QL

COMS W4115: Fall 2015

Final Report

Anshul Gupta, Evan Tarrh, Gary Lin,
Matt Piccolella, Mayank Mahajan

Table of Contents

Section 1: Introduction

JavaScript Object Notation (JSON) is an open-standard format that uses human-readable format to capture attribute-value pairs. JSON has gained prominence replacing XML encoded-data in browser-server communication, particularly with the explosion of RESTful APIs and AJAX requests that often make use of JSON. While domain-specific languages like SQL and PostgreSQL work with relational databases, languages like AWK specialize in processing datatables, especially tab-separated files. We noticed a need for a language designed to interact with JSON data, to quickly search through JSON structures and run meaningful queries on the JSON data, all the while using a syntax that aligned much more closely with the actual structure of the data we were using.

Section 2: Language Tutorial

You can refer to the LRM to understand the syntax and the rules of our language. But to get started use a JSON file that looks something like this:

```
{
        "count" : 5,
        "owner" : "Matt",
        "number" : 5.4,
        "friends" : [
          {
            "name" : "Anshul",
            "age" : 12
          },
          {
            "name" : "Evan",
            "age" : 54
          },
          {
            "name" : "Gary",
            "age" : 21
          },
          {
            "name" : "Mayank",
            "age" : 32
          }
        ],
        "inner" : {
          "name" : "Hello!"
        }
       }
```

and you can begin to work through it with a simple code that looks like:

```
json test = json("sample.json")

where (elem["age"] > 20) as elem {
 string s = elem["name"]
 print(s)
} in test["friends"]
```

Here’s a quick walk-through of the functionality offered by this simple program:

1. You import the sample.json file that is shown above using the `json(“sample.json”)` command
2. The where loop is one of the most important features of our language. It accepts an array in the JSON structure and iterates over each element of the array. The where clause can be used to filter out items based on criteria for each of the JSON elements you’ll be passing stepping through.
3. In this case the value of the “friends” attribute in the JSON file is an array so the where .. in .. syntax accepts that array
4. Each item in the array can be referred with the variable that occurs after the `as` keyword, in this case that variable is `elem`
5. This program checks if the `age` attribute in the current item in the array is greater than 20
6. For each value of `elem` that matches the condition defined inside the `where()` syntax, the program extracts the name of that `elem` and prints it out to standard output.

The `where` loop shown in this example is a glimpse into how you can use our language, QL, to meaningfully interact with JSON data.

Section 3: Language Reference Manual

3.1 Introduction

JavaScript Object Notation (JSON) is an open-standard format that uses human-readable format to capture attribute-value pairs. JSON has gained prominence replacing XML encoded-data in browser-server communication, particularly with the explosion of RESTful APIs and AJAX requests that often make use of JSON.

While domain-specific languages like SQL and PostgreSQL work with relational databases, languages like AWK specialize in processing datatables, especially tab-separated files. We noticed a need for a language designed to interact with JSON data, to quickly search through JSON structures and run meaningful queries on the JSON data, all the while using a syntax that aligned much more closely with the actual structure of the data we were using.

3.2 Lexical Conventions

3.2.1 Identifiers

Identifiers are combinations of letters and numbers. They must start with a lowercase letter, and can be any combination of lowercase letters, uppercase letters, and numbers. Lowercase letters and uppercase letters are seen as being distinct. Identifiers can refer to three things in our language: variables, functions, and function arguments.

3.2.2 Keywords

The following words are defined as keywords and are reserved for the use of the language; thus, they cannot be used as identifiers to name either a variable, a function, or a function argument:

```
int, float, bool, string, json, array, where, in, as, for, while, return, function,
True, False, if, else, void, not
```

3.2.3 Comments

We reserve the symbol `#~~` to introduce a comment and the symbol `~~#` to close a comment. Comments cannot be nested, and they do not occur within string literals. A comment looks as follows:

```
#~~ This is a comment. ~~#
```

3.2.4 Literals

Our language supports several different types of literals.

3.2.4.1 int literals

A string of numeric digits of arbitrary size that does not contain a decimal point. Integers can have an optional ‘-’ at the beginning of the string of numbers to indicate a negative number.

3.2.4.2 float literals

QL is following Brian Kernighan and Dennis Ritchie's explanation in *The C Programming Language*: "A floating constant consists of an integer part, a decimal part, a fraction part, an e, and an optionally signed integer exponent. The integer and fraction parts both consist of a sequence of digits. Either the integer part, or the fraction part (not both) may be missing; either the decimal point or the e and the exponent (not both) may be missing." Floats can also contain an optional ‘-’ at the beginning of the float to indicate a negative value.

3.2.4.3 bool literals

Booleans can take on one of two values: `True` or `False`. Booleans in QL are capitalized.

3.2.4.4 string literals

A sequence of ASCII characters surrounded by double quotation marks on both sides.

3.3 Data Types

3.3.1 Primitive Types

The primitive types in QL are statically typed; in other words, the type of a variable is known at compile time. This occurs when the right side of the assignment is a literal of a data type. The primitive types can be declared and then initialized later (their value is null in the interim) or declared and initialized in-line.

3.3.1.1 Integers (int)

Integers are signed, 8-byte literals denoting a number as a sequence of digits e.g. `5,6,-1,0`.

3.3.1.2 Floating Point Numbers (float)

Floats are signed, 8-byte single-precision floating point numbers e.g. `-3.14`, `4e10`, `.1`, `2.`.

3.3.1.3 Boolean (bool)

Booleans are defined by the `True` and `False` keywords. Only boolean types can be used in logical expressions e.g. `True,False`.

3.3.1.4 String (string)

Since our language doesn't contain characters, strings are the only way of expressing zero or more characters in the language. Each string is enclosed by two quotation marks e.g. `"e"`, `"Hello, world!"`.

3.3.2 Non-Primitive Types

All non-primitive data types are passed by a reference in memory. They can each be declared and initialized later (their value is null in the interim) or declared and initialized in line.

3.3.2.1 Arrays (array)

Arrays represent multiple instances of one of the primitive data types represented as contiguous memory. Each array must contain only a single type of primitives; for example, we can have either an array of `int`, an array of `float`, an array of`bool`, and an array of `string`, but no combinations of these types. Note that nested arrays are not allowed in QL. The size of the array is fixed at the time of its creation e.g. `array(10)`. Arrays in QL are statically typed since the type of a variable is known at compile time.

3.3.2.2 JSON (json)

Since the language must search and return results from JSON files, it supports JSONs as a non-primitive type. A `json`object can be created directly from a filename of a valid JSON. For example, one could write: `json a = json("file1.json")`. During runtime, the generated java code will check if the contents of the file make up a valid JSON. This means that JSONs are dynamically typed in QL.

JSONs are statically inferred but checked dynamically in QL.

3.4 Expressions

Expressions in QL can be one of the following types. A statement in our language can be composed of just an expression but it's much more useful to use them in other statements like if-else constructs, loops and assign statements.

3.4.1 Data Type Literal

Expressions can be just a literal, as defined for our language in Section 3.2.4 above. This allows us to directly input a value where needed.

e.g in int a = 5 the 5 on the right hand side of the assignment operator is a Data Type Literal of integer type, used to assign a value to the variable a.

3.4.2 Identifier

Expressions can be just an identifier, as defined for our language in Section 3.2.1 above. This allows us to use variables as values where needed.

e.g in int b = a the a on the right hand side of the assignment operator is an Identifier of integer type, used to assign a value to the variable b.

3.4.3 Bracket Selector

This can be used in two different ways:

- [int index]: accesses value at index of an array variable
    - Return type is the same as the array's type.
    - This square bracket notation can be used to assign a value into a variable. Example of QL Code:

    -
      ```
      array int a = [1;2;3;4]
      int b = a[2]
      ```
    - At the end of this program, b is equal to 3.
- [string key]: accesses value at key of a JSON variable
    - Return type is inferred from the value in JSON. The type can be one of three things: a value (int, float, bool, string), an array, and a json.
    - QL performs static inferring when a declared variable is assigned to a json variable with bracket selectors. The program will check what the type of the left hand side of the assignment is and infer that the json with bracket selectors will resolve to that type.
    - Example of QL Code:

    -
      ```
      json a = json("sample.json")
      int b = a["value"]
      ```

    - It is unclear what a["value"] is so our compiler infers that it will be an integer, since the left hand side of the assignment is an int. This happens in our static semantic check.

This operator can be nested, e.g.: ["data"]["views"]["total"]. It associates from left to right. This means that each additional bracket selector will go one level deeper into the JSON by getting the value of corresponding key.

Below is a program containing different examples of the [] operator. file1.json is the JSON file we will be using in this example.

`file1.json:`

```
 { "data": {
    "views": {
        "total": 80
    },
    "items": {
        "category": "News"
    },
    "users": [
        "Matt",
        "Evan",
        "Gary"
    ]
}
```

`bracket-example.ql:`

```
json file1 = json("file1.json")

#~~ file1["data"]["views"]["total"] statically inferred as an int ~~#
int total = file1["data"]["views"]["total"]
```

`total` equals 80 here.

Here is an example of obtaining a JSON object by using a bracket selector on another JSON object. Say that the json variable b equals this json below.

```
b = {
    "size":10,
    "links": {
        "1": 1,
        "2": 2,
        "3": 3
    }
}
```

Let's use the bracket selector on b. QL allows for commands like `json links = b["links"]`. The links variable would then look as follows:

```
links = {
    "1" : 1,
    "2" : 2,
    "3" : 3
}
```

3.4.4 Binary Operator

3.4.4.1 Multiplication: *

left associative

```
e1 * e2
```

This operation is only valid when both e1 and e2 are integers or floats. When e1 and e2 are ints, this operator will return an int. When e1 and e2 are floats, this operator will return a float.

For all other combinations of types, we throw an error (incompatible data types).

Below is an example of the `*` operator:

```
int a = 5 * 6

float b = 1.0 * 10.0
```

The program above will have a equal to 30 and b equal to 10.0.

3.4.4.2 Division: /

left associative

```
e1 / e2
```

This operation is only valid when both e1 and e2 are integers or floats. When e1 and e2 are ints, this operator will return an int. When e1 and e2 are floats, this operator will return a float.

For all other combinations of types, we throw an error (incompatible data types).

Below is an example of the `/` operator:

```
int a = 10 / 2
float b = 100.0 / 20.0
```

The program above will have a equal to 5 and b equal to 5.0.

3.4.4.3 Addition: +

left associative

```
e1 + e2
```

This operation is only valid when both e1 and e2 are integers, floats, or strings. When e1 and e2 are ints, this operator will return an int. When e1 and e2 are floats, this operator will return a float. When e1 and e2 are strings, this operator will return a string.

For all other combinations of types, we throw an error (incompatible data types).

Below is an example of the + operator:

```
int a = 1 + 2
float b = 10.1 + 4.1
string c = "hello " + "goat"
```

The program above will have a equal to 3, b equal to 14.2, and c equal to "hello goat".

3.4.4.4 Subtraction: -

left associative

```
e1 - e2
```

This operation is only valid when both e1 and e2 are integers or floats. When e1 and e2 are ints, this operator will return an int. When e1 and e2 are floats, this operator will return a float.

For all other combinations of types, we throw an error (incompatible data types).

Below is an example of the - operator:

```
int a = 10 - 1
float b = 10.0 - 1.9
```

The program above will have a equal to 9 and b equal to 8.1.

3.4.5 Boolean Expressions

Boolean expressions are fundamentally important to the decision constructs used in our language, like the if-else block and inside the conditional statements for loops like while, for and where. Each boolean expression must evaluate to `True` or `False`.

3.4.5.1 Boolean Literal

Boolean expressions can be just a boolean literal, which could be the keyword `True` or `False`.

e.g in if(True) the `True` inside the if conditional is a Boolean Literal.

3.4.5.2 Identifier of boolean variable

Expressions can be just an identifier, as defined for our language in Section 2.1 above. This allows us to use variables as values where needed. QL performs static semantic checking to ensure that the identifier used as a Boolean expression has been defined earlier with bool type.

e.g in if(a) the a inside the if conditional is a Identifier that must be of bool type.

3.4.5.3 not : negation

- not bool_expr evaluates bool_expr as a boolean first and then returns the opposite of the bool_expr (if bool_expr was True, return False; if bool_expr was False, return True)

If the not operator is used on anything other than a bool, we throw an error.

3.4.5.4 Equivalency operators

Operators and the types they can be used on

- == : equivalence
  - string == string
  - int == int
  - float == float
- != : non-equivalence
  - string != string
  - int != int
  - float != float
- > : greater than
  - int > int

    - float > float
- < : less than,
    - int < int
    - float < float
- >= : greater than or equal to,
    - int >= int
    - float >= float
- <= : less than or equal to
    - int <= int
    - float <= float

Each of these operators act on two operands, each of an expr as defined in Section 4.4 above. It is important to note that neither of the operands of the equivalency operator can actually be of boolean types themselves. The operator returns a bool.

Our static semantic checker checks at compile time if the operands on either side of the equivalency operators are of the same data type or not. Since QL does not support type casting, in case the data types fail to match, the compiler reports an error.

Examples of this operator:

- 3 == 3, checks for equality between the two integer literals
- 5.0 != 3, fails to compile because the two operands are of different data types
- a == 5 + 4, evaluates both operands, each an expr before applying the equivalency boolean operator. As such, the data type of a is obtained from the symbol table and then 5 + 4 is evaluated before checking for equality. In the case that, a is not of type int, as inferred from the operand that evaluates to 9, the compiler reports an error.
- a > 5 == 3 fails to work because although the precedence rules evaluate this boolean expression from left to right, a > 5 returns a type of bool which cannot be used in the == operators.

3.4.5.5 Logical operators

- expr1 & expr2: evaluates expr1 and expr2 as booleans (throws error if this is not possible), and returns True if they both evaluate to True; otherwise, returns False.
- expr1 | expr2: evaluates expr1 and expr2 as booleans (throws error if this is not possible), and returns True if either evaluate to True; otherwise, returns False.

3.4.6 Function Calls

A function-call invokes a previously declared function by matching the unique function name and the list of arguments, as follows:

```
<function_identifier> <LPAREN> <arg1> <COMMA> <arg2> <COMMA> ... <RPAREN>
```

This transfers the control of the program execution to the invoked function and waits for it to return before proceeding with computation. Some examples of possible function calls are:

```
array int a = [4;2;1;3]
int b = length(a)
```

The variable b is now equal to 4.

3.5.0 Statements

3.5.1 Declaring Variables

To declare a variable, a data type must be specified followed by the variable name and an equals sign. The right side of the equals sign depends on what type of data type has been declared. If it is a primitive data type, then the user has to specify the corresponding literal of that data type. If the data type is non-primitive, then the user has to enumerate either the array it is assigning into the variable or the JSON constructor with the corresponding JSON file name passed in. In addition, variables can be declared and assigned as another previously declared variable of the same data type.

This is the specific grammar for declaring a variable.

```
<var_decl>:
    | <ARRAY> <array_data_type> <id> <EQUALS> <list_of_literals>
    | <ARRAY> <array_data_type> <id> <EQUALS> <ARRAY> <LPAREN> <int_literal> <RPAREN>
    | <assignment_data_type> <id> <EQUALS> <expr>
<expr>:
    | <literal>
    | <id>
    | ... (other expressions)
```

Examples of the declaration of variables:

```
int i = 0
float f = 1.4 * 5e5
bool b = True
string s = "goats"
array int nums = array(10)
array string strs = ["So","many","features","it's","remarkable"]
```

3.5.2 Updating Variables

To update a variable, the variable on the left side of the equals sign must already by declared. The right side of the equals sign follows the same rules as section 5.1's explanation of declaring variables. The only distinction is this time, there does not need to be a data type prior to the variable name on the left hand side of the equals sign.

This is the specific grammar for reassigning a variable.

```
<var_update>:
    | <id> <EQUALS> <expr>
    | <id> <LSQUARE> <int_literal> <RSQUARE> <EQUALS> <expr>
<expr>:
    | <literal>
    | <id>
    | ... (other expressions)
```

Examples of updating variables (assuming these variables were previously declared as the same type):

```
nums[3] = 42
i = 5 * 9
f = -0.01
s = "GOATS"
```

3.5.3 Return statements

The final statement of the body of a function must be a return statement. A function's return statement must correspond to the return type that was specified after the colon in the function declaration.

This is how our grammar handles return statements:

```
<RETURN> <expr>
```

3.5.4 Function Declaration

Function declarations in QL all start with the function keyword, followed by the function identifier, parentheses with parameter declarations inside, a colon, a return type, and brackets with the function body inside.

3.5.4.1 Parameter declarations

The parameter declarations inside the parentheses are the same as the left hand side of a variable declaration. The variable data type followed by the identifier. These variable declarations are separated by commas.

This is QL's grammar for parameter declarations.

```
<parameter_declaration> :
    | <arg_decl>
    | <parameter_declaration> <COMMA> <arg_decl>
<arg_decl>:
    | <data_type> <id>
```

3.5.4.2 Colon and Return Type

The colon functions in our language as the specifier of a function return type. Before this colon is an argument list and immediately after this colon comes our function return type, which can be any of the data types previously discussed.

This is how our grammar uses colons:

```
<LPAREN> <parameter_declaration> <RPAREN> <COLON> <return_type>
```

3.5.4.3 Grammar for Function Declarations

This is QL's grammar for function declarations.

```
<FUNCTION> <id> <LPAREN> <parameter_declaration> <RPAREN> <COLON> <return_type>
<LCURLY> <stmt_list> <RCURLY>
```

Here is an example of QL code.

```
function average (float x, float y, float z) : float {
  float a = x + y + z
  return a / 3.0
}
```

3.5.5 Loop statements

The loop statements in QL allow us to iteratively call a block of statements in our code.

3.5.5.1 where loops

The where loop is a key feature of QL that allows the user to search through a JSON array and execute a set of statements for all the JSON array elements (key, value pairs by structure) that match a certain boolean condition. For example, consider the following JSON file opened in QL using the `json temp = json("sample.json")` command:

```
{
  "count" : 5,
  "int_index" : 0,
  "owner" : "Matt",
  "number" : 5.4,
  "friends" : [
    {
      "name" : "Anshul",
      "age" : 12
    },
    {
      "name" : "Evan",
      "age" : 54
    },
    {
      "name" : "Gary",
      "age" : 21
    },
    {
      "name" : "Mayank",
      "age" : 32
    }
  ]
}
```

We can run the where loop on the `temp["friends"]` array, with each element of the array resembling the following structure:

```
{
  "name" : "Anshul",
  "age" : 12
}
```

```
{
  "name" : "Evan",
  "age" : 54
}

{
  "name" : "Gary",
  "age" : 21
}

{
  "name" : "Mayank",
  "age" : 32
}
```

A where loop must start with the where keyword, followed by a boolean condition enclosed in parentheses. This condition will be checked against every element in the JSON. The next element is the as <identifier>, which allows the user to associate the current element of the array being processed using the <identifier>. Following this is a {, which marks the beginning of the body code which is applied to each element for which the condition evaluates to true. A closing } signifies the end of the body. After the closing brace, there is a mandatory "in" keyword, which is followed by the JSON array through which the clause will iterate to extract elements.

```
where (<boolean_condition>) as <identifier> {
    #~~ List of statements ~~#
} in <json_array>
```

The scoping rules make the <identifier> available to the <boolean_condition> and the block statements enclosed in the braces. The <json_array> is referenced using the Bracket Selector notation in Section 4.3 above.

For the sample.json file opened using the temp JSON variable shown above, a where loop to print the names of all friends over the age of 21 would look like this in QL:

```
where (friend["age"] >= 21) as friend {
    string name = friend["name"]
    print(name)
} in temp["friends"]
```

3.5.5.2 for loops

The for loop starts with the for keyword, followed by a set of three expressions separated by commas and enclosed by parentheses. The first expression is the initialization, where temporary variables can be initialized. The second expression is the boolean condition; at each iteration through the loop, the boolean condition will be checked. The loop will execute as long as the boolean condition is satisfied, and will exit as soon as the condition evaluates to false. The third expression is the update expression, where variables can be updated at each stage of the loop. Following these three expressions is an open { , followed by a list of statements, and then a close }.

```
for (<initialization>, <boolean_condition>, <update>) {
    #~~ List of statements ~~#
}
```

The <initialization> and the <update> are each assignment statements, as defined in section 5.1 and 5.2 above. The<boolean_condition> is a boolean expression, as defined in section 4.5 above.

3.5.5.3 while loops

The while loop is initiated by the while keyword, followed by a boolean expression enclosed within a set of matching parentheses. After this, there is a block of statements, enclosed by { and }, which are executed in succession as long as the the condition represented by the boolean expression is no longer satisfied.

```
while (<boolean_condition>) {
    #~~ List of statements ~~#
}
```

3.5.6 Conditional Statement

Conditional statements are crucial to the program flow and execute a segment of the code based on a boolean expression.

3.5.6.1 if/else clauses

The if-else clause checks the truth of the boolean condition, and executes the corresponding list of statements depending if the boolean condition provided is True or False. Only the if statement is required and the else statement is optional.

```
if (<boolean_condition>) {
    #~~ List of statements ~~#
} else {
    #~~ List of statements ~~#
}
```

3.6 Built-In Functions

Two built-in functions are included with the language for convenience for the user.

3.6.1 length

length(arr) accepts as its parameter an array, and returns an integer equal to the number of elements in the array.

3.6.2 print

We also include a built-in print function to print strings and primitive types.

```
print(<expr>)
```

Here, <expr> must evaluate to a primitive type. Attempting to print something that is not a primitive will result in an error.

Section 4: Project Plan

We met weekly for the entire semester without fail with the exception of the Wednesday before Thanksgiving break. Our team meetings would be 90 minutes after class on Wednesdays and we'd book a conference room for two weeks ahead to have a consistent location. In the first 15 minutes or so, we would present our weekly updates to our assigned TA, Richard Townsend, and discuss any outstanding issues with him while getting feedback on our progress. In addition, we would set small weekly goals for ourselves and inform Richard about them. Even though these were non-binding, they helped us deliver consistently on the project.

The remaining time of our meetings was spent on discussing the upcoming week's goals. In the later parts of the semester, we also began to use this time to go over merge conflicts that came up while working asynchronously over the week. We organized coding sprints for two or three team members and shared it with our entire team so they were open to all members of the team to join and give inputs on.

Development

We split up our requirements into smaller deliverables and based on the difficulty of the problem, internally assigned one or two team members to each task. Each major feature got its own branch on our Github repository. Upon code completion for that feature, the team members in charge of that feature would open a Pull Request off of our master branch and at least two other team members would review the PR at the least. This style of development enabled us to work asynchronously rather effectively. Each comment on the PR was addressed by the creators before being merged into master. If there was a change to our Parser, then it was clearly articulated in the PR's description so that we could collectively evaluate if it was absolutely necessary.

While building and testing the different features of our compiler, if we came across a bug in one of the preceding steps then we opened a new Github Issue for it and one of the team members took ownership of investigating into it and creating a solution. A few cases required our collective decision and we deferred those to our weekly meeting.

Testing

We strictly adopted a test-driven development style. Our test suite was setup for our Hello, World demo and we stuck with that, adding to the number of tests in our test suite.

As a rule, no feature or bug fix was considered complete till at least two (or three depending on the complexity of the additions) new tests were added to specifically test those features or bug fixes. On each branch, we ran our complete test suite before merging to master to ensure that no additions on that branch broke any of our previous tests.

Style Manifesto

Write QL code.

Values

When writing QL code, strive for readability, simplicity, and brevity, in that order.

Hierarchy

Each QL program should follow a strict hierarchy; that is to say, all components of the program should be compartmentalized in the following order:

1. Program header (Comment explaining program and any required files)
2. JSON imports
3. Global variables
4. Function declarations
5. Program code (incl. where loops)

6. Mandatory newline

Indentation

QL doesn’t care about whitespace, apart from newlines. To that end, indentation is only a convention. We recommend indenting whenever a new scope is created, e.g. inside a function or a loop (where/while/for). Closing brackets should align with the indentation level of the line of code that created the scope being closed.

Empty Lines

There are times where it is appropriate to include an empty line for readability. We recommend an empty line immediately before creation of a new scope (e.g. declaring a function or creating a loop). If the programmer wishes, he can separate “thoughts” of code with newlines, as is common when programming in other languages. **N.B.**: All QL programs must end with an empty line of code.

Comments

Comments begin with `#~~` and end in `~~#`. Every QL program should begin with a comment explaining the usage of the program and any required files (especially JSON in a certain format). Apart from that, comments should be used sparingly. Never use inline comments (i.e., any line of code should not have a comment on the same line). Only use comments where a programmer who is already familiar with QL needs extra context to understand the code.

Naming

Functions and variables should make use of lowerCamelCase when named. Names should be verbose, but not overly so (for example, prefer “endLat” over “el” or “endingLatitude”).

Line Length

QL’s naive parsing of newlines mean that lines of code are occasionally quite long. This is OK.

Project Timeline

Here are some deadlines that we set internally to make sure we maintained a certain level of progress throughout the project. We met weekly with our TA, Richard Townsend, to update him on our progress with these dates and get feedback on the rate of our development.

Task	**Completion Date**
Scanner.mll	November 2
Parser.ml	November 9
Hello, World program	November 16
Semantic Check	December 2
Semantic to JAST	December 9
Code Generation	December 16

Roles and Responsibilities

We started with the following designations:

> * Manager: Matthew Piccolella
> * Systems Architect: Anshul Gupta
> * Tester: Evan Tarrh
> * Language Guru: Gary Lin
> * Systems Integrator: Mayank Mahajan

But the actual division of roles has been fluid at best. We split work by feature and bug fixes instead. This is represented in our git log and our git contributions page for our project.

Software Environment

Communication: Slack
Version Control/Host: Github
Text Editor: Sublime Text 2
Compilers: OCaml version 4.02.3 and Java version 1.8.0_25

Git Log

```
commit faa88c786b49e2b346201e262cdca48a1125d2a7 (HEAD, origin/master,
origin/HEAD, master)
Author: Gary Lin <garylinrule@gmail.com>
Date:   Mon Dec 21 19:46:56 2015 -0500

      Gary's readthrough of LRM -- fixed typos and formatting

commit 559a4f42d47bed842194ed30f1c28ea29502852a
Author: Evan Tarrh <evantarrh@gmail.com>
Date:   Mon Dec 21 18:32:57 2015 -0500

      Aesthetic improvements to style guide

commit 0403afe32da8713c92cd29f7683cd65b21de7c9f
Merge: 8c8d717 18129a4
Author: Evan Tarrh <evantarrh@gmail.com>
Date:   Mon Dec 21 18:30:36 2015 -0500

      Merge branch 'master' of https://github.com/mayankmahajan24/QL

commit 8c8d717544948e64d51b8edb614843a1f5122cda
Author: Evan Tarrh <evantarrh@gmail.com>
Date:   Mon Dec 21 18:30:25 2015 -0500

      Add style guide to docs folder
```

```
commit 18129a4c155e3f40983a0c833138e36c5e9c20c4
Author: Evan Tarrh <evantarrh@gmail.com>
Date:   Mon Dec 21 17:12:03 2015 -0500

      Update README.md with instructions for running code and contributing.

commit a830809ffd31816ce3719bc300720144f00f71c2
Merge: c52bb9e afaeab6
Author: Evan Tarrh <evantarrh@gmail.com>
Date:   Mon Dec 21 16:54:12 2015 -0500

      Merge branch 'master' of https://github.com/mayankmahajan24/QL

commit c52bb9ee30ba0e225c98c1e5911ce896e72f13c0
Author: Evan Tarrh <evantarrh@gmail.com>
Date:   Mon Dec 21 16:54:05 2015 -0500

      Fix buildpath in QL executable

commit afaeab6b01a9c9ae5f549c5f5b378427265adba1
Author: Mayank Mahajan <mayank.mahajan@gmail.com>
Date:   Mon Dec 21 16:47:42 2015 -0500

      More headers

commit cdf33bdad807dd7cb928e0c29d48e16fd31697f4
Author: Mayank Mahajan <mayank.mahajan@gmail.com>
Date:   Mon Dec 21 16:36:05 2015 -0500

      added headers to file

commit 40aad31c4fa0eab567136217c89df066be9f1d49
Merge: 032d250 ebddca0
Author: Matthew Piccolella <mjp2220@columbia.edu>
Date:   Mon Dec 21 16:14:48 2015 -0500

      Merge pull request #53 from mayankmahajan24/lrm

      Lrm

commit ebddca00d44b97c040367554b7c97baad5e930b9
Merge: 6836828 68ecf74
Author: Gary Lin <garylinrule@gmail.com>
```

```
Date:   Mon Dec 21 16:13:42 2015 -0500

      Merge branch 'lrm' of https://github.com/mayankmahajan24/QL into lrm

commit 68368281405d1f52932b8897cb7ee508b9004385
Author: Gary Lin <garylinrule@gmail.com>
Date:   Mon Dec 21 16:13:34 2015 -0500

      removed todos lets go

commit 68ecf74aa0744ed93041b2823e96c5deabe47b1b
Author: Evan Tarrh <evantarrh@gmail.com>
Date:   Mon Dec 21 15:56:25 2015 -0500

      small changes to LRM after team discussion

commit 33eddd3944918a30f54ef5acacdee2c59ce1dceb
Author: Evan Tarrh <evantarrh@gmail.com>
Date:   Mon Dec 21 15:26:21 2015 -0500

      reformatting to be print-ready

commit 1eb1a62facb8c72f0d47080e50e611734b19b07f
Author: Evan Tarrh <evantarrh@gmail.com>
Date:   Mon Dec 21 15:10:32 2015 -0500

      extensive changes to formatting of LRM

commit 032d250eb7cdb130677207ac0707c3ab862f485c
Merge: 40f2e00 8a7278f
Author: Matthew Piccolella <mjp2220@columbia.edu>
Date:   Mon Dec 21 14:22:29 2015 -0500

      Merge pull request #52 from
mayankmahajan24/integration_for_real_this_time

    Integration for real this time

commit 8a7278f16d30fe1f4061a123615c4958ac87384b
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Mon Dec 21 14:19:45 2015 -0500

      Small aesthetic change
```

```
commit b5140206424485473e1ae767f08130c802ad6917
Merge: cc37c85 8088515
Author: Gary Lin <garylinrule@gmail.com>
Date:   Mon Dec 21 14:12:22 2015 -0500

      fixed merge conflict

commit cc37c85808a973006eff36ca11cd2f818549f2e5
Author: Gary Lin <garylinrule@gmail.com>
Date:   Mon Dec 21 14:10:54 2015 -0500

      moving json explanation

commit 80885150f6025ec563f6750862db5e92e5fb7bf6
Author: Evan Tarrh <evantarrh@gmail.com>
Date:   Mon Dec 21 14:00:49 2015 -0500

      Adding table of contents and standardizing 'JSON'

commit e7bc055bb1ebfcbcd2cd5e996eb258b9e33ddb95
(origin/integration_for_real_this_time)
Author: Evan Tarrh <evantarrh@gmail.com>
Date:   Sun Dec 20 17:14:19 2015 -0500

      formatting and compressing integration tests

commit 2b390290b32470984a74194fdb68ca3d9b4ab04b
Author: Evan Tarrh <evantarrh@gmail.com>
Date:   Sun Dec 20 15:46:45 2015 -0500

      adding fitbit integration test. remarkable

commit a57f87ac97a35b67058ec3e5d017e9e1df79601c
Author: Evan Tarrh <evantarrh@gmail.com>
Date:   Sun Dec 20 15:22:03 2015 -0500

      fixed an array typing problem and a function problem,  and wrote a
test just to make sure

commit 9e6535c9742dfe7b3540e4763a7bbf6265a93cd1 (origin/lrm)
Merge: 11401a5 1df3fad
Author: mjp2220 <mjp2220@columbia.edu>
```

```
Date:   Sun Dec 20 15:03:52 2015 -0500

      Merge branch 'lrm' of https://github.com/mayankmahajan24/QL into lrm

commit 11401a5ad683951758ce8c70139e81bf4a0d42b0
Author: Gary Lin <garylinrule@gmail.com>
Date:   Sun Dec 20 02:10:01 2015 -0500

      mostly done, just editing, lets fucking go

commit ff75c5d127168186c1595cb913440df15e1a04ad
Author: Gary Lin <garylinrule@gmail.com>
Date:   Sun Dec 20 02:01:14 2015 -0500

      ok

commit 16a48d5665dace6c727e22f5aaa56aabebd2261f
Author: Gary Lin <garylinrule@gmail.com>
Date:   Sun Dec 20 02:00:05 2015 -0500

      lets fucking go

commit 6d252aba69043091514bab73f4dc8695c3eef31d
Author: Anshul Gupta <anshul.dlf@gmail.com>
Date:   Sun Dec 20 01:52:38 2015 -0500

      fml

commit fca473f6ef01800a926bc7356ea354fb4dc556ac
Author: Gary Lin <garylinrule@gmail.com>
Date:   Sun Dec 20 01:47:51 2015 -0500

      read here for star wars spoilers

commit b4e986f56ebdc678debb8cb761e11f0861ed0f75
Author: Gary Lin <garylinrule@gmail.com>
Date:   Sun Dec 20 01:19:01 2015 -0500

      anshul lookie here

commit fef11174b1bb534217d1047367311b3dd8e32262
Author: Anshul Gupta <anshul.dlf@gmail.com>
Date:   Sun Dec 20 01:18:08 2015 -0500
```

```
    the struggle is real

commit bb7014028ecd4635df358fb5be423feb953e0a80
Author: Gary Lin <garylinrule@gmail.com>
Date:   Sun Dec 20 00:48:17 2015 -0500

    took out toc for now

commit 73155254abf81a377eaf173d2d224255401747ae
Author: Gary Lin <garylinrule@gmail.com>
Date:   Sun Dec 20 00:47:39 2015 -0500

    complete restructuring

commit ef4dd4ca95bd5c1b4c9e8cda0719f4772c8d8009
Author: Gary Lin <garylinrule@gmail.com>
Date:   Sun Dec 20 00:44:43 2015 -0500

    restructuring

commit 185673d6bc69e1254a107f0ca3c9f33473e34ae0
Author: Gary Lin <garylinrule@gmail.com>
Date:   Sun Dec 20 00:06:29 2015 -0500

    anshul look for this shit

commit 22393daeb92b1fff0a0b6c041ca9291a9421f327
Author: Gary Lin <garylinrule@gmail.com>
Date:   Sat Dec 19 22:44:33 2015 -0500

    here anshul k gupta

commit 27ff66564d8152cb97c3034c363918210f98b93c
Author: Gary Lin <garylinrule@gmail.com>
Date:   Fri Dec 18 01:08:36 2015 -0500

    updated lrm skeleton to reflect ql + updated table of contents

commit 41b1cc7873997a6a83eddbe270ff441d10c13e02
Author: Gary Lin <garylinrule@gmail.com>
Date:   Fri Dec 18 00:42:48 2015 -0500
```

```
      the original page break didnt work

commit fa660c8a10cd0ee70c5b483b4663fec8db788c3f
Author: Gary Lin <garylinrule@gmail.com>
Date:   Fri Dec 18 00:40:00 2015 -0500

      page breaks

commit 9097befc92d3bea24085c299deb587f5a11c40bc
Author: Gary Lin <garylinrule@gmail.com>
Date:   Fri Dec 18 00:28:02 2015 -0500

      constructing table of contents

commit 83021308c005f4f49ccf4ecc2302d320462ea734
Author: Gary Lin <garylinrule@gmail.com>
Date:   Fri Dec 18 00:21:23 2015 -0500

      moving stuff around, adding expressions section

commit 8e20aad537e30ffffb265064e8db77d105419331
Author: Gary Lin <garylinrule@gmail.com>
Date:   Fri Dec 18 00:05:25 2015 -0500

      moving stuff around testing

commit 27c350c9702607b64ab9c01951580fa105f4679e
Author: Evan Tarrh <evantarrh@gmail.com>
Date:   Sun Dec 20 14:01:16 2015 -0500

      separate script for integration tests

commit 5d0a0079d8ebed6f6e333650d95063cf92692942
Author: Evan Tarrh <evantarrh@gmail.com>
Date:   Sun Dec 20 13:48:14 2015 -0500

      adding working integration tests 1-4

commit b01d25e28d5fa921ce068679d140c8ba71a39d9b
Author: Evan Tarrh <evantarrh@gmail.com>
Date:   Sun Dec 20 13:30:45 2015 -0500

      extensive bugfixes, preparing for integration tests
```

```
commit 946b9f0f241ffdcd1b57d45e88b394e2fed2f474
Author: Evan Tarrh <evantarrh@gmail.com>
Date:   Sun Dec 20 13:25:31 2015 -0500

      configuring gitignore for large jsons

commit 1df3fad558d92d33372f822dba85203cde425c30
Author: Gary Lin <garylinrule@gmail.com>
Date:   Sun Dec 20 02:10:01 2015 -0500

      mostly done, just editing, lets fucking go

commit fe6877c99e74a7fb65981a564819096914d27563
Author: Gary Lin <garylinrule@gmail.com>
Date:   Sun Dec 20 02:01:14 2015 -0500

      ok

commit 8e972b99240e145cf196988a51fc470dc1ac99b6
Author: Gary Lin <garylinrule@gmail.com>
Date:   Sun Dec 20 02:00:05 2015 -0500

      lets fucking go

commit 11082ddc3235352fae48367d49bf77aaa653abcd
Author: Anshul Gupta <anshul.dlf@gmail.com>
Date:   Sun Dec 20 01:52:38 2015 -0500

      fml

commit bcaf1c79f1049a8e9b3a07c607ea0bcccd8fdea3
Author: Gary Lin <garylinrule@gmail.com>
Date:   Sun Dec 20 01:47:51 2015 -0500

      read here for star wars spoilers

commit 1951d924a8da020dd2adf15ff1dbc2d8c4785be5
Author: Gary Lin <garylinrule@gmail.com>
Date:   Sun Dec 20 01:19:01 2015 -0500

      anshul lookie here
```

```
commit eebbbc63a2831a8b536602ad4a50769f99092ea6
Author: Anshul Gupta <anshul.dlf@gmail.com>
Date:   Sun Dec 20 01:18:08 2015 -0500

      the struggle is real

commit 47b1ceab6c1a84b850c6aefaa4158433d09d9c50
Author: Gary Lin <garylinrule@gmail.com>
Date:   Sun Dec 20 00:48:17 2015 -0500

      took out toc for now

commit b71d49f51a14899c78611a460f50faf12eba4af4
Author: Gary Lin <garylinrule@gmail.com>
Date:   Sun Dec 20 00:47:39 2015 -0500

      complete restructuring

commit e4ae0a6040c6eb64f8bcdc6e4f39765099962980
Author: Gary Lin <garylinrule@gmail.com>
Date:   Sun Dec 20 00:44:43 2015 -0500

      restructuring

commit 698a233a10c36734ed989dda165e0a32828db65e
Author: Gary Lin <garylinrule@gmail.com>
Date:   Sun Dec 20 00:06:29 2015 -0500

      anshul look for this shit

commit 62f724b9a64d04fd14c1f585ba15de477657ce7e
Author: Gary Lin <garylinrule@gmail.com>
Date:   Sat Dec 19 22:44:33 2015 -0500

      here anshul k gupta

commit 40f2e0067383adf5528c193a7f9eadf90a9ac7c5
Merge: 22e19bf c1760cb
Author: Matthew Piccolella <mjp2220@columbia.edu>
Date:   Sat Dec 19 15:46:12 2015 -0500

      Merge pull request #50 from mayankmahajan24/nit_changes
```

```
    Nit changes

commit c1760cb98ee46629d56244ade62a71b1cfb1ef1a (origin/nit_changes)
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Sat Dec 19 15:44:27 2015 -0500

    Getting rid of 'We' in error messages

commit 8e88517bb333874a30c47de0045e6d75c808bda8
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Sat Dec 19 15:25:27 2015 -0500

    Finishing up miscellaneous changes

commit 2d7b7534da0a0d0f180cd289e29869e193b93f1c
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Sat Dec 19 15:05:23 2015 -0500

    Making some aesthetic changes, better error messages

commit 22e19bf66c757976fabc08bc02ceb37bef0efbe5
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Sat Dec 19 14:55:11 2015 -0500

    Quick fix that broke all the tests

commit 0b4bf87cd7c5760838971642db73ec78e122322b
Merge: ceb19a7 9b85c6f
Author: Matthew Piccolella <mjp2220@columbia.edu>
Date:   Sat Dec 19 14:49:45 2015 -0500

    Merge pull request #49 from mayankmahajan24/compile

    Changing so we don't use stdin to compile

commit 7f1d854d279ac5e851ec1d50ae4b23ae330e752c
Author: Gary Lin <garylinrule@gmail.com>
Date:   Fri Dec 18 01:08:36 2015 -0500

    updated lrm skeleton to reflect ql + updated table of contents

commit d29a603a00f743b249a1b75c88befaacfedb0728
Author: Gary Lin <garylinrule@gmail.com>
```

```
Date:   Fri Dec 18 00:42:48 2015 -0500

      the original page break didnt work

commit 4899f788319b79029e8a882811cfa8a45c97a8c9
Author: Gary Lin <garylinrule@gmail.com>
Date:   Fri Dec 18 00:40:00 2015 -0500

      page breaks

commit 3fb8b44f0feb36e32e2c3175d4d4eb7cfc9ed11a
Author: Gary Lin <garylinrule@gmail.com>
Date:   Fri Dec 18 00:28:02 2015 -0500

      constructing table of contents

commit c12b9a141d391cdfd4a21128bf37088d77f1303f
Author: Gary Lin <garylinrule@gmail.com>
Date:   Fri Dec 18 00:21:23 2015 -0500

      moving stuff around, adding expressions section

commit 3526ea6d89dd1ede481d2bb32ebf02c7f90f687c
Author: Gary Lin <garylinrule@gmail.com>
Date:   Fri Dec 18 00:05:25 2015 -0500

      moving stuff around testing

commit 9b85c6fc121e003297c1f8ce13d1dfe3472ba52c (origin/compile)
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Thu Dec 17 22:07:30 2015 -0500

      Changing so we don't use stdin to compile

commit ceb19a7d05e55109cb440381347c611a8cf35a07
Merge: 4a4398c 3b5357f
Author: Mayank Mahajan <mayank.mahajan@gmail.com>
Date:   Thu Dec 17 14:10:21 2015 -0500

      Merge pull request #48 from mayankmahajan24/where_gary_mayank

      Where gary mayank
```

```
commit 3b5357fbed644207893d9a9b9a2ff385c1fefab7 (origin/where_gary_mayank,
where_gary_mayank)
Author: Mayank Mahajan <mayank.mahajan@gmail.com>
Date:   Thu Dec 17 04:57:20 2015 -0500

      Added where tests boi

commit 73a30f3519ed78f28e2599bb252299fe23a991c6
Author: Mayank Mahajan <mayank.mahajan@gmail.com>
Date:   Thu Dec 17 04:38:04 2015 -0500

      Made new branch for debug, removed print statements

commit b3cc87bf34bf3257a501eb0df2d8fbbb4ba6b2ca (origin/debug, debug)
Author: Mayank Mahajan <mayank.mahajan@gmail.com>
Date:   Thu Dec 17 04:34:04 2015 -0500

      ANOTHER ONE

commit 4dbbb6d6b35fb1c2ae52e3f0e2f45e0464a115a3
Author: Mayank Mahajan <mayank.mahajan@gmail.com>
Date:   Thu Dec 17 04:33:45 2015 -0500

      Finished where LETS GOOOOOOOOOO WE OUT HERE BITCHES )))~~~

commit 3df54b686713dfe12f7f5093febe0a11a9cbdf2b
Author: Mayank Mahajan <mayank.mahajan@gmail.com>
Date:   Thu Dec 17 04:33:09 2015 -0500

      Finished where

commit 70025d504f089f73b3f89c2f234d7e20230ca978
Author: Gary Lin <garylinrule@gmail.com>
Date:   Thu Dec 17 03:15:42 2015 -0500

      code gen now works for where and it compiles

commit dcfc1e47009416e3696b7e021da830267dd0ea30
Author: Mayank Mahajan <mayank.mahajan@gmail.com>
Date:   Thu Dec 17 01:58:52 2015 -0500

      Added semantic checking for Where clause
```

```
commit 01ddc2abe759947726962428261bc35032db0427
Author: Gary Lin <garylinrule@gmail.com>
Date:   Wed Dec 16 20:30:09 2015 -0500

      added print statements to check stuff, why is json_selector_update is
not working correctly

commit a421861521fc1367f26a06c4d6beefc1d49de58a
Author: Gary Lin <garylinrule@gmail.com>
Date:   Wed Dec 16 19:18:23 2015 -0500

      lets get it -- adding json typing 13 test and where codegen

commit cde76ca99c49715d0ff443d35a813f8636335d38
Author: Mayank Mahajan <mayank.mahajan@gmail.com>
Date:   Wed Dec 16 19:09:46 2015 -0500

      Added basic JAST for array select assign

commit 04d40d7e7d417a4d9efb3221005a7544719a672d
Author: Mayank Mahajan <mayank.mahajan@gmail.com>
Date:   Wed Dec 16 18:48:08 2015 -0500

      Rebased with master from matt's global scope changes

commit 2941a2ccb4bd9f5280309a80515ab29e585f4e73
Author: Mayank Mahajan <mayank.mahajan@gmail.com>
Date:   Wed Dec 16 18:45:46 2015 -0500

      Array select assign semantic check done

commit e50c1a5363ec2a529702cbd611bbb94f244bfc8a
Author: Mayank Mahajan <mayank.mahajan@gmail.com>
Date:   Tue Dec 15 21:34:30 2015 -0500

      return nothing with json array

commit a362114841e884c4c27abb9b3f4aef559e5dbb3f
Author: Mayank Mahajan <mayank.mahajan@gmail.com>
Date:   Tue Dec 15 21:30:46 2015 -0500

      Fleshed out array_select_assign in semantic
```

```
commit 8b4b986d9fc844b41d71200fabed72bc0d8bda3a
Author: Gary Lin <garylinrule@gmail.com>
Date:   Tue Dec 15 19:48:45 2015 -0500

      where in progress

commit 0ab0e59196dffe672c35c925f5b86f3e487ac036
Author: Mayank Mahajan <mayank.mahajan@gmail.com>
Date:   Tue Dec 15 18:49:27 2015 -0500

      Added where to jast and semantic-to-jast

commit 7fef062b30bc8361666874f9a27e0b84f9ad1227
Author: Gary Lin <garylinrule@gmail.com>
Date:   Tue Dec 15 18:44:04 2015 -0500

      branch shit fixed

commit ba851c178aa5023e38a284b9c171570507d83de2
Author: Gary Lin <garylinrule@gmail.com>
Date:   Tue Dec 15 18:42:46 2015 -0500

      branch problems

commit 4a4398c967aff8012d609392f1b6c586ea6e2325
Merge: 9489a67 d179205
Author: Matthew Piccolella <mjp2220@columbia.edu>
Date:   Wed Dec 16 17:44:43 2015 -0500

      Merge pull request #46 from mayankmahajan24/scoping_issues

      Scoping issues

commit d17920581a131a82fddaccb53682a7e1d7c25869
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Wed Dec 16 17:19:05 2015 -0500

      A couple aesthetic changes

commit 37540c26fb19bd23bb4a15a45619de25dc648a3e
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Wed Dec 16 16:14:39 2015 -0500
```

```
      Finishing global scoping changes

commit 753f53aa437e921d36555c6d5ba1d0d015478128
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Wed Dec 16 16:00:11 2015 -0500

      Getting the tests to pass, adding some scopes for testing

commit 8b63107f42a61cf185dcb559ad2dc8952bd22c88
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Wed Dec 16 15:20:15 2015 -0500

      Fixing issues with JSON

commit bbdff5f4e118975eadeccfdd545f29b3bd61451e
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Wed Dec 16 14:48:42 2015 -0500

      Starting to fix our scoping, working on global variables

commit 9489a670947a637f53d55c6be15f24b4ad01c012
Merge: b4820d8 ca34a23
Author: Matthew Piccolella <mjp2220@columbia.edu>
Date:   Tue Dec 15 18:09:20 2015 -0500

      Merge pull request #39 from mayankmahajan24/json_code_gen

      Json code gen

commit ca34a23d85f3ca8c00df60fed0d7aecc07589fd3
Merge: 957df46 0de348d
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Tue Dec 15 18:08:29 2015 -0500

      Merge branch 'json_code_gen' of https://github.com/mayankmahajan24/QL
into json_code_gen

commit 957df46df4edbf558c0413add878bf59ab435b3e
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Tue Dec 15 18:07:40 2015 -0500

      Fixing outputs with new if/json tests
```

```
commit 1bc3559c26fe07fc0faf316b289887b2c74672c2
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Tue Dec 15 17:49:44 2015 -0500

      Addressing aesthetic concerns

commit 44ca4e1f750b2d7e6661ebfd1b11cfbb20a11bf2
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Tue Dec 15 17:22:59 2015 -0500

      Getting the JSON accesses working, now going back and fixing broken
tests

commit 6212dbc5c103ddb2dfc42c858e62ab6defbc37ca
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Tue Dec 15 13:45:06 2015 -0500

      Got the first working JSON test, also broke some stuff but will
continue

commit a89d2f407471bede600ab9a4ca1793b22f41fc92
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Tue Dec 15 12:50:31 2015 -0500

      Getting JSON test to fail, got reading from file working

commit a522d94b22f2461bcb38ae39c24d484bbe31a1a0
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Tue Dec 15 12:34:52 2015 -0500

      Adding a sample file to show how we use JSON

commit f566e3686c110fe45f556d2580703554672144a3
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Tue Dec 15 11:47:54 2015 -0500

      Getting the JSON build workflow working

commit a4fd224a777e3a687a1828861c45468ad4850e83
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Tue Dec 15 17:30:03 2015 -0500

      Getting the last of the JSON stuff done
```

```
commit 0884322493d8c5415bf49695ce5239b22997b8cf
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Tue Dec 15 17:22:59 2015 -0500

      Getting the JSON accesses working, now going back and fixing broken
tests

commit d280234cc34acb0f2cb303ba41f68b1e96724aea
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Tue Dec 15 13:45:06 2015 -0500

      Got the first working JSON test, also broke some stuff but will
continue

commit 6874a2a5e40ea20e65577380b3fe55de21114917
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Tue Dec 15 12:50:31 2015 -0500

      Getting JSON test to fail, got reading from file working

commit 1d3a42738f8f02e92551b6c75f909f265185c1c7
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Tue Dec 15 12:34:52 2015 -0500

      Adding a sample file to show how we use JSON

commit c3b868928f2db18f054126215ce06d6db781e8fb
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Tue Dec 15 12:08:13 2015 -0500

      Figured out how to run with the given jar

commit 4492f0346df3a32cf4264177b1c63145d1635275
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Tue Dec 15 11:47:54 2015 -0500

      Getting the JSON build workflow working

commit b4820d864006300f61bc365e9b716dc15e122f04
Merge: f35df89 2ac9833
Author: Anshul Gupta <anshul.dlf@gmail.com>
Date:   Tue Dec 15 17:55:58 2015 -0500
```

```
    Merge pull request #33 from mayankmahajan24/ifelse_mayank_gary

    Ifelse mayank gary

commit 2ac98338cddf73336f0b6022fc59b1773bf0b441
Merge: 2fb3c63 9ee4819
Author: Anshul Gupta <anshul.dlf@gmail.com>
Date:   Tue Dec 15 17:54:26 2015 -0500

    Merge branch 'ifelse_mayank_gary' of
https://github.com/mayankmahajan24/QL into ifelse_mayank_gary

commit 0de348d2876191228c42c3f461a54ecf1190069e
Merge: 5785e36 85b3af6
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Tue Dec 15 17:51:26 2015 -0500

    Merge branch 'json_code_gen' of https://github.com/mayankmahajan24/QL
into json_code_gen

commit 5785e366e2c721411c5ec3aec37d577017c1156e
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Tue Dec 15 17:49:44 2015 -0500

    Addressing aesthetic concerns

commit 2fb3c63a6107c338b5af787f6b600a40223356f2
Author: Anshul Gupta <anshul.dlf@gmail.com>
Date:   Tue Dec 15 17:49:28 2015 -0500

    continued

commit 6d1ee2074f435d2d404cfb1147f6183ce5785be3
Author: Anshul Gupta <anshul.dlf@gmail.com>
Date:   Tue Dec 15 17:47:55 2015 -0500

    in progress

commit 0a15687e19a4557656dd9deafd67b483622c3bef
Author: Anshul Gupta <anshul.dlf@gmail.com>
Date:   Tue Dec 15 17:45:51 2015 -0500
```

```
      merge conflict resolution fml

commit bab6699cb5e281ee6b962e030c073526b7889c42
Author: Anshul Gupta <anshul.dlf@gmail.com>
Date:   Sat Dec 12 04:56:28 2015 -0500

      updated all tests. everything good on this end

commit e86edaf44cda76c37c48615903280dcaf2ef4e3f
Author: Anshul Gupta <anshul.dlf@gmail.com>
Date:   Sat Dec 12 04:45:07 2015 -0500

      completed for loops stuff

commit 5d4efd758ad597bc3f8dfc0a83ade7f1b6a2e5e5
Author: Anshul Gupta <anshul.dlf@gmail.com>
Date:   Sat Dec 12 04:20:35 2015 -0500

      completed while tests and code gen

commit b0c8b0844fd19ee5113dc0910f9f272efa20b0c8
Author: Anshul Gupta <anshul.dlf@gmail.com>
Date:   Sat Dec 12 03:47:09 2015 -0500

      skeleton jast while loop

commit 722e824111fa6c9d0b1196bf94fba738d3687523
Author: Mayank Mahajan <mayank.mahajan@gmail.com>
Date:   Thu Dec 10 18:06:38 2015 -0500

      Added tests for if/else

commit d23a5028b3efd4d62a2d0389ae42f1a3adf3cf38
Author: Mayank Mahajan <mayank.mahajan@gmail.com>
Date:   Thu Dec 10 17:32:42 2015 -0500

      Added if/else functionality

commit 6c66c2ed84ed11552f2c2686a045a6da44d82b8f
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Tue Dec 15 17:30:03 2015 -0500

      Getting the last of the JSON stuff done
```

```
commit 0cffc630d0afc2a416c365b325e4a4a74a6de215
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Tue Dec 15 17:22:59 2015 -0500

      Getting the JSON accesses working, now going back and fixing broken
tests

commit 32e44a9cfd32ed98968fa067e15401d7b28d8f6f
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Tue Dec 15 13:45:06 2015 -0500

      Got the first working JSON test, also broke some stuff but will
continue

commit d0ee7531a75ebe35283e8bbf6706676dc0f07f1b
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Tue Dec 15 12:50:31 2015 -0500

      Getting JSON test to fail, got reading from file working

commit bce5cedfa2cd26ebaaccd88316e28bd1c44ca141
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Tue Dec 15 12:34:52 2015 -0500

      Adding a sample file to show how we use JSON

commit 2a9bf8c0846f99ff888252f3ef4357bf7b56f30a
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Tue Dec 15 12:08:13 2015 -0500

      Figured out how to run with the given jar

commit 024c5108701019904a6350a07e978e211e951e19
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Tue Dec 15 11:47:54 2015 -0500

      Getting the JSON build workflow working

commit 85b3af69a3cf646caa92a2eb3973747476487165
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Tue Dec 15 17:30:03 2015 -0500
```

```
      Getting the last of the JSON stuff done

commit 9ee481957ac4ab2b0b2abd3871bd2fbdf1bb9884
Merge: 907bca3 08aead9
Author: garymlin <garylinrule@gmail.com>
Date:   Tue Dec 15 17:25:48 2015 -0500

      Merge pull request #34 from mayankmahajan24/where/for

      While/for

commit 2ed95a51701a3957ae2f9d1d6c43e0d1fb37d3d8
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Tue Dec 15 17:22:59 2015 -0500

      Getting the JSON accesses working, now going back and fixing broken
tests

commit f35df89eb09ae534b3fa2d8128e56d5fb06986da
Merge: 7e98e99 ac1e138
Author: Anshul Gupta <anshul.dlf@gmail.com>
Date:   Tue Dec 15 17:17:27 2015 -0500

      Merge pull request #37 from mayankmahajan24/array_improvements

      Array improvements

commit 08aead99d42bf2f1220dfc0157b59187a4217517
Author: Anshul Gupta <anshul.dlf@gmail.com>
Date:   Tue Dec 15 14:50:49 2015 -0500

      fixed order of eval for for and where syntax

commit bcce595e350226dc88716baa8a7bc8db06454dde
Author: Anshul Gupta <anshul.dlf@gmail.com>
Date:   Tue Dec 15 14:42:35 2015 -0500

      fixing order of eval for if/else constructs

commit 6b4fb83fee7d60b688cc43d2c097e674acd1d8c1
Author: Anshul Gupta <anshul.dlf@gmail.com>
Date:   Tue Dec 15 14:39:45 2015 -0500
```

```
      fixing the order of eval in the while loop

commit e343c51a25ae346adec13975eefb7a8cdd4dd5e4
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Tue Dec 15 13:45:06 2015 -0500

      Got the first working JSON test, also broke some stuff but will
continue

commit 7f55df3f2e4ac4c7107d5c6e04b9d6c7d82ebe40
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Tue Dec 15 12:50:31 2015 -0500

      Getting JSON test to fail, got reading from file working

commit e1215ddad8c245a739493f5daa649fecf6926fa3
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Tue Dec 15 12:34:52 2015 -0500

      Adding a sample file to show how we use JSON

commit 7861979925272dd56f2f1afba0efccea37d2aa79
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Tue Dec 15 12:08:13 2015 -0500

      Figured out how to run with the given jar

commit 783dec021a7bfbac8f0995e27f8501caedf9cc6a
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Tue Dec 15 11:47:54 2015 -0500

      Getting the JSON build workflow working

commit ac1e1380f9ca8c1bb7848fe9c7602c401557aeae
Author: Evan Tarrh <evantarrh@gmail.com>
Date:   Mon Dec 14 20:37:21 2015 -0500

    making compatible with JSON typing changes

commit 42d408c416b3680dba91c5954fb0bcf641c0816d
Author: Evan Tarrh <evantarrh@gmail.com>
Date:   Thu Dec 10 21:18:26 2015 -0500
```

```
      arrays are no longer immutable! lol

commit 67a385763386bbf6c8297fd76007069bbbdb051a
Author: Evan Tarrh <evantarrh@gmail.com>
Date:   Thu Dec 10 18:24:40 2015 -0500

      implemented length(array)

commit f5bc06149d2eee8b853345d6e97063031a523a1b
Author: Evan Tarrh <evantarrh@gmail.com>
Date:   Thu Dec 10 16:35:27 2015 -0500

      implementing fixed-length array creation, still trying to fix tests

commit 7e98e996dea425b10c39559d3dd669dac9ef6968
Merge: 22b55c1 d0a76f5
Author: Matthew Piccolella <mjp2220@columbia.edu>
Date:   Sun Dec 13 23:21:23 2015 -0500

      Merge pull request #35 from mayankmahajan24/json_auto_typing

      Adding json auto typing

commit d0a76f560df18b9d8c9a1902425570f4dbaaecc5
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Sun Dec 13 21:23:03 2015 -0500

      Finishing automatic typing, fixing some tests, getting rid of all the
warnings

commit 1b4eb2b63b0c6e8bfa6b6998457f749e7ce9147a
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Sun Dec 13 16:59:35 2015 -0500

      Finishing inferring of JSON types in expressions, still have to do
bool exprs

commit fe4d5ebf4c44679b35acaa6c353d411ae1f2aa23
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Sun Dec 13 15:15:51 2015 -0500

      Rewriting check_expr_type so we return an env with any types we
defined in the expression
```

```
commit 2f5dd967382f47e0def6d472e6ea36cf3f8c70df
Author: Anshul Gupta <anshul.dlf@gmail.com>
Date:   Sat Dec 12 04:56:28 2015 -0500

      updated all tests. everything good on this end

commit b66c909106f3103272f2cb6e8194e27899aaf94c
Author: Anshul Gupta <anshul.dlf@gmail.com>
Date:   Sat Dec 12 04:45:07 2015 -0500

      completed for loops stuff

commit 6ac3d4f077ff5886da52c574d9bec27a768117c3
Author: Anshul Gupta <anshul.dlf@gmail.com>
Date:   Sat Dec 12 04:20:35 2015 -0500

      completed while tests and code gen

commit 64ac615989ca9b18b307621e8d21959ffafd9472
Author: Anshul Gupta <anshul.dlf@gmail.com>
Date:   Sat Dec 12 03:47:09 2015 -0500

      skeleton jast while loop

commit fb9d9cbe91e79a1490764b715fadb019d25bbc72
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Fri Dec 11 18:20:28 2015 -0500

      Fixing test that makes me go crazy

commit 99f460db67bc56051e9e99a5ad25078e1acfce56
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Fri Dec 11 18:19:35 2015 -0500

      Getting function mapping for JSON types

commit 4ad8adf54e124d5aa1e413b29d5d7669fbb234bd
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Fri Dec 11 16:29:15 2015 -0500

      The start of json auto typing
```

```
commit 22b55c158a263d6e9c8f2d21278829915d1ee6af
Merge: cfbf2d3 3e1fa74
Author: Matthew Piccolella <mjp2220@columbia.edu>
Date:   Fri Dec 11 13:43:35 2015 -0500

      Merge pull request #19 from mayankmahajan24/json_sast

      JSON changes

commit 3e1fa740ed2f8e4933f33655ae17b1e493575991
Merge: 4b8e5b6 4258711
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Fri Dec 11 13:31:40 2015 -0500

      Merge branch 'json_sast' of https://github.com/mayankmahajan24/QL
into json_sast

commit 4b8e5b6a09bd36981b629e1a5726c96ff319462f
Author: Anshul Gupta <anshul.dlf@gmail.com>
Date:   Mon Dec 7 05:22:25 2015 -0500

      wait it actually might be working

commit f9bc9ceb7f5840ad7be55fe04813494337dfaade
Author: Anshul Gupta <anshul.dlf@gmail.com>
Date:   Mon Dec 7 03:21:52 2015 -0500

      preventing json aliasing

commit db560f01bc1ddc02e338b25031017d3cf569793d
Author: Anshul Gupta <anshul.dlf@gmail.com>
Date:   Mon Dec 7 05:34:21 2015 -0500

      relevant tests. all 51 pass

commit 0dc4df20e8fdd6aa9f8f5748182e94d4bd618dbc
Author: Anshul Gupta <anshul.dlf@gmail.com>
Date:   Mon Dec 7 05:22:25 2015 -0500

      wait it actually might be working

commit d28c2d77c974adc77b157b75f98d95fcc553f7e9
Author: Anshul Gupta <anshul.dlf@gmail.com>
```

```
Date:   Mon Dec 7 04:42:10 2015 -0500

      oh come on

commit 1583e0d6a95eb802b5e66fd0c13ed6cbde3c95f4
Author: Anshul Gupta <anshul.dlf@gmail.com>
Date:   Mon Dec 7 03:21:52 2015 -0500

      preventing json aliasing

commit cfbf2d3aa69de5de7c5c71413481a01d2440944c
Merge: dc127cc a3a6be8
Author: Matthew Piccolella <mjp2220@columbia.edu>
Date:   Thu Dec 10 22:16:46 2015 -0500

      Merge pull request #32 from mayankmahajan24/remove_compile_lol

      removing unnecessary compile.ml

commit 907bca3f791d5bcbb0b35ac42dc10513e92d1d76
Author: Mayank Mahajan <mayank.mahajan@gmail.com>
Date:   Thu Dec 10 18:06:38 2015 -0500

      Added tests for if/else

commit a3a6be84e6e20a9c2e5c1e11c8b8ec3e800f3ecc
Author: Evan Tarrh <evantarrh@gmail.com>
Date:   Thu Dec 10 17:44:07 2015 -0500

      removing unnecessary files

commit 558880db6795d2d79c8535d1828d86c9f6b16d04
Author: Mayank Mahajan <mayank.mahajan@gmail.com>
Date:   Thu Dec 10 17:32:42 2015 -0500

      Added if/else functionality

commit 42587114f93673dc8d10d458cac81d70e06a3386
Merge: 98f7c8a 32ed1b3
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Thu Dec 10 16:40:37 2015 -0500
```

```
      Merge branch 'json_sast' of https://github.com/mayankmahajan24/QL
into json_sast

commit 98f7c8a491b887bb1439953208b7cd4eeda5d15d
Author: Anshul Gupta <anshul.dlf@gmail.com>
Date:   Mon Dec 7 05:34:21 2015 -0500

      relevant tests. all 51 pass

commit 77a5a481b0b5f2833f3d4b2007d965d2f01a3149
Author: Anshul Gupta <anshul.dlf@gmail.com>
Date:   Mon Dec 7 05:22:25 2015 -0500

      wait it actually might be working

commit 8097e542ff826d592f0f7a441b3bdcbf3ea86fb6
Author: Anshul Gupta <anshul.dlf@gmail.com>
Date:   Mon Dec 7 04:42:10 2015 -0500

      oh come on

commit 8befdc3958f92c44449fbc70413458b04569c95b
Author: Anshul Gupta <anshul.dlf@gmail.com>
Date:   Mon Dec 7 03:21:52 2015 -0500

      preventing json aliasing

commit dc127cc73c4be8c63fadfd2ed2c293a3601fb6c6
Merge: 94a3538 f5df843
Author: garymlin <garylinrule@gmail.com>
Date:   Thu Dec 10 00:14:14 2015 -0500

      Merge pull request #23 from mayankmahajan24/jast_mayank_gary

      Jast mayank gary

commit f5df8438caa7d13ce30058b1d0c111ce98c1d5bc
Author: Gary Lin <garylinrule@gmail.com>
Date:   Thu Dec 10 00:11:52 2015 -0500

      fixed returning bool problem + test cases + everything passes...
letsssssss gooooooooo
```

```
commit 1e3bdc79bbfbd20634a3fd28d9a8d8b9b07ab9ca
Author: Mayank Mahajan <mayank.mahajan@gmail.com>
Date:   Wed Dec 9 19:42:16 2015 -0500

      Fixed issue with doubles floats, we are converting floats to doubles
when we print to Java

commit e9d782c5bc5a26214c42bdc9b906f47e6b09044d
Author: Gary Lin <garylinrule@gmail.com>
Date:   Wed Dec 9 02:22:17 2015 -0500

      add func_decl tests

commit beff4f0e49f74f1c38e0f3fcb6bdce069280287f
Author: Gary Lin <garylinrule@gmail.com>
Date:   Wed Dec 9 02:11:57 2015 -0500

      completed function declaration

commit 26b28ff9ac2f5538c363a33797be723392fe341c
Author: Mayank Mahajan <mayank.mahajan@gmail.com>
Date:   Wed Dec 9 01:30:04 2015 -0500

      Fixed issue with defining statements, function name gets messed up
still?

commit 9fd3873588c8e90959695c439f86d08cb270e1a3
Merge: 2bb47db 4590b9f
Author: Gary Lin <garylinrule@gmail.com>
Date:   Wed Dec 9 00:58:17 2015 -0500

      "rebasing?"
      Merge branch 'jast_mayank_gary' of
https://github.com/mayankmahajan24/QL into jast_mayank_gary

commit 2bb47dbf71395d8f95ec6499180a77fc1a7291e4
Author: Mayank Mahajan <mayank.mahajan@gmail.com>
Date:   Wed Dec 9 00:54:17 2015 -0500

      Trying one fix with converting function body, this is broken atm

commit 8a3a2edc3fbc35a1666f73ff2c11addfb8d0613e
Author: Gary Lin <garylinrule@gmail.com>
```

```
Date:   Tue Dec 8 17:49:29 2015 -0500

      some test for function declaration, still working on function body

commit 5af891a5e07b341814111f1fd0099c87356f9be4
Author: Mayank Mahajan <mayank.mahajan@gmail.com>
Date:   Tue Dec 8 01:36:36 2015 -0500

      Added function call codegen

commit 4590b9fffcb680f16a7c98b6ced7140f0ee81d24
Author: Mayank Mahajan <mayank.mahajan@gmail.com>
Date:   Wed Dec 9 00:54:17 2015 -0500

      Trying one fix with converting function body, this is broken atm

commit 94a353891b1d53e050e0fe6257343cda5ff63984
Merge: 2d88587 ddfeb9e
Author: Evan Tarrh <evantarrh@gmail.com>
Date:   Tue Dec 8 21:18:21 2015 -0500

      Merge pull request #22 from mayankmahajan24/fix_parser

      Removing parser conflicts & improving endline handling

commit ddfeb9e0aea00704664f2a5c5e662e0e165db9fd
Author: Evan Tarrh <evantarrh@gmail.com>
Date:   Tue Dec 8 21:05:03 2015 -0500

    removing unnecessary file and improving test script

commit 178dbde9f2159b746b60a9ac8c46753bcaf4aa05
Author: Gary Lin <garylinrule@gmail.com>
Date:   Tue Dec 8 17:49:29 2015 -0500

      some test for function declaration, still working on function body

commit 0de9e6bba5780a1924b7a89ede36c75bec677528
Author: Mayank Mahajan <mayank.mahajan@gmail.com>
Date:   Tue Dec 8 01:36:36 2015 -0500

      Added function call codegen
```

```
commit 2d885873a7cf3787de722a890f2bd7cf0dcc746e
Merge: 40562fe e5271d1
Author: garymlin <garylinrule@gmail.com>
Date:   Tue Dec 8 00:16:05 2015 -0500

      Merge pull request #21 from mayankmahajan24/gary

      not allowing reassigning of booleans

commit 66ec28ee029142361df9187f250d6521fe96d235
Author: Evan Tarrh <evantarrh@gmail.com>
Date:   Mon Dec 7 20:25:17 2015 -0500

      removing parser.conflicts because oops

commit a6900832a4ad3e464396bd4ae62af34fdfdd86bb
Author: Evan Tarrh <evantarrh@gmail.com>
Date:   Mon Dec 7 20:19:01 2015 -0500

      Fixing bugs and improving test script

commit 1ca6a04e69117b8126ddf79cf0aac781f3594ee5
Author: Evan Tarrh <evantarrh@gmail.com>
Date:   Mon Dec 7 19:48:41 2015 -0500

      fixing shift/reduce conflicts in parser, adding endline as statement

commit e5271d1c8d0e4e4f791c543b21f94663aaf4ef35
Author: Gary Lin <garylinrule@gmail.com>
Date:   Mon Dec 7 19:58:22 2015 -0500

      LETS FUCKING GO

commit e88014c358754b0aa96e89cd478e256fc5b49d5f
Author: Gary Lin <garylinrule@gmail.com>
Date:   Mon Dec 7 19:57:07 2015 -0500

      i fucked up

commit 40562fe5fc02d9f188dec213d2e560f8dbaa1047
Merge: ce9ada8 a2831eb
Author: Matthew Piccolella <mjp2220@columbia.edu>
Date:   Mon Dec 7 19:52:20 2015 -0500
```

```
      Merge pull request #18 from mayankmahajan24/code_gen_skeleton

      Code gen skeleton

commit 0beb28f2f7cf6af16411cedee82a102631576059
Author: Gary Lin <garylinrule@gmail.com>
Date:   Mon Dec 7 19:48:46 2015 -0500

      not allowing reassigning of booleans

commit 32ed1b3dcaa9158495471e52cb6b9c0a3b569eb2
Author: Anshul Gupta <anshul.dlf@gmail.com>
Date:   Mon Dec 7 05:34:21 2015 -0500

      relevant tests. all 51 pass

commit 456fadcf488980cf2ffb20a8ee24e009eb14c86a
Author: Anshul Gupta <anshul.dlf@gmail.com>
Date:   Mon Dec 7 05:22:25 2015 -0500

      wait it actually might be working

commit 2636ef6c539e3fb7160fa61895dc3457542c8e27
Author: Anshul Gupta <anshul.dlf@gmail.com>
Date:   Mon Dec 7 04:42:10 2015 -0500

      oh come on

commit c65931bd88a6552ad0d1e17ac7918a8de584eb83
Author: Anshul Gupta <anshul.dlf@gmail.com>
Date:   Mon Dec 7 03:21:52 2015 -0500

      preventing json aliasing

commit a2831eb84ffaec94315bbef2c3ea4eb9a73ce5a9
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Mon Dec 7 01:17:55 2015 -0500

      Adding script to run code

commit 716bc84e7a7efa80290a1da46518f484d17615f8
Author: mjp2220 <mjp2220@columbia.edu>
```

```
Date:   Sun Dec 6 00:06:39 2015 -0500

      Got boolean expressions working, hell yes

commit 0786c60cac013e8afe4a24983aff3a7efc423786
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Fri Dec 4 18:12:43 2015 -0500

      Updating vars and arrays

commit 2b499147f03ca0016145e3658187d4d0763eb4f4
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Fri Dec 4 16:43:26 2015 -0500

      Getting output for all the assign tests

commit 6dcda710c8492b70193a51afbf5a791462e26b97
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Fri Dec 4 00:34:58 2015 -0500

      Trying to get printing of integers working

commit 4bda6bc1bd41b1fcede1f55c64af3c549e2b1704
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Thu Dec 3 17:48:19 2015 -0500

      Adding ability to pass a file name, might want to switch later so
file name is inferred from input

commit d6467570f56f0c7d4573fe1712c1807958979fba
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Thu Dec 3 17:36:36 2015 -0500

      Getting a working skeleton with JAST and code generation

commit ce9ada804264fa70dd1cae87b2bdc08dcf9a9b68
Merge: 0aad6e3 d40dd76
Author: Matthew Piccolella <mjp2220@columbia.edu>
Date:   Thu Dec 3 15:13:40 2015 -0500

      Merge pull request #15 from mayankmahajan24/sast_loops

      Sast loops
```

```
commit d40dd762972dd3133b398a7d9759ccefd067b2d3
Merge: d96307c adf7ced
Author: Evan Tarrh <evantarrh@gmail.com>
Date:   Wed Dec 2 18:45:16 2015 -0500

      Merge pull request #17 from mayankmahajan24/sast_loops_evan

      Semantic checking for where

commit adf7ced4642e53a045588318f9aa0503d463c3eb
Author: Mayank Mahajan <mayank.mahajan@gmail.com>
Date:   Wed Dec 2 17:45:28 2015 -0500

      Removed NOBUENO, added AnyType checking for bool binop exps

commit f527d9c4fb11d07423294569bdfd940cc542078a
Author: Evan Tarrh <evantarrh@gmail.com>
Date:   Wed Dec 2 03:47:20 2015 -0500

      adding tests

commit a7015722eac3155b20d61448cf1bb2e4dad6c8f0
Author: Evan Tarrh <evantarrh@gmail.com>
Date:   Wed Dec 2 03:33:50 2015 -0500

      adding different parsing options for where

commit fbd0984df91ebb9e9e6114b6fb4835aa4c7365db
Author: Evan Tarrh <evantarrh@gmail.com>
Date:   Wed Dec 2 03:24:23 2015 -0500

      excellent fix

commit 5260dec96ea5eba9ea709abe5ce61cb877ead1f9
Author: Evan Tarrh <evantarrh@gmail.com>
Date:   Wed Dec 2 03:20:05 2015 -0500

      modifying where conditions and improving test

commit d7da9bca1dbf6d50c5f96ade7c44e3cb6297584a
Author: Evan Tarrh <evantarrh@gmail.com>
Date:   Wed Dec 2 02:55:01 2015 -0500
```

```
      first pass at where loops. tests do not yet pass but everything
compiles

commit d96307c6a40ccd7fb5dc36bcae332f202d0e74be
Author: Mayank Mahajan <mayank.mahajan@gmail.com>
Date:   Tue Dec 1 20:40:04 2015 -0500

      Fixed while tests again, accepting expressions as statements

commit 6e35f24288a1961096d92a377aca240b12488727
Author: Mayank Mahajan <mayank.mahajan@gmail.com>
Date:   Tue Dec 1 20:25:24 2015 -0500

      Fixed while tests

commit 9f063afb702f8e3542fec6e2e32589478e1fe36f
Author: Mayank Mahajan <mayank.mahajan@gmail.com>
Date:   Tue Dec 1 19:31:03 2015 -0500

      Changed boolean literals and added tests for boolean assignment

commit 4dfe89954649d95d30573e0be3c0e2a81e4ba5d2
Author: Gary Lin <garylinrule@gmail.com>
Date:   Tue Dec 1 19:18:27 2015 -0500

      assigning bool to bool_expr

commit e30c029f34a6b886e996520e568e41988f1518e4
Author: Mayank Mahajan <mayank.mahajan@gmail.com>
Date:   Tue Dec 1 18:59:34 2015 -0500

      Added while tests

commit bcaa2d51e81266750ae5a072ea3a5b2287b55e7a
Author: Gary Lin <garylinrule@gmail.com>
Date:   Tue Dec 1 18:48:28 2015 -0500

    change parser for while formatting

commit 1a3b6fe47aba9eccdb2228fe8f264daf05f5d54f
Author: Gary Lin <garylinrule@gmail.com>
Date:   Tue Dec 1 18:43:50 2015 -0500
```

```
      tests for 'for' loop

commit bf75c2e0ca47fe001d8a7c43e9c300ba9f84febe
Author: Mayank Mahajan <mayank.mahajan@gmail.com>
Date:   Tue Dec 1 18:08:46 2015 -0500

      Added ifelse tests

commit d7f4882818ed28095de6ca6e6301de1205a64def
Author: Gary Lin <garylinrule@gmail.com>
Date:   Tue Dec 1 18:07:52 2015 -0500

      ifelse and for tests

commit ada2462950f07b06d5422d882f78445c24701a73
Author: Mayank Mahajan <mayank.mahajan@gmail.com>
Date:   Tue Dec 1 18:00:10 2015 -0500

      Added boolean checking to if and while

commit 49fc29e5eaca45a1ebbe7855d962b62780cb4249
Author: Gary Lin <garylinrule@gmail.com>
Date:   Tue Dec 1 17:54:30 2015 -0500

      testing condition on ifelse

commit 42f550c91a7355c001da3232080990fa21eaa82b
Author: Mayank Mahajan <mayank.mahajan@gmail.com>
Date:   Tue Dec 1 17:48:26 2015 -0500

      Added while functionality

commit 4ab1de1cec83bcb379d631c0b22d257833574823
Author: Gary Lin <garylinrule@gmail.com>
Date:   Mon Nov 30 21:24:51 2015 -0500

      2nd iteration lettsss gooo

commit 06e0fd639efa0a3f7afdeab4ec244eb362744202
Author: Gary Lin <garylinrule@gmail.com>
Date:   Mon Nov 30 21:05:49 2015 -0500
```

```
      first test of for

commit de07ee0f477b182bfee9c2c1df1912dc8d172abc
Author: Gary Lin <garylinrule@gmail.com>
Date:   Mon Nov 30 19:28:24 2015 -0500

      first hunk staged baby

commit 0aad6e3ab9809db6753f2183e7f616daa990ff6b
Merge: 24cb611 26935f0
Author: Anshul Gupta <anshul.dlf@gmail.com>
Date:   Mon Nov 30 00:01:30 2015 -0500

      Merge pull request #14 from mayankmahajan24/sast_anshul_mayank

      Sast anshul mayank

commit 26935f02e7fb0ad2d4d962e4b027c3f859dc9a17
Author: Anshul Gupta <anshul.dlf@gmail.com>
Date:   Mon Nov 30 00:01:05 2015 -0500

      aesthetic changes to satisfy man's innate desire to achieve universal
harmony

commit a9241ced823e480928543882b155ec1a890efd6d
Author: Anshul Gupta <anshul.dlf@gmail.com>
Date:   Sat Nov 28 13:49:37 2015 -0500

      finished if/else clause and update variable functionality

commit e173b3d6002f3876620b3d3bf9790832cadd0fb1
Author: Anshul Gupta <anshul.dlf@gmail.com>
Date:   Sat Nov 28 13:05:50 2015 -0500

      all 22 tests pass. array initialization stuff completed

commit 638ace4890cc0283045e2d6b622d7aa0c035976a
Author: Anshul Gupta <anshul.dlf@gmail.com>
Date:   Sat Nov 28 11:55:28 2015 -0500

      removing unnecessary .conflicts file

commit 24cb611a4f5ed097b2214af238eef84ea1f77855
```

```
Merge: 00e3b65 a8144f1
Author: Matthew Piccolella <mjp2220@columbia.edu>
Date:   Sat Nov 28 11:09:51 2015 -0500

      Merge pull request #12 from mayankmahajan24/sast

      Adding semantic checking to our parser

commit 4b5bae1f218556a1517d07d714f50faba4e6b443
Author: Mayank Mahajan <mayank.mahajan@gmail.com>
Date:   Tue Nov 24 01:48:08 2015 -0500

      Shit

commit a8144f1bf0987a133e0df63aed0faddc75b6209f
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Mon Nov 23 20:47:12 2015 -0500

      Getting rid of accidental parser conflicts file

commit 4e38ee7d3f9ea21f78304dab020a9ef5f4b3629e
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Mon Nov 23 20:37:46 2015 -0500

      Checking if JSON can do double int access

commit 0c3c6382a7d993e4bab5953c1041df30bb0366ce
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Mon Nov 23 20:22:34 2015 -0500

      Adding multidimensional array test

commit 850af1ea6935912a252a778ef9e91c60620eed8d
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Mon Nov 23 20:21:00 2015 -0500

      Implementing json and array type checking with tests

commit 603c067a5a0f144fe1038ab6ad9d6678278f77cd
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Mon Nov 23 16:45:09 2015 -0500

      Adding call as an expression type
```

```
commit 8ba55e06bd903702961e7930edcb48fe7555bc21
Merge: 25cb92c 3d5c8ef
Author: Matthew Piccolella <mjp2220@columbia.edu>
Date:   Mon Nov 23 16:14:31 2015 -0500

      Merge pull request #11 from mayankmahajan24/sast_matt_evan

      Merging this into sast so we can review

commit 3d5c8efe13412e8d615d97c9e3be0549af37f016
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Mon Nov 23 16:13:23 2015 -0500

      Adding recap of the tests

commit b7181ff054a3f1f39e46105c13958b41ac43dd17
Merge: bbc7703 d674f25
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Mon Nov 23 13:35:37 2015 -0500

      Merge branch 'sast_matt_evan' of
https://github.com/mayankmahajan24/QL into sast_matt_evan

commit bbc7703c54c362a53679e27563e52e13922f6468
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Mon Nov 23 13:33:23 2015 -0500

      Verifying function calls, passed parameter types

commit 4018bc04ea631efc98acc5b01fd0a50a714e7673
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Mon Nov 23 04:57:48 2015 -0500

      Removing junk file

commit c9347320af400fea1ebbfe5642b82733e43044e9
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Mon Nov 23 04:56:49 2015 -0500

      Getting functions done, adding return type verification

commit b1b8128549d161d2ea0fa5426b1aa23082281bde
```

```
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Mon Nov 23 03:30:15 2015 -0500

      Adding function definition

commit a4cb6575021ce8c12437b5ce1634821e3ad78b68
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Mon Nov 23 02:55:11 2015 -0500

      Adding variable assignment, revamping test suite

commit 2f4352508bd5eb27a22b9a77c39d96fa3d0fe807
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Sun Nov 22 23:59:15 2015 -0500

      Getting binary operators working, writing our first fail test

commit e090b68c822942acadc23188d5e9107cd8cf7971
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Sun Nov 22 21:26:45 2015 -0500

      Fixing broken test compares, adding some expression stuff

commit d674f2573801d8d8eba882419e509806c5c4810e
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Mon Nov 23 04:57:48 2015 -0500

      Removing junk file

commit 0f73cb51eab2df7a6eabad19f632578671ea0c31
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Mon Nov 23 04:56:49 2015 -0500

      Getting functions done, adding return type verification

commit 60b69ecb2ca6f93d71bfb2024ca759a2dcc8b481
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Mon Nov 23 03:30:15 2015 -0500

      Adding function definition

commit 1704892d095563ed8d218bab933a0409f6e16ede
Author: mjp2220 <mjp2220@columbia.edu>
```

```
Date:   Mon Nov 23 02:55:11 2015 -0500

      Adding variable assignment, revamping test suite

commit 5caa5e759e0161f45ca0f93ac942aa8fa79963f7
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Sun Nov 22 23:59:15 2015 -0500

      Getting binary operators working, writing our first fail test

commit a25c684fcc9a848db81639cccc9af23c8ef4619f
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Sun Nov 22 21:26:45 2015 -0500

      Fixing broken test compares, adding some expression stuff

commit 25cb92c6ba7f73b1f7504f906dd0415a2cfcb6e6
Author: Gary Lin <garylinrule@gmail.com>
Date:   Sun Nov 22 20:57:20 2015 -0500

      error checking mismatch

commit 00e3b65da8fcb45779849ec66a17620e0c407a86
Author: Anshul Gupta <anshul.dlf@gmail.com>
Date:   Sun Nov 22 20:36:53 2015 -0500

      sast in progress, good going. weather is looking good

commit bf8fb227aec28726bea05676b78e8dd56591e043
Merge: deefda7 89c3fdc
Author: Matthew Piccolella <mjp2220@columbia.edu>
Date:   Mon Nov 16 22:54:50 2015 -0700

      Merge pull request #10 from mayankmahajan24/hello_world

      Hello world

commit 89c3fdce7e0c3ee4311cca6f01705198ac065d59
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Mon Nov 16 22:21:13 2015 -0500

      No need for test
```

```
commit 0f9c7c481d7d29dc73aa89c70c2717561bb57ba6
Merge: 3a9e66f 04936b4
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Mon Nov 16 22:20:43 2015 -0500

      Merge branch 'hello_world' of https://github.com/mayankmahajan24/QL
into hello_world

commit 3a9e66fb4e945ff27e0c632c8e1cc5c2f99bc1e2
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Mon Nov 16 15:59:49 2015 -0500

      Adding script to run QL files

commit 04936b41c0a93d44cbb51ff365509e55970b60c9
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Mon Nov 16 15:59:49 2015 -0500

      Adding script to run QL files

commit deefda74310f1faa762398c5d394275f18bb90d9
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Mon Nov 16 15:32:47 2015 -0500

      Combining two gitignores into one in root

commit 904abffc22dd87ea18a155f99975643b2ae62d86
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Mon Nov 16 15:28:44 2015 -0500

      Make all the dependencies before running tests and clean after

commit 476b61e63f1e110418e205f69925f8bd77d5db0a
Merge: eb225aa 6c5eea2
Author: Matthew Piccolella <mjp2220@columbia.edu>
Date:   Mon Nov 16 13:18:17 2015 -0700

      Merge pull request #8 from mayankmahajan24/ast_matt

      Hello, world Demo Running

commit 6c5eea2e1efd5928f413bf7ff53db7f33a0f22d9
Author: mjp2220 <mjp2220@columbia.edu>
```

```
Date:   Mon Nov 16 13:58:37 2015 -0500

      Removing unneeded test files

commit 4b38161cf9877bbb497ba93bee526b7ca09f8748
Merge: 94c7c37 411a859
Author: Matthew Piccolella <mjp2220@columbia.edu>
Date:   Mon Nov 16 11:55:46 2015 -0700

      Merge pull request #9 from mayankmahajan24/ast_matt_2

      Making tests compare against output

commit 411a8590b66a7734b313eb70013784b5d18b5537
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Mon Nov 16 13:48:13 2015 -0500

      Removing generated output files, git ignore them for future

commit 3aecaa5f83ecffa58e142f5f704a0d9a3bbb4a0d
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Mon Nov 16 13:45:49 2015 -0500

      Getting rid of the output files

commit 4efa2e428814d7c9c5da47b6d95ffaf7a6c0ae60
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Mon Nov 16 13:44:02 2015 -0500

      Making tests compare against output

commit 94c7c371406daa40a168a9b44f8ea1cc99037609
Author: Evan Tarrh <evantarrh@gmail.com>
Date:   Mon Nov 16 03:04:18 2015 -0500

      successfully running test suite on hello world

commit bebd9a28adde70042e9251fb207b371a536988ee
Author: Mayank Mahajan <mayank.mahajan@gmail.com>
Date:   Mon Nov 16 02:13:28 2015 -0500

      Finished first version of hello world ya bishhhhh
```

```
commit f95593cca912c4e2fd09f1482b9ad52e517e6526
Merge: d77c58f e960782
Author: Mayank Mahajan <mayank.mahajan@gmail.com>
Date:   Mon Nov 16 01:37:08 2015 -0500

      Merge branch 'ast_matt' of https://github.com/mayankmahajan24/QL into
ast_matt

commit d77c58f014dd8b5077178468cca85f1ccb426ded
Author: Mayank Mahajan <mayank.mahajan@gmail.com>
Date:   Mon Nov 16 01:36:51 2015 -0500

      Confirmed stmt list is emptygit status

commit e9607822398a58ba4432350bf85468c24b544920
Author: Gary Lin <garylinrule@gmail.com>
Date:   Mon Nov 16 01:36:40 2015 -0500

      git ignore

commit 91bd8642b8d858815afae7e760e39b00c80744e9
Author: Mayank Mahajan <mayank.mahajan@gmail.com>
Date:   Mon Nov 16 01:30:45 2015 -0500

      Fixed issues with translate

commit 9b150d102802cd11bb658bcc6a9d8bb76770e2bc
Author: Anshul Gupta <anshul.dlf@gmail.com>
Date:   Mon Nov 16 00:53:49 2015 -0500

      continued work

commit 5bac1d5b6538fc650ed39f75f3cacb8420945050
Author: Anshul Gupta <anshul.dlf@gmail.com>
Date:   Sun Nov 15 02:50:14 2015 -0500

      skeleton hello world

commit d591d0f24c644c7dbd72147a9eaf826e65c7e389
Author: Anshul Gupta <anshul.dlf@gmail.com>
Date:   Sun Nov 15 02:27:13 2015 -0500

      structure of compile is in order. implementation to be implemented
```

```
commit df9ff0c00ff301868c130913d7b1804239274608
Author: Anshul Gupta <anshul.dlf@gmail.com>
Date:   Sun Nov 15 02:26:40 2015 -0500

      very basic tests

commit ca69d6384a2f132a02b5d619cd36bf3f84a8165e
Author: Anshul Gupta <anshul.dlf@gmail.com>
Date:   Sun Nov 15 02:26:12 2015 -0500

      getting started on sast.ml to be continued later

commit 45819ab850dd12924eacd6036080b39d0e23b938
Author: Anshul Gupta <anshul.dlf@gmail.com>
Date:   Sun Nov 15 02:25:53 2015 -0500

      updated makefile for hello world

commit 04cdd92f8e974cf4f55a768e162496c735df9fd5
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Thu Nov 12 15:56:38 2015 -0500

      Fixing all the syntax errors in AST, getting a working Makefile

commit 320a58eb11d2ef6f42439d2e51f1b97af15a40a0
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Thu Nov 12 13:49:05 2015 -0500

      Removing merge conflict

commit 0b6b20adbc0f90419ef266531491c94bad36d01f
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Wed Nov 11 18:59:06 2015 -0500

      Commenting this out real quick

commit 8e75a872c16b9a0ba42e5a27fc383b06d029009c
Merge: 425b14e 62fdcdc
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Wed Nov 11 18:58:14 2015 -0500
```

```
      Merge branch 'ast_matt' of https://github.com/mayankmahajan24/QL into
ast_matt

commit 425b14e8f15c852cf931518a49f8d2a0e9ed9984
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Wed Nov 11 18:55:08 2015 -0500

      Wrapping up a basic QL file

commit 1c3c1f6bbd9aa5fd24b2d041506b639414d2802a
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Wed Nov 11 18:45:04 2015 -0500

      Making some more janky changes

commit d21be1f4c163052f6b998d4ecb6c8618e9e9522d
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Wed Nov 11 18:10:17 2015 -0500

      Writing some rough stuff

commit b8d91f0f3938d35438964367980f3a4911427c66
Author: Gary Lin <garylinrule@gmail.com>
Date:   Wed Nov 11 18:44:06 2015 -0500

      2nd iteration of ast

commit 62fdcdccadc1c1fea864f2e073f181fbf6799cee
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Wed Nov 11 18:10:17 2015 -0500

      Writing some rough stuff

commit 9cf6922deb63973bda518375aff96ef24ca120d6
Author: Gary Lin <garylinrule@gmail.com>
Date:   Wed Nov 11 00:17:53 2015 -0500

      completed the first iteration of the AST

commit 7e942c94bb8f1ee48df072e61909b419f460d604
Merge: 0fe27fa b9c7f11
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Mon Nov 9 13:56:16 2015 -0500
```

```
      Merge branch 'ast' of https://github.com/mayankmahajan24/QL into ast

commit 0fe27fa9ee0e5f602a33cb2868d32844fab63d6e
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Mon Nov 9 13:54:55 2015 -0500

      Adding AST, updating Makefile

commit b9c7f1154d3fbf96a3e10d6b9d25373d8dd6bf0c
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Mon Nov 9 13:54:55 2015 -0500

      Adding AST, updating Makefile

commit eb225aac77d7562e0c79e074f3f52707d7fd843c
Merge: 21851b0 1863e96
Author: Matthew Piccolella <mjp2220@columbia.edu>
Date:   Mon Nov 9 13:29:35 2015 -0500

      Merge pull request #6 from mayankmahajan24/parser

      Parser

commit 1863e962b82461884ac7f3a9aa307964588cc8c4
Merge: cdbdc48 fff0394
Author: Matthew Piccolella <mjp2220@columbia.edu>
Date:   Mon Nov 9 12:28:22 2015 -0500

      Merge pull request #5 from mayankmahajan24/adding-tester-to-parser

      First pass at testing script, pending valid scanner

commit cdbdc48e7d694bea3f72f297461b4e73e46d965c
Author: Gary Lin <garylinrule@gmail.com>
Date:   Sun Nov 8 20:03:56 2015 -0500

      fixed shift/reduce conflicts, added makefile to compile scanner and
parser?

commit fff0394bdaa22836b47d44d168b7e39226beaa7f
Author: Evan Tarrh <evantarrh@gmail.com>
Date:   Sun Nov 8 17:21:36 2015 -0500
```

```
      First pass at testing script, pending valid scanner

commit ade0e5449dfed7e3faf028a0fa8e558e141d986c
Merge: 37a13a5 b645fd4
Author: Evan Tarrh <evantarrh@gmail.com>
Date:   Sun Nov 8 16:06:30 2015 -0500

      Merge pull request #4 from mayankmahajan24/matt_parser

      Adding in where changes, fixing boolean expressions

commit b645fd46d8beac2e67d8e5947188e3e1b4c37357
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Sun Nov 8 16:01:30 2015 -0500

      Removing generated files

commit 365151d4ec9f87015e72516cdab1ac755aa1e9ba
Merge: 056b992 5a0dcb4
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Sun Nov 8 16:00:58 2015 -0500

      Merge branch 'matt_parser' of https://github.com/mayankmahajan24/QL
into matt_parser

commit 056b9926eb23e63b38e08f76a3d8332888738bf6
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Sun Nov 8 15:57:21 2015 -0500

      Adding in where changes, fixing boolean expressions

commit 5a0dcb4f6e9fde0b959991dc906cff829875d72d
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Sun Nov 8 15:57:21 2015 -0500

      Adding in where changes, fixing boolean expressions

commit 37a13a5d3e98b07444cc605c43048541dbb584f8
Author: Mayank Mahajan <mayank.mahajan@gmail.com>
Date:   Sun Nov 8 02:09:51 2015 -0500

      Formatting changes
```

```
commit 9d1d7550e61f87b8b22d18adc9a5ad9a35cd99fc
Author: Anshul Gupta <anshul.dlf@gmail.com>
Date:   Sun Nov 8 01:00:52 2015 -0500

      removing menhir errors

commit 1974cb690d8b7080c91c3518b62fbb82014e373c
Author: Mayank Mahajan <mayank.mahajan@gmail.com>
Date:   Sun Nov 8 00:06:38 2015 -0500

      Finished writing parser, didn't test at all lol

commit de4c92f6dcedcb6bc370ad00fee730b369d21790
Author: Mayank Mahajan <mayank.mahajan@gmail.com>
Date:   Sat Nov 7 23:38:39 2015 -0500

      Changed brackets in parser

commit 629ac154f052a136f8a3e363e317e880464d7dec
Author: Mayank Mahajan <mayank.mahajan@gmail.com>
Date:   Sat Nov 7 23:36:06 2015 -0500

      Renamed all bracketsgit branch

commit 10987c85413d85ec09d3af70bb8a6c8d80c02627
Author: Anshul Gupta <anshul.dlf@gmail.com>
Date:   Sat Nov 7 23:03:42 2015 -0500

      to revisit data types

commit be277b6fc7cf57846fb056d2013ef72a90285cb9
Author: Mayank Mahajan <mayank.mahajan@gmail.com>
Date:   Sat Nov 7 22:53:06 2015 -0500

      Started rhs

commit 71e15abebb0ef00ebb708aee28730ba8163b7132
Author: Mayank Mahajan <mayank.mahajan@gmail.com>
Date:   Sat Nov 7 22:23:11 2015 -0500

      Fixed merge conflict1
```

```
commit 7d489c8dbaa8ce4856d4b85516ee153c653d59c3
Author: Anshul Gupta <anshul.dlf@gmail.com>
Date:   Sat Nov 7 22:21:40 2015 -0500

      splendid saturday night

commit 81021ef104af6b73252b9f57a5a01221748d802f
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Fri Nov 6 12:07:51 2015 -0500

      Finishing up if/else/elseif statements

commit 6542c61f908cf4e35a8c7a048732616be4795f9d
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Fri Nov 6 11:46:08 2015 -0500

      Adding rules for the where statement, among other things

commit 8093ad5bf3f7fb7360edc47a7d408e6f27b50c64
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Fri Nov 6 11:20:08 2015 -0500

      Starting to write the parser

commit 21851b02f0f8d18bd421ff6227e68759dceae51a
Merge: 268805a 4a829d0
Author: Evan Tarrh <evantarrh@gmail.com>
Date:   Wed Nov 4 18:13:50 2015 -0500

      Merge pull request #3 from mayankmahajan24/scanner_parser

      Scanner parser

commit 4a829d0274167ec5a931e4f58bbb20abf08f3b7b
Author: Gary Lin <garylinrule@gmail.com>
Date:   Wed Nov 4 18:10:43 2015 -0500

      changes to flt and num to allow for unary -

commit 268805a9409fc0c00d0420661c829dbd284677e4
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Wed Nov 4 16:56:43 2015 -0500
```

```
      Adding the PDFs for LRM and proposal

commit f09bc16cc61fa0afa0e07d6b6cf21237f4e6ba7d
Merge: 902ab91 71283a7
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Wed Nov 4 16:48:15 2015 -0500

      Merge branch 'language_reference' of
https://github.com/mayankmahajan24/QL

commit 902ab91180c5d95f9221544e581d3343e0232218
Merge: 6576998 d92641a
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Wed Nov 4 16:46:19 2015 -0500

      Merge branch 'language_reference'

commit d92641a03ffd948a27336d87ca118ee8f319cb67
Merge: 2afa8d2 6576998
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Wed Nov 4 16:45:41 2015 -0500

      Merge branch 'master' of https://github.com/mayankmahajan24/QL into
language_reference

commit 71283a7555303a92916360475a758f811a5c0709
Author: Mayank Mahajan <mayank.mahajan@gmail.com>
Date:   Mon Oct 26 00:58:44 2015 -0400

      added concatenation section

commit 100e3ca7bf36135c84bffae7ad1ca75eeb856680
Author: Mayank Mahajan <mayank.mahajan@gmail.com>
Date:   Mon Oct 26 00:49:28 2015 -0400

      Removed duplicate function call sections

commit 29773390bb56f32ee9a2ecf41d7509adb3b6153c
Author: Mayank Mahajan <mayank.mahajan@gmail.com>
Date:   Mon Oct 26 00:45:29 2015 -0400

      Removed some code snippet
```

```
commit ebd57836ead2edfbbd2d5354c3ae6c42de66d5d2
Author: Mayank Mahajan <mayank.mahajan@gmail.com>
Date:   Mon Oct 26 00:42:33 2015 -0400

      Changed double to float

commit 7329305c671a392fa3698a73b0f3dbf4aa866eee
Author: Gary Lin <garylinrule@gmail.com>
Date:   Mon Oct 26 00:20:08 2015 -0400

      gary's revision

commit d9dcfa4303973dbcb69f3c86b8fe1fc16a2be707
Merge: cea287b 6576998
Author: Gary Lin <garylinrule@gmail.com>
Date:   Mon Oct 26 00:05:18 2015 -0400

      Merge branch 'master' of https://github.com/mayankmahajan24/QL into
scanner_parser

commit cea287b1ff78de048e2e269b2b5bc7d3f8fa5faa
Author: Gary Lin <garylinrule@gmail.com>
Date:   Mon Oct 26 00:04:36 2015 -0400

      added logical operators

commit b3d921e66c02a0560f6fd2c5b80312dd9bd486c6
Author: Anshul Gupta <anshul.dlf@gmail.com>
Date:   Sun Oct 25 19:24:04 2015 -0400

      typo introduction

commit 65769989ce1a4faa93b19f14c3c87bde6dd82d75
Author: Mayank Mahajan <mayank.mahajan@gmail.com>
Date:   Sun Oct 25 16:41:59 2015 -0400

      Removed merge conflict

commit 2afa8d2f7381159cbc0147d04307c2713a667f3b
Merge: 029a023 34054bb
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Sun Oct 25 15:39:26 2015 -0400
```

```
      Merge branch 'language_reference' of
https://github.com/mayankmahajan24/QL into language_reference

commit 029a023a001dab9c4f3325caa283a655ee8a9b35
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Sun Oct 25 15:39:20 2015 -0400

      Adding colon description

commit 34054bbb91a935ae29edbbbac671b23bc47c6016
Author: Evan Tarrh <evantarrh@gmail.com>
Date:   Sun Oct 25 15:37:32 2015 -0400

      adding print to stlib

commit 55d1b62396944506d0ae0061265b0485b26b0b90
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Sun Oct 25 15:25:41 2015 -0400

      Getting GitPrint to work

commit d2cc414b7d95660665b910a919c0dcc5e3431429
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Sun Oct 25 15:21:08 2015 -0400

      dummy

commit 21496dc8869ca6a559e07ae348357d2ad1875ce4
Merge: 3af821f 71f8339
Author: Evan Tarrh <evantarrh@gmail.com>
Date:   Sun Oct 25 15:15:19 2015 -0400

      Merge branch 'language_reference' of
https://github.com/mayankmahajan24/QL into language_reference

commit 3af821f08bd488b4dcfd29ff506fa36189a4cfec
Merge: 0031cc2 9bb7b36
Author: Evan Tarrh <evantarrh@gmail.com>
Date:   Sun Oct 25 15:15:12 2015 -0400

      fixing quoootes

commit 71f8339934abdb5780f35ec17bd9ed5b14087074
```

```
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Sun Oct 25 15:14:12 2015 -0400

      Last stupid thing

commit 0031cc2a2ec1d4234ede02c080d050e8108c3377
Author: Evan Tarrh <evantarrh@gmail.com>
Date:   Sun Oct 25 15:12:58 2015 -0400

      adding the rest of section 4, general cleanup

commit 9bb7b36c30f463a528154a5b4a4c9e124a2b3157
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Sun Oct 25 15:12:54 2015 -0400

      Fixing the shitty quotes

commit 5b58e769ac756dd59637543fa226b80d39235b41
Merge: 83f1b30 ccdb30e
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Sun Oct 25 14:59:12 2015 -0400

      Merge branch 'language_reference' of
https://github.com/mayankmahajan24/QL into language_reference

commit 83f1b304a08d0919c292dd6634f74b58a93cc367
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Sun Oct 25 14:58:58 2015 -0400

      Finishing loop statements

commit ccdb30e76e32fce8c280de37902a69f564ad1f60
Merge: 9b53682 4e18c24
Author: Evan Tarrh <evantarrh@gmail.com>
Date:   Sun Oct 25 14:53:43 2015 -0400

      Merge branch 'language_reference' of
https://github.com/mayankmahajan24/QL into language_reference

commit 9b5368277c9e32da34ea5210ec69ccd481cb73df
Author: Evan Tarrh <evantarrh@gmail.com>
Date:   Sun Oct 25 14:53:34 2015 -0400
```

```
      adding matt's work back in (sweat smile emoji)

commit 4e18c24d049ef7a590f3e059a44be8a1e979b31d
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Sun Oct 25 14:49:27 2015 -0400

      Updating some of this nonsense

commit 34377a026434c2d76ea0ced0834e53be8a39b58f
Author: Gary Lin <garylinrule@gmail.com>
Date:   Sun Oct 25 14:45:55 2015 -0400

      makefile to test scanner.mll

commit 2e6124d3472375d99a3e090393b1e1153e3ae189
Merge: a8da647 781a015
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Sun Oct 25 14:42:04 2015 -0400

      Merge branch 'language_reference' of
https://github.com/mayankmahajan24/QL into language_reference

commit a8da6472bcaf186efa79f2ed592b0e1e57f3f29a
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Sun Oct 25 14:41:56 2015 -0400

      Finishing up statements

commit 8dd0d06c648bc78bff594f2cafa3f85a83c6ca0e
Author: Gary Lin <garylinrule@gmail.com>
Date:   Sun Oct 25 14:36:37 2015 -0400

      1st iteration of scanner

commit 781a01539526a66856b9616dc54c4520df2d8353
Merge: c255562 4a1e15f
Author: Evan Tarrh <evantarrh@gmail.com>
Date:   Sun Oct 25 14:34:52 2015 -0400

      Merge branch 'language_reference' of
https://github.com/mayankmahajan24/QL into language_reference

commit c255562fa0653dff86f95c2a81374a6280fec7a5
```

```
Author: Evan Tarrh <evantarrh@gmail.com>
Date:   Sun Oct 25 14:34:45 2015 -0400

      cleaning up operators

commit 35746c640cc48f9faf4cf67056a16239a9484a58
Merge: ffad2f4 4a1e15f
Author: Mayank Mahajan <mayank.mahajan@gmail.com>
Date:   Sun Oct 25 14:33:41 2015 -0400

      Merge branch 'language_reference' of
https://github.com/mayankmahajan24/QL

commit 4a1e15f0ec64b52152d0b60bd49ef0eb1d46ad90
Merge: 5969f0b d72b212
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Sun Oct 25 14:33:04 2015 -0400

      Merge branch 'language_reference' of
https://github.com/mayankmahajan24/QL into language_reference

commit 5969f0b108a25ea8551c6c0ac7783e6d0a6dead0
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Sun Oct 25 14:32:41 2015 -0400

      Fleshing out statements

commit d72b212f007759bc40ed0502557379d43f2101eb
Merge: f9878ac c5e9778
Author: Evan Tarrh <evantarrh@gmail.com>
Date:   Sun Oct 25 14:19:18 2015 -0400

      fixing merge conflicts

commit f9878acd1d21b096342b291e422bb94d5686b3e1
Author: Evan Tarrh <evantarrh@gmail.com>
Date:   Sun Oct 25 14:18:07 2015 -0400

      adding std lib functions & operators (incomplete) to markdown

commit c5e97789ea88d74d94cfbd796a23b0c6c7b0396d
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Sun Oct 25 14:13:26 2015 -0400
```

```
      Adding beginning of 4

commit bd0b8207163886583374798edf2a85ca4f5e386c
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Sun Oct 25 13:55:24 2015 -0400

      Adding lexical conventions

commit 71771b1f3aacd989f8011597a14d032ab039dd83
Author: Gary Lin <garylinrule@gmail.com>
Date:   Sun Oct 25 13:47:22 2015 -0400

      additions to scanner

commit 6580eb181a7ed7a2596d7216c14251e4af962d72
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Sun Oct 25 13:43:22 2015 -0400

      Aligning everything to the left

commit 793b74a274aea7d9d0d43c5673211bbf642bf276
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Sun Oct 25 13:27:40 2015 -0400

      Trying a fix

commit 128996f0a612e58d77f2da2d009e92988ff1c38c
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Sun Oct 25 13:23:06 2015 -0400

      Adding data types'

commit ecb05e7356df21b3ec34743a8565a2f6487f393b
Merge: a759364 413f8a3
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Sat Oct 24 19:27:08 2015 -0400

      Merge branch 'language_reference' of
https://github.com/mayankmahajan24/QL into language_reference

commit a759364b2411e460e00ecc08ed34850d178a1e17
Author: mjp2220 <mjp2220@columbia.edu>
```

```
Date:   Sat Oct 24 19:24:04 2015 -0400

      Adding the skeleton of the markdown LRM

commit 413f8a387acf6d5e8ae00602b176dabe3b0053ef
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Sat Oct 24 19:24:04 2015 -0400

      Adding the skeleton of the markdown LRM

commit 7ba4dd21f759d3f54693eada4c94efc16290c29f
Author: Gary Lin <garylinrule@gmail.com>
Date:   Sat Oct 24 16:46:55 2015 -0400

      adding branch

commit ffad2f4b99e729b6d13e376504e67f4d5b2c3e00
Merge: 5770f49 52c6ec6
Author: Evan Tarrh <evantarrh@gmail.com>
Date:   Sat Oct 24 16:43:42 2015 -0400

      Merge pull request #1 from mayankmahajan24/directory_structure

    Adding directory structure with blank files

commit 52c6ec6027959c7bb7a52dce83f8565b510780d0
Author: mjp2220 <mjp2220@columbia.edu>
Date:   Sat Oct 24 16:41:38 2015 -0400

      Adding directory structure with blank files

commit 5770f49add2c340c89efffcc6b8487a6362199c8
Author: Mayank Mahajan <mayank.mahajan@gmail.com>
Date:   Wed Oct 21 16:52:45 2015 -0400
      Initial commit
```

Section 5: Architectural Design

The compiler can be described as a pipeline of working parts, which translate an input file `*.ql` into a corresponding `.java` file. Assuming that the user has Java installed on their machine, this file can be compiled using the `ql` command. For example, to compile a `.ql` file called `test.ql` and create an output Java file `output.java`, the command to be run from the main directory would be `compiler/qlc test.ql output` The resulting file, `output.java`, can then be executed with the `ql` command, which will compile the Java file and execute it with the proper classpath that contains a JSON package for Java. For this example, the command would be `compiler/ql output`.

Below is a block diagram that outlines each of the major components of the translator:

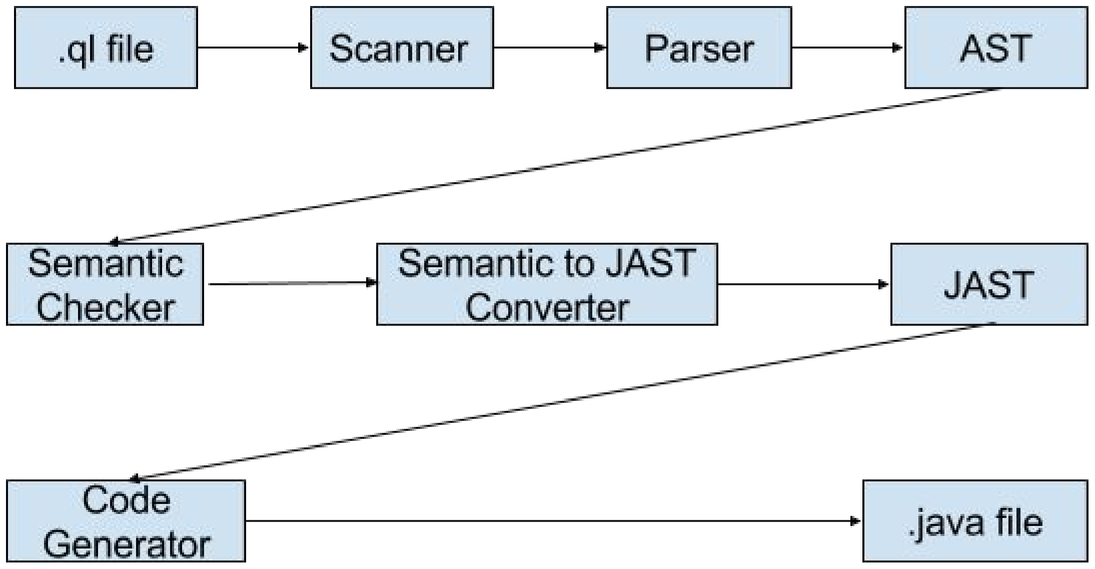

Figure 1. Structure of the QL Compiler

The walkthrough of the compiler as is follows: A `.ql` file is passed to the scanner, where superfluous characters like whitespace are removed and the rest of the file is split into tokens and passed to the parser. The parser contains a set of rules that dictate the possible derivations for each intermediate node in the parse tree, and at the lowest level, the parameters are passed to a function that will be implemented later in the semantic checker. The AST contains all of the types of values that might result from the parser, and defines the signatures for each of the functions called in the parser. The resulting AST is then passed to the semantic checker, which processes the program top down and constructs a symbol table containing information about JSON selector types, variable names and types, and function names and arguments. The checker makes sure that all of the types are consistent throughout the function and accurate, and throws exceptions if there is a type mismatch somewhere in the code. This step is also where type inferencing is performed with JSONs; as JSON selectors are used in the program, their types are automatically inferred from the line of code they belong to. As the rest of the program is checked, that JSON selector is assumed to be of the type that was inferred before, and if this leads to a type mismatch, the semantic checker throws an error. After checking the semantics of the code, the AST + Symbol Table are passed to the Semantic to JAST converter, which receives an AST node and adds the appropriate Java-specific information to output a corresponding Java AST (JAST) node. After creating the JAST, which has all of the Java types and functions, similar to the AST, the compiler passes the JAST to the code generator, which takes each statement in the JAST and outputs the respective line of Java code that expresses the same meaning. These lines are appended to a long string which then represents the program code, and after processing all of the JAST statements, the string is written to a `.java` file. As mentioned previously, that file can now be compiled and executed with `./ql`

Although we did have certain roles assigned to us, rarely did just one person complete an entire section of the compiler by themselves. Below is a rough distribution of the components among members of the team.

Gary:

- Scanner (constructed scanner)
- Parser (skeleton, fixed the shift/reduce and reduce/reduce errors)
- AST/Semantic Checker/Semantic to JAST/Code Generator/Tests (Array Assignment, Booleans, Function declaration, Function calls, Where clause)

Anshul:

- Parser (Array initialization)

- AST/Semantic Checker/Semantic to JAST/Code Generator/Tests (Array Initialization, For/While, Scoping Rules for variable declarations within block statements, If/else constructs)
- Final report text

Matt:

- Parser/Architectures for and large portions of Semantic Checking and Code Generation/JSON auto-typing/JSON code generation

Mayank:

- Parser (Array assignment, Boolean Expressions, Brackets)
- AST/Semantic Checker/Semantic to JAST/Code Generator/Tests (Array Assignment, Booleans, If/else, Function declaration, Function calls, Where clause)

Evan:

- AST/Semantic Checker/Semantic to JAST/Code Generator/Tests (Arrays, Functions, Where clause, Integration tests)
- Demos for Presentation

Section 6: Test Plan

Example Programs

Example 1

springcourses.ql

This program is a great example of how to get quick and meaningful results from a dataset with QL. It's easy to iterate through only certain elements of JSON files, and it highlights the usefulness of treating JSONs as first-class objects.

```
#~~
Prints all COMS courses offered next spring.

Requires coursedata.json with at least the following fields:
{"Term":20161,"DepartmentCode":"COMS","CourseTitle":"ANALYSIS OF
ALGORITHMS I"},
~~#

json courses = json("coursedata.json")

where (course["Term"] == 20161) as course {
  if (course["DepartmentCode"] == "COMS") {
    string courseTitle = course["CourseTitle"]
    print(courseTitle)
  }
} in courses["courses"]
```

Here's what it compiles to (with whitespace cleaned up):

```
import java.io.FileReader;
import java.util.Iterator;
import org.json.simple.JSONArray;
import org.json.simple.JSONObject;
import org.json.simple.parser.JSONParser;
public class SpringCourses {
  public static JSONArray staticArrr;
  public static Iterator staticItt;
  static JSONObject courses;

  public static void main(String[] args) {
    try {
      courses = (JSONObject) (new JSONParser()).parse(new
FileReader("coursedata.json"));
      staticArrr = (JSONArray)(JSONArray)courses.get("courses");
      staticItt = staticArrr.iterator();

      while(staticItt.hasNext()){
        JSONObject course = (JSONObject) staticItt.next();

        if(((Long) course.get("Term")).intValue() == 20161) {
          if (((String)course.get("DepartmentCode")).equals("COMS"))
{
            String courseTitle = (String)course.get("CourseTitle");
            System.out.println(courseTitle);
            ;
          } else {
          }
        }
      }
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

Example 2

`citibikespeed.ql`

This program shows off some of QL's more powerful features. There are functions ranging from simple to complicated, and they are used in several meaningful contexts: inside of `where` loops, inline as part of a binary operation, inside of other functions. We are able to pass data from JSONs directly into those functions (e.g. convertSecondsToHours(ride["tripduration"])), without any typing problems. This keeps the loop where we iterate through the data as simple as possible, and we are able to analyze thousands of datapoints to synthesize a meaningful result.

```
#~~
Prints average speed of CitiBikers in MPH.

Requires bikedata.json with at least the following fields:
{"tripduration":1110,"start station latitude":40.74025878,"start
station longitude":-73.98409214,"end station
latitude":40.71893904,"end station longitude":-73.99266288},
~~#

json bikeData = json("bikedata.json")
float totalHours = 0.0
float totalMiles = 0.0

function convertSecondsToHours(float seconds) : float {
  return seconds / 3600.0
}

function sqrt(float f) : float {
  float epsilon = 0.0000001
  float t = f
  float diff = 100000000.0
  while (diff > epsilon * t) {
    float tmp = f / t
    t = tmp + t
    t = t / 2.0
    diff = f / t
    diff = t - diff
  }
  return t
}

function computeDistance(float startLat, float startLong, float
endLat, float endLong) : float {
  #~~ Each degree of latitude is approximately 69 miles apart ~~#
  float latDiff = endLat - startLat
  latDiff = 69.0 * latDiff

  #~~ At NYC's latitude, each degree of longitude is approximately
53 miles apart ~~#
  float longDiff = endLong - startLong
  longDiff = 53.0 * longDiff
  return sqrt(latDiff * latDiff + longDiff * longDiff)
}

where (True) as ride {
  float rideMiles = computeDistance(ride["start station latitude"],
ride["start station longitude"], ride["end station latitude"],
ride["end station longitude"])

  if (rideMiles > 0.0) {
    totalMiles = totalMiles + rideMiles
    totalHours = totalHours +
convertSecondsToHours(ride["tripduration"])
  }
} in bikeData["rides"]

print("The average speed of CitiBikers (in MPH):")
print(totalMiles / totalHours)
```

Here's what it compiles to (again, with whitespace cleaned up):

```
import java.io.FileReader;
import java.util.Iterator;
import org.json.simple.JSONArray;
import org.json.simple.JSONObject;
import org.json.simple.parser.JSONParser;
public class test {
  public static JSONArray staticArrr;
  public static Iterator staticItt;
  static JSONObject bikeData;
  static double totalHours;
  static double totalMiles;

  public static void main(String[] args) {
      try {
        bikeData = (JSONObject) (new JSONParser()).parse(new
FileReader("bikedata.json"));
        totalHours = 0.;
        totalMiles = 0.;

        staticArrr = (JSONArray)(JSONArray)bikeData.get("rides");
        staticItt = staticArrr.iterator();
        while(staticItt.hasNext()){
          JSONObject ride = (JSONObject) staticItt.next();

          if(true) {
            double rideMiles = computeDistance(
              ((Number) ride.get("start station
latitude")).doubleValue(),
              ((Number) ride.get("start station
longitude")).doubleValue(),
              ((Number) ride.get("end station
latitude")).doubleValue(),
              ((Number) ride.get("end station
longitude")).doubleValue());

            if (rideMiles > 0.) {
              totalMiles = totalMiles + rideMiles;
              totalHours = totalHours +
convertSecondsToHours(((Number)
ride.get("tripduration")).doubleValue());
            } else {
            }
          }
        }
      System.out.println("The average speed of CitiBikers (in
MPH):");
      System.out.println(totalMiles / totalHours);
    } catch (Exception e) {
      e.printStackTrace();
    }
  }

  public static double sqrt(double f){
    double epsilon = 1e-07;
    double t = f;
    double diff = 100000000.;

    while (diff > epsilon * t) {
      double tmp = f / t;
      t = tmp + t;
      t = t / 2.;
      diff = f / t;
      diff = t - diff;
    }
    return t;
  }

  public static double computeDistance(double startLat,
    double startLong, double endLat, double endLong) {

    double latDiff = endLat - startLat;
    latDiff = 69. * latDiff;
    double longDiff = endLong - startLong;

    longDiff = 53. * longDiff;
    return sqrt(latDiff * latDiff + longDiff * longDiff);
  }

  public static double convertSecondsToHours(double seconds){
    return seconds / 3600.;
  }
}
```

Test Suite

Feature	**No. of tests**
Arrays	9
Assignment	9
Booleans	4
For Loops	6
Functions	15
“Hello, world!”	2
If/Else	9
JSON	22
Where loops	4

While loops	4
Integration tests	5
Total tests	**90**

We used a modified version of Professor Edwards' script to run the test suite. We added functionality for tests that are meant to fail—although, truth be told, those were rarely as informative in solving problems as our positive tests. We also included a similar, but separate, script called run_integration_tests.sh that runs our integration tests. We separated the integration tests because they require large JSON files that we didn't want to include in our core repository, so if someone were to simply git clone our project and run run_tests.sh, all tests will pass as expected. Both scripts can be found with the rest of our source code.

Although one person in our group was given the role of "tester", it should be noted that every person contributed massively to our test suite. Our attitude was that any feature that was ever implemented should be accompanied by at least one test, and every bug that was fixed should also be accompanied by a test to make sure the bug would not appear again.

Generally, we followed the principle of test-driven development: we wrote our tests first, and then kept working until they all passed. This had an impact that was directly correlated to the quality of those tests: as long as we wrote tests effectively, we got excellent results. One noteworthy downside to our test-driven approach is that many of the tests we wrote in the early stages of development became trivial as time passed, so that many features were tested duplicatively. For example, though our assignment tests were helpful early on, they ended up being almost useless once all of our other tests involved assignments.

Finally, our integration tests were designed after we felt we had implemented a working version of our language. These tests involved more creative ways to use our language, and there are the longest QL programs to date. They were especially rewarding to work on, as we got to see our language do meaningful things with real data, and they had the bonus effect of catching previously unforeseen problems.

Section 7: Lessons Learned

Matt

Something I learned was that sometimes you have to have difficult conversations with members of your team to keep the project on track. It was tough at times to balance the different workloads on our team; some members had 4 classes, while others had 7 classes. It felt that work defaulted to people who had less work with other courses, and I learned that it's important to set expectations for the members of the team for how much work is expected of them. Also, I learned that a committed group is good at calling other group members out for lagging, which kept our entire project on track.

Evan

One big takeaway for me was that it's important to take initiative instead of waiting for things to be delegated. I found that the moments where I was contributing the best happened when I decided to step up and accomplish something that needed to be done, and it helped ensure that nothing slipped through the cracks, even though we were a really well-managed group. Also, I found that I learned the most when pair programming—it was a great way to get used to working with one another, and helped us get up to speed much faster than would have otherwise been possible.

Anshul

Somehow (much against our own expectations) we did a great job dividing work throughout the semester and that helped reduce the stress for us. We were pretty much done with a fully working compiler a week before the deadline so we could work on the Final Report at our own pace. One thing I think we did well was use Github Issues effectively: each time we came across a serious bug, we opened a new issue for it and either self-assigned it or somebody else picked it up. This allowed us to fully dedicate our focus to the task we were working on.

Mayank

I learned a lot about what it is to work in a team, and actually I'm (secretly) disappointed we didn't have very many problems with work distribution throughout the semester, which I believe is almost completely due to the fact that we are all smart, hard-working, and curious individuals who all want a part in creating the language. Besides learning what pull requests are actually used for in Git, I learned that it's important to stand up for yourself in a group setting and set realistic goals rather than trying to achieve a lofty and idealistic milestone. We set fair deadlines for ourselves, and in exchange we worked hard to not miss a single one, and that ended up allowing us to finish much earlier than a bunch of other teams that are still writing code two days before the project is due. Lastly, pair programming is an amazing thing and really makes the process less stressful when two people are working on a task together since they can bounce ideas off each other and avoid getting stuck.

Gary

There were two main takeaways for me. The first is that paired programming is awesome. Prior to this project, I haven't really done formal paired programming and didn't realize how useful it is, especially with more difficult tasks in a less familiar language (OCaml). Having two heads collaborating to solve a problem is seriously much more effective than just banging your head on the keyboard by yourself. The second is that having a good team dynamic for a very involved project is important. Although we all had specific titles, our actual job roles were very fluid. Everyone played a role and touched every part of the pipeline, which I think speaks to the teamwork of our group. There were a lot of other groups with teammates that had siloed roles and very unevenly divided work. Fortunately, the work division amongst our team was fairly split across all pipelines, which helped to maintain a healthy team dynamic and allowed everyone to understand the entire workflow of our compiler.

Advice for Future Teams

- Start early
- Keep the momentum going throughout the semester. Lots of teams finish their scanner and parser super fast but then slow down when the real nitty gritty stuff piles on.
- Rank your features from most important to least important, and if a feature of less importance begins to take too much time, re-evaluate whether you should have it in the language.
- Don’t be afraid to do a lot of pair programming. #bettertogether
- Write tests extensively, creatively, and constantly.
- Keep a constant backlog of bugs and TODOs. Make sure every group member always knows the state of what is working and what isn’t.

Section 8: Appendix

Full Code

/

```
run_unit_tests.sh/
#
# QL
#
# Manager: Matthew Piccolella
# Systems Architect: Anshul Gupta
# Tester: Evan Tarrh
# Language Guru: Gary Lin
# Systems Integrator: Mayank Mahajan

#!/bin/sh

# Adapted from Prof. Edwards' test script for MicroC.

QL="compiler/qlc"
PASS=0
FAIL=0
# Time limit for operations
ulimit -t 100

globalerror=0
globallog=run_tests.log
rm -f $globallog
error=0

make all &> /dev/null

SignalError() {
  if [ $error -eq 0 ] ; then
   echo "FAILED"
    error=1
  fi
 echo "  $1"
}

# Compare <outfile> <reffile> <difffile>
# Compares the outfile with reffile.  Differences, if any, written to difffile
Compare() {
  generatedfiles="$generatedfiles $3"
  echo diff -b $1 $2 ">" $3 1>&2
  diff -b "$1" "$2" > "$3" 2>&1 || {
    SignalError "$1 differs"
    echo "FAILED $1 differs from $2" 1>&2
  }
}

RunPass() {
  echo $* 1>&2
  eval $* || {
    SignalError "$1 failed on $*"
    return 1
  }
}

RunFail() {
  echo $* 1>&2
  eval $* || {
    error=1
  }
}

Check() {
  error=0

  # strip ".ql" off filename
  basename=`echo $1 | sed 's/.ql//'`

  echo "$basename..."

  echo 1>&2
  echo "###### Testing $basename" 1>&2

  if [[ "$basename" =~ .*-fail.* ]]; then
   RunFail $QL $1 "Test"

    if [ $error -eq 0 ] ; then
     echo "FAIL: This test should not have passed"
      let FAIL+=1
    else
     echo "OK"
      let PASS+=1
      rm ${basename}-gen.out
    fi

 else
   RunPass $QL $1 "Test"  &&
    touch ${basename}-gen.out &&
    javac -classpath build/json-simple-1.1.1.jar:. Test.java &&
    java -classpath build/json-simple-1.1.1.jar:. Test > ${basename}-gen.out &&
    generatedfiles="$generatedfiles ${basename}-gen.out"
    Compare ${basename}-gen.out ${basename}-exp.out ${basename}.i.diff

    if [ $error -eq 0 ] ; then
     echo "OK"
      echo "###### SUCCESS" 1>&2
      rm ${basename}-gen.out
      rm ${basename}.i.diff
      let PASS+=1
    else
     echo "###### FAILED" 1>&2
      let FAIL+=1
      globalerror=$error
    fi
 fi

 rm Test.java
}

shift `expr $OPTIND - 1`

if [ $# -ge 1 ]
then
 files=$@
else
 files="tests/test-*.ql"
fi

make all >& /dev/null

for file in $files
do
 Check $file 2>> $globallog
done

echo "Tests passed: $PASS. Tests failed: $FAIL."

exit $globalerror
```

run_integration_tests.sh/

```
#
# QL
#
# Manager: Matthew Piccolella
# Systems Architect: Anshul Gupta
# Tester: Evan Tarrh
# Language Guru: Gary Lin
# Systems Integrator: Mayank Mahajan

#!/bin/sh

# Adapted from Prof. Edwards' test script for MicroC.

QL="compiler/qlc"
PASS=0
FAIL=0
# Time limit for operations
ulimit -t 100

globalerror=0
globallog=run_tests.log
rm -f $globallog
error=0

make all &> /dev/null

SignalError() {
  if [ $error -eq 0 ] ; then
   echo "FAILED"
     error=1
  fi
 echo "  $1"
}

# Compare <outfile> <reffile> <difffile>
# Compares the outfile with reffile.  Differences, if any, written to difffile
Compare() {
  generatedfiles="$generatedfiles $3"
  echo diff -b $1 $2 ">" $3 1>&2
  diff -b "$1" "$2" > "$3" 2>&1 || {
    SignalError "$1 differs"
    echo "FAILED $1 differs from $2" 1>&2
  }
}

Run() {
  echo $* 1>&2
  eval $* || {
    SignalError "$1 failed on $*"
    return 1
  }
}

```

```
Check() {
  error=0

  # strip ".ql" off filename
  basename=`echo $1 | sed 's/.ql//'`

  echo "$basename..."

  echo 1>&2
  echo "###### Testing $basename" 1>&2

  Run $QL $1 "Test"  &&
  touch ${basename}-gen.out &&
  javac -classpath build/json-simple-1.1.1.jar:. Test.java &&
  java -classpath build/json-simple-1.1.1.jar:. Test > ${basename}-gen.out &&
  generatedfiles="$generatedfiles ${basename}-gen.out"
  Compare ${basename}-gen.out ${basename}-exp.out ${basename}.i.diff

  if [ $error -eq 0 ] ; then
   echo "OK"
    echo "###### SUCCESS" 1>&2
    rm ${basename}-gen.out
    rm ${basename}.i.diff
    let PASS+=1
  else
   echo "###### FAILED" 1>&2
    let FAIL+=1
    globalerror=$error
  fi

 rm Test.java
}

shift `expr $OPTIND - 1`

if [ $# -ge 1 ]
then
 files=$@
else
 files="tests/integration-*.ql"
fi

make all >& /dev/null

for file in $files
do
 Check $file 2>> $globallog
done
```

```

echo "Tests passed: $PASS. Tests failed: $FAIL."
if [ ! $FAIL -eq 0 ]
then
 echo "Looks like some tests failed. Make sure you have the necessary JSON files added."
fi

exit $globalerror
```

Makefile/

```
#
# QL
#
# Manager: Matthew Piccolella
# Systems Architect: Anshul Gupta
# Tester: Evan Tarrh
# Language Guru: Gary Lin
# Systems Integrator: Mayank Mahajan

all:
    cd compiler; make clean; make

clean:
    cd compiler; make clean
```

compiler/

compiler/scanner.ml

```
(*
 * QL
 *
 * Manager: Matthew Piccolella
 * Systems Architect: Anshul Gupta
 * Tester: Evan Tarrh
 * Language Guru: Gary Lin
 * Systems Integrator: Mayank Mahajan
 *)

{ open Parser }

let num = ['0'-'9']+
let exp = 'e' ('+'|'-')? ['0'-'9']+
let flt = '.'['0'-'9']+ exp?
    | ['0'-'9']+ ( ('.' ['0'-'9']* exp?) | exp)

```

```
let boolean = "True" | "False"

rule token = parse
(* Whitespace *)
| [' ' '\t' '\r']  { token lexbuf }
| '\n'+             { ENDLINE }

(* Comments *)
| "#~~"            { comment lexbuf }

(* Punctuation *)
| '('          { LPAREN }     | ')'          { RPAREN }
| '{'          { LCURLY }     | '}'          { RCURLY }
| ';'          { SEMICOLON } | ','          { COMMA }
| '['          { LSQUARE }    | ']'         { RSQUARE }

(* Math Operators *)
| '+'          { PLUS }       | '-'          { MINUS }
| '*'          { TIMES }      | '/'          { DIVIDE }
| '='          { ASSIGN }

(* Equivalency Operators *)
| "!="         { NEQ }        | "=="         { EQ }
| "<="         { LEQ }        | '<'          { LT }
| ">="         { GEQ }        | '>'          { GT }
| "not"        { NOT }

(* Logical Operators *)
| "&"         { AND }        | "|"         { OR }

(* Conditional Keywords *)
| "if"         { IF }         | "else"       { ELSE }


(* Loop Keywords *)
| "where"      { WHERE }      | "in"         { IN }
| "as"         { AS }         | "for"        { FOR }
| "while"      { WHILE }

(* Function Keywords *)
| "function"   { FUNCTION }  | "return"     { RETURN }
| ':'          { COLON }

(* Type Keywords *)
| "int"        { INT }        | "float"      { FLOAT }
| "void"       { VOID }       | "string"     { STRING }
| "json"       { JSON }       | "array"      { ARRAY }
| "bool"       { BOOL }
```

```

(* end of file *)
| eof { EOF }

(* Identifiers *)
| ['a'-'z']['a'-'z' 'A'-'Z' '0'-'9']* as var { ID(var) }

(* Literals *)
| num as intlit      { INT_LITERAL(int_of_string intlit) }
| flt as fltlit      { FLOAT_LITERAL(float_of_string fltlit) }
| boolean as boollit { BOOL_LITERAL(boollit) }
| '"' ([^'"']* as strlit) '"' { STRING_LITERAL(strlit) }
| '\'' ([^'\'']* as strlit) '\'' {STRING_LITERAL(strlit)}

(* Comment Eater *)
and comment = parse
| "~~#" { token lexbuf }      (* End of comments *)
| _    { comment lexbuf }    (* Still a comment *)
```

compiler/parser.mly

```
//////////////////////////////////////////
////// QL: Parser ///////////////////////
////// Manager: Matthew Piccolella ///////
////// Systems Architect: Anshul Gupta ///
////// Tester: Evan Tarrh ///////////////
////// Language Guru: Gary Lin //////////
////// Systems Integrator: Mayank Mahajan/
//////////////////////////////////////////

%{ open Ast %}

////////////////////////////////////////////////
/////////////////// TOKENS ///////////////////
////////////////////////////////////////////////

/* Functions */
%token FUNCTION RETURN COLON

/* Loops */
%token WHERE IN AS FOR WHILE

/* Conditionals */
%token IF ELSE

/* Math Operators */
%token PLUS MINUS TIMES DIVIDE

```

```
/* Equivalency Operators */
%token NEQ, EQ, LEQ, GEQ, LT, GT, NOT

/* Punctuation */
%token LPAREN RPAREN LCURLY RCURLY LSQUARE RSQUARE SEMICOLON COMMA EOF ENDLINE

/* Types */
%token INT FLOAT VOID STRING JSON ARRAY BOOL

/* Assignment */
%token ASSIGN

/* Boolean Operators */
%token AND OR

%token <int> INT_LITERAL
%token <float> FLOAT_LITERAL
%token <string> STRING_LITERAL
%token <string> BOOL_LITERAL
%token <string> ID

/* Precedence */

%nonassoc NOELSE

%right ASSIGN
%left AND OR
%left NEQ EQ LEQ GEQ LT GT NOT
%left PLUS MINUS
%left TIMES DIVIDE

%start program
%type <Ast.program> program

%%

/* Start Program */
program:
    stmt_list EOF    { List.rev $1 }

/* Literals */
literal:
    primitive_literal { $1 }
  | json_literal     { $1 }

primitive_literal:
    INT_LITERAL     { Literal_int($1) }
  | FLOAT_LITERAL   { Literal_float($1) }
  | STRING_LITERAL   { Literal_string($1) }
  | BOOL_LITERAL     { Literal_bool($1) }

array_literal:
  LSQUARE primitive_literal_list_opt RSQUARE    { $2 }

primitive_literal_list_opt:
                                                          { [] }
  | primitive_literal_list                                { List.rev $1 }

primitive_literal_list:
  primitive_literal                                       { [$1] }
  | primitive_literal_list SEMICOLON primitive_literal    { $3 :: $1 }

json_literal:
    JSON LPAREN STRING_LITERAL RPAREN { Json_from_file($3) } /*String literal refers to filename*/

/* Variables */
/* ~~~~~~~~~~~~~~~~~~~ PLEASE REVISIT ~~~~~~~~~~~~~~~~~~~ */
data_type:
    INT     { "int" }
  | FLOAT   { "float" }
  | BOOL    { "bool" }
  | STRING  { "string" }
  | ARRAY   { "array" }
  | JSON    { "json" }

assignment_data_type:
    INT     { "int" }
  | FLOAT   { "float" }
  | STRING  { "string" }
  | JSON    { "json" }
  /* What happened to booleans? */

return_type:
    data_type   { $1 }
  | VOID        { "void" }

formals_opt:
    /* Nothing */   { [] }
  | formal_list     { List.rev $1 }

formal_list:
    arg_decl                    { [$1] }
  | formal_list COMMA arg_decl  { $3 :: $1 }

arg_decl:
```

```
    data_type ID
    {{
        var_type   = $1;
        var_name   = $2;
    }}


////////////////////////////////////////////////////
//////////////////// STATEMENTS ////////////////////
////////////////////////////////////////////////////

stmt_list:
  /* Nothing */    { [] }
  | stmt_list stmt {$2 :: $1}

stmt:
    expr ENDLINE                                 { Expr($1) }
  | FOR LPAREN assignment_stmt COMMA bool_expr COMMA
    assignment_stmt RPAREN LCURLY
    ENDLINE stmt_list RCURLY ENDLINE             { For($3, $5, $7, $11) }
  | WHILE LPAREN bool_expr RPAREN LCURLY
    ENDLINE stmt_list RCURLY ENDLINE             { While($3, $7) }
  | WHERE LPAREN bool_expr RPAREN AS ID
    LCURLY stmt_list RCURLY
    IN expr ENDLINE                              { Where($3, $6, $8, $11) }
  | if_else_stmt                                 { $1 }
  | assignment_stmt ENDLINE                      { $1 }
  | FUNCTION ID LPAREN formals_opt RPAREN COLON
    return_type LCURLY ENDLINE stmt_list RCURLY
    ENDLINE                                      { Func_decl($2, $4, $7, $10) }
  | RETURN expr ENDLINE                          { Return($2) }
  | ENDLINE                                      { Noop }

/* Different forms of if_else */
if_else_stmt:
  IF LPAREN bool_expr RPAREN
    LCURLY stmt_list RCURLY
    ENDLINE %prec NOELSE                         { If($3, $6, []) }
  | IF LPAREN bool_expr RPAREN
    LCURLY stmt_list RCURLY
    ELSE LCURLY stmt_list RCURLY ENDLINE         { If($3, $6, $10) }
  | IF LPAREN bool_expr RPAREN
    LCURLY stmt_list RCURLY ENDLINE
    ELSE LCURLY stmt_list RCURLY ENDLINE         { If($3, $6, $11) }

/* Assignment */
assignment_stmt:
    ARRAY assignment_data_type ID ASSIGN array_literal  { Array_assign($2, $3, $5)
}
    | ARRAY assignment_data_type ID ASSIGN ARRAY
      LPAREN INT_LITERAL RPAREN                          {
Fixed_length_array_assign($2, $3, $7) }
    | ARRAY assignment_data_type ID ASSIGN ID bracket_selector_list {
Array_select_assign($2, $3, $5, List.rev $6)}
    | assignment_data_type ID ASSIGN expr                { Assign($1, $2, $4) }
    | BOOL ID ASSIGN bool_expr                           { Bool_assign("bool", $2,
$4) }
    | ID ASSIGN expr                                     { Update_variable($1, $3) }
    | ID bracket_selector ASSIGN expr                    { Update_array_element($1,
$2, $4) }

bracket_selector_list:
  bracket_selector { [$1] }
  | bracket_selector_list bracket_selector { $2 :: $1 }

bracket_selector: LSQUARE expr RSQUARE { $2 }


actuals_opt:
    /* Nothing */ { [] }
  | actuals_list  { List.rev $1 }

actuals_list:
    expr                    { [$1] }
  | actuals_list COMMA expr { $3 :: $1 }

/* Expressions */
expr:
    literal                       { $1 }
  | ID                            { Id($1) }
  | 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) }
  | ID LPAREN actuals_opt RPAREN  { Call($1, $3) }
  | LPAREN expr RPAREN            { $2 }
  | ID bracket_selector_list      { Bracket_select($1, List.rev $2) }

bool_expr:
    BOOL_LITERAL                                { Literal_bool($1) }
  | ID                                          { Id($1) }
  | 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) }
  | bool_expr AND bool_expr                       { Bool_binop($1, And,    $3) }
  | bool_expr OR bool_expr                        { Bool_binop($1, Or,    $3) }
  | NOT bool_expr                                 { Not($2) }
```

compiler/ast.ml

```
(*
 * QL
 *
 * Manager: Matthew Piccolella
 * Systems Architect: Anshul Gupta
 * Tester: Evan Tarrh
 * Language Guru: Gary Lin
 * Systems Integrator: Mayank Mahajan
 *)

type bool_op = Equal | Neq | Less | Leq | Greater | Geq
type math_op = Add | Sub | Mult | Div
type binop_op = Equal | Neq | Less | Leq | Greater | Geq |
                Add | Sub | Mult | Div
type conditional = And | Or

type expr =
    | Literal_int of int
    | Literal_float of float
    | Literal_string of string
    | Literal_bool of string
    | Literal_array of expr list
    | Id of string
    | Binop of expr * math_op * expr
    | Call of string * expr list
    | Json_from_file of string
    | Bracket_select of string * expr list

type arg_decl = {
   var_type   : string;
```

```
    var_name    : string;
}

type bool_expr =
    | Literal_bool of string
    | Binop of expr * bool_op * expr
    | Bool_binop of bool_expr * conditional * bool_expr
    | Not of bool_expr
    | Id of string

type data_type =
    | Int
    | Float
    | Bool
    | String
    | Json
    | Array of data_type
    | AnyType

type stmt =
    | Expr of expr
    | For of stmt * bool_expr * stmt * stmt list
    | While of bool_expr * stmt list
    | Where of bool_expr * string * stmt list * expr
    | If of bool_expr * stmt list * stmt list
    | Return of expr
    | Assign of string * string * expr
    | Update_variable of string * expr
    | Update_array_element of string * expr * expr
    | Array_assign of string * string * expr list
    | Fixed_length_array_assign of string * string * int
    | Array_select_assign of string * string * string * expr list
    | Bool_assign of string * string * bool_expr
    | Func_decl of string * arg_decl list * string * stmt list
    | Noop

type program = stmt list
```

compiler/environment.ml

```
(*
 * QL
 *
 * Manager: Matthew Piccolella
 * Systems Architect: Anshul Gupta
 * Tester: Evan Tarrh
 * Language Guru: Gary Lin
 * Systems Integrator: Mayank Mahajan
 *)

open Ast;;

module FunctionMap = Map.Make(String);;
module VariableMap = Map.Make(String);;
module ArrayTypeMap = Map.Make(String);;
module JsonSelectorMap = Map.Make(String);;

exception VarAlreadyDeclared;;
exception VarNotDeclared;;
exception FunctionAlreadyDeclared;;
exception FunctionNotDeclared;;
exception IncorrectFunctionParameterTypes;;
exception MixedTypeArray;;
exception ArrayInferTypeMismatch;;
exception JsonSelectorTypeMismatch;;
exception JsonSelectorAlreadyUsed;;
exception IncorrectArrayAssignmentSize;;
exception DatatypeDefaultValueMissing;;

```

```
type func_info = {
  id : string;
  return : data_type;
  args : data_type list;
  arg_names: string list;
}

type symbol_table = {
  func_map: func_info FunctionMap.t;
  var_map: data_type VariableMap.t;
  array_type_map: data_type ArrayTypeMap.t;
  json_selector_map: data_type JsonSelectorMap.t;
}

let create =
  {
    func_map = FunctionMap.empty;
    var_map = VariableMap.empty;
    array_type_map = ArrayTypeMap.empty;
    json_selector_map = JsonSelectorMap.empty;
  }

let update f_map v_map a_type_map js_map =
  {
    func_map = f_map;
    var_map = v_map;
    array_type_map = a_type_map;
    json_selector_map = js_map;
  }

let string_to_data_type (s : string) = match s
  with "int" -> Int
  | "float" -> Float
  | "bool" -> Bool
  | "string" -> String
  | "array" -> Array(AnyType)
  | "json" -> Json
  | _ -> raise (Failure "String does not match a particular data type. Something went wrong.")

let rec data_type_to_string (dt : data_type) = match dt
  with Int -> "int"
  | Float -> "float"
  | Bool -> "bool"
  | String -> "string"
  | Array(t) -> "array of " ^ (data_type_to_string t)
  | Json -> "json"
  | _ -> raise (Failure "Data Type doesn't have a corresponding string.")

let declare_var (id : string) (data_type : string) (env : symbol_table) =
  if VariableMap.mem id env.var_map then
    raise VarAlreadyDeclared
  else
    let update_var_map = VariableMap.add id (string_to_data_type(data_type)) env.var_map in
    update env.func_map update_var_map env.array_type_map env.json_selector_map

let var_type (id : string) (env : symbol_table) =
  if VariableMap.mem id env.var_map then
    VariableMap.find id env.var_map
  else
    raise VarNotDeclared

let create_func (func_name: string) (ret_type : string) (args : arg_decl list) =
  {
    id = func_name;
    return = (string_to_data_type ret_type);
    args = List.map (fun arg -> string_to_data_type arg.var_type) args;
    arg_names = List.map (fun arg -> arg.var_name) args;
  }

let define_array_type (expected_type: data_type)
  (inferred_type : data_type list) (env : symbol_table) (id : string) =
  if List.length inferred_type != 0 then
    let first_type = List.hd inferred_type in
    (* Verify all types are the same *)
    List.iter (fun (data_type) -> if first_type != data_type then raise MixedTypeArray) inferred_type;
    (if first_type == expected_type then
      let update_array_type_map = ArrayTypeMap.add id first_type env.array_type_map in
      update env.func_map env.var_map update_array_type_map env.json_selector_map
    else
      raise ArrayInferTypeMismatch)
  else
    (* Empty array created *)
    let update_array_type_map = ArrayTypeMap.add id expected_type env.array_type_map in
    update env.func_map env.var_map update_array_type_map env.json_selector_map

let array_type (id : string) (env : symbol_table) =
  if ArrayTypeMap.mem id env.array_type_map then
    ArrayTypeMap.find id env.array_type_map
  else
    raise VarNotDeclared

let rec define_func_vars (func_vars : arg_decl list) (env : symbol_table) = match func_vars
```

```
  with [] -> env
  | head::body ->
    let new_env = declare_var head.var_name head.var_type env in
    define_func_vars body new_env

let declare_func (func_name : string) (ret_type : string) (args : arg_decl list) (env : symbol_table) =
  if FunctionMap.mem func_name env.func_map then
    raise FunctionAlreadyDeclared
  else
    let update_func_map = FunctionMap.add func_name (create_func func_name ret_type args) env.func_map in
    update update_func_map env.var_map env.array_type_map env.json_selector_map

let verify_func_call (func_name: string) (args : data_type list) (env : symbol_table) =
  match func_name with
    "print" | "length" -> ()
    | _ ->
    if FunctionMap.mem func_name env.func_map then
      let declared_func = FunctionMap.find func_name env.func_map in
      let type_pairs = List.combine args declared_func.args in
      List.iter (fun (left, right) ->
        if left != right then
          if left != AnyType && right != AnyType then
            raise IncorrectFunctionParameterTypes
      ) type_pairs
    else
      raise FunctionNotDeclared

let func_return_type (func_name : string) (env : symbol_table) =
  match func_name with
    "print" -> String
    | "length" -> Int
    | _ ->
      if FunctionMap.mem func_name env.func_map then
        let declared_func = FunctionMap.find func_name env.func_map in
        declared_func.return
      else
        raise FunctionNotDeclared

let json_selector_update (id : string) (data_type : string) (env :
symbol_table) =
  if JsonSelectorMap.mem id env.json_selector_map then
    let json_selector_type = JsonSelectorMap.find id env.json_selector_map in
    if (string_to_data_type data_type) != json_selector_type then
      raise JsonSelectorTypeMismatch
    else
      env
  else
    let update_json_selector = JsonSelectorMap.add id (string_to_data_type
data_type) env.json_selector_map in
    update env.func_map env.var_map env.array_type_map update_json_selector

let json_selector_type (id : string) (env : symbol_table) =
  if JsonSelectorMap.mem id env.json_selector_map then
    JsonSelectorMap.find id env.json_selector_map
  else
    AnyType

(* Important: when loops terminate, we need just the JSON map to persist to
the new env. *)
let overwrite_js_map env new_env =
  update env.func_map env.var_map env.array_type_map
new_env.json_selector_map
```

compiler/semantic.ml

```ocaml
(*
* QL
*
* Manager: Matthew Piccolella
* Systems Architect: Anshul Gupta
* Tester: Evan Tarrh
* Language Guru: Gary Lin
* Systems Integrator: Mayank Mahajan
*)

open Ast;;
open Environment;;

type data_type = Int | Float | Bool | String | Array | Json | AnyType

exception ReturnStatementMissing;;
exception ImproperBraceSelection;;
exception ImproperBraceSelectorType;;
exception MultiDimensionalArraysNotAllowed;;
exception NotBoolExpr;;
exception BadBinopType;;
exception IncorrectWhereType;;
exception UpdatingBool;;
exception IncorrectSelectorId;; (*When you use selectors with IDs that aren't
jsons or arrays*)
exception UniterableType;;

let ast_data_to_data (dt : Ast.data_type) = match dt
    with Int -> Int
    | Float -> Float
    | Bool -> Bool
    | String -> String
    | Array(i) -> Array
    | Json -> Json
    | _ -> AnyType

let ast_data_to_string (dt : Ast.data_type) = match dt
    with Int -> "int"
    | Float -> "float"
    | Bool -> "bool"
    | String -> "string"
    | Array(i) -> "array"
    | Json -> "json"
    | AnyType -> "anytype"

let data_to_ast_data (dt : data_type) = match dt
    with Int -> Ast.Int
    | Float -> Ast.Float
    | Bool -> Ast.Bool
    | String -> Ast.String
    | Array -> Ast.Array(Ast.Int)
    | Json -> Ast.Json
    | _ -> Ast.AnyType

let string_to_data_type (s : string) = match s
    with "int" -> Int
    | "float" -> Float
    | "bool" -> Bool
    | "string" -> String
    | "array" -> Array
    | "json" -> Json
    | _ -> raise (Failure "Unsupported Data Type.")

let string_data_literal (expr : Ast.expr) = match expr
        with Literal_int(i) -> string_of_int i
    | Literal_float(i) -> string_of_float i
    | Literal_bool(i) -> i
    | Literal_string(i) -> i
    | _ -> raise (Failure "There is no defined way to print this expression.")

let check_binop_type (left_expr : data_type) (op : Ast.math_op) (right_expr : data_type) = match (left_expr, op, right_expr)
    with (Int, _, Int) -> Int
    | (Float, _, Float) -> Float
    | (String, Add, String) -> String
    | (AnyType, _, Int) -> Int
    | (AnyType, _, Float) -> Float
    | (AnyType, Add, String) -> String
    | (Int, _, AnyType) -> Int
    | (Float, _, AnyType) -> Float
    | (String, Add, AnyType) -> String
    | (AnyType, Add, AnyType) -> AnyType
    | (AnyType, _, AnyType) -> Float
    | (_, _, _) ->
        raise (Failure "Cannot perform binary operations with provided arguments.")

(*Possibly add int/float comparison*)
let check_bool_expr_binop_type (left_expr : data_type) (op : Ast.bool_op) (right_expr : data_type) = match op
        with Equal | Neq -> (match (left_expr, right_expr)
            with (Int, Int) -> Bool
            | (Float, Float) -> Bool
            | (String, String) -> Bool
            | (Bool, Bool) -> Bool
            | (AnyType, _) -> Bool
            | (_, AnyType) -> Bool
            | _ -> raise (Failure "Cannot perform boolean binary operations with provided arguments.")
            )
        | Less | Leq | Greater | Geq -> (match (left_expr, right_expr)
            with (Int, Int) -> Bool
            | (Float, Float) -> Bool
            | (AnyType, _) -> Bool
            | (_, AnyType) -> Bool
            | _ -> raise (Failure "Cannot perform comparison binary operations with provided arguments.")
            )

let rec check_bracket_select_type (d_type : data_type) (selectors : expr list) (env : symbol_table) (id : string) (serial : string) = match d_type
    with Array ->
        if List.length selectors != 1 then raise MultiDimensionalArraysNotAllowed;
        (* We can ignore the env because we're explicitly updating later. *)
        let (expr_type, _) = check_expr_type (List.hd selectors) (env) in
        if expr_type != Int && expr_type != AnyType then raise ImproperBraceSelectorType;
        let ast_array_type = array_type (id) (env) in
        let data_type = ast_data_to_data ast_array_type in
        if expr_type == AnyType then
            let expr_head = List.hd selectors in
            let json_update_map = json_selector_update (serialize (expr_head) (env)) "int" (env) in
            (data_type, json_update_map)
        else
            (data_type, env)

    (* Return the type being stored for this particular array *)
    | Json ->
        List.iteri (fun index expr ->
                let (expr_type, _) = check_expr_type (expr) (env) in
                (* Might need to infer the type if we JSON select, as it could be a string or int. *)
                if expr_type != String && index == 0 then (raise
ImproperBraceSelectorType);
                if expr_type != String && expr_type != Int then (raise ImproperBraceSelectorType);
        ) selectors;
        (ast_data_to_data (json_selector_type (serial) (env)), env)
    | _ -> raise ImproperBraceSelection

and check_expr_type (expr : Ast.expr) (env: Environment.symbol_table) = match expr
    with Literal_int(i) -> (Int,env)
    | Literal_float(i) -> (Float,env)
    | Literal_bool(i) -> (Bool,env)
    | Literal_string(i) -> (String,env)
    | Literal_array(i) -> (Array,env)
    | Json_from_file(i) -> (Json,env)
    | Binop(left_expr, op, right_expr) ->
        let (left_type, left_env) = (check_expr_type left_expr env) in
        let (right_type, right_env) = (check_expr_type right_expr left_env) in
        let resulting_type = check_binop_type left_type op right_type in
        (match (left_type, right_type) with
            (AnyType, AnyType) -> (AnyType, env)
      | (AnyType, _) ->
          let serialized = serialize left_expr right_env in
          let new_env = json_selector_update serialized (ast_data_to_string (data_to_ast_data (right_type))) (right_env)
             in
          (right_type, new_env)
      | (_, AnyType) ->
          let serialized = serialize right_expr right_env in
          let new_env = json_selector_update serialized (ast_data_to_string(data_to_ast_data(left_type))) right_env in
          (left_type, new_env)
      | (_,_) -> (resulting_type, right_env))
    | Id(i) -> (ast_data_to_data((var_type i env)), env)
    | Call(func_name, arg_list) ->
        let arg_types = List.map (fun expr ->
            let (expr_type, _) = (check_expr_type (expr) (env)) in
            (data_to_ast_data(expr_type))) arg_list in
        verify_func_call func_name arg_types env;
        let func_return_type = func_return_type func_name env in (match func_name
            with "print" ->    (ast_data_to_data(func_return_type), env)
            | "length" -> (ast_data_to_data(func_return_type), env)
            | _ -> let func_args = (FunctionMap.find func_name env.func_map) in
                let func_arg_types = func_args.args in
                let new_json_mapping = List.fold_left2 (fun env expr expected_type -> (match expr
                    with Bracket_select(id, selectors) -> json_selector_update (serialize (expr) (env)) (ast_data_to_string expected_type) (env)
                    | _ -> env
                )) env arg_list func_arg_types in
                (ast_data_to_data(func_return_type), new_json_mapping)
        )
    | Bracket_select(id, selectors) ->
        let selector_ast_data_type = var_type id env in
        let selector_data_type = ast_data_to_data selector_ast_data_type in
        let serialized = serialize (expr) (env) in
        (check_bracket_select_type (selector_data_type) (selectors) (env) (id) serialized)

and serialize (expr : Ast.expr) (env : symbol_table) = match expr
    with Bracket_select(id, selectors) ->
        let serialized = List.fold_left (fun acc x ->
            let (expr_type,_) = check_expr_type (x) (env) in
            (match expr_type
                with String -> acc ^ "[\"" ^ (serialize_literal x) ^ "\"]"
                | Int -> acc ^ (serialize_literal x)
                | _ -> acc
            )
            ) id (List.rev selectors) in
        serialized
        (* This function is hard to write - hard to serialize an arbitrary expression. *)
    | _ -> raise (Failure "Cannot serialize a non-Bracket Select type.")

and serialize_literal (literal : expr) = match literal
        with Literal_int(i) -> string_of_int i
    | Literal_float(i) -> string_of_float i
    | Literal_bool(i) -> i
    | Literal_string(i) -> i
    | Id(i) -> i
    | Bracket_select(id, selectors) -> (List.fold_left (
        fun str expr -> str ^ (serialize_literal (expr)))) id selectors
    | _ -> raise (Failure "Printing this is undefined.")

let rec map_json_types (expr : Ast.expr) (env : symbol_table) (data_type : string) = match expr
    with Binop(left_expr, op, right_expr) ->
        let left_env = (map_json_types (left_expr) (env) (data_type)) in
        let right_env = (map_json_types (right_expr) (left_env) (data_type)) in
        right_env
    | Bracket_select(id, selectors) ->
        json_selector_update (serialize (expr) (env)) (data_type) (env)
    | _ -> env

let json_selector_found (expr : Ast.expr) (env : symbol_table) = match expr
    with Bracket_select(id, selectors) ->
        let selector_ast_data_type = var_type id env in
        let selector_data_type = ast_data_to_data selector_ast_data_type in
        if selector_data_type == Json then
            (List.iteri (fun index expr ->
                (* If we use a non-declared JSON value, we'll have an error. Think about how to infer this. *)
                let (expr_type,_) = check_expr_type (expr) (env) in
                if expr_type != String && index = 0 then raise ImproperBraceSelectorType;
                if expr_type != String && expr_type != Int then raise ImproperBraceSelectorType;
            ) selectors;
            true)
        else
            false
    | _ -> false

let equate e1 e2 =
    if (e1 != e2) && (e1 != AnyType) && (e2 != AnyType) then raise (Failure "data_type mismatch")

let string_data_literal (expr : Ast.expr) = match expr
        with Literal_int(i) -> string_of_int i
    | Literal_float(i) -> string_of_float i
    | Literal_bool(i) -> i
    | Literal_string(i) -> i
    | _ -> raise (Failure "Printing this is undefined.")

let handle_expr_statement (expr : Ast.expr) (env: Environment.symbol_table) = match expr
    with Call(f_name, args) -> (match f_name with
        "print" ->
             if List.length args != 1 then
                raise (Failure "Print only takes one argument.")
            else
```

```
                let (_, _) = (check_expr_type (List.hd args) (env)) in
                env
        | "length" ->
             if List.length args != 1 then
                raise (Failure "Length only takes one argument.")
            else
                let (_, _) = (check_expr_type (List.hd args) (env)) in
                env
        | _ ->
            let arg_types = List.map (fun expr ->
                let (expr_type, _) = (check_expr_type (expr) (env)) in
                (data_to_ast_data(expr_type))) args in
            verify_func_call f_name arg_types env;
            let func_args = FunctionMap.find f_name env.func_map in
            let func_arg_types = func_args.args in
            let new_json_mapping = List.fold_left2 (fun env expr expected_type -> (match expr
                with Bracket_select(id, selectors) -> json_selector_update (serialize (expr) (env)) (ast_data_to_string expected_type) (env)
                | _ -> env
                )) env args func_arg_types in
            new_json_mapping
        )
    | _ -> env

let handle_json (json_expr : Ast.expr) (env : Environment.symbol_table) = match json_expr
    with Id(i) -> (match var_type i env
        with Json -> Json
        | _ -> raise IncorrectWhereType)
    | Json_from_file(json) -> Json
    | _ -> raise IncorrectWhereType

let rec handle_bool_expr (bool_expr : Ast.bool_expr) (env :
```

```
Environment.symbol_table) = match bool_expr
    with Literal_bool(i) -> (Bool,env)
    | Binop(e1, op, e2) ->
            let (l_type, left_env) = (check_expr_type e1 env) in
            let (r_type, right_env) = (check_expr_type e2 left_env) in
            let _ = check_bool_expr_binop_type (l_type) (op) (r_type) in
            (match (l_type, r_type)
                with (AnyType, AnyType) ->
                    (* NOTE: We're defining things that are compared as floats, so we can define them as something. *)
                    let new_left_env = json_selector_update (serialize (e1) (env)) "float" (right_env) in
                    let new_right_env = json_selector_update (serialize (e2) (env)) "float" (new_left_env) in
                    (Bool, new_right_env)
                | (AnyType, _) ->
                    let serialized = serialize e1 right_env in
                    let new_env = json_selector_update serialized (ast_data_to_string (data_to_ast_data (r_type))) (right_env) in
                        (Bool, new_env)
                | (_, AnyType) ->
                    let new_env = json_selector_update (serialize (e2) (right_env)) (ast_data_to_string (data_to_ast_data (l_type))) (right_env) in
                    (Bool, new_env)
                | (_, _) -> (Bool, right_env)
            )
    | Bool_binop(e1, conditional, e2) ->
        let (_, left_env) = handle_bool_expr (e1) (env) in
        let (_, right_env) = handle_bool_expr (e2) (left_env) in
         (Bool, right_env)
    | Not(e1) ->
        let (_,new_env) = handle_bool_expr (e1) (env) in
        (Bool, new_env)
    | Id(i) -> match var_type i env
```

```
        with Bool -> (Bool,env)
        | _ -> raise NotBoolExpr

let rec check_statement (stmt : Ast.stmt) (env : Environment.symbol_table) = match stmt
    with Expr(e1) ->
        let updated_expr = (handle_expr_statement (e1) (env)) in
        updated_expr
    | Update_variable (id, e1) ->
        let ast_dt = var_type id env in
            if ast_dt == Bool then
                raise UpdatingBool
            else
                let data_type = ast_data_to_data ast_dt in
                    if data_type == Json then
                        raise (Failure "JSON aliasing is not supported in QL.")
                    else
                        let (right,new_env) = check_expr_type (e1) (env) in
                            equate data_type right;
                            (match right
                                with AnyType -> json_selector_update (serialize (e1) (new_env)) (ast_data_to_string ast_dt) (new_env)
                                | _ -> new_env
                            )
     | If(bool_expr, then_stmt, else_stmt) ->
        let (_,new_env) = handle_bool_expr bool_expr env in
        let post_if_env = check_statements (List.rev then_stmt) (new_env) in
        let override_if = overwrite_js_map new_env post_if_env in
        let post_else_env = check_statements (List.rev else_stmt) (override_if) in
        let post_loop_env = overwrite_js_map override_if post_else_env in
        post_loop_env
      | Update_array_element (id, e1, e2) ->
```

```
          let ast_array_data_type = array_type id env in
        let data_type = ast_data_to_data ast_array_data_type in
        let (right, new_env) = check_expr_type (e2) (env) in
            equate data_type right;
            env;
    | For(init_stmt, bool_expr, update_stmt, stmt_list) ->
        let init_env = check_statement init_stmt env in
        let (_,new_env) = handle_bool_expr bool_expr init_env in
        let update_env = check_statement update_stmt new_env in
        let post_loop_env = check_statements (List.rev stmt_list) (update_env) in
            (* We need to worry about scoping here. I think we want all the things in bool expr to count. *)
        overwrite_js_map new_env post_loop_env
    | While(bool_expr, body) ->
        let (_,while_env) = handle_bool_expr bool_expr env in
        let post_loop_env = check_statements (List.rev body) (while_env) in
        (* Same thing here. We might want to be returning while_env *)
        overwrite_js_map while_env post_loop_env
    | Where(bool_expr, id, stmt_list, json_object) ->
        let update_env = declare_var id "json" env in
        let (_,where_env) = handle_bool_expr bool_expr update_env in
        let serialized = serialize json_object where_env in
        let serial_env = (match (json_selector_type serialized where_env) with
            Ast.Array(_) -> where_env
            | Ast.AnyType -> let array_set_env = (json_selector_update serialized (ast_data_to_string (Ast.Array(Ast.AnyType))) where_env) in
                array_set_env
            | _ -> raise UniterableType;) in
        let post_loop_env = (check_statements (List.rev stmt_list) (serial_env)) in
        (* Also here. *)
        overwrite_js_map serial_env post_loop_env
    | Assign(data_type, id, e1) ->
    if (json_selector_found e1 env) = true then
        let updated_env = declare_var id data_type env in
        let final_env = json_selector_update (serialize e1 env) data_type updated_env in
        final_env
    else
        let left = string_to_data_type(data_type) and (right,new_env) = check_expr_type (e1) (env) in
        equate left right;
        let declared_var = declare_var id data_type new_env in
        map_json_types e1 declared_var data_type
    | Array_assign(expected_data_type, id, e1) ->
        let left = data_to_ast_data(string_to_data_type(expected_data_type)) in
            let inferred_type = List.map (fun expr ->
                (* Don't need to use new env because we won't have JSON in the array *)
                let (data_type,_) = (check_expr_type (expr) (env)) in
                data_to_ast_data (data_type)) e1 in
                let declare_var_env = declare_var id "array" env in
                  define_array_type (left) (inferred_type) (declare_var_env) (id)
    | Fixed_length_array_assign(expected_data_type, id, length) ->
        let left = data_to_ast_data(string_to_data_type(expected_data_type)) in
            let declare_var_env = declare_var id "array" env in
                define_array_type left [] declare_var_env id
    | Array_select_assign(expected_data_type, new_var_id, array_id, selectors ) ->
        let left = data_to_ast_data (string_to_data_type expected_data_type) in (match var_type array_id env with Json ->
                let json_type = json_selector_type
                (*THIS PART DOESNT WORK BC WE ONLY PASS THE ID NOT THE
SERIALIZED THING WITH THE SELECTORS*)
                array_id env in  (match json_type with
                Ast.AnyType ->
                let declare_var_env = declare_var new_var_id "array" env in
                    let _ = define_array_type left [] declare_var_env new_var_id in
                    json_selector_update (serialize (Ast.Bracket_select(array_id, selectors)) env) expected_data_type env
                | other_type -> equate (string_to_data_type expected_data_type) (ast_data_to_data other_type);
                    let declare_var_env = declare_var new_var_id "array" env in define_array_type left [] declare_var_env new_var_id
                )
            | _ -> raise IncorrectSelectorId; )

    | Bool_assign(data_type, id, e1) ->
        let left = string_to_data_type(data_type) and (right,new_env) = handle_bool_expr (e1) (env) in
            equate left right;
            declare_var id data_type new_env;
    | Func_decl(func_name, arg_list, return_type, stmt_list) ->
        let func_env = declare_func func_name return_type arg_list env in
        let func_env_vars = define_func_vars arg_list func_env in
        (* TODO: Implement void functions *)
        if (return_type != "void" && (List.length stmt_list) == 0) then raise ReturnStatementMissing;
        let post_func_env = check_function_statements (List.rev stmt_list) func_env_vars return_type in
        overwrite_js_map func_env post_func_env
    | Noop -> env
    | _ -> raise (Failure "Unimplemented functionality.")

and check_statements (stmts : Ast.stmt list) (env : Environment.symbol_table) = match stmts
```

```
    with [] -> env
  | [stmt] -> check_statement stmt env
  | stmt :: other_stmts ->
           let new_env = check_statement stmt env in
          check_statements other_stmts new_env

and check_function_statements stmts env return_type = match stmts
    with [] ->
    if return_type != "void" then raise ReturnStatementMissing else
    env
  | [stmt] ->
      check_return_statement stmt env return_type
  | stmt :: other_stmts ->
          let env = check_statement stmt env in
          check_function_statements other_stmts env return_type

and check_return_statement (stmt : Ast.stmt) (env : Environment.symbol_table) (return_type : string) =
    if return_type != "void" then match stmt
        with Return(expr) ->
            (* Since we're returning, we don't need to declare the returned JSON type as a type *)
            let left = string_to_data_type(return_type) and (right,_) = check_expr_type (expr) (env) in
                equate left right;
                env
        | _ -> raise (Failure "Function must end with return statement.")
    else
        check_statement stmt env

(* entry point into semantic checker *)
let check_program (stmt_list : Ast.program) =
    let env = Environment.create in
    check_statements (stmt_list) (env);
```

compiler/semantic_to_jast.ml

```ocaml
(*
* QL
*
* Manager: Matthew Piccolella
* Systems Architect: Anshul Gupta
* Tester: Evan Tarrh
* Language Guru: Gary Lin
* Systems Integrator: Mayank Mahajan
*)

open Ast;;
open Environment;;
open Jast;;
open Semantic;;

let ql_to_java_type (data_type : string) = match data_type
  with "int" -> "int"
  | "float" -> "double"
  | "string" -> "String"
  | "json" -> "JSONObject"
  | "bool" -> "boolean"
  | "array" -> "JSONArray"
  | _ -> ("Invalid data type " ^ data_type ^ ".")

let convert_math_op (op : Ast.math_op) = match op
  with Add -> Jast.Add
  | Sub -> Jast.Sub
  | Mult -> Jast.Mult
  | Div -> Jast.Div

let convert_bool_op (op : Ast.bool_op) = match op
  with Equal -> Jast.Equal
  | Neq -> Jast.Neq
  | Less -> Jast.Less
  | Leq -> Jast.Leq
  | Greater -> Jast.Greater
  | Geq -> Jast.Geq

let convert_cond (cond : Ast.conditional) = match cond
  with And -> Jast.And
  | Or -> Jast.Or

let rec convert_data_type (data_type : Semantic.data_type) = match data_type
  with Semantic.Int -> Jast.Int
  | Semantic.Float -> Jast.Double
  | Semantic.Bool -> Jast.Bool
  | Semantic.String -> Jast.String
  | Semantic.Json -> Jast.Json
  | Semantic.Array -> Jast.Array(convert_data_type (data_type))
  | Semantic.AnyType -> raise (Failure "We should not have an AnyType here.")

let rec convert_expr (expr : Ast.expr) (symbol_table : Environment.symbol_table) = match expr
  with Ast.Literal_int(i) -> Jast.Literal_int(i)
  | Ast.Literal_string(i) -> Jast.Literal_string(i)
  | Ast.Literal_float(i) -> Jast.Literal_double(i)
  | Ast.Literal_bool(i) -> Jast.Literal_bool(i)
  | Ast.Call(func_name, arg_list) ->
    let converted_args = List.map (fun arg -> (convert_expr (arg) (symbol_table))) arg_list in
    Jast.Call(func_name, converted_args)
  | Ast.Id(i) -> Jast.Id(i)
  | Ast.Binop(left_expr, op, right_expr) ->
    let left_convert = convert_expr left_expr symbol_table in
    let right_convert = convert_expr right_expr symbol_table in
    let op_convert = convert_math_op op in
    Jast.Binop(left_convert, op_convert, right_convert)
  | Ast.Bracket_select(id, selectors) ->
    let id_type = var_type id symbol_table in
    (match id_type
      with Array(data_type) ->
        let head_expr = (convert_expr (List.hd selectors) (symbol_table)) in
        Jast.Array_select(id, head_expr)
      | _ ->
        let selector_exprs = List.map (fun select -> (convert_expr (select)
(symbol_table))) selectors in
        let serialized = serialize (expr) (symbol_table) in
        let json_type = json_selector_type (serialized) (symbol_table) in
        let java_type = ql_to_java_type (ast_data_to_string (json_type)) in
        let selector_types = List.map (fun expr ->
          let (expr_type,_) = check_expr_type (expr) (symbol_table) in
          convert_data_type (expr_type)
        ) selectors in
        Jast.Bracket_select(id, java_type, selector_exprs, selector_types)
    )
  | Ast.Json_from_file(i) -> Jast.Json_object(i)
  | Ast.Literal_array(expr_list) ->
    let sast_exprs = List.map (fun expr -> (convert_expr (expr)
(symbol_table))) expr_list in
    Jast.Array_initializer(sast_exprs)


let rec convert_bool_expr (op : Ast.bool_expr) (symbol_table :
Environment.symbol_table) = match op
  with Ast.Literal_bool(i) -> Jast.Literal_bool(i)
  | Ast.Binop(left_exp, op, right_expr) -> Jast.Binop((convert_expr
(left_exp) (symbol_table)), (convert_bool_op (op)), (convert_expr (right_expr)
(symbol_table)))
  | Ast.Bool_binop(left_exp, cond, right_exp) ->
Jast.Bool_binop((convert_bool_expr (left_exp) (symbol_table)), (convert_cond
cond), (convert_bool_expr (right_exp) (symbol_table)))
  | Ast.Not(exp) -> Jast.Not((convert_bool_expr (exp) (symbol_table)))
  | Ast.Id(i) -> Jast.Id(i)

let convert_arg_decl (arg_decl : Ast.arg_decl) =
  {
    var_type = ql_to_java_type arg_decl.var_type;
    var_name = arg_decl.var_name;
  }

let rec build_expr_list (jast_exprs: Jast.expr list) (exprs: Ast.expr list) (symbol_table: Environment.symbol_table) =
  match exprs
    with [head] -> List.rev (jast_exprs@[(convert_expr head symbol_table)])
    | head :: tl -> (build_expr_list (jast_exprs@[(convert_expr head symbol_table)]) tl symbol_table)
    | _ -> []

let rec convert_statement (stmt : Ast.stmt) (symbol_table : Environment.symbol_table) = match stmt
  with Ast.Assign(data_type, id, e1) ->
    let corresponding_data_type = ql_to_java_type data_type in
    Jast.Assign(corresponding_data_type, id, (convert_expr (e1) (symbol_table)))
  | Ast.Expr(e1) -> Jast.Expr(convert_expr e1 symbol_table)
  | Ast.Array_assign(expected_data_type, id, e1) ->
    let expr_list = List.map (fun expr -> (convert_expr (expr) (symbol_table))) e1 in
    Jast.Array_assign((ql_to_java_type (expected_data_type)), id, expr_list)
  | Ast.Fixed_length_array_assign(expected_data_type, id, length) ->
    Jast.Fixed_length_array_assign((ql_to_java_type (expected_data_type)), id, length)
  | Ast.Update_variable (id, e1) ->
    let update_expr = convert_expr e1 symbol_table in
    Jast.Update_variable(id, update_expr)
  | Ast.Update_array_element (id, e1, e2) ->
    let index_expr = convert_expr e1 symbol_table in
      let update_expr = convert_expr e2 symbol_table in
        Jast.Update_array_element(id, index_expr, update_expr)
  | Ast.Bool_assign(data_type, id, e1) -> Jast.Bool_assign(id, (convert_bool_expr (e1) (symbol_table)))
  | Ast.Return(e1) -> Jast.Return(convert_expr e1 symbol_table)
  | Ast.Func_decl(id, arg_decl_list, return_type, body) ->
    let jast_arg_decl_list = List.map convert_arg_decl arg_decl_list in
    let jast_body = build_list [] body symbol_table in
    Jast.Func_decl(id, jast_arg_decl_list, ql_to_java_type return_type, jast_body)
  | Ast.If(condition, body, else_body) ->
    let jast_body = build_list [] body symbol_table in
    let jast_else_body = build_list [] else_body symbol_table in
    Jast.If(convert_bool_expr condition symbol_table, jast_body, jast_else_body)
  | Ast.While(condition, body) ->
    let jast_body = build_list [] body symbol_table in
    Jast.While(convert_bool_expr condition symbol_table, jast_body)
  | Ast.For(init, condition, update, body) ->
    let jast_init = convert_statement init symbol_table in
    let jast_condition = convert_bool_expr condition symbol_table in
    let jast_update = convert_statement update symbol_table in
    let jast_body = build_list [] body symbol_table in
    Jast.For(jast_init, jast_condition, jast_update, jast_body)
  | Ast.Array_select_assign(expected_data_type, new_var_id, array_id, selectors) ->
    Jast.Array_select_assign((ql_to_java_type expected_data_type), new_var_id, array_id, (build_expr_list [] selectors symbol_table))
  | Ast.Where(condition, id, body, json_array) ->
    let jast_condition = convert_bool_expr condition symbol_table in
    let jast_body = build_list [] body symbol_table in
```

```
    let jast_json_array = convert_expr json_array symbol_table in
    Jast.Where(jast_condition, id, jast_body, jast_json_array)
  | Ast.Noop -> Jast.Noop

and build_list (jast_body: Jast.stmt list) (body: Ast.stmt list)
(symbol_table: Environment.symbol_table) =
  match body
    with [head] -> List.rev (jast_body@[(convert_statement head
symbol_table)])
    | head :: tl -> (build_list (jast_body@[(convert_statement head
symbol_table)]) tl symbol_table)
    | _ -> []

let convert_semantic (stmt_list : Ast.program) (symbol_table :
Environment.symbol_table) =
  List.map (fun stmt -> convert_statement (stmt) (symbol_table)) stmt_list
```

compiler/jast.ml

```
(*
 * QL
 *
 * Manager: Matthew Piccolella
 * Systems Architect: Anshul Gupta
 * Tester: Evan Tarrh
 * Language Guru: Gary Lin
 * Systems Integrator: Mayank Mahajan
 *)

open Ast

type math_op = Add | Sub | Mult | Div
type conditional = And | Or
type bool_op = Equal | Neq | Less | Leq | Greater | Geq

```

```ocaml
type data_type =
    | Int
    | Double
    | Bool
    | String
    | Json
    | Array of data_type

type arg_decl = {
    var_type   : string;
    var_name   : string;
}

type expr =
    | Literal_int of int
    | Literal_double of float
    | Literal_string of string
    | Literal_bool of string
    | Array_initializer of expr list
    | Id of string
    | Binop of expr * math_op * expr
    | Call of string * expr list
    | Json_object of string
    | Bracket_select of string * string * expr list * data_type list
    | Array_select of string * expr

type bool_expr =
    | Literal_bool of string
    | Binop of expr * bool_op * expr
    | Bool_binop of bool_expr * conditional * bool_expr
    | Not of bool_expr
    | Id of string

type stmt =
    | Assign of string * string * expr
    | Expr of expr
    | Array_assign of string * string * expr list
    | Fixed_length_array_assign of string * string * int
    | Array_select_assign of string * string * string * expr list
    | Update_variable of string * expr
    | Update_array_element of string * expr * expr
    | Bool_assign of string * bool_expr
    | Return of expr
    | Func_decl of string * arg_decl list * string * stmt list
    | If of bool_expr * stmt list * stmt list
    | While of bool_expr * stmt list
    | For of stmt * bool_expr * stmt * stmt list
    | Where of bool_expr * string * stmt list * expr
    | Noop

type program = stmt list
```

compiler/code_generator.ml

```
(*
 * QL
 *
 * Manager: Matthew Piccolella
 * Systems Architect: Anshul Gupta
 * Tester: Evan Tarrh
 * Language Guru: Gary Lin
 * Systems Integrator: Mayank Mahajan
 *)

open Jast;;
open Str;;

let rec range i j = if i > j then [] else i :: (range (i+1) j)

let convert_operator (op : math_op) = match op
  with Add -> "+"
  | Sub -> "-"
  | Mult -> "*"
  | Div -> "/"

let convert_bool_operator (op : bool_op) = match op
  with Equal -> "=="
  | Neq -> "!="
  | Less -> "<"
  | Leq -> "<="
  | Greater -> ">"
  | Geq -> ">="

let convert_cond_op (cond : conditional) = match cond
  with And -> "&&"
  | Or -> "||"

let cast_json_access (prog_str : string) (data_type : string) = match
data_type
  with "int" -> "((Long) " ^ prog_str ^ ").intValue()"
  | "double" -> "((Number) " ^ prog_str ^ ").doubleValue()"
  | "JSONObject" -> ""
  | "boolean" -> ""
  | "String" ->  "(String)" ^ prog_str
  | "JSONArray" -> "(JSONArray)" ^ prog_str
  | _ -> raise (Failure (data_type^"Failure in Java casting a JSON access."))

let rec comma_separate_list (expr_list : Jast.expr list) = match expr_list
  with [] -> ""
  | head :: exprs ->
    if List.length exprs != 0 then
      (handle_expression head) ^ "," ^ (comma_separate_list exprs)
    else
```

```
   (handle_expression head)

and handle_expression (expr : Jast.expr) = match expr
 with Call(func_name, expr_list) -> (match func_name
  with "print" ->
   "System.out.println(" ^ (handle_expression (List.hd expr_list)) ^ ");\n"
  | "length" -> (match List.hd expr_list
   with Id(i) ->
    i ^ ".length\n"
   | _ -> raise (Failure "Cannot find the length of a non-variable.")
   )
  | _ -> func_name ^ "(" ^ comma_separate_list expr_list ^ ")"
 )
 | Literal_string(i) -> "\"" ^ i ^ "\""
 | Literal_int(i) -> string_of_int i
 | Literal_double(i) -> string_of_float i
 | Id(i) -> i
 | Array_select(id, interior) ->
  let select_index = handle_expression interior in
  id ^ "[" ^ select_index ^ "]"
 | Literal_bool(i) ->
  (match i
   with "True" -> "true"
   | "False" -> "false"
   | _ -> "bad")
 | Binop(left_expr, op, right_expr) -> handle_expression left_expr ^ " " ^ convert_operator op ^ " " ^ handle_expression right_expr
 | Json_object(file_name) -> "(JSONObject) (new JSONParser()).parse(new FileReader(\""^ file_name ^ "\"))"
 | Array_initializer(expr_list) -> "{" ^ comma_separate_list (expr_list) ^ "}"
 | Bracket_select(id, data_type, expr_list, expr_types) ->
  if List.length expr_list == 1 then (
   cast_json_access (id ^ ".get(" ^ handle_expression (List.hd expr_list) ^ ")") data_type
   )
  else
   let expr_type_pairs = List.combine expr_list expr_types in
   let expr_num_range = range 0 ((List.length expr_list) - 1) in
   let prog_str = List.fold_left2 (fun prog_str expr_pair index ->
    let (expr, expr_type) = expr_pair in
    if index != ((List.length expr_list) - 1) then (
     if index = 0 then (
      (match (List.nth expr_types (index + 1))
       with Int -> "((JSONArray) " ^ id ^ ".get(" ^ handle_expression (expr) ^ "))"
       | String -> "((JSONObject) " ^ id ^ ".get(" ^ handle_expression(expr) ^ "))"
       | _ -> raise (Failure "Should not have a non-Int or String JSON selector. Check semantic.")
      )
     )
     else (
      (match (List.nth expr_types (index + 1))
       with Int -> "((JSONArray) " ^ prog_str ^ ".get(" ^ handle_expression (expr) ^ "))"
       | String -> "((JSONObject) " ^ prog_str ^ ".get(" ^ handle_expression(expr) ^ "))"
       | _ -> raise (Failure "Should not have a non-Int or String JSON selector. Check semantic.")
      )
     )
    )
    else (
     cast_json_access (prog_str ^ ".get(" ^ handle_expression (expr) ^ ")") data_type
    )
```

```ocaml
      ) "" expr_type_pairs expr_num_range in
      prog_str

let rec handle_bool_expr (expr : Jast.bool_expr) = match expr
  with Literal_bool(i) ->
    (match i
      with "True" -> "true"
      | "False" -> "false"
      | _ -> "bad")
  | Binop(left_expr, op, right_expr) -> (match op, right_expr
    with (Equal, Literal_string(s)) -> "(" ^ (handle_expression left_expr) ^ ").equals(\"" ^ s ^ "\")"
    | _ -> (handle_expression left_expr) ^ " " ^ (convert_bool_operator op) ^ " " ^ (handle_expression right_expr)
  )
  | Bool_binop(left_expr, cond, right_expr) ->
    (handle_bool_expr left_expr) ^ " " ^ (convert_cond_op cond) ^ " " ^ (handle_bool_expr right_expr)
  | Not(expr) ->
    "!" ^ (handle_bool_expr expr)
  | Id(i) -> i

let rec comma_separate_arg_list (arg_decl_list : Jast.arg_decl list) = match arg_decl_list
  with [] -> ""
  | head :: arg_decls ->
    if List.length arg_decls != 0 then
      head.var_type ^ " " ^ head.var_name ^ "," ^ (comma_separate_arg_list arg_decls)
    else
      head.var_type ^ " " ^ head.var_name

let remove_semicolon (str : string) =
  Str.global_replace (Str.regexp_string ";") "" str
```

```

let rec handle_statement (stmt : Jast.stmt) (prog_string : string) (var_string: string) (func_string : string) (in_block : bool) = match stmt
  with Expr(expr) ->
    let expr_string = handle_expression expr in
    let new_prog_string = prog_string ^ expr_string ^ ";" in
    (new_prog_string, var_string, func_string)
  | Assign(data_type, id, expr) ->
    let expr_string = handle_expression expr in
    if not in_block then (
      let declare_string = "static " ^ data_type ^ " " ^ id ^ ";\n" in
      let assign_string = id ^ " = " ^ expr_string ^ ";" in
      (prog_string ^ assign_string, var_string ^ declare_string, func_string)
    )
    else (
      let assign_string = data_type ^ " " ^ id ^ " = " ^ expr_string ^ ";\n" in
      (prog_string ^ assign_string, var_string, func_string)
    )
  | Array_assign(expected_data, id, e1) ->
    if not in_block then (
      let new_var_string = var_string ^ "static " ^ expected_data ^ "[] " ^ id ^ " = {" ^ (comma_separate_list (e1)) ^ "};" in
      (prog_string, new_var_string, func_string)
    ) else (
      let new_prog_string = prog_string ^ expected_data ^ "[] " ^ id ^ " = {" ^ (comma_separate_list (e1)) ^ "};" in
      (new_prog_string, var_string, func_string)
    )
  | Fixed_length_array_assign(expected_data, id, length) ->
      if not in_block then (
      let new_var_string = var_string ^ "static " ^ expected_data ^ "[] " ^ id ^ " = new " ^ expected_data ^ "[" ^ string_of_int length ^ "];\n" in
      (prog_string, new_var_string, func_string)
    ) else (
      let new_prog_string = prog_string ^ expected_data ^ "[] " ^ id ^ " = new " ^ expected_data ^ "[" ^ string_of_int length ^ "];\n" in
      (new_prog_string, var_string, func_string)
    )
  | Jast.Update_variable(id, expr) ->
    let new_prog_string = prog_string ^ id ^ " = " ^ (handle_expression (expr)) ^ ";\n" in
    (new_prog_string, var_string, func_string)
  | Jast.Update_array_element(id, e1, e2) ->
    let new_prog_string = prog_string ^ id ^ "[" ^ (handle_expression(e1)) ^ "] = " ^ (handle_expression(e2)) ^ ";\n" in
    (new_prog_string, var_string, func_string)
  | Bool_assign(id, expr) ->
    (* Why can't we reassign to a boolean? Seems broken *)
    let new_prog_string = prog_string ^ "boolean " ^ id ^ " = " ^ (handle_bool_expr expr) ^ ";" in
    (new_prog_string, var_string, func_string)
  | Func_decl(id, arg_decl_list, return_type, body) ->
    let (prog, var, func) = handle_statements body "" "" "" true in
    let new_func_string = func_string ^ "public static " ^ return_type ^ " " ^ id ^ "(" ^ comma_separate_arg_list arg_decl_list ^ ")" ^ "{\n" ^ prog ^ "}\n" in
     (prog_string, var_string, new_func_string)
  | Return(e1) ->
    let new_prog_string = prog_string ^ "return " ^ (handle_expression e1) ^ ";" in
    (new_prog_string, var_string, func_string)
  | If(condition, body, else_body) ->
    let (prog, var, func) = handle_statements body "" "" "" true in
    let (else_prog, else_var, else_func) = handle_statements else_body "" "" "" true in
    let new_prog_string = prog_string ^ "if (" ^ handle_bool_expr condition ^ ") {" ^ prog ^ "} else {\n" ^ else_prog ^ "}" in
    (new_prog_string, var_string, func_string)
  | While(condition, body) ->
    let (prog, var, func) = handle_statements body "" "" "" true in
    let new_prog_string = prog_string ^ "while (" ^ handle_bool_expr condition ^ ") {" ^ prog ^ "}\n" in
    (new_prog_string, var_string, func_string)
  | For(init, condition, update, body)  ->
    let (init_stmt, init_var, init_func) = handle_statement init "" "" "" true in
    let condition_stmt = handle_bool_expr condition in
    let (update_stmt, update_var, update_func) = handle_statement update "" "" "" false in
    let (body_stmt, update_var, body_func) = handle_statements body "" "" "" true in
    let new_prog_string = prog_string ^
      "for (" ^ init_stmt ^ condition_stmt ^ ";" ^ remove_semicolon(update_stmt) ^ ") {\n"
          ^ body_stmt ^ "}\n" in
    (new_prog_string, var_string, func_string)
  | Where(condition, id, body, json_array) ->
    let json_var_string = handle_expression json_array in
    let json_array_declaration = "\nstaticArrr = (JSONArray)" ^ json_var_string ^ ";\n" in
    let it_declaration = "staticItt = staticArrr.iterator();\n" in
    let while_declaration = "while(staticItt.hasNext()){ \n" in
    let elem_declaration = "JSONObject " ^ id ^ " = (JSONObject) staticItt.next(); \n" in
    let if_declaration = "if(" ^ (handle_bool_expr condition) ^ ") { \n" in
    let (body_declaration, update_var, body_func) = handle_statements body "" "" "" true in
    let closing_braces_declaration = "} \n }" in

    let new_prog_string = (prog_string ^ json_array_declaration ^ it_declaration ^ while_declaration ^ elem_declaration ^ if_declaration ^
body_declaration ^ closing_braces_declaration) in
    (new_prog_string, var_string, func_string)
  | _ ->
    (prog_string, var_string, func_string)

and handle_statements (stmt_list : Jast.program) (prog_string : string) (var_string: string) (func_string : string) (in_block : bool)  = match stmt_list
    with [] -> (prog_string, var_string, func_string)
  | [stmt] ->
    let (prog, var, func) = (handle_statement stmt prog_string var_string func_string in_block) in
    (prog ^ "\n", var, func)
  | stmt :: other_stmts ->
      let (new_prog_string, new_var_string, new_func_string) = (handle_statement stmt (prog_string ^ "\n") var_string func_string in_block) in
      handle_statements other_stmts new_prog_string new_var_string new_func_string in_block

let print_to_file (prog_str : string) (file_name : string) =
  let file = (open_out file_name) in
    Printf.fprintf file "%s" prog_str;;

let program_header (class_name : string) =
  let header_string = "
 import java.io.FileReader;\n
 import java.util.Iterator;\n
 import org.json.simple.JSONArray;\n
 import org.json.simple.JSONObject;\n
 import org.json.simple.parser.JSONParser;\n" in
  let prog_string  = header_string ^ "public class " ^ class_name ^ " { \n
   public static JSONArray staticArrr;\n
   public static Iterator staticItt;\n
   " in
  prog_string
```

```ocaml

let main_method = "public static void main(String[] args) {
     try {
   "

let program_footer = " } catch (Exception e) {\ne.printStackTrace();\n}\n}"

let generate_code (program : Jast.program) (file_name : string) =
  let (prog, var, funcs) = handle_statements program "" "" "" false in
  let final_program = (program_header file_name) ^ var ^ main_method ^ prog ^ program_footer ^ funcs ^ " \n}" in
  let java_program_name = file_name ^ ".java" in
  print_to_file final_program java_program_name;;
```

compiler/Makefile

```makefile
#
# QL
#
# Manager: Matthew Piccolella
# Systems Architect: Anshul Gupta
# Tester: Evan Tarrh
# Language Guru: Gary Lin
# Systems Integrator: Mayank Mahajan

OCAMLC=ocamlc
OCAMLDEP=ocamldep
OCAMLLEX=ocamllex
OCAMLYACC=ocamlyacc
OBJS=ast.cmo parser.cmo scanner.cmo environment.cmo semantic.cmo jast.cmo semantic_to_jast.cmo code_generator.cmo qlc.cmo

qlc: $(OBJS)
	ocamlc -o qlc str.cma $(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 scanner.ml parser.ml parser.mli *.cmo *.cmi *.out *.diff qlc

# Generated by ocamldep
# TODO: Make this generate our actual dependencies
ast.cmo :
ast.cmx :
environment.cmo : ast.cmo
environment.cmx : ast.cmx
semantic.cmo : environment.cmo ast.cmo
semantic.cmx : environment.cmx ast.cmx
sast.cmo : ast.cmo
sast.cmx : ast.cmx
jast.cmo : ast.cmo
jast.cmx : ast.cmx
parser.cmo : ast.cmo parser.cmi
parser.cmx : ast.cmx parser.cmi
qlc.cmo : scanner.cmo parser.cmi ast.cmo semantic.cmo jast.cmo
semantic_to_jast.cmo code_generator.cmo
qlc.cmx : scanner.cmx parser.cmx ast.cmx semantic.cmx jast.cmx
```

```
semantic_to_jast.cmx code_generator.cmx
scanner.cmo : parser.cmi
scanner.cmx : parser.cmx
parser.cmi : ast.cmo
```

compiler/qlc.ml

```
(*
* QL
*
* Manager: Matthew Piccolella
* Systems Architect: Anshul Gupta
* Tester: Evan Tarrh
* Language Guru: Gary Lin
* Systems Integrator: Mayank Mahajan
*)

type action = Raw | Ast | Semantic | Semantic_to_jast | Code_generator

let _ =
  if Array.length Sys.argv > 2 then
    let file_name = Sys.argv.(1) in
    let file_executable = Sys.argv.(2) in
    let file = open_in (file_name) in
    let lexbuf = Lexing.from_channel file in
    let program = Parser.program Scanner.token lexbuf in
    let symbol_table = Semantic.check_program program in
    let jast = Semantic_to_jast.convert_semantic program symbol_table in
    Code_generator.generate_code jast file_executable
  else (
    if Array.length Sys.argv > 1 then (
      print_endline "Please provide a name for your executable.";
    )
    else (
      print_endline "Please provide both a QL file and a name for your
executable.";
    )
  )
```

compiler/ql

```
#!/bin/bash
# Make sure we have a command line argument we're happy with/
if [ $# -lt 1 ]; then
 echo "Please pass a QL file name"
  exit 1
else
 if [[ $1 == *.ql ]]; then
   echo "Please do not include the .ql file extension"
    exit 1
  fi
fi

# Try to compile our file
javac -classpath ../build/json-simple-1.1.1.jar $1.java >& /dev/null
if [ $? -eq 0 ]; then
 java -classpath ../build/json-simple-1.1.1.jar:. $1
else
 echo "Compiler error"
fi
```

tests/

tests/test-array-1.ql

```
array string a = ["matt";"is";"cool";"hes";"the";"bomb"]
string matt = a[0]
print(matt)
print(a[2])
print(a[4])
```

```
tests/test-array-2-fail.ql

array float a = [1.0; 1.1; 2.4; 3.1; 6.7]
int b = a[2]

tests/test-array-3-fail.ql

array float a = [1.0; 1.1; 2.4; 3.1; 6.7]
int b = a[2][1]

tests/test-array-4-fail.ql

array float a = [1.0; 1.1; "oops"; 3.1; 6.7]

tests/test-array-5-fail.ql

array string a = [1; 7; 2; 3; 6]
tests/test-array-6-fail.ql

array int a = [1; 7; 2; 3; 6]
string myname = a[0]
tests/test-array-7.ql

array int a = array(5)
print(a[3])

tests/test-array-8.ql

array int a = array(5)
int b = length(a)
print(b)

tests/test-array-9.ql
```

```
array string a = ["matt";"is";"cool";"hes";"the";"bomb"]
a[0] = "evan"
print(a[0])
print(a[1])
print(a[2])
```

tests/test-assign-1.ql

```
int a = 5
print(a)
```

tests/test-assign-2.ql

```
int a = 5 + 5 + (4 + 2)
float b = ((5.0 + 3.3) + 1.22)
string c = "matt" + "is" + "goat"
print(a)
print(b)
print(c)
```

tests/test-assign-3-fail.ql

```
int a = 5 + 3.0
float b = 4.2 + 1
int c = "matt" + 3
```

tests/test-assign-4.ql

```
int a = 5
int b = 10 + a
int c = 10 + a + b
print(c)
```

tests/test-assign-5-fail.ql

```
int x = 10
int y = x + 4 + z
```

tests/test-assign-6-fail.ql

```
int x = 10
int x = 12
```

tests/test-bool-1.ql

```
bool b = True & False
print(b)
print(True)
```

tests/test-bool-2-fail.ql

```
int a = 1
bool b = True & a
```

tests/test-bool-3.ql

```
float a = 1.5
bool b = True & 6 < 5 | 1.5 == a
print(b)
```

tests/test-bool-4-fail.ql

```
bool a = True
a = False
```

tests/test-for-1.ql

```
int a = 10
```

```
for(int i = 1, i < 5, i = i + 1) {
    for (int j = 1, j < 5, j = j + 1) {
        print (a + i + j)
    }
}
```

tests/test-for-2-fail.ql

```
for(int a = 3 , a < 5 , a = a + 1){
    b = 1
}
```

tests/test-for-3-fail.ql

```
for(int a = 3 , a = 5 , a = a + 1){
    int b = 1
}
```

tests/test-for-4-fail.ql

```
for(a = 3 , a < 5 , a = a + 1){
    int b = 1
}
```

tests/test-for-5-fail.ql

```
for(int a = 3 , a < 5 , a + 1){
    int b = 1
}
```

tests/test-for-6.ql

```
array string arr = ["The"; "quick"; "brown"; "fox"; "jumped"; "the"; "lazy";
"dog"]
```

```
string sentence = ""

for(int k = 0 , k < 8 , k = k + 1){
    #~~ Because why not ~~#
    int temp = k

    if(temp == 7) {
        sentence = sentence + arr[temp] + "."
    } else {
        sentence = sentence + arr[temp] +  " "
    }
}

print(sentence)
```

tests/test-func_decl-1.ql

```
int a = 5
int b = a + 7
function add (int x, int y, int z) : int {
  int d = x + y + z
  int w = 5
  return d
}
print(add(5,add(1,2,3),5))
```

tests/test-func_decl-2.ql

```
array string mylist = ["If you "; "see this "; "call 732-600-5766"]
function concat (string s1, string s2, string s3) : string {
    return s1 + s2 + s3
}
print(concat(mylist[0], mylist[1], mylist[2]))
```

```
tests/test-func_decl-3.ql

float a = 4.0e2
float b = a + 7.5e-1
function add (float x, float y, float z) : float {
  return x + y + z
}

print(add(1.0e1,b,2.0e-2))

tests/test-func_decl-4.ql

function concat (int a) : bool {
    return True
}

print(concat(1))

tests/test-function-1.ql

int a = 5
int b = a + 7
function increment(int c, int amount) : int {
  int x = c + amount + 1
  int y = x + c + 20
  return y
}
int x = increment(a, b)
print(x)

tests/test-function-2-fail.ql

function multiply(int x) : int {
}
```

```
tests/test-function-3-fail.ql

function sort(int y) : array {
  int x = y + 5
}
int z = y + 4

tests/test-function-4.ql

int a = 5
int b = a + 7
function add(int x, int y, float z) : int {
  int d = x + y
  return d
}
print(add(a,b,4.55))

tests/test-function-5-fail.ql

function myStringPrint(string s) : int {
  int d = 5
  return d
}
myStringPrint(5)

tests/test-function-6.ql

int a = 5
int b = a + 7
function add(int x, int y, float z) : int {
  int d = x + y + a
  return d
}
```

```
int z = add(5,add(1,2,3.3),5.5)
print(z)
```

tests/test-function-7-fail.ql

```
float a = 5.5
float b = a + 10.0
function multiply(float x, float y) : float {
  float z = x * y
  return z
}
function add(int x, int y) : int {
  return x + y
}
int x = 1
add(x, multiply(a,b))
```

tests/test-function-8.ql

```
function whosTheBest() : int {
  print("Evan")
  return 0
}

whosTheBest()
```

tests/test-hello-1.ql

```
print("Hello world, this is QL.")
```

tests/test-hello-2.ql

```
print(1234)
```

tests/test-ifelse-1.ql

```
if (5 > 4) {
  int a = 5
  a = 2
} else {
  int b = 2
  b = 1
}
```

tests/test-ifelse-2-fail.ql

```
int b = 5
if (5 > 4) {
  b = 5.5
} else {
  b = 10
}
```

tests/test-ifelse-3-fail.ql

```
int b = 5
if ("hello" > 8) {
  b = 8
} else {
  b = 10
}
```

tests/test-ifelse-4.ql

```
int b = 5
if (b > 8) {
  b = 8
} else {
  b = 10
}
print(b)
```

tests/test-ifelse-5.ql

```
int a = 1
if (5 > 4) {
  a = 2
} else {
  a = 3
}
print(a)
```

tests/test-ifelse_print-1.ql

```
int a = 5
if (a < 5) {
    print("a is so small!")
} else {
    print("a is so big!")
}
```

tests/test-ifelse_print-2.ql

```
int a = 5
if (a < 5) {
    print("a is so small!")
}
else {
}
```

tests/test-ifelse_print-3.ql

```
string success = "You did it!"
int tries = 0
for (tries = tries, tries < 3, tries = tries + 1) {
    if (tries == 2) {
        print(success)
    } else {
    print("try again, good sir")
    }
}
```

tests/test-ifelse_print-4.ql

```
int a = 8

if (a < 10) {
    if (a < 6) {
        print("a is so small!")
    }
    else {
        print("a is just kinda small")
    }
}
```

```
else {
        if (a > 40) {
            print("A IS GARGANTUAN")
        }
        else {
            print("a is just kinda big")
        }
}
```

tests/test-json-1.ql

```
json a = json("sample.json")
int b = a["count"]
print(b)
float d = a["number"]
print(d)
string name = a["owner"]
print(name)
```

tests/test-json-2.ql

```
json a = json("sample.json")
int b = a["friends"][1]["age"]
print(b)
string c = a["friends"][2]["name"]
print(c)
```

tests/test-json-3-fail.ql

```
json a = json("my_json")
json b = json("my_json2")
a = json("my_json3")
```

tests/test-json-4.ql

```
json a = json("sample.json")
int b = a["count"]
print(b)
int c = a["int_index"]
print(c)
```

tests/test-json-5.ql

```
json a = json("sample.json")
if (a["count"] > 0) {
    print("Stephen is a techer")
}
```

tests/test-json-6.ql

```
json a = json("sample.json")
int b = a["int_index"] + a["count"]
print(b)
```

tests/test-json-7-fail.ql

```
json a = json("file.json")
int x = a["count"]
float y = a["count"]
```

tests/test-json-8-fail.ql

```
json a = json("my_json")
int b = a[2]["matt"][2.3]
```

tests/test-json-selector-1.ql

```
json a = json("sample.json")
```

```
array int b = a["sum"]
array int c = a["sum"]
```

tests/test-json-typing-1-fail.ql

```
json a = json("my_file")
int x = 1 + 2 + 3 + 4 + 5 + a["num1"]
float c = a["num1"]
```

tests/test-json-typing-10.ql

```
json a = json("sample.json")
if (a["count"] > 4) {
  print("hello")
}
int f = a["count"]
```

tests/test-json-typing-11.ql

```
json m = json("sample.json")
for(int a = 3 , m["count"] < 5 , a = a + 1) {
  int b = 1
}

int x = m["count"]
print(x)
```

tests/test-json-typing-12-fail.ql

```
json c = json("hello.json")
for(int a = 3 , c["count"] < 5.5 , a = a + 1){
  int b = 1
}
```

```
int x = c["count"]
```

tests/test-json-typing-13.ql

```
json a = json("sample.json")
array json fr = a["friends"]
```

tests/test-json-typing-2-fail.ql

```
json a = json("my_file")
string x = "matt" + "is" + a["num1"]
float y = 1.0 + a["num2"]
int z = a["num1"]
```

tests/test-json-typing-3-fail.ql

```
function increment(int c) : int {
  int x = c + 10
  return x
}
json a = json("hello.json")
int z = increment(a["matt"])
float f = a["matt"]
```

tests/test-json-typing-4-fail.ql

```
function increment(int c) : int {
  int x = c + 10
  return x
}
json a = json("dude.json")
increment(a["dude"])
string f = a["dude"]
```

tests/test-json-typing-5-fail.ql

```
function increment(int c) : int {
  int x = c + 10
  return x
}
json a = json("hello.json")
increment(a["matt"])
float f = 2.0
f = a["matt"]
```

tests/test-json-typing-6-fail.ql

```
json a = json("hello.json")
if (a["matt"] > 5) {
  print("hello")
}
string f = a["matt"]
```

tests/test-json-typing-7-fail.ql

```
json a = json("hello.json")
if (a["count1"] > a["count2"]) {
  print("hello")
}
string f = a["count1"]
```

tests/test-json-typing-8-fail.ql

```
json a = json("hello.json")
string x = "hello" + a["matt1"]
bool y = a["matt1"] != 5.0
```

tests/test-json-typing-9.ql

```
json a = json("sample.json")
array float y = [4.3; 5.0; 1.2]
float z = y[a["int_index"]]
int xx = a["int_index"]
print(z)
print(xx)
```

```
tests/test-math-1.ql

float x = 1e1 * 1e1
print(x)
```

```
tests/test-scoping-1.ql

int c = 5

function add(int x, int y) : int {
  int b = 5
  return x + y + c
}

int y = add(10,15)
print(y)
```

```
tests/test-scoping-2-fail.ql

function add(int x, int y) : int {
  int c = 5
  return x + y + c
}

int y = add(10,15)
print(c)
```

tests/test-scoping-3-fail.ql

```
int a = 5

function test(int x, int y) : int {
  int a = 10
  return x
}
```

tests/test-updatevar-1.ql

```
int a = 5 + 5 + (4 + 2)
float b = ((5.0 + 3.3) + 1.22)
string c = "matt" + "is" + "goat"
print(a)
print(b)
print(c)
a = 7
b = 5.5
c = "lol"
print(a)
print(b)
print(c)
```

tests/test-updatevar-2-fail.ql

```
int a = 5 + 5 + (4 + 2)
float b = ((5.0 + 3.3) + 1.22)
string c = "matt" + "is" + "goat"

a = 7.5
```

tests/test-updatevar-3-fail.ql

```
array int a = [6;5;5]
#~~ Array literals can only be used instantiation ~~#
a = [2;4]
```

tests/test-where-syntax-1.ql

```
json test = json("sample.json")

where (True) as dummyVar {

} in test["friends"]
```

tests/test-where-syntax-2.ql

```
json test = json("sample.json")

#~~ Prints all names whose ages are greater than 20 ~~#

where (elem["age"] > 20) as elem {
  string s = elem["name"]
  print(s)
} in test["friends"]
```

tests/test-where-syntax-3-fail.ql

```
where () as var {
  int a = 1
} in json("test.json")
```

tests/test-where-syntax-4.ql

```
json djk = json("sample2.json")

#~~ Prints all names whose ages are greater than 20 ~~#
```

```
where (True) as sayings {
  string s = sayings["quote"]
  print(s)
} in djk["quotes"]

where (show["count"] < 5) as show {
  string s = show["city"]
  int month = show["dates"][1]["month"]
  int day = show["dates"][1]["day"]
  int year = show["dates"][1]["year"]
  if(year == 2016) {
      print("city:")
      print(s)
      print("month:")
      print(month)
      print("day:")
      print(day)
      print("year:")
      print(year)
  }
} in djk["shows"]
```

tests/test-while-1.ql

```
int a = 1

while (a < 5) {
    a = a + 1
    int temp = 1
    while (temp <= a) {
        print(a)
        temp = t
```

tests/test-array-1.ql

```
array string a = ["matt";"is";"cool";"hes";"the";"bomb"]
string matt = a[0]
print(matt)
print(a[2])
print(a[4])
```

tests/test-array-2-fail.ql

```
array float a = [1.0; 1.1; 2.4; 3.1; 6.7]
int b = a[2]
```

tests/test-array-3-fail.ql

```
array float a = [1.0; 1.1; 2.4; 3.1; 6.7]
int b = a[2][1]
```

tests/test-array-4-fail.ql

```
array float a = [1.0; 1.1; "oops"; 3.1; 6.7]
```

tests/test-array-5-fail.ql

```
array string a = [1; 7; 2; 3; 6]
```

tests/test-array-6-fail.ql

```
array int a = [1; 7; 2; 3; 6]
string myname = a[0]
```

tests/test-array-7.ql

```
array int a = array(5)
print(a[3])
```

tests/test-array-8.ql

```
array int a = array(5)
int b = length(a)
print(b)
```

tests/test-array-9.ql

```
array string a = ["matt";"is";"cool";"hes";"the";"bomb"]
a[0] = "evan"
print(a[0])
print(a[1])
print(a[2])
```

tests/test-assign-1.ql

```
int a = 5
print(a)
```

tests/test-assign-2.ql

```
int a = 5 + 5 + (4 + 2)
float b = ((5.0 + 3.3) + 1.22)
string c = "matt" + "is" + "goat"
print(a)
print(b)
print(c)
```

tests/test-assign-3-fail.ql

```
int a = 5 + 3.0
float b = 4.2 + 1
int c = "matt" + 3
```

tests/test-assign-4.ql

```
int a = 5
int b = 10 + a
int c = 10 + a + b
print(c)
```

tests/test-assign-5-fail.ql

```
int x = 10
int y = x + 4 + z
```

tests/test-assign-6-fail.ql

```
int x = 10
int x = 12
```

tests/test-bool-1.ql

```
bool b = True & False
print(b)
print(True)
```

tests/test-bool-2-fail.ql

```
int a = 1
bool b = True & a
```

tests/test-bool-3.ql

```
float a = 1.5
bool b = True & 6 < 5 | 1.5 == a
print(b)
```

tests/test-bool-4-fail.ql

```
bool a = True
a = False
```

tests/test-for-1.ql

```
int a = 10
for(int i = 1, i < 5, i = i + 1) {
    for (int j = 1, j < 5, j = j + 1) {
        print (a + i + j)
    }
}
```

tests/test-for-2-fail.ql

```
for(int a = 3 , a < 5 , a = a + 1){
    b = 1
}
```

tests/test-for-3-fail.ql

```
for(int a = 3 , a = 5 , a = a + 1){
    int b = 1
}
```

tests/test-for-4-fail.ql

```
for(a = 3 , a < 5 , a = a + 1){
    int b = 1
}
```

tests/test-for-5-fail.ql

```
for(int a = 3 , a < 5 , a + 1){
    int b = 1
}
```

tests/test-for-6.ql

```
array string arr = ["The"; "quick"; "brown"; "fox"; "jumped"; "the"; "lazy";
"dog"]
string sentence = ""

for(int k = 0 , k < 8 , k = k + 1){
    #~~ Because why not ~~#
    int temp = k

    if(temp == 7) {
        sentence = sentence + arr[temp] + "."
    } else {
        sentence = sentence + arr[temp] +  " "
    }
}

print(sentence)
```

tests/test-func_decl-1.ql

```
int a = 5
int b = a + 7
function add (int x, int y, int z) : int {
  int d = x + y + z
  int w = 5
  return d
}
print(add(5,add(1,2,3),5))
```

tests/test-func_decl-2.ql

```
array string mylist = ["If you "; "see this "; "call 732-600-5766"]
function concat (string s1, string s2, string s3) : string {
    return s1 + s2 + s3
}
print(concat(mylist[0], mylist[1], mylist[2]))
```

tests/test-func_decl-3.ql

```
float a = 4.0e2
float b = a + 7.5e-1
function add (float x, float y, float z) : float {
  return x + y + z
}

print(add(1.0e1,b,2.0e-2))
```

tests/test-func_decl-4.ql

```
function concat (int a) : bool {
    return True
}

print(concat(1))
```

tests/test-function-1.ql

```
int a = 5
int b = a + 7
function increment(int c, int amount) : int {
  int x = c + amount + 1
  int y = x + c + 20
  return y
}
int x = increment(a, b)
print(x)

tests/test-function-2-fail.ql

function multiply(int x) : int {
}

tests/test-function-3-fail.ql

function sort(int y) : array {
  int x = y + 5
}
int z = y + 4

tests/test-function-4.ql

int a = 5
int b = a + 7
function add(int x, int y, float z) : int {
  int d = x + y
  return d
}
print(add(a,b,4.55))

tests/test-function-5-fail.ql

function myStringPrint(string s) : int {
  int d = 5
  return d
}
myStringPrint(5)
```

tests/test-function-6.ql

```
int a = 5
int b = a + 7
function add(int x, int y, float z) : int {
  int d = x + y + a
  return d
}
int z = add(5,add(1,2,3.3),5.5)
print(z)
```

tests/test-function-7-fail.ql

```
float a = 5.5
float b = a + 10.0
function multiply(float x, float y) : float {
  float z = x * y
  return z
}
function add(int x, int y) : int {
  return x + y
}
int x = 1
add(x, multiply(a,b))
```

tests/test-function-8.ql

```
function whosTheBest() : int {
  print("Evan")
  return 0
}

whosTheBest()
```

tests/test-hello-1.ql

```
print("Hello world, this is QL.")
```

tests/test-hello-2.ql

```
print(1234)
```

tests/test-ifelse-1.ql

```
if (5 > 4) {
  int a = 5
  a = 2
} else {
  int b = 2
  b = 1
}
```

tests/test-ifelse-2-fail.ql

```
int b = 5
if (5 > 4) {
  b = 5.5
} else {
  b = 10
}
```

tests/test-ifelse-3-fail.ql

```
int b = 5
if ("hello" > 8) {
  b = 8
} else {
  b = 10
}
```

tests/test-ifelse-4.ql

```
int b = 5
if (b > 8) {
  b = 8
} else {
  b = 10
}
print(b)
```

tests/test-ifelse-5.ql

```
int a = 1
if (5 > 4) {
  a = 2
} else {
  a = 3
}
print(a)
```

tests/test-ifelse_print-1.ql

```
int a = 5
if (a < 5) {
    print("a is so small!")
} else {
    print("a is so big!")
}
```

tests/test-ifelse_print-2.ql

```
int a = 5
```

```
if (a < 5) {
    print("a is so small!")
}
else {








}
```

tests/test-ifelse_print-3.ql

```
string success = "You did it!"
int tries = 0
for (tries = tries, tries < 3, tries = tries + 1) {
    if (tries == 2) {
        print(success)
    } else {
    print("try again, good sir")
    }
}
```

tests/test-ifelse_print-4.ql

```
int a = 8
```

```
if (a < 10) {
    if (a < 6) {
        print("a is so small!")
    }
    else {
        print("a is just kinda small")
    }
}
else {
        if (a > 40) {
            print("A IS GARGANTUAN")
        }
        else {
            print("a is just kinda big")
        }
}
```

tests/test-json-1.ql

```
json a = json("sample.json")
int b = a["count"]
print(b)
float d = a["number"]
print(d)
string name = a["owner"]
print(name)
```

tests/test-json-2.ql

```
json a = json("sample.json")
int b = a["friends"][1]["age"]
print(b)
string c = a["friends"][2]["name"]
print(c)
```

tests/test-json-3-fail.ql

```
json a = json("my_json")
json b = json("my_json2")
a = json("my_json3")
```

tests/test-json-4.ql

```
json a = json("sample.json")
int b = a["count"]
print(b)
int c = a["int_index"]
print(c)
```

tests/test-json-5.ql

```
json a = json("sample.json")
if (a["count"] > 0) {
    print("Stephen is a techer")
}
```

tests/test-json-6.ql

```
json a = json("sample.json")
int b = a["int_index"] + a["count"]
print(b)
```

tests/test-json-7-fail.ql

```
json a = json("file.json")
int x = a["count"]
float y = a["count"]
```

```
tests/test-json-8-fail.ql

json a = json("my_json")
int b = a[2]["matt"][2.3]

tests/test-json-selector-1.ql

json a = json("sample.json")
array int b = a["sum"]
array int c = a["sum"]

tests/test-json-typing-1-fail.ql

json a = json("my_file")
int x = 1 + 2 + 3 + 4 + 5 + a["num1"]
float c = a["num1"]

tests/test-json-typing-10.ql

json a = json("sample.json")
if (a["count"] > 4) {
  print("hello")
}
int f = a["count"]

tests/test-json-typing-11.ql

json m = json("sample.json")
for(int a = 3 , m["count"] < 5 , a = a + 1) {
  int b = 1
}

int x = m["count"]
print(x)
```

tests/test-json-typing-12-fail.ql

```
json c = json("hello.json")
for(int a = 3 , c["count"] < 5.5 , a = a + 1){
  int b = 1
}

int x = c["count"]
```

tests/test-json-typing-13.ql

```
json a = json("sample.json")
array json fr = a["friends"]
```

tests/test-json-typing-2-fail.ql

```
json a = json("my_file")
string x = "matt" + "is" + a["num1"]
float y = 1.0 + a["num2"]
int z = a["num1"]
```

tests/test-json-typing-3-fail.ql

```
function increment(int c) : int {
  int x = c + 10
  return x
}
json a = json("hello.json")
int z = increment(a["matt"])
float f = a["matt"]
```

tests/test-json-typing-4-fail.ql

```
function increment(int c) : int {
  int x = c + 10
  return x
}
json a = json("dude.json")
increment(a["dude"])
string f = a["dude"]
```

tests/test-json-typing-5-fail.ql

```
function increment(int c) : int {
  int x = c + 10
  return x
}
json a = json("hello.json")
increment(a["matt"])
float f = 2.0
f = a["matt"]
```

tests/test-json-typing-6-fail.ql

```
json a = json("hello.json")
if (a["matt"] > 5) {
  print("hello")
}
string f = a["matt"]
```

tests/test-json-typing-7-fail.ql

```
json a = json("hello.json")
if (a["count1"] > a["count2"]) {
  print("hello")
}
string f = a["count1"]
```

tests/test-json-typing-8-fail.ql

```
json a = json("hello.json")
string x = "hello" + a["matt1"]
bool y = a["matt1"] != 5.0
```

tests/test-json-typing-9.ql

```
json a = json("sample.json")
array float y = [4.3; 5.0; 1.2]
float z = y[a["int_index"]]
int xx = a["int_index"]
print(z)
print(xx)
```

tests/test-math-1.ql

```
float x = 1e1 * 1e1
print(x)
```

tests/test-scoping-1.ql

```
int c = 5

function add(int x, int y) : int {
  int b = 5
  return x + y + c
}

int y = add(10,15)
print(y)
```

tests/test-scoping-2-fail.ql

```
function add(int x, int y) : int {
  int c = 5
  return x + y + c
}

int y = add(10,15)
print(c)
```

tests/test-scoping-3-fail.ql

```
int a = 5

function test(int x, int y) : int {
  int a = 10
  return x
}
```

tests/test-updatevar-1.ql

```
int a = 5 + 5 + (4 + 2)
float b = ((5.0 + 3.3) + 1.22)
string c = "matt" + "is" + "goat"
print(a)
print(b)
print(c)
a = 7
b = 5.5
c = "lol"
print(a)
print(b)
print(c)
```

tests/test-updatevar-2-fail.ql

```
int a = 5 + 5 + (4 + 2)
float b = ((5.0 + 3.3) + 1.22)
string c = "matt" + "is" + "goat"

a = 7.5
tests/test-updatevar-3-fail.ql

array int a = [6;5;5]
#~~ Array literals can only be used instantiation ~~#
a = [2;4]
tests/test-where-syntax-1.ql

json test = json("sample.json")

where (True) as dummyVar {

} in test["friends"]

tests/test-where-syntax-2.ql

json test = json("sample.json")

#~~ Prints all names whose ages are greater than 20 ~~#

where (elem["age"] > 20) as elem {
  string s = elem["name"]
  print(s)
} in test["friends"]

tests/test-where-syntax-3-fail.ql

where () as var {
  int a = 1
} in json("test.json")

tests/test-where-syntax-4.ql

json djk = json("sample2.json")

#~~ Prints all names whose ages are greater than 20 ~~#

where (True) as sayings {
  string s = sayings["quote"]
  print(s)
} in djk["quotes"]

where (show["count"] < 5) as show {
  string s = show["city"]
  int month = show["dates"][1]["month"]
  int day = show["dates"][1]["day"]
  int year = show["dates"][1]["year"]
  if(year == 2016) {
      print("city:")
      print(s)
      print("month:")
      print(month)
      print("day:")
      print(day)
      print("year:")
      print(year)
  }
} in djk["shows"]

tests/test-while-1.ql

int a = 1
```

```
while (a < 5) {
    a = a + 1
    int temp = 1
    while (temp <= a) {
        print(a)
        temp = t
    }
}
```

tests/test-while-2-fail.ql

```
int a = 1
while (a = a + 1) {
    a = a + 2
}
```

tests/test-while-3.ql

```
int a = 1
string s = "*"
while (a < 5) {
    s = s + "*"
    a = a + 1
}
print (s)
```

tests/test-while-4-fail.ql

```
int b = 1
while (a < 5) {
    b = b + 1
}
```