shows the flow structure of the parsing solution algorithm.



# Conclusion

## Key Features

* Mixed integer and float argument support.
  + Resolves as float result
* Leverage on Java.Math library functions
  + Able to handle infinite results nicely

## Retrospection

* Main concept is to align lexical design with production grammar rules
* Unable to implement newline terminal token – may need more time to understand JLex/CUP
* Resolved mixed integer and float arguments by creating separate grammar branch

## Enhancement Opportunities

* Better error handling – currently relies on default action
* Newline token instead of semi-colon for interactive program invocation