

ChemLAB
Final Report
COMS W4115 - Programming Languages & Translators
Professor Stephen Edwards

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

December 17, 2014

Contents

Introduction	3
1 Language Tutorial	4
1.1 Program Execution	4
1.2 Variables	4
1.3 Control Flow	4
1.4 Functions	5
1.5 Printing to stdout	5
2 Language Reference Manual	6
2.1 Types	6
2.1.1 Primitive Types	6
2.1.2 Non-Primitive Types	7
2.1.3 Type Inference	8
2.2 Lexical Conventions	8
2.2.1 Identifiers	8
2.2.2 Keywords	8
2.2.3 Literals	8
2.2.4 Punctuation	9
2.2.5 Comments	9
2.2.6 Operators	10
2.3 Syntax	10
2.3.1 Expressions	11
2.3.2 Statements	14
2.3.3 Scope	15
2.4 Built-in Functions	16
3 Project Plan	17
3.1 Proposed Plan	17
3.2 What Actually Happened	18

3.3	Team Responsibilities	18
3.4	Project Log	19
4	Architectural Design	20
4.1	Scanning	20
4.2	Parsing and Abstract Syntax Tree	20
4.3	Semantic Analysis	21
4.4	Java Generation	21
5	Test Plan	22
5.1	Introduction	22
5.2	Sample Test Cases	22
6	Lessons Learned	26
6.1	Alice Chang	26
6.2	Martin Ong	27
A	Code Listing	28
B	Project Log	49

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, stoichiometric 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

`make` creates an executable `chemlab`. To compile and run a `.chem` program, simply run the executable `chemlab` with your `.chem` file as the only argument. `./chemlab <program name>.chem`

It then compiles the ChemLab file into Java bytecode, which is then executed on a Java virtual machine.

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. All programs in ChemLAB must contain one “main” function which is the starting point for the program. 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 main(int A, int B){
    print A;
}
```

This is a function that takes in no parameters:

```
function main(){
print "Hello World";
}
```

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

2.1 Types

2.1.1 Primitive Types

There are four primitive types in ChemLab: boolean, int, double, and string.

Boolean

The boolean data type has only two possible values: true and false. The boolean data type can be manipulated in boolean expressions involving the AND, OR, and NOT operators.

Integers

Much like in the Java programming language, the int data type is represented with 32-bits and in signed two's complement form. It has a minimum value of -2^{31} and maximum value of 2^{31} . There is no automatic type conversion between a variable of type int and of type double. In fact, an error will occur when the two primitive types are intermixed.

Double

Much like in the Java programming language, a double is a double-precision 64-bit IEEE 754 floating point with values ranging from $4.94065645841246544e - 324d$ to $1.79769313486231570e + 308d$ (positive or negative). Double should be used under any circumstance when there are decimal values.

String

Unlike in the C programming language, a string is a primitive type rather than a collection of characters. A string is a sequence of characters surrounded by double

quotes “”. Our language supports string concatenation. In the context of strings, the “+” operator concatenates two strings together to form a new string.

2.1.2 Non-Primitive Types

The language comes built-in with lists, elements, molecules, equation.

Lists

A list is a collection of items that maintains the order in which the items were added much like an ArrayList in Java. The type of items in a list must be declared and the type must remain consistent throughout the lifetime of the program. A list is declared in a syntax very similar to declaration in Java:

```
<type> <identifier>[] = [ element_1, element_2, ....., element_n]
```

Element

Since there are only 118 elements, it could have been possible to hard code each element into the language. However, we chose not to do this to give the user a greater degree of flexibility in terms of declaring the properties of the element they want to consider because isotopes of elements have different amounts of neutrons and some elements can exist in more than one state. Element is declared with (atomic number, mass number, charge). The element type is the basic building block provided by the program that can be used to create molecules, compounds, etc. Elements are immutable.

$^{12}_6C$ is represented as: `element C(6, 12, 0);`

$^{14}_6C$ is represented as: `element C(6, 14, 0);`

Molecule

For the purpose of the language, there is no distinction between molecule or compound and both are declared the same way. A molecule is declared as a list of elements surrounded by braces.

NaCl is represented as: `molecule NaCl {[Na, Cl]}`

Equation

Equation is declared in the following way: (list of elements/molecules on left side of reaction, list of elements/molecules on right side of reaction). Underneath, it is essentially, two lists that keep track of the two sides of the equation.

`<equationName>.right` or `<equationName>.left` allows easy access to one side of the equation. Once declared, an equation is immutable.

$NaOH + HCl \rightarrow NaCl + H_2O$ is represented as:


```
equation NaClReaction = {[NaOH, HCl], [NaCl, H2O]};
```

2.1.3 Type Inference

The language is not type-inferred, making it necessary to explicitly declare types.

2.2 Lexical Conventions

2.2.1 Identifiers

An identifier is a sequence of letters or digits in which the first character must be a uppercase letter. Our language is case sensitive, so upper and lower case letters are considered different.

2.2.2 Keywords

The following identifiers start with a lowercase letter and are reserved for use as keywords, and may not be used otherwise:

- | | | |
|------------|------------|---------|
| • int | • equation | • true |
| • double | • if | • false |
| • string | • else | • print |
| • boolean | • while | • call |
| • element | • function | |
| • molecule | • return | |

2.2.3 Literals

Literals are values written in conventional form whose value is obvious. Unlike variables, literals do not change in value. An integer or double literal is a sequence of digits. A boolean literal has two possible values: true or false.

2.2.4 Punctuation

These following characters have their own syntactic and semantic significance and are not considered operators or identifiers.

Punctuator	Use	Example
,	List separator, function parameters	<code>function int sum(int a, int b);</code>
;	Statement end	<code>int x = 3;</code>
"	String declaration	<code>string x = "hello";</code>
[]	List delimiter	<code>int x[] = [1, 2, 3];</code>
{}	Statement list delimiting, and element/molecule/equation declaration	<code>if(expr) { statements }</code>
()	Conditional parameter delimiter, expression precedence	<code>while(i > 2)</code>

2.2.5 Comments

Much like in the C programming language, the characters `/*` introduce a comment, which terminates with the characters `*/`. Single line comments start with `//` and end at the new line character `\n`.

2.2.6 Operators

Operator	Use	Associativity
=	Assignment	Right
==	Test equivalence	Left
!=	Test inequality	Left
>	Greater than	Left
<	Less than	Left
>=	Greater than or equal to	Left
<=	Less than or equal to	Left
&&	AND	Left
	OR	Left
.	Access	Left
*	Multiplication	Left
/	Division	Left
+	Addition	Left
-	Subtraction	Left
^	Concatenate	Left
%	Modulo	Left

The precedence of operators is as follows (from highest to lowest):

1. * / %
2. + -
3. < > <= >=
4. == !=
5. &&
6. ||
7. .
8. ^
9. =

2.3 Syntax

A program in ChemLab consists of at least one function, where one of them is named “main”. Within each function there is a sequence of zero or more valid ChemLab state-

ments.

2.3.1 Expressions

An expression is a sequence of operators and operands that produce a value. Expressions have a type and a value and the operands of expressions must have compatible types. The order of evaluation of subexpressions depends on the precedence of the operators but, the subexpressions themselves are evaluated from left to right.

Constants

Constants can either be of type boolean, string, int, or double.

Identifiers

An identifier can identify a primitive type, non-primitive type, or a function. The type and value of the identifier is determined by its designation. The value of the identifier can change throughout the program, but the value that it can take on is restricted by the type of the identifier. Furthermore, after an identifier is declared, there can be no other identifiers of the same name declared within the scope of the whole program.

```
int x = 3;
x = true; //syntax error
boolean x = 5; //error, x has already been declared
```

Binary Operators

Binary operators can be used in combination with variables and constants in order to create complex expressions. A binary operator is of the form : `<expression> <binary-operator> <expression>`

Arithmetic operators Arithmetic operators include `*`, `/`, `%`, `+`, and `-`. The operands to an arithmetic operator must be numbers. the type of an arithmetic operator expression is either an int or a double and the value is the result of calculating the expression. Note, can not do arithmetic operations when the values involved are a mix of int and double.

`expression * expression`

The binary operator `*` indicates multiplication. It must be performed between two int types or two double types. No other combinations are allowed.

expression / expression

The binary operator / indicates division. The same type considerations as for multiplication apply.

expression % expression

The binary operator % returns the remainder when the first expression is divided by the second expression. Modulo is only defined for int values that have a positive value.

expression + expression

The binary operator + indicates addition and returns the sum of the two expressions. The same type considerations as for multiplication apply.

expression - expression

The binary operator - indicates subtraction and returns the difference of the two expressions. The same type considerations as for multiplication apply.

Relational operators Relational operators include <, >, <=, >=, ==, and !=. The type of a relational operator expression is a boolean and the value is true if the relation is true while it is false if the relation is false.

expression1 > expression2

The overall expression returns true if expression1 is greater than expression 2

expression1 < expression2

The overall expression returns true if expression1 is less than expression 2

expression1 >= expression2

The overall expression returns true if expression1 is greater than or equal to expression 2

expression1 <= expression2

The overall expression returns true if expression1 is less than or equal to expression 2

expression1 == expression2

The overall expression returns true if expression1 is equal to expression 2.

expression1 != expression2

The overall expression returns true if expression1 is not equal to expression 2

Assignment operator The assignment operator (=) assigns whatever is on the right side of the operator to whatever is on the left side of the operator

expression1 = expression2

expression1 now contains the value of expression2

Access operator The access operator is of the form `expression.value`. The expression returns the value associated with the particular parameter. The expression must be of a non-primitive type.

Logical operators Logical operators include AND (`&&`) and OR (`||`). The operands to a logical operator must both be booleans and the result of the expression is also a boolean.

`expression1 && expression2`

The overall expression returns true if and only if `expression1` evaluates to true and `expression2` also evaluates to true.

`expression1 || expression2`

The overall expression returns true as long as `expression1` and `expression2` both do not evaluate to false.

Parenthesized Expression

Any expression surrounded by parentheses has the same type and value as it would without parentheses. The parentheses merely change the precedence in which operators are performed in the expression.

Function Creation

The syntax for declaration of a function is as follows

```
function functionName (type parameter1, type parameter 2, ...) {  
    statements  
}
```

The function keyword signifies that the expression is a function. Parameter declaration is surrounded by parentheses where the individual parameters are separated by commas. All statements in the function must be contained within the curly braces. A good programming practice in ChemLab is to declare all the functions at the beginning of the program so that the functions will definitely be recognized within the main of the program.

Function Call

Calling a function executes the function and blocks program execution until the function is completed. When a function is called, the types of the parameter passed into the function must be the same as those in the function declaration. The way to call a function is

as follows using the Call keyword: `call functionName(param1, param2, etc...)` When a function with parameters is called, the parameters passed into the function are evaluated from left to right and copied by value into the function's scope. `functionName()` if there are no parameters for the function

2.3.2 Statements

A statement in ChemLab does not produce a value and it does not have a type. An expression is not a valid statement in ChemLab.

Selection Statements

A selection statement executes a set of statements based on the value of a specific expression. In ChemLab, the main type of selection statement is the if-else statement. An if-else statement has the following syntax:

```
if( expression){  
  
}else{  
  
}
```

Expression must evaluate to a value of type boolean. If the expression evaluates to true, then the statements within the first set of curly brackets is evaluated. If the expression evaluates to false, then the statements in the curly brackets following else is evaluated. If-else statements can be embedded within each other. Much like in the C programming language, the dangling if-else problem is resolved by assigning the else to the most recent else-less if. Unlike in Java, an if must be followed by an else. A statement with only if is not syntactically correct.

```
if ( ){  
    if ( ){  
  
    }else{  
  
    }  
}else{  
  
}
```

Iteration Statements

ChemLab does not have a for loop unlike most programming languages. The only iteration statement is the while loop. The while statement evaluates an expression before going into the body of the loop. The expression must be of type boolean and the while loop will continue executing so long as the expression evaluates to true. Once the expression evaluates to false, the while loop terminates. The while loop syntax is as follows:

```
while ( expression ) {  
    statements  
}
```

Note that if values in the expression being evaluated are not altered through each iteration of the loop, there is a risk of going into an infinite loop.

Return Statements

A return statement is specified with the keyword return. In a function, the expression that is returned must be of the type that the function has declared. The syntax of a return statement is: return expression;

The return statement will terminate the function it is embedded in or will end the entire program if it is not contained within a function.

2.3.3 Scope

A block is a set of statements that get enclosed by braces. An identifier appearing within a block is only visible within that block. However, if there are two nested blocks, an identifier is recognizable and can be edited within the nested block.

```
function int notRealMethod(int x){  
    int y = 4;  
    while(x>5){  
        while(z>2){  
            y++;  
        }  
    }  
}
```

In this case, y is recognizable within the second while loop and its value will be incremented. One must also note that, functions only have access to those identifiers that are either declared within their body or are passed in as parameters.

2.4 Built-in Functions

Balance Equations

Given an unbalanced equation, this utility will be able to compute the correct coefficients that go in front of each molecule to make it balanced

Molar Mass Calculation

Given a molecule, this utility will be able to compute the total molar mass of the molecule

Naming of Molecules

Given a molecule, the utility will print out the name in correct scientific notation (ex. H_2O will be printed as Dihydrogen Monoxide)

Printing of Equations

Given an equation, the utility will print out the equation in correct scientific notation

Amount of Moles

Given the element and the amount of grams of the element, this utility will return the amount of moles of the element.

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 OCaml yacc 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

See Appendix B.

Chapter 4

Architectural Design

The architectural design of ChemLAB can be divided into the following steps

1. Scanning
2. Parsing
3. Semantic Analysis
4. Java code generation
5. Running the Java code

4.1 Scanning

The ChemLAB scanner tokenizes the input into ChemLAB readable units. This process involves discarding whitespaces and comments. At this stage, illegal character combinations are caught. The scanner was written with ocamllex.

4.2 Parsing and Abstract Syntax Tree

The parser generates an abstract syntax tree based on the tokens that were provided by the scanner. Any syntax errors are caught here. The parser was written with ocamllyacc.

4.3 Semantic Analysis

The semantic analyzer takes in the AST that was generated by the parser and checks the AST for type errors as well as to make sure that statements and expressions are written in a way that corresponds to the syntax defined by the language. A semantically checked AST (SAST) is not generated. If no errors are thrown, then we can assume that it is safe to use the AST to generate Java code.

4.4 Java Generation

The module walks the AST and generates Java code corresponding to the program. All of the code is put into two Java files. One contains graphics and one contains everything else related to the program. The Java code is generated but not compiled. This needs to be done by the ChemLAB script which will run the javac command.

Chapter 5

Test Plan

5.1 Introduction

To ensure that one person's change and updates would not affect the changes others made previously, an automated test was put in place to run through all the tests to make sure everything that worked before still continued to work. Testing was done using a bash shell script to automate the process. The shell script compiles and runs all the test files and compares them with the expected output. Test cases were written to test individual components of the language such as arithmetic, conditional loops, printing, etc.

5.2 Sample Test Cases

Listing 5.1: Hello World test

```
1 /* Test 1: Hello World (comments, print) */  
2  
3 function main() {  
4     print "Hello , world!";  
5 }
```

Listing 5.2: Int and String Variable Assignment

```
1 /* Test 2: int and string variable assignment */  
2  
3 function main() {  
4     int a;  
5     int b;  
6     string s;
```

```

7
8   a = 2;
9   b = 3;
10  s = "ChemLAB";
11
12  print a;
13  print b;
14  print s;
15 }

```

Listing 5.3: Arithmetic test

```

1  /* Test 3: Arithmetic Expressions */
2
3  function main()
4  {
5      print 0;
6      print 1;
7
8      /* Plus, minus, multiply, divide, mod */
9      print 1+1;
10     print 4-1;
11     print 2*2;
12     print 15/3;
13     print 41%7;
14
15     /* Precedence */
16     print 90-6*8;
17
18     /* Parenthesis */
19     print (1+2*3-4)*28/2;
20
21     /* Negative Numbers */
22     // print -3-39;
23     // print 14*-3;
24
25     /* Decimals */
26     // print 2.1;
27     // print 5.0/2.0;
28     // print 42.0/99.0;
29 }

```

Listing 5.4: String Concatenation

```

1  /* Test 4: String concatenation */
2
3  function main()
4  {
5      string a;
6      string b;

```



```

7   string c;
8   a = "Hello";
9   b = "world";
10  c = "!";
11  print a ^ ", " ^ b ^ c;
12  }

```

Listing 5.5: If Condition

```

1  /* Test 5: If Conditional, Boolean */
2
3  function main() {
4      int x;
5      int y;
6      x = 17;
7      y = 42;
8      if (x < y) {
9          print x ^ " is less than " ^ y;
10     } else {
11         print "Test Failed";
12     }
13 }

```

Listing 5.6: Nested If Condition

```

1  /* Test 6: Nested If Else */
2
3  function main()
4  {
5      int x;
6      int y;
7      x = 17;
8      y = 39;
9
10     if (x != y) {
11         y = y + 2;
12
13         if (x > y) {
14             print "Inner If Failed";
15         } else {
16             y = y + 1;
17         }
18     } else {
19         print "Outer If Failed";
20     }
21
22     print y;
23 }

```

Listing 5.7: While Loop

```
1  /* Test 7: While Loop */
2
3  function main()
4  {
5      int a;
6      int b;
7      a = 0;
8      b = 3;
9
10
11     while(a < b)
12     {
13         a = a + 1;
14         print a;
15     }
16 }
```

Listing 5.8: Draw

```
1  function main()
2  {
3      int a;
4      element C(12,13,14);
5      a = C.mass;
6      print a;
7
8  }
```

Listing 5.9: Balance

```
1  function main ()
2  {
3      element C(12,12,12);
4  }
```

Chapter 6

Lessons Learned

6.1 Alice Chang

“Never have I spent so much time on so little code that does so much” adequately sums up my experience this semester in Edwards Programming Languages and Translators class. Indeed, it was a perpetual struggle at first to get the hang of OCaml, which was like no other language I had tackled in the past. Yeah sure, it was essentially java, but upside down and insides out. Initially I entered this class with little knowledge of how a parser or compiler worked. Composing a project proposal knowing so little felt like a clumsy and fruitless attempt to fly when we barely knew how to walk. Yet throughout the course of the semester, I’ve gained much more than knowledge to build a compiler but also skills to work in a team and most importantly the ability to reassure the heart at times of desperation that everything was going to be all right despite the rapidly approaching deadline.

Our team was one man (or woman) short as we had three members. Despite of the slight disadvantage, we learned to view it as a mixed blessing as it was easy to find time to meet up. However as we soon learned, three heads was not always better than one, only when put together did we slowly start to compose our compiler. We experimented with multiple ways of programming: The Lonely All-Nighter in which we all stayed up coding separate codes that worked individually but would not compile as a whole and eventually The Cozy Campfire solution in which one person was primarily in charge of coding and two people gathered around providing feedback. Yet the Cozy Campfire also had its downfalls too lots of ideas being expressed simultaneously and very little progress. Essentially it was like two overly opinionated backseat drivers bickering back and forth while the driver sat in baffled silence. We had so many ideas going on at once that it was often difficult for the programmer to follow so eventually we broke down our ideas into small milestones and accomplished them through a step-by-step procedure.

As we near our presentation date, we've gotten closer as a team and learned to manage our time well, communicate with teammates, and decipher cryptic existing code. Like soldiers in combat, our team suffered through endless out-of-bounds errors and bonded through several panic attacks when GitHub repeatedly crashed on us. Yet at the end of this class we'll have earned our wings to soar through parsers, interpreters, and compilers and wear with us these experiences like a badge of noble achievement—that is, at least until next semester when we take another class that will once again challenge our late-night coding abilities. Yet undoubtedly, the lessons learned through this semester will stay with us beyond this class.

6.2 Martin Ong

The most important lesson I learned from this project is that communication between members of the group is paramount and GitHub can be your friend. Often times, the most difficult problem we encountered was trying to understand the code other members have written and be able to incorporate their code in our own work. At times, the lack of communication led to clashes in our work where a person would change code back to what they thought was working, when in fact they were undoing the work of another person. This was also due to our unfamiliarity with GitHub. Before this class, most of us had only used GitHub for individual projects, so when conflicts came up and we had to merge them, often times the response was to freak out. Resolving these conflicts on GitHub were not easy as changes another person made looked like it didn't belong there to the person resolving the conflict. The mantra then was to “just make it work”, so sometimes progress another person made was disappeared in this way.

If we could do this over again, I would definitely split up the project clearly into concrete slices for each member to take ownership on, such as having one person be in charge of one file. This way one person could keep track of everything that still needs to be done for a particular file. We worked in a non-hierarchical way where we would meet up and code together on the same computer. This led to a decrease in productivity, even though everyone could understand what the code did in the end. Having one less person in our group also put us at a disadvantage, because, even though it made it easier to schedule meetings with each other, each of us had to do much more.

I would also create small milestone deadlines to complete throughout the semester to be more efficient. Since this is probably one of the largest coding projects we have ever done, it did not hit us until Thanksgiving break that there were much more than we had anticipated. I believe that if we put more effort in the beginning to get a good foundation for the ast, scanner and parser, it would be easier working with the other components.

I must say, I have learned a lot about working as a team under a coding environment. We

have definitely learned and changed a lot through this project, both in terms of OCaml and working as a team.

Appendix A

Code Listing

Listing A.1: Abstract Syntax Tree (`ast.ml`)

```
1 type operator = Add | Sub | Mul | Div | Mod
2 type rop = Eq | Neq | Lt | Leq | Gt | Geq
3 type re = And | Or
4 type bool = True | False
5 type data_type = IntType | BooleanType | StringType | DoubleType |
   ElementType | MoleculeType | EquationType
6 type element = Element of string
7 type molecule = Molecule of string
8
9 (*
10 type variable =
11     Var of string
12 *)
13 type expr =
14     Binop of expr * operator * expr
15   | Brela of expr * re * expr
16   | Int of int
17   | String of string
18   | Boolean of expr * rop * expr
19   | Double of float
20   | Asn of string * expr
21   (* | Equation of string * variable list * variable list *)
22   (* | Mass of int
23     | Charge of int
24     | Electrons of int *)
25   | Concat of expr * expr
26   (* | Print of expr *)
27   | List of expr list
28   | Call of string * expr list
29   | Access of expr * string
30   | Bracket of expr
```

```

31 | | Null
32 | | Noexpr
33
34 type stmt =
35 | Block of stmt list
36 | Expr of expr
37 | Return of expr
38 | If of expr * stmt * stmt
39 | For of expr * expr * expr * stmt
40 | While of expr * stmt
41 | Print of expr
42 | Balance of molecule list * molecule list
43 | Draw of string * int * int * int * int * int * int * int * int
44
45 type variable_decl = {
46   vname : string;
47   vtype : data_type;
48 }
49
50 type element_decl = {
51   name : string;
52   mass : int;
53   electrons : int;
54   charge : int;
55 }
56
57 type molecule_decl = {
58   mname : string;
59   elements: element list;
60 }
61
62
63
64
65 type par_decl = {
66   paramname : string; (* Name of the variable *)
67   paramtype : data_type; (* Name of variable type *)
68 }
69
70 type func_decl = {
71   fname : string;
72   formals : par_decl list;
73   locals: variable_decl list;
74   elements : element_decl list;
75   molecules : molecule_decl list;
76   body : stmt list;
77 }
78
79 (* type program = {
80   gdecls : var_decl list;

```

```

81   fdecls : func_decl list
82 }
83 *)
84 type program = func_decl list

```

Listing A.2: Scanner (`scanner.mll`)

```

1 { open Parser }
2
3 let digit = ['0'-'9']
4 let letter = ['A'-'Z' 'a'-'z']
5 let element = ['A'-'Z'] ['a'-'z']?   (* Symbol of element such as: H, Cl *)
6
7 rule token = parse
8   [ ' ' '\t' '\r' '\n' ]           { token lexbuf }
9   | "/*"                           { comment lexbuf }
10  | "//"                             { line_comment lexbuf }
11  | '('                             { LPAREN }
12  | ')'                             { RPAREN }
13  | '['                             { LBRACKET }
14  | ']'                             { RBRACKET }
15  | '{'                             { LCURLY }
16  | '}'                             { RCURLY }
17  | "\""                             { STRINGDECL }
18  | ';'                             { SEMI }
19  | ':'                             { COLON }
20  | ','                             { COMMA }
21  | '.'                             { ACCESS }
22  | '+'                             { PLUS }
23  | '-'                             { MINUS }
24  | '*'                             { TIMES }
25  | '/'                             { DIVIDE }
26  | '%'                             { MOD }
27  | '='                             { ASSIGN }
28  | '^'                             { CONCAT }
29  | "=="                            { EQ }
30  | "!="                            { NEQ }
31  | '<'                             { LT }
32  | "<="                            { LEQ }
33  | '>'                             { GT }
34  | ">="                            { GEQ }
35  | "&&"                            { AND }
36  | "||"                            { OR }
37  | '!'                             { NOT }
38  | "—>"                            { ARROW }
39  | "if"                             { IF }
40  | "else"                           { ELSE }
41  | "while"                          { WHILE }
42  | "for"                             { FOR }
43  | "int"                             { INT }

```



```

44 | "double"           { DOUBLE }
45 | "string"          { STRING }
46 | "boolean"         { BOOLEAN }
47 | "element"         { ELEMENT }
48 | "molecule"       { MOLECULE }
49 | "equation"        { EQUATION }
50 | "balance"         { BALANCE }
51 | "mass" as attr    { ATTRIBUTE(attr) }
52 | "charge" as attr  { ATTRIBUTE(attr) }
53 | "electrons" as attr { ATTRIBUTE(attr) }
54 | "function"        { FUNCTION }
55 | "object"          { OBJECT }
56 | "return"          { RETURN }
57 | "print"           { PRINT }
58 | "call"            { CALL }
59 | "draw"            { DRAW }
60 | "true"            { BOOLEAN_LIT(true) }
61 | "false"           { BOOLEAN_LIT(false) }
62 | (digit)+ '.' (digit)+ as lxm { DOUBLE_LIT(float_of_string
    | lxm) }
63 | digit+ as lxm { INT_LIT(int_of_string lxm) }
64 | ['a'-'z'](letter | digit | '_' )* as lxm { ID(lxm) }
65 | '"' ['\"']* '"' as lxm { STRING_LIT(lxm) }
66 | element as lxm { ELEMENT_LIT(lxm) }
67 | (element ['0'-'9']*)+ as lxm { MOLECULE_LIT(lxm) }
68 | eof { EOF }
69 | - as char { raise (Failure("illegal character " ^
70 | Char.escaped char)) }
71
72
73 and comment = parse
74   "*" { token lexbuf }
75   | - { comment lexbuf }
76
77 and line_comment = parse
78   "\n" { token lexbuf }
79   | - { line_comment lexbuf }

```

Listing A.3: Parser (parser.mly)

```

1 % { open Ast
2   let parse_error s = (* Called by parser on error *)
3     print_endline s;
4     flush stdout
5 %}
6
7 %token SEMI LPAREN RPAREN LBRACKET RBRACKET LCURLY RCURLY COMMA STRINGDECL
8   COLON ACCESS CONCAT NOT OBJECT ARROW
9 %token PLUS MINUS TIMES DIVIDE MOD PRINT ASSIGN
10 %token EQ NEQ LT LEQ GT GEQ EQUAL

```

```

10 %token RETURN IF ELSE FOR WHILE INT DOUBLE STRING BOOLEAN ELEMENT MOLECULE
    EQUATION FUNCTION
11 %token INT DOUBLE STRING BOOLEAN ELEMENT MOLECULE EQUATION FUNCTION
12 %token CALL ACCESS DRAW
13 %token BALANCE MASS CHARGE ELECTRONS
14 %token AND OR
15 %token INT BOOLEAN STRING DOUBLE
16 %token <string> DATATYPE ATTRIBUTE
17 %token <bool> BOOLEAN_LIT
18 %token <string> ELEMENT_LIT
19 %token <string> MOLECULE_LIT
20 %token <string> STRING_LIT
21 %token <string> ID
22 %token <int> INT_LIT
23 %token <float> DOUBLE_LIT
24 %token EOF
25
26
27 %nonassoc NOELSE
28 %nonassoc ELSE
29 %right ASSIGN
30 %left CONCAT
31 %left ACCESS
32 %left OR
33 %left AND
34 %left EQ NEQ
35 %left LT GT LEQ GEQ
36 %left PLUS MINUS
37 %left TIMES DIVIDE MOD
38 %nonassoc LPAREN RPAREN
39
40 %start program
41 %type <Ast.program> program
42
43 %%
44 program :
45     /* nothing */           { [] }
46     | program fdecl         { ($2 :: $1) }
47
48 id :
49     ID                       { $1 }
50     | STRING_LIT             { $1 }
51     | ELEMENT_LIT            { $1 }
52     | MOLECULE_LIT           { $1 }
53
54 element :
55     id                       { Element($1) }
56
57 molecule :
58     id                       { Molecule($1) }

```

```

59
60 vdecl:
61     datatype ID SEMI
62     { { vname = $2;
63         vtype = $1;
64     } }
65
66 vdecl_list:
67     /* nothing */ { [] }
68     | vdecl_list vdecl { ($2::$1) }
69
70 stmt:
71     expr SEMI { Expr($1) }
72     | RETURN expr SEMI { Return($2) }
73     | PRINT expr SEMI { Print($2) }
74     | BALANCE LPAREN molecule_list ARROW molecule_list RPAREN SEMI { Balance
75         ($3, $5) }
76     | DRAW LPAREN STRING_LIT COMMA INT_LIT COMMA INT_LIT COMMA INT_LIT COMMA
77         INT_LIT COMMA INT_LIT COMMA INT_LIT COMMA INT_LIT COMMA INT_LIT RPAREN
78         SEMI { Draw($3, $5, $7, $9, $11, $13, $15, $17, $19) }
79     | LCURLY stmt_list RCURLY { Block(List.rev $2) }
80     | IF LPAREN expr RPAREN stmt %prec NOELSE { If($3, $5, Block
81         ([]) ) }
82     | IF LPAREN expr RPAREN stmt ELSE stmt { If($3, $5, $7) }
83     | FOR LPAREN expr SEMI expr SEMI expr RPAREN stmt { For($3, $5, $7,
84         $9) }
85     | WHILE LPAREN expr RPAREN stmt { While($3, $5) }
86
87 stmt_list:
88     /* nothing */ { [] }
89     | stmt_list stmt { ($2 :: $1) }
90
91 datatype:
92     INT { IntType }
93     | BOOLEAN { BooleanType }
94     | STRING { StringType }
95     | DOUBLE { DoubleType }
96
97 expr:
98     INT_LIT { Int($1) }
99     | id { String($1) }
100     /*| EQUATION id LCURLY element_list ARROW element_list RCURLY { Equation(
101         $2, $4, $6) }*/
102     | expr ACCESS ATTRIBUTE { Access($1, $3) }
103     | expr PLUS expr { Binop($1, Add, $3) }
104     | expr MINUS expr { Binop($1, Sub, $3) }
105     | expr TIMES expr { Binop($1, Mul, $3) }
106     | expr DIVIDE expr { Binop($1, Div, $3) }
107     | expr MOD expr { Binop($1, Mod, $3) }

```

```

103 | expr EQ expr { Boolean($1, Eq, $3) }
104 | expr NEQ expr { Boolean($1, Neq, $3) }
105 | expr LT expr { Boolean($1, Lt, $3) }
106 | expr GT expr { Boolean($1, Gt, $3) }
107 | expr LEQ expr { Boolean($1, Leq, $3) }
108 | expr GEQ expr { Boolean($1, Geq, $3) }
109
110 | expr AND expr { Brela($1, And, $3) }
111 | expr OR expr { Brela($1, Or, $3) }
112 | expr CONCAT expr { Concat($1, $3) }
113 | id ASSIGN expr { Asn($1, $3) }
114 | CALL id LPAREN actuals_opt RPAREN { Call($2, $4) }
115 | LPAREN expr RPAREN { Bracket($2) }
116
117 edecl:
118 ELEMENT id LPAREN INT_LIT COMMA INT_LIT COMMA INT_LIT RPAREN SEMI
119 {{
120     name = $2;
121     mass = $4;
122     electrons = $6;
123     charge = $8
124 }}
125
126 edecl_list:
127 /* nothing */ { [] }
128 | edecl_list edecl { List.rev ($2 :: $1)}
129
130 element_list:
131 element { [$1] }
132 | element_list COMMA element { ($3 :: $1)}
133
134 molecule_list:
135 molecule { [$1] }
136 | molecule_list COMMA molecule { ($3 :: $1)}
137
138
139 mdecl:
140 MOLECULE id LCURLY element_list RCURLY SEMI
141 {{
142     mname = $2;
143     elements = $4;
144 }}
145
146 mdecl_list:
147 /* nothing */ { [] }
148 | mdecl_list mdecl { ($2 :: $1) }
149
150 formals_opt:
151 /* nothing */ { [] }
152 | formal_list { List.rev $1 }

```

```

153
154 formal_list:
155     param_decl          { [$1] }
156   | formal_list COMMA param_decl { $3 :: $1 }
157
158 actuals_opt:
159     /* nothing */        { [] }
160   | actuals_list         { List.rev $1 }
161
162 actuals_list:
163     expr                 { [$1] }
164   | actuals_list COMMA expr { $3 :: $1 }
165
166 param_decl:
167     datatype id
168     { { paramname = $2;
169       paramtype = $1 } }
170
171 fdecl:
172     FUNCTION id LPAREN formals_opt RPAREN LCURLY vdecl_list edecl_list
173       mdecl_list stmt_list RCURLY
174   { {
175     fname = $2;
176     formals = $4;
177     locals = List.rev $7;
178     elements = List.rev $8;
179     molecules = List.rev $9;
180     body = List.rev $10
181   } }

```

Listing A.4: Semantic Checker (`semantic.ml`)

```

1  open Ast
2  open Str
3
4  type env = {
5      mutable functions : func_decl list;
6  }
7
8  let function_equal_name name = function
9      func -> func.fname = name
10
11 let function_fparam_name name = function
12     par -> par.paramname = name
13
14 let function_var_name name = function
15     variable -> variable.vname = name
16
17 (* Checks whether a function has been defined duplicately *)
18 let function_exist func env =

```

```

19   let name = func.fname in
20   try
21     let _ = List.find (function_equal_name name) env.functions in
22     let e = "Duplicate function: " ^ name ^ " has been defined more than
23         once" in
24         raise (Failure e)
25   with Not_found -> false
26
27   (*Checks if function has been declared*)
28   let exist_function_name name env = List.exists (function_equal_name name) env
29       .functions
30
31   let get_function_by_name name env =
32     try
33       let result = List.find (function_equal_name name) env.functions in
34       result
35     with Not_found -> raise (Failure ("Function " ^ name ^ " has not been declared
36         !"))
37
38   let get_formal_by_name name func =
39     try
40       let result = List.find (function_fparam_name name) func.formals in
41       result
42     with Not_found -> raise (Failure ("Formal Param" ^ name ^ " has not been
43         declared!"))
44
45   let get_variable_by_name name func =
46     try
47       let result = List.find (function_var_name name) func.locals in
48       result
49     with Not_found -> raise (Failure ("Local Variable " ^ name ^ " has not been
50         declared!"))
51
52   let count_function_params func = function
53     a -> let f count b =
54       if b = a
55       then count+1
56       else count
57   in
58   let count = List.fold_left f 0 func.formals in
59   if count > 0
60   then raise (Failure ("Duplicate parameter in function " ^ func.fname))
61   else count
62
63   let count_function_variables func = function

```

```

64 | a -> let f count b =
65 |   if b = a
66 |     then count+1
67 |     else count
68 | in
69 |   let count = List.fold_left f 0 func.locals in
70 |     if count > 0
71 |       then raise (Failure("Duplicate variable in function " ^ func.fname))
72 |       else count
73 |
74 | (*Determines if a formal paramter with the given name  fname  exits in
   | the given function*)
75 |
76 | let exists_formal_param func fname =
77 | try
78 |   List.exists (function_fparam_name fname) func.formals
79 | with Not_found -> raise (Failure ("Formal Parameter " ^ fname ^ " should
   | exist but was not found in function " ^ func.fname))
80 |
81 |
82 | (*Determines if a variable declaration with the given name  vname  exists
   | in the given function*)
83 |
84 | let exists_variable_decl func vname =
85 | try
86 |   List.exists (function_var_name vname) func.locals
87 | with Not_found -> raise (Failure ("Variable " ^ vname ^ " should exist but
   | was not found in function " ^ func.fname))
88 |
89 |
90 |
91 |
92 | let dup_param_name func fname =
93 |   let name = func.formals in
94 |   try
95 |     List.find (function name -> name.paramname = fname.paramname ) name
96 |   with Not_found -> raise (Failure ("Duplicate param names"))
97 |
98 |
99 |
100 | let get_fparam_type func fname =
101 |   let name = func.formals in
102 |   try
103 |     let fparam = List.find(function_fparam_name fname) name in
104 |     fparam.paramtype
105 |   with Not_found -> raise (Failure ("Formal param should exist but not
   | found"))
106 |
107 |
108 | (*given variable name, get type*)

```

```

109 let get_var_type func vname =
110   let name = func.locals in
111   try
112     let var = List.find(function_var_name vname) name in
113     var.vtype
114   with Not_found -> raise (Failure ("Variable should exist but not found"))
115
116 (*
117 let param_exist func =
118   let name = func.formals in
119   try
120     let _ = List.iter (fun f -> List.find (exists_formal_param func f) ) name
121     in
122     let e = "Duplicate param: " ^ name ^ "has been defined more than once" in
123     raise (Failure e)
124   with Not_found -> false
125
126 let get_fparam_type func fpname =
127   try
128     let fparam =
129 *)
130 (*Determines if the given identifier exists*)
131 let exists_id name func = (exists_variable_decl func name) || (
132   exists_formal_param func name)
133
134 (*see if there is a function with given name*)
135 let find_function func env =
136   try
137     let _ = List.find (function_equal_name func) env.functions in
138     true (*return true on success*)
139   with Not_found -> raise Not_found
140
141 let is_int s =
142   try ignore (int_of_string s); true
143   with _ -> false
144
145 let is_float s =
146   try ignore (float_of_string s); true
147   with _ -> false
148
149 let is_letter s = string_match (regexp "[A-Za-z]") s 0
150
151 let is_string s = string_match (regexp "\\\".*\\\"") s 0
152
153 let is_string_bool = function "true" -> true | "false" -> true | _ -> false
154
155 let rec is_num func = function
156   Int(_) -> true
157   | Double(_) -> true

```



```

157 | Binop(e1, -, e2) -> (is_num func e1) && (is_num func e2)
158 | _ -> false
159
160 let rec is_boolean func = function
161   Boolean(_) -> true
162   | _ -> false
163
164 (*check if variable declation is valid*)
165
166 (*
167
168 let valid_vdecl func =
169   let _ = List.map (function func.locals) ->
170   let e = "Invalid variable declaration for " ^ nm ^ "' in compute function
171   " ^ func.fname ^ "\n" in
172   let be = e ^ "The only allowed values for initializing boolean
173   variables are 'true' and 'false.'" ^ "\n" in
174   match vtype with
175   | "Int" -> if is_string value then true else raise (Failure e)
176   | "Double" -> if is_float value then true else raise (Failure e)
177   | "String" -> if is_int value then true else raise (Failure e)
178   | "Boolean" -> if is_string_bool value then true else raise (
179   Failure be)) func.locals
180   in
181   true
182 *)
183
184 let rec get_expr_type e func =
185   match e with
186   | String(s) -> StringType
187   | Int(s) -> IntType
188   | Double(f) -> DoubleType
189   | Boolean(_, -, _) -> BooleanType
190   | Binop(e1, -, e2) -> get_expr_type e1 func
191   | Brela(e1, -, e2) -> BooleanType
192   | Asn(expr, expr2) -> get_expr_type expr2 func
193   (* | Equation(s, vlist, vlist2) -> EquationType *)
194   | Concat(s, s2) -> StringType
195   | Bracket(e1) -> get_expr_type e1 func
196   | Access(id, attr) -> IntType (* Call only returns mass, charge, or
197   electrons *)
198   | _ -> raise( Failure("!!! Need to implement in get_expr_type !!!") )
199
200 let rec valid_expr (func : Ast.func_decl) expr env =
201   match expr with
202   | Int(_) -> true
203   | Double(_) -> true
204   | Boolean(_) -> true

```

```

202 | String(-) -> true
203 | Binop(e1,-,e2) -> let t1 = get_expr_type e1 func and t2 = get_expr_type
    e2 func in
204   begin
205     match t1, t2 with
206     | DoubleType, DoubleType -> true
207     | IntType, IntType -> true
208     | -, - -> raise (Failure "Types for binary expresion must be matching
    int or double")
209   end
210 | Brela (e1,-,e2) -> let t1 = get_expr_type e1 func and t2 = get_expr_type
    e2 func in
211   begin
212     match t1, t2 with
213     | BooleanType, BooleanType -> true
214     | -, - -> raise (Failure "Invalid type for AND, OR expression")
215   end
216 | Asn(id, expr2) ->
217   begin
218     let t1 = get_var_type func id and t2 = get_expr_type expr2 func in
219     match t1,t2 with
220     | StringType, StringType -> true
221     | IntType, IntType -> true
222     | DoubleType, DoubleType -> true
223     | ElementType, ElementType -> true
224     | MoleculeType, MoleculeType -> true
225     | EquationType, EquationType -> true
226     | -, - -> raise(Failure ("DataTypes do not match up in an assignment
    expression to variable "))
227   end
228 | Concat(e1,e2) ->
229   begin
230     match get_expr_type e1 func, get_expr_type e2 func with
231     | StringType, StringType -> true
232     | -, - -> raise(Failure("Concatenation only works between two strings"
    ))
233   end
234 | Call(f_name,-) -> exist_function_name f_name env
235 | List(e_list) -> let _ = List.map (fun e -> valid_expr func e env) e_list
    in true
236 | - -> raise( Failure("!!! Need to implement in valid_expr !!!") )
237
238 (*Print(e1) ->
239   let t1 = get_expr_type expr func in
240   match t1 with
241   | "String" -> true
242   | "int" -> true
243   | "double" -> true
244   | "boolean" -> true
245   | "element" -> true

```

```

246 |     "molecule" -> true
247 |     "equation" -> true
248 |     _ -> raise(Failure("Can't print type"))*)
249
250
251
252 let has_return_stmt list =
253   if List.length list = 0
254   then false
255   else match (List.hd (List.rev list)) with
256     Return(_) -> true
257     | _ -> false
258
259
260 (* let if_else_has_return_stmt stmt_list =
261   let if_stmts = List.filter (function If(_,_,_) -> true | _ -> false)
262     stmt_list in
263   let rets = List.map (
264     function
265       If(_,s1,s2) ->
266         begin
267           match s1,s2 with
268             Block(lst1),Block(lst2) -> (has_return_stmt lst1) && (
269               has_return_stmt lst2)
270             | _ -> raise(Failure("Error"))
271         end
272       | _ -> false
273     ) if_stmts in
274   List.fold_left (fun b v -> b || v) false rets *)
275
276 let has_return_stmt func =
277   let stmt_list = func.body in
278   if List.length stmt_list = 0
279   then false
280   else match List.hd (List.rev stmt_list), func.fname with
281     Return(e),"main" -> raise(Failure("Return statement not permitted in
282       main method"))
283     | _, "main" -> false
284     | Return(e), _ -> true
285     | _, _ -> false
286
287 (*Returns the type of a given variable name *)
288 let get_type func name =
289   if exists_variable_decl func name (* True if there exists a var of that
290     name *)
291   then get_var_type func name
292   else
293     if exists_formal_param func name
294     then get_fparam_type func name

```

```

292     else (*Variable has not been declared as it was not found*)
293         let e = "Variable \" ^ name ^ "\" is being used without being
                declared in function \" ^ func.fname ^ "\" in
294         raise (Failure e)
295
296
297 (* Check that the body is valid *)
298 let valid_body func env =
299     (* Check all statements in a block recursively, will throw error for an
        invalid stmt *)
300     let rec check_stmt = function
301         Block(stmt_list) -> let _ = List.map(func s -> check_stmt s) stmt_list
                in
302         true
303     | Expr(expr) -> let _ = valid_expr func expr env in
        true
304     | Return(expr) -> let _ = valid_expr func expr env in
        true
305     | If(condition, then_stmts, else_stmts) -> let cond_type = get_expr_type
        condition func in
306     begin
307         match cond_type with
308         BooleanType ->
309             if (check_stmt then_stmts) && (check_stmt else_stmts)
310             then true
311             else raise( Failure("Invalid statements in If statement
                within function \" ^ func.fname ^ "\""))
312         | _ -> raise( Failure("Condition of If statement is not a valid
                boolean expression within function \" ^ func.fname ^ "\"") )
313     end
314
315 end
316 | For(init, condition, do_expr, stmts) -> let cond_type = get_expr_type
        condition func in
317     let _ = valid_expr func do_expr env in
318     let _ = valid_expr func init env in
319     begin
320         match cond_type with
321         BooleanType ->
322             if check_stmt stmts
323             then true
324             else raise( Failure("Invalid statements in For loop
                within function \" ^ func.fname ^ "\""))
325         | _ -> raise( Failure("Condition of For loop is not a valid
                boolean expression within function \" ^ func.fname ^ "\"")
                )
326     end
327
328 | While(condition, stmts) -> let cond_type = get_expr_type condition func
        in
329     begin
330         match cond_type with
        BooleanType ->

```

```

331         if check_stmt stmts
332         then true
333         else raise( Failure("Invalid statments in While loop within
334             function \"\" ^ func.fname ^ "\"") )
335     | -> raise( Failure("Condition of While loop is not a valid
336         boolean expression within function \"\" ^ func.fname ^ "\"") )
337 end
338 | Print(expr) -> let expr_type = get_expr_type expr func in
339     begin
340         match expr_type with
341         | StringType -> true
342         | IntType -> true
343         | -> raise( Failure("Print in function \"\" ^ func.fname ^ "\"
344             does not match string type") )
345     end
346 in
347     let _ = List.map(fun s -> check_stmt s) func.body in
348     true
349
350 let valid_func env f =
351     let duplicate_functions = function_exist f env in
352     (* let duplicate_parameters = count_function_params f in *)
353     let v_body = valid_body f env in
354     let _ = env.functions <- f :: env.functions (* Adding function to
355         environment *) in
356     (not duplicate_functions) && (* (not duplicate_parameters) && *)
357     v_body
358
359 let check_program flist =
360     let (environment : env) = { functions = [] (* ; variables = [] *) } in
361     let _validate = List.map ( fun f -> valid_func environment f ) flist in
362     (* let _ = print_endline "\nSemantic analysis completed successfully.\n
363         nCompiling...\n" in *)
364     true

```

Listing A.5: Compiler, Code Generation (compile.ml)

```

1 open Ast
2 open Str
3 open Printf
4 open Parser
5 open Helper
6 module StringMap = Map.Make(String);;
7
8 let string_of_type = function
9     IntType -> "int"
10    | BooleanType -> "Boolean"
11    | StringType -> "String"
12    | DoubleType -> "double"
13    | _ -> ""

```

```

14
15 let string_of_op = function
16   Add -> "+"
17   | Sub -> "-"
18   | Mul -> "*"
19   | Div -> "/"
20   | Mod -> "%"
21
22 let string_of_rop = function
23   Gt -> ">"
24   | Geq -> ">="
25   | Lt -> "<"
26   | Leq -> "<="
27   | Eq -> "=="
28   | Neq -> "!="
29
30 let string_of_re = function
31   And -> "&&"
32   | Or -> "||"
33
34 let string_of_boolean = function
35   True -> string_of_bool true
36   | False -> string_of_bool false
37
38 let string_of_element = function
39   Element(e)-> e
40
41 let string_of_molecule = function
42   Molecule(m)-> m
43
44 let string_of_mdecl_balance mdecl = mdecl.mname
45
46 let rec string_of_expr = function
47   Int(i) -> string_of_int i
48   | Double(d) -> string_of_float d
49   | Boolean(e1, rop, e2) -> string_of_expr e1 ^ string_of_rop rop ^
      string_of_expr e2
50   | String (s) -> s
51   | Asn(id, left) -> id ^ " = " ^ (string_of_expr left)
52   (* | Seq(s1, s2) -> (string_of_expr s1) ^ " ; " ^ (string_of_expr s2) *)
53   | Call(s,l) -> s ^ "(" ^ String.concat "" (List.map string_of_expr l) ^ ")"
54   | Access(o,m) -> (string_of_expr o) ^ "." ^ m ^ "(" ^ ")"
55   | Binop (e1, op, e2) ->
      (string_of_expr e1) ^ " " ^ (string_of_op op)
      ^ " " ^ (string_of_expr e2)
56   | Brela (e1, op, e2) ->
      (string_of_expr e1) ^ " " ^ (string_of_re op)
      ^ " " ^ (string_of_expr e2)
57   | Noexpr -> ""
58   | Null -> "NULL"
59
60
61
62

```

```

63 | Concat(s1, s2) -> string_of_expr s1 ^ "+" ^ string_of_expr s2
64 | List(elist) -> "[" ^ String.concat ", " (List.map string_of_expr elist
    ) ^ "]"
65 (* | Print(s) -> "System.out.println(" ^ string_of_expr s ^ ");" *)
66 (* | Equation(name, rlist, plist) -> "equation " ^ name ^ "{" ^ String.
    concat ", " (List.map string_of_var rlist) ^ "--" ^ String.concat ", "
    (List.map string_of_var plist) ^ "}" *)
67 (* | Mass(num) -> string_of_int num ^ ".Mass()"
68 | Charge(num) -> string_of_int num ^ ".Charge()"
69 | Electrons(num) -> string_of_int num ^ ".Electrons()"
70 *) | Bracket(e) -> "(" ^ string_of_expr e ^ ")"
71
72 let string_of_edekl edekl = "Element " ^ edekl.name ^ "= new Element(" ^ (
    string_of_int edekl.mass) ^ ", " ^ (string_of_int edekl.electrons) ^ ", " ^
    (string_of_int edekl.charge) ^ ");\n"
73 let string_of_mdecl mdecl = "ArrayList<Element> " ^ mdecl.mname ^ "1 = new
    ArrayList<Element>(Arrays.asList(" ^ String.concat ", " (List.map
    string_of_element mdecl.elements) ^ "));\n" ^
74 "Molecule " ^ mdecl.mname ^ "= new Molecule(" ^ mdecl.mname ^ "1);\n"
75
76 let string_of_pdecl pdecl = string_of_type pdecl.paramtype ^ " " ^ pdecl.
    paramname
77 let string_of_pdecl_list pdecl_list = String.concat "" (List.map
    string_of_pdecl pdecl_list)
78 let string_of_vdecl vdecl = string_of_type vdecl.vtype ^ " " ^ vdecl.vname ^
    ";\n"
79
80 let rec string_of_stmt = function
81   Block(stmts) ->
82     "{\n" ^ String.concat "" (List.map string_of_stmt stmts) ^ "}\n"
83 | Expr(expr) -> string_of_expr expr ^ ";\n"
84 | Return(expr) -> "return " ^ string_of_expr expr ^ ";\n"
85 | If(e, s1, s2) -> "if (" ^ string_of_expr e ^ ")\n" ^ (string_of_stmt s1
    ) ^ "\n" ^ "else\n" ^ (string_of_stmt s2) ^ "\n"
86 (* | If(e, s, Block([])) -> "if (" ^ string_of_expr e ^ ")\n" ^ (
    string_of_stmt s) ^ "\n" *)
87 | For(e1, e2, e3, s) ->
88   "for (" ^ string_of_expr e1 ^ " ; " ^ string_of_expr e2 ^ " ; " ^
89   string_of_expr e3 ^ ") " ^ string_of_stmt s ^ "\n"
90 | While(e, s) -> "while (" ^ string_of_expr e ^ ") {" ^ (string_of_stmt s
    ) ^ "}\n"
91 | Print(s) -> "System.out.println(" ^ string_of_expr s ^ ");\n"
92 | Draw(s, e1, e2, e3, e4, e5, e6, e7, e8) -> "randx = (int) (Math.random
    ()*400); randy = (int) (Math.random()*400); scene.add(new AtomShape(
    randx, randy, " ^ s ^ ", " ^
93   (string_of_int e1) ^ ", " ^
94   (string_of_int e2) ^ ", " ^
95   (string_of_int e3) ^ ", " ^
96   (string_of_int e4) ^ ", " ^
97   (string_of_int e5) ^ ", " ^

```

```

98     (string_of_int e6) ^ "," ^
99     (string_of_int e7) ^ "," ^
100    (string_of_int e8) ^ "));";
101    | Balance(llist, rlist) -> "Balance(\"" ^ String.concat " , " (List.map
    string_of_molecule llist) ^ " == " ^ String.concat " , " (List.map
    string_of_molecule rlist) ^ "\");";
102
103
104
105    let string_of_vdecl vdecl =
106        string_of_type vdecl.vtype ^ " " ^ vdecl.vname ^ ";";
107
108    let string_of_fdecl fdecl =
109        if fdecl.fname = "main" then "public static void main(String args[])\n{\n
    " ^
110        String.concat "" (List.map string_of_vdecl fdecl.locals) ^
111        String.concat "" (List.map string_of_eddecl fdecl.elements) ^
112        String.concat "" (List.map string_of_mdecl fdecl.molecules) ^
113        String.concat "" (List.map string_of_stmt fdecl.body) ^
114        "}\n"
115    else
116        "public static void " ^ fdecl.fname ^ "(" ^ String.concat " , " (List.map
    string_of_pdecl fdecl.formals) ^ ")\n{\n" ^
117        String.concat "" (List.map string_of_vdecl fdecl.locals) ^
118        String.concat "" (List.map string_of_eddecl fdecl.elements) ^
119        String.concat "" (List.map string_of_mdecl fdecl.molecules) ^
120        String.concat "" (List.map string_of_stmt fdecl.body) ^
121        "}\n"
122
123    let string_of_fdecl_list fdecl_list =
124        String.concat "" (List.map string_of_fdecl fdecl_list)
125
126    let string_of_program (vars, funcs) =
127        String.concat "" (List.map string_of_vdecl (List.rev vars)) ^ "\n" ^
128        String.concat "\n" (List.map string_of_fdecl (List.rev funcs)) ^ "\n"
129
130
131
132
133    (* match nth molecule.elements 0 with
134    | [] -> 0
135    | hd :: tl -> hd.mass + mass_sum tl;; *)
136
137
138    let rec charge_sum molecule = match molecule with
139    | [] -> 0
140    | hd :: tl -> hd.charge + charge_sum tl;;
141
142
143    let contains s1 s2 =

```



```

144     let re = Str.regexp_string s2
145     in
146         try ignore (Str.search_forward re s1 0); true
147         with Not_found -> false
148
149 let program program prog_name =
150     let graphic_boolean a b =
151         if (contains (string_of_fdecl_list program) "graphics") then a else b
152         in
153         let prog_string = Helper.balance_head ^ prog_name ^ Helper.balance_mid ^
154             (graphic_boolean "public static final SceneComponent scene = new
155             SceneComponent();" "") ^ Helper.balance_mid1 ^ prog_name ^ Helper.
156             balance_mid15 ^ Helper.balance_mid2 ^ (string_of_fdecl_list program)
157             ^ Helper.balance_end in
158         let out_chan = open_out (Printf.sprintf "%s.java" prog_name) in
159         ignore(Printf.fprintf out_chan "%s" prog_string);
160         close_out out_chan;
161         ignore(Sys.command(Printf.sprintf "javac %s.java" prog_name));
162         ignore(Sys.command(Printf.sprintf "java %s" prog_name));
163
164 if (contains (string_of_fdecl_list program) "graphics") then
165     let graphics_string = Helper.balance_head ^ "ChemGRAPH extends JFrame" ^
166         Helper.balance_mid ^ "public static final SceneComponent scene = new
167         SceneComponent();" ^ Helper.balance_mid1 ^ "ChemGRAPH" ^ Helper.
168         balance_mid15 ^ "setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
169         setSize(500, 500); add(scene, BorderLayout.CENTER);" ^ Helper.
170         balance_mid2 ^ (string_of_fdecl_list program) ^ Helper.balance_end in
171     let out_chan = open_out ("ChemGRAPH.java") in
172     ignore(Printf.fprintf out_chan "%s" graphics_string); close_out
173         out_chan;
174     (ignore(Sys.command ("javac ChemGRAPH.java SceneEditor.java")));
175 if (contains (string_of_fdecl_list program) "graphics") then ignore(Sys.
176     command("java SceneEditor"));

```

Listing A.6: Top-level Executable (chemlab.ml)

```

1 exception NoInputFile
2 exception InvalidProgram
3
4 let usage = Printf.sprintf "Usage: chemlab FILE_NAME"
5 (* Get the name of the program from the file name. *)
6 let get_prog_name source_file_path =
7     let split_path = (Str.split (Str.regexp_string "/") source_file_path) in
8     let file_name = List.nth split_path ((List.length split_path) - 1) in
9     let split_name = (Str.split (Str.regexp_string ".") file_name) in
10     List.nth split_name ((List.length split_name) - 2)
11
12 (* Entry Point: starts here *)
13 let _ =
14     try

```

```

15   let prog_name =
16       if Array.length Sys.argv > 1 then
17           get_prog_name Sys.argv.(1)
18       else raise NoInputFile in
19
20   let input_channel = open_in Sys.argv.(1) in
21
22   let lexbuf = Lexing.from_channel input_channel in
23   let prog = Parser.program Scanner.token lexbuf in
24       (* if Semantic.check_program prog *)
25       (* then *) Compile.program prog prog_name
26       (* else raise InvalidProgram *)
27   with
28   | NoInputFile -> ignore(Printf.printf "Please provide a name for a
      ChemLAB file.\n");exit 1
29   | InvalidProgram -> ignore(Printf.printf "Invalid program. Semantic
      errors exist.\n");exit 1

```

language=bash]../test.sh

Appendix B

Project Log

projectlog.txt

```
1 commit 2a40d291f771f870f403f17f44c908514ac7aa95
2 Author: Martin Ong <mo2454@columbia.edu>
3 Date: Tue Dec 16 22:58:11 2014 -0500
4
5     Update test cases
6
7 commit a19a452c8dd8d4606178e73f5adcb52cde682c31
8 Author: Martin Ong <mo2454@columbia.edu>
9 Date: Tue Dec 16 22:57:57 2014 -0500
10
11     Changed a lot of stuff
12
13     Tests changed
14
15 commit e6f2c1d800afa90c5ff2ca2003b351249bbd453f
16 Author: Martin Ong <mo2454@columbia.edu>
17 Date: Mon Dec 15 21:23:52 2014 -0500
18
19     Outputs error when it doesn't match file for test
20
21 commit d4913a22046441b46dca2c74e2897f26cfc0ed9c
22 Author: Martin Ong <mo2454@columbia.edu>
23 Date: Mon Dec 15 21:08:58 2014 -0500
24
25     Delete test7
26
27 commit 1bf6acc7a9444876c63dbb1fc67afda06f040ade
28 Merge: df2360e 63ff646
29 Author: Martin Ong <mo2454@columbia.edu>
30 Date: Mon Dec 15 21:08:38 2014 -0500
31
```

```

32 Merge branch 'Semantic-2'
33
34 Conflicts:
35     ast.mli
36     compile.ml
37     parser.mly
38     scanner.mll
39     semantic.ml
40     test/test6.chem
41
42 commit 63ff64621063684e4581b22aedd8703d1927a8f
43 Author: Martin Ong <mo2454@columbia.edu>
44 Date: Mon Dec 15 21:02:05 2014 -0500
45
46     Fix test cases, no parse errors
47
48 commit e0a6035f24b233601584b89a2a383fec59e23d8
49 Author: Martin Ong <mo2454@columbia.edu>
50 Date: Mon Dec 15 20:58:01 2014 -0500
51
52     Added single line comments
53
54 commit 7046b6e22e5fdb85dc36291b1ff8867cf8c224c6
55 Author: Martin Ong <mo2454@columbia.edu>
56 Date: Mon Dec 15 20:43:02 2014 -0500
57
58     Removed duplicate file
59
60 commit e108acbc8e2cd7427f4c774fa42dd97e90894f21
61 Author: Martin Ong <mo2454@columbia.edu>
62 Date: Mon Dec 15 20:41:44 2014 -0500
63
64     Updated make clean
65
66 commit 729e19f4b38c289bd9be713adc0a063de25e7c40
67 Author: Martin Ong <mo2454@columbia.edu>
68 Date: Mon Dec 15 20:35:03 2014 -0500
69
70     Updated test script
71
72 commit 025c95cbe8191abae9cf8cd45254a4ac06dcf683
73 Author: Martin Ong <mo2454@columbia.edu>
74 Date: Mon Dec 15 19:02:46 2014 -0500
75
76     Update parser, scanner, ast to include mod, concat
77
78 commit b2f53f3d6cde3a5a68aebbf5f3d585452f14d80
79 Author: Martin Ong <mo2454@columbia.edu>
80 Date: Mon Dec 15 19:01:44 2014 -0500
81

```

```

82      Check tests against expected
83
84      Put expected outputs in folder exp
85
86      commit 14d6922b3a0699b4eaf8aabdb77180333c751350
87      Author: Martin Ong <mo2454@columbia.edu>
88      Date:   Mon Dec 15 17:48:57 2014 -0500
89
90      Updated all test files
91
92      commit 222606b774b902e111ca0ce5f7829d981d048a90
93      Author: Martin Ong <mo2454@columbia.edu>
94      Date:   Mon Dec 15 17:48:32 2014 -0500
95
96      Updated all test files
97
98      commit df2360ecf133f33537d4c7b9cbf202f1eb281846
99      Author: Martin Ong <martinong@users.noreply.github.com>
100     Date:   Mon Dec 15 14:43:13 2014 -0500
101
102     Update README.md
103
104     commit 6598a0242fa8b231aa1f0a2a3089a039aa426339
105     Author: Martin Ong <mo2454@columbia.edu>
106     Date:   Sun Dec 14 23:43:06 2014 -0500
107
108     Formatted code listings
109
110     commit 9f9678207a2e467ab461a0361773734ea4481c7c
111     Author: Martin Ong <mo2454@columbia.edu>
112     Date:   Sun Dec 14 23:16:05 2014 -0500
113
114     Added image in
115
116     commit 366b2fe0e0d3c183e2719837986a8edcc3bd7d5d
117     Author: Martin Ong <mo2454@columbia.edu>
118     Date:   Sun Dec 14 23:10:07 2014 -0500
119
120     Restructured file structure
121
122     commit 70b301b99934cbc86ccaf65fc229c86112b8a173
123     Author: Martin Ong <mo2454@columbia.edu>
124     Date:   Sun Dec 14 23:07:28 2014 -0500
125
126     Initial Commit for Final Report in Latex
127
128     commit ac5847f846e619b2a533f3233d146b1a673d1afb
129     Author: detectiveconan2 <ggl2110@columbia.edu>
130     Date:   Sun Dec 14 23:05:00 2014 -0500
131

```

```

132     picture
133
134 commit e58efc348f0952ac3e2aef1e2278e609f41667db
135 Author: detectiveconan2 <ggl2110@columbia.edu>
136 Date:   Sun Dec 14 23:03:49 2014 -0500
137
138     final paper
139
140 commit f40dbf7d0d44c68ce772ef004c371b335dfb6bf3
141 Author: detectiveconan2 <ggl2110@columbia.edu>
142 Date:   Sun Dec 14 22:18:27 2014 -0500
143
144     final paper
145
146 commit 9f69d7e7493e8c693921b39124280b94c7dbce56
147 Author: detectiveconan2 <ggl2110@columbia.edu>
148 Date:   Sun Dec 14 21:33:55 2014 -0500
149
150     final paper parts
151
152 commit 3b65b7b4142d1f4243c74a23c8968d991e25fbb4
153 Author: Alice Chang <avc2120@columbia.edu>
154 Date:   Sun Dec 14 15:34:36 2014 -0500
155
156     deleted contents of ChemLAB.java
157
158 commit 17f9a2020385a6966bcb8367f22c0e1aa75abd2f
159 Merge: 299e376 1c2d5c9
160 Author: Alice Chang <avc2120@columbia.edu>
161 Date:   Sun Dec 14 13:15:16 2014 -0500
162
163     Merge branch 'Alice--2'
164
165     Conflicts:
166         compile.ml
167         parser.mly
168
169 commit 299e37699aec217bffacfe28b89e89a383c105a1
170 Merge: 0f76289 f8168c2
171 Author: Martin Ong <martinong@users.noreply.github.com>
172 Date:   Sun Dec 14 11:28:08 2014 -0500
173
174     Merge pull request #5 from martinong/Semantic-2
175
176     Major Debug
177
178 commit 0f76289e7697f8ff8fbe95d88b3a0bfc0bd95e7b
179 Author: Martin Ong <mo2454@columbia.edu>
180 Date:   Sun Dec 14 11:24:45 2014 -0500
181

```

```

182 Merge branch 'Semantic-2'
183
184 commit f8168c2443120ea304a5c48f163fce9e295251a8
185 Author: Martin Ong <mo2454@columbia.edu>
186 Date: Sun Dec 14 02:02:05 2014 -0500
187
188 Major Debug
189
190 If, For, While loops fix
191 Concat works
192
193 commit 6d4cc1a94356b409ed1c803f4ad6f1dc2df2f05c
194 Merge: 141c700 c6c3293
195 Author: Martin Ong <mo2454@columbia.edu>
196 Date: Sun Dec 14 01:06:23 2014 -0500
197
198 Merge branch 'Martin-Semantic'
199
200 Conflicts:
201 ChemLAB.class
202 chemlab.ml
203 compile.ml
204 semantic.ml
205
206 commit c6c329373c317929593db4c5fa44a7492e83082e
207 Author: Martin Ong <mo2454@columbia.edu>
208 Date: Sun Dec 14 00:56:34 2014 -0500
209
210 Remove Element, Molecule and Equation types
211
212 commit 2f8a8086f101c5a85d3f4abc8331598471cdc264
213 Author: Martin Ong <mo2454@columbia.edu>
214 Date: Sun Dec 14 00:50:28 2014 -0500
215
216 Add string_of_type to return the java string
217
218 commit c4863f980a81d9dd7befb04fce2fbab3de909533
219 Author: Martin Ong <mo2454@columbia.edu>
220 Date: Sun Dec 14 00:41:14 2014 -0500
221
222 Change type from a string to "data_type"
223
224 commit f70c574f9a82028852aa8e83b7e50e7aa53cc755
225 Author: Martin Ong <mo2454@columbia.edu>
226 Date: Sun Dec 14 00:11:23 2014 -0500
227
228 Semantic check compiles
229
230 Checks for valid body. Still buggy (Maybe problems with get_expr_type)
231

```

```

232 | commit fe0c5a310de9125a27f4e109619209190d1f3403
233 | Author: Martin Ong <mo2454@columbia.edu>
234 | Date: Sun Dec 14 00:10:18 2014 -0500
235 |
236 | Update spacing of test file
237 |
238 | commit 1c2d5c9f611fa7ca532d4f6d99f32acdd724fa66
239 | Author: Alice Chang <avc2120@columbia.edu>
240 | Date: Sat Dec 13 21:37:42 2014 -0500
241 |
242 | works
243 |
244 | commit e4a11cc5944a5a0d2da73b44351c92c168403e71
245 | Author: Alice Chang <avc2120@columbia.edu>
246 | Date: Sat Dec 13 18:55:23 2014 -0500
247 |
248 | readme edited
249 |
250 | commit 7b50d47da002ff48c9ba56245d3cd074fe8858a6
251 | Author: Alice Chang <avc2120@columbia.edu>
252 | Date: Sat Dec 13 18:51:45 2014 -0500
253 |
254 | cleaned up
255 |
256 | commit 430fb2ea4d8aa1f27e5e9774504b636e8c6d71fb
257 | Merge: 5de8527 46b797c
258 | Author: detectiveconan2 <ggl2110@columbia.edu>
259 | Date: Sat Dec 13 18:35:51 2014 -0500
260 |
261 | Merge remote-tracking branch 'origin/Martin-Semantic' into Martin-
    | Semantic
262 |
263 | Conflicts:
264 | semantic.ml
265 |
266 | commit 46b797cb8cf1e386957a1db01fc5ac3fcb42dab3
267 | Author: Martin Ong <mo2454@columbia.edu>
268 | Date: Sat Dec 13 18:34:31 2014 -0500
269 |
270 | Validate Body statements
271 |
272 | commit 5de85273bf243307cdeceb78820e42dbf12176dd
273 | Author: detectiveconan2 <ggl2110@columbia.edu>
274 | Date: Sat Dec 13 18:33:47 2014 -0500
275 |
276 | updated semantic
277 |
278 | commit 141c700104e66341f7c54198c9aa1a7dbeb0a7bf
279 | Author: detectiveconan2 <ggl2110@columbia.edu>
280 | Date: Sat Dec 13 18:29:15 2014 -0500

```



```

281
282     semantic-check exp
283
284 commit a4d54011235fa544a7f57a86a4ad89772986b65b
285 Author: Alice Chang <avc2120@columbia.edu>
286 Date:   Sat Dec 13 18:24:53 2014 -0500
287
288     fixed
289
290 commit 73e696882a285c9552ad94845d4b347ce941f548
291 Author: Alice Chang <avc2120@columbia.edu>
292 Date:   Sat Dec 13 18:19:45 2014 -0500
293
294     only does graphics if needs
295
296 commit 96abeb25b77b6918b70ba49808edd2d5c0f34f10
297 Author: detectiveconan2 <ggl2110@columbia.edu>
298 Date:   Sat Dec 13 17:31:48 2014 -0500
299
300     semantic - add if else
301
302 commit 1816cada36af54d7f15e47ea29317212317e8b39
303 Author: Alice Chang <avc2120@columbia.edu>
304 Date:   Sat Dec 13 16:41:08 2014 -0500
305
306     graphics works
307
308 commit a961d8d2b9d24cde506318675ff8e7612e9bf328
309 Author: Alice Chang <avc2120@columbia.edu>
310 Date:   Sat Dec 13 14:18:30 2014 -0500
311
312     Added Graphics
313
314 commit 9638709611d3f8a44fb68e6c2f75719db60099db
315 Author: Alice Chang <avc2120@columbia.edu>
316 Date:   Fri Dec 12 16:09:45 2014 -0500
317
318     atom name added
319
320 commit 8d689b57835fdffb51147b6153699980ecec9b2c
321 Author: Alice Chang <avc2120@columbia.edu>
322 Date:   Fri Dec 12 14:41:02 2014 -0500
323
324     added access
325
326 commit 40e7c4c05fc091c84b9905a0e076090b124dfba7
327 Author: detectiveconan2 <ggl2110@columbia.edu>
328 Date:   Thu Dec 11 16:19:49 2014 -0500
329
330     more type checking-semantic

```

```

331
332 commit c1a55300d13fa05fceb6e30f063e35549d44a096
333 Author: detectiveconan2 <ggl2110@columbia.edu>
334 Date: Thu Dec 11 15:39:32 2014 -0500
335
336 semantic can check type of formals and vars
337
338 commit 3770dfa639145856494eeaa98356dc729ecf0f25
339 Author: Martin Ong <mo2454@columbia.edu>
340 Date: Thu Dec 11 15:36:29 2014 -0500
341
342 Semantic check function parameters
343
344 commit d4b62ce73837a83e44475d3b1bb851033649bf65
345 Author: Alice Chang <avc2120@columbia.edu>
346 Date: Thu Dec 11 14:44:44 2014 -0500
347
348 test8
349
350 commit 4061602fb31778cc3a894b91fe4a27f334827cf8
351 Author: Alice Chang <avc2120@columbia.edu>
352 Date: Thu Dec 11 14:44:14 2014 -0500
353
354 added while and test
355
356 commit 772979df7eb1f8fd4938aa218306bc653f12128e
357 Merge: b9e106a 4665f8b
358 Author: detectiveconan2 <ggl2110@columbia.edu>
359 Date: Thu Dec 11 14:13:43 2014 -0500
360
361 semantic
362
363 commit b9e106ad836a537a7c8b8839bb5ec1736f0106e4
364 Author: detectiveconan2 <ggl2110@columbia.edu>
365 Date: Thu Dec 11 14:12:00 2014 -0500
366
367 semantic - one error
368
369 commit 4665f8b74515c207e8f53801222aa35bab4ab13b
370 Merge: 65d1725 c448186
371 Author: Alice Chang <avc2120@columbia.edu>
372 Date: Wed Dec 10 22:09:38 2014 -0500
373
374 Merge branch 'Alice-1'
375
376 commit c4481864bb83166f93ca80348439f3b5dda0b454
377 Author: Alice Chang <avc2120@columbia.edu>
378 Date: Wed Dec 10 22:08:34 2014 -0500
379
380 Compiler Done!

```

```

381
382 commit 25b9f6191d3f6e6c4f97a67c82b450ca63be7d82
383 Author: Martin Ong <mo2454@columbia.edu>
384 Date:   Wed Dec 10 20:09:32 2014 -0500
385
386     Update gitignore to exclude .java files
387
388 commit b0a25d3a3c8073b076ef1dde5251a292ec8a84f9
389 Author: Alice Chang <avc2120@columbia.edu>
390 Date:   Wed Dec 10 15:41:01 2014 -0500
391
392     hello world compiles!
393
394 commit 65d17258dfe5e71bf9fbf2a411ed4008629e6e6e
395 Author: Alice Chang <avc2120@columbia.edu>
396 Date:   Wed Dec 10 15:13:33 2014 -0500
397
398     Merge branch 'Hello-World'
399
400     Conflicts:
401         ast.mli
402
403 commit 71b5686c9a48c57d1ad1359bdea07e674055ab2b
404 Author: Martin Ong <martinong@users.noreply.github.com>
405 Date:   Wed Dec 10 14:44:35 2014 -0500
406
407     Update .gitignore
408
409     Ignore test files generated by compiler
410
411 commit f36895bb0df639376bae04cbfd62ee3af3d35d29
412 Merge: 8f1698b af20856
413 Author: Alice Chang <avc2120@columbia.edu>
414 Date:   Wed Dec 10 14:44:12 2014 -0500
415
416     Merge branch 'master' of https://github.com/martinong/ChemLAB
417
418 commit 8f1698b02b35965390ea9a0c6f0b54520821008d
419 Author: Alice Chang <avc2120@columbia.edu>
420 Date:   Wed Dec 10 14:44:10 2014 -0500
421
422     deleted test files
423
424 commit 290b4962d82b491f2a709d216d760ce9fd3d53eb
425 Merge: 442fc78 24048cb
426 Author: Alice Chang <avc2120@columbia.edu>
427 Date:   Wed Dec 10 14:43:42 2014 -0500
428
429     Merge branch 'master' of https://github.com/martinong/ChemLAB
430

```

```

431     Conflicts:
432         ast.mli
433
434     commit af208566e2650dad6b689807448d7c6ef548df15
435     Merge: 24048cb df403a5
436     Author: Martin Ong <mo2454@columbia.edu>
437     Date:   Wed Dec 10 14:36:24 2014 -0500
438
439         Merge branch 'Martin'
440
441     commit df403a53220400f9965d9552a5e8ecbb1937d251
442     Author: Martin Ong <mo2454@columbia.edu>
443     Date:   Wed Dec 10 12:39:02 2014 -0500
444
445         Compiles and runs balancing one equation
446
447     commit 2d933805ef58312bce84937c4fdc8fa03c9f74d2
448     Author: Martin Ong <mo2454@columbia.edu>
449     Date:   Wed Dec 10 12:24:19 2014 -0500
450
451         Change main class name from "ChemLAB"
452
453     commit 24048cb02b5dafa8fae9e46bb61cdf7fad0cebda
454     Author: Martin Ong <mo2454@columbia.edu>
455     Date:   Sat Dec 6 02:01:51 2014 -0500
456
457         Minor touch ups
458
459     commit 4946d0dfad380cb72aff8e9cf8dc4a62fc9bef5f
460     Author: Martin Ong <martinong@users.noreply.github.com>
461     Date:   Sat Dec 6 02:00:49 2014 -0500
462
463         Delete chemistry.class
464
465     commit 06b9a52f11e9be2b03a9f96773490d3168bf8dbe
466     Author: Martin Ong <martinong@users.noreply.github.com>
467     Date:   Sat Dec 6 02:00:41 2014 -0500
468
469         Delete ChemLAB.class
470
471     commit ead2bf34ba65795ef9c77bf0195ad72981d8c1c3
472     Merge: e2410d9 a3a1787
473     Author: Martin Ong <mo2454@columbia.edu>
474     Date:   Sat Dec 6 01:55:14 2014 -0500
475
476         Merge branch 'Hello-World'
477
478     Conflicts:
479         .gitignore
480         ChemLAB.class

```

```

481     chemistry.class
482     compile.ml
483
484 commit e2410d9af6095fb2fee86053f08bce20694ad141
485 Merge: 93fcef3 ea0a11e
486 Author: Martin Ong <mo2454@columbia.edu>
487 Date:   Sat Dec 6 01:39:47 2014 -0500
488
489     Merge branch 'Gabe'
490
491     Conflicts:
492         Parse.java
493         ast.mli
494         semantic.ml
495
496 commit 93fcef39b26203931fec19e903e0154d2b345280
497 Merge: c0ce58b d8bec35
498 Author: Martin Ong <martinong@users.noreply.github.com>
499 Date:   Sat Dec 6 01:29:27 2014 -0500
500
501     Merge pull request #1 from martinong/Martin
502
503     Martin
504
505 commit d8bec354554c0bd18d61272caf8edca1e0de5e6e
506 Author: Martin Ong <mo2454@columbia.edu>
507 Date:   Sat Dec 6 01:27:32 2014 -0500
508
509     Changed "chemlab.ml" so that it can take arguments
510
511 commit ea0a11efe340c6d1affacaa45560ae1253fffd16
512 Author: detectiveconan2 <ggl2110@columbia.edu>
513 Date:   Sat Dec 6 00:01:55 2014 -0500
514
515     semantic compiles now
516
517 commit b9107a8684d79759cd09f4e19b32cebb90ee8c3f
518 Author: Martin Ong <mo2454@columbia.edu>
519 Date:   Fri Dec 5 23:06:26 2014 -0500
520
521     Make test.sh fancier
522
523 commit a3a1787fd9f556e3c509d3f39bbd7047881d70b3
524 Author: Alice Chang <avc2120@columbia.edu>
525 Date:   Fri Dec 5 23:02:35 2014 -0500
526
527     outputs equation
528
529 commit 442fc7809ee3bb259903b07d64e313851c026355
530 Author: Alice Chang <avc2120@columbia.edu>

```

```

531 Date:    Fri Dec 5 22:56:38 2014 -0500
532
533     prints out equation from compiler
534
535 commit c0ce58b6fd1241e14fc28d71535addbc635e8f7b
536 Author: Martin Ong <mo2454@columbia.edu>
537 Date:    Fri Dec 5 22:33:51 2014 -0500
538
539     Updated so that *.class is cleaned in "make clean"
540
541 commit 567834da2f2a67c1dc553f7dfad9290afb1e4837
542 Author: Martin Ong <mo2454@columbia.edu>
543 Date:    Fri Dec 5 22:28:40 2014 -0500
544
545     Ignore *.class files
546
547 commit acafa50b818fd91608cdd0d55cd6baf0aab0efe4
548 Merge: 7cb40f0 fe5a59a
549 Author: Martin Ong <mo2454@columbia.edu>
550 Date:    Fri Dec 5 22:23:57 2014 -0500
551
552     Merge branch 'Hello-World'
553
554 commit 531f8c2d1fb79436366258d0c1f75e1c7509b5f3
555 Author: Alice Chang <avc2120@columbia.edu>
556 Date:    Fri Dec 5 19:32:28 2014 -0500
557
558     fixed
559
560 commit fe5a59ac6382179d1860157455ad80be309acdf7
561 Author: Alice Chang <avc2120@columbia.edu>
562 Date:    Thu Dec 4 16:59:18 2014 -0500
563
564     edited!
565
566 commit 47093cb30085385a1f5c9ca92fd7397468c7fb2c
567 Author: Alice Chang <avc2120@columbia.edu>
568 Date:    Thu Dec 4 16:44:37 2014 -0500
569
570     error free!
571
572 commit f2befff33231a7638e9cee0945fcf46f557b59d5
573 Author: Alice Chang <avc2120@columbia.edu>
574 Date:    Thu Dec 4 14:56:21 2014 -0500
575
576     Fixed chemistry.java
577
578 commit cde87eb4db67e168cfe8cf45a7072e1adc93fa1f
579 Author: detectiveconan2 <ggl2110@columbia.edu>
580 Date:    Thu Dec 4 00:29:17 2014 -0500

```

```

581
582     function for CHEMLAB
583
584 commit 2ac58c076c8e6931694a2e2adff48b9079b2c1cf
585 Author: Alice Chang <avc2120@columbia.edu>
586 Date:   Wed Dec 3 20:07:09 2014 -0500
587
588     makefile changed and parser fixed
589
590 commit 1067f2c04e1f2b524ddba20545350861acd3c12f
591 Author: detectiveconan2 <ggl2110@columbia.edu>
592 Date:   Wed Dec 3 00:02:01 2014 -0500
593
594     semantic analyzer, added some checking for func
595
596 commit ca835111c8227af4590dfb3480b5fbe16eb92f2a
597 Author: Alice Chang <avc2120@columbia.edu>
598 Date:   Tue Dec 2 18:07:26 2014 -0500
599
600     compiler
601
602 commit b2010c8ffaad346d291fa0f80b5914ddab94ac4f
603 Author: Alice Chang <avc2120@columbia.edu>
604 Date:   Tue Dec 2 15:16:17 2014 -0500
605
606     added files and java program
607
608 commit d08499229e2538b818567096475f7ba0f8a67239
609 Author: Alice Chang <avc2120@columbia.edu>
610 Date:   Sun Nov 30 14:42:06 2014 -0500
611
612     added semantic check
613
614 commit b1d3aa91e1c93d02203ce6f89adc9b85ff58eadf
615 Author: Alice Chang <avc2120@columbia.edu>
616 Date:   Sun Nov 30 02:08:37 2014 -0500
617
618     first draft of parser done and working
619
620 commit 7a08f7b6cea8d7769a495c82e92634f5de3ad6a1
621 Author: Alice Chang <avc2120@columbia.edu>
622 Date:   Sun Nov 30 02:02:47 2014 -0500
623
624     fixed equation and molecule
625
626 commit 5bf35f528137afcedc3354dbccc5b233c334647d
627 Author: Alice Chang <avc2120@columbia.edu>
628 Date:   Sun Nov 30 01:57:25 2014 -0500
629
630     fixed elements and molecules

```

```

631
632 commit b98a5eebd57850d3895717de441afad6f8c60a06
633 Author: Alice Chang <avc2120@columbia.edu>
634 Date: Thu Nov 27 13:50:08 2014 -0500
635
636     edited ast
637
638 commit dc74852b700b71ab04374785303d3f435bf958b3
639 Author: Alice Chang <avc2120@columbia.edu>
640 Date: Wed Nov 26 16:51:14 2014 -0500
641
642     Merged
643
644 commit 0cf4ad2cc11cb83ef5b8f9dc30033fb85b9fb6c9
645 Merge: b8845ed ca1659e
646 Author: Alice Chang <avc2120@columbia.edu>
647 Date: Wed Nov 26 16:42:40 2014 -0500
648
649     Merge branch 'master' into Hello-World
650
651     Conflicts:
652         ast.mli
653         chemlab.ml
654         parser.mly
655         scanner.mll
656         test2.chem
657         test3.chem
658
659 commit b8845ed2c416bb5e628d4b2a83f2865737ba578e
660 Author: Alice Chang <avc2120@columbia.edu>
661 Date: Wed Nov 26 16:40:03 2014 -0500
662
663     Changed Makefile and Ast
664
665 commit ca1659ec93bfc1522b0bf7aa6bf4264130d860fe
666 Author: Alice Chang <avc2120@columbia.edu>
667 Date: Wed Nov 26 16:15:08 2014 -0500
668
669     Test cases work
670
671 commit fdd1517b2781e78ca4f2987004722ec5b54df4ed
672 Author: Alice Chang <avc2120@columbia.edu>
673 Date: Wed Nov 26 11:39:05 2014 -0500
674
675     added function functionality
676
677 commit fa99248d5d95e6c31576419b9ffbb25a7138976a
678 Author: Alice Chang <avc2120@columbia.edu>
679 Date: Tue Nov 25 16:01:02 2014 -0500
680

```



```

681 All test cases work
682
683 commit 04732499b6c2b5f61f13563b6613db6670c5559f
684 Author: Alice Chang <avc2120@columbia.edu>
685 Date: Tue Nov 25 15:39:51 2014 -0500
686
687 Added And Or
688
689 commit f40f7f6bb740370b5ed98ad719d146958300c38a
690 Author: Alice Chang <avc2120@columbia.edu>
691 Date: Tue Nov 25 15:31:54 2014 -0500
692
693 All test cases working from 1-9 exempt 2
694
695 commit 0ca59079eae956fbbd1021e7a2b84f4147f7fd24
696 Author: Alice Chang <avc2120@columbia.edu>
697 Date: Tue Nov 25 15:12:17 2014 -0500
698
699 Conditional and Arithmetic Working
700
701 commit cea344f925510da17345ead652a6e6e5185ce6e
702 Author: Alice Chang <avc2120@columbia.edu>
703 Date: Tue Nov 25 14:02:28 2014 -0500
704
705 Equation Declaration Works
706
707 commit e3495ba8aaae28c0462a81842bc663effe4a9e51
708 Author: Alice Chang <avc2120@columbia.edu>
709 Date: Tue Nov 25 13:42:16 2014 -0500
710
711 molecule declaration works
712
713 commit 7cb40f049e0a22498febc81c92ef7d9f9ddca1d7
714 Author: Martin Ong <mo2454@columbia.edu>
715 Date: Mon Nov 24 22:58:59 2014 -0500
716
717 Arithmetic parsed
718
719 Work on statement lists
720
721 commit d00a90a5f9d1298d3e6c493f261bf8a47c7d2cd2
722 Author: Martin Ong <mo2454@columbia.edu>
723 Date: Mon Nov 24 21:50:03 2014 -0500
724
725 Parser for test2
726
727 commit 88ae50e8901e9a7048066afca003e0a0f666576c
728 Author: Alice Chang <avc2120@columbia.edu>
729 Date: Mon Nov 24 21:49:16 2014 -0500
730

```

```

731     sat test2 done
732
733 commit 234a653fac8ced141e6dd814ed40698e281374f9
734 Author: Martin Ong <mo2454@columbia.edu>
735 Date:   Mon Nov 24 21:30:21 2014 -0500
736
737     Hello World!
738
739     Able to parse hello world test (test1)
740
741 commit 7503e6c2f58d566ba96b33868f4c16444a2e2795
742 Author: Martin Ong <mo2454@columbia.edu>
743 Date:   Mon Nov 24 21:22:50 2014 -0500
744
745     This makes now
746
747 commit bbfeb62c86c41fbd985f70199c891ecd5d6a78f9
748 Author: Alice Chang <avc2120@columbia.edu>
749 Date:   Mon Nov 24 19:12:50 2014 -0500
750
751     Original
752
753 commit dc1dae4471accb5ed2074d540c1bd99fa554c8f1
754 Author: Alice Chang <avc2120@columbia.edu>
755 Date:   Mon Nov 24 18:34:40 2014 -0500
756
757     deleted fdecl
758
759 commit 9d2415ffa9b0cc5d8fc0a257890af73855dcb906
760 Author: Alice Chang <avc2120@columbia.edu>
761 Date:   Mon Nov 24 18:25:57 2014 -0500
762
763     errors fixed
764
765 commit 25d667606ff66750854fbd54c314caf55b8a3057
766 Author: Alice Chang <avc2120@columbia.edu>
767 Date:   Mon Nov 24 18:22:28 2014 -0500
768
769     edited parser element
770
771 commit 16ef686ec8d9e691f54a1f242fc45bcaa6e54219
772 Author: Martin Ong <mo2454@columbia.edu>
773 Date:   Mon Nov 24 18:18:49 2014 -0500
774
775     Fixed test cases to include data type declaration
776
777 commit 9cb52f71c1fd94f11618d8444a93b6b46f134d45
778 Author: Martin Ong <mo2454@columbia.edu>
779 Date:   Wed Nov 19 21:53:47 2014 -0500
780

```

```

781     Debug
782
783 commit 5e0dddc6e268304f8ad7676148b723007dd887ee4
784 Author: Martin Ong <mo2454@columbia.edu>
785 Date:   Wed Nov 19 21:13:16 2014 -0500
786
787     Debug
788
789 commit 2bdd02c48a3464098e113e66bb8132748a65a75e
790 Merge: e0eefa4 04d0f25
791 Author: detectiveconan2 <ggl2110@columbia.edu>
792 Date:   Wed Nov 19 21:12:06 2014 -0500
793
794     Merge remote-tracking branch 'origin/master'
795
796     Conflicts:
797         ast.mli
798
799 commit 04d0f255868c7abe378ecd43dbb1adca91f753c5
800 Author: Martin Ong <mo2454@columbia.edu>
801 Date:   Wed Nov 19 20:54:33 2014 -0500
802
803     Debug Parser
804
805 commit bff270d0abdfb74cb70fa2b6a5756a9aee0758ee
806 Author: Alice Chang <avc2120@columbia.edu>
807 Date:   Wed Nov 19 20:42:07 2014 -0500
808
809     fixed list
810
811 commit 68c8327bde098c37da46ccab0e177bde4013c5c8
812 Author: Alice Chang <avc2120@columbia.edu>
813 Date:   Wed Nov 19 20:41:09 2014 -0500
814
815     edited list
816
817 commit d7ed25864869f53997fb067e46060f1bbf8e16ed
818 Author: Alice Chang <avc2120@columbia.edu>
819 Date:   Wed Nov 19 20:39:05 2014 -0500
820
821     fixed fdec
822
823 commit 641518e46dfa8eblaeed2d0bbfa912ffd22ce882
824 Author: Alice Chang <avc2120@columbia.edu>
825 Date:   Wed Nov 19 20:37:18 2014 -0500
826
827     Parser Partial Done
828
829 commit 23bba766a45febd2f1cf92b10460a86500951228
830 Author: Martin Ong <mo2454@columbia.edu>

```

831 Date: Wed Nov 19 20:35:24 2014 -0500
 832
 833 Test Stuff
 834
 835 commit e0eefa4514ac538a27fe1b96f901d78b26495949
 836 Author: detectiveconan2 <ggl2110@columbia.edu>
 837 Date: Wed Nov 19 19:18:30 2014 -0500
 838
 839 AST update
 840
 841 commit 06a83d97776e13124e9a71676bbbed5f424343d4d
 842 Author: Martin Ong <mo2454@columbia.edu>
 843 Date: Wed Nov 19 17:41:10 2014 -0500
 844
 845 Random commit
 846
 847 commit 74148dce3f31f7e5d0de52897cfe0d66dca9bdf6
 848 Author: Alice Chang <avc2120@columbia.edu>
 849 Date: Wed Nov 19 17:37:27 2014 -0500
 850
 851 Edits
 852
 853 commit 962f328af685ed094842ccf4a276fea2f17a31af
 854 Author: Martin Ong <mo2454@columbia.edu>
 855 Date: Tue Oct 21 17:08:59 2014 -0400
 856
 857 Merge parser and scanner with Martin
 858
 859 commit e088ac16fa73bd4891b7bec18f52f4cb70ecd9bd
 860 Author: detectiveconan2 <ggl2110@columbia.edu>
 861 Date: Tue Oct 21 16:56:37 2014 -0400
 862
 863 Parser-Gabriel
 864
 865 commit b655c806a5ba804249e878a38edcbc0304e978c9
 866 Author: detectiveconan2 <ggl2110@columbia.edu>
 867 Date: Tue Oct 21 16:53:09 2014 -0400
 868
 869 Wrote Scanner-Gabriel
 870
 871 Hi
 872
 873 commit 2ef66f2ca7e5ce900b0f49763680986a30c0cef8
 874 Merge: 42222cf 3a7fcb5
 875 Author: Martin Ong <mo2454@columbia.edu>
 876 Date: Tue Oct 21 16:49:44 2014 -0400
 877
 878 Merge branch 'master' of <https://github.com/martinong/ChemLAB>
 879
 880 Conflicts:

```

881 |         chemlab.ml
882 |
883 | commit 42222cf3a01bd523f1916d3fccfef3570a17853b
884 | Author: Martin Ong <mo2454@columbia.edu>
885 | Date:   Tue Oct 21 16:44:06 2014 -0400
886 |
887 |     Removed implementation stuff
888 |
889 | commit 3a7fcb5b5ba170ce7812396167a090dae519ad42
890 | Author: Alice Chang <avc2120@columbia.edu>
891 | Date:   Mon Oct 20 21:07:45 2014 -0400
892 |
893 |     added print hash map variable
894 |
895 | commit b0443e0c15fa46243a601bac1f3012ed568fcf72
896 | Author: Martin Ong <mo2454@columbia.edu>
897 | Date:   Mon Oct 20 20:19:22 2014 -0400
898 |
899 |     Cleaned merge mess
900 |
901 | commit c3d8b8b80e9c2ebdf017ed3b49dc726408509fdb
902 | Merge: 5e75369 144cfc7
903 | Author: Alice Chang <avc2120@columbia.edu>
904 | Date:   Mon Oct 20 20:13:44 2014 -0400
905 |
906 |     edited
907 |
908 | commit 5e75369a41970d398dd945a9feecaf6e65be5f9b
909 | Author: Alice Chang <avc2120@columbia.edu>
910 | Date:   Mon Oct 20 20:11:45 2014 -0400
911 |
912 |     edited
913 |
914 | commit 144cfc7f6719e62ce8fd0f521b56627c6fc7582a
915 | Author: Martin Ong <mo2454@columbia.edu>
916 | Date:   Mon Oct 20 20:11:31 2014 -0400
917 |
918 |     Comment working, variables are in progress
919 |
920 | commit bb5f80d4583b4ac13ac921db8402827b6a370812
921 | Author: Martin Ong <mo2454@columbia.edu>
922 | Date:   Mon Oct 20 19:04:16 2014 -0400
923 |
924 |     Print function working
925 |
926 |     Includes test file for printing
927 |
928 | commit bacdb7c691b8f8b52723451f771e03b05eea68a5
929 | Merge: 846faec 865fd5e
930 | Author: Alice Chang <avc2120@columbia.edu>

```

```

931 Date:   Mon Oct 20 18:06:17 2014 -0400
932
933     Merge branch 'master' of https://github.com/martinong/ChemLAB
934
935     Conflicts:
936         scanner.mll
937
938 commit 846faec24570fb722c87850050e93f100eebb5
939 Author: Alice Chang <avc2120@columbia.edu>
940 Date:   Mon Oct 20 18:04:42 2014 -0400
941
942     Parser and Scanner Edited
943
944 commit 865fd5ebba8f48e28420f1ef096a428bf7b83dae
945 Author: Martin Ong <mo2454@columbia.edu>
946 Date:   Sat Oct 11 15:29:51 2014 -0400
947
948     Tried to add print function
949
950 commit 2c364e4b2b2c6760cb72f1574f32143a3c9d656b
951 Author: Alice Chang <avc2120@columbia.edu>
952 Date:   Sat Oct 11 15:02:25 2014 -0400
953
954     Added tokens
955
956 commit 95b7a81b2ce176522160826f6fbf619138d0fec1
957 Author: Martin Ong <mo2454@columbia.edu>
958 Date:   Sat Oct 11 14:51:05 2014 -0400
959
960     Ignore files
961
962 commit c129eb2af310efc9c65b9891fdafe6c1bc333a16
963 Author: Alice Chang <avc2120@columbia.edu>
964 Date:   Sat Oct 11 14:48:52 2014 -0400
965
966     First Edit
967
968 commit 0f4bc817af1bdcc1cdf3bb47415678ce719e73b3
969 Author: Martin Ong <mo2454@columbia.edu>
970 Date:   Sat Oct 11 14:44:11 2014 -0400
971
972     Updated name in makefile
973
974 commit 561e083a8896858fb1125b8fa36730f43a8d0060
975 Author: Martin Ong <mo2454@columbia.edu>
976 Date:   Sat Oct 11 14:32:51 2014 -0400
977
978     Changed name from calc to chemlab
979
980 commit daa59975d3beaf15a469f93c7444ed69dd9e5a1e

```

```
981 Author: Martin Ong <mo2454@columbia.edu>
982 Date:   Sat Oct 11 14:30:04 2014 -0400
983
984     Added variables and sequencing
985
986     From homework 1 problem 3
987
988 commit 3ac1b97628ae7492ebb0a0b059c8c3c3838cf5ce
989 Author: Martin Ong <mo2454@columbia.edu>
990 Date:   Sat Oct 11 14:26:49 2014 -0400
991
992     Calculator parser from COMS W4115
993
994 commit b4bff48721d629be42c09396b8e056319c08fd9e
995 Author: Martin Ong <martinong@users.noreply.github.com>
996 Date:   Sat Oct 11 13:48:27 2014 -0400
997
998     Initial commit
```