Chimico

The Chemists Weapon

September 7, 2021

Language Specification Document

Written by:

Narayana Sreehitha Reddy - `cs19btech11016`

Sri Hari Malla - `cs19btech11039`

Sehouriey Uppala - `cs19btech11015`

Sanjay Krishna Ambati - `cs19btech11013`

Likitha Langaru - `cs19btech11008`

Harshitha Sagiraju - `cs19btech11032`

Venkata Surya Sai Guntreddi - `cs19btech11014`

Chimico

-The Chemists weapon

1.INTRODUCTION TO CHIMICO

1.1 Description:

Chimico is a language predominantly build for chemists. The chemists are playing a vital role in the present world, by producing important chemicals and drugs as and when needed. Assisting them, technically using the computational power we have, makes it much more convenient for the chemists is what we thought and is the idea for the coming project. It is being builded to assist the operations like stoichiometric calculations, Redox reactions and ElectroChemistry. This language will make the above topics relatively easy as we will be having data types and methods that a chemist can use it with an ease and with minimal knowledge on technology. The data types are build such that, a beginner can get through the usage with in no time.

Sometimes, the calculations seems very simple to do, but later while dealing with large number of elements and molecules, one finds the difficulty underneath it by solving it with Human Brain. So, we are trying to assist them by building this language **Chimico**. With this language one can easily solve complex problems to a familiar extent. It saves both time and effort which can be used elsewhere, where it is actually needed.

We are looking forward it to complete this using **Flex** and **Bison**. At present, this language is confined to some of the topics, at later stage if time permits, we are willing to add many complex operations to make life easier.

1.2 Uses:

As mentioned above, the problems we are trying to solve using this language are procedural. For solving each of the problem, we have a stipulated way to solve.

- For Example, calculating the mass of the compound, takes in the empirical formula and gives out the mass. Simple application, while dealing with a reaction, if person doesn't recall the weight of so complex compound, one can easily use this language to calculate the mass.
- An extension to that is, Given the mass partition of elements, finding the molecular formula of the compund formed. Application to this is, after carrying a sucessful reaction, if the products are unknown, one can use this language to get the result.

As we know, there is no definite ending, a chemist can use this language as far as it is possible. Based on the demanding use, new features are included.

1.3 Design Goals:

So, to stay balanced, here are some of the design goals of the project. * Don't let any user alter the built-in utility functions. * Allowing users to easily use the methods without any difficulty. * Give users access to information they need to complete their task * Allow users to easily see and use the edit tools they need(manual)

2.chimico -- language familiarity

syntax ;; statementa and control flow;; funcs ;; balance reactions

The scope of programming languages in various fields is always increasing. For many scientists, computer literacy has become a must-have talent.

2.1 Features of Chimico Language

Chimico follows a similar syntax of C++ for control flow related stuctures. Its a structural programming language, and the statements are executed line by line. This section walks you through essential Features of chimico that helps you to understand the basic interface of Chimico Language.

2.1.1 Primitive Data types

2.1.1.1 Numeric Data types

Numeric Types of size 32-bit, 64-bit are present in Chimico Language. The Numeric Types are same as that of 'int' and 'double' in C++ or any other language.
The 'int' data type can store values from -2^{31} to $2^{31}-1$.
The 'double' data type is 64-bit long. Much like in C++, double is a double-precision floating point and follows IEEE 754 specifications. According to the IEEE 754, a binary64 is specified as follows:

```
 Sign bit              : 1-bit
Exponent              : 11-bits
Significand Precision : 53 bits(52 are explicitly stored)
```

2.1.1.2 Boolean Data Types

Boolean Data Types can only contain two values either 'true' or 'false'.

2.1.1.3 Strings

In Chimico Language, Strings are primitive Data Types. Even characters are stored in strings. There is no seperate Data Type to store characters since they are also treated as strings of length one.

A string is sequence of characters surounded by double quotes.
* Note: Only double Quotes surrounded sequences are considered as strings.

2.1.2 Non Primitive Data Types

Chimico Language comes with the following built-in types:

2.1.2.1 Element

Element is the basic building block provided in Chimico Language, It is used to store the elements present in the periodic table and possibly their isotopes and isobars. Elements are immutable. Elements can be stored using their atomic numbers, but to ensure we support definition of isotopes and different states of elements, an Element is defined using the following data set :

- Atomic Number
- Mass Number
- Charge Value

These Elements are used in further programming to represent molecules and equations.
$^{12}_{6}C$ is represented as element C = (6,12.0,0) ; $^{14}_{6}C$ is represented as element C = (6,14.0,0) ;

2.1.2.2 Molecule

Maintaining an Analogy to natural chemistry study, Element and Molecule represent the same thing the chemistry dictionaries contains. But to generalize and keep the syntax simple 'moleclue' is used to declare both molecules and compounds. Molecule is stored in the form of a map in C++. The Elements are used as keys and number of elements are stored as values.
For example Sodium Chloride is defined in the folowing format:

```
element Na = (11,22.98,0) ;
element Cl = (17,35.457,0) ;
molecule NaCl = {Na:1, Cl:1} ;
```

- Note: Chimico is a statically Typed Language. Variables should be defined explicitly.

2.2 Syntax of Chimico Language

2.2.1 Comments in Chimico Language

Chimico follows similar syntax as in general C or C++ for both single line and multiline Comments. * Single Line Comments - # Two backslashes are used to represent the beginning of single line comment, the rest of the line till the end is considered as comment. * Multiline Comments - #* Multiline line comments start with a backslash followed by a asterisk and end with an asterisk and backslash *#

```
# Single Line Comment
#* multiline
    Comments *#
```

2.2.2 Keywords

Keywords are reserved words used for specific purposes. They cant be used as variables or other purposes.

```
int         double          string        bool
element     molecule        func          true
false       for             while         if
else        write           read
```

2.2.3 Operators

Operator	Description	Associativity
*	Multiplication	Left
/	Division	Left
%	Modulo	Left
+	Addition	Left
-	Subtraction	Left

Operator	Description	Associativity
>	Greater than	Left
<	Less than	Left
>=	Greater than or equal to	Left
<=	Less than or equal to	Left
==	Test equivalence	Left
!=	Test inequality	Left
&&	AND	Left
||	OR	Left
.	Access	Left
=	Assignment	right

Order Of precedence:
+ / %
+ -
< > <= =>
== !=
&&
||
.
=

2.3 Statements:

Chimico will include basic control flow.

2.3.1 if-else statements:

In Chimoco, if-else statements has the following syntax :

```
if( expression)
{
     #Statements to be executed are mentioned here
}

else
{
    #Statements to be executed are mentioned here
}
```

Inside the 'if' statement condition, the expression should be of type 'Boolean' only.
The statements present inside the 'if' statement, will be executed if the expression inside if condition evaluates to be true, otherwise, (i.e, if false) the statements inside the 'else' statement is executed.

2.3.2 'while' loop iteration:

In Chimoco, while loop has the following syntax :

```
while ( expression )
{
    #Statements to be executed are mentioned here
}
```

If the expression is evaluated to be false, then the while loop breaks, else the statements in the while loop continue to execute until the loop breaks.

2.3.3 'for' loop iteration:

In Chimoco, for loop has the following syntax :

```
for (initialise ; condition ; increment or decrement)
{
    #Statements to be executed are mentioned here
}
```

Execution of statements inside for loop continues, as long as for loop condition is satisfied.

2.3.4Functions:

```
Function declaration:

func int AtomicNumber(element x)
{
   return x.atomno;
}
```

The keyword 'func' should be declared ahead of the function to mention that it is a function declaration.
Functions require function name, return type, and type of parameters.

2.4 Expressions:

An expression is a sequence of operators and operands which produces a value. The expression will be evaluated from left to right.

2.4.1 Constants:

```
data types string, boolean, int, double these can come under constants.
```

2.4.2 Identifiers:

An identifier is a sequence of letters and digits. The first character must be a letter; the underscore _ counts as a letter. Upper and lower case letters are different.Both primitive and non primitive types can be identified by it. The type and value of the identifier is determined by its designation. We can change the value of the identifier throughout the program, we can't declare more than one identifier with the same name.

```
string str = "chimico" ;
str = 3;  # as it is type safe it will show error.
bool  a= 4 // error as a will store bool but we are assigning int to it.
int str =  4 ;  #   error str has already declared
```

2.4.3 Binary Operators

- ***Arithmetic Operator:*** Arithmetic operators include %, +, *, / and -. The operands to an arithmetic operator must be int or float. The output will be int or float.
- ***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.
- ***Access operator:*** Expression.value, this is how the access operator looks like. Here expression must be of non-primitive type. It returns the value associated with that particular parameter.
- ***Logical operators:*** The operands to a logical operator must both be booleans and the result of the expression is also a boolean. Logical operators include OR (||) and AND (&&)
- ***Relational operators:*** Relational operators include <=, >=,<, >, ==, and !=. The operands will be int or float. The output will be bool.

2.5 BUILT-IN UTILITIES:

- ***Balancing the Equations:*** When an unbalanced equation is given as parameter, this built-in function computes the correct coefficients of each compound so that the given equation is balanced and return as an array of hashmaps Ex:

```
balance([{Zn,HCl},{ZnCl2,H2}]);
   # This returns  {1,2},{1,1}
```

- ***Calculation of molar mass:*** When a molecule or an element is given as input, this function returns the molar mass of the given element or molecule upto three decimal places.
  Ex:

```
element H=(1,1.007,0);
H.molar_mass();
       # This returns 1.007
```

- ***Molecules naming:*** When a molecule(i.e its formula) is given as a parameter, the built-in function prints out its correct chemical name.
  Ex:

```
NaOH.chem_name(); # Only works for basic compunds, still not completely sure with implementation.
```

This returns Sodium Hydroxide.

- ***Finding element:*** Given atomic number and atomic weight of an element, this function will return the element corresponding to the given atomic number and atomic weight.
  Ex:

```
 element random = (8,15.999,0) ;
random.name();
   # This prints the atomic symbol or name of the element - O
```

- ***Calculation of charge:*** When given a molecule, this will return the charge of the molecule.
  Ex:

```
element Na=(11,22.989,+1);
element Cl=(17,35.453,-1);
molecule NaCl=(Na:1,Cl:1);
NaCl.charge();
    # This will return 0.
```

- ***Finding Empirical formula:*** When given a molecule, this will return the empirical formula of the given molecule.
  Ex:

```
element H=(1,1.007,0);
element C=(6,12.010,0);
molecule C6H6=(C:6,H:6);
C6H6.emp_for();
    # This will return molecule CH.
```

3. Non-Trivial Input Programs

3.1 Example

```
func int main()
{
  element Na = (11,23,1);
  element Cl = (17,35.5,-1);

  if(Na.charge() > 0) write(Na.name() + "is a positive ion");
  else if(Na.charge() <0 ) write(Na.name() + "is a negative ion");
  else write(Na.name() + "is a neutral element");

  if(Cl.charge() > 0) write(Cl.name() + "is a positive ion");
  else if(Cl.charge() <0 ) write(Cl.name() + "is a negative ion");
  else write(Cl.name() + "is a neutral element");
}
```

Expected Output:
Na^{+} is a positive ion
Cl^{-} is a negative ion

3.2 Example

```
func double main()
{
  element H=(1,1.007,0);
  element C=(6,12.010,0);
  element O=(8,15.999,0);
  molecule C6H12O6=(C:6,H:12,O:6);
  molecule XX= C6H12O6.emp_for();
  write("molar mass of emprical formula of given molecule is" XX.molar_mass());


}
```