ChemLab Language Reference Manual
COMS W4115

Alice Chang (Syst. Architect, Lang Guru), Gabriel Lu (PM, Lang Guru), Martin Ong (Syst. Architect, Tester)
avc2120, ggl2110, mo2454

October 27, 2014

Chapter 1: Introduction

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

Chapter 2: Types

2.1 Primitive Types
There are four primitive types in ChemLab: **boolean**, **int**, **double**, and **string**

2.1.1 Boolean
The boolean data type has only two possible values: **true** and **false**.

2.1.2 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.

2.1.3 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.

2.1.4 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 ‘‘ " ’’.

2.2 Non-Primitive Types
The language comes built-in with **lists**, **elements**, **molecules**, **equation, objects**

2.2.1 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].

2.2.2 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.

$^{12}_{6}C$

element C = {6, 12, 0};

$^{14}_{6}C$

element C = {6, 14, 0};

2.2.3 Molecule
For the purpose of the language, there is no distinction between molecule or compound and both are declared the same way. The implementation of a molecule is essentially a hash map with the elements as keys and the number of each element as the values. Once declared, molecules are immutable.

molecule NaCl = {Na: 1; Cl:1}

2.2.4 Equation
Equation is declared in the following way: (list of elements/molecules on right side of reaction, array of elements/molecules on left 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. One declared, an equation is immutable.
$NaOH + HCl \rightarrow NaCl + H_20$

equation NaClReaction = { [NaOH, HCl], [NaCl, H_2O] };

2.2.5 Objects
The types element, molecule, and equation were developed to include information needed for typical chemistry calculations. However, for more complex calculations, it may be necessary to include further information. Thus, users have the option to define their own types.

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

Chapter 3: Lexical Conventions

3.1 Identifiers
An identifier is a sequence of letters or digits in which the first character must be a uppercase letter. Upper and lower case letters are considered different.

3.2 Keywords

The following identifiers are reserved for use as keywords, and may not be used otherwise:

int	double	string	boolean	element
molecule	equation	if	else	while
function	return	true	false	print
object				

3.3 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.

3.4 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	function int sum(int a, int b)
;	Statement end	int x = 3;
" "	String declaration	string x = "hello"
[ ]	List delimiter	int x[] = [1,2,3,4]
{ }	Statement list delimiter, and element/molecule/equation declaration	if (expr) { statements }
( )	Conditional parameter delimiter, expression precedence	while(i > 2)

3.5 Comments

Much like in the C programming language, the characters /* introduce a comment, which terminates with the characters */. There are no single line comments in ChemLab.

3.6 Operators

Operator	Use	Associativity
=	Assignment	Right
= =	Test equivalence	Non-associative
!=	Test inequality	Non-associative
>	Greater than	Non-associative
<	Less than	Non-associative
>=	Greater than or equal to	Non-associative
<=	Less than or equal to	Non-associative
&&	AND	Non-associative
||	OR	Non-associative
.	Access	Left
*	Multiplication	Left
/	Division	Left

+	Addition	Left
-	Subtraction	Left
^	Exponentiation	Left
%	Modulo	Left

The precedence of operators is as follows:

.
* / % ^
+ -
< > <= >= && ||
= = !=
=

Chapter 4: Syntax

A program in ChemLab consists of a sequence of zero or more valid ChemLab statements.

4.1 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.

4.1.1 Constants

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

4.1.2 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
```

4.1.3 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.

a. *expression * expression*: The binary operator * indicates multiplication. It must be performed between two int types or two double types. No other combinations are allowed.
b. *expression / expression*: The binary operator / indicates division. The same type considerations as for multiplication apply.
c. *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.
d. *expression + expression*: The binary operator + indicates addition and returns the sum of the two expressions. The same type considerations as for multiplication apply.
e. *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.
  a. *expression1 > expression2*: The overall expression returns true if expression1 is greater than expression 2
  b. *expression1 < expression2*: The overall expression returns true if expression1 is less than expression 2
  c. *expression1 >= expression2*: The overall expression returns true if expression1 is greater than or equal to expression 2
  d. *expression1 <= expression2*: The overall expression returns true if expression1 is less than or equal to expression 2
  e. *expression1 == expression2*: The overall expression returns true if expression1 is equal to expression 2.
  f. *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.
  a. *expression1 && expression2*: The overall expression returns true if and only if expression1 evaluates to true and expression2 also evaluates to true.

b. *expression1* || *expression2*: The overall expression returns true as long as expression1 and expression2 both do not evaluate to false.

4.1.4 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.

4.1.5 Function Creation

All functions in ChemLab must have a return type. The syntax for declaration is as follows

```
function returnType functionName (type parameter1, type parameter 2, .......) {
        statements
        return x // where x has to be of type returnType
}
```

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 and there must be a return statement among these statements. 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.

4.1.6 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:
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

4.1.7 Object Creation

An object is created using the object keyword followed by a name and a list of properties surrounded by braces. Each property is an identifier separated by property declarations. Syntax for object declaration is as follows:

```
object objectName{
        properties-list
}
```

where *properties-list* is a list of identifier-expression pairs. The expressions of the identifiers must be given at the time of object creation. Object properties may be accessed using the following syntax: *objectName.Property*.

4.2 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.

4.2.1 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{

}
```

4.2.2 Iteration Statements

ChemLab does not have a for loop unlike most programming languages. The only iteration statement is the while loop. The while statements evaluates an expression before going into the body of the loop. The expression must be of type boolean and the while loop while 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.

4.2.3 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.

4.3 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.

Chapter 5: 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_20 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.