ChemLAB

COMS W4115 - Programming Languages & Translators

Professor Stephen Edwards

Alice Chang (avc2120) Gabe Lu (ggl2110) Martin Ong (mo2454)

December 14, 2014

Contents

Introduction **2**

1 Language Tutorial **3**
1.1 Program Execution 3
1.2 Variables 3
1.3 Control Flow 3
1.4 Functions 4
1.5 Printing to stdout 4

2 Language Reference Manual **5**

3 Project Plan **6**
3.1 Proposed Plan 6
3.2 What Actually Happened 7
3.3 Team Responsibilities 7
3.4 Project Log 8

4 Architectural Design **9**

5 Test Plan **10**

6 Lessons Learned **11**

A Code **12**

Introduction

ChemLab is a language that will allow users to conveniently manipulate chemical elements. It can be used to solve chemistry and organic chemistry problems including, but not limited to, stoichiometeric calculations, oxidation-reduction reactions, acid-base reactions, gas stoichiometry, chemical equilibrium, thermodynamics, stereochemistry, and electrochemistry. It may also be used for intensive study of a molecule's properties such as chirality or aromaticity. These questions are mostly procedural and there is a general approach to solving each specific type of problem. For example, to determine the molecular formula of a compound: 1) use the mass percents and molar mass to determine the mass of each element present in 1 mole of compound 2) determine the number of moles of each element present in 1 mole of compound. Albeit these problems can generally be distilled down to a series of plug-and-chug math calculations, these calculations can become extremely tedious to work out by hand as molecules and compounds become more complex (imagine having to balance a chemical equation with Botox: $C_{6760}H_{10447}N_{1743}O_{2010}S_{32}$). Our language can be used to easily create programs to solve such problems through the use of our specially designed data types and utilities.

Chapter 1

Language Tutorial

1.1 Program Execution

To compile a `.chem` program, simply use the compilation shell script with your `.chem` file as the only argument.

`./compile.sh yourProgram.chem`

After compilation, if there are no errors, there will be a Java program that gets created. One can the compile the Java program into an executable using `javac`.

1.2 Variables

Variables in ChemLAB must be declared as a specific type. To use a variable, declare the type of the variable, and assign it to the value that you want like this:

```
int myNum = 5;
String hello = "World";
```

1.3 Control Flow

ChemLAB supports "if/else" statements:

```
if(10>6){
print("inside the if");
else{
```

```
print("inside the else");
}
```

ChemLAB supports "while loops":

```
while(i > 0){
print(i);
i = i-1;
}
```

1.4 Functions

Functions are the basis of ChemLAB. Functions can be passed any amount of parameters and are declared using the function keyword. The parameters within a function declaration must have type specifications.

This is a function that takes in two parameters:

```
function add (int a, int b){
return a+b;
}
```

This is a function that takes in no parameters:

```
function noParam(){
print("Hello World");
return 1;
}
```

1.5 Printing to stdout

To print to stdout, simply use the built-in function "print"

```
print(6);
print("Hello World");
```

Chapter 2

Language Reference Manual

Chapter 3

Project Plan

Like any project, careful planning and organization is paramount to the success of the project. More importantly however, is the methodical execution of the plan. Although we originally developed a roadmap for success as well as implemented a number of project management systems, we did not follow the plan as intended. This section outlines our proposed plans for making ChemLAB happen and the actual process that we went through.

3.1 Proposed Plan

We had originally planned to use the waterfall model in our software development process in which we would first develop a design for our language, followed by implementation, and finally testing. The idea was for all team members to dedicate complete focus to each stage in the project. Especially since we only had three members on our team, our roles were not as distinct and everyone had the chance to work, at least in some capacity, in all the roles. We intended to meet consistently each week on for at least two hours. During our meetings, each member was suppose to give an update about what he or she had been working on the past week as well as plans for the upcoming week and any challenges he or she faced that required the attention of the rest of the group. To help facilitate communication and the planning of meetings, we used Doodle to vote on what times were best for meetings. Also, in order to improve team dynamics, we planned to meet at least once every two weeks outside the context of school in order to hang out and have fun. Development would occur mostly on Mac OS and Windows 7, using the latest versions of OCaml, Ocamllex, and OCamlyacc for the compiler. We used Github for version control and makefiles to ease the work of compiling and testing code. The project timeline that we had laid out at the beginning was as follows:

- Sept 24th: Proposal Due Date
- Oct 2nd: ChemLAB syntax roughly decided upon
- Oct 23th: Scanner/Parser/AST unambiguous and working
- Oct 27th: LRM Due Date
- Nov 9th: Architectural design finalized
- Dec 5th: Compile works, all tests passed
- Dec 12th: Project report and slides completed
- Dec 17th: Final Project Due Date

3.2 What Actually Happened

This graph was pulled from Github reflecting the number of commits being made over the span of this semester. Due to schedule conflicts and a false sense of security, we did not start intensely working on the project until after Thanksgiving break. Since we did not coordinate the development of the Scanner, AST, and parser with the writing of the LRM, our language did not have as concrete a structure as we had hoped. Furthermore, we did not have enough time to implement some of the features in our language such as object-orientation or more built-in functions. As we were developing the software, we did make sure to allow testing at all steps in the design process. In the test script, we had identifiers for how far in the compilation process we wanted the program to run. Thus, we were able to maintain testing capabilities even before all of our code was ready. We discuss the testing procedure in more detail in a subsequent section.

3.3 Team Responsibilities

This subsection describes the contributions made by each team member:

- Project Proposal - Gabriel L/Alice C/Martin O
- Scanner - Gabriel L
- AST - Alice C/Gabriel L/Martin O
- Parser - Alice C/Martin O
- LRM - Gabriel L
- Code Generation - Alice C

- Semantic Analyzer -Gabriel L/Martin O
- Testing - Martin O
- Final Report - Gabriel L/Martin O

3.4 Project Log

We add later

Chapter 4

Architectural Design

Chapter 5

Test Plan

Chapter 6

Lessons Learned

Appendix A

Code