

## JO - JSON Object Language

### Reference Manual

#### Team

```
[ { "Name" : "Abhinav Bajaj", "UNI" : "ab3900", "Role" : "System Architect" },  
  { "Name" : "Arpit Gupta", "UNI" : "ag3418", "Role" : "Language Guru" },  
  { "Name" : "Chase Larson", "UNI" : "col2107", "Role" : "Manager" },  
  { "Name" : "Sriharsha Gundappa", "UNI" : "sg3163", "Role" : "Verification & Validation" } ]
```

# CONTENTS

1. Introduction
2. Language Tutorial
  - a. Program Execution
  - b. Variable declaration
  - c. Operators
  - d. Control Flow
  - e. Function Declaration
3. Reference Manual
  - a. Introduction
  - b. Lexical Convention
  - c. Type and Type Inference
  - d. Operators
  - e. Expressions
  - f. Statement
  - g. Scope
  - h. Built-in functions
4. Project Plan
  - a. Process
  - b. Style guide
  - c. Software environment
  - d. Team Role
  - e. Timeline
  - f. Project Log
5. Architecture Design
  - a. Pre processor
  - b. Scanner
  - c. Parser
  - d. Analyzer
  - e. Symbol Table
  - f. Code Generation
  - g. Team Contribution
6. Test Plan
  - a. Test suite automation
  - b. Test case selection
  - c. Test suites
  - d. Example programs
7. Lessons Learned
8. Appendix

# 1. INTRODUCTION

JSON or JavaScript Object Notation is an open standard format that uses human-readable text to transmit data objects consisting of attribute–value pairs. It is used as lightweight data interchange format to transmit data between a server and web application. JSON is also emerging as a preferred format in “NoSQL” databases. While languages like Python and Java have libraries to handle JSON data, they are not a native aspect of the language. JSON is presently a data format, rather than something fundamental to the language, like the object of an object oriented language, or the function of a functional language. With rise of trends in Big Data, Internet of Things, No-SQL databases, we believe that our language can provide a platform for building applications for these technologies with ease.

JO is simple yet powerful language to handle and manipulate JSON data. The language will treat JSON object as first class citizens and provide built-in functions that operate on these objects. These basic functions can be used to define complex libraries and applications like merging JSON, finding diff in JSON, SQL like queries on JSON objects. Our language attempts to facilitate any data operations by handling a lot of the business logic of handling JSON and their manipulations under the hood, and allowing the programmer to use JSON in a more native and intuitive way.

## 2. Language Tutorial

### 2.1 Program Execution

All programs in JO require a main function. The main function is called first upon executing a program and has no return statement.

```
func main ()  
    [your code]  
end
```

To Compile and run a JO program, run the following shell script:

To compile and run a JO program, run the following script:

```
./runjo.sh <file-name>
```

### 2.2 Variable Initialization

Variables are declared using the "=" sign. The type is inferred and does not need to be specified upon declaration. The following is an example declaration of each type supported in JO.

|                |                            |
|----------------|----------------------------|
| <b>String</b>  | a = "Hello World"          |
| <b>Number</b>  | b = 10                     |
| <b>JSON</b>    | c = #{ "name": "harsha" }# |
| <b>List</b>    | d = [4,6,"seven"]          |
| <b>Boolean</b> | e = true                   |
| <b>null</b>    | f = null                   |

### 2.3 Operators

#### 2.3.1 Basic Operators

The following comparison operators work on all data types.

If a = 3 and b = 2:

|                   |        |               |
|-------------------|--------|---------------|
| <b>Equality</b>   | a == b | returns false |
| <b>Not Equals</b> | a != b | returns true  |

JO supports mathematical operations on Numbers.

|                       |        |               |
|-----------------------|--------|---------------|
| <b>Addition</b>       | a ++ b | returns 5     |
| <b>Subtraction</b>    | a -- b | returns 1     |
| <b>Multiplication</b> | a ** b | returns 6     |
| <b>Division</b>       | a // b | returns 1.5   |
| <b>Modulo</b>         | a %% b | returns 1     |
| <b>Greater Than</b>   | a > b  | returns true  |
| <b>Less Than</b>      | a < b  | returns false |

|                              |        |              |
|------------------------------|--------|--------------|
| <b>Greater Than Equal To</b> | a >= b | return true  |
| <b>Less Than Equal To</b>    | a <= b | return false |

Strings can be concatenated together into one string.

If a = "Hello " and b = "World"

|                             |        |                       |
|-----------------------------|--------|-----------------------|
| <b>String Concatenation</b> | a ++ b | returns "Hello World" |
|-----------------------------|--------|-----------------------|

Lists can be concatenated together into one list.

If a = ["a", "b", "c"] and b = ["d"]

|                           |        |                              |
|---------------------------|--------|------------------------------|
| <b>List Concatenation</b> | a ++ b | returns ["a", "b", "c", "d"] |
|---------------------------|--------|------------------------------|

Multiple variables can be concatenated together to form one list where each variable is an element of that list.

If a = "a" and b = ["b", "c"]

|                           |       |                            |
|---------------------------|-------|----------------------------|
| <b>List Concatenation</b> | a + b | returns ["a", ["b", "c"] ] |
|---------------------------|-------|----------------------------|

Any data type can be printed to standard out using the print function. If a JSON is printed, the print function "pretty prints" the JSON.

If a = 5 and b = #{ "name": "harsha", "courses": [1, "PLT", true] }#

|                     |                |   |
|---------------------|----------------|---|
| <b>Print Number</b> | print(a)       | prints: 5   |
| <b>Print JSON</b>   | print (b)      | prints:<br>{<br>"courses":[<br>1,<br>"PLT",<br>true<br>],<br>"name":"harsha"<br>} |
| <b>Print List</b>   | print([1,2,3]) | [1,2,3]   |

### 2.3.2 Operations on Non-Primitive Types

To access an element in a JSON, use "[" following the JSON, with the string key inside the "["].

If a = #{ "name": "harsha", "courses": [1, "PLT", true] }#

|                       |           |                   |
|-----------------------|-----------|-------------------|
| <b>Element Access</b> | a["name"] | returns: "harsha" |
|-----------------------|-----------|-------------------|

To access an element in a List, use "[]" following the List, with the index key inside the "[]". The indexes start with 0 and goes till List length minus one.

If a = [1,2,3,4]

|                       |      |            |
|-----------------------|------|------------|
| <b>Element Access</b> | a[1] | returns: 2 |
|-----------------------|------|------------|

To assign an element in a JSON, you access the key using the above notation and then use the assignment operator "=" to assign an element to that key.

If a = #{ "a": "b" }#

|                           |              |  |
|---------------------------|--------------|--|
| <b>Element Assignment</b> | a["c"] = "d" | results in the JSON:<br>#{ "a": "b", "c": "d" }# |
|---------------------------|--------------|--|

To remove an Element from a JSON, you can use JSON - "string key"

If a = #{ "a": "b", "c": "d" }#

|                        |         |  |
|------------------------|---------|--|
| <b>Element Removal</b> | a - "c" | results in the JSON:<br>#{ "a": "b" }# |
|------------------------|---------|--|

To remove an Element from a List, you can use LIST - element or LIST - LIST

If a = ["a", "b", "c"] , b = ["b", "c"] , d = "c"

|                        |       |                                 |
|------------------------|-------|---------------------------------|
| <b>Element Removal</b> | a - d | results in the List: ["a", "c"] |
| <b>List Removal</b>    | a - b | results in the List: ["a"]      |

To see the data type contained within a JSON, use the typeStruct function. This returns a string that shows the entire JSON, with each element being replaced by a string of its data type.

If a = #{ "name": "harsha", "number": 12223, "list": [], "json": { "name": "harsha" }, "boolean": true }#

|                   |              |   |
|-------------------|--------------|---|
| <b>typeStruct</b> | a.typeStruct | results in the String:<br>{ String : String, String : Number, String : List, String : { String : String }, String : Boolean } |
|-------------------|--------------|---|

## 2.4 Control Flow

JO contains if...else statements to control whether a block of code executes. close if..else.. statements with the keyword "end".

|  |              |
|--|--------------|
| <pre>if ( 5 == 5 )   print ("true") else   print ("false") end</pre> | prints: true |
|--|--------------|

|  |                          |
|--|--------------------------|
| <pre>l = [1,2,3,4] if (5 in l)   print ("true") else   print ("false") end</pre> | <pre>prints: false</pre> |
|--|--------------------------|

JO contains a for loop. For loops operate by iterating over each element in a list. Close for loops with the keyword "end"

If a = [2,3,4]

|                                       |                        |
|---------------------------------------|------------------------|
| <pre>for i in a   print (i) end</pre> | <pre>prints: 234</pre> |
|---------------------------------------|------------------------|

## 2.5 Function Declaration

Use the "func" keyword to define a function. The format is: func functionName(arguments). Close functions with the keyword "end".

Below is an example of the GCD algorithm being defined.

```
func findGCD(a,b)
  if(b == 0)
    return a
  end
  return findGCD(b, a%%b)
end
```

## 3. Language Manual

### 3.1 Introduction

JSON or JavaScript Object Notation is an open standard format that uses human-readable text to transmit data objects consisting of attribute–value pairs. It is used as lightweight data interchange format to transmit data between a server and web application. JSON is also emerging as a preferred format in “NoSQL” databases. While languages like Python and Java have libraries to handle JSON data, they are not a native aspect of the language. JSON is presently a data format, rather than something fundamental to the language, like the object of an object oriented language, or the function of a functional language. With rise of trends in Big Data, Internet of Things, No-SQL databases, we believe that our language can provide a platform for building applications for these technologies with ease.

JO is simple yet powerful language to handle and manipulate JSON data. The language will treat JSON object as first class citizens and provide built-in functions that operate on these objects. These basic functions can be used to define complex libraries and applications like merging JSON, finding diff in JSON, SQL like queries on JSON objects. Our language attempts to facilitate any data operations by handling a lot of the business logic of handling JSON and their manipulations under the hood, and allowing the programmer to use JSON in a more native and intuitive way.

### 3.2 Lexical Convention

#### 3.2.1 Comments

The characters `/*` introduce a multi-line comment, which terminates with the characters `*/`. Multi-line comments cannot be nested within multi-line comments. Single line comments are also written in the same way as multi-line comments, with `/*` and `*/` appearing on the same line.

```
/* single line comments look like this */

/* this is
   how a multiple
   line comment looks like */

/* this however
   /* does not */
   works */
```

#### 3.2.2 Tokens

A token is a string of ASCII characters that is always at least 1 character long. There are different types of tokens in JO. These are described below.



### 3.2.2.1 Identifiers

An identifier consists of a letter followed by other letters, digits and underscores. The letters are the ASCII characters a-z and A-Z. Digits are ASCII characters 0-9. The language is case sensitive. An identifier can not begin with "loopVar".

*letter* → ['a'-'z' 'A'-'Z']

*digit* → ['0'-'9']

*underscore* → '\_'

*identifier* → letter (letter | digit | underscore)\*

### 3.2.2.2 Keywords

Keywords are identifiers reserved by the language. Thus, they are not available for re-definition or overloading by users.

| Keyword                               | Description                     |
|---------------------------------------|---------------------------------|
| Number, String, Bool, Json, List      | Data types                      |
| true, false, null                     | Literals                        |
| for, if, else, elsif, end             | statement constructs            |
| func, return                          | function declaration constructs |
| print type typestruct read makeString | in-built functions              |

### 3.2.2.3 Literals

Literals are expressions with fixed value. In the language there is capability for String, Number, Bool literals

*digit* → ['0'-'9']

*decimal* → '.'

*String Literal* → (.)+

*Number Literal* → (-)? (digit)+ (decimal)? (digit)+

*Bool Literal* → true | false

### 3.2.2.4 Newlines

"EOL" is taken as a newline character.

### 3.2.2.5 Whitespace

Whitespace consists of any sequence of blank and tab characters. Whitespace is used to separate tokens and format programs. The compiler ignores all whitespace. As a result, indentations are insignificant.

## 3.3. Types and Type Inference

### 3.3.1 Primitive Types

There are four types of primitives in JO language. These are Bool, Number, String and Null.

#### 3.3.1.1 Bool

Bool type can either carry a value of true or false. Booleans are considered their own type, meaning that an expression that uses a boolean operator and a non-boolean variable will cause an error. For example -

```
a = true
If (a && 10) will cause error.
```

#### 3.3.1.2 Number

A Number in JO is a double-precision floating-point format storing numbers in 64 bits, where the number (the fraction) is stored in bits 0 to 51, the exponent in bits 52 to 62, and the sign in bit 63:1.7E +/- 308 (15 digits). All numeric types will be stored as Number in JO. Hence, Number contains decimal part by default.

#### 3.3.1.3 String

A string is a sequence of characters surrounded by double quotes “ ”.

#### 3.3.1.4 Null

null is an empty data type. For example -

```
a = null
```

### 3.3.2 Non-Primitive Types

There are two types of non-primitive types or complex data types in JO Language. They are explained below -

#### 3.3.2.1 List

List is an ordered data type of primitive or complex data types. So a list can contain another list as one of its element along with JSON type as element and other Primitive types as elements in the list. Lists are enclosed in []. For example

```
["apple", 45, {"name":"harris"} ]
```

### 3.3.2.2 JSON

JSON or JavaScript Object Notation is an open standard format that uses human-readable text to transmit data objects consisting of attribute–value pairs. JSON is declared within '#' character, ##.

For example -

```
#{  
  "Name": { "First":"Arpit", "Last":"Gupta" },  
  "School": "Columbia",  
  "Age": 22,  
  "Courses": [ "PLT", "ML" ]  
}#
```

### 3.3.3 Type Inference

Data types in JO are expressed using a finite and well-defined set of data types. However when writing a JO program, the data types are not explicitly declared. JO has dynamic type-inference. Assignment statement can update the type of a datatype.

```
func test(l)  
  print (l - "d")  
  return null  
end  
  
func main ()  
  e = ["a", "c", 'd']  
  test(e)  
  e = 5  
end
```

## 3.4. Operators

### 3.4.1 Type of Operators

#### 3.4.1.1 Object Operators

| Operator | Description  |
|----------|--|
| +        | Concatenation, works on any data type arguments, returns list<br>Example<br>1. $5 + 2 = [5, 2]$<br>2. $[5, 2] + 3 = [[5, 2], 3]$<br>3. $\text{JsonA} + \text{JsonB} = [\text{JsonA}, \text{JsonB}]$  |
| -        | Usage : $A - B$ , works when $A$ is a Json or List<br>Removes attributes from $A$ which exactly matches with $B$<br>Valid Data types - <ul style="list-style-type: none"><li>• Json - String (here Operand <math>B</math> acts on the key of Json, rather than Json itself)</li><li>• List - List (removes the list if it exists as an element in other List)</li><li>• List - String</li><li>• List - Json</li><li>• List - Number</li></ul> Example<br>1. $\{ \{ \text{"name"}: \{ \text{first: chase, last: larson} \} \}, \text{marks: [2,3]} \} - \text{"name"} = \{ \text{marks: [2,3]} \}$<br>2. $[\text{"able"}, [\text{"barista"}, \text{"carrie"}]] - [\text{"barista"}, \text{"carrie"}] = [\text{"able"}]$<br>3. $[\text{"able"}, \text{"barista"}, \text{"carrie"}] - [\text{"barista"}, \text{"carrie"}] = [\text{"able"}, \text{"barista"}, \text{"carrie"}]$ |
| []       | 1. $[]$ access values for attributes. Works on Json and List objects <ul style="list-style-type: none"><li>• <math>a = \text{json1}[\text{"Name"}]</math>, returns the value at the attribute.</li><li>• <math>\text{json1}[\text{"Name"}] = \text{"Arpit"}</math>, stores value "Arpit" for attribute "Name"</li><li>• <math>a = \text{list1}[2]</math>, returns 3rd element stored in List</li><li>• <math>\text{list1}[3] = \# \{ \text{"name"} : \text{"arpit"} \} \#</math>, stores Json object at index 3, size of list at this time must be greater than equal to 4.</li></ul> 2. $[]$ - constructs a new list  |
| ==       | Compare two same data types. Returns true if their values match else false   |
| !=       | Compare two same data types. Returns false if their values match else true   |
| =        | Assignment Operator<br>Usage: $A = B$ , where $A$ is a object and $B$ is an expression   |
| .        | Calls function on an object  |
| #{}#     | Constructs a Json object   |

### 3.4.1.2 Mathematical Operators

All Mathematical Operators are only valid for Type Number.

| Operator | Description    | Example                |
|----------|----------------|------------------------|
| ++       | Addition       | 2 ++ 2 results in 4    |
| --       | Subtraction    | 2 -- 2 results in 0    |
| **       | Multiplication | 2 ** 2 results in 4    |
| //       | Division       | 2 // 2 results in 1    |
| >        | Greater Than   | 2 > 1 results in true  |
| <        | Less Than      | 2 < 1 results in false |
| %%       | Modulo         | 7 % 3 results in 1     |

### 3.4.1.3 String Operators

| Operator | Description          | Example                        |
|----------|----------------------|--------------------------------|
| ++       | String concatenation | "JS" ++ "ON" results in "JSON" |

### 3.4.1.4 List Operators

| Operator | Description        | Example                      |
|----------|--------------------|------------------------------|
| ++       | List concatenation | [1] ++ [2] results in [1, 2] |

### 3.4.1.5 Logical Operators

All Logical Operators only valid for type Bool.

| Operator | Description      | Example                               |
|----------|------------------|---------------------------------------|
| &&       | Logical And      | if A and B are true, (A && B) is true |
|          | Logical Or       | if A or B are true, (A    B) is true  |
| !        | Logical Negation | if A is false, !A is true             |

### 3.4.1.6 Membership Operators

| Operator | Description                                      | Example  |
|----------|--|--|
| in       | Results in true if variable is in given list     | A in B: results in true if variable A is found in list B.        |
| not in   | Results in true if variable is not in given list | A not in B: results in true if variable A is not found in list B |

### 3.4.2 Operator Precedence

All operators are evaluated left to right except !, in and not in. !, in and not in have right associativity

The below sequence of operator is in decreasing order of precedence. The operators on the same line have same precedence.

in , not in , !

+, -

\*, //

<, >, <=, >=, %%

=, !=

&&, ||

## 3.5. Expressions

An expression contains at least one operand and zero or more operators that return a value. Operands may be constants, variables, and functions.

Parentheses can be used to group sub-expressions, with the innermost sub-expression being evaluated first. For example -

```
(2 ++ 2) // ((3 -- 1) ** 2)
```

3 -- 1 is evaluated to 2 first. Then 2 ++ 2 and 2 \*\* 2 are both evaluated to 4. Finally 4 // 4 is evaluated resulting in 1.

### 3.5.1 Function Declaration

The declaration specifies the name of the function, list of parameters

The *func* keyword is used signify the declaration of a function. The general form is:

```
func <function_name> (<parameters>)  
    <function_body>  
    return <arg>  
end
```

The keyword *func* must be followed by a space, with the *function name* following. Then the list of *parameters* must be on the same line and enclosed by parentheses. The parameters are separated by commas. Following the closing parentheses, a new line must start before the *function body*.

The *function body* is a series of statements that specifies what the function actually does. It must be on a new line following the function declaration. The function body must contain a *return* statement.

The *return statement* is the keyword *return* followed by the expression to be returned from the function. For a function that should not return anything, it should return null. `main()` function is not required to have a return statement.

For example:

```
func sum(x, y)  
    return ( x ++ y)  
end
```

### 3.5.2 Function Call

You call a function by using its name and supplying any required parameters. If no parameters are required by the function, the parenthesis is still required.

General Form:

<function\_name> (<parameters>)

For example:

```
foo (5, A)
```

Here the function 'foo' is called with the parameters '5' and 'A' (here 'A' is a variable that has been declared previously).



## 3.6. Statements

You write statements to cause action and to control flow within your program.

### 3.6.1 Assignment Statement

An assignment statement consists of a modifiable variable name followed by the assignment operator “=” followed by a valid expression.

```
/* declares a json*/
var = #{ "name" : "Arpit Gupta" , "role" : "Language Guru" } #
/* declares a list*/
var = [ "Arpit" , "Abhinav" , "Harsha" , "Chase" ]
```

### 3.6.2 Expression Statement

An expression statement executes the specified expression.

For example -

```
2 ++ 3
```

```
foo (2, 3)
```

Expression statements are useful when they have some sort of side effect, such as calling a function or storing a variable.

### 3.6.3 If...else... statement

The *if ... else* statement is used to execute part of your program only under certain conditions. The general form is:

```
if (<condition>
    <then-statement(s)>
else if (<condition>)
    <then-statement(s)>
    end
else
    <else-statement>
end
```

The *then-statement* is executed if the condition evaluates to true. The else statement is optional, and it executes only if the condition evaluates to false.

The *condition* must be on the same line and immediately following the *if* keyword. The *condition* must be a boolean expression.

The *then statement* must start on a new line following the <condition>. If an *else* clause is included, the else statement must be on its own line and the *else statement* must be on a new line following the *else* keyword.

The entire statement must terminate with the *end* keyword.

### 3.6.4 For statement

The *for* statement is used to iterate over list, executing a block of code. The general form is:

```
for <iterative expression>
  <statement>
end
```

The iterative expression must be of the form:

```
<variable> in <List>
```

The *statement* is executed once for each element in the list. The iterative expression must be on the same line as the keyword *if*. The *statement* must be on a new line following the iterative expression. The statement must end with the keyword *end*.

For example -

```
listVar = [1,2,3]
for x in listVar
    y = x ++ 1 /* increment value by one and stores in y */
end
print(x)
/*throws error as x is not defined outside the scope of for */
```

### 3.7. Scope

For example, the following program results in an error:

If an object or variable is declared within a function, if-else or a for statement block then it is only visible

```
globalVar = 5
func test()
    for x in varList
        if (x > 1)
            y = x
        end
        z = y
    end
    globalVar = 55 /* Not allowed globalVar are constant */
end
```

within that block. Otherwise, the object or variable is visible across the program.

The variable *y* only has scope within the *if* block, because that is the smallest block within which it is declared. The NUMBER *z* cannot be assigned to the NUMBER *y*, because the program can't "see" the *y* outside of the *if* statement.

## 3.8. Built-in Functions

JO provides utility functions which assists in JSON manipulation. Below are the functions built-in the language.

### 3.8.1 Read

By using the read function one can read file from specified path. This file will be parsed and returned as a Json Object. Individual elements in the file have to be comma separated. Each file is parsed into one List.

Below is the file contents of file “path/to/file.txt”

### 3.8.2 Write

```
{“name”:“harsha” , “itemList”:[1,“PLT”,true]}
```

```
input = write(varName , “path/to/file.txt”) /* appends to file */
```

Write function always append to the file whose path is specified.

### 3.8.3 Print

Print function prints to standard output. Print can take any type as input. If the input is a JSON object then the print function outputs a pretty print of JSON object.

```
print (<expr>) /* prints to standard output */
```

```
print (a)
```

### 3.8.4 Type

Type returns a String of the data type of a variable.

### 8.5 TypeStruct

This is a utility function built inside JSON. It returns a String containing the data type of the attribute-values stored in JSON object.

```
/* Find Json attribute-value data types */  
myJSON = #{ "name": { "first": "chase", "last": "larson" }, "age" :  
23 }#  
myJSON.typeStruct() /* returns String of { String : { String:  
String, String: String}, String: Number } */
```

### 3.8.6 AttrList

This is a utility function built inside JSON. It returns a List of String containing the keys in Json Object

```
/* */  
myJSON = { "name": { "first": "chase", "last": "larson" }, "age" :  
23 }  
myJSON.attrList() /* returns the List ["name" , "last"] */
```

### 3.8.7 makeString

makeString takes any data type supported in JO as input and returns a String data type.

```
/* Convert to String */  
makeString(5) /* returns "5" */  
makeString(aJson)
```

# 4. Project Plan

## 4.1 Process

### 4.1.1 Planning

Weekly meetings were held to discuss what aspects of the project needed to be completed next. These meetings were held both in person and over Google Hangout. During these meetings we discussed the status of the project and outlined what needed to be accomplished next. Each member was then assigned to complete part of what was outlined.

### 4.1.2 Specification

All aspects of the LRM were discussed during group meetings and then the work of writing what was discussed was divided between the team members. After the LRM was completed and the development process began, there were many times that some ambiguity needed to be cleared or that certain elements of the specification needed to be changed. These changes were discussed in our meetings and noted.

### 4.1.3 Development

For development, we utilized Git to allow each member to develop independently. We first worked to get one complete feature, print, functioning end-to-end. Once we had this feature fully implemented, we looked at our LRM and prioritized each feature. We then iterated over fully implementing each feature.

### 4.1.4 Testing

As each feature was developed, a unit test was created to test the full implementation of the feature. Because we had different members working on the front end and back end of each feature, we also ran independent tests on the front and back end. The front end checked to make sure the correct c++ code was generated and the back end checked to make sure that the c++ code generated produced the correct final output. Testing on multiple levels allowed members to work on the same feature at different paces.

## 4.2 Style Guide

As this was the first project we had done using OCaml, we did not initially have a style guide. As time went on, several style practices were adopted to maximize readability and minimize the time required for a team member to understand new code.

- Variables and functions are declared using camel case.
- File names are written in Pascal case.
- Use utility functions to minimize function length.
- All necessary headers should be included in the CPlusPlusCompiler.h file. This file would then be the only file required to include.

## 4.3 Software Environment

Environments used for Development: Mac OS X, Ubuntu.

Languages used in Development: OCaml, C++.

Version Control: Git hosted on github.

Other Tools: Bash shell scripts, Makefiles.

### 4.3 Team Roles

To help make sure the project went forward efficiently, we assigned each team member a role. The member in this role was given the final say when it came to topics that fell under their role. We also assigned a backup to each role. This was done just in case a situation arose where a team member was unable to fulfil their role at a given time. These roles were especially important in the early stages of the project as we discussed the details of the language, the environment we would use, and the plan for how we would go forward with the work.

Members assigned to each role:

| <u>Role</u>               | <u>Main</u> | <u>Backup</u> |
|---------------------------|-------------|---------------|
| Manager                   | Chase       | Abhinav       |
| Language Guru             | Arpit       | Chase         |
| System Architect          | Abhinav     | Sriharsha     |
| Verification & Validation | Sriharsha   | Arpit         |

When it came to writing code, we divided the code into 2 sections: front end (the scanner, parser and type inference) and back end (generation of c++).

Members that were primarily responsible for these sections:

|            |                    |
|------------|--------------------|
| Front End: | Abhinav, Sriharsha |
| Back End:  | Arpit, Chase       |

### 4.4 Timeline

The table below shows when the major milestones were complete.

| <u>Date</u> | <u>Milestone</u>        |
|-------------|-------------------------|
| 2014-09-24  | Proposal Due            |
| 2014-10-27  | LRM Due                 |
| 2014-11-16  | Full Scanner and Parser |
| 2014-11-23  | "Hello World" program   |
| 2014-12-12  | GCD program             |
| 2014-12-17  | Final Project Due       |

The following 2 charts show when code was being pushed to the repository and how substantial the changes were. The first shows the number of commits that were pushed per day over time. The second shows the number of lines of code that were added and deleted over time.

Fig. 1: Commits over time.

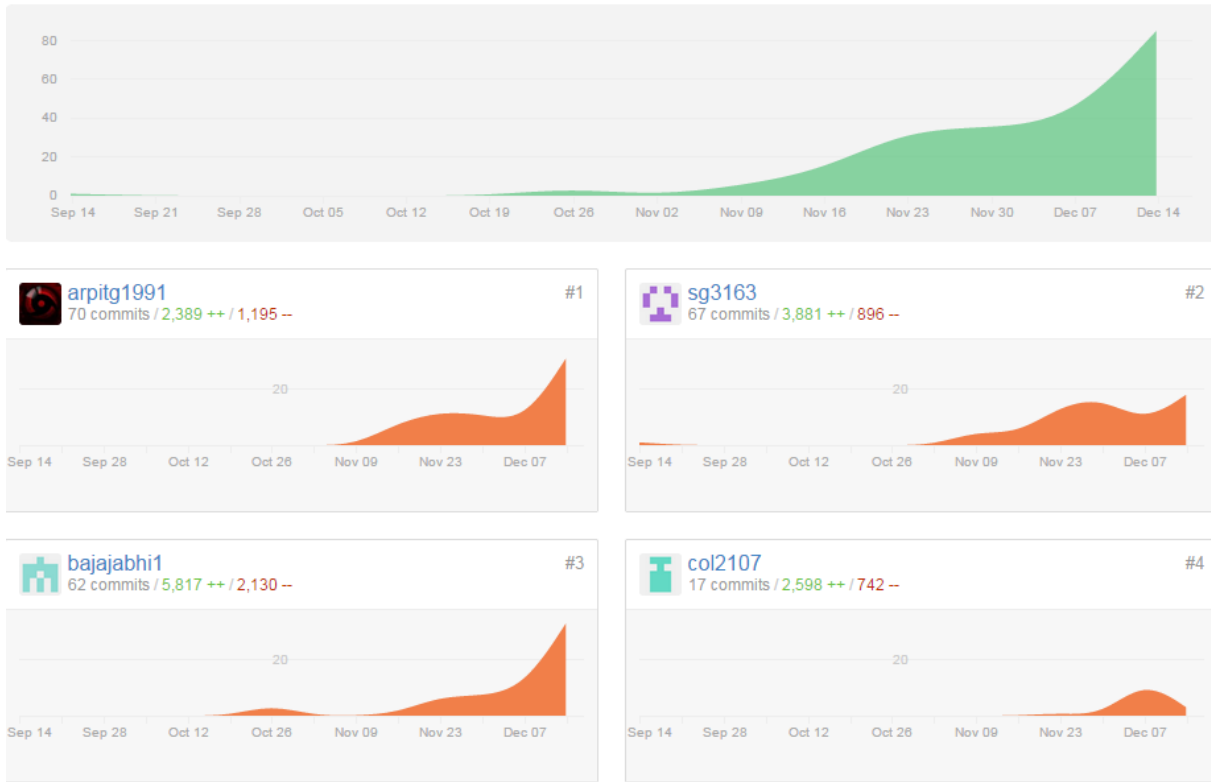
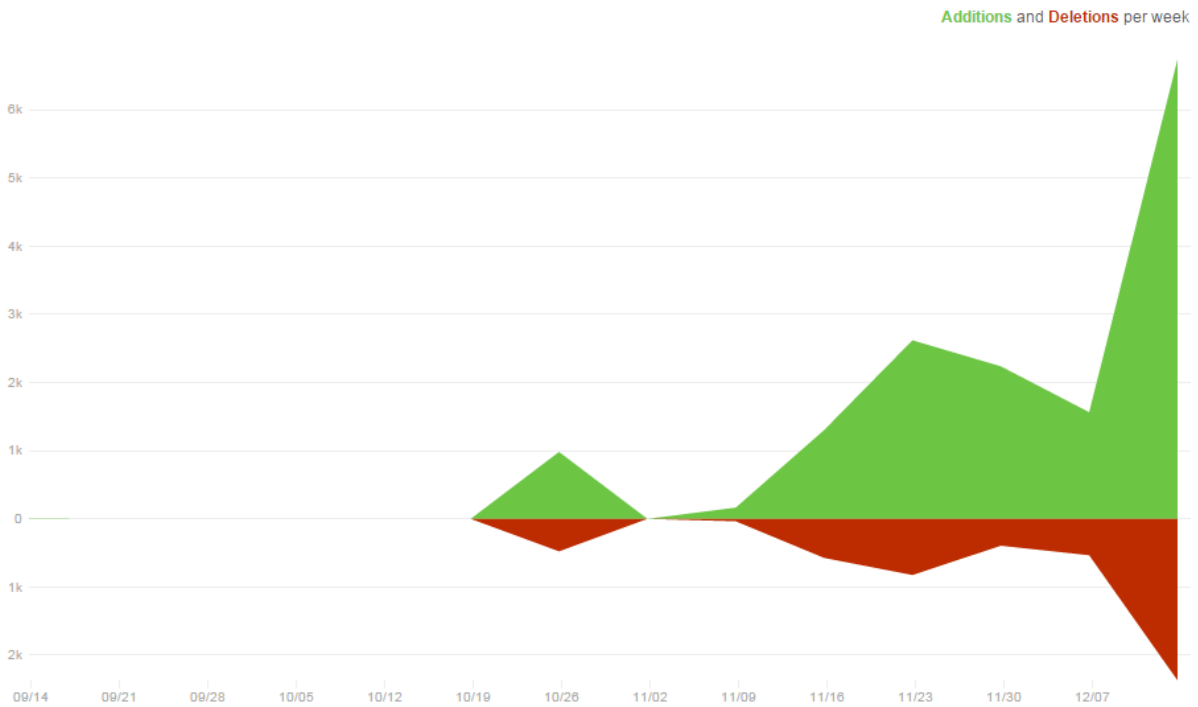


Fig. 2: Additions and Deletions over time.





## 4.5 Project Log

The following is the git log showing the date, author, and subject of each commit.

2014-12-17 - arpitg1991 - Code Freeze. May the force be with you!.  
2014-12-17 - arpitg1991 - Code Freeze. Tonight we dine in hell.  
2014-12-17- Sriharsha Gundappa - merged code  
2014-12-17- Sriharsha Gundappa - reading-test-cases  
2014-12-17 - arpitg1991 - updated file read  
2014-12-17 - bajjabhil - merging after ocaml was drunk  
2014-12-17 - bajjabhil - ocaml is drunk and prints 5 to 5.  
2014-12-17 - arpitg1991 - trying to revert back again, once more  
2014-12-17 - bajjabhil - adding analyzer file  
2014-12-17 - bajjabhil - merged  
2014-12-17 - bajjabhil - changed filename of typecheck to analyzer  
2014-12-17 - arpitg1991 - Merging g branch 'master' of <https://github.com/sg3163/plt2014fall>  
2014-12-17 - arpitg1991 - reverting back to plder style of attribute list  
2014-12-17 - bajjabhil - reverting incorrect change  
2014-12-17 - bajjabhil - added negative num support and sample prog for data proc  
2014-12-17 - bajjabhil - added support for float numbers  
2014-12-17 - bajjabhil - added support for float  
2014-12-17 - Sriharsha Gundappa - Some more test cases  
2014-12-17 - Sriharsha Gundappa - more test cases  
2014-12-17 - Sriharsha Gundappa - Merge branch 'master' of <https://github.com:443/sg3163/plt2014fall>  
merged  
2014-12-17 - Sriharsha Gundappa - Test cases for Concatenation and Negation Operators  
2014-12-17 - bajjabhil - mergedMerge branch 'master' of <https://github.com/sg3163/plt2014fall>  
2014-12-17 - bajjabhil - added dataprocessing example  
2014-12-17 - Abhinav Bajaj - changing Merge.jo, changing getAttrList(), it works, it works  
2014-12-17 - Abhinav Bajaj - Merging g branch 'master' of <https://github.com/sg3163/plt2014fall>  
2014-12-17 - Abhinav Bajaj - Now attrlist works in for statements too, Phew  
2014-12-16 - Sriharsha Gundappa - Custom Function and Element Access  
2014-12-16 - Sriharsha Gundappa - Comparison and Mathematical Operators  
2014-12-16 - Sriharsha Gundappa - Comparison and Mathematical Operators  
2014-12-16 - Sriharsha Gundappa - Merge branch 'master' of <https://github.com:443/sg3163/plt2014fall>  
merging  
2014-12-16 - Sriharsha Gundappa - Nested if and for  
2014-12-16 - bajjabhil - clean up  
2014-12-16 - bajjabhil - clean up  
2014-12-16 - Arpit Gupta - Merge pull request #17 from sg3163/func  
2014-12-16 - Sriharsha Gundappa - Formatted testcases

2014-12-16 - bajjabhil - cleaned up function local  
2014-12-16 - Sriharsha Gundappa - modified test files  
2014-12-16 - Sriharsha Gundappa - Adding Negative Testcases  
2014-12-16 - Abhinav Bajaj - removing func locals  
2014-12-16 - Sriharsha Gundappa - Merge branch 'master' of <https://github.com:443/sg3163/plt2014fall>  
merging  
2014-12-16 - Sriharsha Gundappa - Scope test cases  
2014-12-16 - Abhinav Bajaj - mergedMerge branch 'master' of <https://github.com/sg3163/plt2014fall>  
2014-12-16 - Abhinav Bajaj - added notype handling for operators  
2014-12-16 - Sriharsha Gundappa - Merge branch 'master' of <https://github.com:443/sg3163/plt2014fall>  
Merged function  
2014-12-16 - Sriharsha Gundappa - Resolving Confilct  
2014-12-16 - Arpit Gupta - Mergn to master g branch 'master' of <https://github.com/sg3163/plt2014fall>  
2014-12-16 - Arpit Gupta - bring runCplusCplus back to life  
2014-12-16 - Sriharsha Gundappa - Fixing test cases  
2014-12-16 - Abhinav Bajaj - added attrlist to for\_expr  
2014-12-16 - Abhinav Bajaj - In and Not IN are binary operators now  
2014-12-16 - Abhinav Bajaj - mergedMerge branch 'master' of <https://github.com/sg3163/plt2014fall>  
2014-12-16 - Abhinav Bajaj - if in can take expr in expr now  
2014-12-16 - Arpit Gupta - addinf ++ for list  
2014-12-16 - Arpit Gupta - finalisign merge.jo  
2014-12-16 - Arpit Gupta - updating merge.jo  
2014-12-16 - Arpit Gupta - Mergng element assign g branch 'master' of <https://github.com/sg3163/plt2014fall>  
2014-12-16 - Abhinav Bajaj - updated type checking for elementAccess  
2014-12-16 - Arpit Gupta - updating merge.jo  
2014-12-16 - Arpit Gupta - adding notype for element access  
2014-12-16 - Arpit Gupta - merging merge.jo  
2014-12-16 - Abhinav Bajaj - i am awesome - decl done  
2014-12-16 - Arpit Gupta - changing merge.jo  
2014-12-16 - Arpit Gupta - Merging correction in Element acess g branch 'master' of <https://github.com/sg3163/plt2014fall>  
2014-12-16 - bajjabhil - Merge pull request #16 from sg3163/decl  
2014-12-16 - Abhinav Bajaj - corrected typecheck  
2014-12-16 - Abhinav Bajaj - merge  
2014-12-16 - Arpit Gupta - Mergng g branch 'master' of <https://github.com/sg3163/plt2014fall>  
2014-12-16 - Arpit Gupta - do not remember  
2014-12-16 - Arpit Gupta - updating merge  
2014-12-16 - Arpit Gupta - do not remember  
2014-12-16 - Abhinav Bajaj - mergedMerge branch 'master' of <https://github.com/sg3163/plt2014fall>  
2014-12-16 - Abhinav Bajaj - corrected typecheck for attrList()  
2014-12-16 - Abhinav Bajaj - locals can be reassigned and decl is not required

2014-12-15 - Arpit Gupta - changing print for lists  
2014-12-15 - Arpit Gupta - merging element assign test case Merge branch 'master' of <https://github.com/sg3163/plt2014fall>  
2014-12-15 - Abhinav Bajaj - test case for ElementAssign  
2014-12-15 - Arpit Gupta - adding element access support for C++  
2014-12-15 - Abhinav Bajaj - added element assign and update getElementByOcaml for access  
2014-12-15 - Arpit Gupta - adding element access for list and json  
2014-12-15 - Abhinav Bajaj - trying to remove decl  
2014-12-15 - Chase Larson - Reverting typeStruct to not recurse into Lists  
2014-12-15 - Chase Larson - typeStruct goes into ListType.  
2014-12-15 - Chase Larson - typeStruct. Modified Read to take Json.  
2014-12-15 - Arpit Gupta - adding addToJson i.e. myJson[arpit]  
2014-12-15 - Arpit Gupta - addign merge.jo  
2014-12-15 - Sriharsha Gundappa - Terminating program execution if oCaml Compilation fails  
2014-12-15 - Arpit Gupta - fixing case for prettyPrint  
2014-12-15 - Abhinav Bajaj - updated access to handle expression in it  
2014-12-15 - Abhinav Bajaj - function has notype args, typechecking updated for notype, for loop var has notype, fixed gcd  
2014-12-14 - Abhinav Bajaj - mergedMerge branch 'master' of <https://github.com/sg3163/plt2014fall>  
2014-12-14 - Arpit Gupta - adding != operator  
2014-12-14 - Abhinav Bajaj - mergedMerge branch 'master' of <https://github.com/sg3163/plt2014fall>  
2014-12-14 - Arpit Gupta - adding != operator  
2014-12-14 - Abhinav Bajaj - mergedMerge branch 'master' of <https://github.com/sg3163/plt2014fall>  
2014-12-14 - Arpit Gupta - adding != operator  
2014-12-14 - Abhinav Bajaj - mergeMerge branch 'master' of <https://github.com/sg3163/plt2014fall>  
2014-12-14 - Arpit Gupta - adding != operator  
2014-12-14 - Abhinav Bajaj - corrected test case for type() and makeString()  
2014-12-14 - Abhinav Bajaj - main is mandatory  
2014-12-14 - Abhinav Bajaj - gerge branch 'master' of <https://github.com/sg3163/plt2014fall>  
2014-12-14 - Abhinav Bajaj - updated type()  
2014-12-14 - Abhinav Bajaj - update type()  
2014-12-14 - Arpit Gupta - Merge branch 'master' of <https://github.com/sg3163/plt2014fall>  
2014-12-14 - Arpit Gupta - updating testfiles  
2014-12-14 - Sriharsha Gundappa - Changing the expected values  
2014-12-14 - Sriharsha Gundappa - Merge branch 'master' of <https://github.com/443/sg3163/plt2014fall>  
test  
2014-12-14 - Sriharsha Gundappa - Modified GCD Program  
2014-12-14 - Sriharsha Gundappa - Modified GCD Program  
2014-12-14 - Arpit Gupta - fixed testNotOperator.exp true and False both must start with smalll  
2014-12-14 - Arpit Gupta - fixed pretty-print, now print for json always pretty printsl  
2014-12-14 - Arpit Gupta - fixed minus-operator for list in Ocaml

2014-12-14 - Sriharsha Gundappa - Adding GCD  
2014-12-14 - Sriharsha Gundappa - Refined all Testcases, added expected outputs  
2014-12-14 - Abhinav Bajaj - trying to check main  
2014-12-14 - Arpit Gupta - adding getJoType()  
2014-12-14 - Arpit Gupta - adding minus for JSON too  
2014-12-14 - Arpit Gupta - adding contains(), findIndex, minus()  
2014-12-14 - Arpit Gupta - adding find for lists , contains()  
2014-12-14 - Abhinav Bajaj - gerge branch 'master' of <https://github.com/sg3163/plt2014fall>  
2014-12-14 - Abhinav Bajaj - removed the return mainfunc  
2014-12-14 - Arpit Gupta - adding == operator  
2014-12-14 - Arpit Gupta - adding == operator for all types! Phew  
2014-12-13 - Abhinav Bajaj - gerge branch 'master' of <https://github.com/sg3163/plt2014fall>  
2014-12-13 - Abhinav Bajaj - for update, access update,attrlist update  
2014-12-13 - Chase Larson - cleaned up comments  
2014-12-13 - Chase Larson - Added write  
2014-12-13 - Chase Larson - Added read  
2014-12-13 - Arpit Gupta - adding things to TODO list  
2014-12-13 - Arpit Gupta - prettyPrint done  
2014-12-13 - Arpit Gupta - merging branch-arpit to include PrettyPrint  
2014-12-13 - Arpit Gupta - adding PrettyPrint  
2014-12-13 - Arpit Gupta - merging  
2014-12-13 - Arpit Gupta - chanign todo  
2014-12-13 - Chase Larson - Claiming File Handling  
2014-12-13 - Chase Larson - added more test cases for Logical Operators  
2014-12-13 - Abhinav Bajaj - gerge branch 'master' of <https://github.com/sg3163/plt2014fall>  
2014-12-13 - Arpit Gupta - adding makeString  
2014-12-13 - Arpit Gupta - adding makeString  
2014-12-13 - Abhinav Bajaj - gerge branch 'master' of <https://github.com/sg3163/plt2014fall>  
2014-12-13 - Abhinav Bajaj - updated code for logical, makestring  
2014-12-13 - Chase Larson - Added Logical operators  
2014-12-12 - Chase Larson - claiming item from TODO list  
2014-12-12 - Chase Larson - Merge branch 'cpplib'  
2014-12-12 - Chase Larson - Comparison Operators now Return BoolType  
2014-12-12 - Arpit Gupta - changing TODO  
2014-12-12 - Chase Larson - Comparison Operators for NumType  
2014-12-12 - Chase Larson - Added Mod operator  
2014-12-12 - Abhinav Bajaj - update operators, added && and ||, updated typecheck messages  
2014-12-12 - Abhinav Bajaj - mergedeerge branch 'master' of <https://github.com/sg3163/plt2014fall>  
2014-12-12 - Abhinav Bajaj - trying new add  
2014-12-12 - Chase Larson - Fixed error with comma  
2014-12-12 - Chase Larson - Mathematical operators

2014-12-11 - Chase Larson - Merge branch 'master' of <https://github.com/sg3163/plt2014fall>

2014-12-11 - Sriharsha Gundappa - Type Checking on operators

2014-12-11 - Sriharsha Gundappa - Modified test script

2014-12-11 - Chase Larson - Merge branch 'master' of <https://github.com/sg3163/plt2014fall>

2014-12-11 - Chase Larson - Added +=, -=, \*=, /=, String Concat, and getBoolValue

2014-12-10 - Sriharsha Gundappa - Merge branch 'harshadec4'

2014-12-10 - Abhinav Bajaj - updated pending items

2014-12-10 - Abhinav Bajaj - added IfIn support

2014-12-09 - Abhinav Bajaj - added String concat test file

2014-12-09 - Abhinav Bajaj - adding string concat and mergedMerge branch 'master' of <https://github.com/sg3163/plt2014fall>

2014-12-09 - Abhinav Bajaj - added String concatenation

2014-12-09 - Sriharsha Gundappa - merging

2014-12-09 - Abhinav Bajaj - merged

2014-12-09 - Sriharsha Gundappa - Adding Pretty Print Json testcase and updated pending items for c++

2014-12-09 - Abhinav Bajaj - not operator done

2014-12-09 - Arpit Gupta - Update PendingItems.txt

2014-12-09 - sg3163 - Merge pull request #12 from sg3163/harshadec4

2014-12-09 - Sriharsha Gundappa - Adding Type and MakeString function code

2014-12-09 - Sriharsha Gundappa - Adding Type and MakeString function code

2014-12-09 - Abhinav Bajaj - Item -conv exr to Bool Type when inside If- done, small fixes in type checking in operators

2014-12-08 - sg3163 - Merge pull request #11 from sg3163/harshadec4

2014-12-08 - Sriharsha Gundappa - Added parenthesis to Print

2014-12-08 - Sriharsha Gundappa - Added Null Code, Null and Bool to List and Json

2014-12-08 - Arpit Gupta - adding to do list

2014-12-05 - Arpit Gupta - adding List concatenation

2014-12-05 - Arpit Gupta - changing for loop and adding print via iterator in cPlusPlusCompiler.cpp

2014-12-05 - Arpit Gupta - changing for loop and adding print via iterator in cPlusPlusCompiler.cpp

2014-12-04 - Arpit Gupta - changing for loop in jp.ml and changing loop over list in cPlusPlusCompiler.h

2014-12-04 - Arpit Gupta - adding ListType parser Merge branch 'arpit'

2014-12-04 - Arpit Gupta - adding ListType parser

2014-12-04 - sg3163 - Merge pull request #10 from sg3163/harshadec4

2014-12-04 - Sriharsha Gundappa - modified Print

2014-12-04 - Sriharsha Gundappa - Pending Items

2014-12-04 - Sriharsha Gundappa - Negative test cases

2014-12-04 - Arpit Gupta - Adding foe loop logic in C++ code Merge branch 'master' of <https://github.com/sg3163/plt2014fall>

2014-12-04 - Arpit Gupta - adding For loop logic in C++ code

2014-12-04 - Sriharsha Gundappa - Merge branch 'master' of <https://github.com:443/sg3163/plt2014fall>

2014-12-04 - Sriharsha Gundappa - Removing compilation warning from c++

2014-12-04 - Arpit Gupta - "merging arpit branch. functionality for element add and element access is now available " Merge branch 'arpit'

2014-12-04 - Arpit Gupta - adding element access and element add functionalities

2014-12-04 - Sriharsha Gundappa - Resolving ocaml Compiler warning

2014-12-04 - Sriharsha Gundappa - Resolving ocaml Compiler warning

2014-12-04 - Arpit Gupta - moving functions around

2014-12-03 - Arpit Gupta - Merging branch arpit, with JsonType, ListType Merge branch 'master' into arpit

2014-12-03 - Arpit Gupta - Adding List to CustType

2014-12-03 - Abhinav Bajaj - fixed issues

2014-12-03 - bajjabhil - Merge pull request #8 from sg3163/abhinav

2014-12-03 - Abhinav Bajaj - Merge branch 'abhinav' of <https://github.com/sg3163/plt2014fall> into abhinav

2014-12-03 - Abhinav Bajaj - access updated, shift reduce resolved

2014-12-03 - bajjabhil - Merge pull request #7 from sg3163/master

2014-12-03 - Arpit Gupta - converting form library json type to our own JsonType

2014-12-03 - Abhinav Bajaj - local

2014-12-03 - bajjabhil - Merge pull request #6 from sg3163/abhinav

2014-12-03 - Abhinav Bajaj - Comparison operator tests

2014-12-03 - Abhinav Bajaj - correction of typecheck and mod

2014-12-03 - sg3163 - Merge pull request #5 from sg3163/harshadec3

2014-12-03 - Sriharsha Gundappa - modified clean all

2014-12-03 - bajjabhil - Merge pull request #4 from sg3163/abhinav

2014-12-03 - Abhinav Bajaj - added comminus and complus

2014-12-03 - Sriharsha Gundappa - Adding Testcases for Operators, Conditions etc

2014-12-03 - Sriharsha Gundappa - Addition Testcase

2014-12-03 - Sriharsha Gundappa - Merge branch 'master' of <https://github.com:443/sg3163/plt2014fall>

2014-12-03 - sg3163 - Merge pull request #3 from sg3163/harshanov30

2014-12-03 - Abhinav Bajaj - removed unnecessary files

2014-12-02 - Arpit Gupta - adding logic to loop over JSON objects

2014-12-01 - Sriharsha Gundappa - Merge branch 'harshanov30'

2014-12-01 - Sriharsha Gundappa - Adding Attribute List

2014-12-01 - Sriharsha Gundappa - Adding Attribute List

2014-11-30 - Sriharsha Gundappa - Merge branch 'harshanov30'

2014-11-30 - Sriharsha Gundappa - TypeStruct code

2014-11-30 - Sriharsha Gundappa - TypeStruct code

2014-11-30 - Sriharsha Gundappa - Merge branch 'master' of <https://github.com:443/sg3163/plt2014fall>

2014-11-30 - Sriharsha Gundappa - removing .o files

2014-11-30 - PRANAV BHALLA - merging wjth master Merge branch 'master' of <https://github.com/sg3163/plt2014fall> into arpit

2014-11-30 - PRANAV BHALLA - merging with head ierge branch 'master' of <https://github.com/sg3163/plt2014fall>

2014-11-30 - Sriharsha Gundappa - Started with Merge function  
2014-11-30 - Sriharsha Gundappa - added testcases for Json and List parsing  
2014-11-30 - PRANAV BHALLA - Merge branch 'arpit'  
2014-11-30 - PRANAV BHALLA - no change  
2014-11-30 - Sriharsha Gundappa - merging with arpit2- for loop code  
2014-11-29 - Sriharsha Gundappa - Json and List parsing  
2014-11-28 - Sriharsha Gundappa - List and Json Element Access  
2014-11-27 - [arpitg1991@gmail.com](mailto:arpitg1991@gmail.com) - removing str.cma  
2014-11-26 - [arpitg1991@gmail.com](mailto:arpitg1991@gmail.com) - removing output files  
2014-11-26 - [arpitg1991@gmail.com](mailto:arpitg1991@gmail.com) - removing putput files  
2014-11-26 - [arpitg1991@gmail.com](mailto:arpitg1991@gmail.com) - resolving merge conflicts  
2014-11-26 - [arpitg1991@gmail.com](mailto:arpitg1991@gmail.com) - resolving merge conflicts  
2014-11-26 - [arpitg1991@gmail.com](mailto:arpitg1991@gmail.com) - adding for  
2014-11-26 - Sriharsha Gundappa - Test Files for Function calling  
2014-11-26 - Sriharsha Gundappa - Merge branch 'master' of <https://github.com:443/sg3163/plt2014fall>  
2014-11-26 - Sriharsha Gundappa - Individual test file to run one file  
2014-11-26 - Sriharsha Gundappa - Function Call from Function, Statement Reverse  
2014-11-26 - bajjabhil - Merge pull request #2 from bajjabhil/master  
2014-11-26 - Abhinav Bajaj - updated assign output  
2014-11-26 - Abhinav Bajaj - done ifelse  
2014-11-26 - [arpitg1991@gmail.com](mailto:arpitg1991@gmail.com) - Revert "cleaning"  
2014-11-26 - [arpitg1991@gmail.com](mailto:arpitg1991@gmail.com) - cleaning  
2014-11-25 - [arpitg1991@gmail.com](mailto:arpitg1991@gmail.com) - adding code for for  
2014-11-24 - Sriharsha Gundappa - Complete Test function to check print  
2014-11-24 - Sriharsha Gundappa - Print from Main working with complete test case  
2014-11-24 - Abhinav Bajaj - merge  
2014-11-24 - Abhinav Bajaj - updated testall  
2014-11-24 - Abhinav Bajaj - dded test jo file  
2014-11-24 - Abhinav Bajaj - gerge branch 'master' of <https://github.com/sg3163/plt2014fall>  
2014-11-24 - Abhinav Bajaj - dding test jo  
2014-11-24 - [arpitg1991@gmail.com](mailto:arpitg1991@gmail.com) - changing string parsing  
2014-11-24 - Abhinav Bajaj - added ifelse  
2014-11-24 - Sriharsha Gundappa - removed cpp  
2014-11-24 - Sriharsha Gundappa - main function  
2014-11-24 - Chase Larson - Added Parsing to/from JSON and String  
2014-11-23 - Sriharsha Gundappa - Merge branch 'master' of <https://github.com:443/sg3163/plt2014fall>  
test  
2014-11-23 - Abhinav Bajaj - print only takes customtype now  
2014-11-23 - Sriharsha Gundappa - Merge branch 'master' of <https://github.com:443/sg3163/plt2014fall>  
merging  
2014-11-23 - Sriharsha Gundappa - func change

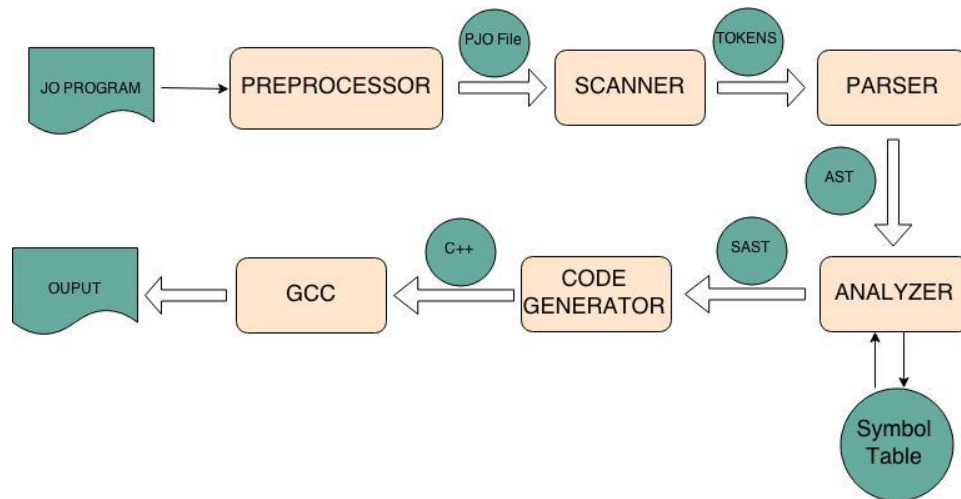
2014-11-23 - [arpitg1991@gmail.com](mailto:arpitg1991@gmail.com) - Merge branch 'master' of <https://github.com/sg3163/plt2014fall>  
2014-11-23 - [arpitg1991@gmail.com](mailto:arpitg1991@gmail.com) - adding print mapping from ocaml to c++  
2014-11-23 - Sriharsha Gundappa - Merge branch 'master' of <https://github.com:443/sg3163/plt2014fall>  
merge Reqd  
2014-11-23 - Sriharsha Gundappa - Function parsing, Json quote regex replace  
2014-11-23 - Sriharsha Gundappa - Function parsing, Json quote regex replace  
2014-11-23 - [arpitg1991@gmail.com](mailto:arpitg1991@gmail.com) - Merge branch 'master' of <https://github.com/sg3163/plt2014fall>  
2014-11-23 - [arpitg1991@gmail.com](mailto:arpitg1991@gmail.com) - changing to static methods  
2014-11-23 - bajjabhil - Merge pull request #1 from bajjabhil/ifelse  
2014-11-23 - Abhinav Bajaj - space removed after namespace decl  
2014-11-23 - Sriharsha Gundappa - removing semicolon and adding more test case  
2014-11-23 - Sriharsha Gundappa - print function  
2014-11-23 - Sriharsha Gundappa - Merge branch 'master' of <https://github.com:443/sg3163/plt2014fall>  
2014-11-23 - Sriharsha Gundappa - Adding function  
2014-11-23 - [arpitg1991@gmail.com](mailto:arpitg1991@gmail.com) - adding print to strType and Numtype  
2014-11-22 - Sriharsha Gundappa - Adding Bool, Json and List types  
2014-11-22 - [arpitg1991@gmail.com](mailto:arpitg1991@gmail.com) - spitting cpp code for int string  
2014-11-22 - [arpitg1991@gmail.com](mailto:arpitg1991@gmail.com) - adding running commands and changing jo.ml  
2014-11-22 - [arpitg1991@gmail.com](mailto:arpitg1991@gmail.com) - adding running commands, and changing jo.ml"  
2014-11-21 - Abhinav Bajaj - somethign is working  
2014-11-20 - Sriharsha Gundappa - typecheck.ml Nov 20th 10:00 PM  
2014-11-20 - Sriharsha Gundappa - Working version of Ast  
2014-11-20 - [arpitg1991@gmail.com](mailto:arpitg1991@gmail.com) - addiing parse code  
2014-11-19 - arpitg1991 - Revert "commiting 2nd style"  
2014-11-19 - Pranav Bhalla - commiting 2nd style  
2014-11-19 - arpitg1991 - modifying parser  
2014-11-19 - arpitg1991 - changing scanner  
2014-11-16 - Sriharsha Gundappa - Merge branch 'master' of <https://github.com:443/sg3163/plt2014fall>  
2014-11-16 - arpitg1991 - cpplus  
2014-11-16 - Sriharsha Gundappa - Parser for scanner.mll  
2014-11-16 - arpitg1991 - c++compiler  
2014-11-16 - arpitg1991 - removing duplicate files  
2014-11-16 - arpitg1991 - adding c plus plus compiler stuff  
2014-11-16 - arpitg1991 - a  
2014-11-16 - guptaar - adding cplusplusCode  
2014-11-15 - Sriharsha Gundappa - Full Scanner  
2014-11-12 - Sriharsha Gundappa - Test All and Clean All test script.  
2014-11-12 - Sriharsha Gundappa - Adding scanner code  
2014-11-12 - Sriharsha Gundappa - Reverting back to c++ from Java  
2014-11-10 - Sriharsha Gundappa - Java Code generation, json and List declaration  
2014-11-01 - Abhinav Bajaj - added stmts



2014-11-01 - Abhinav Bajaj - preprocess added  
2014-11-01 - Abhinav Bajaj - added gitignore  
2014-11-01 - Abhinav Bajaj - added makefile, shell script  
2014-11-01 - Abhi - initial commit  
2014-09-15 - sg3163 - Initial commit

## 5. Architectural Design

The architecture of the JO language compiler is depicted in the below diagram



### 5.1 PreProcessor

The preprocessor reads the input JO program and adds syntactic details such as braces and semicolons replacing the block ends represented by "end" as well. The output of preprocessor is <input\_file\_name>.pjo file.

### 5.2 Scanner

The scanner takes in a .pjo program file and outputs tokens according to the lexical rules of the JO language.

### 5.3 Parser

The parser takes in the tokens from the scanner and uses our grammatical rules to produce an abstract syntax tree.

### 5.4 Analyzer

The analyzer file takes in the abstract syntax tree as given by the parser and performs semantic checks, outputting a typed semantic abstract syntax tree. This file uses a symbol table in order to keep track of global and local variables, making sure that the program is in accordance to our scoping rules. Analyzer also checks that the arguments of operators are valid, that all variables are declared and initialized when they need to be, that two variables are not declared with the same name and that functions return a value. Only the main function is allowed not to return anything. If any input program does not pass these checks then the compiler will raise an error and compilation will halt.

### 5.5 Symbol Table

The symbol table provides the functions for maintaining the environment during the program execution. It provides the below functions to maintain the environment:

- add\_global: makes a global variable visible in the scope of the entire program.
- update\_global: updates a global variable with new type information.

- `add_function`: makes a function name visible in the scope of the entire program.
- `add_local`: adds a local variable to the current scope only.
- `update_local`: updates a local variable with new type information.
- `find_function`: used to check if a particular function name is visible in the current scope.
- `find_variable`: used to check if a particular variable id is visible in the current scope

## 5.6 Code Generation

Code generation takes the input semantic abstract syntax tree and creates the C++ code by creating custom objects that represent the datatypes on C++ side. The code generation uses the C++ library that provides the basic functions corresponding to the operations supported in JO language. The C++ library also uses an open source code that helps in parsing the json standard format on runtime. There are base classes that the custom type objects inherit.

## 5.7 Team Contribution

While every team member worked on every component in some fashion (either as an author or as a debugger), the primary responsibility breakdown for each component is given above in section 4.3

## 6. Test Plan

### 6.1 Test Suite Automation

Shell Script is written to automate the code testing. There are 3 different test suites slightly varying depending on the purpose.

All three test suites executes below steps

- i. Preprocess .jo file to .pjo file using makefile – **MakePreProc**
- ii. Scan, Parse, Check Semantics, output C++ executable is done using makefile **Makefile**
- iii. Execute C++ and output using makefile - **MakeFileCPP**

Different test suites are

- i. **runjo.sh** – To run individual test-case and print out the final program output on screen. It also persists intermediate files like .pjo .cpp .o .output to assist in debugging. It removes the intermediate files upon next run.  
If there is any parse error then script displays parse error  
If there is any type checking error then script displays the type checking exception  
If there is any error while running c++ code then script displays corresponding c++ error  
If program runs successfully then script prints out the output.  
**Execution** - ./runjo.sh testfiles/Merge.jo
- ii. **test.sh** – To run individual test-case and compare it against the expected value. It goes through all the steps as done by runjo.sh and does an extra step of comparison with expected value. Script spits out error file in response to mismatch between expected out put and actual output  
**Execution** - ./test.sh testfiles/Merge.jo
- iii. **testall.sh** – It executes all the steps as of test.sh and does the extra step of running all the test-cases in test-file directory.  
**Execution** - ./testall.sh

### 6.2 Test Case Selection

We have tried to make unit test cases for all features of the language. We have covered both Positive and Negative test cases. Most of the positive test cases are rich in feature, in the sense that each test class covers multiple test varieties. Below are the basic varieties of test cases created

Arithmetic/Mathematic (++ , -- , \*\* , // , > , >= , < , <= , % % )  
Creation of Types (List, Json, Number, Null, String, Bool)  
Dynamic Type Casting  
Comment (/\* \*/)  
Comparison (== , !=)  
Logical (|| , && , !)  
Concatenation (+)  
String Concatenation (++)  
Negation (-)  
Function Calls

- Scope Check
- In and Not In Operators
- If Else
- For loops
- Json and List Element Access
- Json and List Element Assignment
- File Read Function
- File Write Function
- MakeString Function
- Print Function
- JsonPrettyPrint
- TypeStruct Function
- Type Function
- Complex test cases (GCD, Merge)

### 6.3 Test Suites – 6.3.1 - testall.sh

```
#!/bin/sh

if [ ! -f "./preprocessor" ]; then
    make -f MakePreProc >> make.log
fi

if [ ! -f "./jo" ]; then
    make -f Makefile >> make.log
fi

# jo executable
JO="./jo"

# preprocessor executable
PRE="./preprocessor"
TEST_BASE="testfiles"

# Compare <outfile> <reffile> <difffile>
# Compares the outfile with reffile. Differences, if any, written to difffile
Compare() {
    difference=$(diff -b $1 $2)
    echo $difference
    if [ "$difference" != "" ]; then
        echo $difference > $3
    fi
}
```

```

function compileAndRun() {
    basename=`echo $1 | sed 's/.*\\\///
                          s/.jo//'\`
    echo "Running file $basename"
    reffile=`echo $1 | sed 's/.jo$//'\`
    prepfile=$TEST_BASE/$basename'.pjo'
    #echo $prepfile
    basedir="`echo $1 | sed 's/\/[^\/]*$//'\`/"

# gets the path of the test output file
    testoutput=`echo ${basedir}test_outputs/$basename.c.out`

    echo "Preprocessing '$1'"
    $PRE $1 $prepfile && echo "Preprocessor for $1 succeeded"

    echo "Compiling '$prepfile'"
    if [ ! -f $prepfile ]; then
        echo "$prepfile does not exist"
    return
    fi

# converting from JO to C++
    $JO $prepfile > "${reffile}.cpp" && echo "Ocaml to C++ of $1 succeeded"

    if [ ! -s ${reffile}.cpp ] ; then
        echo "Error in Compilation of ${reffile}"
        return
    fi ;

# compiling the C++ file
    if [ -f "${reffile}.cpp" ]; then
        make inputfile=$basename -f MakeFileCPP
    else
        echo "Compiling $1 failed"
        return
    fi

    # running the binary
    if [ -f "${reffile}.out" ]; then
        eval ${reffile}.out >> ${reffile}.output
        Compare ${reffile}.output ${reffile}.exp ${reffile}.error

        rm -rf ${reffile}.fdlp
        rm -rf ${reffile}.pjo
        rm -rf ${reffile}.cpp
        rm -rf ${reffile}.out
        rm -rf ${reffile}.o
        rm -rf ${reffile}.output
        echo "ran $1 successfully"
    else
        echo "C++ to binary of ${reffile}.cpp failed"
    fi
}
files=$TEST_BASE/*.jo

for file in $files
do
    compileAndRun $file
done

```

### 6.3.2 Makefile1 – MakeCPP

```
CC = gcc
CFLAGS = -g -Wall

.PHONY: default
default: preproc

.PHONY: all
all: clean preproc

.PHONY: clean
clean:
    rm -rf *.log *.o *~ a.out* core preprocessor *.pjo

.PHONY: preproc
preproc: preprocessor.c
    $(CC) -o preprocessor preprocessor.c
```

### 6.3.3 Makefile2 – MakeFile

```
OBJS = parser.cmo scanner.cmo symboltable.cmo typecheck.cmo Str.cma
jo.cmo

.PHONY: default
default: jo

.PHONY: all
all: clean jo

jo: $(OBJS)
    ocamlc -o jo $(OBJS)

scanner.ml: scanner.mll
    ocamllex scanner.mll

parser.ml parser.mli: parser.mly
    ocamlyacc parser.mly

%.cmo: %.ml
    ocamlc -c $<

%.cmi: %.mli
    ocamlc -c $<

.PHONY: clean
clean:
    rm -rf jo parser.ml parser.mli scanner.ml *.cmo *.cmi *.log *.o
    *~ a.out* core

# Generated by ocamldep *.ml *.mli
analyzer.cmo: symboltable.cmo ast.cmi sast.cmi
analyzer.cmx: symboltable.cmx ast.cmx sast.cmx
jo.cmo: scanner.cmo parser.cmi ast.cmi Str.cma
jo.cmx: scanner.cmx parser.cmx ast.cmi
parser.cmo: ast.cmi parser.cmi
parser.cmx: ast.cmi parser.cmi
scanner.cmo: parser.cmi
scanner.cmx: parser.cmx
parser.cmi: ast.cmi
Str.cma:
```



### 6.3.4 Makefile3 – MakeFileCPP

```
CC=g++

CFLAGS=-c -w
LFLAGS=-lm

SOURCES=testfiles/$(inputfile).cpp cpp/JSON.cpp cpp/JSONValue.cpp
HEADERS=cpp/JSON.h cpp/JSONValue.h cpp/cPlusPlusCompiler.h

OBJECTS=$(SOURCES:.cpp=.o)

EXECUTABLE=testfiles/$(inputfile).out

all: $(SOURCES) $(EXECUTABLE)

$(EXECUTABLE): $(OBJECTS)
    $(CC) $(LFLAGS) $(OBJECTS) -o $@

.cpp.o:
    $(CC) $(CFLAGS) $< -o $@

clean:
    rm -f $(OBJECTS) $(EXECUTABLE) *~
```

## 6.4 Example Test

### 6.4.1 Merge.jo

```

func Merge (a,b)
    c = #{ }#

    for attr in a.attrList()
        if ( attr in b.attrList()
            if (type(a[attr]) == "Json" && type(b[attr]) == "Json")

                c[attr] = Merge(a[attr],b[attr])
            else
                if (type(a[attr]) == "List" && type(b[attr]) == "List")
                    c[attr] = a[attr] ++ b[attr]
                else
                    c[attr] = a[attr] + b[attr]
                end
            end
        end
    else
        c[attr] = a[attr]
    end
end

for attr in b.attrList()
    if ( attr not in a.attrList() )
        c[attr] = b[attr]
    end
end

return c
end

func MergeUtil (a,b)
    if ( type(a) != "Json" || type(b) != "Json")
        print("Both the arguments must be JSON")
        return null
    end
    return Merge(a,b)
end

func main ()

a = #{"name":"harsha", "innerJson":{"sub":"PLT","mark":[5,6,7]}}#
b = #{"name":"arpit", "innerJson":{"sub":"OS","mark":[7,8,9]}}#
c = MergeUtil (a,b)
print (c)
end

```

**C++ Output (Merge.cpp)**

```

#include <iostream>
#include "../cpp/cPlusPlusCompiler.h"
using namespace std;

CustType* Merge(CustType* a, CustType* b)
{ CustType* c = CustType::parse("{}","JSON");
for (vector<CustType*> :: iterator loopVarattr = (a-> getAttrList()->getListBegin()); loopVarattr !=
(a-> getAttrList()->getListEnd(); ++loopVarattr) {
    CustType* attr = *loopVarattr;
    { if ((b-> getAttrList()->contains(attr))->getBoolValue())
    { if ((*((a-> getElementByOcaml(attr))->getJoType())) == *(CustType::parse("Json","STRING")))
    && *((b-> getElementByOcaml(attr))->getJoType()) == *(CustType::parse("Json","STRING")))-
    >getBoolValue())
    { c->addByKey(attr,Merge(a-> getElementByOcaml(attr), b-> getElementByOcaml(attr)));
    }else {
    if ((*((a-> getElementByOcaml(attr))->getJoType())) == *(CustType::parse("List","STRING"))) &&
    *((b-> getElementByOcaml(attr))->getJoType()) == *(CustType::parse("List","STRING")))-
    >getBoolValue())
    {
    c->addByKey(attr,CustType::add(a-> getElementByOcaml(attr),b-> getElementByOcaml(attr)));

    }else
    {
    c->addByKey(attr,CustType::concat(a-> getElementByOcaml(attr),b-> getElementByOcaml(attr)));

    } } }else
    {
    c->addByKey(attr,a-> getElementByOcaml(attr));

    } } }
for (vector<CustType*> :: iterator loopVarattr = (b-> getAttrList()->getListBegin()); loopVarattr !=
(b-> getAttrList()->getListEnd(); ++loopVarattr) {
    CustType* attr = *loopVarattr;
    {
    if (!(a-> getAttrList()->contains(attr)))->getBoolValue())
    {
    c->addByKey(attr,b-> getElementByOcaml(attr));
    } } }
return c;}

CustType* MergeUtil(CustType* a, CustType* b)
{ if ((*((a->getJoType()) != *(CustType::parse("Json","STRING"))) || *((b->getJoType()) !=
*(CustType::parse("Json","STRING"))))->getBoolValue())
{ CustType::print(CustType::parse("Both the arguments must be JSON","STRING"));
return CustType::parse("null","NULL");
} return Merge(a, b);}

int main()
{ CustType* a =
CustType::parse("{\"name\":\"harsha\", \"innerJson\":{ \"sub\": \"PLT\", \"mark\": [5,6,7] } }", "JSON");
CustType* b =
CustType::parse("{\"name\":\"arpit\", \"innerJson\":{ \"sub\": \"OS\", \"mark\": [7,8,9] } }", "JSON");
CustType* c = MergeUtil(a, b);
CustType::print(c); }

```

## 6.4.2 GCD Program (GCD.jo)

```
a = 45
b = 81

func findGCD(a,b)
    if(b == 0)
        return a
    end
    return findGCD(b, a%%b)
end

func main ()
    print( findGCD(a,b) )
end
```

## GCD C++ code (GCD.cpp)

```
#include <iostream>
#include "../cpp/cPlusPlusCompiler.h"
using namespace std;

CustType* b = CustType::parse("81","NUMBER");

CustType* a = CustType::parse("45","NUMBER");

CustType* findGCD(CustType* a, CustType* b)
{
    if ((*b) == *(CustType::parse("0","NUMBER"))->getBoolValue())
    {
        return a;
    } return findGCD(b, CustType::mod(a,b));}

int main()
{
    CustType::print(findGCD(a, b));
}
```

## **7. Lessons Learned**

### **7.1 Sriharsha Gundappa**

I learnt a good deal of OCaml coding from the project. This is the first ever functional program I used in my life, so it took a while to get used to writing program in functional language. But definitely it is a good addition to your knowledge base. It provides you with a different perspective of solving problems. We planned at every step of project execution, divided tasks and conquered it. We continually reviewed the status of tasks and revised to match the end product, this helped in avoiding surprises in the end. Finally, It felt great to finish project in a short period of time and in an unknown language

Suggestion for teams - First and foremost, select motivated people in your team or get into a team of motivated people. Project gets harder or easier depending on the team you are in. Use a good version control system like git to host your files, get acquainted with it, create as much branches as required. There will always possibility of code getting messed up and you spending lot of time just debugging issues related to merging code. It helped a lot when working offline and also to merge conflicts. During the project execution, I suggest following Regex – (Plan, Code, Revise)\* Code Cleanup Submit.

### **7.2 Chase Larson**

Prioritize. When initially designing the language, have a clear idea of why you want to build the language you propose and focus on one thing you want it to be able to do better than any other language. This will help keep the design from including extraneous features. If something is not important to your main priority, then don't spend too much time or energy on it.

When developing, having a uniform coding style can really help speed up the time it takes to understand and/or debug code that someone else originally wrote.

### **7.3 Abhinav Bajaj**

The JO Language project is my first experience with a functional programming language. OCaml is very power language with great compile time capabilities. Once you learn the language, coding in it is precise and fast. Also this is the first time I have worked on creating a new language itself and understanding the details on how the compilers works. Apart from learning OCaml and compilers, I have learned team management and time management skills as well. One of the most challenging part of the project is managing the team coordinating with everyone and keeping people motivated to contribute and work. Also as the System architect, it was my responsibility to lay down the components of the compiler and taking final call on deciding the output language(C++ in our case) and how we map the constructs of JO language to C++.

In general, my advice for future students is to start as early as possible. There are two major things to focus for the project. First is to understand how all components of the project like Make files, preprocessor, parser, scanner, ast, sast and code generator, running script fit into a single flow. Second is to learning on how to do type checking and other book keeping activities. So, get the HelloWorld program running asap. This will make sure you understand the first part. Only after this stage, each team member can start working individually. Professor suggests to finish the scanner and parser first but I have a different opinion here. Focus on creating a scanner and parser that does 50-60% of the job and then make the HelloWorld and GCD work. After this state, keep on iterating and adding new features of the language. Talk to TAs and Professor for guidance. All the best. It would be fun.

## 7.4 Arpit Gupta

It is better to design a language that does, one thing better than every other language, rather than having a language that tries to do everything, and does nothing extraordinarily. For most part of the project, we were trying to do the latter. It was only during the last week that we realized that is perfectly fine to not support a million features that other languages do.

The most important lesson I learnt was never underestimate the power of recursion. I knew Json has recursive structure, but only once we started writing the code, did I fully realize that it is nothing but recursive structure. It also is the most beautiful thing, because at the end of the day, you are only dealing with numbers, string and booleans, everything else is just formatting.

One thing we did right was, to assign back-up roles, so we had 2 people for each role. Also for all the actual coding part, we had 2 people working on c++ and 2 on oCaml, so whenever somebody need to take some time-off, we could still keep the work going, rather than being stuck because the oCaml guy is not available. For most part of the project, we were trying to do the latter. It was only during the last week that we realized that is perfectly fine to not support a million features that other languages do.

## 8. Appendix

### JSON.cpp

/\*

\* File JSON.cpp part of the SimpleJSON Library - <http://mjpa.in/json>

\*

\* Copyright (C) 2010 Mike Anchor

\*

\* Permission is hereby granted, free of charge, to any person obtaining a copy

\* of this software and associated documentation files (the "Software"), to deal

\* in the Software without restriction, including without limitation the rights

\* to use, copy, modify, merge, publish, distribute, sublicense, and/or sell

\* copies of the Software, and to permit persons to whom the Software is

\* furnished to do so, subject to the following conditions:

\*

\* The above copyright notice and this permission notice shall be included in

\* all copies or substantial portions of the Software.

\*

\* THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS  
OR

\* IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,

\* FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT  
SHALL THE

\* AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR  
OTHER

\* LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING  
FROM,

\* OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER  
DEALINGS IN

\* THE SOFTWARE.

\*/

```
#include "JSON.h"
```

```
/**
```

```
* Blocks off the public constructor
```

```
*
```

```
* @access private
```

```
*
```

```
*/
```

```
JSON::JSON()
```

```
{
```

```
}
```

```
/**
```

```
* Parses a complete JSON encoded string
```

```
* This is just a wrapper around the UNICODE Parse().
```

```
*
```

```
* @access public
```

```
*
```

```
* @param char* data The JSON text
```

```
*
```

```
* @return JSONValue* Returns a JSON Value representing the root, or NULL on error
```

```
*/
```

```
JSONValue *JSON::Parse(const char *data)
```

```
{
```

```
    size_t length = strlen(data) + 1;
```

```
    wchar_t *w_data = (wchar_t*)malloc(length * sizeof(wchar_t));
```



```

#if defined(WIN32) && !defined(__GNUC__)
    size_t ret_value = 0;
    if (mbstowcs_s(&ret_value, w_data, length, data, length) != 0)
    {
        free(w_data);
        return NULL;
    }
#elif defined(ANDROID)
    // mbstowcs seems to misbehave on android
    for(size_t i = 0; i<length; i++)
        w_data[i] = (wchar_t)data[i];
#else
    if (mbstowcs(w_data, data, length) == (size_t)-1)
    {
        free(w_data);
        return NULL;
    }
#endif

JSONValue *value = JSON::Parse(w_data);
free(w_data);
return value;
}

/**
 * Parses a complete JSON encoded string (UNICODE input version)
 *
 * @access public
 */

```

```

* @param wchar_t* data The JSON text
*
* @return JSONValue* Returns a JSON Value representing the root, or NULL on error
*/
JSONValue *JSON::Parse(const wchar_t *data)
{
    // Skip any preceding whitespace, end of data = no JSON = fail
    if (!SkipWhitespace(&data))
        return NULL;

    // We need the start of a value here now...
    JSONValue *value = JSONValue::Parse(&data);
    if (value == NULL)
        return NULL;

    // Can be white space now and should be at the end of the string then...
    if (SkipWhitespace(&data))
    {
        delete value;
        return NULL;
    }

    // We're now at the end of the string
    return value;
}

/**
* Turns the passed in JSONValue into a JSON encode string

```

```

*
* @access public
*
* @param JSONValue* value The root value
*
* @return std::wstring Returns a JSON encoded string representation of the given value
*/
std::wstring JSON::Stringify(const JSONValue *value)
{
    if (value != NULL)
        return value->Stringify();
    else
        return L"";
}

/**
* Skips over any whitespace characters (space, tab, \r or \n) defined by the JSON spec
*
* @access protected
*
* @param wchar_t** data Pointer to a wchar_t* that contains the JSON text
*
* @return bool Returns true if there is more data, or false if the end of the text was reached
*/
bool JSON::SkipWhitespace(const wchar_t **data)
{
    while (**data != 0 && (**data == L' ' || **data == L'\t' || **data == L'\r' || **data == L'\n'))
        (*data)++;
}

```

```

    return **data != 0;
}

/**
 * Extracts a JSON String as defined by the spec - "<some chars>"
 * Any escaped characters are swapped out for their unescaped values
 *
 * @access protected
 *
 * @param wchar_t** data Pointer to a wchar_t* that contains the JSON text
 * @param std::wstring& str Reference to a std::wstring to receive the extracted string
 *
 * @return bool Returns true on success, false on failure
 */
bool JSON::ExtractString(const wchar_t **data, std::wstring &str)
{
    str = L"";

    while (**data != 0)
    {
        // Save the char so we can change it if need be
        wchar_t next_char = **data;

        // Escaping something?
        if (next_char == L'\\')
        {
            // Move over the escape char
            (*data)++;

```

```

// Deal with the escaped char
switch (**data)
{
    case L'": next_char = L'"; break;
    case L'\\': next_char = L'\\'; break;
    case L'/': next_char = L'/'; break;
    case L'b': next_char = L'\b'; break;
    case L'f': next_char = L'\f'; break;
    case L'n': next_char = L'\n'; break;
    case L'r': next_char = L'\r'; break;
    case L't': next_char = L'\t'; break;
    case L'u':
    {
        // We need 5 chars (4 hex + the 'u') or its not valid
        if (!simplejson_wcsnlen(*data, 5))
            return false;

        // Deal with the chars
        next_char = 0;
        for (int i = 0; i < 4; i++)
        {
            // Do it first to move off the 'u' and leave us on the
            // final hex digit as we move on by one later on
            (*data)++;

            next_char <<= 4;

            // Parse the hex digit
            if (**data >= '0' && **data <= '9')
                next_char |= (**data - '0');
        }
    }
}

```

```

else if (**data >= 'A' && **data <= 'F')
    next_char |= (10 + (**data - 'A'));
else if (**data >= 'a' && **data <= 'f')
    next_char |= (10 + (**data - 'a'));
else
{
    // Invalid hex digit = invalid JSON
    return false;
}
}
break;
}

// By the spec, only the above cases are allowed
default:
    return false;
}
}

// End of the string?
else if (next_char == L'')
{
    (*data)++;
    str.reserve(); // Remove unused capacity
    return true;
}

// Disallowed char?
else if (next_char < L' ' && next_char != L'\t')
{

```

```

        // SPEC Violation: Allow tabs due to real world cases
        return false;
    }

    // Add the next char
    str += next_char;

    // Move on
    (*data)++;
}

// If we're here, the string ended incorrectly
return false;
}

/**
 * Parses some text as though it is an integer
 *
 * @access protected
 *
 * @param wchar_t** data Pointer to a wchar_t* that contains the JSON text
 *
 * @return double Returns the double value of the number found
 */
double JSON::ParseInt(const wchar_t **data)
{
    double integer = 0;
    while (**data != 0 && **data >= '0' && **data <= '9')
        integer = integer * 10 + ((*data)++ - '0');
}

```

```

    return integer;
}

/**
 * Parses some text as though it is a decimal
 *
 * @access protected
 *
 * @param wchar_t** data Pointer to a wchar_t* that contains the JSON text
 *
 * @return double Returns the double value of the decimal found
 */
double JSON::ParseDecimal(const wchar_t **data)
{
    double decimal = 0.0;
    double factor = 0.1;
    while (**data != 0 && **data >= '0' && **data <= '9')
    {
        int digit = *(*data)++ - '0';
        decimal = decimal + digit * factor;
        factor *= 0.1;
    }
    return decimal;
}

```

## JSON.h

```

/**

```



\* File JSON.h part of the SimpleJSON Library - <http://mjpa.in/json>

\*

\* Copyright (C) 2010 Mike Anchor

\*

\* Permission is hereby granted, free of charge, to any person obtaining a copy

\* of this software and associated documentation files (the "Software"), to deal

\* in the Software without restriction, including without limitation the rights

\* to use, copy, modify, merge, publish, distribute, sublicense, and/or sell

\* copies of the Software, and to permit persons to whom the Software is

\* furnished to do so, subject to the following conditions:

\*

\* The above copyright notice and this permission notice shall be included in

\* all copies or substantial portions of the Software.

\*

\* THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR

\* IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,

\* FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE

\* AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER

\* LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,

\* OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN

\* THE SOFTWARE.

\*/

#ifndef \_JSON\_H\_

#define \_JSON\_H\_

```

// Win32 incompatibilities
#if defined(WIN32) && !defined(__GNUC__)
    #define wcsncasecmp_wcsnicmp
    static inline bool isnan(double x) { return x != x; }
    static inline bool isinf(double x) { return !isnan(x) && isnan(x - x); }
#endif

#include <vector>
#include <string>
#include <map>

// Linux compile fix - from quaker66
#ifdef __GNUC__
    #include <cstring>
    #include <cstdlib>
#endif

// Mac compile fixes - from quaker66, Lion fix by dabrahams
#if defined(__APPLE__) && __DARWIN_C_LEVEL < 200809L || (defined(WIN32) &&
defined(__GNUC__)) || defined(ANDROID)
    #include <wctype.h>
    #include <wchar.h>

    static inline int wcsncasecmp(const wchar_t *s1, const wchar_t *s2, size_t n)
    {
        int lc1 = 0;

```

```
int lc2 = 0;

while (n--)
{
    lc1 = tolower (*s1);
    lc2 = tolower (*s2);

    if (lc1 != lc2)
        return (lc1 - lc2);

    if (!lc1)
        return 0;

    ++s1;
    ++s2;
}

return 0;
}
#endif

// Simple function to check a string 's' has at least 'n' characters
static inline bool simplejson_wcsnlen(const wchar_t*s, size_t n) {
    if (s == 0)
        return false;
```

```
const wchar_t *save = s;
while (n-- > 0)
{
    if (*(save++) == 0) return false;
}

return true;
}

// Custom types
class JSONValue;
typedef std::vector<JSONValue*> JSONArray;
typedef std::map<std::wstring, JSONValue*> JSONObject;

#include "JSONValue.h"

class JSON
{
    friend class JSONValue;

public:
    static JSONValue* Parse(const char *data);
    static JSONValue* Parse(const wchar_t *data);
    static std::wstring Stringify(const JSONValue *value);
protected:
```

```
static bool SkipWhitespace(const wchar_t **data);
static bool ExtractString(const wchar_t **data, std::wstring &str);
static double ParseInt(const wchar_t **data);
static double ParseDecimal(const wchar_t **data);
private:
    JSON();
};

#endif
```

## **JSONValue.cpp**

```
/*
 * File JSONValue.cpp part of the SimpleJSON Library - http://mjpa.in/json
 *
 * Copyright (C) 2010 Mike Anchor
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this software and associated documentation files (the "Software"), to deal
 * in the Software without restriction, including without limitation the rights
 * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
 * copies of the Software, and to permit persons to whom the Software is
 * furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included in
 * all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS
 * OR
 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
```

\* FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE

\* AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER

\* LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,

\* OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN

\* THE SOFTWARE.

\*/

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <vector>
#include <string>
#include <sstream>
#include <iostream>
#include <math.h>
```

```
#include "JSONValue.h"
```

```
// Macros to free an array/object
```

```
#define FREE_ARRAY(x) { JSONArray::iterator iter; for (iter = x.begin(); iter != x.end(); iter++) { delete *iter; } }
```

```
#define FREE_OBJECT(x) { JSONObject::iterator iter; for (iter = x.begin(); iter != x.end(); iter++) { delete (*iter).second; } }
```

```
/**
```

```

* Parses a JSON encoded value to a JSONValue object
*
* @access protected
*
* @param wchar_t** data Pointer to a wchar_t* that contains the data
*
* @return JSONValue* Returns a pointer to a JSONValue object on success, NULL on error
*/

```

```

JSONValue *JSONValue::Parse(const wchar_t **data)

```

```

{
    // Is it a string?
    if (**data == '"')
    {
        std::wstring str;
        if (!JSON::ExtractString(&((*data)), str))
            return NULL;
        else
            return new JSONValue(str);
    }

    // Is it a boolean?
    else if ((simplejson_wcsnlen(*data, 4) && wcsncasecmp(*data, L"true", 4) == 0) ||
             (simplejson_wcsnlen(*data, 5) && wcsncasecmp(*data, L"false", 5) == 0))
    {
        bool value = wcsncasecmp(*data, L"true", 4) == 0;
        (*data) += value ? 4 : 5;
        return new JSONValue(value);
    }

    // Is it a null?

```

```
else if (simplejson_wcsnlen(*data, 4) && wcsncasecmp(*data, L"null", 4) == 0)
{
    (*data) += 4;
    return new JSONValue();
}
```

```
// Is it a number?
```

```
else if (**data == L'-' || (**data >= L'0' && **data <= L'9'))
{
```

```
    // Negative?
```

```
    bool neg = **data == L'-';
```

```
    if (neg) (*data)++;
```

```
    double number = 0.0;
```

```
    // Parse the whole part of the number - only if it wasn't 0
```

```
    if (**data == L'0')
```

```
        (*data)++;
```

```
    else if (**data >= L'1' && **data <= L'9')
```

```
        number = JSON::ParseInt(data);
```

```
    else
```

```
        return NULL;
```

```
    // Could be a decimal now...
```

```
    if (**data == L'.')
```

```
    {
```

```
        (*data)++;
```



```

// Not get any digits?
if (!(**data >= L'0' && **data <= L'9'))
    return NULL;

// Find the decimal and sort the decimal place out
// Use ParseDecimal as ParseInt won't work with decimals less than 0.1
// thanks to Javier Abadia for the report & fix
double decimal = JSON::ParseDecimal(data);

// Save the number
number += decimal;
}

// Could be an exponent now...
if (**data == L'E' || **data == L'e')
{
    (*data)++;

// Check signage of expo
bool neg_expo = false;
if (**data == L'-' || **data == L'+')
{
    neg_expo = **data == L'-';
    (*data)++;
}

// Not get any digits?
if (!(**data >= L'0' && **data <= L'9'))

```

```
    return NULL;

    // Sort the expo out
    double expo = JSON::ParseInt(data);
    for (double i = 0.0; i < expo; i++)
        number = neg_expo ? (number / 10.0) : (number * 10.0);
}

// Was it neg?
if (neg) number *= -1;

return new JSONValue(number);
}

// An object?
else if (**data == L'{')
{
    JSONObject object;

    (*data)++;

    while (**data != 0)
    {
        // Whitespace at the start?
        if (!JSON::SkipWhitespace(data))
        {
            FREE_OBJECT(object);
```

```
    return NULL;
}

// Special case - empty object
if (object.size() == 0 && **data == L}')
{
    (*data)++;
    return new JSONValue(object);
}

// We want a string now...
std::wstring name;
if (!JSON::ExtractString(&+>(*data)), name)
{
    FREE_OBJECT(object);
    return NULL;
}

// More whitespace?
if (!JSON::SkipWhitespace(data))
{
    FREE_OBJECT(object);
    return NULL;
}

// Need a : now
if ((*data)++ != L':')
{
    FREE_OBJECT(object);
    return NULL;
}
```

```
}
```

```
// More whitespace?
```

```
if (!JSON::SkipWhitespace(data))
```

```
{
```

```
    FREE_OBJECT(object);
```

```
    return NULL;
```

```
}
```

```
// The value is here
```

```
JSONValue *value = Parse(data);
```

```
if (value == NULL)
```

```
{
```

```
    FREE_OBJECT(object);
```

```
    return NULL;
```

```
}
```

```
// Add the name:value
```

```
if (object.find(name) != object.end())
```

```
    delete object[name];
```

```
object[name] = value;
```

```
// More whitespace?
```

```
if (!JSON::SkipWhitespace(data))
```

```
{
```

```
    FREE_OBJECT(object);
```

```
    return NULL;
```

```
}
```

```
// End of object?
```

```
    if (**data == L}')
    {
        (*data)++;
        return new JSONValue(object);
    }

    // Want a , now
    if (**data != L',')
    {
        FREE_OBJECT(object);
        return NULL;
    }

    (*data)++;
}

// Only here if we ran out of data
FREE_OBJECT(object);
return NULL;
}

// An array?
else if (**data == L'[')
{
    JSONArray array;

    (*data)++;

    while (**data != 0)
    {
```

```
// Whitespace at the start?
if (!JSON::SkipWhitespace(data))
{
    FREE_ARRAY(array);
    return NULL;
}

// Special case - empty array
if (array.size() == 0 && **data == L']')
{
    (*data)++;
    return new JSONValue(array);
}

// Get the value
JSONValue *value = Parse(data);
if (value == NULL)
{
    FREE_ARRAY(array);
    return NULL;
}

// Add the value
array.push_back(value);

// More whitespace?
if (!JSON::SkipWhitespace(data))
{
    FREE_ARRAY(array);
    return NULL;
}
```

```
    }

    // End of array?
    if (**data == L']')
    {
        (*data)++;
        return new JSONValue(array);
    }

    // Want a , now
    if (**data != L',')
    {
        FREE_ARRAY(array);
        return NULL;
    }

    (*data)++;
}

// Only here if we ran out of data
FREE_ARRAY(array);
return NULL;
}

// Ran out of possibilites, it's bad!
else
{
    return NULL;
}
}
```

```
/**
 * Basic constructor for creating a JSON Value of type NULL
 *
 * @access public
 */
JSONValue::JSONValue(/*NULL*/)
{
    type = JSONType_Null;
}
```

```
/**
 * Basic constructor for creating a JSON Value of type String
 *
 * @access public
 *
 * @param wchar_t* m_char_value The string to use as the value
 */
JSONValue::JSONValue(const wchar_t *m_char_value)
{
    type = JSONType_String;
    string_value = std::wstring(m_char_value);
}
```

```
/**
 * Basic constructor for creating a JSON Value of type String
 *
 * @access public
```



```
*  
* @param std::wstring m_string_value The string to use as the value  
*/
```

```
JSONValue::JSONValue(const std::wstring &m_string_value)
```

```
{  
    type = JSONType_String;  
    string_value = m_string_value;  
}
```

```
/**
```

```
* Basic constructor for creating a JSON Value of type Bool
```

```
*
```

```
* @access public
```

```
*
```

```
* @param bool m_bool_value The bool to use as the value
```

```
*/
```

```
JSONValue::JSONValue(bool m_bool_value)
```

```
{  
    type = JSONType_Boolean;  
    bool_value = m_bool_value;  
}
```

```
/**
```

```
* Basic constructor for creating a JSON Value of type Number
```

```
*
```

```
* @access public
```

```
*
```

```
* @param double m_number_value The number to use as the value
```

```

*/
JSONValue::JSONValue(double m_number_value)
{
    type = JSONType_Number;
    number_value = m_number_value;
}

/**
 * Basic constructor for creating a JSON Value of type Array
 *
 * @access public
 *
 * @param JSONArray m_array_value The JSONArray to use as the value
 */
JSONValue::JSONValue(const JSONArray &m_array_value)
{
    type = JSONType_Array;
    array_value = m_array_value;
}

/**
 * Basic constructor for creating a JSON Value of type Object
 *
 * @access public
 *
 * @param JSONObject m_object_value The JSONObject to use as the value
 */
JSONValue::JSONValue(const JSONObject &m_object_value)

```

```

{
    type = JSONType_Object;
    object_value = m_object_value;
}

/**
 * The destructor for the JSON Value object
 * Handles deleting the objects in the array or the object value
 *
 * @access public
 */
JSONValue::~JSONValue()
{
    if (type == JSONType_Array)
    {
        JSONArray::iterator iter;
        for (iter = array_value.begin(); iter != array_value.end(); iter++)
            delete *iter;
    }
    else if (type == JSONType_Object)
    {
        JSONObject::iterator iter;
        for (iter = object_value.begin(); iter != object_value.end(); iter++)
        {
            delete (*iter).second;
        }
    }
}

```

```
/**
 * Checks if the value is a NULL
 *
 * @access public
 *
 * @return bool Returns true if it is a NULL value, false otherwise
 */
bool JSONValue::IsNull() const
{
    return type == JSONType_Null;
}
```

```
/**
 * Checks if the value is a String
 *
 * @access public
 *
 * @return bool Returns true if it is a String value, false otherwise
 */
bool JSONValue::IsString() const
{
    return type == JSONType_String;
}
```

```
/**
 * Checks if the value is a Bool
 *
```

```
* @access public
*
* @return bool Returns true if it is a Bool value, false otherwise
*/
bool JSONValue::IsBool() const
{
    return type == JSONType_Boolean;
}

/**
* Checks if the value is a Number
*
* @access public
*
* @return bool Returns true if it is a Number value, false otherwise
*/
bool JSONValue::IsNumber() const
{
    return type == JSONType_Number;
}

/**
* Checks if the value is an Array
*
* @access public
*
* @return bool Returns true if it is an Array value, false otherwise
*/
```

```
bool JSONValue::isArray() const
```

```
{  
    return type == JSONType_Array;  
}
```

```
/**
```

```
* Checks if the value is an Object
```

```
*
```

```
* @access public
```

```
*
```

```
* @return bool Returns true if it is an Object value, false otherwise
```

```
*/
```

```
bool JSONValue::isObject() const
```

```
{  
    return type == JSONType_Object;  
}
```

```
/**
```

```
* Retrieves the String value of this JSONValue
```

```
* Use IsString() before using this method.
```

```
*
```

```
* @access public
```

```
*
```

```
* @return std::wstring Returns the string value
```

```
*/
```

```
const std::wstring &JSONValue::AsString() const
```

```
{  
    return string_value;
```

```
}
```

```
/**
```

```
* Retrieves the Bool value of this JSONValue
```

```
* Use IsBool() before using this method.
```

```
*
```

```
* @access public
```

```
*
```

```
* @return bool Returns the bool value
```

```
*/
```

```
bool JSONValue::AsBool() const
```

```
{
```

```
    return bool_value;
```

```
}
```

```
/**
```

```
* Retrieves the Number value of this JSONValue
```

```
* Use IsNumber() before using this method.
```

```
*
```

```
* @access public
```

```
*
```

```
* @return double Returns the number value
```

```
*/
```

```
double JSONValue::AsNumber() const
```

```
{
```

```
    return number_value;
```

```
}
```

```
/**
 * Retrieves the Array value of this JSONValue
 * Use isArray() before using this method.
 *
 * @access public
 *
 * @return JSONArray Returns the array value
 */
const JSONArray &JSONValue::AsArray() const
{
    return array_value;
}
```

```
/**
 * Retrieves the Object value of this JSONValue
 * Use IsObject() before using this method.
 *
 * @access public
 *
 * @return JSONObject Returns the object value
 */
const JSONObject &JSONValue::AsObject() const
{
    return object_value;
}
```

```
/**
 * Retrieves the number of children of this JSONValue.
```



\* This number will be 0 or the actual number of children

\* if isArray() or IsObject().

\*

\* @access public

\*

\* @return The number of children.

\*/

std::size\_t JSONValue::CountChildren() const

{

  switch (type)

  {

    case JSONType\_Array:

      return array\_value.size();

    case JSONType\_Object:

      return object\_value.size();

    default:

      return 0;

  }

}

/\*\*

\* Checks if this JSONValue has a child at the given index.

\* Use isArray() before using this method.

\*

\* @access public

\*

\* @return bool Returns true if the array has a value at the given index.

\*/

bool JSONValue::HasChild(std::size\_t index) const

```

{
    if (type == JSONType_Array)
    {
        return index < array_value.size();
    }
    else
    {
        return false;
    }
}

/**
 * Retrieves the child of this JSONValue at the given index.
 * Use isArray() before using this method.
 *
 * @access public
 *
 * @return JSONValue* Returns JSONValue at the given index or NULL
 *         if it doesn't exist.
 */
JSONValue *JSONValue::Child(std::size_t index)
{
    if (index < array_value.size())
    {
        return array_value[index];
    }
    else
    {
        return NULL;
    }
}

```

```
}  
}
```

```
/**
```

```
* Checks if this JSONValue has a child at the given key.
```

```
* Use IsObject() before using this method.
```

```
*
```

```
* @access public
```

```
*
```

```
* @return bool Returns true if the object has a value at the given key.
```

```
*/
```

```
bool JSONValue::HasChild(const wchar_t* name) const
```

```
{  
    if (type == JSONType_Object)  
    {  
        return object_value.find(name) != object_value.end();  
    }  
    else  
    {  
        return false;  
    }  
}
```

```
/**
```

```
* Retrieves the child of this JSONValue at the given key.
```

```
* Use IsObject() before using this method.
```

```
*
```

```
* @access public
```

```

*
* @return JSONValue* Returns JSONValue for the given key in the object
*         or NULL if it doesn't exist.
*/
JSONValue* JSONValue::Child(const wchar_t* name)
{
    JSONObject::const_iterator it = object_value.find(name);
    if (it != object_value.end())
    {
        return it->second;
    }
    else
    {
        return NULL;
    }
}

/**
* Creates a JSON encoded string for the value with all necessary characters escaped
*
* @access public
*
* @return std::wstring Returns the JSON string
*/
std::wstring JSONValue::Stringify() const
{
    std::wstring ret_string;

    switch (type)

```

```
{
case JSONType_Null:
    ret_string = L"null";
    break;

case JSONType_String:
    ret_string = StringifyString(string_value);
    break;

case JSONType_Bool:
    ret_string = bool_value ? L"true" : L"false";
    break;

case JSONType_Number:
{
    if (isinf(number_value) || isnan(number_value))
        ret_string = L"null";
    else
    {
        std::wstringstream ss;
        ss.precision(15);
        ss << number_value;
        ret_string = ss.str();
    }
    break;
}

case JSONType_Array:
{
    ret_string = L"[";
```

```

JSONArray::const_iterator iter = array_value.begin();
while (iter != array_value.end())
{
    ret_string += (*iter)->Stringify();

    // Not at the end - add a separator
    if (++iter != array_value.end())
        ret_string += L",";
}
ret_string += L"]";
break;
}

```

case JSONType\_Object:

```

{
    ret_string = L"{";
    JSONObject::const_iterator iter = object_value.begin();
    while (iter != object_value.end())
    {
        ret_string += StringifyString((*iter).first);
        ret_string += L":";
        ret_string += (*iter).second->Stringify();

        // Not at the end - add a separator
        if (++iter != object_value.end())
            ret_string += L",";
    }
    ret_string += L"}";
    break;
}

```

```

}

return ret_string;
}

/**
 * Creates a JSON encoded string with all required fields escaped
 * Works from http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-262.pdf
 * Section 15.12.3.
 *
 * @access private
 *
 * @param std::wstring str The string that needs to have the characters escaped
 *
 * @return std::wstring Returns the JSON string
 */
std::wstring JSONValue::StringifyString(const std::wstring &str)
{
    std::wstring str_out = L"";

    std::wstring::const_iterator iter = str.begin();
    while (iter != str.end())
    {
        wchar_t chr = *iter;

        if (chr == L'"' || chr == L'\\' || chr == L'/')
        {
            str_out += L'\\';

```

```
    str_out += chr;
}
else if (chr == L'\b')
{
    str_out += L"\\b";
}
else if (chr == L'\f')
{
    str_out += L"\\f";
}
else if (chr == L'\n')
{
    str_out += L"\\n";
}
else if (chr == L'\r')
{
    str_out += L"\\r";
}
else if (chr == L'\t')
{
    str_out += L"\\t";
}
else if (chr < L' ' || chr > 126)
{
    str_out += L"\\u";
    for (int i = 0; i < 4; i++)
    {
        int value = (chr >> 12) & 0xf;
        if (value >= 0 && value <= 9)
            str_out += (wchar_t)('0' + value);
```



```

        else if (value >= 10 && value <= 15)
            str_out += (wchar_t)('A' + (value - 10));
        chr <<= 4;
    }
}
else
{
    str_out += chr;
}

    iter++;
}

str_out += L"\n";
return str_out;
}

```

## JSONValue.h

```

/*
 * File JSONValue.h part of the SimpleJSON Library - http://mjpa.in/json
 *
 * Copyright (C) 2010 Mike Anchor
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this software and associated documentation files (the "Software"), to deal
 * in the Software without restriction, including without limitation the rights
 * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
 * copies of the Software, and to permit persons to whom the Software is
 * furnished to do so, subject to the following conditions:

```

\*

\* The above copyright notice and this permission notice shall be included in

\* all copies or substantial portions of the Software.

\*

\* THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR

\* IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,

\* FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE

\* AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER

\* LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,

\* OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN

\* THE SOFTWARE.

\*/

```
#ifndef _JSONVALUE_H_
```

```
#define _JSONVALUE_H_
```

```
#include <vector>
```

```
#include <string>
```

```
#include "JSON.h"
```

```
class JSON;
```

```
enum JSONType { JSONType_Null, JSONType_String, JSONType_Bool, JSONType_Number,
JSONType_Array, JSONType_Object };
```

```
class JSONValue
```

```
{
```

```
    friend class JSON;
```

```
public:
```

```
    JSONValue(/*NULL*/);
```

```
    JSONValue(const wchar_t *m_char_value);
```

```
    JSONValue(const std::wstring &m_string_value);
```

```
    JSONValue(bool m_bool_value);
```

```
    JSONValue(double m_number_value);
```

```
    JSONValue(const JSONArray &m_array_value);
```

```
    JSONValue(const JSONObject &m_object_value);
```

```
    ~JSONValue();
```

```
    bool IsNull() const;
```

```
    bool IsString() const;
```

```
    bool IsBool() const;
```

```
    bool IsNumber() const;
```

```
    bool IsArray() const;
```

```
    bool IsObject() const;
```

```
    const std::wstring &AsString() const;
```

```
    bool AsBool() const;
```

```
    double AsNumber() const;
```

```
    const JSONArray &AsArray() const;
```

```
    const JSONObject &AsObject() const;
```

```
std::size_t CountChildren() const;
bool HasChild(std::size_t index) const;
JSONValue *Child(std::size_t index);
bool HasChild(const wchar_t* name) const;
JSONValue *Child(const wchar_t* name);
```

```
std::wstring Stringify() const;
```

```
protected:
```

```
static JSONValue *Parse(const wchar_t **data);
```

```
private:
```

```
static std::wstring StringifyString(const std::wstring &str);
```

```
JSONType type;
```

```
std::wstring string_value;
```

```
bool bool_value;
```

```
double number_value;
```

```
JSONArray array_value;
```

```
JSONObject object_value;
```

```
};
```

```
#endif
```

**Makefile (c++)**

CC=g++

CFLAGS=-c -Wall

LFLAGS=-lm

SOURCES=runCplusplus.cpp JSON.cpp JSONValue.cpp

HEADERS=JSON.h JSONValue.h cplusplusCompiler.h

OBJECTS=\$(SOURCES:.cpp=.o)

EXECUTABLE=TestJSON

all: \$(SOURCES) \$(EXECUTABLE)

\$(EXECUTABLE): \$(OBJECTS)

\$(CC) \$(LFLAGS) \$(OBJECTS) -o \$@

.cpp.o:

\$(CC) \$(CFLAGS) \$< -o \$@

clean:

rm -f \$(OBJECTS) \$(EXECUTABLE) \*~ SampleOutput.txt

full: clean all

### **SampleInput.txt**

```
{ "name" : "name1",  
  "age" : 25123,  
  "BoolVal" : True,  
  "scores" : [1, 2, 3, {  
    "foo" : "False",  
    "bar" : "123" } ],  
  "sub_object" : {  
    "foo" : "True",  
    "bar" : "123" }  
}
```

### **cPlusPlusCompiler.h**

```
#include <iostream>  
#include <fstream>  
#include <string>  
#include <vector>  
#include <map>  
#include <algorithm>  
#include "./JSON.h"  
#include <sstream>
```

```
using namespace std;
```

```
//SimpleJSON uses
```

```
void print_out(const wchar_t *output)
```

```
{  
    wcout << output;  
    wcout.flush();  
}
```

```
string funcRemoveStr (string s) {
```

```
    int a = 0 ;  
    string p = "\\n" ;  
    while ( a != -1){  
        a = s.find(p) ;  
        if (a == -1)  
            break ;  
        s = s.substr(0,a) + s.substr(a+2);  
    }
```

```
    return s ;
```

```
}
```

```
string wstringToString (wstring w){
```

```
    string result = "" ;  
    char x ;  
    for ( int i = 0 ; w[i] != '\0' ; i ++){  
        x = w[i] ;  
        result += x ;  
    }
```

```
    return result ;
```

```
}
```

```
enum dataType { NUMBER , STRING , BOOL , JSON, LIST} ;
```

```

class CustType {

    public :

    static string data ;

    CustType (string data) {

        //this->dt = NUMBER ;
        this->data = data ;
    }

    CustType () {

    }

    static CustType* parse (string data, string type) ;
    static string typeString ( CustType* t) ;
    static CustType* read(string filename);
    static void print(CustType* data) ;
    static void print (vector<CustType*> :: iterator it) ;
    static void write(CustType* data, string filename);
    static CustType* typeStruct(CustType* input);
    static CustType* typeStructList(CustType* input);
    virtual void print () {
        cout << "Printing in CustType, Oops!\n Somebody needs to implement this in child class\n"
;
    }
    virtual int getType(){
        cout << "getting Type from CustType, Oops!\n Somebody needs to implement this in child
class";
    }
}

```



```

}
virtual map<string , CustType* >::iterator getBeginlterator (){
    map<string , CustType* >::iterator it ;
    cout << "Accesing outside of Json Object " ;
    return it ;
}
virtual map<string , CustType* >::iterator getEndlterator (){
    map<string , CustType* >::iterator it ;
    cout << "Accesing outside of Json Object " ;
    return it ;
}
virtual vector <CustType*> :: iterator getListBegin () {
    vector <CustType*> :: iterator it ;
    cout << "Accesing outside of List Object " ;
    return it ;
}
virtual vector <CustType*> :: iterator getListEnd () {
    vector <CustType*> :: iterator it ;
    cout << "Accesing outside of List Object " ;
    return it ;
}
virtual CustType* getAttrList () {
    JSONObject::iterator it ;
    cout << "Accesing outside of Json Object " ;
    return NULL ;
}
virtual CustType* getElementByOcaml ( CustType* key ) {
    cout << "In CustType, apparently not in JSON or LIST. Calling from some other type\n" ;
    return NULL ;
}

```

```

virtual CustType* getElement ( string key ) {
    cout << "In CustType, apparently not in JSON. Calling from some other type\n" ;
    return NULL ;
}
virtual CustType* getElement ( int index ) {
    cout << "In CustType, apparently not in LIST. Calling from some other type\n" ;
    return NULL ;
}

//Conv to String
virtual string toString () {
    return "\nInside CustType\n" ;
}

//conv to StrType
virtual CustType* makeString () {
    return CustType :: parse ("\nInside CustType\n" , "STRING") ;
}
//print json in prettyPrint format
virtual string prettyPrint(int offset){
    cout << "prettyPrint() is only valid on JSON type objects.\n" ;
}
//append to a list
virtual void add (CustType* el) {
    cout << "In CustType, apparently not in LIST. Calling from some other type\n" ;
}
virtual void addByKey (CustType* index , CustType* el){

```

```

    cout << "In CustType, apparently not in LIST or JSON. Calling from some other type\n" ;
}

//add item to map
virtual void add (string key, CustType* el) {
    cout << "In CustType, apparently not in JSON. Calling from some other type\n" ;

}

/*virtual void addToJson (CustType* key, CustType* el) {
    cout << "In CustType, apparently not in JSON. Calling from some other type\n" ;
}*/
virtual CustType* contains (CustType* t) {
    cout << "Only defined for Lists. \n" ;
    return NULL ;
}
virtual CustType* findIndex (CustType* t){
    cout << "Only defined for Lists. \n" ;
    return NULL ;
}
virtual CustType* minus (CustType* t) {
    cout << "Only defined for Lists. \n" ;
    return NULL ;
}
virtual CustType* getJoType() {
    cout << "Printing in CustType, Ooops!\n Somebody needs to implement this in child
class\n" ;
}

//concat items and form a list
static CustType* concat (CustType* t1, CustType* t2) ;

```

```
//Mathematical Operators
//Add is valid for NUMBER + NUMBER and STRING + STRING (JO operator ++)
//The remaining mathematical operators are valid only for NUMBER, NUMBER
//Returns NULL pointer if arguments are of an invalid type.
static CustType* add(CustType* t1, CustType* t2);

static CustType* subtract(CustType* t1, CustType* t2);

static CustType* multiply(CustType* t1, CustType* t2);

static CustType* divide(CustType* t1, CustType* t2);

static CustType* mod(CustType* t1, CustType* t2);

//Get c++ Boolean value from BoolType
virtual bool getBoolValue() {
    cout << "In CustType, only valid for BoolType\n";
}

//Operator Overloading:
virtual CustType& operator+=(CustType& rhs)
{
    cout << "In CustType, only allowed in NUMBER and STRING\n";
```

```
}
```

```
virtual CustType& operator-=(CustType& rhs)  
{  
    cout << "In CustType, only allowed in NUMBER\n";  
}
```

```
virtual CustType& operator*=(CustType& rhs)  
{  
    cout << "In CustType, only allowed in NUMBER\n";  
}
```

```
virtual CustType& operator/=(CustType& rhs)  
{  
    cout << "In CustType, only allowed in NUMBER\n";  
}  
};
```

```
class BoolType : public CustType {  
    public :  
  
    bool da ;  
    int type ;  
    BoolType (bool da , int type ) : CustType ( ) {  
  
        this -> da = da ;  
        this -> type = type ;
```

```

}
int getType () {
    return BOOL;
}

bool getBoolValue()
{
    return da;
}
CustType* getJoType() {
    return ( CustType :: parse ("Bool" , "STRING")) ;
}
void print () {
    if(da) { cout << "true" ; }
    else { cout << "false" ; }
}
string toString () {
    string ret = "false";
    if ( da)
        ret = "true" ;
    return ret ;
}
CustType* makeString () {
    string ret = "false";
    if ( da)
        ret = "true" ;
    return CustType :: parse (this -> toString() , "STRING") ;
}
};

```

```

class NumType : public CustType {

public :

double da;
int type ;
NumType (double da , int type) : CustType ( ) {
    this -> da = da ;
    this -> type = type ;

}
void print () {
    cout << da ;
}
int getType () {
    return NUMBER ;
}
CustType* getJoType() {
    return ( CustType :: parse ("Number" , "STRING")) ;
}
string toString () {
    string ret ;
    ostringstream str;
    str << this -> da;
    ret = str.str() ;
    return ret ;
}

```

```

CustType* makeString () {

    return CustType :: parse( this -> toString() , "STRING" ) ;

}

NumType& operator+=(CustType& rhs)
    {
    CustType& t1 = rhs;
    NumType& temp = dynamic_cast<NumType&>(t1);
    da+=temp.da;
    return *this;
}

NumType& operator-=(CustType& rhs)
    {
    CustType& t1 = rhs;
    NumType& temp = dynamic_cast<NumType&>(t1);
    da-=temp.da;
    return *this;
}

NumType& operator*=(CustType& rhs)
    {
    CustType& t1 = rhs;
    NumType& temp = dynamic_cast<NumType&>(t1);
    da*=temp.da;
    return *this;
}

```



```

NumType& operator/=(CustType& rhs)
{
    CustType& t1 = rhs;
    NumType& temp = dynamic_cast<NumType&>(t1);
    da/=temp.da;
    return *this;
}
};

```

```

class StringType : public CustType {

public :
    string da;
    int type ;
    StringType (string da , int type ) : CustType ( ) {

        this -> da = da ;
        this -> type = type ;
    }

    StringType(){

    }

    string toString () {
        return da ;
    }

    CustType* makeString () {

```

```

        return CustType :: parse(this -> toString() , "STRING" );

    }

    CustType* getJoType() {
        return (CustType :: parse ("String" , "STRING"));
    }

    void print () {
        cout << da ;
    }

    int getType () {
        return STRING ;
    }

StringType& operator+=(CustType& rhs)
    {
        CustType& t1 = rhs;
        StringType& temp = dynamic_cast<StringType&>(t1);
        da+=temp.da;
        return *this;
    }

};

```

```

class ListType : public CustType {

```

```

    int type ;
    JSONArray data ;

```

```

public :
vector <CustType*> da;
void convToListType() ;
/*(ListType (vector <CustType> da , int type) : CustType ( ) {

    this -> da = da ;
    this -> type = type ;
}*/
ListType(){

}

ListType(JSONArray data, int type) : CustType() {
    this -> data = data ;
    this -> type = type ;
    convToListType() ;
}

ListType(vector <string> v) : CustType ( ) {
    for (vector<string> :: iterator it = v.begin ( ) ; it != v.end ( ) ; it ++ ){
        StringType* s = new StringType ( *it, STRING ) ;
        this -> da.push_back (s) ;
    }
}

int getType () {
    return LIST ;
}

CustType* getJoType() {
    return ( CustType :: parse ("List" , "STRING")) ;
}

void print () {
    /*

```

```

for (vector<CustType*> :: iterator it = da.begin () ; it != da.end () ; ++ it) {
    (*it) -> print () ;
    if ( it != (da.end() - 1) ) {
        cout << ",";
    }
}*/
cout << toString () ;
}

string toString () {
    string ret = "[" ;
    for (vector<CustType*> :: iterator it = da.begin () ; it != da.end () ; ++ it) {
        if ( it != da.begin () )
            ret += "," ;
        ret += (*it) -> toString () ;
    }
    ret += "]" ;
    return ret ;
}

```

```

CustType* makeString () {

    return CustType :: parse(this -> toString() , "STRING" ) ;

}

```

```

CustType* getElementByOcaml (CustType* indexNumType) {
    NumType* indexNum = dynamic_cast<NumType *>(indexNumType);
    double doubleKey = indexNum -> da ;
    int index = (int) doubleKey ;
    if ( index >= da.size())
        return NULL ;
    return da[index] ;
}

CustType* getElement (int index) {

    if ( index >= da.size())
        return NULL ;
    return da[index] ;
}

vector<CustType*> :: iterator getListBegin(){
    return da.begin() ;
}

vector<CustType*> :: iterator getListEnd (){
    return da.end() ;
}

void addByKey (CustType* index , CustType* el){
    NumType* indexNum = dynamic_cast<NumType *>(index);
    double doubleKey = indexNum -> da ;
    int intKey = (int) doubleKey ;
    if ( intKey >= 0 ){
        da [intKey] = el ;
    }
    else {
        cout << "index has value < 0" ;
        exit (1) ;
    }
}

```

```

    }
}
void add ( CustType* el ) {
    da.push_back (el) ;
}
bool operator==(CustType *rhs)
{
    NumType *temp1 = dynamic_cast<NumType *>(this);
    NumType *temp2 = dynamic_cast<NumType *>(rhs);

    return ((temp1->da)==(temp2->da));
}
//returns index of element if found, otherwise returns -1
CustType* findIndex (CustType* c);
CustType* minus (CustType* c);
CustType* contains (CustType * c);

};

```

```

class JsonType : public CustType {

```

```

    //JSONObject da;

```

```

    int type;

```

```

    public :

```

```

    JSONObject data ;

```

```

map <string , CustType* > da ;
void convToJsonType () ;
JsonType(JSONObject data, int type) : CustType() {
    this -> data= data;
    this -> type = type;
    //this -> da = new map <string , CustType* > ;
    convToJsonType() ;
}

JsonType() {}

void print() {/*
    JSONValue *value = new JSONValue(data);
    print_out(value->Stringify().c_str());*/
    cout << prettyPrint(0) ;
}

CustType* getJoType() {
    return ((CustType :: parse ("Json" , "STRING")) ) ;
}

string toString () {
    string ret = "{" ;
    for ( map<string , CustType* > :: iterator it = (this -> da).begin() ; it != (this -> da).end() ; ++ it
) {
        if ( it != (this -> da).begin() )
            ret += "," ;
        ret += "\"\" ;
        ret += it -> first ;
        ret += "\"\" ;

```

```

ret += ":" ;
string valStr = (it -> second) -> toString ();
if ( (it -> second) -> getType () == STRING) {
    ret += "\"";
    ret += valStr ;
    ret += "\"";
}
else {
    ret += valStr ;
}
}
ret += "}";
return ret ;

}

CustType* makeString () {
    return CustType :: parse ( this -> toString () , "STRING") ;
}

string prettyPrint (int offset) {
    string ret = "" ;
    string offsetTabs = "" ;
    for ( int i = 1 ; i <= offset ; i ++){
        offsetTabs += "\t" ;
    }
}

```



```

ret += offsetTabs ;
ret += "{\n" ;
string insideJsonPrint = prettyPrintJsonUtility (this , offset + 1) ;
ret += insideJsonPrint ;
ret += offsetTabs ;
ret += "}\n" ;
return ret ;
}

string prettyPrintJsonUtility ( JsonType* t , int offset) {
    string ret = "" ;
    string offsetTabs = "" ;
    for ( int i = 1 ; i <= offset ; i ++){
        offsetTabs += "\t" ;
    }
    for ( map<string , CustType* > :: iterator it = (t -> da).begin() ; it != (t -> da).end() ; ++ it ) {
        string el = "" ;
        if ( it != (t -> da).begin() )
            el += ",\n" ;
        el += offsetTabs ;
        el += "\"\" ;
        el += it -> first ;
        el += "\"\" ;

        el += ":" ;
        if ( (it -> second) -> getType() == NUMBER || (it -> second) -> getType() == BOOL){
            el += (it -> second) -> toString () ;
        }
        else if ((it -> second) -> getType () == STRING){
            el += "\"\" + (it -> second) -> toString () + "\"\" ;

```

```

}

else if ((it -> second) -> getType () == JSON){
    el += "{\n" ;
    el += prettyPrintJsonUtility ( (JsonType*)(it -> second) , offset + 1 ) ;
    el += offsetTabs ;
    el += "}" ;
}
else{
    el += "[\n" ;
    el += prettyPrintListUtility ( (ListType*)(it -> second) , offset + 1 ) ;
    el += offsetTabs ;
    el += "]" ;
}
ret += el ;
}
ret += "\n" ;
return ret ;
}

string prettyPrintListUtility( ListType* t, int offset){
    string ret = "" ;
    string offsetTabs = "" ;
    for ( int i = 1 ; i <= offset ; i ++){
        offsetTabs += "\t" ;
    }
    for (vector<CustType*> :: iterator it = (t -> da).begin () ; it != (t -> da).end () ; ++ it ) {
        string el = "" ;
        if ( it != (t-> da).begin() )
            el += ",\n" ;

```

```

el += offsetTabs ;

if ( (*it) -> getType() == NUMBER || (*it) -> getType() == BOOL){
    el += (*it) -> toString ();
}
else if ( (*it) -> getType () == STRING){
    el += "\"" + (*it) -> toString () + "\"";
}
else if ((*it) -> getType () == JSON){
    el += "{\n" ;
    el += prettyPrintJsonUtility ((JsonType*)(*it), offset + 1 ) ;
    el += offsetTabs ;
    el += "}";
}
else{
    el += "[\n" ;
    el += prettyPrintListUtility ((ListType*)(*it), offset + 1 ) ;
    el += offsetTabs ;
    el += "]" ;
}
ret += el ;
}
ret += "\n" ;
return ret ;
}

int getType() {
    return JSON;
}

CustType* getAttrList () ;

```

```

vector<string> getStringAttrList () {
    vector <string> attrListStr;

    for ( map<string , CustType* > ::iterator iter = getBeginlterator() ; iter != getEndlterator () ;
iter ++ ) {

        string keyString = iter -> first ;
        attrListStr.push_back(keyString) ;
    }
    return attrListStr ;
}

```

```

map<string , CustType* >::iterator getBeginlterator (){
    map<string , CustType* > :: iterator it = (this -> da).begin();

    return it ;
}

```

```

map<string , CustType* >::iterator getEndlterator (){
    map<string , CustType* > :: iterator it = (this -> da ).end () ;
    return it ;
}

```

```

CustType * getElementByOcaml (CustType* keyStrType) {
    StringType *keyStr = dynamic_cast<StringType *>(keyStrType);
    string key = keyStr -> toString () ;
    if ( da.find(key) == da.end())
        return NULL ;
    else
        return da[key] ;
}

```

```

}
CustType * getElement (string key) {
    if ( da.find(key) == da.end())
        return NULL ;
    else
        return da[key] ;
}
void addToJson (CustType* key, CustType* el){
    StringType *keyStr = dynamic_cast<StringType *>(key);
    string stringKey = keyStr -> toString () ;
    da [stringKey] = el ;
}
void addByKey (CustType* keyStrType, CustType* el) {
    StringType *keyStr = dynamic_cast<StringType *>(keyStrType);
    string key = keyStr -> toString () ;
    da [key] = el ;
}
void add (string key, CustType* el) {
    da [key] = el ;
}
CustType* minus (CustType* c){
    string k = c -> toString () ;
    CustType* t = new JsonType ;
    for ( map<string , CustType* > ::iterator iter = getBeginIterator() ; iter != getEndIterator () ;
iter ++ ) {

        if ( k != iter -> first)
            t -> add (iter -> first,iter -> second) ;

    }
}

```

```

    return t ;
}
CustType* contains (CustType* c){
    if (c ->getType() != STRING){
        cout << "\n JSON :: contains(key) expects a STRING argument\n" ;
        return this ;
    }
    string k = c -> toString() ;
    for ( map<string , CustType* > ::iterator iter = getBeginIterator() ; iter != getEndIterator () ;
iter ++ ) {

        if ( iter -> first == k)
            return (new BoolType(true,BOOL)) ;

    }
    return (new BoolType(false,BOOL)) ;

}

```

```

};
/*****
CLASS DEFINITIONS END HERE
*****/

```

```

NumType* getNum(string data, int type)
{
    const char *cstr = data.c_str();
    char* pEnd;

```

```
double num = strtod(cstr, &pEnd);
NumType* t = new NumType(num, NUMBER);
return t;
}
```

```
StringType* getString (string data, int type){
    /*if ( data.at(0) != "" || data.at(data.length() - 1 ) != "" ) {
        return NULL ;
    }*/
    //data.erase(0,1) ;
    //data.erase(data.length() - 1, 1) ;

    StringType* t = new StringType (data, STRING) ;

    return t ;
}
```

```
ListType* getList (string data, int type){

    std::string str = data;
    const char *cstr = str.c_str();

    JSONValue *value = JSON::Parse(cstr);
    JSONObject root;

    if (value == NULL) {
        return NULL;
    }
```

```

    }
    else {
    if (value->IsObject() == false) {
        return NULL;
    }
    else {
        root = value->AsObject();
    }
    }

    JsonType* t = new JsonType(root, JSON);

    return (ListType *) t -> getElement ("List");
}

BoolType* getBool (string data , int type) {
    BoolType * t = new BoolType ( data == "true" ? 1 : 0 , BOOL );
    return t ;
}

JsonType* getJson (string data, int type){

    std::string str = data;
    const char *cstr = str.c_str();

    JSONValue *value = JSON::Parse(cstr);

```



```

JSONObject root;

if (value == NULL) {
    return NULL;
}
else {
if (value->IsObject() == false) {
    return NULL;
}
else {
    root = value->AsObject();
}
}

JsonType* t = new JsonType(root, JSON);
return t ;
}

CustType* CustType :: parse (string data, string type) {

    data = funcRemoveStr (data) ;

    if (type == "STRING"){

        return getString ( data , STRING) ;

    }
}

```

```

else if (type == "LIST"){

    data = "{ \"List\" : " + data + " }" ;
    //CustType :: dt = LIST ;
    return getList ( data , LIST ) ;
}
else if (type == "JSON"){
    //CustType :: dt = JSON ;
    return getJson (data, JSON) ;
}
else if (type == "BOOL"){
    return getBool (data, JSON) ;
}
else if (type == "NUMBER"){
    //CustType :: dt = NUMBER ;
    return getNum (data, NUMBER) ;
}
else {
    return NULL ;
}
}

```

```

void CustType :: print ( CustType* data) {
    //cout << "I am here bitch" << endl ;
    data -> print () ;
}

```

```

void CustType :: write (CustType * data, string filename) {

```

```
string toWrite = data -> toString();
ofstream file(filename.c_str(), ios::app);
if ( file.is_open() )
{
    file << toWrite << endl;
}
else
{
    cout << "Unable to open file" << endl;
}
}
```

```
void CustType :: print (vector<CustType*> :: iterator it ) {
    (*it) -> print () ;
}
```

```
string CustType :: typeString ( CustType* t) {
```

```
    int typeVal = t -> getType () ;
    string type = "" ;
    switch (typeVal) {
        case NUMBER :    type = "NUMBER" ;
                        break ;
        case STRING :    type = "STRING" ;
                        break ;
        case JSON :    type = "JSON" ;
                        break ;
        case LIST :    type = "LIST" ;
```

```
        break ;
    }
    return type ;
}
```

```
void JsonType :: convToJsonType () {
```

```
    map <string , CustType* > a ;
```

```
    for ( JSONObject::iterator iter = (this -> data).begin() ; iter != (this -> data).end () ; iter ++ ) {
```

```
        string key = wstringToString ( iter -> first ) ;
```

```
        if ( (iter -> second)-> IsString () ) {
```

```
            string val = wstringToString ((iter-> second)-> AsString () ) ;
```

```
            StringType* t = new StringType (val, STRING) ;
```

```
            //cout << val ;
```

```
            a[key] = t ;
```

```
        }
```

```
        else if ( (iter -> second)-> IsBool () ) {
```

```
            bool val = (iter-> second)-> AsBool () ;
```

```
            BoolType* t = new BoolType (val, BOOL) ;
```

```
            //cout << val ;
```

```
            a[key] = t ;
```

```
        }
```

```
        else if ( (iter -> second)-> IsNumber () ) {
```

```
            double val = (iter-> second)-> AsNumber () ;
```

```
            NumType* t = new NumType (val, NUMBER) ;
```

```
            //cout << val ;
```

```

        a[key] = t ;
    }
    else if ( (iter -> second)-> IsObject() ){
        JSONObject val = (iter-> second)-> AsObject () ;
        JsonType* t = new JsonType ( val , JSON ) ;
        a[key] = t ;
    }
    else if ( (iter) -> second -> IsArray () ){
        JSONArray val = (iter -> second )-> AsArray () ;
        ListType* t = new ListType ( val , LIST ) ;
        a[key] = t ;
    }
    else {
        cout << "Here I am, stuck in JsonType :: convToJsonType, I think the JSON library is
screwing up, and throwing random types. " ;
    }
    //cout << endl ;

}

da.insert ( a.begin() , a.end () ) ;
//return a ;
}

void ListType :: convToListType () {

for ( vector<JSONValue*> :: iterator iter = data.begin () ; iter != data.end () ; iter ++ ) {

```

```

if ( (*iter) -> IsString () ) {
    string val = wstringToString ((*iter) -> AsString () );
    StringType* t = new StringType (val, STRING) ;
    //cout << val ;
    da.push_back(t) ;
}
else if ( (*iter) -> IsBool () ) {
    bool val = (*iter) -> AsBool () ;
    BoolType* t = new BoolType (val, BOOL) ;
    //cout << val ;
    da.push_back(t) ;
}
else if ( (*iter) -> IsNumber () ) {
    double val = (*iter) -> AsNumber () ;
    NumType* t = new NumType (val, NUMBER) ;
    //cout << val ;
    da.push_back(t) ;
}
else if ( (*iter) -> IsObject() ){
    JSONObject val = (*iter) -> AsObject () ;
    JsonType* t = new JsonType ( val , JSON) ;
    da.push_back(t) ;
}
else if ( (*iter) -> IsArray () ){
    JSONArray val = (*iter) -> AsArray () ;
    ListType* t = new ListType ( val , LIST ) ;
    da.push_back(t) ;
}
else {

```

```
        cout << "Here I am, stuck in ListType :: convToListType, I think the JSON library is  
screwing up, and throwing random types. " ;
```

```
    }
```

```
    //cout << endl ;
```

```
    }
```

```
}
```

```
CustType* JsonType :: getAttrList () {
```

```
    vector <string> attrListStr;
```

```
    for ( map<string, CustType* > ::iterator iter = getBeginIterator() ; iter != getEndIterator () ;  
iter ++ ) {
```

```
        string keyString = iter -> first ;
```

```
        attrListStr.push_back(keyString) ;
```

```
    }
```

```
    ListType* attrList = new ListType (attrListStr) ;
```

```
    return attrList ;
```

```
}
```

```
CustType* CustType::typeStruct(CustType *input)
```

```
{
```

```
    JsonType *inputJson = dynamic_cast<JsonType *>(input);
```

```
    CustType *returnString = new StringType("{", STRING);
```

```
    CustType *element;
```

```
    for( map<string, CustType*> :: iterator it = (inputJson->da).begin() ; it != (inputJson->da).end() ;  
++ it)
```

```

{
  int type = (it->second)->getType();
  if ( type == NUMBER )
  {
    element = new StringType(" String : Number, ", STRING);
  }
  else if ( type == STRING )
  {
    element = new StringType(" String : String, ", STRING);
  }
  else if ( type == BOOL )
  {
    element = new StringType(" String : Boolean, ", STRING);
  }
  else if ( type == JSON )
  {
    CustType *e1 = new StringType(" String : ", STRING);
    CustType *e2 = CustType::typeStruct(it->second);
    CustType *e3 = new StringType(" ", STRING);
    element = CustType::add(e1, e2);
    element = CustType::add(element, e3);
  }
  else if ( type == LIST )
  {
    //CustType *e1 = CustType::typeStructList(it->second);
    //CustType *e2 = new StringType(" ", STRING);
    //element = CustType::add(e1, e2);

```

//Use next line instead of above 3 if desired behavior is to simply return "List" instead of the types contained within the list.



```
    element = new StringType(" String : List, ", STRING);  
}
```

```
    returnString = CustType::add(returnString, element);  
}
```

```
StringType *str = dynamic_cast<StringType *>(returnString);  
string data = str->da;  
str->da = data.substr(0, data.size()-2);
```

```
StringType *s = new StringType(" }", STRING);  
CustType *toReturn = CustType::add(str, s);
```

```
    return toReturn;  
}
```

```
CustType* CustType::typeStructList(CustType *input)  
{  
    ListType *inputList = dynamic_cast<ListType *>(input);  
    CustType *returnString = new StringType("[", STRING);  
    CustType *element;  
    for (vector<CustType*> :: iterator it = (inputList->da).begin () ; it != (inputList->da).end () ; ++ it)  
    {  
        int type = (*it)->getType();  
        if ( type == NUMBER )  
        {
```

```
    element = new StringType(" Number,", STRING);
}
else if ( type == STRING )
{
    element = new StringType(" String,", STRING);
}
else if ( type == BOOL )
{
    element = new StringType("Boolean, ", STRING);
}
else if ( type == JSON )
{
    CustType *e0 = new StringType(" ", STRING);
    CustType *e1 = CustType::typeStruct(*it);
    CustType *e2 = new StringType(", ", STRING);
    element = CustType::add(e0, e1);
    element = CustType::add(element, e2);
}
else if ( type == LIST )
{
    CustType *e1 = CustType::typeStructList(*it);
    CustType *e2 = new StringType(", ", STRING);
    element = CustType::add(e1, e2);
}

returnString = CustType::add(returnString, element);
}
```

```
StringType *str = dynamic_cast<StringType *>(returnString);
string data = str->da;
str->da = data.substr(0, data.size()-2);
```

```
StringType *s = new StringType(" J", STRING);
CustType *toReturn = CustType::add(str, s);
```

```
return toReturn;
}
```

```
CustType* CustType :: concat (CustType* t1, CustType* t2){
    if ( t1 -> getType () == LIST) {
        t1 -> add ( t2 );
        return t1;
    }
    CustType* t = new ListType ;
    t -> add (t1) ;
    t -> add (t2) ;
    return t ;
}
```

```
// Mathematical Operators for NumType
```

```
CustType* CustType::add(CustType* t1, CustType* t2) {
    if( (t1->getType() == NUMBER) && (t2->getType() == NUMBER) ) {
```

```

NumType *temp1 = dynamic_cast<NumType*>(t1);
NumType *temp2 = dynamic_cast<NumType*>(t2);

double num = (temp1->da) + (temp2->da);
NumType *t = new NumType(num, NUMBER);
return t;
}
else if( (t1->getType() == STRING) && (t2->getType() == STRING) ) {
    StringType *temp1 = dynamic_cast<StringType*>(t1);
    StringType *temp2 = dynamic_cast<StringType*>(t2);

    string str = (temp1->da) + (temp2->da);
    StringType *t = new StringType(str, STRING);
    return t;
}
else if( (t1->getType() == LIST) && (t2->getType() == LIST) ) {
    ListType *temp1 = dynamic_cast<ListType*>(t1);
    ListType *temp2 = dynamic_cast<ListType*>(t2);
    ListType* t = new ListType() ;
    for (vector<CustType*> :: iterator it = (temp1->da).begin () ; it != (temp1->da).end () ; ++ it) {
        t -> add (*it) ;
    }
    for (vector<CustType*> :: iterator it = (temp2->da).begin () ; it != (temp2->da).end () ; ++ it) {
        t -> add (*it) ;
    }

    return t;
}

```

```

else {
    cout << "ERROR: ++ only allowed for NUMBER, NUMBER or STRING, STRING or
LIST,LIST" << endl;
    return NULL;
}

}

```

```

CustType* CustType::subtract(CustType* t1, CustType* t2) {
    if( (t1->getType() == NUMBER) && (t2->getType() == NUMBER) ) {
        NumType *temp1 = dynamic_cast<NumType*>(t1);
        NumType *temp2 = dynamic_cast<NumType*>(t2);

        double num = (temp1->da) - (temp2->da);
        NumType *t = new NumType(num, NUMBER);
        return t;
    }
    else {
        cout << "ERROR: Subtract only allowed for NUMBER, NUMBER" << endl;
        return NULL;
    }
}

```

```

CustType* CustType::multiply(CustType* t1, CustType* t2) {
    if( (t1->getType() == NUMBER) && (t2->getType() == NUMBER) ) {
        NumType *temp1 = dynamic_cast<NumType*>(t1);
        NumType *temp2 = dynamic_cast<NumType*>(t2);

```

```

double num = (temp1->da) * (temp2->da);
NumType *t = new NumType(num, NUMBER);
return t;
}
else {
    cout << "ERROR: Multiply only allowed for NUMBER, NUMBER" << endl;
    return NULL;
}
}

```

```

CustType* CustType::divide(CustType* t1, CustType* t2) {
if( (t1->getType() == NUMBER) && (t2->getType() == NUMBER) ) {
    NumType *temp1 = dynamic_cast<NumType*>(t1);
    NumType *temp2 = dynamic_cast<NumType*>(t2);

    double num = (temp1->da) / (temp2->da);
    NumType *t = new NumType(num, NUMBER);
    return t;
}
else {
    cout << "ERROR: Divide only allowed for NUMBER, NUMBER" << endl;
    return NULL;
}
}

```

```

CustType* CustType::mod(CustType* t1 , CustType* t2) {
if( (t1->getType() == NUMBER) && (t2->getType() == NUMBER) ) {

```

```
NumType *temp1 = dynamic_cast<NumType*>(t1);  
NumType *temp2 = dynamic_cast<NumType*>(t2);
```

```
int num1 = static_cast<int>(temp1->da);  
int num2 = static_cast<int>(temp2->da);
```

```
int mod = num1 % num2;
```

```
double num = static_cast<double>(mod);  
NumType *t = new NumType(num, NUMBER);  
return t;
```

```
}
```

```
else {
```

```
    cout << "ERROR: Mod only allowed for NUMBER, NUMBER" << endl;  
    return NULL;
```

```
}
```

```
}
```

```
//Comparison Operators for NumType
```

```
CustType* operator==(CustType &lhs, CustType &rhs)
```

```
{
```

```
    CustType *t1 = (&lhs);
```

```
    CustType *t2 = (&rhs);
```

```
    bool tempBool ;
```

```
    BoolType *toReturn ;
```

```

if ( t1 -> getType() != t2 -> getType () ){
    //cout << "==" defined on operands of same type" ;
    return (new BoolType(false, BOOL) ) ;
}
if (t1 -> getType() == NUMBER ){
    NumType *temp1 = dynamic_cast<NumType *>(&lhs);
    NumType *temp2 = dynamic_cast<NumType *>(&rhs);
    bool tempBool = ((temp1->da)==(temp2->da));
    toReturn = new BoolType(tempBool, BOOL);
    return toReturn ;
}
else if (t1 -> getType() == STRING ){
    StringType *temp3 = dynamic_cast<StringType *>(&lhs);
    StringType *temp4 = dynamic_cast<StringType *>(&rhs);
    tempBool = ((temp3->da)==(temp4->da));
    toReturn = new BoolType(tempBool, BOOL);
    return toReturn ;
}
else if (t1 -> getType () == BOOL){
    BoolType *temp5 = dynamic_cast<BoolType *>(&lhs);
    BoolType *temp6 = dynamic_cast<BoolType *>(&rhs);
    tempBool = ((temp5->da)==(temp6->da));

    //cout << "\nprinting inside c ++ " << temp5 -> getBoolValue() << temp6 -> getBoolValue()
    << tempBool << endl ;

    toReturn = new BoolType(tempBool, BOOL);
    return toReturn ;
}
else if (t1 -> getType () == LIST){
    ListType *temp7 = dynamic_cast<ListType *>(&lhs);
    ListType *temp8 = dynamic_cast<ListType *>(&rhs);

```



```

if ( (temp7 -> da).size() != (temp8 -> da).size() )
    return (new BoolType(false, BOOL) ) ;

for (vector<CustType*> :: iterator it1 = (temp7 -> da).begin () , it2 = (temp8 -> da).begin () ;
it1 != (temp7 -> da).end () && it2 != (temp8 -> da).end () ; ++ it1 , ++it2) {
    //(*it1) -> print() ;
    //(*it2) -> print () ;

    BoolType* tmpResult = dynamic_cast<BoolType *> ((*it1) == (*it2)) ;
    if (tmpResult -> getBoolValue () == false)
        return (new BoolType(false, BOOL) ) ;
}

return (new BoolType(true, BOOL));
}

else if (t1 -> getType () == JSON) {
    JsonType *temp9 = dynamic_cast<JsonType *>(&lhs);
    JsonType *temp10 = dynamic_cast<JsonType *>(&rhs);
    vector<string> attrList1 = temp9 -> getStringAttrList () ;
    vector<string> attrList2 = temp10 -> getStringAttrList () ;
    if ( attrList1.size () != attrList2.size())
        return (new BoolType(false, BOOL) ) ;

    for ( vector<string> :: iterator it = attrList1.begin () ; it != attrList1.end () ; ++ it) {
        CustType* el2 = temp10 ->getElement(*it) ;
        if ( el2 == NULL)
            return (new BoolType(false, BOOL) ) ;

        CustType* el1 = temp9 ->getElement(*it) ;
        BoolType* tmpResult = dynamic_cast<BoolType *> ((*el1 == *el2)) ;
        if (tmpResult -> getBoolValue () == false)
            return (new BoolType(false, BOOL) ) ;
    }

    return (new BoolType(true, BOOL));
}
}

```

```

else {
    cout << "ERROR: TYPE NOT DEFINED FOR OBJECT" << endl;
    return NULL;
}
}

```

```

CustType* operator!=(CustType &lhs, CustType &rhs)
{
    return (new BoolType( !( (lhs == rhs) ->getBoolValue() ), BOOL ) ); ;
}

```

```

CustType* operator<(CustType &lhs, CustType &rhs)
{
    NumType *temp1 = dynamic_cast<NumType *>(&lhs);
    NumType *temp2 = dynamic_cast<NumType *>(&rhs);
    bool tempBool = ((temp1->da)<(temp2->da));
    BoolType *toReturn = new BoolType(tempBool, BOOL);
    return toReturn;
}

```

```

CustType* operator>(CustType &lhs, CustType &rhs)
{
    NumType *temp1 = dynamic_cast<NumType *>(&lhs);
    NumType *temp2 = dynamic_cast<NumType *>(&rhs);
    bool tempBool = ((temp1->da)>(temp2->da));
    BoolType *toReturn = new BoolType(tempBool, BOOL);
}

```

```
return toReturn;  
}
```

```
CustType* operator<=(CustType &lhs, CustType &rhs)  
{  
    NumType *temp1 = dynamic_cast<NumType *>(&lhs);  
    NumType *temp2 = dynamic_cast<NumType *>(&rhs);  
    bool tempBool = ((temp1->da)<=(temp2->da));  
    BoolType *toReturn = new BoolType(tempBool, BOOL);  
    return toReturn;  
}
```

```
CustType* operator>=(CustType &lhs, CustType &rhs)  
{  
    NumType *temp1 = dynamic_cast<NumType *>(&lhs);  
    NumType *temp2 = dynamic_cast<NumType *>(&rhs);  
    bool tempBool = ((temp1->da)>=(temp2->da));  
    BoolType *toReturn = new BoolType(tempBool, BOOL);  
    return toReturn;  
}
```

```
// Logical Operators for BoolType
```

```
CustType* operator!(CustType &term)  
{  
    BoolType *temp = dynamic_cast<BoolType *>(&term);
```

```
bool tempBool = !(temp->da);
BoolType *toReturn = new BoolType(tempBool, BOOL);
return toReturn;
}
```

```
CustType* operator&&(CustType &t1, CustType &t2)
{
    BoolType *temp1 = dynamic_cast<BoolType *>(&t1);
    BoolType *temp2 = dynamic_cast<BoolType *>(&t2);
```

```
bool tempBool = ((temp1->da) && (temp2->da));
BoolType *toReturn = new BoolType(tempBool, BOOL);
return toReturn;
}
```

```
CustType* operator||(CustType &t1, CustType &t2)
{
    BoolType *temp1 = dynamic_cast<BoolType *>(&t1);
    BoolType *temp2 = dynamic_cast<BoolType *>(&t2);
```

```
bool tempBool = ((temp1->da) || (temp2->da));
BoolType *toReturn = new BoolType(tempBool, BOOL);
return toReturn;
}
```

```

// Reads from text file. Returns a ListType
CustType* CustType::read(string filename)
{
    string fileText = "";
    string line;
    ifstream file (filename.c_str());
    if ( file.is_open() )
    {
        while ( getline(file, line) )
        {
            fileText+=line;
        }
        file.close();
    }
    else { cout << "Unable to open file" << endl; }
    CustType *toReturn = CustType::parse( fileText , "JSON");
    return toReturn;
}

//returns index of element if found, otherwise returns -1
CustType* ListType :: findIndex (CustType* c){
    double i = 0 ;
    for (vector<CustType*> :: iterator it = da.begin () ; it != da.end () ; ++ it) {
        if (( *(*it)) == *c ) -> getBoolValue()
            return (new NumType(i,NUMBER)) ;
        i = i + 1 ;
    }
    i = -1 ;
    return (new NumType(i,NUMBER)) ;
}

```

```

CustType* ListType :: minus (CustType* c){
    ListType* tmp = new ListType ;
    for (vector<CustType*> :: iterator it = da.begin () ; it != da.end () ; ++ it) {

        if (!(>(*it) == *c)->getBoolValue() ){
            tmp -> add ((*it) ) ;

        }

    }
    return (CustType*)tmp ;
}

CustType* ListType :: contains (CustType* c){

    for (vector<CustType*> :: iterator it = da.begin () ; it != da.end () ; ++ it) {
        if (( (*(*it) == *c ) -> getBoolValue())
            return (new BoolType(true,BOOL)) ;

    }

    return (new BoolType(false,BOOL)) ;
}

```

### **runCpluPlus.cpp**

```

#include <iostream>
#include "../cpp/cPlusPlusCompiler.h"
using namespace std;

```

```

CustType* Merge(CustType* a, CustType* b)
{
CustType* c = CustType::parse("{}","JSON");
CustType* g = CustType::parse("[]","LIST");
CustType* f = CustType::parse("1","NUMBER");
CustType* d = a-> getAttrList();
CustType* e = b-> getAttrList();
for (vector<CustType*> :: iterator loopVarattr = (a->getAttrList ()) -> getListBegin(); loopVarattr
!=(a->getAttrList ()) -> getListEnd(); ++loopVarattr) {
    CustType* attr = *loopVarattr;
    {
        if ((b-> getAttrList()->contains(attr)->getBoolValue())
            {
                if ((*((a-> getElementByOcaml(attr))->getJoType()) ==
*(CustType::parse("Json","STRING"))) && *((b-> getElementByOcaml(attr))->getJoType()) ==
*(CustType::parse("Json","STRING")))->getBoolValue())
                    {
                        c->addByKey(attr,Merge(a-> getElementByOcaml(attr), b->
getElementByOcaml(attr)));
                    }
                }
            else
                {
                    if ((*((a-> getElementByOcaml(attr))->getJoType()) ==
*(CustType::parse("List","STRING"))) && *((b-> getElementByOcaml(attr))->getJoType()) ==
*(CustType::parse("List","STRING")))->getBoolValue())
                        {
                            c->addByKey(attr,CustType::add(a-> getElementByOcaml(attr),b->
getElementByOcaml(attr)));
                        }
                    }
                }
            }
        }
    }
}

```

```

    }
    else
    {
        c->addByKey(attr,CustType::concat(a-> getElementByOcaml(attr),b->
getElementByOcaml(attr)));

    }
}
}
else
{
    c->addByKey(attr,a-> getElementByOcaml(attr));

}
}
}
for (vector<CustType*> :: iterator loopVarattr = e->getListBegin(); loopVarattr != e->getListEnd();
++loopVarattr) {
    CustType* attr = *loopVarattr;
    {
    if (!(*(d->contains(attr)))->getBoolValue())
    {
        c->addByKey(attr,b-> getElementByOcaml(attr));

    }
}
}
}

```



```
return c;}
```

```
CustType* MergeUtil(CustType* a, CustType* b)
```

```
{
```

```
if ((*((a)->getJoType()) != *(CustType::parse("Json", "STRING"))) || ((*((b)->getJoType()) != *(CustType::parse("Json", "STRING")))->getBoolValue()))
```

```
{
```

```
CustType::print(CustType::parse("Both the arguments must be JSON", "STRING"));
```

```
return CustType::parse("null", "NULL");
```

```
} return Merge(a, b);}
```

```
int main()
```

```
{
```

```
CustType* a =
```

```
CustType::parse("{\"name\":\"harsha\", \"innerJson\":{ \"sub\": \"PLT\", \"mark\": [5,6,7] } }\", \"JSON\");
```

```
CustType* b =
```

```
CustType::parse("{\"name\":\"arpit\", \"innerJson\":{ \"sub\": \"OS\", \"mark\": [7,8,9] } }\", \"JSON\");
```

```
CustType* c = MergeUtil(a, b);
```

```
CustType::print(c);
```

```
}
```

### **test.c**

```
#include <stdio.h>
```

```
int main(void) {
```

```
int a[2][3] ;
```

```

a[1][1] = 80;

if(80 == *( *(a+1) + (1))) {
    printf("Address MAtches");
} else {
    printf("Address doesn't match");
}

return 0;
}

```

### **test.s**

```

.section __TEXT,__text,regular,pure_instructions
.globl _main
.align 4, 0x90
_main:                ## @main
.cfi_startproc
## BB#0:
    pushq  %rbp
Ltmp2:
    .cfi_def_cfa_offset 16
Ltmp3:
    .cfi_offset %rbp, -16
    movq   %rsp, %rbp
Ltmp4:
    .cfi_def_cfa_register %rbp
    leaq  L_.str(%rip), %rdi

```

```
xorl  %eax, %eax
callq _printf
xorl  %eax, %eax
popq  %rbp
retq
.cfi_endproc
```

```
.section __TEXT,__cstring,cstring_literals
L_.str:                ## @.str
.asciz "Address MAtches"
```

```
.subsections_via_symbols
```

## **MakeFileCPP**

```
CC=g++
```

```
CFLAGS=-c -w
```

```
LFLAGS=-lm
```

```
SOURCES=testfiles/$(inputfile).cpp cpp/JSON.cpp cpp/JSONValue.cpp
```

```
HEADERS=cpp/JSON.h cpp/JSONValue.h cpp/cPlusPlusCompiler.h
```

```
OBJECTS=$(SOURCES:.cpp=.o)
```

EXECUTABLE=testfiles/\$(inputfile).out

all: \$(SOURCES) \$(EXECUTABLE)

\$(EXECUTABLE): \$(OBJECTS)

\$(CC) \$(LFLAGS) \$(OBJECTS) -o \$@

.cpp.o:

\$(CC) \$(CFLAGS) \$< -o \$@

clean:

rm -f \$(OBJECTS) \$(EXECUTABLE) \*~

### **MakePreProc**

CC = gcc

CFLAGS = -g -Wall

.PHONY: default

default: preproc

.PHONY: all

all: clean preproc

.PHONY: clean

clean:

```
rm -rf *.log *.o *~ a.out* core preprocessor *.pjo
```

.PHONY: preproc

preproc: preprocessor.c

```
$(CC) -o preprocessor preprocessor.c
```

## **Makefile**

OBJS = parser.cmo scanner.cmo symboltable.cmo analyzer.cmo Str.cma jo.cmo

.PHONY: default

default: jo

.PHONY: all

all: clean jo

jo: \$(OBJS)

```
ocamlc -o jo $(OBJS)
```

scanner.ml: scanner.mll

```
ocamllex scanner.mll
```

```
parser.ml parser.mli: parser.mly
  ocaml yacc parser.mly
```

```
%.cmo: %.ml
  ocamlc -c $<
```

```
%.cmi: %.mli
  ocamlc -c $<
```

```
.PHONY: clean
clean:
```

```
  rm -rf jo parser.ml parser.mli scanner.ml *.cmo *.cmi *.log *.o *~ a.out* core
```

```
# Generated by ocamldep *.ml *.mli
analyzer.cmo: symboltable.cmo ast.cmi sast.cmi
analyzer.cmx: symboltable.cmx ast.cmx sast.cmx
jo.cmo: scanner.cmo parser.cmi ast.cmi Str.cma
jo.cmx: scanner.cmx parser.cmx ast.cmi
parser.cmo: ast.cmi parser.cmi
parser.cmx: ast.cmi parser.cmi
scanner.cmo: parser.cmi
scanner.cmx: parser.cmx
parser.cmi: ast.cmi
Str.cma:
```

**README.md**

plt2014fall

=====

Columbia University Academic Project for Programming Language and Translators

### **analyzer.ml**

```
open Ast
```

```
open Symboltable
```

```
(*adding testing comment *)
```

```
module StringMap = Map.Make(String)
```

```
let string_of_vtype = function
```

```
  IntType -> "number"
```

```
  | StrType -> "string"
```

```
  | BoolType -> "bool"
```

```
  | JsonType -> "json"
```

```
  | ListType -> "list"
```

```
  | NoType   -> "notype"
```

```
let vtype_of_ocaml_type = function
```

```
  "number" -> IntType
```

```
  | "string" -> StrType
```

```
  | "bool" -> BoolType
```

```
  | "json" -> JsonType
```

```
  | "list" -> ListType
```

```
  | "notype" -> NoType
```

```
  | _ -> NoType
```

```
let get_sast_type = function
  Ast.JsonType -> Sast.JsonType
  | Ast.StrType -> Sast.StrType
  | Ast.IntType -> Sast.IntType
  | Ast.BoolType -> Sast.BoolType
  | Ast.ListType -> Sast.ListType
  | Ast.NoType -> Sast.NoType
```

(\* get variable type according to the name

\* raise error if no name matching in variable list \*)

```
let get_vtype env id =
```

(\* find\_variable method is from the symbol table \*)

```
let t = find_variable id env in
```

```
if t = "" then raise (Failure ("undefined variable: " ^ id)) else t
```

(\* get the type of expression:

\* -> string if one of the two operands having string type

\* -> number/boolean if both of the operands having the same type \*)

```
let get_oper_type t1 t2 =
```

```
if t1 = "json" then "json" else
```

```
if t1 = "list" then "list" else
```

```
if t1 = "string" || t2 = "string" then "string" else
```

```
if t1 = "number" && t2 = "number" then "bool" else
```

```
if t1 = "bool" && t2 = "bool" then "bool" else
```

```
if t1 = "number" && t2 = "bool" then raise (Failure ("cannot use number with bool type inside expression")) else
```



```
    if t1 = "bool" && t2 = "number" then raise (Failure ("cannot use number with bool type inside expression")) else
```

```
    raise (Failure ("type error in get_oper_type"))
```

```
(* TODO ----- NEED TO have checks for boolean op and mathematical op
*)
```

```
let get_bool_equal_oper_type t1 t2 =
```

```
  if t1 = "notype" then "bool" else
```

```
  if t2 = "notype" then "bool" else
```

```
  if t1 = "json" && t2 = "json" then "bool" else
```

```
  if t1 = "list" && t2 = "list" then "bool" else
```

```
  if t1 = "string" && t2 = "string" then "bool" else
```

```
  if t1 = "number" && t2 = "number" then "bool" else
```

```
  if t1 = "bool" && t2 = "bool" then "bool" else
```

```
  raise (Failure ("Cannot Compare different types"))
```

```
let get_math_oper_type t1 t2 =
```

```
  if t1 = "notype" && t2 = "notype" then "notype" else
```

```
  if t1 = "notype" then t2 else
```

```
  if t2 = "notype" then t1 else
```

```
  if t1 = "number" && t2 = "number" then "number" else
```

```
  if t1 = "list" && t2 = "list" then "list" else
```

```
  raise (Failure ("Mathematical operator work on Number types only, ++ works on Number and Lists"))
```

```
let get_logical_oper_type t1 t2 =
```

```
  if t1 = "notype" then t2 else
```

```
if t2 = "notype" then t1 else
if t1 = "bool" && t2 = "bool" then "bool" else
raise (Failure ("Logical operators can work on only bool types"))
```

```
let get_add_oper_type t1 t2 =
  if t1 = "notype" && t2 = "notype" then "number" else
  if t1 = "notype" then t2 else
  if t2 = "notype" then t1 else
  if t1 = "number" && t2 = "number" then "number" else
  if t1 = "string" && t2 = "string" then "string" else
  raise (Failure ("Add operator can work on numbers or strings"))
```

```
let get_comp_oper_type t1 t2 =
  if t1 = "notype" then t2 else
  if t2 = "notype" then t1 else
  if t1 = "number" && t2 = "number" then "bool" else
  raise (Failure ("comparison operators can work on only Number types types"))
```

```
let get_in_oper_type t1 t2 =
  if not(t2 = "list" || t2 = "notype") then raise (Failure ("The 'in' and 'not in' statements must be
checking in a list!"))
  else "bool"
```

```
let check_listexpr env = function
  Ast.ListItemInt(i) -> Sast.ListItemInt(i), "number"
| Ast.ListItemStr(s) -> Sast.ListItemStr(s), "string"
```

(\* Might have to do type checking \*)

let check\_func\_arg lst expr arg\_t = (fst expr)::lst

let match\_oper e1 op e2 =

(\* snd of expr is type \*)

(\*let expr\_t = get\_oper\_type (snd e1) (snd e2) in\*)

(match op with

Ast.Add -> let expr\_t = get\_add\_oper\_type (snd e1) (snd e2) in

if (expr\_t = "number" || expr\_t = "notype") then (Sast.Binop(fst e1, Sast.Add, fst e2), "number") else

if expr\_t = "string" then (Sast.Binop(fst e1, Sast.Add, fst e2), "string") else

raise (Failure ("Add operator can work on number or strings"))

| Ast.Sub -> let expr\_t = get\_math\_oper\_type (snd e1) (snd e2) in

if (expr\_t = "number" || expr\_t = "notype") then (Sast.Binop(fst e1, Sast.Sub, fst e2), "number") else

raise (Failure ("Mathematical operator work on Number types only"))

| Ast.Mult -> let expr\_t = get\_math\_oper\_type (snd e1) (snd e2) in

if (expr\_t = "number" || expr\_t = "notype") then (Sast.Binop(fst e1, Sast.Mult, fst e2), "number") else

raise (Failure ("Mathematical operator work on Number types only"))

| Ast.Div -> let expr\_t = get\_math\_oper\_type (snd e1) (snd e2) in

if (expr\_t = "number" || expr\_t = "notype") then (Sast.Binop(fst e1, Sast.Div, fst e2), "number") else

raise (Failure ("Mathematical operator work on Number types only"))

| Ast.Mod -> let expr\_t = get\_math\_oper\_type (snd e1) (snd e2) in

if (expr\_t = "number" || expr\_t = "notype") then (Sast.Binop(fst e1, Sast.Mod, fst e2), "number") else

raise (Failure ("Mathematical operator work on Number types only"))

(\* equal and not equal have special case for string comparison

we may need to add SAST and Eqs and Neqs \*)

| Ast.Equal -> let expr\_t = get\_bool\_equal\_oper\_type (snd e1) (snd e2) in

```

                if (expr_t = "bool" || expr_t = "notype") then (Sast.Binop(fst e1, Sast.Equal, fst
e2), "bool") else
                    raise (Failure ("Cannot Compare different types"))
            | Ast.Neq -> let expr_t = get_bool_equal_oper_type (snd e1) (snd e2) in
                if (expr_t = "bool" || expr_t = "notype") then (Sast.Binop(fst e1, Sast.Neq, fst
e2), "bool") else
                    raise (Failure ("Cannot Compare different types"))
            | Ast.Less -> let expr_t = get_comp_oper_type (snd e1) (snd e2) in
                if (expr_t = "bool" || expr_t = "notype") then (Sast.Binop(fst e1, Sast.Less,
fst e2), "bool") else
                    raise (Failure ("Comparison operators can work on only Number types types"))
            | Ast.Leq -> let expr_t = get_comp_oper_type (snd e1) (snd e2) in
                if (expr_t = "bool" || expr_t = "notype") then (Sast.Binop(fst e1, Sast.Leq, fst
e2), "bool") else
                    raise (Failure ("Comparison operators can work on only Number types types"))
            | Ast.Greater -> let expr_t = get_comp_oper_type (snd e1) (snd e2) in
                if (expr_t = "bool" || expr_t = "notype") then (Sast.Binop(fst e1, Sast.Greater,
fst e2), "bool") else
                    raise (Failure ("Comparison operators can work on only Number types types"))
            | Ast.Geq -> let expr_t = get_comp_oper_type (snd e1) (snd e2) in
                if (expr_t = "bool" || expr_t = "notype") then (Sast.Binop(fst e1, Sast.Geq, fst
e2), "bool") else
                    raise (Failure ("Comparison operators can work on only Number types types"))
            | Ast.And -> let expr_t = get_logical_oper_type (snd e1) (snd e2) in
                if (expr_t = "bool" || expr_t = "notype") then (Sast.Binop(fst e1, Sast.And, fst
e2), "bool") else
                    raise (Failure ("Logical operators can work on only bool types"))
            | Ast.Or -> let expr_t = get_logical_oper_type (snd e1) (snd e2) in
                if (expr_t = "bool" || expr_t = "notype") then (Sast.Binop(fst e1, Sast.Or, fst e2),
"bool") else
                    raise (Failure ("Logical operators can work on only bool types"))
            | Ast.In -> let expr_t = get_in_oper_type (snd e1) (snd e2) in

```

```

    if (expr_t = "bool" || expr_t = "notype") then (Sast.Binop(fst e1, Sast.In, fst e2),
"bool") else
        raise (Failure ("The 'in' and 'not in' statements must be checking in a
list!"))
    | Ast.NotIn -> let expr_t = get_in_oper_type (snd e1) (snd e2) in
        if (expr_t = "bool" || expr_t = "notype") then (Sast.Binop(fst e1, Sast.NotIn, fst
e2), "bool") else
            raise (Failure ("The 'in' and 'not in' statements must be checking in a
list!"))
    | Ast.Concat -> (Sast.Binop(fst e1, Sast.Concat, fst e2), "list")
    | Ast.Minus -> if (snd e1) = "list" || (snd e1) = "notype" then (Sast.Binop(fst e1, Sast.Minus, fst
e2), "list") else
        if (snd e1) = "json" || (snd e1) = "notype" then (Sast.Binop(fst e1, Sast.Minus, fst e2),
"json") else
            raise (Failure ("Minus operator can work only on list or json"))
)

```

```

let rec check_list_items env = function

```

```

    Ast.Item(e) -> Sast.Item(fst (check_list_element env e))
    | Ast.Seq(e1, sep, e2) -> Sast.Seq(fst (check_list_element env e1), Sast.Comma,
(check_list_items env e2))
    | Ast.Noitem -> Sast.Noitem
and check_list_element env = function
    Ast.LitIntElem(i) -> Sast.LitIntElem(i), "number"
    | Ast.LitStrElem(s) -> Sast.LitStrElem(s), "string"
    | Ast.LitListOfList(items) -> Sast.LitListOfList(check_list_items env items), "list"
    | Ast.LitJsonOfList(items) -> Sast.LitJsonOfList(check_json_items env items), "json"
    | Ast.LitBoolElem(i) -> Sast.LitBoolElem(i), "bool"
    | Ast.LitNullElem(s) -> Sast.LitNullElem(s), "null"

```

```

and check_json_items env = function

```

```

    Ast.JsonItem(e) -> Sast.JsonItem(check_json_keyValue env e)

```

| Ast.JsonSeq(e1, sep, e2) -> Sast.JsonSeq((check\_json\_keyValue env e1), Sast.Comma,  
(check\_json\_items env e2))

| Ast.NoJsonItem -> Sast.NoJsonItem

and check\_json\_keyValue env = function

Ast.JsonValPair(e1, colon, e2) -> Sast.JsonValPair(fst (check\_json\_key env e1) , Sast.Colon,  
fst (check\_json\_value env e2))

and check\_json\_value env = function

Ast.LitIntJsonVal(i) -> Sast.LitIntJsonVal(i), "number"

| Ast.LitStrJsonVal(s) -> Sast.LitStrJsonVal(s), "string"

| Ast.LitJsonOfJson(items) -> Sast.LitJsonOfJson(check\_json\_items env items), "json"

| Ast.LitListOfJson(items) -> Sast.LitListOfJson(check\_list\_items env items), "list"

| Ast.LitBoolJsonVal(i) -> Sast.LitBoolJsonVal(i), "bool"

| Ast.LitNullJsonVal(s) -> Sast.LitNullJsonVal(s), "null"

and check\_json\_key env = function

Ast.LitStrJsonKey(i) -> Sast.LitStrJsonKey(i), "string"

(\* it returns the expr and its type \*)

let rec check\_expr env = function

Ast.LitInt(i) -> Sast.LitInt(i), "number"

| Ast.LitStr(s) -> Sast.LitStr(s), "string"

| Ast.LitJson(items) -> Sast.LitJson(check\_json\_items env items), "json"

| Ast.LitList(items) -> Sast.LitList(check\_list\_items env items), "list"

```

| Ast.LitBool(s) -> Sast.LitBool(s), "bool"

| Ast.LitNull(s) -> Sast.LitNull(s), "null"

| Ast.Id(id) -> Sast.Id(id), (get_vtype env id)

| Ast.Not(e1) ->
  let ret = check_expr env e1 in
  if (snd ret) = "bool" then Sast.Not(fst ret), "bool"
  else raise (Failure("! is applicable to bool expressions only"))

| Ast.Binop(e1, op, e2) -> match_oper (check_expr env e1) op (check_expr env e2)

| Ast.Call(func, el) ->
  (* find_function is from the symbol table *)
  let args = find_function func env in (* return & arguments type list from definition *)
  ( match args with
    [] -> raise (Failure ("undefined function " ^ func))
  | hd::tl -> let new_list = try List.fold_left2 check_func_arg [] (List.map (check_expr env) el)
tl
              with Invalid_argument "arg" -> raise(Failure("unmatched argument list"))
              in Sast.Call(func, List.rev new_list ), "notype" )

| Ast.ElemAccess(id, e) -> let t1 = get_vtype env id in
                          let t2 = check_expr env e in
                          if not ( (t1 = "notype" && (snd t2 = "string" || snd t2 = "notype"
|| snd t2 = "number")) || (t1 = "json" && (snd t2 = "string" || snd t2 = "notype")) || (t1="list" && (snd
t2 = "number" || snd t2 = "notype"))) )
                          then raise (Failure("Elements of List and Json can be
accessed via index and key respectively"))
                          else
                          Sast.ElemAccess (id, (fst t2)), "notype"

```

```

| Ast.TypeStruct(id) -> Sast.TypeStruct(id), "string"
| Ast.AttrList(id) -> Sast.AttrList(id), "list"
| Ast.DataType(expr) -> let (expr, expr_type) = check_expr env expr in
    (Sast.DataType(expr, expr_type)), "string"
| Ast.Read(str) -> Sast.Read(str), "json"
| Ast.MakeString(expr) -> let (expr, expr_type) = check_expr env expr in
    (Sast.MakeString(expr, expr_type)), "string"
| Ast.NoExpr -> Sast.NoExpr, "notype"

```

```

and get_expr_with_type env expr =
  let e = check_expr env expr in snd e

```

(\* convert a variable to its SAST type \*)

```

let convert_to_sast_type x env =
  let s_expr =
    if not (x.vexpr = Ast.NoExpr) then
      fst(check_expr env x.vexpr)
    else Sast.NoExpr
  in
  {
    Sast.vtype = get_sast_type x.vtype;
    Sast.vname = x.vname;
    Sast.vexpr = s_expr;
  }

```

```

let convert_to_vdecl_type x =
  {
    vtype = NoType;
  }

```



```
vname = x;  
vexpr = NoExpr;  
}
```

```
let check_formal env formal =
```

```
  let ret = update_local formal.vname (vtype_of_ocaml_type (get_expr_with_type env  
formal.vexpr)) env in
```

```
  let env = {locals = fst ret; globals = env.globals; functions = env.functions } in  
  convert_to_sast_type formal env, env
```

```
let rec check_formals env formals =
```

```
  match formals with
```

```
    [] -> []
```

```
  | hd::tl -> let f, e = (check_formal env hd) in (f, e)::(check_formals e tl)
```

```
let check_local env local =
```

```
  let ret = update_local local.vname (vtype_of_ocaml_type (get_expr_with_type env  
local.vexpr)) env in
```

```
  let env = {locals = fst ret; globals = env.globals; functions = env.functions } in  
  convert_to_sast_type local env, env
```

```
let rec check_locals env locals =
```

```
  match locals with
```

```
    [] -> []
```

```
  | hd::tl -> let l, e = (check_local env hd) in (l, e)::(check_locals e tl)
```

```
let check_forexpr env = function
```

```
Ast.Forid(id) -> Sast.Forid(id), get_vtype env id
| Ast.AttrList(id) -> Sast.AttrList(id), "list"
```

```
let check_loopvar env = function
```

```
  Ast.LoopVar(id) -> let loopId = convert_to_vdecl_type id in
    let l, e = (check_local env loopId) in e, Sast.LoopVar(id)
```

```
let rec check_stmt env func = function
```

```
  Ast.Vdecl(vdecl) -> let ret = update_local vdecl.vname (vtype_of_ocaml_type
(get_expr_with_type env vdecl.vexpr)) env in
```

```
    if (snd ret) = "exist" then let env = {locals = fst ret; globals = env.globals;
functions = env.functions } in
```

```
      Sast.Assign(vdecl.vname, fst (check_expr env vdecl.vexpr)), env
```

```
    else let env = {locals = fst ret; globals = env.globals; functions =
env.functions } in
```

```
      Sast.Vdecl(convert_to_sast_type vdecl env), env
```

```
| Ast.Expr(expr) -> Sast.Expr(fst (check_expr env expr)), env
```

```
| Ast.Return(expr) -> let e = check_expr env expr in Sast.Return(fst e), env
```

```
| Ast.Print(expr) -> let (expr, expr_type) = check_expr env expr in Sast.Print(expr , expr_type),
env
```

```
| Ast.Block(stmt_list) -> Sast.Block(check_stmt_list env func stmt_list), env
```

```

| Ast.If(expr, stmt1, stmt2) -> let e = check_expr env expr in
    if not(snd e = "bool" || snd e = "notype") then raise (Failure ("The type of the
condition in If statement must be boolean!"))
    else Sast.If(fst e, fst (check_stmt env func stmt1), fst (check_stmt env func
stmt2)), env

```

```

| Ast.For(expr1, expr2, stmt) -> let envNew, e1 = check_loopvar env expr1 in let e2 =
check_forexpr envNew expr2 in
    if not ( snd e2 = "list" || snd e2 = "notype") then raise (Failure("The type of the
expression in a For statement must be list!"))
    else (Sast.For( e1, fst e2, fst (check_stmt envNew func stmt))), envNew

```

```

| Ast.Write(expr, str) -> let (expr, expr_type) = check_expr env expr in (Sast.Write(expr , str)),
env

```

```

| Ast.ElemAssign(id, expr1, expr2) -> let t1 = get_vtype env id in
    let t2 = check_expr env expr1 in
        if not ( (t1 = "notype" && (snd t2 = "string" || snd t2 = "notype"
|| snd t2 = "number")) ||
                (t1 = "json" && (snd t2 = "string" || snd t2 = "notype")) ||
                (t1="list" && (snd t2 ="number" || snd t2 = "notype")) )
            then raise (Failure("Elements of List and Json can be
accessed via index and key respectively!"))
            else
                Sast.ElemAssign (id, (fst t2), fst (check_expr env expr2)),
env

```

and check\_stmt\_list env func = function

```

[] -> []

```

```
| hd::tl -> let s,e = (check_stmt env func hd) in s::(check_stmt_list e func tl)
```

(\* this function will return the updated formals and body as per the abstract syntax tree, the return type, name and locals \*)

```
let check_function env func =
```

```
  match List.hd (List.rev func.body) with
```

```
  Return(_) ->
```

```
    let env = {locals = StringMap.empty; globals = env.globals; functions = env.functions } in
```

```
    (* ret is new env *)
```

```
    let ret = add_function func.fname func.return func.formals env in
```

```
    if StringMap.is_empty ret then raise (Failure ("function " ^ func.fname ^ " is already defined"))
```

```
    (* update the env with functions from ret *)
```

```
    else let env = {locals = env.locals; globals = env.globals; functions = ret } in
```

```
    (* check the formal arguments, returns formal list appended with their env *)
```

```
    let f = check_formals env func.formals in
```

```
    (* get the list of formals from f *)
```

```
    let formals = List.map (fun formal -> fst formal) f in
```

```
    (* get the final env from the last formal *)
```

```
    let env =
```

```
      ( match f with
```

```
        [] -> env
```

```
        | _ -> snd (List.hd (List.rev f)) ) in
```

```
    (* get the stmt list from body *)
```

```
    let body = check_stmt_list env func func.body in
```

```
      { Sast.return = get_sast_type func.return;
```

```
        Sast.fname = func.fname;
```

```
        Sast.formals = formals;
```

```
        Sast.body = body
```

```
}, env
```

```
| _ ->
```

```
  if not (func.fname = "main") then raise (Failure ("Return statement is missing for function  
" ^ func.fname ^ ""))
```

```
else
```

```
let env = {locals = StringMap.empty; globals = env.globals; functions = env.functions } in
```

```
  (* ret is new env *)
```

```
  let ret = add_function func.fname func.return func.formals env in
```

```
  if StringMap.is_empty ret then raise (Failure ("function " ^ func.fname ^ " is already  
defined"))
```

```
  (* update the env with functions from ret *)
```

```
  else let env = {locals = env.locals; globals = env.globals; functions = ret } in
```

```
  (* check the formal arguments, returns formal list appended with their env *)
```

```
  let f = check_formals env func.formals in
```

```
  (* get the list of formals from f *)
```

```
  let formals = List.map (fun formal -> fst formal) f in
```

```
  (* get the final env from the last formal *)
```

```
  let env =
```

```
    ( match f with
```

```
      [] -> env
```

```
      | _ -> snd (List.hd (List.rev f)) ) in
```

```
  (* get the stmt list from body *)
```

```
  let body = check_stmt_list env func func.body in
```

```
    { Sast.return = get_sast_type func.return;
```

```
      Sast.fname = func.fname;
```

```
      Sast.formals = formals;
```

```
      Sast.body = body
```

```
    }, env
```

```
let check_main_function funcs =  
  List.fold_left (fun s e -> s || (e.fname="main")) false funcs
```

```
let rec check_functions env funcs =  
  match funcs with  
  [] -> []  
  | hd::tl -> let f, e = (check_function env hd) in f::(check_functions e tl)
```

(\* returns the global and its env \*)

```
let check_global env global =  
  let ret = update_global global.vname (vtype_of_ocaml_type (get_expr_with_type env  
global.vexpr)) env in  
  (* update the env with globals from ret *)  
  let env = {locals = env.locals; globals = ret; functions = env.functions } in  
  convert_to_sast_type global env, env
```

```
let rec check_globals env globals =  
  match globals with  
  [] -> []  
  | hd::tl -> let g, e = (check_global env hd) in (g, e)::(check_globals e tl)
```

```
let check_program (globals, funcs) =  
  (* create the default environment *)  
  if not (check_main_function funcs) then raise (Failure ("main function is not defined in the  
program"))  
  else
```

```

let env = { locals = StringMap.empty;
           globals = StringMap.empty;
           functions = StringMap.empty }
in
(* return the list of each global appended with its environments, the last global has the final
env *)
let g = check_globals env globals in
(* make a list of globals *)
let globals = List.map (fun global -> fst global) g in
match g with
(* no globals *)
[] -> (globals, (check_functions env (List.rev funcs)))
(* get the environment from the last global *)
| _ -> let e = snd (List.hd (List.rev g)) in (globals, (check_functions e (List.rev funcs)))

```

### **ast.mli**

```

type op = Add | Sub | Mult | Div | Equal | Neq | Less | Leq | Greater | Geq | And | Or | Concat |
Minus | Mod | In | NotIn

```

```

type data_type = StrType | IntType | BoolType | JsonType | ListType | NoType

```

```

type sep = Comma

```

```

type colon = Colon

```

```

type list_expr =

```

ListItemInt of float  
| ListItemStr of string

type list\_element =

LitIntElem of float  
| LitStrElem of string  
| LitListOfList of items  
| LitJsonOfList of json\_items  
| LitBoolElem of string  
| LitNullElem of string

and items =

Item of list\_element  
| Seq of list\_element \* sep \* items  
| Noitem

and json\_key\_type =

LitStrJsonKey of string

and json\_item\_value =

LitIntJsonVal of float  
| LitStrJsonVal of string  
| LitJsonOfJson of json\_items  
| LitListOfJson of items  
| LitBoolJsonVal of string  
| LitNullJsonVal of string

and json\_item =

JsonValPair of json\_key\_type \* colon \* json\_item\_value

and json\_items =

JsonItem of json\_item  
| JsonSeq of json\_item \* sep \* json\_items  
| NoJsonItem



```
type expr =  
  LitInt of float  
| LitStr of string  
| LitJson of json_items  
| LitList of items  
  | LitBool of string  
  | LitNull of string  
| Id of string  
| Not of expr  
| Binop of expr * op * expr  
  | Call of string * expr list  
  | ElemAccess of string * expr  
  | TypeStruct of string  
  | AttrList of string  
| DataType of expr  
  | Read of string  
| MakeString of expr  
| NoExpr
```

```
type var_decl = {  
  vtype : data_type;  
  vname : string;  
  vexpr : expr;  
}
```

```
type for_expr =  
  Forid of string
```

| AttrList of string

type loop\_var =  
    LoopVar of string

type stmt =  
    Block of stmt list  
    | Vdecl of var\_decl  
    | Expr of expr  
    | Return of expr  
    | Print of expr  
    | If of expr \* stmt \* stmt  
    | For of loop\_var \* for\_expr \* stmt  
    | Write of expr \* string  
    | ElemAssign of string \* expr \* expr

type func\_decl = {  
    return : data\_type;  
    fname : string;  
    formals : var\_decl list;  
    body : stmt list;  
}

type program = var\_decl list \* func\_decl list

**cleanall.sh**

```
make -f MakePreProc clean
```

```
make -f Makefile clean
```

```
rm -rf testfiles/*.error
```

```
rm -rf testfiles/*.o
```

```
rm -rf testfiles/*.cpp
```

```
rm -rf testfiles/*.pjo
```

```
rm -rf cpp/*.o
```

```
rm -rf testfiles/*.out
```

```
rm -rf testfiles/*.output
```

## jo.ml

```
open Sast
```

```
let rec string_of_items = function
```

```
  Item(e) -> string_of_elements e
```

```
  | Seq(e, sep, i2) -> string_of_elements e ^ ","
```

```
    ^ (string_of_items i2)
```

```
  | Noitem -> ""
```

```
and string_of_elements = function
```

```
  LitIntElem(l) -> string_of_float l ^ "0" (* ocaml prints 5 to 5. *)
```

```
  | LitStrElem(l) -> Str.global_replace (Str.regexp "\\") "\\\\" l
```

```
  | LitListOfList(l) -> "[" ^ string_of_items l ^ "]"
```

```
  | LitJsonOfList(l) -> "{" ^ json_items l ^ "}"
```

```
  | LitBoolElem(l) -> l
```

```
  | LitNullElem(l) -> l
```

and json\_items = function

  JsonItem(e) -> json\_key\_value e

| JsonSeq(e, sep, i2) -> json\_key\_value e ^ ","

  ^ (json\_items i2)

| NoJsonItem -> ""

and json\_key\_value = function

  JsonValPair(e1, colon, e2) -> json\_key e1 ^ ":" ^ json\_value e2

and json\_key = function

  LitStrJsonKey(l) -> Str.global\_replace (Str.regexp "\\") "\\\" l

and json\_value = function

  LitIntJsonVal(l) -> string\_of\_float l ^ "0" (\* ocaml prints 5 to 5. \*)

| LitStrJsonVal(l) -> Str.global\_replace (Str.regexp "\\") "\\\" l

| LitJsonOfJson(l) -> "{" ^ json\_items l ^ "}"

| LitListOfJson(l) -> "[" ^ string\_of\_items l ^ "]"

| LitBoolJsonVal(l) -> l

| LitNullJsonVal(l) -> l

let get\_for\_id e = match e with

  Forid(id) -> id

| AttrList(id) -> ("^id ^ "-> getAttrList())"

let string\_of\_loop\_var\_t = function

  LoopVar(l) -> l

let rec string\_of\_expr e = match e with

  LitInt(l) -> "CustType::parse(\"\" ^ string\_of\_float l ^ "0" ^ "\",\"NUMBER\")" (\* ocaml prints 5 to 5. \*)

| LitStr(l) -> "CustType::parse(" ^ l ^ "\",\"STRING\")"

```

| LitJson(l) -> "CustType::parse(\{" ^ json_items | ^ "\},\JSON\)"
| LitList(l) -> "CustType::parse(\[" ^ string_of_items | ^ "\],\LIST\)"
| LitBool(l) -> "CustType::parse(\\" ^ l ^ "\",\BOOL\)"
| LitNull(l) -> "CustType::parse(\\" ^ l ^ "\",\NULL\)"
| Id(s) -> s
| Not(e1) -> "!(*(" ^ string_of_expr e1 ^ "))"
| Binop(e1, o, e2) ->

```

( match o with

```

  Add -> "CustType::add("^string_of_expr e1 ^ "," ^ string_of_expr e2 ^)")
  | Sub -> "CustType::subtract("^string_of_expr e1 ^ "," ^ string_of_expr e2 ^)")
  | Mult -> "CustType::multiply("^string_of_expr e1 ^ "," ^ string_of_expr e2 ^)")
  | Div -> "CustType::divide("^string_of_expr e1 ^ "," ^ string_of_expr e2 ^)")
  | Mod -> "CustType::mod("^string_of_expr e1 ^ "," ^ string_of_expr e2 ^)")
  | Or -> "*"("^string_of_expr e1 ^ ") " ^ "||" ^ " " *("^string_of_expr e2 ^)")
  | And -> "*"("^string_of_expr e1 ^ ") " ^ "&&" ^ " " *("^string_of_expr e2 ^)")
  | Geq -> "*"("^string_of_expr e1 ^ ") " ^ ">=" ^ " " *("^string_of_expr e2 ^)")
  | Leq -> "*"("^string_of_expr e1 ^ ") " ^ "<=" ^ " " *("^string_of_expr e2 ^)")
  | Greater -> "*"("^string_of_expr e1 ^ ") " ^ ">" ^ " " *("^string_of_expr e2 ^)")
  | Less -> "*"("^string_of_expr e1 ^ ") " ^ "<" ^ " " *("^string_of_expr e2 ^)")
  | Equal -> "*"("^string_of_expr e1 ^ ") " ^ "==" ^ " " *("^string_of_expr e2 ^)")
  | Neq -> "*"("^string_of_expr e1 ^ ") " ^ "!=" ^ " " *("^string_of_expr e2 ^)")
  | Concat -> "CustType::concat("^string_of_expr e1 ^ "," ^ string_of_expr e2 ^)"
  | Minus -> string_of_expr e1 ^ "->minus("^string_of_expr e2 ^)"
  | In -> string_of_expr e2 ^ "->contains("^string_of_expr e1 ^)"
  | NotIn -> "!(*("^string_of_expr e2 ^ "->contains("^string_of_expr e1 ^)))"

```

)

```

(*| Assign(v, e) -> v ^ " = " ^ string_of_expr e ^ ";"*)
| Call(f, el) ->
  f ^ "(" ^ String.concat " , " (List.map string_of_expr el) ^ ")"

```

```

| ElemAccess(id, e) -> id ^ "-> getElementByOcaml("^string_of_expr e ^ ")"
| TypeStruct(id) -> "CustType::typeStruct(" ^ id ^ ")"
| AttrList(id) -> id ^ "-> getAttrList()"
| DataType(expr, expr_type) -> "(" ^ string_of_expr expr ^ ")->getJoType()"
| Read(str) -> "CustType::read(" ^ str ^ ")"
| MakeString(expr, expr_type) -> "(" ^ string_of_expr expr ^ ")->makeString()"
| NoExpr -> ""

```

```

let string_of_vtype = function

```

```

  IntType -> "CustType*"
| StrType -> "CustType*"
| BoolType -> "CustType*"
| ListType -> "CustType*"
| JsonType -> "CustType*"
| NoType -> "CustType*"

```

```

let string_of_vdecl vdecl =

```

```

  string_of_vtype vdecl.vtype ^ " " ^ vdecl.vname ^ " = " ^ string_of_expr vdecl.vexpr ^ ";\n"

```

```

let rec string_of_stmt = function

```

```

  Vdecl(vdecl) -> string_of_vdecl vdecl
| Expr(expr) -> if compare (string_of_expr expr) "" = 0 then "\n" else string_of_expr expr ^ ";"

| Return(expr) -> (*if fname = "main" then "return 0 " else*) " return " ^ string_of_expr expr ^
";"

```

| Print(expr, expr\_type) -> "CustType::print(" ^ string\_of\_expr expr ^ ");\n"

| Block(stmts) -> "{\n" ^ String.concat "" (List.map string\_of\_stmt stmts) ^ "\n}"

| For(e1, e2, s1) -> "for (vector<CustType\*> :: iterator loopVar" ^ string\_of\_loop\_var\_t e1 ^ "  
" ^ get\_for\_id e2 ^ "->getListBegin()); loopVar" ^ string\_of\_loop\_var\_t e1 ^ " != " ^ get\_for\_id e2 ^ "  
"->getListEnd()); " ^ "++loopVar" ^ string\_of\_loop\_var\_t e1 ^ ") {\n CustType\* "  
string\_of\_loop\_var\_t e1 ^ " = \*loopVar" ^ string\_of\_loop\_var\_t e1 ^ ";\n"  
^ string\_of\_stmt s1 ^ "\n}\n"

| If(e, s, Block([])) -> "if ((" ^ string\_of\_expr e ^ "->getBoolValue())\n" ^ string\_of\_stmt s

| If(e, s1, s2) -> "if ((" ^ string\_of\_expr e ^ "->getBoolValue())\n" ^ string\_of\_stmt s1 ^ "  
"else\n" ^ string\_of\_stmt s2

| Write(expr, str) -> "CustType::write(" ^ string\_of\_expr expr ^ ", " ^ str ^ ");\n"

| Assign(v, e) -> v ^ " = " ^ string\_of\_expr e ^ ";\n"

| ElemAssign(id, expr1, expr2) -> id ^ "->addByKey(" ^ string\_of\_expr expr1 ^ "  
", " ^ string\_of\_expr expr2 ^ ");\n"

let string\_of\_formaldecl vdecl = string\_of\_vtype vdecl.vtype ^ " " ^ vdecl.vname

```

let string_of_fdecl fdecl = (if fdecl.fname = "main" then
"int " ^ fdecl.fname ^ "("
else
string_of_vtype fdecl.return ^ " " ^ fdecl.fname ^ "(" )
^ String.concat ", " (List.map string_of_formaldecl fdecl.formals) ^ ")\\n{\\n" ^
String.concat "" (List.map string_of_stmt fdecl.body ) ^
"}\\n"
let string_of_program (vars, funcs) =
"\\n#include <iostream>\\n#include \"../cpp/cPlusPlusCompiler.h\"\\nusing namespace std;\\n\\n" ^
String.concat "\\n" (List.map string_of_vdecl vars) ^ "\\n" ^
String.concat "\\n" (List.map string_of_fdecl funcs) ^ "\\n"

let _ =
(* first argument is the filename *)
let fname = Sys.argv.(1) in
(* check the extension *)
let index = (if String.contains fname '.' then String.rindex fname '.' else 0 ) in
let suffix = String.sub fname index 4 in
if not (suffix = ".pjo") then raise (Failure ("Invalid type of source file.))
else
let input = open_in fname in
let lexbuf = Lexing.from_channel input in
let program = Parser.program Scanner.token lexbuf in
(* added the type check *)
let program_t = Analyzer.check_program program in
let output = string_of_program program_t in
print_endline output

```



## parser.mly

%{ open Ast %}

%token LPAREN RPAREN LBRACE RBRACE LBRACK RBRACK COMMA SEMI

%token PLUS MINUS TIMES DIVIDE ASSIGN ACCESS COMPLUS COMMUNUS COLON

%token EQ NEQ LT GT LEQ GEQ NOT MOD

%token RETURN IF ELSE HASH NULL

%token AND OR FOR IN NOTIN

%token FUNC END DECL

%token NOTIN READ PRINT TYPE TYPESTRUCT JOIN MAKESTRING ATTRLIST WRITE

%token <float> NUM\_LIT

%token <string> STRING\_LIT

%token <string> JSON\_LIT

%token <string> LIST\_LIT

%token <string> BOOL\_LIT

%token <string> ID

%token EOF

%nonassoc NOELSE

%nonassoc ELSE

%right ASSIGN

%left AND OR

%left EQ NEQ

%left LT GT LEQ GEQ MOD

%left PLUS MINUS

%left TIMES DIVIDE

%left COMPLUS COMMINUS

%right NOT NOTIN IN

%start program

%type <Ast.program> program

%%

program:

{ [], [] }

| program vdecl { (\$2 :: fst \$1) , snd \$1 }

| program fdecl { fst \$1, (\$2 :: snd \$1) }

fdecl:

FUNC ID LPAREN formals\_opt RPAREN LBRACE stmt\_list RBRACE

{{

return = Ast.StrType;

fname = \$2;

formals = \$4;

body = List.rev \$7 }}

formals\_opt:

{ [] }

| formal\_list { List.rev \$1 }

formal\_list:

```
formal          { [$1] }  
| formal_list COMMA formal { $3 :: $1 }
```

formal:

```
ID    { { vtype = NoType;  vname = $1;  vexpr = NoExpr; } }
```

vdecl:

```
ID ASSIGN expr SEMI { { vtype = StrType ;  vname = $1;  vexpr = $3 } }
```

stmt\_list:

```
{ [] }  
| stmt_list stmt { $2 :: $1 }
```

rev\_stmt\_list:

```
stmt_list      { List.rev $1 }
```

*/\* using SEMI ';' to separate stmts \*/*

stmt:

```
vdecl          { Vdecl($1) }  
| expr SEMI    { Expr($1) }  
| RETURN expr_opt SEMI      { Return($2) }  
| PRINT LPAREN expr RPAREN SEMI      { Print($3) }  
| FOR loop_var IN for_expr stmt      { For($2, $4, $5 ) }  
| IF LPAREN expr RPAREN stmt ELSE stmt      { If($3, $5, $7) }  
| IF LPAREN expr RPAREN stmt %prec NOELSE      { If($3, $5, Block([])) }
```

```
| LBRACE rev_stmt_list RBRACE          { Block($2) }
| WRITE LPAREN expr COMMA STRING_LIT RPAREN SEMI  { Write($3, $5) }
| ID LBRACK expr RBRACK ASSIGN expr SEMI        { ElemAssign($1, $3, $6) }
```

for\_expr:

```
ID          { Forid($1) }
| ID ACCESS ATTRLIST LPAREN RPAREN  { AttrList($1) }
```

loop\_var:

```
ID          { LoopVar($1) }
```

expr\_opt:

```
{ NoExpr }
| expr    { $1 }
```

expr:

```
| NUM_LIT          { LitInt($1) }
| STRING_LIT       { LitStr($1) }
| HASH LBRACE json_items RBRACE HASH  { LitJson($3) }
| LBRACK list_items RBRACK            { LitList($2) }
| BOOL_LIT         { LitBool($1) }
| NULL             { LitNull("null") }
| ID              { Id($1) }
| NOT LPAREN expr RPAREN  { Not($3) }
| NOT expr        { Not($2) }
| expr COMPLUS   expr    { Binop($1, Concat, $3) }
| expr COMMINUS  expr    { Binop($1, Minus, $3) }
```

```

| expr PLUS expr      { Binop($1, Add, $3) }
| expr MINUS expr    { Binop($1, Sub, $3) }
| expr TIMES expr    { Binop($1, Mult, $3) }
| expr DIVIDE expr   { Binop($1, Div, $3) }
| expr EQ expr       { Binop($1, Equal, $3) }
| expr NEQ expr      { Binop($1, Neq, $3) }
| expr LT expr       { Binop($1, Less, $3) }
| expr LEQ expr      { Binop($1, Leq, $3) }
| expr GT expr       { Binop($1, Greater, $3) }
| expr GEQ expr      { Binop($1, Geq, $3) }
| expr MOD expr      { Binop($1, Mod, $3) }
| expr AND expr      { Binop($1, And, $3) }
| expr OR expr       { Binop($1, Or, $3) }
| expr IN expr       { Binop($1, In, $3) }
| expr NOTIN expr    { Binop($1, NotIn, $3) }
| ID LPAREN actuals_opt RPAREN { Call($1, $3) }
| ID LBRACK expr RBRACK      { ElemAccess($1, $3) }
| ID ACCESS TYPESTRUCT LPAREN RPAREN { TypeStruct($1) }
| ID ACCESS ATTRLIST LPAREN RPAREN   { AttrList($1) }
| TYPE LPAREN expr RPAREN            { DataType($3) }
| READ LPAREN STRING_LIT RPAREN      { Read($3) }
| MAKESTRING LPAREN expr RPAREN      { MakeString($3) }

```

list\_items:

```

{ Noitem }
| list_element          { Item($1) }
| list_element COMMA list_items { Seq($1, Comma, $3) }

```

list\_element:

```
NUM_LIT          { LitIntElem($1) }
| STRING_LIT     { LitStrElem($1) }
| LBRACK list_items RBRACK { LitListOfList($2) }
| LBRACE json_items RBRACE { LitJsonOfList($2) }
| BOOL_LIT       { LitBoolElem($1) }
| NULL           { LitNullElem("null") }
```

json\_items:

```
{ NoJsonItem }
| json_item      { JsonItem($1)}
| json_item COMMA json_items { JsonSeq($1, Comma, $3) }
```

json\_item:

```
json_item_key COLON json_item_value { JsonValPair($1,Colon,$3) }
```

json\_item\_value:

```
NUM_LIT          { LitIntJsonVal($1) }
| STRING_LIT     { LitStrJsonVal($1) }
| LBRACE json_items RBRACE { LitJsonOfJson($2) }
| LBRACK list_items RBRACK { LitListOfJson($2) }
| BOOL_LIT       { LitBoolJsonVal($1) }
| NULL           { LitNullJsonVal("null") }
```

json\_item\_key:

STRING\_LIT            { LitStrJsonKey(\$1) }

actuals\_opt:

{ [] }

| actuals\_list { List.rev \$1 }

actuals\_list:

expr                 { [\$1] }

| actuals\_list COMMA expr { \$3 :: \$1 }

### **parser.output**

0 \$accept : %entry% \$end

1 program :

2     | program vdecl

3     | program fdecl

4 fdecl : FUNC ID LPAREN formals\_opt RPAREN LBRACE vdecl\_opt stmt\_list RBRACE

5 formals\_opt :

6     | formal\_list

7 formal\_list : formal

8     | formal\_list COMMA formal

9 formal : ID

10 vdecl\_opt :

11       | vdecl\_list

12 vdecl\_list : vdecl

13       | vdecl\_list vdecl

14 vdecl : DECL ID ASSIGN expr SEMI

15 stmt\_list :

16       | stmt\_list stmt

17 rev\_stmt\_list : stmt\_list

18 stmt : expr SEMI

19       | RETURN expr\_opt SEMI

20       | PRINT expr SEMI

21       | IF LPAREN expr RPAREN stmt ELSE stmt

22       | IF LPAREN expr RPAREN stmt

23       | LBRACE rev\_stmt\_list RBRACE

24 expr\_opt :



25 | expr

26 list\_expr : NUM\_LIT

27 | STRING\_LIT

28 expr : NUM\_LIT

29 | STRING\_LIT

30 | LBRACE json\_items RBRACE

31 | LBRACK list\_items RBRACK

32 | BOOL\_LIT

33 | ID

34 | expr PLUS expr

35 | expr MINUS expr

36 | expr TIMES expr

37 | expr DIVIDE expr

38 | expr EQ expr

39 | expr NEQ expr

40 | ID ASSIGN expr

41 | ID LPAREN actuals\_opt RPAREN

42 | MAINFUNC

43 | ID LBRACK list\_expr RBRACK

44 list\_items :

45 | list\_element

46 | list\_element COMMA list\_items

47 list\_element : NUM\_LIT

48        | STRING\_LIT  
49        | LBRACK list\_items RBRACK

50 json\_items :

51        | json\_item

52        | json\_item COMMA json\_items

53 json\_item : STRING\_LIT COLON json\_item\_value

54 json\_item\_value : NUM\_LIT

55        | STRING\_LIT

56 actuals\_opt :

57        | actuals\_list

58 actuals\_list : expr

59        | actuals\_list COMMA expr

60 %entry% : '\001' program

state 0

\$accept : . %entry% \$end (0)

'\001' shift 1

. error

%entry% goto 2

state 1

%entry% : '\001' . program (60)

program : . (1)

. reduce 1

program goto 3

state 2

\$accept : %entry% . \$end (0)

\$end accept

state 3

program : program . vdecl (2)

program : program . fdecl (3)

%entry% : '\001' program . (60)

FUNC shift 4  
DECL shift 5  
\$end reduce 60

vdecl goto 6  
fdecl goto 7

state 4

fdecl : FUNC . ID LPAREN formals\_opt RPAREN LBRACE vdecl\_opt stmt\_list RBRACE (4)

ID shift 8  
. error

state 5

vdecl : DECL . ID ASSIGN expr SEMI (14)

ID shift 9  
. error

state 6

program : program vdecl . (2)

. reduce 2

state 7

program : program fdecl . (3)

. reduce 3

state 8

fdecl : FUNC ID . LPAREN formals\_opt RPAREN LBRACE vdecl\_opt stmt\_list RBRACE (4)

LPAREN shift 10

. error

state 9

vdecl : DECL ID . ASSIGN expr SEMI (14)

ASSIGN shift 11

. error

state 10

fdecl : FUNC ID LPAREN . formals\_opt RPAREN LBRACE vdecl\_opt stmt\_list RBRACE (4)

formals\_opt : . (5)

ID shift 12

RPAREN reduce 5

formals\_opt goto 13

formal\_list goto 14

formal goto 15

state 11

vdecl : DECL ID ASSIGN . expr SEMI (14)

LBRACE shift 16

LBRACK shift 17

MAINFUNC shift 18

NUM\_LIT shift 19

STRING\_LIT shift 20

BOOL\_LIT shift 21

ID shift 22

. error

expr goto 23

state 12

formal : ID . (9)

. reduce 9

state 13

fdecl : FUNC ID LPAREN formals\_opt . RPAREN LBRACE vdecl\_opt stmt\_list RBRACE (4)

RPAREN shift 24

. error

state 14

formals\_opt : formal\_list . (6)

formal\_list : formal\_list . COMMA formal (8)

COMMA shift 25

RPAREN reduce 6

state 15

formal\_list : formal . (7)

. reduce 7

state 16

expr : LBRACE . json\_items RBRACE (30)

json\_items : . (50)

STRING\_LIT shift 26

RBRACE reduce 50

json\_items goto 27

json\_item goto 28

state 17

expr : LBRACK . list\_items RBRACK (31)

list\_items : . (44)

LBRACK shift 29

NUM\_LIT shift 30

STRING\_LIT shift 31

RBRACK reduce 44

list\_items goto 32

list\_element goto 33

state 18

expr : MAINFUNC . (42)

. reduce 42



state 19

expr : NUM\_LIT . (28)

. reduce 28

state 20

expr : STRING\_LIT . (29)

. reduce 29

state 21

expr : BOOL\_LIT . (32)

. reduce 32

state 22

expr : ID . (33)

expr : ID . ASSIGN expr (40)

expr : ID . LPAREN actuals\_opt RPAREN (41)

expr : ID . LBRACK list\_expr RBRACK (43)

LPAREN shift 34  
LBRACK shift 35  
ASSIGN shift 36  
RPAREN reduce 33  
COMMA reduce 33  
SEMI reduce 33  
PLUS reduce 33  
MINUS reduce 33  
TIMES reduce 33  
DIVIDE reduce 33  
EQ reduce 33  
NEQ reduce 33

state 23

vdecl : DECL ID ASSIGN expr . SEMI (14)  
expr : expr . PLUS expr (34)  
expr : expr . MINUS expr (35)  
expr : expr . TIMES expr (36)  
expr : expr . DIVIDE expr (37)  
expr : expr . EQ expr (38)  
expr : expr . NEQ expr (39)

SEMI shift 37  
PLUS shift 38  
MINUS shift 39  
TIMES shift 40  
DIVIDE shift 41  
EQ shift 42

NEQ shift 43

. error

state 24

fdecl : FUNC ID LPAREN formals\_opt RPAREN . LBRACE vdecl\_opt stmt\_list RBRACE (4)

LBRACE shift 44

. error

state 25

formal\_list : formal\_list COMMA . formal (8)

ID shift 12

. error

formal goto 45

state 26

json\_item : STRING\_LIT . COLON json\_item\_value (53)

COLON shift 46

. error

state 27

expr : LBRACE json\_items . RBRACE (30)

RBRACE shift 47

. error

state 28

json\_items : json\_item . (51)

json\_items : json\_item . COMMA json\_items (52)

COMMA shift 48

RBRACE reduce 51

state 29

list\_element : LBRACK . list\_items RBRACK (49)

list\_items : . (44)

LBRACK shift 29

NUM\_LIT shift 30

STRING\_LIT shift 31

RBRACK reduce 44

list\_items goto 49

list\_element goto 33

state 30

list\_element : NUM\_LIT . (47)

. reduce 47

state 31

list\_element : STRING\_LIT . (48)

. reduce 48

state 32

expr : LBRACK list\_items . RBRAK (31)

RBRAK shift 50

. error

state 33

list\_items : list\_element . (45)

list\_items : list\_element . COMMA list\_items (46)

COMMA shift 51

RBRACK reduce 45

state 34

expr : ID LPAREN . actuals\_opt RPAREN (41)

actuals\_opt : . (56)

LBRACE shift 16

LBRACK shift 17

MAINFUNC shift 18

NUM\_LIT shift 19

STRING\_LIT shift 20

BOOL\_LIT shift 21

ID shift 22

RPAREN reduce 56

expr goto 52

actuals\_opt goto 53

actuals\_list goto 54

state 35

expr : ID LBRACK . list\_expr RBRACK (43)

NUM\_LIT shift 55

STRING\_LIT shift 56

. error

list\_expr goto 57

state 36

expr : ID ASSIGN . expr (40)

LBRACE shift 16

LBRACK shift 17

MAINFUNC shift 18

NUM\_LIT shift 19

STRING\_LIT shift 20

BOOL\_LIT shift 21

ID shift 22

. error

expr goto 58

state 37

vdecl : DECL ID ASSIGN expr SEMI . (14)

. reduce 14

state 38

expr : expr PLUS . expr (34)

LBRACE shift 16

LBRACK shift 17

MAINFUNC shift 18

NUM\_LIT shift 19

STRING\_LIT shift 20

BOOL\_LIT shift 21

ID shift 22

. error

expr goto 59

state 39

expr : expr MINUS . expr (35)

LBRACE shift 16

LBRACK shift 17

MAINFUNC shift 18

NUM\_LIT shift 19

STRING\_LIT shift 20

BOOL\_LIT shift 21

ID shift 22



. error

expr goto 60

state 40

expr : expr TIMES . expr (36)

LBRACE shift 16

LBRACK shift 17

MAINFUNC shift 18

NUM\_LIT shift 19

STRING\_LIT shift 20

BOOL\_LIT shift 21

ID shift 22

. error

expr goto 61

state 41

expr : expr DIVIDE . expr (37)

LBRACE shift 16

LBRACK shift 17

MAINFUNC shift 18

NUM\_LIT shift 19  
STRING\_LIT shift 20  
BOOL\_LIT shift 21  
ID shift 22  
. error

expr goto 62

state 42

expr : expr EQ . expr (38)

LBRACE shift 16  
LBRACK shift 17  
MAINFUNC shift 18  
NUM\_LIT shift 19  
STRING\_LIT shift 20  
BOOL\_LIT shift 21  
ID shift 22  
. error

expr goto 63

state 43

expr : expr NEQ . expr (39)

LBRACE shift 16  
LBRACK shift 17  
MAINFUNC shift 18  
NUM\_LIT shift 19  
STRING\_LIT shift 20  
BOOL\_LIT shift 21  
ID shift 22  
. error

expr goto 64

state 44

fdecl : FUNC ID LPAREN formals\_opt RPAREN LBRACE . vdecl\_opt stmt\_list RBRACE (4)  
vdecl\_opt : . (10)

DECL shift 5  
LBRACE reduce 10  
RBRACE reduce 10  
LBRACK reduce 10  
RETURN reduce 10  
IF reduce 10  
MAINFUNC reduce 10  
PRINT reduce 10  
NUM\_LIT reduce 10  
STRING\_LIT reduce 10

BOOL\_LIT reduce 10

ID reduce 10

vdecl goto 65

vdecl\_opt goto 66

vdecl\_list goto 67

state 45

formal\_list : formal\_list COMMA formal . (8)

. reduce 8

state 46

json\_item : STRING\_LIT COLON . json\_item\_value (53)

NUM\_LIT shift 68

STRING\_LIT shift 69

. error

json\_item\_value goto 70

state 47

expr : LBRACE json\_items RBRACE . (30)

. reduce 30

state 48

json\_items : json\_item COMMA . json\_items (52)

json\_items : . (50)

STRING\_LIT shift 26

RBRACE reduce 50

json\_items goto 71

json\_item goto 28

state 49

list\_element : LBRACK list\_items . RBRACK (49)

RBRACK shift 72

. error

state 50

expr : LBRACK list\_items RBRACK . (31)

. reduce 31

state 51

list\_items : list\_element COMMA . list\_items (46)

list\_items : . (44)

LBRACK shift 29

NUM\_LIT shift 30

STRING\_LIT shift 31

RBRACK reduce 44

list\_items goto 73

list\_element goto 33

state 52

expr : expr . PLUS expr (34)

expr : expr . MINUS expr (35)

expr : expr . TIMES expr (36)

expr : expr . DIVIDE expr (37)

expr : expr . EQ expr (38)

expr : expr . NEQ expr (39)

actuals\_list : expr . (58)

PLUS shift 38

MINUS shift 39

TIMES shift 40

DIVIDE shift 41

EQ shift 42

NEQ shift 43

RPAREN reduce 58

COMMA reduce 58

state 53

expr : ID LPAREN actuals\_opt . RPAREN (41)

RPAREN shift 74

. error

state 54

actuals\_opt : actuals\_list . (57)

actuals\_list : actuals\_list . COMMA expr (59)

COMMA shift 75

RPAREN reduce 57

state 55

list\_expr : NUM\_LIT . (26)

. reduce 26

state 56

list\_expr : STRING\_LIT . (27)

. reduce 27

state 57

expr : ID LBRACK list\_expr . RBRACK (43)

RBRACK shift 76

. error

state 58

expr : expr . PLUS expr (34)

expr : expr . MINUS expr (35)

expr : expr . TIMES expr (36)

expr : expr . DIVIDE expr (37)

expr : expr . EQ expr (38)

expr : expr . NEQ expr (39)

expr : ID ASSIGN expr . (40)

PLUS shift 38

MINUS shift 39

TIMES shift 40



DIVIDE shift 41

EQ shift 42

NEQ shift 43

RPAREN reduce 40

COMMA reduce 40

SEMI reduce 40

state 59

expr : expr . PLUS expr (34)

expr : expr PLUS expr . (34)

expr : expr . MINUS expr (35)

expr : expr . TIMES expr (36)

expr : expr . DIVIDE expr (37)

expr : expr . EQ expr (38)

expr : expr . NEQ expr (39)

TIMES shift 40

DIVIDE shift 41

RPAREN reduce 34

COMMA reduce 34

SEMI reduce 34

PLUS reduce 34

MINUS reduce 34

EQ reduce 34

NEQ reduce 34

state 60

expr : expr . PLUS expr (34)

expr : expr . MINUS expr (35)

expr : expr MINUS expr . (35)

expr : expr . TIMES expr (36)

expr : expr . DIVIDE expr (37)

expr : expr . EQ expr (38)

expr : expr . NEQ expr (39)

TIMES shift 40

DIVIDE shift 41

RPAREN reduce 35

COMMA reduce 35

SEMI reduce 35

PLUS reduce 35

MINUS reduce 35

EQ reduce 35

NEQ reduce 35

state 61

expr : expr . PLUS expr (34)

expr : expr . MINUS expr (35)

expr : expr . TIMES expr (36)

expr : expr TIMES expr . (36)

expr : expr . DIVIDE expr (37)

expr : expr . EQ expr (38)

expr : expr . NEQ expr (39)

. reduce 36

state 62

expr : expr . PLUS expr (34)

expr : expr . MINUS expr (35)

expr : expr . TIMES expr (36)

expr : expr . DIVIDE expr (37)

expr : expr DIVIDE expr . (37)

expr : expr . EQ expr (38)

expr : expr . NEQ expr (39)

. reduce 37

state 63

expr : expr . PLUS expr (34)

expr : expr . MINUS expr (35)

expr : expr . TIMES expr (36)

expr : expr . DIVIDE expr (37)

expr : expr . EQ expr (38)

expr : expr EQ expr . (38)

expr : expr . NEQ expr (39)

PLUS shift 38

MINUS shift 39

TIMES shift 40

DIVIDE shift 41

RPAREN reduce 38

COMMA reduce 38

SEMI reduce 38

EQ reduce 38

NEQ reduce 38

state 64

expr : expr . PLUS expr (34)

expr : expr . MINUS expr (35)

expr : expr . TIMES expr (36)

expr : expr . DIVIDE expr (37)

expr : expr . EQ expr (38)

expr : expr . NEQ expr (39)

expr : expr NEQ expr . (39)

PLUS shift 38

MINUS shift 39

TIMES shift 40

DIVIDE shift 41

RPAREN reduce 39

COMMA reduce 39

SEMI reduce 39

EQ reduce 39

NEQ reduce 39

state 65

vdecl\_list : vdecl . (12)

. reduce 12

state 66

fdecl : FUNC ID LPAREN formals\_opt RPAREN LBRACE vdecl\_opt . stmt\_list RBRACE (4)

stmt\_list : . (15)

. reduce 15

stmt\_list goto 77

state 67

vdecl\_opt : vdecl\_list . (11)

vdecl\_list : vdecl\_list . vdecl (13)

DECL shift 5

LBRACE reduce 11

RBRACE reduce 11

LBRACK reduce 11

RETURN reduce 11

IF reduce 11  
MAINFUNC reduce 11  
PRINT reduce 11  
NUM\_LIT reduce 11  
STRING\_LIT reduce 11  
BOOL\_LIT reduce 11  
ID reduce 11

vdecl goto 78

state 68

json\_item\_value : NUM\_LIT . (54)

. reduce 54

state 69

json\_item\_value : STRING\_LIT . (55)

. reduce 55

state 70

json\_item : STRING\_LIT COLON json\_item\_value . (53)

. reduce 53

state 71

json\_items : json\_item COMMA json\_items . (52)

. reduce 52

state 72

list\_element : LBRACK list\_items RBRACK . (49)

. reduce 49

state 73

list\_items : list\_element COMMA list\_items . (46)

. reduce 46

state 74

expr : ID LPAREN actuals\_opt RPAREN . (41)

. reduce 41

state 75

actuals\_list : actuals\_list COMMA . expr (59)

LBRACE shift 16

LBRACK shift 17

MAINFUNC shift 18

NUM\_LIT shift 19

STRING\_LIT shift 20

BOOL\_LIT shift 21

ID shift 22

. error

expr goto 79

state 76

expr : ID LBRACK list\_expr RBRACK . (43)

. reduce 43

state 77

fdecl : FUNC ID LPAREN formals\_opt RPAREN LBRACE vdecl\_opt stmt\_list . RBRACE (4)

stmt\_list : stmt\_list . stmt (16)



LBRACE shift 80  
RBRACE shift 81  
LBRACK shift 17  
RETURN shift 82  
IF shift 83  
MAINFUNC shift 18  
PRINT shift 84  
NUM\_LIT shift 19  
STRING\_LIT shift 20  
BOOL\_LIT shift 21  
ID shift 22  
. error

expr goto 85  
stmt goto 86

state 78

vdecl\_list : vdecl\_list vdecl . (13)

. reduce 13

state 79

expr : expr . PLUS expr (34)

expr : expr . MINUS expr (35)

expr : expr . TIMES expr (36)  
expr : expr . DIVIDE expr (37)  
expr : expr . EQ expr (38)  
expr : expr . NEQ expr (39)  
actuals\_list : actuals\_list COMMA expr . (59)

PLUS shift 38  
MINUS shift 39  
TIMES shift 40  
DIVIDE shift 41  
EQ shift 42  
NEQ shift 43  
RPAREN reduce 59  
COMMA reduce 59

80: reduce/reduce conflict (reduce 15, reduce 50) on RBRACE

80: shift/reduce conflict (shift 26, reduce 15) on STRING\_LIT

state 80

stmt : LBRACE . rev\_stmt\_list RBRACE (23)  
expr : LBRACE . json\_items RBRACE (30)  
stmt\_list : . (15)  
json\_items : . (50)

STRING\_LIT shift 26  
LBRACE reduce 15  
RBRACE reduce 15  
LBRACK reduce 15

RETURN reduce 15  
IF reduce 15  
MAINFUNC reduce 15  
PRINT reduce 15  
NUM\_LIT reduce 15  
BOOL\_LIT reduce 15  
ID reduce 15

stmt\_list goto 87  
rev\_stmt\_list goto 88  
json\_items goto 27  
json\_item goto 28

state 81

fdecl : FUNC ID LPAREN formals\_opt RPAREN LBRACE vdecl\_opt stmt\_list RBRACE . (4)

. reduce 4

state 82

stmt : RETURN . expr\_opt SEMI (19)

expr\_opt : . (24)

LBRACE shift 16

LBRACK shift 17

MAINFUNC shift 18

NUM\_LIT shift 19  
STRING\_LIT shift 20  
BOOL\_LIT shift 21  
ID shift 22  
SEMI reduce 24

expr goto 89  
expr\_opt goto 90

state 83

stmt : IF . LPAREN expr RPAREN stmt ELSE stmt (21)  
stmt : IF . LPAREN expr RPAREN stmt (22)

LPAREN shift 91  
. error

state 84

stmt : PRINT . expr SEMI (20)

LBRACE shift 16  
LBRACK shift 17  
MAINFUNC shift 18  
NUM\_LIT shift 19  
STRING\_LIT shift 20  
BOOL\_LIT shift 21

ID shift 22

. error

expr goto 92

state 85

stmt : expr . SEMI (18)

expr : expr . PLUS expr (34)

expr : expr . MINUS expr (35)

expr : expr . TIMES expr (36)

expr : expr . DIVIDE expr (37)

expr : expr . EQ expr (38)

expr : expr . NEQ expr (39)

SEMI shift 93

PLUS shift 38

MINUS shift 39

TIMES shift 40

DIVIDE shift 41

EQ shift 42

NEQ shift 43

. error

state 86

stmt\_list : stmt\_list stmt . (16)

. reduce 16

state 87

stmt\_list : stmt\_list . stmt (16)

rev\_stmt\_list : stmt\_list . (17)

LBRACE shift 80

LBRACK shift 17

RETURN shift 82

IF shift 83

MAINFUNC shift 18

PRINT shift 84

NUM\_LIT shift 19

STRING\_LIT shift 20

BOOL\_LIT shift 21

ID shift 22

RBRACE reduce 17

expr goto 85

stmt goto 86

state 88

stmt : LBRACE rev\_stmt\_list . RBRACE (23)

RBRACE shift 94

. error

state 89

expr\_opt : expr . (25)

expr : expr . PLUS expr (34)

expr : expr . MINUS expr (35)

expr : expr . TIMES expr (36)

expr : expr . DIVIDE expr (37)

expr : expr . EQ expr (38)

expr : expr . NEQ expr (39)

PLUS shift 38

MINUS shift 39

TIMES shift 40

DIVIDE shift 41

EQ shift 42

NEQ shift 43

SEMI reduce 25

state 90

stmt : RETURN expr\_opt . SEMI (19)

SEMI shift 95

. error

state 91

stmt : IF LPAREN . expr RPAREN stmt ELSE stmt (21)

stmt : IF LPAREN . expr RPAREN stmt (22)

LBRACE shift 16

LBRACK shift 17

MAINFUNC shift 18

NUM\_LIT shift 19

STRING\_LIT shift 20

BOOL\_LIT shift 21

ID shift 22

. error

expr goto 96

state 92

stmt : PRINT expr . SEMI (20)

expr : expr . PLUS expr (34)

expr : expr . MINUS expr (35)

expr : expr . TIMES expr (36)

expr : expr . DIVIDE expr (37)

expr : expr . EQ expr (38)

expr : expr . NEQ expr (39)



SEMI shift 97  
PLUS shift 38  
MINUS shift 39  
TIMES shift 40  
DIVIDE shift 41  
EQ shift 42  
NEQ shift 43  
. error

state 93

stmt : expr SEMI . (18)

. reduce 18

state 94

stmt : LBRACE rev\_stmt\_list RBRACE . (23)

. reduce 23

state 95

stmt : RETURN expr\_opt SEMI . (19)

. reduce 19

state 96

stmt : IF LPAREN expr . RPAREN stmt ELSE stmt (21)

stmt : IF LPAREN expr . RPAREN stmt (22)

expr : expr . PLUS expr (34)

expr : expr . MINUS expr (35)

expr : expr . TIMES expr (36)

expr : expr . DIVIDE expr (37)

expr : expr . EQ expr (38)

expr : expr . NEQ expr (39)

RPAREN shift 98

PLUS shift 38

MINUS shift 39

TIMES shift 40

DIVIDE shift 41

EQ shift 42

NEQ shift 43

. error

state 97

stmt : PRINT expr SEMI . (20)

. reduce 20

state 98

stmt : IF LPAREN expr RPAREN . stmt ELSE stmt (21)

stmt : IF LPAREN expr RPAREN . stmt (22)

LBRACE shift 80

LBRACK shift 17

RETURN shift 82

IF shift 83

MAINFUNC shift 18

PRINT shift 84

NUM\_LIT shift 19

STRING\_LIT shift 20

BOOL\_LIT shift 21

ID shift 22

. error

expr goto 85

stmt goto 99

state 99

stmt : IF LPAREN expr RPAREN stmt . ELSE stmt (21)

stmt : IF LPAREN expr RPAREN stmt . (22)

ELSE shift 100

LBRACE reduce 22  
RBRACE reduce 22  
LBRACK reduce 22  
RETURN reduce 22  
IF reduce 22  
MAINFUNC reduce 22  
PRINT reduce 22  
NUM\_LIT reduce 22  
STRING\_LIT reduce 22  
BOOL\_LIT reduce 22  
ID reduce 22

state 100

stmt : IF LPAREN expr RPAREN stmt ELSE . stmt (21)

LBRACE shift 80  
LBRACK shift 17  
RETURN shift 82  
IF shift 83  
MAINFUNC shift 18  
PRINT shift 84  
NUM\_LIT shift 19  
STRING\_LIT shift 20  
BOOL\_LIT shift 21  
ID shift 22  
. error

```
expr goto 85
stmt goto 101
```

```
state 101
```

```
  stmt : IF LPAREN expr RPAREN stmt ELSE stmt . (21)
```

```
  . reduce 21
```

State 80 contains 1 shift/reduce conflict, 1 reduce/reduce conflict.

55 terminals, 23 nonterminals

61 grammar rules, 102 states

### **preprocessor.c**

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <assert.h>
#include <ctype.h> /* For isspace(). */
#include <stddef.h> /* For size_t. */
```

```
#define MAX_BUFFER 4096
```

```
static void die(const char *message)
{
    perror(message);
    exit(1);
}
```

```
const char *getFileExtension(const char *fileName) {
    const char *dot = strrchr(fileName, '.');
    if(!dot || dot == fileName) return "";
    return dot + 1;
}
```

```
void remove_whitespace(char *str) {
    char *p;
    size_t len = strlen(str);

    for(p = str; *p; p ++, len --) {
        while(isspace(*p)) memmove(p, p+1, len--);
    }
}
```

```
int is_empty(const char *s) {
    while (*s != '\0') {
        if (!isspace(*s))
            return 0;
    }
}
```

```
    s++;  
}  
return 1;  
}
```

```
int main(int argc, char const *argv[])  
{  
    if (argc != 3) {  
        fprintf(stderr, "%s\n", "usage: ./preprocessor <jo program file> <pjo program file>");  
        exit(1);  
    }  
    char *fileName = (char *) argv[1];  
    char *outputFileName = (char *) argv[2];  
  
    // check input file extension  
    if (strcmp("jo", getFileExtension(fileName)) != 0)  
    {  
        die("file extension must be jo");  
    }  
  
    // check output file extension  
    if (strcmp("pjo", getFileExtension(outputFileName)) != 0)  
    {  
        die("output file extension must be pjo");  
    }  
}
```

```
FILE *input;
if ((input = fopen(fileName, "r")) == NULL) {
    die("fopen() failed");
}
```

```
FILE *output;
if ((output = fopen(outputFileName, "w")) == NULL) {
    die("fopen() failed");
}
```

```
char buffer[MAX_BUFFER];
```

```
while (fgets(buffer, sizeof(buffer), input) != NULL) {
```

```
    size_t len = strlen(buffer) - 1;
    if (buffer[len] == '\n') {
        buffer[len] = '\0';
    }
    if (strstr(buffer, "*/") != NULL) {
        fprintf(output, "%s\n", buffer);
    }
    else if (strstr(buffer, "/*") != NULL) {
        fprintf(output, "%s\n", buffer);
    }
    else if (strstr(buffer, "func ") != NULL) {
        fprintf(output, "%s {\n", buffer);
    }
}
```



```

else if (strstr(buffer, "int ") != NULL) {
    fprintf(output, "%s;\n", buffer);
}
/* else if (strstr(buffer, "path ") != NULL) {
    fprintf(output, "%s;\n", buffer);
}
else if (strstr(buffer, "dict ") != NULL) {
    fprintf(output, "%s;\n", buffer);
} */
else if (strstr(buffer, "list ") != NULL) {
    fprintf(output, "%s;\n", buffer);
}
else if (strstr(buffer, "string ") != NULL) {
    fprintf(output, "%s;\n", buffer);
}
else if (strstr(buffer, "bool ") != NULL) {
    fprintf(output, "%s;\n", buffer);
}
else if (strstr(buffer, "for ") != NULL) {
    fprintf(output, "%s {\n", buffer);
}
else if ((strstr(buffer, "if (") != NULL || strstr(buffer, "if(") != NULL) && (strstr(buffer, ")") !=
NULL)) {
    fprintf(output, "%s {\n", buffer);
}
else if ((strstr(buffer, "if (") != NULL || strstr(buffer, "if(") != NULL) && (strstr(buffer, ")") ==
NULL)) {
    fprintf(output, "%s\n", buffer);
}
/*else if (strstr(buffer, ")") != NULL) {
    fprintf(output, "%s {\n", buffer);
}

```

```

*/
else if (strstr(buffer, "else") != NULL) {
    int i;
    int counter = 0;
    for (i = 0; i < strlen(buffer); ++i)
    {
        if (buffer[i] == ' ') {
            fprintf(output, "%c", buffer[i]);
            counter++;
        }
    }
    fprintf(output, "} %s {\n", buffer + counter);
}
else if (strstr(buffer, "end") != NULL) {
    int i;
    for (i = 0; i < strlen(buffer); i++){
        if (buffer[i] == 'e') {
            buffer[i] = '}';
        } else if (buffer[i] == 'n') {
            buffer[i] = '\n';
        } else if (buffer[i] == 'd') {
            buffer[i] = '\0';
        } else {

        }
    }
    fprintf(output, "%s", buffer);
}
else {

```

```
    if (is_empty(buffer)) {
        remove_whitespace(buffer);
        fprintf(output, "\n");
    } else {
        fprintf(output, "%s;\n", buffer);
    }
}
}
fclose(input);
fclose(output);
return 0;
}
```

### **runCommands.sh**

```
#!/bin/bash
ocamllex scanner.mll
ocamlyacc parser.mly
ocamlc -c ast.mli
ocamlc -c parser.mli
ocamlc -c scanner.ml
ocamlc -c parser.ml
ocamlc -c sast.mli
ocamlc -c symboltable.ml
ocamlc -c analyzer.ml
#load Str.cma
ocamlc -c jo.ml
ocamlc -o jo parser.cmo scanner.cmo symboltable.cmo analyzer.cmo Str.cma jo.cmo
```

## runjo.sh

```
#!/bin/sh
```

```
if [ ! -f "./preprocessor" ]; then  
make -f MakePreProc >> make.log  
fi
```

```
if [ ! -f "./jo" ]; then  
make -f Makefile >> make.log  
fi
```

```
# jo executable  
JO="./jo"
```

```
# preprocessor executable  
PRE="./preprocessor"  
TEST_BASE="testfiles"
```

```
# Compare <outfile> <reffile> <difffile>  
# Compares the outfile with reffile. Differences, if any, written to difffile  
Compare() {  
difference=$(diff -b $1 $2)  
echo $difference  
if [ "$difference" != "" ]; then  
echo $difference > $3  
fi
```

```
}
```

```
function compileAndRun() {  
  basename=`echo $1 | sed 's/.*\///'  
  s/.jo//`  
  echo "Running file $basename"  
  reffile=`echo $1 | sed 's/.jo$//`  
  prepfle=$TEST_BASE/$basename'.pjo'  
  #echo $prepfle  
  basedir=`echo $1 | sed 's/^[^]*$//`/'  
  
  #Remove all Old Generated File  
  rm -rf ${reffile}.fdlp  
  rm -rf ${reffile}.cpp  
  rm -rf ${reffile}.out  
  rm -rf ${reffile}.o  
  rm -rf ${reffile}.output  
  
  # gets the path of the test output file  
  testoutput=`echo ${basedir}test_outputs/$basename.c.out`  
  
  echo "Preprocessing '$1'"  
  $PRE $1 $prepfle && echo "Preprocessor for $1 succeeded"  
  
  echo "Compiling '$prepfle'"  
  if [ ! -f $prepfle ]; then
```

```
echo "$preprofile does not exist"
return
fi
# converting from JO to C++
$JO $preprofile > "${reffile}.cpp" && echo "Ocaml to C++ of $1 succeeded"

if [ ! -s ${reffile}.cpp ]; then
echo "Error in Compilation of ${reffile}"
return
fi ;

# compiling the C++ file
if [ -f "${reffile}.cpp" ]; then
    make inputfile=$basename -f MakeFileCPP
else
    echo "Compiling $1 failed"
    return
fi

# running the binary
if [ -f "${reffile}.out" ]; then
    eval ${reffile}.out
    echo
fi

}
```

```
files=$TEST_BASE/*.jo
```

```
if [ -f $1 ]; then  
compileAndRun $1  
else  
echo "$1 doesnt exist"  
fi
```

### **sast.mli**

```
type op_t = Add | Sub | Mult | Div | Equal | Neq | Less | Leq | Greater | Geq | And | Or | Concat |  
Minus | Mod | In | NotIn
```

```
type data_type_t = StrType | IntType | BoolType | JsonType | ListType | NoType
```

```
type sep_t = Comma
```

```
type colon_t = Colon
```

```
type list_expr_t =  
  ListItemInt of float  
  | ListItemStr of string
```

```
type list_element_t =  
  LitIntElem of float
```

```
| LitStrElem of string
| LitListOfList of items_t
| LitJsonOfList of json_items_t
| LitNullElem of string
| LitBoolElem of string
and items_t =
  Item of list_element_t
| Seq of list_element_t * sep_t * items_t
| Noitem
and json_key_type_t =
  LitStrJsonKey of string
and json_item_value_t =
  LitIntJsonValue of float
| LitStrJsonValue of string
| LitJsonOfJson of json_items_t
| LitListOfJson of items_t
| LitBoolJsonValue of string
| LitNullJsonValue of string
and json_item_t =
  JsonValPair of json_key_type_t * colon_t * json_item_value_t
and json_items_t =
  JsonItem of json_item_t
| JsonSeq of json_item_t * sep_t * json_items_t
| NoJsonItem

type expr_t =
  LitInt of float
| LitStr of string
| LitJson of json_items_t
| LitList of items_t
```



- | LitBool of string
- | LitNull of string
- | Id of string
- | Not of expr\_t
- | Binop of expr\_t \* op\_t \* expr\_t
- | Call of string \* expr\_t list
- | ElemAccess of string \* expr\_t
- | TypeStruct of string
- | AttrList of string
- | DataType of expr\_t \* string
- | Read of string
- | MakeString of expr\_t \* string
- | NoExpr

```
type var_decl_t = {  
  vtype : data_type_t;  
  vname : string;  
  vexpr : expr_t;  
}
```

```
type for_expr_t =  
  Forid of string  
  | AttrList of string
```

```
type loop_var_t =  
  LoopVar of string
```

```
type stmt_t =  
  Expr of expr_t  
| Block of stmt_t list  
| Vdecl of var_decl_t  
| Return of expr_t  
| Print of expr_t * string  
| For of loop_var_t * for_expr_t * stmt_t  
| If of expr_t * stmt_t * stmt_t  
| Write of expr_t * string  
| Assign of string * expr_t  
| ElemAssign of string * expr_t * expr_t
```

```
type func_decl_t = {  
  return : data_type_t;  
  fname : string;  
  formals : var_decl_t list;  
  body : stmt_t list;  
}
```

```
type program_t = var_decl_t list * func_decl_t list
```

### **scanner.ml**

```
{ open Parser }
```

```
let letter = ['a' - 'z' 'A' - 'Z']
```

```
let digit = ['0' - '9']
```

```
let quote = ""
```

rule token = parse

```
[ ' '\r' '\n' '\t' ] { token lexbuf }  
| "/"* "      { comment lexbuf }  
| '('        { LPAREN }  
| ')'        { RPAREN }  
| '{'        { LBRACE }  
| ','        { COMMA }  
| '}'        { RBRACE }  
| '['        { LBRACK }  
| ']'        { RBRACK }  
| '.'        { ACCESS }  
| "++"       { PLUS }  
| "--"       { MINUS }  
| "***"      { TIMES }  
| "/"        { DIVIDE }  
| '='        { ASSIGN }  
| ';'        { SEMI }  
| "=="       { EQ }  
| "!="       { NEQ }  
| '<'        { LT }  
| "<="       { LEQ }  
| '>'        { GT }  
| ">="       { GEQ }  
| "&&"       { AND }  
| "||"       { OR }  
| '!'        { NOT }  
| '-'        { COMMUNUS }  
| '+'        { COMPLUS }
```

```

| ':'      { COLON }
| '#'     { HASH }
| "%%"    { MOD }
| "func"  { FUNC }
| "if"    { IF }
| "else"  { ELSE }
| "for"   { FOR }
| "in"    { IN }
| "end"   { END }
| "not in" { NOTIN }
| "read"  { READ }
| "write" { WRITE }
| "print" { PRINT }
| "type"  { TYPE }
| "typeStruct" { TYPESTRUCT }
| "attrList" { ATTRLIST }
| "makeString" { MAKESTRING }
| "return"  { RETURN }
| "null"    { NULL }
| eof      { EOF }      (* do as microC *)
| ('-?' digit+) | ('-?' digit* '!' digit+) as lit      { NUM_LIT(float_of_string lit) }
| quote [^"]* quote as lit { STRING_LIT(lit) }
| "true" | "false" as lit { BOOL_LIT (lit) }
| letter | (letter | digit | '_' ) * as id { ID(id) }
| _ as char { raise (Failure("illegal character " ^ Char.escaped char)) }

```

and comment = parse

```

"*/"    { token lexbuf }
| _     { comment lexbuf }

```

## **symboltable.ml**

```
open Ast
```

```
module StringMap = Map.Make(String)
```

```
type env = {  
  locals:    string StringMap.t;  
  globals:  string StringMap.t;  
  functions: string list StringMap.t;  
}
```

```
let string_of_vtype = function
```

```
  IntType -> "number"  
| StrType -> "string"  
| BoolType -> "bool"  
| JsonType -> "json"  
| ListType -> "list"  
| NoType -> "notype"
```

```
let find_variable name env =
```

```
  try StringMap.find name env.locals  
  with Not_found -> try StringMap.find name env.globals  
  with Not_found -> ""
```

```
let find_function name env =
```

```
try StringMap.find name env.functions
with Not_found -> []
```

```
let add_local name v_type env =
  if StringMap.mem name env.locals then StringMap.empty
  else StringMap.add name (string_of_vtype v_type) env.locals
```

```
let update_local name v_type env =
  if StringMap.mem name env.locals then StringMap.add name (string_of_vtype v_type)
env.locals, "exist"
  else StringMap.add name (string_of_vtype v_type) env.locals, "added"
```

```
let add_global name v_type env =
  if StringMap.mem name env.globals then StringMap.empty
  else StringMap.add name (string_of_vtype v_type) env.globals
```

```
let update_global name v_type env =
  StringMap.add name (string_of_vtype v_type) env.globals
```

(\* from the ast \*)

```
let get_arg_type = function
  v -> string_of_vtype v.vtype
```

```
let add_function name return_type formals env =
  if StringMap.mem name env.functions then StringMap.empty
  else let f = List.map get_arg_type formals in
```

```
StringMap.add name (string_of_vtype (return_type)::f) env.functions
```

## **test.sh**

```
#!/bin/sh
```

```
if [ ! -f "./preprocessor" ]; then  
make -f MakePreProc >> make.log  
fi
```

```
if [ ! -f "./jo" ]; then  
make -f Makefile >> make.log  
fi
```

```
# jo executable  
JO="./jo"
```

```
# preprocessor executable  
PRE="./preprocessor"  
TEST_BASE="testfiles"
```

```
# Compare <outfile> <reffile> <difffile>  
# Compares the outfile with reffile. Differences, if any, written to difffile  
Compare() {  
difference=$(diff -b $1 $2)  
echo $difference  
if [ "$difference" != "" ]; then
```

```
echo $difference > $3
```

```
fi
```

```
}
```

```
function compileAndRun() {
```

```
  basename=`echo $1 | sed 's/.*\///
```

```
s/.jo//`
```

```
  echo "Running file $basename"
```

```
  reffile=`echo $1 | sed 's/.jo$//`
```

```
  prefile=$TEST_BASE/$basename'.pjo'
```

```
  #echo $prefile
```

```
  basedir="" echo $1 | sed 's/^[^]*$//`/'
```

```
#Remove all Old Generated File
```

```
rm -rf ${reffile}.fdlp
```

```
rm -rf ${reffile}.cpp
```

```
rm -rf ${reffile}.out
```

```
rm -rf ${reffile}.o
```

```
rm -rf ${reffile}.output
```

```
# gets the path of the test output file
```

```
testoutput=`echo ${basedir}test_outputs/$basename.c.out`
```

```
echo "Preprocessing '$1'"
```

```
$PRE $1 $prefile && echo "Preprocessor for $1 succeeded"
```



```
echo "Compiling '$prepfile'"
if [ ! -f $prepfile ]; then
echo "$prepfile does not exist"
return
fi
# converting from JO to C++
$JO $prepfile > "${reffile}.cpp" && echo "Ocaml to C++ of $1 succeeded"

if [ ! -s ${reffile}.cpp ]; then
echo "Error in Compilation of ${reffile}"
return
fi ;

# compiling the C++ file
if [ -f "${reffile}.cpp" ]; then

    make inputfile=$basename -f MakeFileCPP

else
    echo "Compiling $1 failed"
    return
fi

# running the binary
if [ -f "${reffile}.out" ]; then
eval ${reffile}.out >> ${reffile}.output
```

```
Compare ${reffile}.output ${reffile}.exp ${reffile}.error
```

```
rm -rf ${reffile}.o
```

```
echo "ran $1 successfully"
```

```
else
```

```
echo "C++ to binary of ${reffile}.cpp failed"
```

```
fi
```

```
}
```

```
files=$TEST_BASE/*.jo
```

```
if [ -f $1 ]; then
```

```
    compileAndRun $1
```

```
else
```

```
    echo "$1 doesnt exist"
```

```
fi
```

### **testall.sh**

```
#!/bin/sh
```

```
if [ ! -f "./preprocessor" ]; then
```

```
    make -f MakePreProc >> make.log
```

```
fi
```

```
if [ ! -f "./jo" ]; then
    make -f Makefile >> make.log
fi
```

```
# jo executable
JO="./jo"
```

```
# preprocessor executable
PRE="./preprocessor"
TEST_BASE="testfiles"
```

```
# Compare <outfile> <reffile> <difffile>
# Compares the outfile with reffile. Differences, if any, written to difffile
Compare() {
    difference=$(diff -b $1 $2)
    echo $difference
    if [ "$difference" != "" ]; then
        echo $difference > $3
    fi
}
```

```
function compileAndRun() {
    basename=`echo $1 | sed 's/.*\V//
        s/./jo/'`
    echo "Running file $basename"
```

```
reffile=`echo $1 | sed 's/.jo$//'`
preprofile=$TEST_BASE/$basename'.pjo'
#echo $preprofile
basedir="" echo $1 | sed 's/^[^]*$//`/'

# gets the path of the test output file
testoutput=`echo ${basedir}test_outputs/$basename.c.out`

echo "Preprocessing '$1'"
$PRE $1 $preprofile && echo "Preprocessor for $1 succeeded"

echo "Compiling '$preprofile'"
if [ ! -f $preprofile ]; then
    echo "$preprofile does not exist"
    return
fi

# converting from JO to C++
$JO $preprofile > "${reffile}.cpp" && echo "Ocaml to C++ of $1 succeeded"

if [ ! -s ${reffile}.cpp ]; then
    echo "Error in Compilation of ${reffile}"
    return
fi ;

# compiling the C++ file
if [ -f "${reffile}.cpp" ]; then
```

```
    make inputfile=$basename -f MakeFileCPP
else
    echo "Compiling $1 failed"
    return
fi

# running the binary
if [ -f "${reffile}.out" ]; then
    eval "${reffile}.out >> ${reffile}.output"
    Compare ${reffile}.output ${reffile}.exp ${reffile}.error

    rm -rf ${reffile}.fdlp
    rm -rf ${reffile}.pjo
    rm -rf ${reffile}.cpp
    rm -rf ${reffile}.out
    rm -rf ${reffile}.o
    rm -rf ${reffile}.output
    echo "ran $1 successfully"
else
    echo "C++ to binary of ${reffile}.cpp failed"
fi

}

files=$TEST_BASE/* .jo
```

```
for file in $files
do
  compileAndRun $file
done
```

### **testreadinput.txt**

```
{"name":"harsha", "courses":["PLT","OS"]}
```

### **dataproc3.jo**

```
func main()
```

```
  input = read("testfiles/yelpPlaces.json")
```

```
  dataList = input["data"]
```

```
  c = "{ String : String, String : List, String : String, String : String, String : Number, String :  
Number, String : String, String : List, String : Boolean, String : String, String : Number, String  
: List, String : Number, String : String, String : String, String : String }"
```

```
  allMatched = true
```

```
  for jsonEl in dataList
```

```
    if (c != jsonEl.typeStruct())
```

```
      print("not matched")
```

```
      allMatched = false
```

```
    end
```

```
  end
```

```
if (allMatched)
  print ("\n\n\n\nAll Json elements from file have the correct structure!!!\n\n\n\n")
end
```

```
print ("Adding a Json that has a different one\n\n\n\n")
```

```
print ("Changing a list inside that had just a single string to become just that string, without the list\n\n\n\n")
```

```
d = #{"business_id": "lu-oeVzv8ZgP18NIB0UMqg", "full_address": "3320 S Hill StSouth East
LALos Angeles, CA 90007", "schools": ["University of Southern California"], "open": true,
"categories": ["Medical Centers", "Health and Medical"], "photo_url": "http://s3-
media1.ak.yelpcdn.com/bphoto/SdUWxREuWuPvvot6faxfXg/ms.jpg", "city": "Los Angeles",
"review_count": 2, "name": "Southern California Medical Group", "neighborhoods": "South East
LA", "url": "http://www.yelp.com/biz/southern-california-medical-group-los-angeles", "longitude":
-118.274281, "state": "CA", "stars": 3.5, "latitude": 34.019710000000003, "type": "business"}#
```

```
if (d.typeStruct() == c)
  print ("Matched the wrong one too\n")
else
  print ("Found the wrong one!!\n")
end
```

```
end
```

## **dataproc2.jo**

```
func main()
  input = read("testfiles/yelpPlaces.json")
```

```
datalist = input["data"]
```

```
c = #{"business_id": "lu-oeVzv8ZgP18NIB0UMqg", "full_address": "3320 S Hill StSouth East  
LALos Angeles, CA 90007", "schools": ["University of Southern California"], "open": true,  
"categories": ["Medical Centers", "Health and Medical"], "photo_url": "http://s3-  
media1.ak.yelpcdn.com/bphoto/SdUWxREuWuPvvot6faxfXg/ms.jpg", "city": "Los Angeles",  
"review_count": 2, "name": "Southern California Medical Group", "neighborhoods": ["South East  
LA"], "url": "http://www.yelp.com/biz/southern-california-medical-group-los-angeles", "longitude":  
-118.274281, "state": "CA", "stars": 3.5, "latitude": 34.019710000000003, "type": "business"}#
```

```
d = #{"business_id": "lu-oeVzv8ZgP18NIB0UMqg", "full_address": "3320 S Hill StSouth East  
LALos Angeles, CA 90007", "schools": ["University of Southern California"], "open": true,  
"categories": ["Medical Centers"], "photo_url": "http://s3-  
media1.ak.yelpcdn.com/bphoto/SdUWxREuWuPvvot6faxfXg/ms.jpg", "city": "Los Angeles",  
"review_count": 2, "name": "Southern California Medical Group", "neighborhoods": ["South East  
LA"], "url": "http://www.yelp.com/biz/southern-california-medical-group-los-angeles", "longitude":  
-118.274281, "state": "CA", "stars": 3.5, "latitude": 34.019710000000003, "type": "business"}#
```

```
if(c in datalist)
```

```
  print("found\n")
```

```
end
```

```
if(d in datalist)
```

```
  print("found Second")
```

```
else
```

```
  print("Second not found")
```

```
end
```

```
end
```



### **dataproc.jo**

```
func processYelp(category, data)
  for entry in data
    categoryList = entry["categories"]
    if (category in categoryList)
      write (entry, "testfiles/processed.txt")
    end
  end
  return null
end
```

```
func main()
  input = read("testfiles/yelpPlaces.json")
  datalist = input["data"]
  processYelp("Pubs", datalist)
end
```

### **TypeFunction.jo**

```
/* Type Function simple types */
```

```
func main ()
```

```
  a = true
```

```
  b = "asdfasdf"
```

```
  c = false
```

```
d = 345534
```

```
e = []
```

```
f = #{}#
```

```
print(type(a))
```

```
print(type(b))
```

```
print(type(c))
```

```
print(type(d))
```

```
print(type(e))
```

```
print(type(f))
```

```
end
```

## **TypeFunction.exp**

```
BoolStringBoolNumberListJson
```

## **TestTypeStruct2.jo**

```
/* Nested Json TypeStruct test */
```

```
func main ()
```

```
  a = #{ "glossary": { "title": "example glossary", "GlossDiv": { "title": "S", "GlossList":  
{"GlossEntry": { "ID": "SGML", "SortAs": "SGML", "GlossTerm": "Standard Generalized Markup  
Language", "Acronym": "SGML", "Abbrev": "ISO 8879:1986", "GlossSee": "markup" } } } }  
}#
```

```
  b = a.typeStruct()
```

```
  print (b)
```

end

### **TestTypeStruct2.exp**

```
{ String : { String : { String : { String : { String : String, String : String, String : String, String : String, String : String, String : String }}, String : String }, String : String } }
```

### **TestTypeStruct.jo**

```
/* Test How TypeStruct function work */
```

```
func main ()
```

```
  a = #{"name":"harsha", "number":12223, "list":[], "json":{"name":"harsha"}, "boolean":true}#
```

```
  b = a.typeStruct()
```

```
  print (b)
```

```
end
```

### **TestTypeStruct.exp**

```
{ String : Boolean, String : { String : String }, String : List, String : String, String : Number }
```

### **TestStringConcat.jo**

```
/* Test String Concatenation */
```

```
func main ()
```

```
  a = "hi"
```

```
b = "bye"  
print(a ++ b)  
end
```

### **TestStringConcat.exp**

```
hibye
```

### **TestNotOperator.jo**

```
func main ()  
  a = true  
  b = false  
  if( !(a==b))  
    print (a)  
  end  
  if(!a)  
    print(b)  
  end  
  if(!a ==b)  
    print(a)  
  end  
end
```

### **TestNotOperator.exp**

```
truetrue
```

### **TestNestedIfCondition.jo**

```
/* Test If Condition */
```

```
func main ()

if ( 5 == 5 )
  print ("true")

  if(true)
    print("linside 1st nested loop")

    if("hi" == "hi")
      print("Inside 2nd Nested If")
    end
  end

end
else
  print ("false")
end

end
```

### **TestNestedIfcondition.exp**

truelinside 1st nested loopInside 2nd Nested If

### **TestNestedForCondition.jo**

/\* Test Nested For Condition \*/

```
func main ()
  a = #{"a":[1,2,3], "b":[4,5,6], "c":[7,8,9]}#
  d = a.attrList()

  for i in d
    c = a[i]
    for j in c
      print (j)
    end
  end
end
```

### **TestNestedForCondition.exp**

123456789

### **TestIfIn.jo**

```
func main ()
  a = "hi"
  b = "bye"
  c = ["bbye", "hello", "bye"]
  if(a in c)
    print(a)
  end
  if(b in c)
    print(b)
  else
    print("Not found")
  end
end
```

```
if(1==2)
  print("equal")
end
if(1!=2)
  print("not equal")
else
  print("equal")
end
end
```

### **TestIfIn.exp**

```
byenot equal
```

### **TestIfCondition.jo**

```
/* Test If Condition */
```

```
func main ()
```

```
  if ( 5 == 5 )
    print ("true")
  else
    print ("false")
  end
```

```
end
```

### **TestIfCondition.exp**

```
true
```

### **TestForCondition.jo**

```
/* Test For Condition */
```

```
func main ()
```

```
  a = [2,3,4]
```

```
  for i in a
```

```
    print (i)
```

```
  end
```

```
end
```

### **TestForCondition.exp**

```
234
```

### **TestAttrList.jo**

```
/* Test How AttrList function work */
```

```
func main ()
```

```
  a = #{"name":"harsha", "courses":["PLT", "OS"]}#
```

```
  b = a.attrList()
```



```
print (b)
end
```

### **TestAttrList.exp**

```
[courses,name]
```

### **ScopeCheck2.jo**

```
/* Check variable Scope */
```

```
func checkScope(x)
  x = "Inside Check Scope"
  print (x)
  return null
end
```

```
func main ()
  x = "Inside Main Function"
  print (x)
  checkScope(x)
end
```

### **ScopeCheck2.exp**

```
Inside Main FunctionInside Check Scope
```

### **ScopeCheck.jo**

```
/* Check variable Scope */
```

```
x = "Global Declaration"
```

```
func main ()  
  print (x)  
  x = "Inside Main Function"  
  print (x)  
end
```

### **ScopeCheck.exp**

```
Global Declaration  
Inside Main Function
```

### **ReadFunction.jo**

```
/* Test Read function */
```

```
func main ()  
  a = read ("testfiles/testreadinput.txt")  
  print (a)  
end
```

### **ReadFunction.exp**

```
{  
  "courses": [  
    "PLT",  
    "OS"  
  ],  
}
```

```
"name":"harsha"  
}
```

### **PrintHelloWorld.jo**

```
/* Declare a Global Variable and print in Main function */
```

```
a = "Hello World"
```

```
func main ()  
  print (a)  
end
```

### **PrintHelloWorld.exp**

```
Hello World
```

### **PrettyPrintJson.jo**

```
/* Test Json Pretty Print */
```

```
a = #{"name":"harsha","courses":[1,"PLT",true]}#
```

```
func main ()  
  print (a)  
end
```

## PrettyPrintJson.exp

```
{
  "courses":[
    1,
    "PLT",
    true
  ],
  "name":"harsha"
}
```

## NotInOperator

```
/* Not In Operator */
```

```
func main ()
```

```
  a = [2243,1232,3454]
```

```
  b = 1232 not in a
```

```
  c = [{"a","b"}, 45, "ccc"]
```

```
  print(b)
```

```
  print("ccc" not in c)
```

```
  print("ab" not in c)
```

```
  print(["a","c"] not in c)
```

```
end
```

## NotInOperator.exp

```
falsefalsetruetrue
```

## NegativeWrongFunctionCall.jo

```
/* Invalid Function Call */
```

```
func testMeth ()  
  print ("testMethod")  
  return null  
end
```

```
func main ()  
  invalidFun()  
end
```

### **NegativeWrongDeclaration.jo**

```
/* Wrong declaration */
```

```
func main ()  
  a = "highlight"  
  b = treu  
  
  print(b)  
end
```

### **NegativeWriteFunction.jo**

```
/* Test Write function */
```

```
func main ()
```

```
a = "Text to write into file"
write (a, /Users/harsha/testfilewriting.txt)
end
```

### **NegativeVariableUse2.jo**

```
/* Variable not declared */
```

```
func testMeth (a)
  c = b
  print (c)
  return null
end
```

```
func main ()
  a = "highlight"
  testMeth(a)
end
```

### **NegativeVariableUse.jo**

```
/* Variable not declared */
```

```
func testMeth ()
  print ("testMethod")
  return null
end
```

```
func main ()  
  print (a)  
  a = "highlight"  
end
```

### **NegativeStringConcatTest.jo**

```
/* String Concatenation with number */
```

```
func main ()  
  a = "hi"  
  b = 121  
  print(a ++ b)  
end
```

### **NegativeReadDeclaration.jo**

```
/* Test Read function */
```

```
func main ()  
  b = 23324  
  a = read (b)  
  print (a)  
end
```

### **NegativeOpTestCase.jo**

```
/* Mathematical Operator on List */
```

```
func main ()  
a = [5]  
b = 6  
  
print (a == b)  
  
end
```

### **NegativeMathOpTestcases.jo**

*/\* Mathematical Operator on List \*/*

```
func main ()  
a = [5]  
b = 6  
print (a ++ b)  
end
```

### **NegativeInvalidList.jo**

*/\* Invalid List Declaration \*/*

```
func main ()  
a = ["name",asdfdf]  
print (a)  
end
```



### **NegativeInvalidJson.jo**

*/\* Invalid Json Declaration \*/*

```
func main ()  
  a = #{"name":asdfdf}  
end
```

### **NegativeFunctionNoReturn.jo**

*/\* Function call with different arguments \*/*

```
func testMeth (a,b)  
  print (a)  
  return null  
end
```

```
func main ()  
  a = "highlight"  
  testMeth(a)  
end
```

### **NegativeFunctionCall.jo**

*/\* Function call with different arguments \*/*

```
func testMeth (a)  
  print (a)  
end
```

```
func main ()
  a = "highlight"
  testMeth(a)
end
```

### **NegativeForusage.jo**

```
/* Test If Condition */
```

```
func main ()
  a = "Hello"

  for i in a
    print (i)
  end

end
```

### **NegativeElementAccess.jo**

```
a = #{"a":"b"}#
```

```
func main()
  print(a[true])
end
```

### **NegationOperator3.jo**

```
/* Negation Operator List minus Json, Number */
```

```
func main ()  
    list1 = ["able", "barista", {"carrie":"Its a name"}]  
    list2 = [1121, 2323]  
  
    json1 = #{"carrie":"Its a name"}#  
    num = 2323  
  
    print (list1 - json1)  
    print (list2 - num)  
end
```

### **AllTypeCreation.jo**

```
/* Creation of all Types */
```

```
func main ()  
    a = "Hello World all variable types created"  
    b = 10  
    c = #{"name": "harsha"}#  
    d = [4,6,7]  
    e = true  
    f = null  
    print (a)  
end
```

## **AllTypeCreation.exp**

Hello World all variable types created

## **CommentInsideFunction.jo**

```
/* Testing Comment Inside a Function */
```

```
a = 45
```

```
b = 81
```

```
func findGCD(a,b)
```

```
    if(b == 0)
```

```
        return a
```

```
    end
```

```
    /*
```

```
    This is a GCD Program
```

```
    */
```

```
    return findGCD(b, a%%b)
```

```
end
```

```
func main ()
```

```
    print( findGCD(a,b) )
```

```
end
```

## **CommentInsideFunction.exp**

9

## **ComparisonOperator.jo**

```
/* Comparison equal and not equal */
```

```
func main ()
```

```
  a = 5
```

```
  b = 6
```

```
  c = "f"
```

```
  d = "l"
```

```
  if(a==b)
```

```
    print ("hi")
```

```
  end
```

```
  if (c!=d)
```

```
    print ("hello")
```

```
  end
```

```
end
```

**ComparisonOperator.exp**

hello

**ComparisonOperator2.jo**

```
/* Comparison Operator with booleans */
```

```
func main ()
```

```
a = true
b = false

if(a && b)
  print ("hi")
end

if (a || b)
  print ("hello")
end

if (a || b && true)
  print ("hello1")
end

end
```

end

### **ComparisonOperator2.exp**

hellohello1

### **ComparisonOperator3.jo**

```
/* Comparison Operator with Not */
```

```
func main ()
  a = true
  b = false

  c = 5
  d = 6
```

```
if(!b)
    print ("hi")
end

if (a && !b)
    print ("hello")
end

if (!(c==d))
    print ("hello1")
end
```

end

### **ComparisonOperator3.exp**

hihellohello1

### **ConcatenationOperator1.jo**

```
/* Concatenation on Numbers and String */
```

```
func main ()
    a = 59
    b = 600
    c = "Hello"
    d = "world"
    e = a+b
    f = c+d
```

```
    print (b + a)
    print (c + d)
    print (f + e)
end
```

### **ConcatenationOperator1.exp**

```
[600,59][Hello,world][Hello,world,[59,600]]
```

### **ConcatenationOperator2.jo**

```
/* Concatenation of Json */
```

```
func main ()
    a = #{"name": "harsha", "innerJson": {"sub": "PLT", "mark": [5, 6, 7]}}#
    b = #{"name": "arpit"}#

    print (a + b)
    print (a + 5)
    print (a + "json")
end
```

ConcatenationOperator2.exp

```
 [{"innerJson": {"mark": [5, 6, 7], "sub": "PLT"}, "name": "harsha"}, {"name": "arpit"}] [{"innerJson": {"mark": [5, 6, 7], "sub": "PLT"}, "name": "harsha"}, 5] [{"innerJson": {"mark": [5, 6, 7], "sub": "PLT"}, "name": "harsha"}, json]
```

### **ConcatenationOperator3.jo**

```
/* Concatenation of List */
```

```
func main ()
```



```
a = [[1,2,"a"]]
b = ["hi", "hello"]
c = 5
d = "hey"

print (a + b)
print (a + c +d)
end
```

### **ConcatenationOperator3.exp**

```
[[1,2,a],[hi,hello]][[1,2,a],[hi,hello],5,hey]
```

### **CustomFunction1.jo**

```
/* Testing Function call from Main and printing inside callee */
```

```
a = "Hello World"
```

```
func test(b)
    print (b)
    return 0
end
```

```
func main ()
    print (a)
    test(a)
end
```

### **CustomFunction1.exp**

Hello WorldHello World

### **CustomFunction2.jo**

```
/* Testing Variable declaration inside function and printing */
```

```
func createVarAndPrint ()  
    a = "This is a String"  
    print (a)  
    return 0  
end
```

```
func main ()  
    createVarAndPrint()  
end
```

### **CustomFunction2.exp**

This is a String

### **CustomFunction3.jo**

```
/* Testing Function Return and print */
```

```
func returnAValue(a)  
    return a ++ 100  
end
```

```
func callAnotherMethod(a)  
    b = returnAValue(a ++ 10)  
    print (b)
```

```
        return 0
end

func main ()
    callAnotherMethod(10)
end
```

### **CustomFunction3.exp**

120

dataproc.jo

```
func processYelp(category, data)
    for entry in data
        categoryList = entry["categories"]
        if (category in categoryList)
            write (entry, "testfiles/processed.txt")
        end
    end
end
return null
end
```

```
func main()
    input = read("testfiles/yelpPlaces.json")
    datalist = input["data"]
    processYelp("Pubs", datalist)
end
```

**dataproc2.jo**

```

func main()

  input = read("testfiles/yelpPlaces.json")

  datalist = input["data"]

  c = #{"business_id": "Iu-oeVzv8ZgP18NIB0UMqg", "full_address":
"3320 S Hill StSouth East LALos Angeles, CA 90007", "schools":
["University of Southern California"], "open": true, "categories":
["Medical Centers", "Health and Medical"], "photo_url": "http://s3-
medial.ak.yelpcdn.com/bphoto/SdUWxREuWuPvvot6faxfXg/ms.jpg", "city":
"Los Angeles", "review_count": 2, "name": "Southern California Medical
Group", "neighborhoods": ["South East LA"], "url":
"http://www.yelp.com/biz/southern-california-medical-group-los-
angeles", "longitude": -118.274281, "state": "CA", "stars": 3.5,
"latitude": 34.019710000000003, "type": "business"}#

  d = #{"business_id": "Iu-oeVzv8ZgP18NIB0UMqg", "full_address":
"3320 S Hill StSouth East LALos Angeles, CA 90007", "schools":
["University of Southern California"], "open": true, "categories":
["Medical Centers"], "photo_url": "http://s3-
medial.ak.yelpcdn.com/bphoto/SdUWxREuWuPvvot6faxfXg/ms.jpg", "city":
"Los Angeles", "review_count": 2, "name": "Southern California Medical
Group", "neighborhoods": ["South East LA"], "url":
"http://www.yelp.com/biz/southern-california-medical-group-los-
angeles", "longitude": -118.274281, "state": "CA", "stars": 3.5,
"latitude": 34.019710000000003, "type": "business"}#

  if(c in datalist)
    print("found\n")
  end

  if(d in datalist)
    print("found Second")
  else
    print("Second not found")
  end
end

```

end

### **dataproc3.jo**

```
func main()
    input = read("testfiles/yelpPlaces.json")
    dataList = input["data"]

    c = "{ String : String, String : List, String : String, String
: String, String : Number, String : Number, String : String,
String : List, String : Boolean, String : String, String : Number,
String : List, String : Number, String : String, String : String,
String : String }"

    allMatched = true

    for jsonEl in dataList
        if (c != jsonEl.typeStruct())
            print("not matched")
            allMatched = false
        end
    end

    if (allMatched)
        print ("\n\n\n\nAll Json elements from file have the correct
structure!!!\n\n\n\n")
    end

    print ("Adding a Json that has a different one\n\n\n\n")
```

```
print ("Changing a list inside that had just a single string to
become just that string, without the list\n\n\n\n")
```

```
d = #{ "business_id": "Iu-oeVzv8ZgP18NIBOUMqg", "full_address":
"3320 S Hill StSouth East LALos Angeles, CA 90007", "schools":
["University of Southern California"], "open": true, "categories":
["Medical Centers", "Health and Medical"], "photo_url": "http://s3-
medial.ak.yelpcdn.com/bphoto/SdUWxREuWuPvvot6faxfXg/ms.jpg", "city":
"Los Angeles", "review_count": 2, "name": "Southern California Medical
Group", "neighborhoods": "South East LA", "url":
"http://www.yelp.com/biz/southern-california-medical-group-los-
angeles", "longitude": -118.274281, "state": "CA", "stars": 3.5,
"latitude": 34.019710000000003, "type": "business"}#
```

```
if (d.typeStruct() == c)
    print ("Matched the wrong one too\n")
else
    print ("Found the wrong one!!\n")
end
```

```
end
```

## **DynamicTyping.jo**

```
/* Change type of variable at run-time */
```

```
func test(l)
    print(l - "d")
    return null
end
```

```
func main()
```

```
e = ["a", "c", "d"]
test(e)
e = 5
print(e)
end
```

### DynamicTyping.exp

[a,c]5

### ElementAccess.jo

```
/* Element Access in Json */

func main ()
  b = #{"name": "harsha", "innerJson": {"sub": "PLT", "mark": [5, 6, 7]}}#
  d = b["innerJson"]
  print (d)
end
```

### ElementAccess.exp

```
{
  "mark":[
    5,
    6,
    7
  ],
  "sub":"PLT"
```

```
}
```

### ElementAccess2.jo

```
/* Element Access in List */
```

```
func main ()
```

```
    a = [1, "2", [4, ["gw", 3, 4], "name"], {"name": "harsha",  
"innerJson": {"sub": "PLT", "mark": [5, 6, 7]}, "number": 2324} ]
```

```
    print (a[3])
```

```
end
```

### ElementAccess2.exp

```
{
```

```
    "innerJson":{
```

```
        "mark":[
```

```
            5,
```

```
            6,
```

```
            7
```

```
        ],
```

```
        "sub": "PLT"
```

```
    },
```

```
    "name": "harsha",
```

```
    "number": 2324
```

```
}
```

### ElementAssign.jo



```
/* Element Assignment in a Json */
```

```
a = #{"a":"b"}#
```

```
func main()
```

```
  a["c"] = ["d"]
```

```
  a["school"] = #{"names":["Columbia","Caltech"]}#
```

```
  print(a["c"])
```

```
  print(a)
```

```
end
```

**ElementAssign.exp**

```
[d]{
```

```
  "a":"b",
```

```
  "c":[
```

```
    "d"
```

```
  ],
```

```
  "school":{
```

```
    "names":[
```

```
      "Columbia",
```

```
      "Caltech"
```

```
    ]
```

```
  }
```

```
}
```

**ElementAssign2.jo**

```
/* Element Assignment in a List */
```

```
func main()
```

```
    a = [1,2,3,4]
```

```
    a[0] = "High"
```

```
    a[1] = "School"
```

```
    a[2] = #{"name":"Veer"}#
```

```
    print(a[0])
```

```
    print(a)
```

```
end
```

**ElementAssign2.exp**

```
High[High,School,{"name":"Veer"},4]
```

**FileWriteFunction.jo**

```
/* Test Write function */
```

```
func main ()
```

```
    a = "Text to write into file"
```

```
    write (a, "/Users/harsha/testfilewriting.txt")
```

```
end
```

**ForStatement.jo**

```
/* Declare a Global Variable and print in Main function */
```

```
a = [1, 2 ,3 ]
```

```
func main ()  
    for b in a  
        print (b)  
    end  
end
```

### **ForStatement.exp**

123

### **GCD.jo**

```
/* Test GCD function */
```

```
a = 45
```

```
b = 81
```

```
func findGCD(a,b)  
    if(b == 0)  
        return a  
    end  
    return findGCD(b, a%%b)  
end
```

```
func main ()  
    print( findGCD(a,b) )  
end
```

## GCD.exp

9

## InOperator.jo

```
/* In Operator */

func main ()

    a = [2243,1232,3454]
    b = 1232 in a

    c = [{"a","b"}, 45, "ccc"]

    print(b)
    print("ccc" in c)
    print("ab" in c)
    print(["a","b"] in c)
end
```

## InOperator.exp

Truetruetruetrue

## MakeStringFunction.jo

```
/* Use makeString and Concatente Strings */
```

```
func main ()
```

```
a = 2243
b = ["abc", "bac"]
c = #{"name": "harsha"}#

print(makeString(a))
print(makeString(b))
print(makeString(c))
end
```

### **MakeStringFunction.exp**

```
2243[abc,bac>{"name": "harsha"}
```

### **MakeStringFunction2.jo**

```
/* MakeString Function of Number, List and Json */
```

```
func main ()

a = 2243
b = ["abc", "bac"]
c = #{"name": "harsha"}#

d = makeString(a) ++ makeString(b)

print(d)

print(makeString(d) ++ makeString(c))
```

end

### **MakeStringFunction2.exp**

```
2243[abc,bac]2243[abc,bac>{"name":"harsha"}
```

### **MathematicalOperator.jo**

```
/* Mathematical Operator */
```

```
func main ()  
    a = 5  
    b = 6  
    print (a ++ b)  
    print (b -- a)  
    print (a ** b)  
    print (a // 5)  
    print (a > b)  
    print (a >= b)  
    print (a < b)  
    print (a <= b)  
    print (b %% a)
```

end

### **MathematicalOperator.exp**

```
111301falsefalse>true>true1
```

### **MathematicalOperator2.jo**

```
/* Mathematical Operator */
```

```
func main ()  
    print (5 ++ 6 ** 3 ** 2 // 3 -- 2)  
    print (9 %% 3 ** 2)  
end
```

### **MathematicalOperator2.exp**

153

### **Merge.jo**

```
func Merge (a,b)  
    c = #{ }#  
    d = a.attrList()  
    e = b.attrList()  
    for attr in d  
        if ( attr in e)  
            if (type(a[attr]) == "Json" && type(b[attr]) == "Json")  
  
                c[attr] = Merge(a[attr],b[attr])  
            else  
                if (type(a[attr]) == "List" && type(b[attr]) == "List")  
                    c[attr] = a[attr] ++ b[attr]  
                else  
                    c[attr] = a[attr] + b[attr]  
                end  
            end  
        end  
    else  
        c[attr] = a[attr]  
    end  
end  
for attr in e
```

```
        if ( attr not in d )
            c[attr] = b[attr]
        end
    end
end
return c
end
```

```
func MergeUtil (a,b)
    if ( type(a) != "Json" || type(b) != "Json")
        print("Both the arguments must be JSON")
        return null
    end
    return Merge(a,b)
end
```

```
func main ()

a = #{"name": "harsha", "innerJson": {"sub": "PLT", "mark": [5, 6, 7]}}#
b = #{"name": "arpit", "innerJson": {"sub": "OS", "mark": [7, 8, 9]}}#
c = MergeUtil (a,b)
print (c)
end
```

### **Merge.exp**

```
{
    "innerJson":{
        "mark":[
            5,
            6,
            7,
```



```
7,  
8,  
9  
],  
"sub":[  
"PLT",  
"OS"  
]  
},  
"name":[  
"harsha",  
"arpit"  
]  
}
```

### **NegationOperator1.jo**

```
/* Negation Operator Json minus string */
```

```
func main ()  
    json = #{"name": {"first":"chase", "last": "larson"}, "marks":  
[2,3]}#  
    print (json - "name")  
end
```

NegationOperator1.exp

```
{  
"marks":[  
2,  
3
```

```
]
}
```

### **NegationOperator2.jo**

```
/* Negation Operator List minus List, String */
```

```
func main ()
    list1 = ["able", "barista", "carrie"]
    list2 = ["barista", "carrie"]
    str = "barista"
    print (list1 - list2)
    print (list2 - str)
end
```

### **NegationOperator2.exp**

```
[able,barista,carrie][carrie]
```

### **NegationOperator3.exp**

```
[able,barista][1121]
```