WebLang Final Report

Christophe Rimann
cjr2185

Ryan Bernstein
rb3234

Jordan Vega
jmv2177

Brendan Burke
btb2121

Julian Serra
jjs2269

Columbia University

COMS-W4115
December 21, 2017

Contents

1. **Introduction**

Weblang is an imperative programming language designed to simplify interactions with RESTful APIs. It provides users with the ability to make structures for specific applications, with callable endpoints, while eliminating the hassle of authentication and identification. Moreover, it eases the combination of information gathered from multiple APIs, allowing for exciting possibilities for programs that interact with multiple APIs. This language is designed specifically to handle conventional return types from these interfaces (primarily JSON), and allows developers to process data and program with it efficiently.

WebLang utilizes a host of C and C++ libraries/wrappers to interact with servers using HTTP protocol, targeting the LLVM compiler. It is the goal of the language to allow programmers to easily and efficiently interact with RESTful applications.

1.1 Inspiration

Programming is becoming increasingly tied to the web. Most recent, relevant, and useful applications that we use in our everyday lives have some sort of web interaction through HTTP protocol. These servers, growing increasingly reliant on one another, are effectively sharing information to provide users with richer services. The de facto standard for much of this communication is Representational state transfer, or REST. These RESTful APIs are becoming – in many cases – more important than the websites they help serve [1]. Many successful companies, such as IFTTT, Zapier, and Microsoft Flow, have made entire businesses out of doing nothing but connecting different APIs [2] [3] [4]. These seamless interactions between different web services begin to approximate the overall inspiration behind weblang, but with the goal of reducing the rigidity with which these connections are implemented. WebLang aims to provide programmers with a simple tool that consolidates interaction and facilitates consumption of RESTful applications.

1.2 Target Users

The target users of WebLang are developers seeking a streamlined method of incorporating RESTful APIs into their applications. These may be either developers experienced with using RESTful APIs who have grown tired of tedious authentication and authorization procedures, or developers looking for a hands-off introduction to RESTful APIs. Weblang developers should have prior experience interacting with JSON formatted data.

2. Language Usage Tutorial

Before users begin programming in Weblang they should follow this brief tutorial, which explains how to setup the weblang environment and some language basics. For a more in-depth explanation of language specifics, refer to the **Language Reference Manual** in section 3 of this document.

2.1 Setting up your Environment

1. Clone the Weblang Repository with the following command:

```
git clone https://github.com/rybern/plt.git
```

2. Install haskell-stack by following these directions.

   - If using Mac OSX with Homebrew installed, simply enter the following into the command line:

   ```
    brew install haskell—stack
   ```

3. Install the Nix package manager. Now when using a stack command such as *stack build* or *stack exec*, instead use *stack build –nix* or *stack exec –nix*

4. Install LLVM 4.

```
$ nix—shell
$ cabal new—build llvm—hs
```

5. In this directory, run the following commands:

```
stack ——install—ghc
stack build
stack exec weblang
```

2.2 Hello World!

Something consistent among all programming language introductions is the *Hello World!* program example. This is how to execute it using a simple print statement in Weblang.

```
exampleFunction arg : Str —> Str
  x = "Hello World!"
  log x
  x
```

Here we are defining a new function *exampleFunction* that takes one argument, *arg*. The function expects a String (Str) input and also returns a Str, as designated by the *Str -> Str* notation. To print the String, use the reserved word log followed by the String or variable to print.

2.3 Creating the Executable and Running the Program

Say the above Hello World example is in a file called *HelloWorld.wl*. The binary can then be created from the Command Line using the *weblang* build function as follows:

```
./weblang HelloWorld.wl
```

Once the executable is created, the function may be called from the Command Line by running the executable followed by the function name and argument as follows:

```
./HelloWorld exampleFunction hi
```

Note that the argument passed into this particular function is not actually used, so the string ”hi” could be replace by any valid string.

2.4 Function Arguments

All Weblang functions take in exactly one argument, but if multiple arguments are needed they can easily be passed in via an array. For example, a program to find the sum of two numbers would look like this:

```
gcdExample arg : Arr —> Num
  sum = (arg.[0] + arg.[0])  // Accessing and adding the two numbers
  log sum                    // Printing the result to the console
  sum                        // Returning the result as a Num
```

The two numbers were passed in as an array of length two, where they were then accessed according to their location in the array using Weblang dot-indexing notation. Weblang functions return whatever is declared or assigned on the last line of the function, so sum was rewritten once more to ensure it is returned.

2.5 Example Program Connecting to an Endpoint

One of the most functional applications of Weblang is the ability to easily interact with RESTful APIs. The following example shows how to use the import reserved word to connect to the *gdax* cryptocurrency trading API and retrieve the current price of Bitcoin. Within the import statement the developer specifies the url of the API, any keys, secrets, or headers needed, and then the details of the target endpoint. In this example, the developer is connecting to the ”btc-usd/ticker” endpoint of the gdax api, specifying the the request is not a post request, and ultimately assigning the entire command to a function *getBitcoinPrice*. Calling this function will make a request to the specified API and return the corresponding JSON data.

Now that there is a way of attaining this data, the developer can create functions to interact with it. The *bitcoin* function defined below takes in a Str argument, calls the getBitcoinPrice function to pull JSON data from the API as a String, and converts that string to a JSON object with the built-in *jn* function. Having the data as a JSON object enables the built-in *get* function to pull out the value associated with the key "price" and assign it to the variable *precio*. Lastly, the variable *precio* is rewritten on the last line of the function to ensure that its value is returned.

```
import {url: "https://api.gdax.com/products/", key:"", secret:"", header:"",
  endpoints:[{fnName:"getBitcoinPrice", endpoint:"btc-usd/ticker", is_post: false}]}

bitcoin arg : Str -> Str
  x = getBitcoinPrice arg
  res = jn x
  precio = (get [res, "price"])
  precio
```

3. Language Reference Manual

3.1 Lexical Conventions

3.1.1 Identifiers

Identifiers are used to name functions, types and variables. These use ASCII letters [A-Z, a-z], the underscore character, and decimals, but they must start with an ASCII letter. Identifiers are case sensitive, which means that an identifier such as random_api is treated as a different identifier from Random_api. Moreover, identifiers must not be equivalent to any of our reserved keywords (listed in the following section) as, naturally, the use of these keywords would result in errors.

3.1.2 Reserved Keywords

WebLang has a set of specific identifiers and functions which cannot be used for by the programmer for any other purpose (such as functions or variable names). These keywords are enumerated below, and will be explored in further depth throughout the manual.

1. `helper`: A function type
2. `type`: A function type
3. `if`: control flow
4. `else`: control flow
5. `foreach`: control flow
6. `true`: boolean
7. `false`: boolean
8. `import`: utilized to declare an API, allowing it for use within the file
9. `include`: similar to C/C++ include, it gives access to functions declared within another file.

In addition, each of the names of the primitive types in section 3.1 are reserved.

3.1.3 Comments

In WebLang, one can make single-line comments, as well as multi-line comments. In a single line everything after // is a single line comment, as in the following two examples:

```
// This is a single line comment
post_joke joke_dest : String -> Nothing //this is also a comment
```

To make multiple line comments, enclose content between /* and */:

```
     /* This is a comment
     that spans several lines
     because it looks better
     like this sometimes*/
```

3.1.4 Literals

WebLang Literals are type defined values that are interpreted exactly as they are defined. There are literals for each primitive type in WebLang:

- JSON Object Literals
- JSON Array Literals
- String Literals
- Number Literals

See detailed descriptions of these primitives (section 3.1) outlining the composition of types.

3.1.5 Operators

WebLang uses the following operators that are reserved elements in the language. For more information on the function of each operator, see the operators section below.

= -> − +
∗ / % >
< == !=

3.1.6 Separators and Punctuation

Separators define scope and relations between variables, as well as start and end points for function declarations.

- {} - curly braces are utilized to define JSON objects (see Types section for more information)
- [] - brackets are utilized to define JSON Arrays, as well as for array and object access.
- Whitespace - Weblang is a whitespace delimitted language. Whitespace is used not only for separating variable and function declarations (as in most languages), it is also used for scoping. See scoping sections for more information. We consider the ASCII SP, ASCII FF, and the ASCII HT characters to be whitespace.
- New line - the new line character separates two statements, as in python. We consider ASCII LF and ASCII CR to be new line characters. An important caveat: due to the fact that JSON can often be very long and unwieldy because of large data transfers, it does not make sense to obey new line syntax with JSON. As such, the compiler will ignore all new lines and whitespace within JSON types, utilizing the separators built in to the types (for example braces/colons/commas for objects, and brackets/commas for arrays). This way, users can structure JSON in ways that make clear and intuitive sense to them.

3.2 Types

Types are primitive types paired with a predicate on that primitive type. A type with name A could be written either as A, or as A[p(val)], where p is a predicate on a value with primitive type the same as A's primitive type, and val is a locally bound name representing a constituent of A. A[p(val)] is a type with the same primitive type as A, but with an additional predicate p.

3.2.1 Primitives

The primitive type hierarchy is as follows:

Any: the most general, least descriptive type of WebLang. Every other type is an example of Any.

`JSON`: The majority of our primitives implement JSON, which is based on the JavaScript ECMA 262 specification.

`String`: Weblang utilizes ASCII strings to represent textual data. Like Javascript, a single character is treated as a single character String and we do not support a type for chars.

`Number`: A primitive corresponding to a doubleprecision 64bit binary format value

`Object`: A key-value pair container, with the key as a string and value as anything. Keys and values are separated with an equals sign, and key-value pairs are separated with commas.

`Array`: A traditional array structure.

`Bool`: A boolean representing true or false

`Null`: Absence of a value within a JSON object or array. Distinct from Nothing, which is absence of a value for overarching Any type.

`Nothing`: Absence of value within WebLang. Different from Null, which is absence of value within a JSON object or array

`Type`: A value representing a Type

Parent-children relationships in the hierarchy are is-a relationships, so a descendant can be used transparently as one of its ancestors. For example, a String can be used as a JSON value, and any value can be used as an Any.
All values in WebLang are of a primitive type, except for functions, which are of type A `->` B, where A and B are primitive types.

3.2.2 Using Types

Any type with name A can be used as A or as A[p(val)]. Additionally, some primitive container types have additional, more descriptive representations. The additional information from these representations will sometimes be checked at compile-time, and are always checked at run-time.

Array types also can be represented as:

```
[Type1, Type2, Type3]
[Type1, Type2, Type3, ...]
[Type1, Type2, Type3...]
```

The first instance represents an array with three elements, which are of Type1, Type2, and Type3 in that order.
The second instance represents an array that starts with three elements of those types, but can contain anything afterward.
The third instance represents an array that contains an element of Type1 and then of Type2, and then zero or more elements of Type3.

Object types also can be represented as:

```
{key1 : Type1, key2 : Type2, key3 : Type3}
```

This represents a JSON object that contains at least pairs with keys key1, key2, and key3 with types Type1, Type2 and Type3 respectively.

3.2.3 Function Types

When functions are declared, type annotations of the form A `->` B are required. A function is guaranteed at compile-time to be called with a value that is the same primitive type as A, and to return the same primitive type as B. In addition, at runtime, the function's input is checked against A's predicate, and the function's output is checked against B's predicate. If either predicate fails, the program will exit with an appropriate error message.

3.2.4 Types as Values

Types can be assigned to variables as values, which is to say you can represent a type through the use of an identifier.

3.2.5 Type checking

The built-in function : can be used to check if a value matches both the primitive type and the predicate of a given type, with the form [value] : [type].

For example, `123 : Object` would evaluate to false, `123 : Number[val < 10]` would evaluate to false, and `123 : Number` would evaluate to true.

3.2.6 User Defined Types

Users can define types that are derived from primitive types. There are effectively aliases on other types. Users may use the following syntax:

```
type [name—of—type] [value—name] : [already—existing—type]
  [predicate on value—name]
```

as in:

```
type A a : B
  p(a)
```

This will create a type called A and will behave as B, but with the additional predicate p(a).
For example:

```
type Integer i : Number
  integral i
```

This will create a new type Integer that is represented as a number, but will additionally check that the number is integral.

3.3 Imports and Namespaces

`import` is a built-in function that takes an API specification and authorization information, and returns a namespace with the API's endpoints available.
import has the following type:

```
import : { url : String,
           key : String,
           secret : String,
           header : [String...],
           endpoints : [Object {fnName: String, endpoint: String, is_post:
              Boolean}...],
         } -> Nothing
```

- `url` is the server address, for example "http://google.com/" or "127.0.0.1"
- `key` is the api key, if needed
- `secret` the api secret, if needed
- `header` header definitions for the http request
- `endpoints` is an array of objects where each object has an endpoint specified, whether its a post or get request, and a fnName, where the value of fnName is how can call call the endpoint in the weblang program.

`import` brings the endpoints into the current namespace. The endpoints brought into the namespace by import behave as functions, with the corresponding function type from their API specification. When an endpoint is used, a network call is made to the server at the port with the given authentication to that endpoint, with the endpoints arguments sent. Essentially, there are two ways to use API calls in WebLang: user-defined or built-in. Since API calls in of and themselves are a type, users may define custom API calls in their code. Alternatively, users can include one of the premade API calls. To do this, the user places an import statement at the beginning of the program.
An example of importing the gdax API:

```
 import {
        url: "https://api.gdax.com/products/",
        key:"",
        secret:"",
        headers:"",
        endpoints:
        [
        {fnName:"getBitcoinPrice", endpoint:"btc-usd/ticker", is_post:false},
        {fnName:"getEtherPrice", endpoint:"eth-usd/ticker", is_post:false},
        {fnName:"getLitecoinPrice", endpoint:"ltc-usd/ticker", is_post:false}
        ]
      }
```

3.4 Includes

`include` is a built-in function that takes an external Weblang file path and makes any functions from that file accessible to the current file. This can be used to include standard library functions or functions written elsewhere in a project. An example of using includes to sort an array with the standard library function is as follows:

```
include "examples/stdlib.wl"

sortThisArray arg : Arr -> Arr
  x = sort arg          // Uses the sort function defined inside of stdlib.wl
  arg                   // Returns the sorted array
```

3.5 Scoping

Weblang is statically scoped in a way that will be familiar to most C/C++ users. Unlike these languages, however, Weblang uses whitespace to denote different levels of scope. All statements at the same level of uninterrupted indentation will be within the same scope, and accordingly will have access to anything declared within that scope. A new scope is created by increasing the indentation level by two spaces; it is terminated by decreasing the indentation level back two spaces. Weblang uses an open scoping mechanism, so scopes begin their lives with access to all the symbols declared in their outer scopes. Because Weblang does not differentiate between declaring and defining, symbols cannot be redeclared (i.e. they cannot hide symbols declared outside the scope). Symbols declared within a scope will terminate when the scope ends (when the level of indentation decreases by two spaces).

Scopes can only be created under certain circumstances; otherwise, indentation will be rejected by the parser. These circumstances are:

- Within a function (the function body scope will always begin two spaces in)
- Within the body of a foreach loop
- Within an if/else statement

3.6 Functions

Functions in WebLang take one argument and return at most one value. Operators, all of which are built-in, however, may take two inputs. The function header consists of a function name, a single argument, a colon, and the input and output types separated by an arrow. The function body consists of a variable number of declarations and function calls. An example function declaration foo that takes a String argument x and returns nothing would be written as follows:

```
foo x : String -> Nothing
  // statements here
```

WebLang functions can be called by stating the function name followed by its argument. An example function call for foo defined above would be written as follows:

```
foo "hello"
```

3.6.1 Endpoint Functions

Endpoint functions are the default function type in WebLang. Any function without the helper reserved word at the beginning of the declaration is an endpoint function. When a WebLang program is compiled, it generates a server binary and provides an endpoint for each endpoint function. For example, having an endpoint function foo written in program.wl, and having the Weblang server running a port defined as 8000 will expose the /foo/ endpoint locally at `127.0.0.1:8000/program/foo`. A post request needs to be made at this endpoint, where the body sent should be an object with the key as arg and value being what you would like to pass to the endpoint, because all Weblang functions need one argument.

```
{'arg':'val'} //body that should be sent to post request at endpoint
```

3.6.2 Helper Functions

Helper functions in WebLang are user-defined functions that do not result in a new endpoint upon compilation. These functions follow the same declaration syntax described above, but are initiated with the reserved word `helper`. For example, the following is the declaration for a helper function bar with String parameter x:

```
helper Bar x : String -> Nothing
  // statements here
```

3.6.3 Function calls

Functions are called by calling their identifier followed by an argument. As mentioned in the import section, functions may be called from other weblang files when using includes or other APIs when using import.

```
foo "hello"
getBitcoinPrice "" //if we had imported "gdax api" as in the import example.
```

3.6.4 Variable Assignment from Function

A variable may be assigned the return value of a function by separating the two with a single = sign. For example, the following sets the value of a variable exampleVar to the return value of the function foo with argument “hello”:

```
        exampleVar = foo "hello"
```

3.6.5 Arguments

Each endpoint or helper function takes one argument of any WebLang type. Each operator takes two arguments. All arguments passed to functions and operators are passed by value. Note that because generally we expect functions to require more than one argument, we expect this behavior to typically be passing in a JSON dictionary or array (as is almost always the case with RESTful requests).

Weblang performs a best effort conversion of the argument. When functions are called as endpoints (as they are often meant to be), they will be by definition need to be passed a string, because HTTP only transfers strings. This would theoretically mean that all functions that are to be exposed should only take strings. Because this would severely limit Weblang's ability to statically check semantics, Weblang will instead attempt to parse an argument to a function into a native weblang type prior to entering the function body. This means that a function being called as an endpoint that accepts as input an object can accept a string, but have it be used as an object within the function. This only works when the function is called as an endpoint; attempting to pass a string to a function that accepts an object from within weblang will throw a type error. This allows programmers to write a function that can be called both locally by passing an object and also called via endpoint passing a string without having to write any specific functionality to convert the type of the argument.

3.6.6 Recursion

Recursive function calls are formatted identically to traditional function calls. Both Helper and Endpoint functions may be called recursively.

3.6.7 log

Log is a built in function that takes one argument as string. It prints whatever argument it is given and print it to stdout. If the argument is not a string (i.e. a Number, JSON object, etc.), it will attempt to cast it to string; if it is unable to do so, it will an error at compile time. As such, it is highly recommended that user defined types have a way to cast to string.

3.6.8 Other builtins

- cat: Takes an array of two elements. It converts each element to string, concates the newly produced strings and return the concatenation.
- jn: Takes in a string and returns either an object or an array, or throws an exception if it is unable to detect what kind of container is present in the string. It does so by parsing the string using the JSON specification.
- addToObj: Takes in an array of three elements. The first element is the object to add to, the second element is the key, and the third is the value. The key must be a string, but the value can be any type (including another object). It returns a new object containing the key value pair (as well as all key value pairs in the original object). If the key is already contained by the object prior, it will replace/update the value of the key with the new value and return the new object containing the updated value. In either case, it does not modify the existing object.
- push: Takes in an array of two elements. The first element is an array, the second is the value to be added to the array. The value can be of any type (including another array or object). It returns an array containing every element of the array passed in, plus the new value at the end (it does not modify the existing array).
- update: Takes in an array of three elements. The first is an array, the second is of type num and is an index for an element in the array, and the third is a value. The value can be of any type (including another array or object). It returns an array containing every element of the array passed in, but replacing the value at the index with the value passed in (it does not modify the existing array). If the index is outside the bounds of the array, an error is thrown.

- equals: Takes in an array of two elements. The elements can be of any type. Equals will perform a comparison of the two elements, returning true if the objects are structurally the same (i.e. if they are strings, it is equivalent to Java's equals method, while if they are objects, every element within the object will be compared to determine equivalency), or false if they are not. Although this method works for Nums as well, == is preferred in that case because == is directly implemented in LLVM, so it is more efficient.
- get: takes an array of two elements. The first element is a container (array or object), while the second is either a num or a string. If the first element is an array, the second should be a num (the index); if the first is an object, the second should be a string (the value). If passed an array/num, it will return (by value) the ith element of the array, or throw an error if ith is out of bounds. If passed an object/string, it will return the value corresponding to the key within the object, or throw an error if the key is not found.
- toNum: Takes in a string, and returns a num. If it cannot convert the string into a number, it throws an error.
- Type checks:
  - isNum: takes in a value of any type, and returns a boolean based on whether or not the value is of num type.
  - isBool: takes in a value of any type, and returns a boolean based on whether or not the value is of bool type.
  - isString: takes in a value of any type, and returns a boolean based on whether or not the value is of string type.
  - isArr: takes in a value of any type, and returns a boolean based on whether or not the value is an array.
  - isObj: takes in a value of any type, and returns a boolean based on whether or not the value is an object.
- avg: Takes in an array of num values and returns the average of the nums.
- arrconcat: Takes in an array of two arrays and returns one combined array.
- contains: Takes in an array containing one array and either a string or num, and returns a boolean. Iterates through the selected array – the first element in the function argument – and checks if it contains the specified element. If true, the function returns 1. Otherwise, the function returns 0.
- sort: Takes in an array of num values and sorts the array in ascending order. The sorting algorithm used to implement this builtin is the bubblesort.
- fixedArr: Takes in a num value x and creates an array of length x. The contents of this array are the numbers 0 through x-1 in ascending order. This array may then be used in foreach loops to simulate the "for i in range x" syntax of Python.
- gcd: Takes in a an array of two num values and returns the greatest common denominator between them.

3.7 Control Statements

Weblang executes statements from top to bottom and left to right. But when using control statements, this breaks up the flow of the execution by integrating logical execution of code by using loops or branching with if/else statements.

Looping: foreach

The `foreach` statement allows the user to iterate over an array or a JSON object as in python. If iterating over a JSON object, the loop will loop over the outermost keys in the object.
Examples:

```
\\Statement writes to std out all the values in array
foreach val in [1,2,"hello"] {
  log val
}

\\Statement writes to std out all the keys in array
obj = {"foo": "bar"}

foreach key in arrKeys {
  log key //will just print key to stdout
    //In this case, will only print foo
}
```

Due to the fact that arrays and objects can contain multiple types, a common pattern in WebLang is to check the type utilizing the : built in function within each loop and execute based on that. For example:

```
foreach val in [1,2,"hello"]
  if(val : Number)
    log (val+1) //only triggers if array contains Number
  else
    log val
}
```

Because foreach operates on arrays or objects only, for loops as expected in Java or C must be approximated. This can easily be done by creating an Array of Numbers and iterating over that.

To simulate iteration over a value range such as the python *for i in range(x)* syntax where x is some number to iterate to from 0, a fixed length array can be created and iterated over as follows:

```
x = fixedArr 5  // Creates an array of length 5 with values [0, 1, 2, 3, 4]
foreach i in x
  log i         // Iterates through x and logs 0, 1, 2, 3, 4 sequentially
```

if else

The `if` statement tells the program to execute a certain block of code when a test evaluates to true; while the `else` clause follows the if should the if fail. Elses latch on to the nearest if at the same indentation level. The body of an if or else must be at the same indentation level (see Scoping for more details). All ifs must have a matching else, because of weblang's return semantics (weblang returns the value of the last executed line; if there is an unmatched if, the behavior could be problematic). That being said, if a user would like to have an if statement that should do nothing if it fails, the user can just put an object of the same return type within the else, as seen in the example below.

```
\\Statement writes to std out all the values in array
if(true)
  foreach val in [1,2,"hello"]
  log val
else
  0


\\Statement writes bar to stdout
if(false)
  log "foo"
else
  log "bar"
```

If statements will attempt to evaluate their condition as an expression, and then cast the condition to a boolean. This means that weblang numerical expressions are cast to booleans in the evaluation of the if statement. This uses C-like semantics: 0 represents false, while all other numeric values represent true.

```
\\Only the else will execute.
if(1-1)
  log "this won't execute"
else
  log "this will"
```

3.8 Expressions

An expression is composed of one of the following:

- An operand followed by an operator followed by an operand
- Initializing an object
- Accessing a:
  - Object
  - JSON Object
  - Array
- An expression between ()
- Any of the subsections below

3.8.1 Arithmetic

An arithmetic expression consists of an operand followed by one or more operators. Operands can be variables, constants, and expressions.

```
"15" //Expression evaluates to String
2 + 2 //Expression evaluates to Number 4
10.1 + 1.0 //Expression evaluates to Number 11.1
{"foo":"bar"} + {"bar":"foo"}
//Expression evaluates to {"foo":"bar", "bar": "foo"}
```

3.8.2 Function Call

A call to a function that returns a value is considered an expression.

```
post_dad_joke "What time did the man go to the dentist? Tooth hurt—y."
//Evaluates to Nothing
```

3.8.3 Object, Array

Values of an object can be accessed via modified bracket notation (dotted bracket notation): accessing the “color” key of Object car is *car.[color]*.

Array values can also be accessed utilizing bracket notation as in C, Python, or Java, but separated by a dot; accessing the 5th element of array a is *a.[5]*.

Values may be appended to a copy of an existent array by utilizing the push function and assigning it to a new variable; adding a number 1 to the end of an array a and assigning the altered version of a to a new array c is *c = push [a, 1]*. The argument to push is an array with 2 elements: the initial array and the value to append. Utilizing copy semantics, the array a remains unchanged.

Values may replace the element at a specific index of an existent array by utilizing the update function and assigning it to a new copy of the array; adding a number 1 to the 3rd index of an array a and assigning the altered version of a to a new array c is *c = update [a, 1, 3]*. The argument to update is an array with 3 elements: the initial array, the value to be inserted, and the index to insert the new value. The array a remains unchanged.

Examples of the above are shown here:

```
obj = { "array": [1,2,3] }
obj //Evaluates to Object {  a r r a y  :[1,2,3]}
Obj.array //Evaluates to JSON Array [1,2,3]
obj.array.[0] //Evaluates to 1

sample = [1, 2, 3]
pushedSample = push [sample, 9] // Evaluates to [1, 2, 3, 9]
updatedSample = update [sample, 9, 0] // Evaluates to [9, 2, 3]
```

3.8.4 Operators

An operator specifies a built in operation to be performed on operands. An operator can have one or two operands depending on what purpose it serves.

Assignment Operator

The assignment operator is used to store values into variables. As with most well used languages (Java, C, Python, etc.) Weblang uses "=" to store the value of the right side to the variable specified by the left side. The left side of the assignment operator may not be a literal or constant values. It will always evaluate the right side before assigning it (we use applicative evaluation).

```
obj = { "array": [1,2,3] }
```

Arithmetic Operators

The standard arithmetic operations addition, subtraction, multiplication, division, and modulo are included in WebLang.

1. Addition: Addition is performed on two values of type number. Examples are provided:

```
5 + 5 // Evaluates to 10
5.4 + 3.1 // Evaluates to 8.5
```

2. Subtraction: Subtraction is performed on two values of type number. Examples are provided:

```
5 − 5 // Evaluates to 0
4.2 − 1.3 // Evaluates to 2.9
```

3. Multiplication: Multiplication is performed on two values of type number. Examples are provided:

```
5 ∗ 5 // Evaluates to 25
4.2 ∗ 3.1 // Evaluates to 13.02
```

4. Division: Division is performed on two values of type number. Examples are provided:

```
5/5 // Evaluates to 1
6.4/2 // Evaluates to 3.1
```

5. Modulo: Modulo is performed on two values of type number, but the numbers must be whole integers. Examples are provided:

```
6 % 2 // Evaluates to 0
5 % 2 // Evaluates to 1
```

Conditional Operators

Conditional operators are used to determine how two operands relate to each other. As such, they will always take two values as inputs. The result of an operator is either `true` or `false`.

The conditional operators are:

Less than	<
Less than or equal to	<=
Equality	==
Greater than	>
Greater than or equal to	>=

Examples are as follows:

```
a = 1
b = 2
d = {  a r r  : [1,2,3]}
c = (a<b) //Evaluates to true
c = (<=x) //Evaluates to true
c = (w>x) //Evaluates to false
c = (w>=x) //Evaluates to true
c = d == Null //Evaluates to false
c = d != Null //Evaluates to true
```

Furthermore, conditional operators can be chained together using && and || operators. The && operator behaves like logical and, while the || operator behaves like logical or.

```
c = true && false //evaluates to false
c = true && true //evaluates to true
c = true || true //evaluates to true
c = false || false //evaluates to true
```

3.8.5 Operator Precedence

When multiple operators are used, the operations are grouped based on rules of precedence. Below is a list of precedence, if two or more operators have equal precedence, the operators are applied from left to right. Parentheses can be used to manually overwrite WebLang's precedence rules.

1. Method or helper calls, object or array access
2. Object set operations
3. Multiplication, division
4. Addition, subtraction
5. Expressions
6. Assignment expressions

Because of this ordering, chaining function calls requires

3.9 Compiler Output

The program outputted upon successful compilation (running ./weblang [filename].wl) is an executable with the original filename, minus the extension. By default, WebLang programs can be run as scripts by running filename [functionname] from the command line, where [functionname] is the name of some some function within your compiled executable. Because of the way include works, any function in the include tree can be run in this way. All functions called in this fashion from the command line must be passed an argument; if the function does not use it, any argument will do.

3.9.1 Running a Server

A primary component of weblang is its ability to be run as a server. Running the included executable (runWeblangServer) will start up a server that automatically exposes all the endpoints (non helper functions) it can find (it looks for all executables within the directory it is contained in). The url corresponding to each function is of the form 127.0.0.1:[port]/programname/functionname.

3.9.2 Options when running a server

The only option passed to run a server is the port number the server should bind to. This is done like so: ./runWeblangServer 8000. This option is required for the server to run.

3.9.3 Compilation example

The entire compilation pipeline for a file called example.wl is as follows:

```
/*
Contents of example.wl:
test arg : Obj -> Str
  log "hi"
*/

./weblang example.wl //produces executable "example"


/*
Running our example program as a script -
we must pass in the function we'd like to call
(the concept of main does not exist).
*/
./example test a //outputs hi


/*
Running our example program as a server:
when runWeblangServer is run, it collects
all executables in the current folder.
*/

./runWeblangServer 8000
/*
We can now send a POST request to 127.0.0.1:8000/example/test, with body {'arg
   ':'a'}
and receive "hi" in response.
*/
```

4. Project Plan

4.1 Planning Logistics

4.1.1 Weekly Sprints

In order to ensure that we were making steady progress towards completing Weblang, we arranged weekly meetings – typically on Mondays or Fridays – to discuss our goals and delegate work. Throughout the first portion of the semester, these meetings primarily served the purpose of addressing broad language goals and discussing milestone assignments such as the Project Proposal and Language Reference Manual. As the semester progressed, so did the technical specificity of our conversations during these meetings. We began assigning one another specific technical work and making final decisions regarding language design, compiler implementation, and the test suite.

In addition, we typically met with our mentor – Lizzie Paquette – during her Monday office hours. During these meetings Lizzie would view our current progress, answer any questions we had regarding the viability of certain goals or implementation strategies, and let us know what step we needed to take to meet the next deadline.

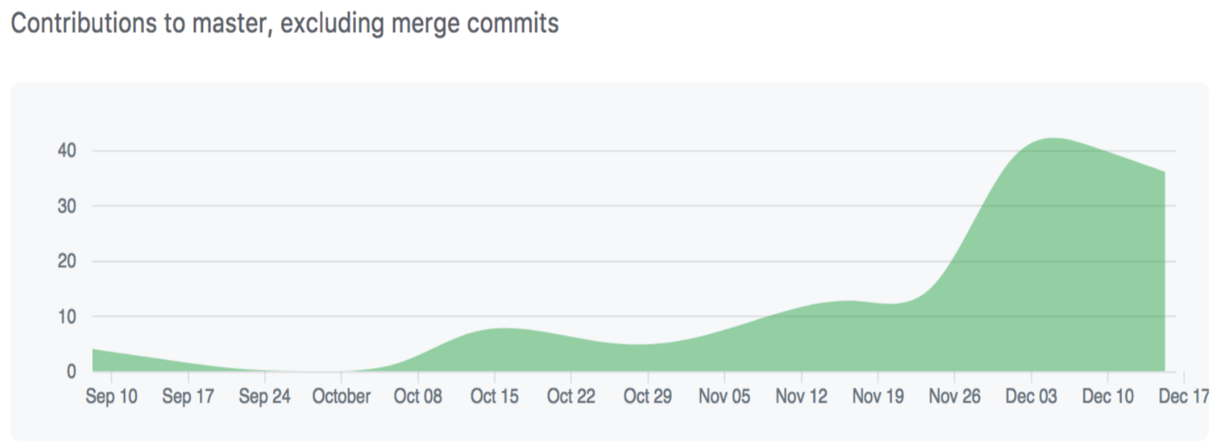
4.1.2 Team Communication

Proper communication throughout the semester proved to be paramount to successfully completing our tasks. In order to maintain constant communication among the entire team, we utilized a Slack group. Within this group we had different channels to encapsulate conversations relevant to specific portions of our project, such as *General, Language Reference Manual, Testing, Demos*, and several others. Having these different channels helped organize information to ensure that, for example, an important note regarding something such as the Language Reference Manual didn't get lost in a long series of logistical messages.

4.1.3 Development Workflow

All of our development was conducted within the confines of a virtual machine hosted on Google Cloud, as it simplified working with the Nix package manager. Here we each worked off of our shared repository under Git version control, contributing to separate branches that required pull request approval from at least one teammate. We utilized pull requests as opposed to unauthenticated merges to maximize the amount of review each piece of code received before its integration into the overall codebase. This allowed us to easily catch potential issues that may have otherwise taken a significant amount of time to locate and debug.

The following graphic plots the number of commits to master (excluding merge commits) over the course of the semester. As shown in the graph, there were significantly fewer commits in the former half of the semester while we were having more conceptual discussions. As the semester progressed, we more actively developed Weblang and, as a result, committed to the repository with increased regularity.



We maintained the quality of this repository by utilizing continuous integration via *Travis CI.* Travis CI would run our test suite every time someone on the team made a pull request, showing the output in the log of the pull request. When looking over whether or not to approve a pull request, the team member conducting the review could see the Travis output and make sure that the committed code didn't cause any of the tests to fail.

4.2 Style Guide

Maintaining stylistic consistency in our code was extremely important throughout the development of Weblang. This made comprehending and debugging code written by the rest of the team during development significantly easier, as it eliminated any learning curve regarding another team members code style. Some of these development style choices were as follows:

- Always use 2 spaces for indentation as opposed to using tabs. (Lexer will actually reject use of tabs).
- Keep lines to a maximum of 80 characters (not including spaces).
- Use logical function and variable naming conventions to improve readability.
- Comment code frequently with clear messages that make intentions abundantly clear.
- Include meaningful git commit messages

4.3 Project Timeline

Throughout the semester we addressed each of our Weblang deliverables and set deadlines for when we wanted to have them completed. Certain parts of the project, such as the proposal, LRM, and final deliverable adhered to specific dates to stay aligned with the course. We assigned the rest of our milestones looser, more general deadlines.

Milestone	Date
Proposal	September 27th
LRM	October 16th
Scanner	Mid October
Parser	Early November
Interpreter	Early-Mid November
Implement Server	Mid November
"Hello World!"	November 18th
Finalize Test Suite	Late November
Finalize JSON and Client Libs	Early December
Finalize Stdlib	Early December
Demo	December 19th
Final Deliverable	December 20th

4.4 Team Member Roles

While roles were defined at the beginning of the semester, team members often expanded outside the scope of their initial assignment to contribute in other areas.

- **Ryan Bernstein:** System Architecture
- **Christophe Rimann:** Project and Memory Management
- **Jordan Vega:** Language Design
- **Brendan Burke:** Testing and Documentation
- **Julian Serra:** Testing and Documentation

4.5 Software Development Tools

- **Languages:** Haskell, C, C++. Build script in bash.
- **Programming Editor:** Vim
- **Version Control:** Github
- **Testing:** Python
- **Continuous Integration:** Travis CI
- **Documentation:** Overleaf, Google Slides
- **Communication:** Slack

4.6 Project Log

```
20a2efb - Wed Dec 20 04:12:47 2017 +0000 bburke95@gmail.com: added log.txt for report
78b9fd4 - Tue Dec 19 10:14:04 2017 -0500 noreply@github.com: Merge pull request #56 from rybern/more_tests
a68f7e3 - Tue Dec 19 14:43:30 2017 +0000 jserra17@cmc.edu: more tessts
c7e21ff - Tue Dec 19 03:43:40 2017 -0500 noreply@github.com: Merge pull request #55 from rybern/bitcoin_demo
b60a593 - Tue Dec 19 08:37:07 2017 +0000 jserra17@cmc.edu: test fix
58248c9 - Tue Dec 19 08:25:08 2017 +0000 jserra17@cmc.edu: debug trav
7b14233 - Tue Dec 19 08:15:16 2017 +0000 jserra17@cmc.edu: fixing post test
4272794 - Tue Dec 19 08:01:37 2017 +0000 jserra17@cmc.edu: Merge branch 'master' of https://github.com/rybern/plt into bitcoin_demo
a4b2443 - Tue Dec 19 07:57:26 2017 +0000 jserra17@cmc.edu: Merge branch 'updatedbitcoin' of https://github.com/rybern/plt into bitcoin_demo
4a27246 - Tue Dec 19 07:56:54 2017 +0000 infobiac1@gmail.com: stdlib fixed
a04fef9 - Tue Dec 19 02:55:46 2017 -0500 noreply@github.com: Merge pull request #53 from rybern/type-extras
8a3c80b - Tue Dec 19 07:55:19 2017 +0000 jserra17@cmc.edu: Merge branch 'updatedbitcoin' of https://github.com/rybern/plt into bitcoin_demo
727108a - Tue Dec 19 07:51:12 2017 +0000 jserra17@cmc.edu: merging chirstophes pr
9e8b49d - Tue Dec 19 02:49:05 2017 -0500 ryanbernstein1@gmail.com: undid change in weblang
6a5e71b - Tue Dec 19 07:48:57 2017 +0000 infobiac1@gmail.com: remove reference to stdlib (currently there's a mismatched type somewhere in there)
f909a1b - Tue Dec 19 07:36:44 2017 +0000 jserra17@cmc.edu: added headers
01bafc3 - Tue Dec 19 07:32:11 2017 +0000 infobiac1@gmail.com: updated bitcoin to use include
b2c47c1 - Tue Dec 19 02:23:10 2017 -0500 ryanbernstein1@gmail.com: fixing more tests
8f1e1dc - Tue Dec 19 02:22:39 2017 -0500 ryanbernstein1@gmail.com: added error message for incorrect return type
190d9d0 - Tue Dec 19 02:08:26 2017 -0500 ryanbernstein1@gmail.com: maybe fixed tests
91f043a - Tue Dec 19 07:00:23 2017 +0000 jserra17@cmc.edu: adding some tests and emailGCD plus voice2price edit to include stdlib
b343bed - Tue Dec 19 01:57:51 2017 -0500 ryanbernstein1@gmail.com: merge master
922338a - Tue Dec 19 01:54:23 2017 -0500 ryanbernstein1@gmail.com: added type check boolean operator, and/or boolean operators
b56425f - Tue Dec 19 00:35:09 2017 -0500 ryanbernstein1@gmail.com: first attempt to add asserts/type asserts, need more testing
77c1a87 - Tue Dec 19 05:12:14 2017 +0000 jserra17@cmc.edu: Merge branch 'master' of https://github.com/rybern/plt into bitcoin_demo
77237b5 - Mon Dec 18 23:58:09 2017 -0500 noreply@github.com: Merge pull request #52 from rybern/add-includes
f20a08f - Mon Dec 18 23:46:57 2017 -0500 ryanbernstein1@gmail.com: added include functionality
4cc728b - Mon Dec 18 23:25:51 2017 -0500 noreply@github.com: Merge pull request #51 from rybern/header
50f20e2 - Tue Dec 19 04:18:29 2017 +0000 jmv2177@columbia.edu: header in
```

```
imports
e7222ac - Tue Dec 19 02:57:09 2017 +0000 jserra17@cmc.edu: Merge branch 'master' of https://github.com/rybern/plt into bitcoin_demo
879c4b8 - Mon Dec 18 21:50:40 2017 -0500 noreply@github.com: Merge pull request #50 from rybern/slack
610549b - Tue Dec 19 02:48:22 2017 +0000 jmv2177@columbia.edu: slack example updated
cfe6ed8 - Mon Dec 18 19:28:52 2017 -0500 noreply@github.com: Merge pull request #49 from rybern/voice2price
25207ec - Mon Dec 18 23:20:22 2017 +0000 infobiac1@gmail.com: th4t 5w33t d3m0
dd37670 - Mon Dec 18 17:57:19 2017 -0500 noreply@github.com: Merge pull request #48 from rybern/standard
3105e7a - Mon Dec 18 22:30:18 2017 +0000 bburke95@gmail.com: fixed problem with foreach nested in if on conains method
a64e288 - Mon Dec 18 22:04:26 2017 +0000 jserra17@cmc.edu: working average
110fc43 - Mon Dec 18 21:35:24 2017 +0000 jserra17@cmc.edu: some demo examples
d3c4047 - Mon Dec 18 21:09:24 2017 +0000 bburke95@gmail.com: Merge branch 'master' of https://github.com/rybern/plt into standard
535c061 - Mon Dec 18 16:07:49 2017 -0500 noreply@github.com: Merge pull request #47 from rybern/concat
ae02bf9 - Mon Dec 18 21:06:44 2017 +0000 bburke95@gmail.com: fixed contains to  handle strings
d174a3e - Mon Dec 18 20:48:14 2017 +0000 infobiac1@gmail.com: Merge branch 'master' of https://github.com/rybern/plt into concat
1814e26 - Mon Dec 18 20:46:12 2017 +0000 infobiac1@gmail.com: string equality (and object equality via str equality) now works
83174fd - Mon Dec 18 15:15:18 2017 -0500 noreply@github.com: Merge pull request #46 from rybern/concat
15a517e - Mon Dec 18 20:08:18 2017 +0000 infobiac1@gmail.com: str concat
05807c6 - Mon Dec 18 14:52:32 2017 -0500 noreply@github.com: Merge pull request #45 from rybern/fixdumbquotes
0dccaf0 - Mon Dec 18 19:45:34 2017 +0000 infobiac1@gmail.com: Merge branch 'master' of https://github.com/rybern/plt into fixdumbquotes
5c074a4 - Mon Dec 18 14:32:58 2017 -0500 noreply@github.com: Merge pull request #44 from rybern/demo_and_tests
f683087 - Mon Dec 18 19:31:49 2017 +0000 infobiac1@gmail.com: rando
0046c66 - Mon Dec 18 14:01:19 2017 -0500 noreply@github.com: Merge pull request #43 from rybern/fixdumbquotes
594dcde - Mon Dec 18 18:04:44 2017 +0000 infobiac1@gmail.com: auth works
f164eaa - Mon Dec 18 17:39:33 2017 +0000 infobiac1@gmail.com: merging
9428a45 - Mon Dec 18 17:36:37 2017 +0000 jserra17@cmc.edu: sample and travis check
cb27f8d - Mon Dec 18 17:30:45 2017 +0000 infobiac1@gmail.com: v1 text
1ac30da - Mon Dec 18 12:17:28 2017 -0500 noreply@github.com: Merge pull request #42 from rybern/auth
e3862f3 - Mon Dec 18 17:15:59 2017 +0000 jmv2177@columbia.edu: adding key and secret to import in test
0d0ddd6 - Mon Dec 18 17:07:13 2017 +0000 jmv2177@columbia.edu: key and secret coming from imports
08cea9a - Mon Dec 18 11:25:30 2017 -0500 noreply@github.com: Merge pull request #41 from rybern/argsparsed
1a8fd3b - Mon Dec 18 16:17:41 2017 +0000 infobiac1@gmail.com: removing print
93a309d - Mon Dec 18 10:51:13 2017 -0500 noreply@github.com: Merge pull request #40 from rybern/standard
```

```
f1c0584 - Mon Dec 18 15:49:03 2017 +0000 infobiac1@gmail.com: jn checks if obj
     first (backwards compatible)
ee0e6c8 - Mon Dec 18 15:44:44 2017 +0000 infobiac1@gmail.com: Arguments
    autoparsed (don't need to call jn on args anymore), some error checks
edef40f - Mon Dec 18 15:41:25 2017 +0000 bburke95@gmail.com: got rid of dummy
    test function
ffe4d48 - Mon Dec 18 10:37:50 2017 -0500 noreply@github.com: Merge pull
    request #39 from rybern/server
d85dca0 - Mon Dec 18 15:34:41 2017 +0000 jmv2177@columbia.edu: removing logs
01124b0 - Mon Dec 18 15:33:13 2017 +0000 jmv2177@columbia.edu: Merge branch '
    master' into server
17cddf9 - Mon Dec 18 15:31:22 2017 +0000 jmv2177@columbia.edu: handling empty
    return
6a44c9a - Sun Dec 17 23:37:16 2017 -0500 noreply@github.com: Merge pull
    request #38 from rybern/standard
a38b6cc - Mon Dec 18 04:07:11 2017 +0000 bburke95@gmail.com: added sort
    function to sort an array
c9278cc - Mon Dec 18 03:07:54 2017 +0000 bburke95@gmail.com: Merge branch '
    master' of https://github.com/rybern/plt into standard
f12043b - Sun Dec 17 22:00:33 2017 -0500 noreply@github.com: Merge pull
    request #37 from rybern/flexibleclient
3bc3374 - Mon Dec 18 02:53:39 2017 +0000 infobiac1@gmail.com: making post/get
    more flexible
b9f37d2 - Sun Dec 17 20:56:28 2017 -0500 noreply@github.com: Merge pull
    request #36 from rybern/jsonlibheader
8aabc88 - Mon Dec 18 00:57:53 2017 +0000 infobiac1@gmail.com: Merge branch '
    master' of https://github.com/rybern/plt into jsonlibheader
8054893 - Mon Dec 18 00:57:34 2017 +0000 infobiac1@gmail.com: header file
f38b975 - Sun Dec 17 19:23:26 2017 -0500 noreply@github.com: Merge pull
    request #35 from rybern/add-import
d266ba9 - Sun Dec 17 23:54:01 2017 +0000 jmv2177@columbia.edu: fixing tests
6879dd0 - Sun Dec 17 23:39:45 2017 +0000 jmv2177@columbia.edu: fixing get test
fd03e9e - Sun Dec 17 23:07:55 2017 +0000 jmv2177@columbia.edu: fixing post and
     get
e7ff30d - Sun Dec 17 23:05:03 2017 +0000 jmv2177@columbia.edu: readding
    deleted
b3dea93 - Sun Dec 17 23:04:47 2017 +0000 jmv2177@columbia.edu: correct example
bc7cf4b - Sun Dec 17 23:04:34 2017 +0000 jmv2177@columbia.edu: post handles
    two args
a9b51e9 - Sun Dec 17 22:42:19 2017 +0000 bburke95@gmail.com: Merge branch '
    master' of https://github.com/rybern/plt into standard
ce0184d - Sun Dec 17 22:39:08 2017 +0000 jmv2177@columbia.edu: removing unused
fbea11c - Sun Dec 17 22:29:48 2017 +0000 bburke95@gmail.com: function for
    checking if num or string is in an array in stdlib
76c79a4 - Sun Dec 17 22:28:40 2017 +0000 jmv2177@columbia.edu: imports with
    slack example
69cb6b8 - Sun Dec 17 15:55:09 2017 -0500 ryanbernstein1@gmail.com: typo
c6b0db7 - Sun Dec 17 15:51:01 2017 -0500 ryanbernstein1@gmail.com: added
    endpoints to available functions at checking-time
dcefe6d - Sun Dec 17 20:30:39 2017 +0000 jmv2177@columbia.edu: Merge branch '
    master' into add-import
6b9f410 - Sun Dec 17 20:28:49 2017 +0000 jmv2177@columbia.edu: imports working
b7e06f0 - Sun Dec 17 14:12:22 2017 -0500 noreply@github.com: Merge pull
    request #33 from rybern/test_mass
```

```
f82263b - Sun Dec 17 18:31:35 2017 +0000 jserra17@cmc.edu: Merge branch '
    master' of https://github.com/rybern/plt into test_mass
e956660 - Sun Dec 17 13:28:15 2017 -0500 noreply@github.com: Merge pull
    request #34 from rybern/localcpr
a7bd9d0 - Sun Dec 17 18:17:48 2017 +0000 infobiac1@gmail.com: adding the cpr
    lib to our repo (so we don't have to use jordans - hopefully itll work for
     ryan now)
6e9fa44 - Sun Dec 17 18:13:41 2017 +0000 infobiac1@gmail.com: remove origin
    cpr-example
f187463 - Sun Dec 17 09:10:49 2017 +0000 jserra17@cmc.edu: further debugging
    trav
2a25c99 - Sun Dec 17 08:59:22 2017 +0000 jserra17@cmc.edu: print output for
    travis debug
91eb42b - Sun Dec 17 03:56:16 2017 -0500 noreply@github.com: Merge pull
    request #32 from rybern/standard
eb0d45f - Sun Dec 17 08:51:22 2017 +0000 jserra17@cmc.edu: forgot yml file :(
08e47db - Sun Dec 17 08:50:35 2017 +0000 jserra17@cmc.edu: travis woes fixed
    at last
07df5dd - Sun Dec 17 08:40:46 2017 +0000 jserra17@cmc.edu: travis syntax fix
ca3125d - Sun Dec 17 08:33:29 2017 +0000 jserra17@cmc.edu: checking travis
    success
117aa74 - Sun Dec 17 08:28:26 2017 +0000 jserra17@cmc.edu: adding more tests
3d8d054 - Sun Dec 17 08:05:26 2017 +0000 bburke95@gmail.com: added gcd to
    stdlib.wl
6e74450 - Sun Dec 17 07:34:25 2017 +0000 bburke95@gmail.com: adding
    createFixedArr function to stdlib
46a26fa - Sat Dec 16 22:54:11 2017 -0500 noreply@github.com: Merge pull
    request #31 from rybern/nestedparsing
b256e86 - Sun Dec 17 03:48:54 2017 +0000 infobiac1@gmail.com: parse nested
    objs correctly
b37a1d4 - Sun Dec 17 02:11:18 2017 +0000 jmv2177@columbia.edu: merging
db0673c - Sat Dec 16 20:27:38 2017 -0500 noreply@github.com: Merge pull
    request #30 from rybern/functiontypes
a7c97d7 - Sun Dec 17 01:21:05 2017 +0000 infobiac1@gmail.com: Should be all
    the exposed function types
2548384 - Sat Dec 16 19:17:30 2017 -0500 noreply@github.com: Merge pull
    request #27 from rybern/server
25cab08 - Sat Dec 16 19:00:28 2017 -0500 noreply@github.com: Merge pull
    request #29 from rybern/bashupdate
97ae714 - Sat Dec 16 23:56:21 2017 +0000 infobiac1@gmail.com: weblang script
    2.0 (c)
c405201 - Sat Dec 16 18:18:16 2017 -0500 noreply@github.com: Merge pull
    request #28 from rybern/clientreturn
b68c258 - Sat Dec 16 23:14:36 2017 +0000 infobiac1@gmail.com: Remove print
606c127 - Sat Dec 16 23:13:24 2017 +0000 infobiac1@gmail.com: Super hacky fix
    to client return
34c52f2 - Sat Dec 16 22:36:55 2017 +0000 jmv2177@columbia.edu: deleting
fa1a074 - Sat Dec 16 22:27:22 2017 +0000 jmv2177@columbia.edu: Merge branch '
    master' of https://github.com/rybern/plt
531d428 - Sat Dec 16 22:25:36 2017 +0000 jmv2177@columbia.edu: fixing server
    args
f7a2907 - Sat Dec 16 17:16:52 2017 -0500 noreply@github.com: Merge pull
    request #26 from rybern/arrayupdates
5b52096 - Sat Dec 16 21:53:46 2017 +0000 infobiac1@gmail.com: update, push
```

```
    work for arrays
2372c53 - Sat Dec 16 21:07:49 2017 +0000 infobiac1@gmail.com: isBool
deb1909 - Sat Dec 16 20:34:40 2017 +0000 infobiac1@gmail.com: this was
    annoying me
89c678a - Sat Dec 16 14:25:01 2017 -0500 noreply@github.com: Merge pull
    request #25 from rybern/post
5b05b62 - Sat Dec 16 19:06:12 2017 +0000 jmv2177@columbia.edu: working with
    Semantic
e1c1350 - Sat Dec 16 18:50:45 2017 +0000 jmv2177@columbia.edu: Merge branch '
    master' into post
42c3fb6 - Sat Dec 16 18:49:35 2017 +0000 jmv2177@columbia.edu: post with json
    objs and add
70d0910 - Sat Dec 16 13:34:58 2017 -0500 noreply@github.com: Merge pull
    request #24 from rybern/static-analysis
3392d8c - Sat Dec 16 13:34:14 2017 -0500 ryanbernstein1@gmail.com: remove
    unnecessary extern
14d484e - Sat Dec 16 13:03:53 2017 -0500 ryanbernstein1@gmail.com: added
    typing example
d12ecb2 - Sat Dec 16 12:59:17 2017 -0500 ryanbernstein1@gmail.com: updated
    tests
da0f221 - Sat Dec 16 12:52:14 2017 -0500 ryanbernstein1@gmail.com: merged
    master; double implemented modulo
f19e907 - Sat Dec 16 12:46:15 2017 -0500 noreply@github.com: Merge pull
    request #23 from rybern/mod
ff49be1 - Sat Dec 16 17:29:32 2017 +0000 infobiac1@gmail.com: removing useless
     print
6e94428 - Sat Dec 16 17:29:07 2017 +0000 infobiac1@gmail.com: mods working (
    for gcd)
fbdf77f - Fri Dec 15 23:52:18 2017 -0500 ryanbernstein1@gmail.com: added types
    +conditions example
e1691c3 - Fri Dec 15 23:50:19 2017 -0500 ryanbernstein1@gmail.com: added pre-
    and post-condition checking. also added % operator
25cd2b8 - Fri Dec 15 21:50:54 2017 -0500 ryanbernstein1@gmail.com:
    transitioned to a weak typechecking scheme where errors are only thrown
    when the typechecker is certain
f3f9288 - Fri Dec 15 21:15:14 2017 -0500 ryanbernstein1@gmail.com: First stab
    at static type checking. Issues with container types and builtins
5ee1a92 - Fri Dec 15 18:10:00 2017 -0500 noreply@github.com: Merge pull
    request #22 from rybern/tests
b5fecd0 - Fri Dec 15 23:03:05 2017 +0000 jserra17@cmc.edu: cleanup
119d356 - Fri Dec 15 23:00:05 2017 +0000 jserra17@cmc.edu: first travis with
    few tests (but passing)
5a9f315 - Fri Dec 15 17:59:34 2017 -0500 ryanbernstein1@gmail.com: typo fix
d4a003f - Fri Dec 15 17:57:50 2017 -0500 ryanbernstein1@gmail.com: little
    cleanup endpoint code
ed8e374 - Fri Dec 15 17:53:30 2017 -0500 ryanbernstein1@gmail.com: first
    attempt at import statements. currently not building the whole argument to
     get/post, but the structure might be right
e0fe457 - Fri Dec 15 22:47:20 2017 +0000 jserra17@cmc.edu: Merge branch 'tests
    ' of https://github.com/rybern/plt into tests
590e960 - Fri Dec 15 22:42:40 2017 +0000 jserra17@cmc.edu: changes to tests
    and readme
34a16b0 - Fri Dec 15 19:48:18 2017 +0000 jserra17@cmc.edu: added travis
2206232 - Fri Dec 15 14:47:39 2017 -0500 noreply@github.com: Merge pull
```

```
    request #21 from rybern/make_server
3e9cb35 - Fri Dec 15 19:46:05 2017 +0000 jmv2177@columbia.edu: fix make server
     and copy
ff73fd9 - Fri Dec 15 14:40:35 2017 -0500 noreply@github.com: Merge pull
    request #20 from rybern/server_fargs
a5ace84 - Fri Dec 15 19:38:37 2017 +0000 jmv2177@columbia.edu: server to
    handle args
4476bf4 - Fri Dec 15 17:55:27 2017 +0000 jserra17@cmc.edu: Merge branch '
    master' of https://github.com/rybern/plt into tests
20faeb1 - Fri Dec 15 17:55:10 2017 +0000 jserra17@cmc.edu: commit before new
    merge
8f5c4b4 - Fri Dec 15 12:49:40 2017 -0500 noreply@github.com: Merge pull
    request #19 from rybern/objliterals
f18f4a2 - Fri Dec 15 12:48:47 2017 -0500 noreply@github.com: Merge pull
    request #18 from rybern/checktypes
a9ebb00 - Fri Dec 15 04:41:18 2017 +0000 infobiac1@gmail.com: jn works to
    parse arrays from string now too
214d821 - Fri Dec 15 04:26:49 2017 +0000 infobiac1@gmail.com: access objects
    like you would an array
1b1f64b - Fri Dec 15 03:25:01 2017 +0000 infobiac1@gmail.com: object literals
    working
18702b7 - Fri Dec 15 02:14:37 2017 +0000 infobiac1@gmail.com: Can add to
    object now
7809f68 - Fri Dec 15 01:01:53 2017 +0000 infobiac1@gmail.com: added toNum
    function
e280b8d - Thu Dec 14 23:55:11 2017 +0000 infobiac1@gmail.com: Doubles are now
    accessible in the same fashion as strings from json objects
7287192 - Thu Dec 14 23:01:33 2017 +0000 jserra17@cmc.edu: initial tests,
    script works
95327cc - Thu Dec 14 16:28:18 2017 -0500 noreply@github.com: Merge pull
    request #17 from rybern/checktypes
fcb01b3 - Thu Dec 14 20:55:53 2017 +0000 infobiac1@gmail.com: booleans work in
     if statements
1033cf9 - Thu Dec 14 20:48:02 2017 +0000 infobiac1@gmail.com: leq-geq-eq
    working
28e4e7b - Thu Dec 14 19:56:05 2017 +0000 infobiac1@gmail.com: accidentally
    overwrote this earlier
6ed5b3a - Thu Dec 14 19:55:44 2017 +0000 infobiac1@gmail.com: isObj, isArr
    working
87b3c8c - Thu Dec 14 18:57:56 2017 +0000 infobiac1@gmail.com: isNum, isString
2543f84 - Mon Dec 11 13:49:04 2017 -0500 noreply@github.com: Merge pull
    request #16 from rybern/extra-parsing
4f083ad - Sat Dec 9 02:03:27 2017 -0500 ryanbernstein1@gmail.com: added
    constant stub
f202ede - Sat Dec 9 01:37:30 2017 -0500 ryanbernstein1@gmail.com: no longer
    exposing helper functions
f142f7e - Sat Dec 9 01:31:38 2017 -0500 ryanbernstein1@gmail.com: common
    operators are lexed and parsed, following java's rules for precedence.
    Also fixed issue with newlines and trailing spaces
7b18f67 - Fri Dec 8 22:38:27 2017 -0500 ryanbernstein1@gmail.com: allowing
    general terms like if/then/else as indices
2954c25 - Fri Dec 8 22:32:59 2017 -0500 ryanbernstein1@gmail.com: added
    indexing lexing+parsing+ast, llvm works for arrays, need to be able to
    check type to get further
```

```
7814136 - Fri Dec 8 14:22:21 2017 -0500 ryanbernstein1@gmail.com: parsing top-level import
4c23d41 - Fri Dec 8 14:03:47 2017 -0500 ryanbernstein1@gmail.com: added stubs for true/false/null literals
912dd05 - Fri Dec 8 13:59:55 2017 -0500 ryanbernstein1@gmail.com: parsing, lexing true+false
430f21e - Wed Dec 6 22:25:18 2017 -0500 noreply@github.com: Merge pull request #15 from rybern/housekeeping
9cc15b2 - Wed Dec 6 22:22:01 2017 -0500 ryanbernstein1@gmail.com: Fixed conditional comparison and return type
831b4a3 - Wed Dec 6 21:18:11 2017 -0500 ryanbernstein1@gmail.com: added scoping example
c0cc194 - Wed Dec 6 21:17:10 2017 -0500 ryanbernstein1@gmail.com: added scoping to code blocks, so that varibles defined inside blocks can't be accessed outside
b2f815e - Wed Dec 6 21:08:03 2017 -0500 ryanbernstein1@gmail.com: adding reassignment example
35418cd - Wed Dec 6 21:07:41 2017 -0500 ryanbernstein1@gmail.com: variable assignment now modifies the existing value
f2e2bc6 - Wed Dec 6 20:56:09 2017 -0500 ryanbernstein1@gmail.com: consolidated the way strings are allocated in various places
555fe52 - Wed Dec 6 20:51:48 2017 -0500 ryanbernstein1@gmail.com: foreach loops now return the array, functions now return i32* pointers so they can be used as values, no longer parsing function names to/from json
b0a14e0 - Wed Dec 6 18:56:58 2017 -0500 ryanbernstein1@gmail.com: got rid of ending newline
6047e2e - Wed Dec 6 18:52:09 2017 -0500 ryanbernstein1@gmail.com: typo
43e2cb7 - Wed Dec 6 18:39:09 2017 -0500 ryanbernstein1@gmail.com: changed return of foreach loop to i32 0. this is still not good, since loops can be rhs of assignments, but at least it can be used at the end of functions now
18836db - Wed Dec 6 18:35:30 2017 -0500 ryanbernstein1@gmail.com: use variables for wordy llvm types
a324911 - Wed Dec 6 18:30:51 2017 -0500 ryanbernstein1@gmail.com: Fixed some code style/formatting issues, fixed the double execution of assignments
5710c19 - Wed Dec 6 18:26:31 2017 -0500 ryanbernstein1@gmail.com: removed chapter3 example code
12241db - Wed Dec 6 18:24:58 2017 -0500 ryanbernstein1@gmail.com: Added simple build script for .wl -> binary
eacf2d0 - Tue Dec 5 13:38:44 2017 -0500 noreply@github.com: Merge pull request #14 from rybern/jsonstringfix
beebf5b - Tue Dec 5 18:28:55 2017 +0000 infobiac1@gmail.com: fixing json for new model
660c28b - Tue Dec 5 10:51:22 2017 -0500 noreply@github.com: Merge pull request #13 from rybern/workingfor
dcee594 - Tue Dec 5 10:43:17 2017 -0500 noreply@github.com: Merge pull request #12 from rybern/args_fix
87dbdc4 - Tue Dec 5 15:40:33 2017 +0000 jmv2177@columbia.edu: use argv + 2
03b04fc - Tue Dec 5 15:38:09 2017 +0000 infobiac1@gmail.com: fixing forloops, all vars now first store pointer, getvar loads that ptr
44769f8 - Tue Dec 5 00:29:11 2017 -0500 noreply@github.com: Merge pull request #11 from rybern/arrayaccess
0369d5d - Tue Dec 5 05:25:12 2017 +0000 infobiac1@gmail.com: array access, functions with 2 args
```

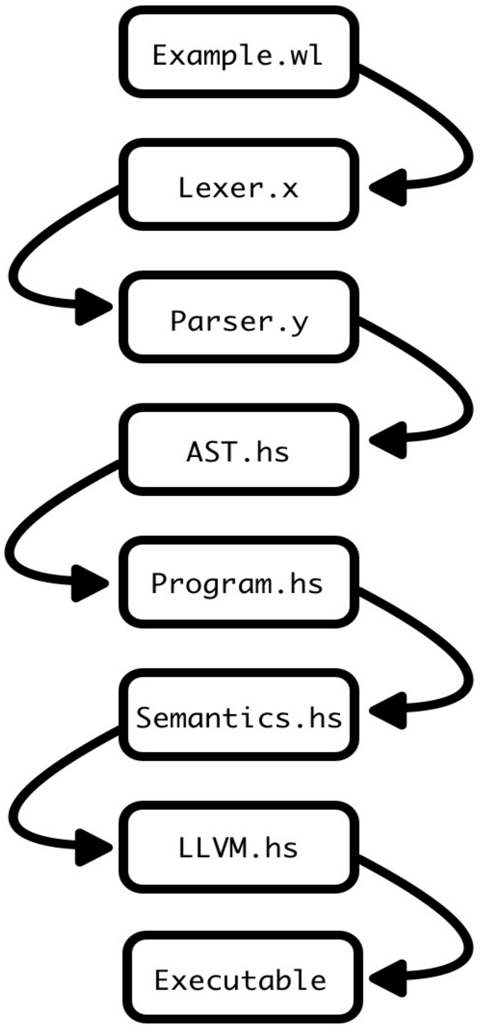
```
69e57bf - Mon Dec 4 22:59:36 2017 -0500 noreply@github.com: Merge pull request #10 from rybern/function_args
b56de20 - Tue Dec 5 03:55:09 2017 +0000 jmv2177@columbia.edu: merging with master and using cmd argv +2 for func
d5b7112 - Mon Dec 4 22:51:31 2017 -0500 noreply@github.com: Merge pull request #9 from rybern/forloops
923c7d6 - Tue Dec 5 01:38:49 2017 +0000 infobiac1@gmail.com: merging hell
52d63a7 - Mon Dec 4 23:03:54 2017 +0000 jmv2177@columbia.edu: using cmd args for function arg
ecc6ab5 - Mon Dec 4 21:48:20 2017 +0000 infobiac1@gmail.com: FOR LOOPS SONNNN
2819b07 - Mon Dec 4 15:34:41 2017 -0500 ryanbernstein1@gmail.com: forgot case without else
84e3bf8 - Mon Dec 4 15:23:10 2017 -0500 ryanbernstein1@gmail.com: Merge AST changes
79af8f0 - Mon Dec 4 20:09:20 2017 +0000 jserra17@cmc.edu: fix on lost changes
7f8f849 - Mon Dec 4 19:59:14 2017 +0000 jserra17@cmc.edu: updated makefile with slack
0773c6e - Mon Dec 4 14:44:43 2017 -0500 noreply@github.com: Merge pull request #8 from rybern/varassign
4f335ad - Mon Dec 4 14:43:38 2017 -0500 noreply@github.com: Merge branch 'master' into varassign
435af4b - Mon Dec 4 19:41:55 2017 +0000 jserra17@cmc.edu: working variable assignment
2ccf743 - Mon Dec 4 14:12:43 2017 -0500 noreply@github.com: Merge pull request #7 from rybern/functions
7fce60c - Mon Dec 4 19:10:43 2017 +0000 jmv2177@columbia.edu: command line args and slack demo
2ac0f8b - Mon Dec 4 18:33:40 2017 +0000 infobiac1@gmail.com: porting conditionals to use new doubles, some forloop work
705adaf - Mon Dec 4 12:47:11 2017 -0500 ryanbernstein1@gmail.com: First pass over AST->IR, consolidates If/Else blocks and ForEach blocks
2fcad4d - Mon Dec 4 07:03:42 2017 +0000 infobiac1@gmail.com: adding pattern matching to makefile
cbcef7d - Mon Dec 4 06:14:59 2017 +0000 infobiac1@gmail.com: Arrays now work!
9e503ba - Mon Dec 4 05:46:22 2017 +0000 infobiac1@gmail.com: fixing some binop stuff
e8cf9dd - Mon Dec 4 03:28:51 2017 +0000 jserra17@cmc.edu: var assign and slack
d82299f - Mon Dec 4 02:45:10 2017 +0000 infobiac1@gmail.com: types are all json under the hood
87e4087 - Sun Dec 3 22:17:12 2017 +0000 jmv2177@columbia.edu: fixing server maker
9f12d7f - Sun Dec 3 21:53:26 2017 +0000 jmv2177@columbia.edu: server to work with main args
23d4079 - Sun Dec 3 21:01:17 2017 +0000 jmv2177@columbia.edu: endpoint control flow
4f1bad7 - Sun Dec 3 19:18:15 2017 +0000 jmv2177@columbia.edu: for christophe
c526b98 - Sun Dec 3 04:49:22 2017 +0000 jmv2177@columbia.edu: Merge branch 'master' into functions
3640631 - Sun Dec 3 02:39:48 2017 +0000 infobiac1@gmail.com: rewriting extern function calls so we can add them more easily
cde8d9a - Sun Dec 3 01:18:03 2017 +0000 jmv2177@columbia.edu: using command line args
82ea7b2 - Fri Dec 1 18:54:55 2017 -0500 noreply@github.com: Merge pull request #5 from rybern/functions
```

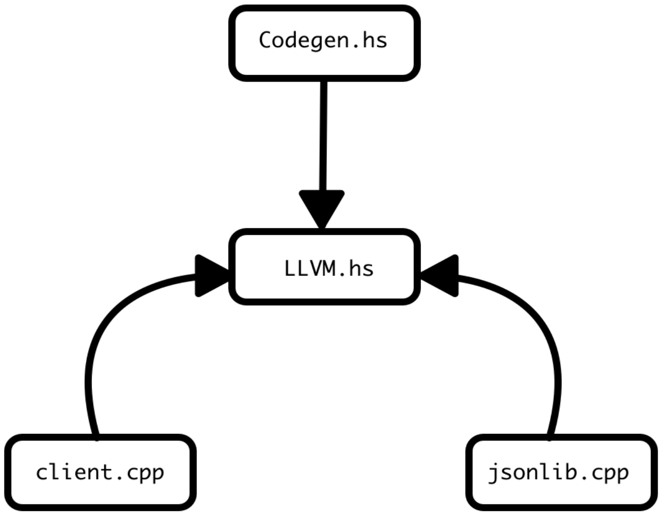
```
0fd79b3 - Fri Dec 1 18:54:36 2017 -0500 noreply@github.com: Merge branch 'master' into functions
a963b2b - Fri Dec 1 23:53:03 2017 +0000 jmv2177@columbia.edu: more complex function calls
aceb621 - Fri Dec 1 23:39:49 2017 +0000 jmv2177@columbia.edu: adding simple function call WL
c1a518e - Fri Dec 1 23:38:32 2017 +0000 jmv2177@columbia.edu: working function calls
c4d3fc3 - Wed Nov 29 12:09:19 2017 -0800 noreply@github.com: Merge pull request #4 from rybern/jsonification
10c333a - Wed Nov 29 12:08:00 2017 -0800 noreply@github.com: Merge pull request #3 from rybern/binop
7c70aa4 - Wed Nov 29 19:32:50 2017 +0000 jmv2177@columbia.edu: removing unneeded files
4d3406a - Wed Nov 29 14:07:06 2017 -0500 jjs2269@columbia.edu: decency fix
f54eec4 - Wed Nov 29 17:24:43 2017 +0000 infobiac1@gmail.com: binops
ec7381c - Wed Nov 29 05:59:44 2017 +0000 infobiac1@gmail.com: merging testerama in
1474b21 - Wed Nov 29 05:55:05 2017 +0000 infobiac1@gmail.com: first go at conditionals
aa04777 - Wed Nov 29 00:57:32 2017 +0000 jserra17@cmc.edu: working get
fec6ff0 - Wed Nov 29 00:54:19 2017 +0000 jmv2177@columbia.edu: missing echo-server files
5749eb3 - Tue Nov 28 17:11:46 2017 -0500 jjs2269@columbia.edu: added inital test script and expected folder and test
1434ddf - Mon Nov 27 00:46:12 2017 +0000 jmv2177@columbia.edu: adding echo server
d45b748 - Sun Nov 26 23:34:56 2017 +0000 jmv2177@columbia.edu: merging
48dd399 - Thu Nov 23 03:45:17 2017 +0000 infobiac1@gmail.com: Merge branch 'jsonification' of https://github.com/rybern/plt into jsonification
3092378 - Thu Nov 23 03:45:03 2017 +0000 infobiac1@gmail.com: adding functions to create/access all types
22fde76 - Mon Nov 20 23:22:24 2017 +0000 jmv2177@columbia.edu: merging with upstream
4d658df - Mon Nov 20 17:33:55 2017 -0500 ryanbernstein1@gmail.com: first attempt at pointer arrays to pass to json creation, not tested
572bbeb - Mon Nov 20 16:47:01 2017 -0500 ryanbernstein1@gmail.com: added simple example
25943cf - Mon Nov 20 16:45:16 2017 -0500 ryanbernstein1@gmail.com: simplify and refactor LLVM.hs
b3135b8 - Mon Nov 20 15:57:17 2017 -0500 ryanbernstein1@gmail.com: added to clean
3bff162 - Mon Nov 20 15:55:41 2017 -0500 ryanbernstein1@gmail.com: Quality of life improvements
57da510 - Mon Nov 20 19:53:33 2017 +0000 infobiac1@gmail.com: json works in nested calls AS LONG AS its in brackets
945db80 - Mon Nov 20 14:28:55 2017 -0500 ryanbernstein1@gmail.com: make array element evaluation more general
21c65d0 - Mon Nov 20 19:11:08 2017 +0000 infobiac1@gmail.com: commiting just in case
abd33e7 - Mon Nov 20 16:08:05 2017 +0000 infobiac1@gmail.com: Version that shows mem loss in valgrind (yay?)
ab34947 - Mon Nov 20 08:01:33 2017 +0000 infobiac1@gmail.com: Seems to be storing
```

```
dfcb538 - Mon Nov 20 04:31:39 2017 +0000 infobiac1@gmail.com: basic linking
    with chapter3 works (because i only have string types not really tho
ec89371 - Mon Nov 20 04:03:06 2017 +0000 infobiac1@gmail.com: first try at
    linking
95781fb - Sat Nov 18 23:18:26 2017 +0000 infobiac1@gmail.com: rudimentary json
     wrapper with strings
9fda8b6 - Sat Nov 18 22:33:48 2017 +0000 infobiac1@gmail.com: patching
    makefile to work on new installation
efb3d02 - Sat Nov 18 16:32:44 2017 +0000 infobiac1@gmail.com: cloned rapidjson
     into repo
ac7363c - Sat Nov 18 16:27:09 2017 +0000 infobiac1@gmail.com: fixing merge
    conflicts
8330eef - Tue Nov 14 06:14:01 2017 +0000 jmv2177@columbia.edu: dynamic
    allocation of array of strings
7343fa8 - Mon Nov 13 21:18:55 2017 +0000 infobiac1@gmail.com: Merge branch ’
    master’ of https://github.com/rybern/plt
6eba5ad - Mon Nov 13 16:03:00 2017 -0500 ryanbernstein1@gmail.com: refactor
    for more general value allocation
cf499cd - Mon Nov 13 15:55:27 2017 -0500 noreply@github.com: Merge pull
    request #2 from rybern/hello-world
b700632 - Mon Nov 13 20:50:20 2017 +0000 jmv2177@columbia.edu: oops
d923d12 - Mon Nov 13 20:46:44 2017 +0000 jmv2177@columbia.edu: dynamic
    alloctaion
dc5189c - Mon Nov 13 13:59:01 2017 -0500 ryanbernstein1@gmail.com: small
    makefile change
863ec97 - Mon Nov 13 15:18:16 2017 +0000 jmv2177@columbia.edu: server to use
    endpoint with executable
d4f88b4 - Mon Nov 13 15:03:17 2017 +0000 jmv2177@columbia.edu: server to use
    executable
947231e - Sun Nov 12 22:06:11 2017 -0500 ryanbernstein1@gmail.com: started
    codegen. hello world works, but currently doesn’t depend on the actual
    string content, it just prints hello world.
27349b0 - Fri Nov 10 18:04:09 2017 -0500 ryanbernstein1@gmail.com: linking
    example
0efae9e - Fri Nov 10 17:19:17 2017 -0500 ryanbernstein1@gmail.com: added
    chapter3 Makefile
98ed8c8 - Fri Nov 10 17:08:54 2017 -0500 ryanbernstein1@gmail.com: added
    specifying the output assempy file
d5bfb4e - Fri Nov 10 16:28:48 2017 -0500 ryanbernstein1@gmail.com: added
    example assembly file for chapter3
5c20479 - Fri Nov 10 16:25:19 2017 -0500 ryanbernstein1@gmail.com: Added test
    llvm assembly output to chapter3 example
578b5d3 - Fri Nov 10 16:07:37 2017 -0500 ryanbernstein1@gmail.com: Added
    chapter3 example from Stephan’s tutorial, and updated it to work with
    stackage lts-9.12
08899ea - Tue Oct 31 17:55:04 2017 +0000 jmv2177@columbia.edu: Fixing readmen
7675f9f - Mon Oct 30 22:29:11 2017 +0000 jmv2177@columbia.edu: Fixing README
3f36980 - Mon Oct 30 22:27:42 2017 +0000 jmv2177@columbia.edu: Adding weblang
    server
a26e556 - Mon Oct 30 11:23:36 2017 -0400 ryanbernstein1@gmail.com: added some
    things i forgot: operators, includes
aad35e0 - Tue Oct 24 19:40:14 2017 -0400 ryanbernstein1@gmail.com: added
    haskell+llvm article
1ad187d - Mon Oct 23 22:02:45 2017 -0400 ryanbernstein1@gmail.com: Tokens show
```

```
     line numbers
289 c1f272a — Mon Oct 23 21:55:45 2017 —0400 ryanbernstein1@gmail.com: parse
    errors now show line/col numbers. newlines in json and type signatures
    work now — newlines do nothing when adjacent to colons, commas or arrows
290 52da20a — Mon Oct 23 20:37:20 2017 —0400 ryanbernstein1@gmail.com: pretty much
     finished parsing. lines can currently only be in json containers after
    the brackets and commas
291 80595e8 — Mon Oct 23 15:08:33 2017 —0400 ryanbernstein1@gmail.com: executable
    now called weblang
292 ca4eda7 — Mon Oct 23 15:07:38 2017 —0400 ryanbernstein1@gmail.com: Added
    pretty printing of the AST
293 68c5467 — Mon Oct 23 14:47:24 2017 —0400 ryanbernstein1@gmail.com: Mostly
    finished Lexer, added .x and .y files to sources in stack.yaml so ——force—
    dirty no longer needed
294 a8a4684 — Fri Oct 20 19:36:08 2017 —0400 ryanbernstein1@gmail.com: Added to
    and improved lexing, added a basic AST and parser, very simple
    interpretter
295 aab8eaa — Thu Oct 19 15:56:45 2017 —0400 ryanbernstein1@gmail.com: forgot to
    add build files
296 54d6fac — Thu Oct 19 15:50:27 2017 —0400 ryanbernstein1@gmail.com: Added a
    first iteration of lexing with Alex.
297 dea38ce — Thu Oct 19 15:18:09 2017 —0400 ryanbernstein1@gmail.com: added Lexer
     as module
298 3eb6dbc — Thu Oct 19 15:16:01 2017 —0400 ryanbernstein1@gmail.com: first stab
    at lexing
299 fd1f4b2 — Thu Oct 19 11:03:02 2017 —0400 ryanbernstein1@gmail.com: removed nix
    : false so I can build on NixOS
300 286f731 — Thu Oct 19 10:59:37 2017 —0400 ryanbernstein1@gmail.com: added link
301 28fe3fd — Mon Oct 16 16:05:16 2017 —0400 noreply@github.com: Merge pull
    request #1 from rybern/intall
302 ee04ff0 — Mon Oct 16 16:04:13 2017 —0400 jomivega400@gmail.com: Formatting
303 c909b0b — Mon Oct 16 16:01:46 2017 —0400 jomivega400@gmail.com: Adding Mac
    instructions with brew
304 8e41a39 — Wed Sep 20 16:50:46 2017 —0400 ryanbernstein1@gmail.com: markdown is
     hard
305 6941c02 — Wed Sep 20 16:46:03 2017 —0400 ryanbernstein1@gmail.com: added
    compilation instructions
306 dbc6e23 — Fri Sep 15 14:47:01 2017 —0400 ryanbernstein1@gmail.com: formatting
307 42c7503 — Fri Sep 15 14:44:58 2017 —0400 ryanbernstein1@gmail.com: added some
    links to README
308 4fdb3db — Fri Sep 15 14:09:38 2017 —0400 ryanbernstein1@gmail.com: Added some
    tools we'll use: Alex, Happy, LLVM
309 b947f74 — Fri Sep 15 13:41:20 2017 —0400 noreply@github.com: Initial commit
```

5. System Architecture





5.1 Compilation Process

Weblang's compiler is comprised of several files, each serving a unique purpose. There are three primary components: src, which serves as the actual translator (taking in weblang and writing out LLVM IR), client, which contains a C wrapper we wrote that is necessary for the Get/Post requests generated within LLVM the LLVM IR, and jsonlib, which contains a C++ wrapper we wrote that actually performs all object storage, as well as provides a bunch of functionality on that storage.

- src:
  - Main.hs: The program that is called to run the whole process.
  - Lexer.x: scans/lexes the program to create tokens. It does so with the help of:
    - Lexer/Types.hs: Contains list of all token types that we want recognize and lex.
    - Lexer/Utils.hs: Contains helper functions for parsing whitespace correctly.
  - Parser.y: Parses tokens passed from lexer to construct an AST.
  - AST.hs: Our representation of how a program looks.
  - Program.hs: This takes the AST produced by the parser and cleans it up for our needs.
  - Semantics.hs: This program performs semantic checking on the updated AST.
  - Codegen.hs: A helper module for LLVM.hs that contains some wrappers around our LLVM wrapper.
  - LLVM.hs: Where the magic happens.
- client:
  - client.cpp: Our wrapper to provide get/post functionality.
  - cpr-example The library we wrap around.
- jsonlib
  - jsonlib.cpp: Our wrapper for memory management/object manipulation
  - rapidjson: The library we wrap around.

These files work together through our weblang bash script, which first builds the main compiler (src), then runs the weblang file through it, then links the memory management component (jsonlib) and client in order to produce an executable.

5.1.1 Lexer.x

Takes in a stream of ASCII text and processes it into tokens. Notably, whitespace is not discarded, because it is crucial to our scoping. Instead, it is tokenized as a "Position" indicator. The functions in the Lexer/Utils.hs file provides this position saving functionality. If text that is not syntactically correct, it will be rejected at this stage (and a message printing the line number and position of the problematic character will be presented to the user).

5.1.2 Parser.y + AST.hs

The parser takes in the tokens produced by the Lexer and attempts to convert it into a correct Abstract Syntax Tree of the form displayed in AST.hs using a context free grammer. If it is deemed to be syntactically correct, it terminates, returning the AST (which is then printed). If not, it will display the line it believes to be syntactically correct.

5.1.3 Program.hs

Program.hs takes in the AST and modifies it by looking at the position tokens it finds in the AST and transforming them into scopes in the AST. It also performs some basic semantic checking from a scoping perspective, checking to make sure that foreach loops and if/else statements have bodies (as both require bodies).

5.1.4 Semantics.hs

The static semantic analyzer consumes the newly updated AST. It enforces our semi-static type system by checking types of everything it can know for sure. This includes any declared literals and functions that we know have defined input/output types. It also includes function input/output types. However, due to the semi polymorphic nature of JSON and our underlying functions, we are not always able to determine the types at compile time. This includes functions like get (which attempts to get from an array or object - because arrays/objects can contain different types, we do not know what get will return) and jn (which takes in a string and converts it to some json container, like array or object - because we do not know what the result of the string will be, we cannot determine the type). Weblang allows these to pass, assuming they are correct, and checks the types at run time.

5.1.5 Codegen.hs/LLVM.hs

LLVM.hs (with the assistance of functions written in Codegen.hs, we should took and modified from the excelent Kaleidescope for haskell tutorial) converts the AST into LLVM IR. It does so by using LLVM-hs and LLVM-hs-pure. Because memory management occurs in C++, as part of the Codegen process, it converts all primitives into calls to the jsonlib wrapper. Additionally, it is at this stage that import statements are converted into functions that call the client wrapper.

5.1.6 client.cpp + cpr-example

client.cpp is linked to every weblang executable. It contains three functions, post, get, and exposed_post. Each function uses cpr functions to create the http client, and send the http request given the URL, key, secret, header, and payload. post and get are only used internally: they are the functions called by endpoints imported via import (i.e. when Codegen converts imported endpoints into get/post requests, it uses these functions). exposed_post allows users to call post from directly within weblang, rather than forcing an import.

5.1.7 jsonlib.cpp + rapidjson

jsonlib.cpp is linked to every weblang executable. All memory management is performed by functions in this file, as well as some of the functions we expose. Everything in weblang is actually stored in a JSON representation powered by these functions. For instance, when a string is created, LLVM.hs first allocates memory for it and stores it as an array of ints, and then passes it to a jsonlib function which stores it as a json object. When the string is needed, jsonlib returns a pointer to a memory location containing an array of ints that LLVM can then use in its internal representation. Jsonlib also contains functions helper functions like concatenation, equality checking, and isType.

6. **Testing**

Seeing as Weblang is comprised of many moving parts, it was important to regularly test that each language feature worked as specified. This was accomplished by writing test programs to isolate specific Weblang functionalities and ensure that all features continued to work throughout the development process.

6.1 Testing Process

Once a new feature was developed, it could not be deemed safe to integrate into the language until it was accepted by the scanner, parser, semantic checker, and code generator. After the program's validity was ensured, we needed to test that it was actually doing what it was supposed to. To accomplish this, each Weblang test file is compared to a corresponding expected output text file.

1. Write a test file in Weblang with a corresponding text file to match the expected output.

2. Build the test file's executable using the *weblang* build script.

3. Assert that all tests have passed.

6.2 Regression Test Suite

The Weblang test suite may be executed via the *test_script.py* python script. This script iterates through each feature test file, compiles and runs it, and compares its output to a corresponding expected output text file. If the outputs match, the test passes. Otherwise, the test fails and the tester is made aware of the issue. Evaluating each test case at once helps ensure that no feature is adversely impacted by changes to the code base.

The *test_script.py* file is located at the root *plt* directory, whereas the test cases and expected output files are located at *plt/test/tests* and *plt/test/expected* respectively. Moreover, logs for tests are saved in the *plt/test/* directory, where on can check detailed logs or simple output logs.

6.3 Continuous Integration

In order to consistently run our test suite throughout development, we used Travis CI. This allowed us to always be checking whether or not our newly implemented features were breaking some existent piece of code as we contributed to the repository. The Travis output is present on Github under the review page for each pull request, making it easier for team members to check that all tests have passed before accepting a merge into the master branch.

6.4 Test Script

test_script.py

```
import os
import filecmp
import datetime

class bcolors:
     HEADER = '\033[95m'
     OKBLUE = '\033[94m'
     OKGREEN = '\033[92m'
     WARNING = '\033[93m'
     FAIL = '\033[91m'
     ENDC = '\033[0m'
     BOLD = '\033[1m'
     UNDERLINE = '\033[4m'

def compilefile(f,test,logfile,test_files,detailed_logs):
    os.system('echo "\n"[Testing '+ test+ ' at ' +str(datetime.datetime.now())
        + '] >> ' +logfile)
    os.system('./weblang '+test_files+'/'+test+'.wl > errors_warnings 2>&1' )
    os.system('./'+test+' test'+test+' a'+' > test_output 2>&1')
    #os.system('cat errors_warnings')
    #os.system('cat test_output')
    os.system('cat test_output>>'+ logfile)
    os.system('cat errors_warnings>>'+ detailed_logs)
    output = 'test_output'
    return output


##### START TEST SCRIPT #####
test_files = "test/tests"
expected_files = "test/expected"
tests = os.listdir(test_files)
expected = os.listdir(expected_files)
testcount = 0
passed = 0
logfile = 'test/test_log'
detailed_logs = 'test/detailed_log'
os.system('echo "STARTING TEST" > '+logfile)
for f in tests:
    if('.wl' in f):
        testcount+=1
        test = f.split('.')[0]
        output = compilefile(f,test,logfile,test_files,detailed_logs)
        equal = filecmp.cmp((expected_files+'/'+test),output)
        if(equal):
            print(bcolors.OKGREEN+"[Passed] "+test+bcolors.ENDC)
            passed+=1
            os.system('rm '+test)
        else:
            print(bcolors.FAIL+"[Failed] "+test+bcolors.ENDC)

```

```
os.system('rm  test_output')
os.system('rm  errors_warnings')
os.system('echo "———————————————————————— NEW TEST
    ——————————————————————————————">>'+ logfile)
print(bcolors.HEADER+bcolors.BOLD+"Passed "+str(passed)+" out of "+str(
    testcount)+" tests."+bcolors.ENDC)
try:
    assert(passed == testcount)
except AssertionError:
    exit(1)
```

6.5 Test Cases

Accessors.wl

```
testAccessors arg : Str -> Num
  a = [1,2,[3,if 1 then 4 else 0],5]
  c = a.[a.[1]].[if 1 then 1 else 0] // should be 4
  log c
  d = {hi:"yo",red:"5",p:5}
  log d.["hi"]
  log d.["p"]
```

Accessors.wl - Expected Output

```
4
yo
5
```

AddSub.wl

```
testAddSub arg : Str -> Num
  x = 5
  y = 7
  z = y - x
  log z
  j = y+z
  log j
```

AddSub.wl - Expected Output

```
2
9
```

AllTypes.wl

```
testAllTypes arg : Str -> Bool
  a = 5
  b = "hi"
  c = ["hi", 5]
  d = isString a
  e = isString b
  f = isNum a
  g = isNum b
  h = isArr c
  i = isArr (c.[0])
  j = isString c.[0]
  k = jn "{\"hi\":\"3\"}"
  l = isString k
  m = isNum k
  n = isObj k
  o = isObj a
  p = isObj b
  q = true
  r = isObj q
  s = isBool q
  t = isNum q
  log d
  log e
  log f
  log g
  log h
  log i
  log j
  log k
  log l
  log m
  log n
  log o
  log p
  log q
  log r
  log s
  log t
```

AllTypes.wl - Expected Output

```
false
true
true
false
true
false
true
{hi:3}
false
false
true
false
false
true
false
true
false
```

Arg.wl

```
testArg arg : Str -> Str
  logThis "Arguments work"

logThis arg : Str -> Str
  log arg
```

Arg.wl - Expected Output

```
Arguments work
```

Array.wl

```
testArray arg : Str -> Str
  array [5,0]

array arg : Arr -> Str
  a = 5
  b = geta [[5, 3], 0]
  log b
  d = ["right", "wrong"]
  e = geta [d, 0]
  log e
  ["hey", "what"]
  [4, "four"]
  log a
  log 5
  log arg.[0]
  "bye"
```

Array.wl - Expected Output

```
5
right
5
5
5
```

Assert.wl

```
testAssert arg : Str -> Str
  x = 90
  assert x > 80
  log "one assert good"
  assert x > 100
  "bye"
```

Assert.wl - Expected Output

```
one assert good
Assertion failed!
```

Average.wl

```
include "examples/stdlib.wl"

testAverage arg : Str -> Str
  x = [8,2]
  y = [10,20,30]
  log (avg x)
  z = avg y
  log z
  "bye"
```

Average.wl - Expected Output

```
5
20
```

Binops.wl

```
testBinops arg : Str -> Num
  log (8/2)
  log (5+3)
```

Binops.wl - Expected Output

```
4
8
```

Bools.wl

```
testBools arg : Str -> Str
  log (false || true)
  log (false || false)
  log (true && false)
  log true
  log false
  "string return"
```

Bools.wl - Expected Output

```
true
false
false
true
false
```

Cat.wl

```
testCat arg : Str -> Str
  a = "h"
  b = "ello"
  log (cat [a,"ello"])
  log (cat [a,b])
```

Cat.wl - Expected Output

```
hello
hello
```

Conditional.wl

```
testConditional arg : Str -> Str
  x = true
  if x
    log "Inside if"
  else
    log "Inside else"

  log "Should be inside if above me"
```

Conditional.wl - Expected Output

```
Inside if
Should be inside if above me
```

ConditionalElse.wl

```
testConditionalElse arg : Str -> Str
  x = false
  if x
    log "Inside if"
  else
    log "Inside else"

  log "Should be inside else above me"
```

ConditionalElse.wl - Expected Output

```
Inside else
Should be inside else above me
```

For.wl

```
testFor arg : Str -> Arr
  foreach i in [1,2,3,4]
    log i
```

For.wl - Expected Output

```
1
2
3
4
```

ForReassign.wl

```
testForReassign arg : Str -> Num
  x = 1
  foreach i in [1,2,3,4]
    x = x + 1
  log x
```

ForReassign.wl - Expected Output

```
5
```

Gcd.wl

```
testGcd arg : Str -> Str
  x = [27,18]
  gcd x
  gcd [36,12]
  "bye"
```

Gcd.wl - Expected Output

```
9
12
```

Get.wl

```
import {url: "https://api.gdax.com/products/", key : "", secret: "", header
    :"",
  endpoints:[{fnName:"getEtherPrice", endpoint:"eth-usd/ticker", is_post:false
    }] }

testGet arg : Str -> Bool
  response = getEtherPrice arg
  res = jn response
  check = isObj res
  log check
```

Get.wl - Expected Output

```
true
```

isArr.wl

```
testIsArr arg : Str -> Bool
  a = ["one",5]
  b = 7
  check1 = isArr a
  check2 = isArr b
  check3 = isArr a.[1]
  log check1
  log check2
  log check3
```

isArr.wl - Expected Output

```
true
false
false
```

isBool.wl

```
testIsBool arg : Str -> Bool
  a = 5
  b = true
  c = "true"
  d = isNum a
  check1 = isBool a
  check2 = isBool b
  check3 = isBool c
  check4 = isBool d
  log check1
  log check2
  log check3
  log check4
```

isBool.wl - Expected Output

```
false
true
false
true
```

isNum.wl

```
testIsNum arg: Str -> Bool
  a = 5
  b = "hi"
  c = jn "{\"one\": 6}"
  d = (get [c, "one"])
  check1 = isNum a
  check2 = isNum b
  check3 = isNum c
  check4 = isNum d
  log check1
  log check2
  log check3
  log check4
```

isNum.wl - Expected Output

```
true
false
false
true
```

isObj.wl

```
testIsObj arg : Str -> Bool
  a = 5
  b = "hi"
  c = ["hi", 5]
  d = isString c.[0]
  e = jn "{\"hi\":\"3\"}"
  f = jn "{\"test\":{\"one\":\"two\"},\"arr\":[1,2,3],\"num\":7}"
  check1 = isObj a
  check2 = isObj b
  check3 = isObj c
  check4 = isObj d
  check5 = isObj e
  check6 = isObj f
  check7 = isObj f.["test"]
  check8 = isObj f.["arr"]
  check9 = isObj f.["num"]
  log check1
  log check2
  log check3
  log check4
  log check5
  log check6
  log check7
  log check8
  log check9
```

isObj.wl - Expected Output

```
false
false
false
false
true
true
true
false
false
```

isString.wl

```
testIsString arg : Str -> Bool
  a = "yes"
  b = 9
  c = ["yes again",9]
  check1 = isString a
  check2 = isString b
  check3 = isString c
  check4 = isString c.[0]
  check5 = isString c.[1]
  log check1
  log check2
  log check3
  log check4
  log check5
```

isString.wl - Expected Output

```
true
false
false
true
false
```

JsonAdd.wl

```
testJsonAdd arg : Str -> Str
  testjson = jn "{\"test\":\"Json get works\"}"
  result = get [testjson,"test"]
  log result
  added = addToObj [testjson, "test2", "Json add works"]
  test = get [added,"test2"]
  log test
```

JsonAdd.wl - Expected Output

```
Json get works
Json add works
```

JsonDoubles.wl

```
testJsonDoubles arg : Str -> Str
  x = jn "{\"one\": 69, \"two\":\"get your mind out of the gutter\"}"
  log (get [x, "one"])
  log (get [x, "two"])
```

JsonDoubles.wl - Expected Output

```
69
get your mind out of the gutter
```

Log.wl

```
testLog arg : Str -> Str
  log "Logging works"
```

Log.wl - Expected Output

```
Logging works
```

Mod.wl

```
testMod arg : Str -> Str
  x = 10
  y = 2
  z = 3
  log(x%y)
  log(x%z)
  "str return"
```

Mod.wl - Expected Output

```
0
1
```

Post.wl

```
import {url: "https://hooks.slack.com/services/T74RW7J0N/B891X5YNN/", key: "",
    secret:"", header:"",
  endpoints:[{fnName:"sendSlackMsg", endpoint:"BaQHlflLTmQQNKHH3EE6PrR1",
      is_post:true}] }

testPost arg : Str -> Obj
  body = {}
  body = addToObj [body, "text", "Running test suite"]
  body = addToObj [body, "channel", "#testing"]
  x = sendSlackMsg body
  log x
  body
```

Post.wl - Expected Output

```
ok
```

PostCondition.wl

```
testPostCondition arg : Str -> Num
  x = "string"
  log x
```

PostCondition.wl - Expected Output

```
string
Post-condition not met in function testPostCondition
```

Pre.wl

```
testPre arg : Num -> Str
  x = "string"
  log x
  x
```

Pre.wl - Expected Output

```
Pre-condition not met in function testPre
```

StrEquality.wl

```
testStrEquality arg : Str -> Str
  x = "hello"
  y = "hola"
  z = "hello"
  w = "hello "
  log (equals [x,"hello"])
  log (equals [x,y])
  log (equals [x,z])
  log (equals [x,w])
  "bye"
```

StrEquality.wl - Expected Output

```
true
false
true
false
```

Type.wl

```
type A a : Num
  log "check a"
  a > 0

type B b : A
  log "check b"
  b > 1

type C c : B
  log "check c"
  c > 2

testType arg : Str -> Str
  x = 8
  if x :? C
    log "matches"
  else
    log "doesn't match"

  y = 1
  if y :? C
    log "matches"
  else
    log "doesn't match"
```

Type.wl - Expected Output

```
check a
check b
check c
matches
check a
check b
check c
doesn't match
```

Type.wl

```
type A a : Num
  log "check a"
  a > 0

type B b : A
  log "check b"
  b > 1

type C c : B
  log "check c"
  c > 2

testType arg : Str -> Str
  x = 8
  if x :? C
    log "matches"
  else
    log "doesn't match"

  y = 1
  if y :? C
    log "matches"
  else
    log "doesn't match"
```

Type.wl - Expected Output

```
check a
check b
check c
matches
check a
check b
check c
doesn't match
```

Var.wl

```
testVar arg : Str -> Str
  variable = "Variables work"
  log variable
```

Var.wl - Expected Output

```
Variables work
```

7. Example Programs

Our example programs are focused mainly on interacting with the slack API, the messaging API (to send text messages), and several cryptocurrency exchange API's. We pull information from one API and pass it to another, to show off the usability of weblang for interacting with these RESTful services. While the sample programs are not varied in the content of the API's (i.e. too much crypto), they correctly show off the ease of use and functionality that this language has to offer.

7.1 Sending a Slack Message

This program imports the slack webhook endpoint, and uses it to send a message passed in via the slacks function argument. It could be called from the command line or from a different file (or the same one) using include.

```
import {url: "https://hooks.slack.com/services/T74RW7J0N/B891X5YNN/",
  key: "",
  secret:"",
  header: "",
  endpoints:
      [{fnName:"sendSlackMsg", endpoint:"BaQHlflLTmQQNKHH3EE6PrR1", is_post:
         true}] }

slack arg : Str -> Obj
  sendSlackMsg {text: arg}
  {}
```

7.2 Crypto Currency: Voice to price

This program is one of the more involved ones we have written in weblang. While it may not be the prettiest to look at, it does a good job in displaying includes, control flow, and object management in a variety of ways. The program itself receives as an argument a coin name and an output name, either slack or text (during our demonstration, these arguments were received using a phone via voice, hence the name voice to price). With those arguments, the program determines what endpoint it should call to send (via text or slack) the latest price of the specified cryptocurrency. If the input is average, the program will call the getAvgPrice function included in the `bitcoin_average.wl` file, which gets bitcoin prices from 5 different exchanges and determines the average price among them.

```
include "examples/bitcoin_average.wl"

processMsg arg : Arr -> Obj
```

```
count = 0
prices = []
price = 0
foreach x in arg
  if count==0
    count = count+1
  else
    if count==1
      count = count+1
    else
      if(equals [x, "average"])
        price = getAvgPrice ""
      else
        if(equals [x, "litecoin"])
          price = litecoin ""
        else
          if(equals [x, "ethereum"])
            price = ether ""
          else
            if(equals [x, "bitcoin"])
              price = bitcoin ""
            else
              price = 0
              er = cat [x," not found"]
              log er
      sendtext = ""
      if(equals [x, "average"])
        sendtext = "bitcoin average price is $"
      else
        sendtext = cat [x, " price is $"]
      sendtext = cat [sendtext, price]
      prices = push [prices, sendtext]
      log sendtext
st = ""
foreach p in prices
  st = cat [st, p]
  st = cat [st, "\n"]
if (equals [arg.[1], "slack"])
  js="{\"text\":\""
  js=cat [js,st]
  js=cat [js,"\"}"]
  payload = jn js
  sendSlackMsg payload
  payload
else
  if (equals [arg.[1], "text"])
    payload = {}
    payload = addToObj [payload, "message", (cat [st,""])]
    sendJordanTxt payload
    payload
  else
    er = cat [arg.[1], " not found"]
    log er
```

7.3 Bitcoin Average Price

The file included by program mentioned above. Gets prices from 5 different exchanges and takes the average. Note that the included file `coin_helpers.wl` has the necessary imports to call the endpoints at each exchange. Notice how these functions are defined as helper functions and are therefore not exposed to be called as endpoints when running the server.

```
include "examples/coin_helpers.wl"
include "examples/stdlib.wl"

helper getAvgPrice arg : Str -> Str
  arr = []
  gdaxprice = gdax arg
  cexprice = cex arg
  bitfinexprice = bitfinex arg
  bitstampprice = bitstamp arg
  arr = [gdaxprice, cexprice, bitfinexprice, bitstampprice]
  geminiprice = gemini arg
  arr = push [arr, geminiprice]
  average = avg arr
  x = cat ["",average]
  x

helper gdax arg : Str -> Num
  x = getBitcoinPrice arg
  res = jn x
  precio = (get [res, "price"])
  if isString precio
    precio = toNum precio
  else
    0
  precio

helper cex arg : Str -> Num
  x = cexBitcoinPrice arg
  res = jn x
  precio = get [res, "ask"]
  if isString precio
    precio = toNum precio
  else
    0
  precio

helper bitfinex arg : Str -> Num
  x = bitfinexBitcoinPrice arg
  res = jn x
  precio = res.[0]
  if isString precio
    precio = toNum precio
  else
    0
  precio

helper gemini arg : Str -> Num
```

```
  x = geminiBitcoinPrice arg
  res = jn x
  precio = get [res, "ask"]
  if isString precio
    precio = toNum precio
  else
    0
  precio

helper bitstamp arg : Str -> Num
  x = bitstampBitcoinPrice arg
  res = jn x
  precio = get [res, "ask"]
  if isString precio
    precio = toNum precio
  else
    0
  precio
```

7.4 Get Latest Prices

Also included above, this program makes a call to three different endpoints on gdax, getting the price of the assets listed, accessing the json for the correct pairing, and returning the price.

```
include "examples/coin_imports.wlh"

import {url: "https://api.gdax.com/products/",
  key:"",
  secret:"",
  header:"",
  endpoints:
      [{fnName:"getBitcoinPrice", endpoint:"btc-usd/ticker", is_post:false},
      {fnName:"getEtherPrice", endpoint:"eth-usd/ticker", is_post:false},
    {fnName:"getLitecoinPrice", endpoint:"ltc-usd/ticker", is_post:false}] }

bitcoin arg : Str -> Str
  x = getBitcoinPrice arg
  res = jn x
  precio = (get [res, "price"])
  precio

ether arg : Str -> Str
  x = getEtherPrice arg
  res = jn x
  precio = (get [res, "price"])
  precio

litecoin arg : Str -> Str
  x = getLitecoinPrice arg
  res = jn x
  precio = (get [res, "price"])
  precio
```

8. Team Reflection

8.1 Ryan Bernstein

I really enjoyed working on this project. While I've had experience building medium-large pieces of software before, I don't have very much experience building it closely with a group as large as five. We had a good time, and working together was much easier than I would have expected from a group our size. It was very helpful to assign roles, especially because our language implementation naturally segmented into domains like data types, codegen and networking. The use of great tools like GitHub and Slack also helped a lot.

Language-wise, it was interesting to see how our original ideas were replaced by reality - we hedged some of our more ambitious features, like the more complicated nested type system, but we also nailed some of our stretch goals like declarative API specification with OAuth support. Many of the things I thought would be easily, like global constants and runtime data types, turned out to be very challenging, while things like nested primitives, turned out easier. I feel like I now have a much better idea of where the work is distributed in language building.

8.2 Brendan Burke

Working on this involved semester long undertaking provided a great opportunity to both apply the concepts we were learning in the course and also learn to develop a product as a team. Having weekly TA meetings in addition to our regularly scheduled group meetings kept us focused on the task at hand and assured that we didn't procrastinate important aspects of the project. I don't think finishing this product would be possible without the strong system of communication we had via Slack, as we were able to constantly be in touch with one another and separate different aspects of the project into their own channels within our Slack group. Here we would post weekly assignments for the team to have completed by the next meeting so that we were always making gradual progress towards completing our goal.

An understated aspect of this project that I think is incredibly important is choosing a product the entire team is genuinely interested in. We all agreed that the existent methods of communicating with RESTful APIs left much to be desired, and we were determined to develop a product to address the issue. Now that Weblang is complete with the functionalities we originally had in mind, I can honestly say that it is a tool I would gladly use going forward with API related data-integrated development.

8.3 Christophe Rimann

I thought this project was super interesting. Prior, I had never really done any functional programming (beyond dipping my toes in it with Python), and at first I had a really hard time wrapping around it. At some point over thanksgiving, though, I finally wrapped my head around Monads; once I got that down, I actually really liked it. I also really liked getting my hands dirty with memory management. I had had some exposure to pointers/memory management from Advanced Programming, but nothing like this. We chose to use the rapidjson library to hold all our objects in memory, and although rapidjson was a great at parsing json, it was not meant to maintain memory in the way we used it. That meant we had to really abuse the library in order to get it to work in the way we intended (for instance, all our pointers are int * because that is the closest thing LLVM has to void *). This was both challenging and really fun (for instance, array access under the hood looks kind of like: (int *)(&((*((Document*)d))[idx]))). Overall, this project was stressfull at times, but overall really fun.

8.4 Julian Serra

This project was very useful in teaching us how to correctly and efficiently assign roles and responsibilities. It was tremendously important to assign todos within the team that were achievable within a shorter period of time, and not assign huge responsibilities that seemed abstract and would leave people unsure of where to begin. Weekly meetings and sprints make the work more manageable, and allow for making steady progress throughout the semester. Communication is key and testing, continuous integration, and code reviews are tremendously important. Attempting a project like this without version control would be like attending class naked: doable, but idiotic.

8.5 Jordan Vega

Working on Weblang was really fun and I enjoyed applying what was taught in class along with concepts and skills learned while taking Advanced Programming. It also made it possible to create demos in the space of Internet of Things. Taking a LISP class concurrently helped me understand the Ocaml snippets in class and made Haskell attractive. At first, I was not a huge fun of functional programming, yet after completing this project, I was amazed with how much Haskell could do. We wrote some Haskell code, that can parse and produce infinitely more code than what was written.

I enjoyed getting more exposure to pointers/memory management, using LLVM to understand lower level programming. Furthermore, creating the server and client libraries were fun, as we had to create them as generic as possible so that they could work with as many APIs as possible.

Lastly was also a good experience of working in a team to meet weekly deliverables and coordinating tasks and meetings. On top of that, doing code reviews for other teammates really helped me improve my haskell, LLVM, and course understanding.

8.6 General Advice For Future Teams

Begin the process as soon as possible and prioritize organization. Having an entire semester to complete this project makes it easy to procrastinate and ultimately compress the bulk of the workload into a short timespan. Avoiding this is key, as both the quality of your work and your sanity will begin to diminish the longer you put off meeting your project milestones. Also, it is important to have a reliable method of communicating with the entire team, such as a Slack group with separate channels to categorize discussions.

9. Weblang Code Listing

9.1 Lexer.x

```
{
module Lexer (
    tokenize
  , LexToken (..)
  ) where

import Lexer.Types
import Lexer.Utils
}

/%wrapper "posn"

$digit = 0-9
$alpha = [a-zA-Z]
$newline = [\n\r\f]
$space = [\ ]
@empty_lines = ($newline ($space* $newline)*)+

tokens :-
  \" ( \n | [^\"\\] | \\. )* \"                               { \pos s -> withPos pos $ QuoteToken (parseQuoted s) }
  "/*" ( $newline | [^\*] | \*+ ($newline | [^\/]) )* "*/"    ;
  ^$space+                                                    { \pos s -> withPos pos $ IndentToken (length s) }
  $space* @empty_lines $space+                                { \pos -> withPos pos . IndentToken . length . takeWhile (== ' ') . reverse }
  @empty_lines                                                { \pos s -> withPos pos $ NewlineToken }
  $white+                                                     ;
  "//".*                                                      ;
  \-? $digit+ (\. $digit+)?                                   { \pos s -> withPos pos $ NumberToken (read s) }
  \.                                                          { \pos s -> withPos pos $ DotToken }
  "if"                                                        { \pos s -> withPos pos $ IfToken }
  "then"                                                      { \pos s -> withPos pos $ ThenToken }
  "else"                                                      { \pos s -> withPos pos $ ElseToken }
  "foreach"                                                   { \pos s ->
     withPos pos $ ForeachToken }
  "in"                                                        { \pos s ->
     withPos pos $ InToken }
  "do"                                                        { \pos s ->
     withPos pos $ DoToken }
  "type"                                                      { \pos s ->
     withPos pos $ TypeToken }
  "helper"                                                    { \pos s ->
     withPos pos $ HelperToken }
  "assert"                                                    { \pos s ->
     withPos pos $ AssertToken }
  "include"                                                   { \pos s ->
     withPos pos $ IncludesToken }
  "import"                                                    { \pos s ->
     withPos pos $ ImportToken }
  "null"                                                      { \pos s ->
     withPos pos $ NullToken }
  "true"                                                      { \pos s ->
     withPos pos $ TrueToken }
  "false"                                                     { \pos s ->
     withPos pos $ FalseToken }
  \[                                                          { \pos s ->
     withPos pos $ LeftSquareBracketToken }
  \]                                                          { \pos s ->
     withPos pos $ RightSquareBracketToken }
  \(                                                          { \pos s ->
     withPos pos $ LeftParenToken }
  \)                                                          { \pos s ->
     withPos pos $ RightParenToken }
  \{                                                          { \pos s ->
     withPos pos $ LeftCurlyBracketToken }
  \}                                                          { \pos s ->
     withPos pos $ RightCurlyBracketToken }
  \,                                                          { \pos s ->
     withPos pos $ CommaToken }
  \:\?                                                        { \pos s ->
     withPos pos $ ColonQueToken }
  \:\!                                                        { \pos s ->
     withPos pos $ ColonExcToken }
  \:                                                          { \pos s ->
     withPos pos $ ColonToken }
  "->"                                                        { \pos s ->
     withPos pos $ ArrowToken }
  $alpha [$alpha $digit \_ \']*                               { \pos s ->
     withPos pos $ VarToken s }
  \+                                                          { \pos s ->
     withPos pos $ PlusToken }
  \-                                                          { \pos s ->
     withPos pos $ MinusToken }
  \*                                                          { \pos s ->
     withPos pos $ MultiplyToken }
  \/                                                          { \pos s ->
     withPos pos $ DivideToken }
  \/%                                                        { \pos s ->
     withPos pos $ ModToken }
  \=\=                                                       { \pos s ->
     withPos pos $ EQToken }
  \=                                                         { \pos s ->
     withPos pos $ EqualsToken }
  \<\=                                                       { \pos s ->
     withPos pos $ LEQToken }
  \>\=                                                       { \pos s ->
     withPos pos $ GEQToken }
  \<                                                         { \pos s ->
     withPos pos $ LTToken }
  \>                                                         { \pos s ->
     withPos pos $ GTToken }
  \|\|                                                       { \pos s ->
     withPos pos $ OrToken }
  \&\&                                                       { \pos s ->
     withPos pos $ AndToken }

{
tokenize :: String -> [Pos LexToken]
tokenize = normalizeNewlines . alexScanTokens

withPos :: AlexPosn -> a -> Pos a
withPos (AlexPn _ line col) a = Pos line col a
}
```

9.2 Parser.y

```
{
module Parser (parse) where

import qualified Data.Map as Map
import Data.Map (Map)
import Data.Monoid
import Prelude hiding (EQ, LEQ, GEQ, GT, LT)

import Lexer.Types
import AST
}

/%name parse
/%tokentype  { Pos LexToken }
/%error      { happyError }

/%token
  quoted    { Pos _ _ (QuoteToken $$) }
  '+'       { Pos _ _ (PlusToken) }
  '-'       { Pos _ _ (MinusToken) }
  '*'       { Pos _ _ (MultiplyToken) }
  '/'       { Pos _ _ (DivideToken) }
  '%'       { Pos _ _ (ModToken) }
  '=='      { Pos _ _ (EQToken) }
  '<='      { Pos _ _ (LEQToken) }
  '>='      { Pos _ _ (GEQToken) }
  '<'       { Pos _ _ (LTToken) }
  '>'       { Pos _ _ (GTToken) }
  '||'      { Pos _ _ (OrToken) }
  '&&'      { Pos _ _ (AndToken) }
  '['       { Pos _ _ (LeftSquareBracketToken) }
  ']'       { Pos _ _ (RightSquareBracketToken) }
  '{'       { Pos _ _ (LeftCurlyBracketToken) }
  '}'       { Pos _ _ (RightCurlyBracketToken) }
  '('       { Pos _ _ (LeftParenToken) }
  ')'       { Pos _ _ (RightParenToken) }
  ','       { Pos _ _ (CommaToken) }
  '.'       { Pos _ _ (DotToken) }
  '='       { Pos _ _ (EqualsToken) }
  ':'       { Pos _ _ (ColonToken) }
  ':?'      { Pos _ _ (ColonQueToken) }
  ':!'      { Pos _ _ (ColonExcToken) }
  arrow     { Pos _ _ (ArrowToken) }
  var       { Pos _ _ (VarToken $$) }
  line      { Pos _ _ (NewlineToken) }
  indent    { Pos _ _ (IndentToken $$) }
  num       { Pos _ _ (NumberToken $$) }
  helper    { Pos _ _ (HelperToken) }
  null      { Pos _ _ (NullToken) }
  true      { Pos _ _ (TrueToken) }
  false     { Pos _ _ (FalseToken) }
  if        { Pos _ _ (IfToken) }
  then      { Pos _ _ (ThenToken) }
  else      { Pos _ _ (ElseToken) }
  foreach   { Pos _ _ (ForeachToken) }
  in        { Pos _ _ (InToken) }
  do        { Pos _ _ (DoToken) }
  type      { Pos _ _ (TypeToken) }
  assert    { Pos _ _ (AssertToken) }
  includes  { Pos _ _ (IncludesToken) }
  import    { Pos _ _ (ImportToken) }

/%%

Program
  : line TopLevel Program       { $2 <> $3 }
  | line TopLevel               { $2 }

TopLevel
  : FunctionDeclaration        { AST [] [] [] [$1] [] }
  | Constant                   { AST [] [] [$1] [] [] }
  | CustomType                 { AST [] [$1] [] [] [] }
  | Includes                   { AST [$1] [] [] [] [] }
  | Import                     { AST [] [] [] [] [$1] }

Import
  : import Term                { Import $2 }

```

```
Includes
  : includes quoted            { Includes $2 }

Constant
  : var ’=’ Term               { ($1, $3) }

CustomType
  : type var var ’:’ Type               { ($2, NewType $5 $3 []) }
  | type var var ’:’ Type Expressions  { ($2, NewType $5 $3 $6) }

FunctionDeclaration
  : var var ’:’ Type arrow Type Expressions           { ($1, Function $4 $6 $2 $7 False) }
  | helper var var ’:’ Type arrow Type Expressions    { ($2, Function $5 $7 $3 $8 True) }

Type
  : var ’{’ Term ’}’  { Type $1 (Just $3) }
  | var               { Type $1 Nothing }

Expressions
  : indent Expression Expressions { ($1, $2) : $3 }
  | indent Expression             { [($1, $2)] }

Expression
  : var ’=’ Term  { Assignment /$1 $3 }
  | Term          { Unassigned $1 }
  | assert Term1  { Assert $2 }

Term
  : ForeachInDo           { $1 }
  | foreach var in Term6  { ForeachIn $2 $4 }
  | if Term1              { If $2 }
  | IfThenElse            { $1 }
  | Term1                 { $1 }

Term1
  : Term1 ’||’ Term2      { OperatorTerm Or $1 $3 }
  | Term2                 { /$1 }

Term2
  : Term2 ’&&’ Term3      { OperatorTerm And $1 $3 }
  | Term3                 { /$1 }

Term3
  : Term4 ’==’ Term4       { OperatorTerm EQ $1 $3  }
  | Term4 ’>=’ Term4       { OperatorTerm GEQ $1 $3  }
  | Term4 ’<=’ Term4       { OperatorTerm LEQ $1 $3  }
  | Term4 ’>’ Term4        { OperatorTerm GT $1 $3  }
  | Term4 ’<’ Term4        { OperatorTerm LT $1 $3  }
  | Term4                  { /$1 }

Term4
  : Term4 ’+’ Term5       { OperatorTerm Plus $1 $3  }
  | Term4 '-' Term5       { OperatorTerm Minus $1 $3  }
  | Term5                 { /$1 }

Term5
  : Term5 '*' Term6       { OperatorTerm Multiply $1 $3  }
  | Term5 '/' Term6       { OperatorTerm Divide $1 $3  }
  | Term5 '/%' Term6       { OperatorTerm Modulus $1 $3  }
  | Term5 ':!' Type       { TypeAssert $1 $3  }
  | Term5 ':?' Type       { TypeCheck $1 $3  }
  | Term6                 { $1 }

Term6
  : var Term7                 { FunctionCall $1 $2 }
  | else                      { Else }
  | do                        { Do }
  | Term7                     { $1 }

Term7
  : '(' Term ')'          { $2 }
  | var                   { Variable $1 }
  | Literal               { Literal $1 }
  | Term7 '.' '[' Term ']'   { Accessor $1 $4 }

IfThenElse
  : if Term1 then Term else Term1  { IfThenElse $2 $4 $6 }

ForeachInDo
  : foreach var in Term1 do Term1  { ForeachInDo $2 $4 $6 }

Literal
  : quoted                            { (StrVal $1) }
  | num                               { (NumVal $1) }
  | '[' ']'                           { ArrVal [] }
  | '[' indent ']'                    { ArrVal [] }
  | '[' ArrayTerms indent ']'         { ArrVal $2 }
  | '[' indent ArrayTerms ']'         { ArrVal $3 }
  | '[' indent ArrayTerms indent ']'  { ArrVal $3 }
  | '[' ArrayTerms ']'                { ArrVal $2 }
  | '{' '}'                           { ObjVal Map.empty }
  | '{' indent '}'                    { ObjVal Map.empty }
  | '{' ObjectTerms indent '}'        { (ObjVal $2) }
  | '{' indent ObjectTerms '}'        { (ObjVal $3) }
  | '{' indent ObjectTerms indent '}' { (ObjVal $3) }
  | '{' ObjectTerms '}'               { (ObjVal $2) }
  | null                              { NullVal }
  | true                              { TrueVal }
  | false                             { FalseVal }

ArrayTerms
  : Term ',' ArrayTerms           { $1 : $3 }
  | Term                          { [ $1 ] }

ObjectTerms
  : var ':' Term ',' ObjectTerms  { Map.insert $1 $3 $5 }
  | var ':' Term                  { Map.singleton $1 $3 }

{
happyError :: [Pos LexToken] -> a
happyError (Pos line col t:ts) = error $ "Parse error on token at line " ++
    show line ++ " col " ++ show col ++ ". Token:\n    " ++ show t ++ "\n"
}
```

9.3 AST.hs

```
{-# LANGUAGE DeriveGeneric, DeriveAnyClass, FlexibleInstances #-}
module AST where

import qualified Data.Map as Map
import Data.Map (Map)

-- for pretty printing
import GHC.Generics
import Text.PrettyPrint.GenericPretty

type ValName = String
type FnName = String
type TypeName = String
type OperatorName = String
type ExpressionBlock = [(Int, Expression)]

data AST = AST {
    includes :: [Includes]
  , customTypes :: [(TypeName, NewType)]
  , constants :: [(ValName, Term)]
  , fnDeclarations :: [(FnName, Function)]
  , imports :: [Import]
  } deriving (Show, Generic, Out)

data Import = Import {
    server :: Term
  } deriving (Show, Generic, Out)

data Includes = Includes {
    sourceAddress :: String
  } deriving (Show, Generic, Out)

data Type = Type {
    parentType :: TypeName
  , predicate :: Maybe Term
  } deriving (Show, Generic, Out)

data NewType = NewType {
    shortType :: Type
  , inhabitant :: ValName
  , longPredicate :: ExpressionBlock
  } deriving (Show, Generic, Out)

```

```
data Function = Function {
    inputType :: Type
  , outputType :: Type
  , arg :: ValName
  , body :: ExpressionBlock
  , helper :: Bool
  } deriving (Show, Generic, Out)

data Expression = Assignment ValName Term
                | Unassigned Term
                | Assert Term
                deriving (Show, Generic, Out)

data Term = Variable ValName
          | Accessor Term Term
          | FunctionCall FnName Term
          | OperatorTerm Operator Term Term
          | Literal PrimValue
          | If Term
          | Else
          | IfThenElse Term Term Term
          | ForeachInDo ValName Term Term
          | ForeachIn ValName Term
          | Do
          | TypeCheck Term Type
          | TypeAssert Term Type
          deriving (Show, Generic, Out)

data Operator = Plus
              | Minus
              | Multiply
              | Divide
              | Modulus
              | EQ
              | LEQ
              | GEQ
              | GT
              | LT
              | And
              | Or
              deriving (Show, Generic, Out, Eq, Ord)

data PrimValue = StrVal String
               | NumVal Double
               | ArrVal [Term]
               | ObjVal (Map String Term)
               | NullVal
               | TrueVal
               | FalseVal
               deriving (Show, Generic, Out)

instance Monoid AST where
  mempty = AST [] [] [] [] []
  mappend (AST ais ats acs afs ams) (AST bis bts bcs bfs bms) =
```

```haskell
    AST (ais ++ bis) (ats ++ bts) (acs ++ bcs) (afs ++ bfs) (ams ++ bms)

-- for pretty printing maps
instance (Out a, Out b) => Out (Map a b) where
  docPrec i a = docPrec i (Map.toList a)
  doc a = doc (Map.toList a)
  docList as = docList (map Map.toList as)
```

9.4 Program.hs

```haskell
{-# LANGUAGE DeriveGeneric, DeriveAnyClass, FlexibleInstances #-}
module Program ( module X
               , astToProgram
               , Program (..)
               , ExpressionBlock (..)
               , Expression (..)
               , Term (..)
               , Function (..)
               , PrimValue (..)
               , Import (..)
               , Type (..)
               , PrimType (..)
               , Endpoint (..)
               , Method (..)
               ) where

import qualified Data.Map as Map
import Data.Map (Map)
import qualified AST as AST
import Control.Monad.State
import Control.Monad.Loops
import Data.Graph
import Data.Maybe
import Data.List
import AST as X
       ( AST
       , Operator (..)
       , ValName (..)
       , FnName (..)
       , TypeName (..)
       , OperatorName (..)
       , NewType (..)
       )
import GHC.Generics
import Text.PrettyPrint.GenericPretty

data Type = Type {
    predicates :: [(ValName, ExpressionBlock)]
  , baseType :: PrimType
  } deriving (Show, Generic, Out)

type TypeMap = Map TypeName Type

```

```haskell
data PrimType = StrType
              | NumType
              | ArrType
              | ObjType
              | NullType
              | BoolType
              deriving (Show, Generic, Out, Eq)

defaultInhabitant = "val"

topologicalOrder :: (Show b, Show a, Ord a) => (b -> [a]) -> [(a, b)] -> [(a, b)]
topologicalOrder f = map (\(b, a, _) -> (a, b)) . map unSCC . stronglyConnCompR . map (\(a, b) -> (b, a, f b))
  where unSCC (AcyclicSCC node) = node
        unSCC (CyclicSCC nodes) =
          error $ "There is a cycle in the type definitions for the types: " ++ show nodes

transTypes :: [(TypeName, AST.NewType)] -> TypeMap
transTypes astTypes = foldl' addType initialTypes ordered
  where ordered = topologicalOrder (\t -> [AST.parentType (AST.shortType t)]) astTypes
        initialTypes = let fnCheck f = ("val", [Unassigned $ FunctionCall f (Variable "val")])
                       in Map.fromList [ ("Str", Type [fnCheck "isString"] StrType)
                                       , ("Num", Type [fnCheck "isNum"] NumType)
                                       , ("Arr", Type [fnCheck "isArr"] ArrType)
                                       , ("Obj", Type [fnCheck "isObj"] ObjType)
                                       , ("Null", Type [] NullType)
                                       , ("Bool", Type [fnCheck "isBool"] BoolType)
                                       ]
        addType m (name, astType) = Map.insert name (transType m astType) m

transInlineType :: TypeMap -> AST.Type -> Type
transInlineType m (AST.Type parentName shortPred) =
  case parentName `Map.lookup` m of
    Nothing -> error $ "Type " ++ parentName ++ " not found"
    Just (Type parentPreds baseType) ->
      Type {
          baseType = baseType
        , predicates = parentPreds ++
                       maybeToList ((\term -> ( defaultInhabitant
                                              , [Unassigned $ transSimpleTerm m term]))
                                     <$> shortPred)
        }

transType :: TypeMap -> AST.NewType -> Type
transType m (AST.NewType (AST.Type parentName shortPred) valName longPred) =
  case parentName `Map.lookup` m of
    Nothing -> error $ "Parent type " ++ parentName ++ " not found"
    Just (Type parentPreds baseType) ->
      Type {
          baseType = baseType
        , predicates = parentPreds ++
                       [(valName, transExpressions m longPred)] ++
                       maybeToList ((\term -> ( defaultInhabitant
                                              , [Unassigned $ transSimpleTerm
                                                 m term]))
                                    <$> shortPred)
        }

indentIncrement = 2

astToProgram :: AST -> Program
astToProgram ast = Program {
    types = types
  , constants = map (\(n, v) -> (n, transSimpleTerm types v)) $ AST.constants
     ast
  , fnDeclarations = map (\(n, f) -> (n, transFunction types f)) $ AST.
     fnDeclarations ast
  , imports = map (transImport types) $ AST.imports ast
  }
  where types = transTypes $ AST.customTypes ast

transFunction :: TypeMap -> AST.Function -> Function
transFunction types astFunc = Function {
    inputType = transInlineType types $ AST.inputType astFunc
  , outputType = transInlineType types $ AST.outputType astFunc
  , arg = AST.arg astFunc
  , body = transExpressions types $ AST.body astFunc
  , helper = AST.helper astFunc
  }

transImport :: TypeMap -> AST.Import -> Import
transImport types (AST.Import t) = parseImportArg $ transSimpleTerm types t

transExpressions :: TypeMap -> AST.ExpressionBlock -> ExpressionBlock
transExpressions types = evalState (whileJust (transExpression types) return)

takeNext :: State [a] (Maybe a)
takeNext = do
  ls <- get
  case ls of
    [] -> return Nothing
    (x:xs) -> do
      put xs
      return (Just x)

takeIndented :: TypeMap -> Int -> State AST.ExpressionBlock ExpressionBlock
takeIndented types n = transExpressions types <$> takeIndented'
  where takeIndented' = do
```

```haskell
          next <- takeNext
          case next of
            Nothing -> return []
            Just expr@(n', _) ->
              if n' >= n
              then do
                rest <- takeIndented'
                return $ expr : rest
              else do
                modify (expr:)
                return $ []

transExpression :: TypeMap -> State AST.ExpressionBlock (Maybe Expression)
transExpression types = do
  next <- takeNext
  case next of
    Nothing -> return Nothing
    Just (n, AST.Assignment v t) -> (Just . Assignment v) <$> transTerm types
        (n, t)
    Just (n, AST.Unassigned t) -> (Just . Unassigned) <$> transTerm types (n,
        t)
    Just (n, AST.Assert t) -> (Just . Assert) <$> transTerm types (n, t)

transTerm :: TypeMap -> (Int, AST.Term) -> State AST.ExpressionBlock Term
transTerm types (n, AST.If t) = do
  thenBlock <- takeIndented types (n + indentIncrement)
  next <- takeNext
  elseBlock <- case next of
    Nothing -> return []
    Just (elseInc, AST.Unassigned AST.Else) ->
      if elseInc /= n
      then error $ "Found an else expression with indent " ++ show elseInc ++
          ", expected indent " ++ show n
      else takeIndented types (n + indentIncrement)
    Just x -> do
      modify (x:)
      return []
  return $ IfThenElse (transSimpleTerm types t) thenBlock elseBlock
transTerm types (n, AST.ForeachIn v t) = do
  doBlock <- takeIndented types (n + indentIncrement)
  case doBlock of
    [] -> error $ "Empty body of a ForeachIn block"
    exprs -> return $ ForeachInDo v (transSimpleTerm types t) exprs
transTerm types (_, t) = return $ transSimpleTerm types t

transSimpleTerm :: TypeMap -> AST.Term -> Term
transSimpleTerm _ (AST.Variable v) = Variable v
transSimpleTerm types (AST.Accessor a b) = Accessor (transSimpleTerm types a)
    (transSimpleTerm types b)
transSimpleTerm types (AST.FunctionCall n a) = FunctionCall n (transSimpleTerm
    types a)
transSimpleTerm types (AST.OperatorTerm n a b) = OperatorTerm
  n (transSimpleTerm types a) (transSimpleTerm types b)
transSimpleTerm types (AST.Literal v) = Literal (transPrim types v)
transSimpleTerm types (AST.TypeCheck v t) = TypeCheck (transSimpleTerm types v
    ) (transInlineType types t)
transSimpleTerm types (AST.TypeAssert v t) = TypeAssert (transSimpleTerm types
     v) (transInlineType types t)
transSimpleTerm types (AST.IfThenElse p a b) = IfThenElse
  (transSimpleTerm types p) [Unassigned $ transSimpleTerm types a] [Unassigned
      $ transSimpleTerm types b]
transSimpleTerm types t@(AST.If _) = error $ "unexpected If term: " ++ show t
transSimpleTerm types (AST.Else) = error "unexpected Else term"
transSimpleTerm types t@(AST.ForeachIn _ _) = error $ "unexpected ForeachIn
    term: " ++ show t
transSimpleTerm types (AST.Do) = error "unexpected Do term"

transPrim :: TypeMap -> AST.PrimValue -> PrimValue
transPrim _ (AST.StrVal s) = (StrVal s)
transPrim _ (AST.NumVal s) = (NumVal s)
transPrim types (AST.ArrVal s) = (ArrVal (map (transSimpleTerm types) s))
transPrim types (AST.ObjVal s) = (ObjVal (fmap (transSimpleTerm types) s))
transPrim _ AST.NullVal = NullVal
transPrim _ AST.TrueVal = TrueVal
transPrim _ AST.FalseVal = FalseVal

data Program = Program {
    types :: TypeMap
  , constants :: [(ValName, Term)]
  , fnDeclarations :: [(FnName, Function)]
  , imports :: [Import]
  } deriving (Show, Generic, Out)

data Function = Function {
    inputType :: Type
  , outputType :: Type
  , arg :: ValName
  , body :: ExpressionBlock
  , helper :: Bool
  } deriving (Show, Generic, Out)

data Import = Import URL Key Secret Header [Endpoint]
            deriving (Show, Generic, Out)

type ExpressionBlock = [Expression]

data Expression = Assignment ValName Term
                | Unassigned Term
                | Assert Term
                deriving (Show, Generic, Out)

data Term = Variable ValName
          | Accessor Term Term
          | FunctionCall FnName Term
          | OperatorTerm Operator Term Term
          | Literal PrimValue
          | IfThenElse Term ExpressionBlock ExpressionBlock
          | ForeachInDo ValName Term ExpressionBlock
          | TypeCheck Term Type
          | TypeAssert Term Type
          deriving (Show, Generic, Out)

data PrimValue = StrVal String
               | NumVal Double
               | ArrVal [Term]
               | ObjVal (Map String Term)
               | NullVal
               | TrueVal
               | FalseVal
               deriving (Show, Generic, Out)


data Method = Post | Get
            deriving (Eq, Show, Generic, Out)
type EndpointFnName = String
type EndpointEndpoint = String
type URL = String
type Key = String
type Secret = String
type Header = String
data Endpoint = Endpoint EndpointFnName EndpointEndpoint Method
              deriving (Show, Generic, Out)

parseImportArg :: Term -> Import
parseImportArg (Literal (ObjVal obj)) = Import url key secret header
    endpoints
  where getVal objName obj key = fromMaybe (error $ key ++ " missing from " ++
        objName) (Map.lookup key obj)
        getImpVal = getVal "import statement" obj
        url = case getImpVal "url" of
          (Literal (StrVal url)) -> url
          _ -> error "url key in import statement should be a string value"
        key = case getImpVal "key" of
          (Literal (StrVal key)) -> key
          _ -> error "auth key in import statement missing. If no key is
             required, use emtpy string"
        secret = case getImpVal "secret" of
          (Literal (StrVal secret)) -> secret
          _ -> error "auth secret in import statement missing. If no secret is
              required, use emtpy string"
        header = case getImpVal "header" of
          (Literal (StrVal header)) -> header
          _ -> error "header in import statement missing."
        endpoints = case getImpVal "endpoints" of
          (Literal (ArrVal endpointTerms)) -> flip map endpointTerms $ \t ->
             case t of
            (Literal (ObjVal endpointObj)) ->
              let getEndpVal = getVal "endpoint statement" endpointObj
                  name = case getEndpVal "fnName" of
                    (Literal (StrVal name)) -> name
                    _ -> error "endpoint's fnName should be a string"
                  endpoint = case getEndpVal "endpoint" of
```

```
                    (Literal (StrVal endpoint)) -> endpoint
                    _ -> error "endpoint should be a string"
                  method = case getEndpVal "is_post" of
                    (Literal TrueVal) -> Post
                    (Literal FalseVal) -> Get
                    _ -> error "endpoint is_post should be true/false"
              in Endpoint name endpoint method

            _ -> error "endpoint values in import statement should be object
              literals"
          _ -> error "endpoint key in import statement should be an array
            value"
parseImportArg _ = error "Import called with non-primitive object argument"
```

9.5 Semantics.hs

```
{-# LANGUAGE RecordWildCards, StrictData, Strict #-}
module Semantics where

import Prelude hiding (LT, GT, EQ)
import Control.Monad.State
import qualified Data.Map as Map
import Data.Map (Map)
import Data.List
import System.IO.Unsafe
import System.IO
import System.Exit

import Program

data Context = Context {
    signatures :: Map String (Type', Type')
  , typeMap :: Map String Type'
  , opSignatures :: Map Operator (Type', Type', Type')
  }

type Type' = Maybe PrimType

error' s = unsafePerformIO $ do
  hPutStrLn stderr s
  exitFailure

match :: Type' -> Type' -> Bool
match a b = case (==) <$> a <*> b of
  Nothing -> True
  Just True -> True
  Just False -> False

noMatch a b = not $ match a b

lastOr :: a -> [a] -> a
lastOr x xs = if null xs then x else last xs

```

```
checkProgram :: Program -> Bool
checkProgram (Program {..}) = and $
                              (map (checkFunction context . snd)
                                 fnDeclarations)
                              ++ (map (checkType context . snd) . Map.toList $
                                  types)
  where signatures = Map.fromList $ map (\(fnName, (Function {inputType = inT,
      outputType = outT})) ->
                                            (fnName, (Just $ baseType inT,
                                               Just $ baseType outT)))
                                    fnDeclarations
        importedSignatures = mconcat . map importSignatures $ imports
        allSignatures = signatures `Map.union` builtinSignatures `Map.union`
           importedSignatures
        context = Context allSignatures (fmap (Just . baseType) types)
           operatorSignatures

importSignatures :: Import -> Map String (Type', Type')
importSignatures (Import url key secret header endpoints) = Map.fromList .
   flip map endpoints $ \(Endpoint fnname _ _) ->
  (fnname, (Nothing, Nothing))

checkType :: Context -> Type -> Bool
checkType context (Type {..}) = maybe True (\t ->
                                              error' $ "Expect type predicates
                                                  to be boolean,"
                                                    ++ " but found one with
                                                       type " ++ show t)
                                . find (noMatch (Just BoolType))
                                . map last
                                . filter (not . null)
                                . map (\(var, block) ->
                                          evalState (evaluate block) (
                                             initialTypes var))
                                $ predicates
  where evaluate body = mapM (checkExpression context) body
        initialTypes var = Map.fromList [(var, Just baseType)]

checkFunction :: Context -> Function -> Bool
checkFunction context@(Context {..}) (Function {..}) =
  if Just (baseType outputType) `match` foundType
  then True
  else error $ "Function's return type does not match the final value (
     expecting " ++ show (baseType outputType) ++ " but found " ++ show (
     evalState evaluate initialTypes) ++ ")"
  where initialTypes = Map.fromList [(arg, Just $ baseType inputType)]
        evaluate = (lastOr Nothing) <$> mapM (checkExpression context) body
        foundType = evalState evaluate initialTypes

checkExpression :: Context -> Expression -> State (Map String Type') Type'
checkExpression context@(Context {..}) (Assignment var term) = do
  res <- checkTerm context term
  modify $ Map.insert var res
  return res
```

```
checkExpression context@(Context {..}) (Unassigned term) = checkTerm context
    term
checkExpression context@(Context {..}) (Assert term) = do
  termType <- checkTerm context term
  if termType `noMatch` Just BoolType
    then error' $ "Can't call an assert on a non-boolean operand of type " ++
        show termType
    else return (Just BoolType)

checkScopedBlock :: Context -> ExpressionBlock -> State (Map String Type')
    Type'
checkScopedBlock context expressions = do
  namespace <- get
  res <- lastOr Nothing <$> mapM (checkExpression context) expressions
  return res

checkTerm :: Context -> Term -> State (Map String Type') Type'
checkTerm context@(Context {..}) t = do
  m <- get
  case t of
    Variable var ->
      case var `Map.lookup` m of
        Nothing -> error' $ "Called an unassigned variable " ++ var
        Just varType -> return varType
    Accessor val index -> do
      valType <- checkTerm context val
      indexType <- checkTerm context index
      case valType of
        Just ArrType ->
          if noMatch indexType (Just NumType)
          then error' $ "Attempting to index an array with non-Num value: " ++
               show index
          else return Nothing
        Just ObjType ->
          if noMatch indexType (Just StrType)
          then error' $ "Attempting to index an object with non-Str value: "
              ++ show index
          else return Nothing
        Just _ -> error' $ "Attempting to index a non-collection value: " ++
            show val
        Nothing -> return Nothing
    FunctionCall fnName arg -> do
      case fnName `Map.lookup` signatures of
        Nothing -> error' $ "Attempting to use undefined function " ++ fnName
        Just (inType, outType) -> do
          argType <- checkTerm context arg
          if argType `noMatch` inType
            then error' $ "Attempting to call function " ++ show fnName ++
                 " with argument of base type " ++ show argType ++
                 ", but function expects a base type " ++ show (inType)
            else return $ outType
    OperatorTerm op t1 t2 -> do
      case op `Map.lookup` opSignatures of
        Nothing -> error' $ "Attempting to use undefined operator " ++ show op
```

```haskell
        Just (arg1Type, arg2Type, retType) -> do
          t1Type <- checkTerm context t1
          if arg1Type `noMatch` t1Type
            then error' $ "Left argument to operator " ++ show op ++ " should be of type "
                 ++ show arg1Type ++ ", but it was of type " ++ show t1Type
            else do t2Type <- checkTerm context t2
                    if arg2Type `noMatch` t2Type
                      then error' $ "Right argument to operator " ++ show op ++ " should be of type "
                           ++ show arg2Type ++ ", but it was of type " ++ show t2Type
                      else return retType
    Literal prim -> return $ checkLiteral prim
    IfThenElse pred block1 block2 -> do
      predType <- checkTerm context pred
      if predType `noMatch` Just BoolType && predType `noMatch` Just NumType
        then error' $ "Can't use a non-bool, non-number value as a predicate in an if statement: " ++ show predType
        else do block1Type <- checkScopedBlock context block1
                block2Type <- checkScopedBlock context block2
                if block1Type `noMatch` block2Type
                  then error' $ "The branches of an if statement have two different return types: "
                       ++ show block1Type ++ " vs " ++ show block2Type
                  else return block1Type
    ForeachInDo var arr block -> do
      arrType <- checkTerm context arr
      if arrType `noMatch` Just ArrType
        then error' $ "Can't loop over the non-array value with type " ++ show arrType
        else do withoutVar <- get
                let withVar = Map.insert var Nothing withoutVar
                put withVar
                res <- checkScopedBlock context block
                put withoutVar
                return (Just ArrType)
    TypeCheck term tp -> do
      termType <- checkTerm context term
      if not (checkType context tp)
        then error' "Error in type in type check"
        else if termType `noMatch` Just (baseType tp)
             then error' $ "Typecheck will never be true! Real type is " ++ show termType
             else return (Just BoolType)
    TypeAssert term tp -> do
      termType <- checkTerm context term
      if not (checkType context tp)
        then error' "Error in type in type check"
        else if termType `noMatch` Just (baseType tp)
             then error' $ "TypeAssert will never be true! Real type is " ++ show termType
             else return termType

```

```haskell
checkLiteral :: PrimValue -> Type'
checkLiteral (StrVal _) = Just StrType
checkLiteral (NumVal _) = Just NumType
checkLiteral (ArrVal _) = Just ArrType
checkLiteral (ObjVal _) = Just ObjType
checkLiteral NullVal = Just NullType
checkLiteral TrueVal = Just BoolType
checkLiteral FalseVal = Just BoolType

builtinSignatures = Map.fromList [ ("log", (Nothing, Nothing))
                                 , ("isString", (Nothing, Just BoolType))
                                 , ("isArr", (Nothing, Just BoolType))
                                 , ("isNum", (Nothing, Just BoolType))
                                 , ("isObj", (Nothing, Just BoolType))
                                 , ("isBool", (Nothing, Just BoolType))
                                 , ("jn", (Just StrType, Nothing))
                                 , ("addToObj", (Just ArrType, Just ObjType))
                                 , ("push", (Just ArrType, Nothing))
                                 , ("update", (Just ArrType, Just ArrType))
                                 , ("clientPost", (Nothing, Nothing))
                                 , ("clientGet", (Nothing, Nothing))
                                 , ("toNum", (Just StrType, Just NumType))
                                 , ("get", (Nothing, Nothing))
                                 , ("geta", (Just ArrType, Nothing))
                                 , ("cat", (Just ArrType, Just StrType))
                                 , ("equals", (Nothing, Just BoolType))
                                 ]

operatorSignatures = fmap (\(a, b, c) -> (Just a, Just b, Just c))
    operatorSignatures'
operatorSignatures' = Map.fromList [ (Plus, (NumType, NumType, NumType))
                                   , (Minus, (NumType, NumType, NumType))
                                   , (Multiply, (NumType, NumType, NumType))
                                   , (Divide, (NumType, NumType, NumType))
                                   , (Modulus, (NumType, NumType, NumType))
                                   , (EQ, (NumType, NumType, BoolType))
                                   , (LEQ, (NumType, NumType, BoolType))
                                   , (GEQ, (NumType, NumType, BoolType))
                                   , (GT, (NumType, NumType, BoolType))
                                   , (LT, (NumType, NumType, BoolType))
                                   , (And, (BoolType, BoolType, BoolType))
                                   , (Or, (BoolType, BoolType, BoolType))
                                   ]
```

9.6 LLVM.hs

```haskell

{-# LANGUAGE RecordWildCards #-}
module LLVM where

import Prelude hiding (EQ, LEQ, GEQ, GT, LT)
import Program
import qualified LLVM.AST as AST
import qualified LLVM.AST.AddrSpace as AST
import qualified LLVM.Module as Module
import qualified LLVM.Internal.Context as Context
import qualified LLVM.AST.Constant as AST hiding (GetElementPtr, FCmp, ICmp,
    PtrToInt, FPToUI, ZExt, And, Or)
import qualified LLVM.AST.FloatingPointPredicate as Floatypoo
import qualified LLVM.AST.IntegerPredicate as Intypoo
import qualified LLVM.AST.Float as Fl
import Codegen
import Control.Monad
import Control.Monad.State
import Data.Maybe
import Data.Word
import Data.String
import Data.Char
import qualified Data.Map as Map

writeModule :: FilePath -> AST.Module -> IO ()
writeModule fp m =
  Context.withContext (\context -> Module.withModuleFromAST context m (\m' ->
      write context m'))
  where write context m' = Module.writeLLVMAssemblyToFile (Module.File fp) m'

buildModule :: Program -> AST.Module
buildModule p = runLLVM moduleHeader (buildLLVM p)

llvmI8 = AST.IntegerType 8
llvmI32 = AST.IntegerType 32
llvmI32Pointer = (AST.PointerType llvmI32 (AST.AddrSpace 0))
llvmI32PointerPointer = (AST.PointerType llvmI32Pointer (AST.AddrSpace 0))
llvmStringPointer = (AST.PointerType llvmI8 (AST.AddrSpace 0))
llvmPointerStringfPointer = (AST.PointerType llvmStringPointer (AST.AddrSpace
    0))
llvmDouble = AST.FloatingPointType AST.DoubleFP

moduleHeader = runLLVM (emptyModule "WebLang") $ do
  external llvmI32Pointer "json_from_string" [(llvmI32Pointer, AST.Name (
      fromString "s"))];
  external llvmI32Pointer "json_object" [(llvmI32PointerPointer, AST.Name(
      fromString "s"))
                                      , (llvmI32, (fromString "s"))];
  external llvmI32Pointer "is_json_object" [(llvmI32Pointer, AST.Name (
      fromString "s"))];
  external llvmI32Pointer "add_to_json_object" [(llvmI32Pointer, AST.Name (
      fromString "s"))
                                      , (llvmI32Pointer, AST.Name (fromString "
                                          s"))
                                      , (llvmI32Pointer, AST.Name (fromString "
                                          s"))];
  external llvmI32 "exit" [(llvmI32, AST.Name (fromString "s"))];
  external llvmI32 "puts" [(llvmStringPointer, AST.Name (fromString "s"))];
  external llvmI32 "floor" [(llvmDouble, AST.Name (fromString "s"))];
  external llvmI32 "round" [(llvmDouble, AST.Name (fromString "s"))];
  external llvmI32 "ceil" [(llvmDouble, AST.Name (fromString "s"))];
external llvmI32 "strcmp" [(llvmStringPointer, AST.Name (fromString "s")),
                                          (llvmStringPointer, AST.Name (fromString "s"))];
external llvmI32Pointer "jgets" [ (llvmI32Pointer, AST.Name (fromString "s"))
                                      , (llvmI32Pointer, AST.Name (fromString "s"))];
external llvmI32 "test" [(llvmStringPointer, AST.Name (fromString "s"))];
external llvmI32Pointer "post" [(llvmStringPointer, AST.Name (fromString "s")),
                                (llvmI32Pointer, AST.Name (fromString "s")),
                                (llvmStringPointer, AST.Name (fromString "s")),
                                (llvmStringPointer, AST.Name (fromString "s")),
                                (llvmStringPointer, AST.Name (fromString "s"))];
external llvmI32Pointer "exposed_post" [(llvmI32Pointer, AST.Name (fromString "s"))];
external llvmI32Pointer "get"[(llvmStringPointer, AST.Name (fromString "s")),
                                (llvmI32Pointer, AST.Name (fromString "s")),
                                (llvmStringPointer, AST.Name (fromString "s")),
                                (llvmStringPointer, AST.Name (fromString "s")),
                                (llvmStringPointer, AST.Name (fromString "s"))];
external llvmI32Pointer "json_string" [(llvmStringPointer, AST.Name (fromString "s"))];
external llvmI32Pointer "is_json_string" [ (llvmI32Pointer, AST.Name (fromString "s"))];
external llvmStringPointer "tostring" [(llvmI32Pointer, AST.Name (fromString "s"))];
external llvmI32Pointer "is_string_equal" [(llvmI32Pointer, AST.Name (fromString "s"))
                                      , (llvmI32Pointer, AST.Name (fromString "s"))];
external llvmI32Pointer "concat" [(llvmI32Pointer, AST.Name (fromString "s"))
                                      , (llvmI32Pointer, AST.Name (fromString "s"))];
external llvmI32Pointer "json_double" [(llvmDouble, AST.Name (fromString "s"))];
external llvmI32Pointer "to_json_double" [(llvmI32Pointer, AST.Name (fromString "s"))];
external llvmI32Pointer "is_json_double" [(llvmI32Pointer, AST.Name (fromString "s"))];
external llvmDouble "get_json_double" [(llvmI32Pointer, AST.Name (fromString "s"))];
external llvmI32Pointer "json_array" [ (llvmI32PointerPointer, AST.Name (fromString "s"))
                                        , (llvmI32, (fromString "s"))];
external llvmI32Pointer "is_json_array" [(llvmI32Pointer, AST.Name (
    fromString "s"))];
  external llvmI32Pointer "get_json_from_array" [ (llvmI32Pointer, AST.Name (fromString "s"))
                                                , (llvmI32, (fromString "s"))];
  external llvmI32Pointer "push_to_json_array" [ (llvmI32Pointer, AST.Name (fromString "s"))
                                               , (llvmI32Pointer, (fromString "s"))];
  external llvmI32Pointer "replace_json_array_element" [(llvmI32Pointer, AST.Name (fromString "s"))
                                                       , (llvmI32Pointer, (fromString "s"))
                                                       , (llvmI32Pointer, (fromString "s"))];
  external llvmI32Pointer "create_arr_iter" [(llvmI32Pointer, AST.Name (fromString "s"))];
  external llvmI32Pointer "arr_next_elem" [ (llvmI32Pointer, AST.Name (fromString "s"))
                                          , (llvmI32Pointer, AST.Name (fromString "s"))];
  external llvmI32Pointer "json_bool" [(llvmI32, AST.Name (fromString"s"))];
  external llvmI32Pointer "is_json_bool" [(llvmI32Pointer, AST.Name (fromString "s"))];
  external llvmI32Pointer "parse_function_arg" [(llvmI32Pointer, AST.Name (fromString "s"))];

externs = Map.fromList [
      ("log", "puts"),
      ("jn", "json_from_string"),
      ("isObj", "is_json_object"),
      ("clientPost", "exposed_post"),
      ("clientGet", "get"),
      ("jnum", "json_double"),
      ("toNum", "to_json_double"),
      ("getdoub", "get_json_double"),
      ("tostring", "tostring"),
      ("getfst", "create_arr_iter"),
      ("getnext", "arr_next_elem"),
      ("scmp", "strcmp"),
      ("floor", "floor"),
      ("isString", "is_json_string"),
      ("isNum", "is_json_double"),
      ("isArr", "is_json_array"),
      ("jbool", "json_bool"),
      ("isBool", "is_json_bool")
  ]

extern2args = Map.fromList [
      ("get", "jgets"),
      ("geta", "get_json_from_array"),
      ("push", "push_to_json_array"),
      ("cat", "concat"),
      ("equals", "is_string_equal")
  ]


extern3args = Map.fromList [
      ("addToObj", "add_to_json_object"),
      ("update", "replace_json_array_element")
  ]

boolOperators = Map.fromList [
    (Or, AST.Or)
  , (And, AST.And)
  ]

eqOperators = Map.fromList [
    (EQ, fcmp Floatypoo.OEQ)
  , (LEQ, fcmp Floatypoo.OLE)
  , (GEQ, fcmp Floatypoo.OGE)
  , (LT, fcmp Floatypoo.OLT)
  , (GT, fcmp Floatypoo.OGT)
  ]

numOperators = Map.fromList [
    (Plus, fadd)
  , (Minus, fsub)
  , (Multiply, fmul)
  , (Divide, fdiv)
  , (Modulus, fmod)
  ]

opFns = Map.empty

buildLLVM :: Program -> LLVM ()
buildLLVM p = do
  mapM_ importLLVM (imports p)
  mapM_ constantLLVM (constants p)
  let fns = fnDeclarations p
  mapM_ functionLLVM fns
  functionLLVMMain fns

importLLVM :: Import -> LLVM [String]
importLLVM (Import url key secret header endpoints) = mapM (endpointFnLLVM url
    key secret header) endpoints

endpointFnLLVM :: String -> String -> String -> String -> Endpoint -> LLVM
   String
endpointFnLLVM url key secret header (Endpoint fnname endpoint method) =
   define llvmRetType fnname fnargs llvmBody >> return fnname
  where arg = "arg"
        fnargs = toSig arg
        llvmRetType = llvmI32Pointer
        llvmBody = createBlocks . execCodegen $ do
          entry <- addBlock entryBlockName
          setBlock entry
          let argptr = local (AST.Name (fromString arg))
          let path = url ++ "/" ++ endpoint
          let binding = if method == Post then "post" else "get"

          url <- rawStringLLVM path
          key <- rawStringLLVM key
          secret <- rawStringLLVM secret
          header <- rawStringLLVM header
          res <- call (externf (AST.Name (fromString binding))) [url, argptr,
             key, secret, header]
          ret (Just res)

constantLLVM :: (ValName, Term) -> LLVM ()
constantLLVM (name, term) = do
  -- looks like constants with GlobalVariable won't work, since we need to
     execute code to use JSON interop
  -- maybe we could declare globals as initially null, then generate code to
     change them
  error "constants unimplemented"

toSig :: String -> [(AST.Type, AST.Name)]
toSig x = [(llvmI32Pointer, AST.Name (fromString x))]

mainSig :: [(AST.Type, AST.Name)]
mainSig = [ (llvmI32Pointer, AST.Name (fromString "argc"))
          , (llvmPointerStringfPointer, AST.Name (fromString "argv"))]

functionLLVMMain :: [(FnName, Function)] -> LLVM ()
functionLLVMMain fns = do
  define llvmRetType "main" mainSig llvmBody
  where llvmRetType = llvmI32
        llvmBody = createBlocks $ execCodegen $ do
          entry <- addBlock entryBlockName
          setBlock entry
          let fnNames = map fst . filter (not . helper . snd) $ fns
          argv1 <- argvAt 1
          argv2 <- argvAt 2
          mapM_ (\f -> createEndpointCheck f argv1 argv2) fnNames
          ret $ Just (cons $ AST.Int 32 0)


createEndpointCheck :: String -> AST.Operand -> AST.Operand -> Codegen AST.
   Name
createEndpointCheck fnName cmdRef arg = do
  compare <- rawStringLLVM fnName
  compStrRes <- llvmCallExt2 cmdRef compare "strcmp"
  let equal = AST.ICmp Intypoo.EQ compStrRes (cons $ AST.Int 32 0) []
  refEq <- instr $ equal
  iff <- addBlock fnName
  continue <- addBlock "continue"
  cbr refEq iff continue

  setBlock iff
  strargs <- functionCallLLVM "json_string" arg
  args <- functionCallLLVM "parse_function_arg" strargs
  functionCallLLVM fnName args
  br continue
  iff <- getBlock

  setBlock continue

argvAt :: Integer -> Codegen AST.Operand
argvAt idx = do
  let argv = local (AST.Name (fromString "argv"))
  let ptr = AST.GetElementPtr True argv [cons $ AST.Int 32 idx] []
  ref <- instr $ ptr
  let load = AST.Load False ref Nothing 1 []
  op <- instr $ load
  return op

functionLLVM :: (FnName, Function) -> LLVM ()
functionLLVM (name, (Function {..})) = define llvmRetType name fnargs llvmBody
  where llvmRetType = llvmI32Pointer
        fnargs = toSig arg
        llvmBody = createBlocks $ execCodegen $ do
          entry <- addBlock entryBlockName
          setBlock entry
          let argptr = local (AST.Name (fromString arg))

          typeAssertionLLVM ("Pre-condition not met in function " ++ name)
             inputType argptr

          l <- alloca llvmI32Pointer
          store l argptr
          assign arg l
          res <- expressionBlockLLVM body

          typeAssertionLLVM ("Post-condition not met in function " ++ name)
             outputType res

          ret (Just res)

typeCheckLLVM :: Type -> AST.Operand -> Codegen [AST.Operand]
typeCheckLLVM (Type {..}) val = forM predicates $ \(var, predBlock) -> do

  withoutVar <- symtab <$> get
  l <- alloca llvmI32Pointer
  store l val
  assign var l
  res <- expressionBlockLLVM predBlock
  modify $ \state -> state {symtab = withoutVar}

  return res

typeAssertionLLVM :: String -> Type -> AST.Operand -> Codegen ()
typeAssertionLLVM msg t val = typeCheckLLVM t val >>= mapM_ (assertionLLVM msg
   )

assertionLLVM :: String -> AST.Operand -> Codegen ()
assertionLLVM message res = do
  failureBlock <- addBlock "type-assertion-failed"
  exitBlock <- addBlock "iexit"

  boolasdoub <- functionCallLLVM "getdoub" res
  branchval <- fcmp Floatypoo.ONE (cons $ AST.Float (Fl.Double 0.0))
      boolasdoub
  cbr branchval exitBlock failureBlock

  setBlock failureBlock
  messageString <- rawStringLLVM message
  call (externf (AST.Name (fromString "puts"))) [messageString]
  call (externf (AST.Name (fromString "exit"))) [cons $ AST.Int 32 1]
  br exitBlock
  ielse <- getBlock

  setBlock exitBlock
  return ()

expressionBlockLLVM :: ExpressionBlock -> Codegen AST.Operand
expressionBlockLLVM exprs = last <$> mapM expressionLLVM exprs

expressionLLVM :: Expression -> Codegen AST.Operand
expressionLLVM (Unassigned term) = termLLVM term
expressionLLVM (Assignment name term) = do
  maybeVal <- getvar name
  ptr <- termLLVM term
  l <- case maybeVal of
    Nothing -> do
      l <- alloca llvmI32Pointer
      assign name l
      return l
    Just val -> return val
  store l ptr
  return ptr
expressionLLVM (Assert term) = do
  ptr <- termLLVM term
  assertionLLVM "Assertion failed!" ptr
  return ptr

scopedBlockLLVM :: ExpressionBlock -> Codegen AST.Operand
scopedBlockLLVM exprs = do
  symTable <- symtab <$> get
  res <- expressionBlockLLVM exprs
  modify $ \state -> state {symtab = symTable}
  return res

termLLVM :: Term -> Codegen AST.Operand
termLLVM (FunctionCall fname arg) = do
  op <- termLLVM arg
  functionCallLLVM fname op
termLLVM (Accessor tTerm indexTerm) = do
  t <- termLLVM tTerm
  index <- termLLVM indexTerm
  element <- call
                (externf (AST.Name (fromString "jgets")))
                [t, index]

  return element
termLLVM (OperatorTerm opp t1 t2) = do
  val1 <- termLLVM t1
  val2 <- termLLVM t2
  double1 <- functionCallLLVM "getdoub" val1
  double2 <- functionCallLLVM "getdoub" val2
  case Map.lookup opp numOperators of
    Just ap -> do
      result <- ap double1 double2
      functionCallLLVM "jnum" result
    Nothing -> case Map.lookup opp eqOperators of
      Just ap -> do
        result <- ap double1 double2
        int32 <- instr $ AST.ZExt result llvmI32 []
        functionCallLLVM "json_bool" int32
      Nothing -> case Map.lookup opp boolOperators of
        Just oper -> do
          int1 <- call (externf (AST.Name (fromString "round"))) [double1]
          int2 <- call (externf (AST.Name (fromString "round"))) [double2]
          res <- instr $ oper int1 int2 []
          call (externf (AST.Name (fromString "json_bool"))) [res]
        Nothing -> error $ "unimplemented operator " ++ show opp

termLLVM (IfThenElse bool tr fal) = do
  iff <- addBlock "iff"
  ielse <- addBlock "ielse"
  iexit <- addBlock "iexit"
  bool <- termLLVM bool
  boolasdoub <- functionCallLLVM "getdoub" bool
  branchval <- fcmp Floatypoo.ONE (cons $ AST.Float (Fl.Double 0.0))
      boolasdoub
  cbr branchval iff ielse

  setBlock iff
  tval <- scopedBlockLLVM tr
  br iexit
  iff <- getBlock

  setBlock ielse
  fval <- scopedBlockLLVM fal
  br iexit
  ielse <- getBlock

  setBlock iexit
  phi llvmI32Pointer [(tval, iff), (fval, ielse)]

termLLVM (ForeachInDo var container body) = do
  loop <- addBlock "loop"
  exit <- addBlock "exit"

  l <- alloca llvmI32Pointer
  pcontainer <- termLLVM container
  firstel <- functionCallLLVM "getfst" pcontainer
  store l firstel
  assign var l
  ptrAsInt <- instr $ AST.PtrToInt firstel llvmI32 []
  test <- icmp Intypoo.NE (cons $ AST.Int 32 0) ptrAsInt
  cbr test loop exit

  setBlock loop
  scopedBlockLLVM body
  curr <- load l
  next <- llvmCallExt2 curr pcontainer "arr_next_elem"
  store l next
  ptrAsInt <- instr $ AST.PtrToInt next llvmI32 []
  test <- icmp Intypoo.NE (cons $ AST.Int 32 0) ptrAsInt
  cbr test loop exit

  setBlock exit
  return pcontainer

termLLVM (Literal prim) = primLLVM prim
termLLVM (Variable var) = do
  maybeVal <- getvar var
  case maybeVal of
    Nothing -> error $ "Local variable not in scope: " ++ show var
    Just val -> load val
termLLVM (TypeAssert term t) = do
  val <- termLLVM term
  typeAssertionLLVM "Type assertion failed!" t val
  return val
termLLVM (TypeCheck term t) = do
  val <- termLLVM term
  (first:rest) <- typeCheckLLVM t val
  foldM (boolOp AST.And) first rest

boolOp :: (AST.Operand -> AST.Operand -> AST.InstructionMetadata -> AST.
    Instruction)
       -> AST.Operand
       -> AST.Operand
       -> Codegen AST.Operand
boolOp oper val1 val2 = do
  double1 <- functionCallLLVM "getdoub" val1
  double2 <- functionCallLLVM "getdoub" val2
  int1 <- call (externf (AST.Name (fromString "round"))) [double1]
  int2 <- call (externf (AST.Name (fromString "round"))) [double2]
  res <- instr $ oper int1 int2 []
  call (externf (AST.Name (fromString "json_bool"))) [res]

primLLVM :: PrimValue -> Codegen AST.Operand
primLLVM (ArrVal arr) = do
  elemPtrs <- mapM termLLVM arr
  ptrArray <- buildPtrArray elemPtrs
  llvmCallJsonArr ptrArray (length elemPtrs)
```

```
primLLVM (ObjVal obj) = do
  elemPtrs <- mapM termLLVM obj
  ptrArray <- buildObjPtrArray (Map.toList elemPtrs)
  llvmCallJsonObj ptrArray (length elemPtrs)
primLLVM (NumVal num) = functionCallLLVM "jnum" (cons (AST.Float (Fl.Double
    num)))
primLLVM (StrVal s) = stringLLVM s
primLLVM (NullVal) = nullLLVM
primLLVM (TrueVal) = trueLLVM
primLLVM (FalseVal) = falseLLVM

nullLLVM :: Codegen AST.Operand
nullLLVM = error "need to build a null builder"

trueLLVM :: Codegen AST.Operand
trueLLVM = do
  functionCallLLVM "json_bool" (cons $AST.Int 32 (fromIntegral 1))

falseLLVM :: Codegen AST.Operand
falseLLVM = do
  functionCallLLVM "json_bool" (cons $AST.Int 32 (fromIntegral 0))

llvmCallJsonArr :: AST.Operand -> Int -> Codegen AST.Operand
llvmCallJsonArr elemPtrArray n = call (externf (AST.Name (fromString "
    json_array")))
  [elemPtrArray, (cons $ AST.Int 32 (fromIntegral n))]

llvmCallJsonObj :: AST.Operand -> Int -> Codegen AST.Operand
llvmCallJsonObj elemPtrArray n = call (externf (AST.Name (fromString "
    json_object")))
  [elemPtrArray, (cons $ AST.Int 32 (fromIntegral n))]

functionCallLLVM :: String -> AST.Operand -> Codegen AST.Operand
functionCallLLVM fn arg = do
  case Map.lookup fn externs of
    Just fn2 -> do
      llvmCallExt arg fn2
    Nothing -> case Map.lookup fn extern2args of
      Just fn3 -> do
        llvmCallExt2args arg fn3
      Nothing -> case Map.lookup fn extern3args of
          Just fn4 -> do
            llvmCallExt3args arg fn4
          Nothing -> llvmCallFunc fn arg

llvmCallExt :: AST.Operand -> String -> Codegen AST.Operand
llvmCallExt op func =
  if func == "puts"
  then do
    st <- functionCallLLVM "tostring" op
    call (externf (AST.Name (fromString func))) [st]
    return op
  else if func == "get" || func =="post"
  then do
```

```
    res <- call (externf (AST.Name (fromString func))) [op]
    ret <- functionCallLLVM "jn" res
    return ret
  else call (externf (AST.Name (fromString func))) [op]

llvmCallExt2 :: AST.Operand -> AST.Operand -> String -> Codegen AST.Operand
llvmCallExt2 op op2 func = call (externf (AST.Name (fromString func))) [op, op2]

llvmCallExt2args :: AST.Operand -> String -> Codegen AST.Operand
llvmCallExt2args op func = do
  op1 <- call (externf (AST.Name (fromString "get_json_from_array"))) [op, cons $ AST.Int 32 (fromIntegral 0)]
  op2 <- call (externf (AST.Name (fromString "get_json_from_array"))) [op, cons $ AST.Int 32 (fromIntegral 1)]
  if func == "get_json_from_array"
  then do
    idx <- functionCallLLVM "getdoub" op2
    let conv = AST.FPToUI idx llvmI32 []
    intidx <- instr $conv
    llvmCallExt2 op1 intidx func
  else llvmCallExt2 op1 op2 func

llvmCallExt3args :: AST.Operand -> String -> Codegen AST.Operand
llvmCallExt3args op func = do
  op1 <- call (externf (AST.Name (fromString "get_json_from_array"))) [op, cons $AST.Int 32 (fromIntegral 0)]
  op2 <- call (externf (AST.Name (fromString "get_json_from_array"))) [op, cons $AST.Int 32 (fromIntegral 1)]
  op3 <- call (externf (AST.Name (fromString "get_json_from_array"))) [op, cons $AST.Int 32 (fromIntegral 2)]
  call (externf (AST.Name (fromString func))) [op1, op2, op3]

llvmCallFunc :: String -> AST.Operand -> Codegen AST.Operand
llvmCallFunc fnName op = call (externf (AST.Name (fromString fnName))) [op]

--llvmCallGetArrIter :: AST.Operand -> Codegen AST.Operand
--llvmCallGetArrIter arr
--llvmArrayToPointer :: AST.Constant -> AST.Constant
--llvmArrayToPointer arr = AST.GetElementPtr True arr [AST.Int 32 0]

llvmCharArrayType :: Int -> AST.Type
llvmCharArrayType n = AST.ArrayType (fromIntegral n :: Word64) llvmI8

buildPtrArray :: [AST.Operand] -> Codegen AST.Operand
buildPtrArray ptrs = do
  mem <- instr $
    AST.Alloca llvmI32Pointer (Just (cons (AST.Int 32 (fromIntegral (length ptrs))))) 0 []
  forM_ [0..(length ptrs)-1] $ \i -> do
    let tempptr = AST.GetElementPtr True mem [cons $ AST.Int 32 (fromIntegral i)] []
    tempmem <- instr $ tempptr
    instr $ AST.Store False tempmem (ptrs!!i) Nothing (fromIntegral 0) []
  return mem

buildObjPtrArray :: [(String, AST.Operand)] -> Codegen AST.Operand
buildObjPtrArray ptrs = do
  mem <- instr $
    AST.Alloca llvmI32Pointer (Just (cons (AST.Int 32 (fromIntegral (2*(length
         ptrs)))))) 0 []
  forM_ [0..(length ptrs)-1] $ \i -> do
    let tempptr1 = AST.GetElementPtr True mem [cons $ AST.Int 32 (2*(
        fromIntegral i))] []
    tempmem1 <- instr $ tempptr1
    op <- stringLLVM (fst (ptrs!!i))
    instr $AST.Store False tempmem1 op Nothing (fromIntegral 0) []
    let tempptr2 = AST.GetElementPtr True mem [cons $ AST.Int 32 ((2*(
        fromIntegral i))+1)] []
    tempmem2 <- instr $ tempptr2
    instr $AST.Store False tempmem2 (snd (ptrs!!i)) Nothing (fromIntegral 0)
        []
  return mem

stringToLLVMString :: String -> AST.Constant
stringToLLVMString s = AST.Array llvmI8 (map charToLLVMInt s ++ [AST.Int 8 0])

charToLLVMInt :: Char -> AST.Constant
charToLLVMInt = AST.Int 8 . fromIntegral . ord

rawStringLLVM :: String -> Codegen AST.Operand
rawStringLLVM s = do
  let ptr =
        AST.Alloca (llvmCharArrayType (1+length s)) (Just (cons (AST.Int 32 1)
           )) 0 []
  op <- instr $ ptr
  let arrayS = stringToLLVMString s
  _ <- instr $ AST.Store False op (cons arrayS) Nothing 0 []
  let ref = AST.GetElementPtr True op [cons $ AST.Int 8 0, cons $ AST.Int 8 0]
       []
  op2 <- instr $ ref
  return op2

stringLLVM :: String -> Codegen AST.Operand
stringLLVM s = do
  op <- rawStringLLVM s
  functionCallLLVM "json_string" op
```

9.7 Codegen.hs

```
{-# LANGUAGE OverloadedStrings #-}
{-# LANGUAGE GeneralizedNewtypeDeriving #-}

module Codegen where

import Data.Word
import Data.String
```

```haskell
import Data.List
import Data.Function
import qualified Data.Map as Map

import Control.Monad.State
import Control.Applicative

import LLVM.AST hiding (type')
import LLVM.AST.Global
import qualified LLVM.AST as AST

import qualified LLVM.AST.Linkage as L
import qualified LLVM.AST.Constant as C
import qualified LLVM.AST.Attribute as A
import qualified LLVM.AST.CallingConvention as CC
import qualified LLVM.AST.FloatingPointPredicate as FP
import qualified LLVM.AST.IntegerPredicate as Intypoo

-------------------------------------------------------------------------------
-- Module Level
-------------------------------------------------------------------------------


newtype LLVM a = LLVM (State AST.Module a)
  deriving (Functor, Applicative, Monad, MonadState AST.Module )

runLLVM :: AST.Module -> LLVM a -> AST.Module
runLLVM mod (LLVM m) = execState m mod

emptyModule :: String -> AST.Module
emptyModule label = defaultModule { moduleName = fromString label }

addDefn :: Definition -> LLVM ()
addDefn d = do
  defs <- gets moduleDefinitions
  modify $ \s -> s { moduleDefinitions = defs ++ [d] }

globalVar ::  Type -> String -> C.Constant -> LLVM ()
globalVar ty label val = addDefn $
  GlobalDefinition $ globalVariableDefaults {
    name        = AST.Name (fromString label)
  , isConstant  = True
  , type'       = ty
  , initializer = Just val
  }

define ::  Type -> String -> [(Type, Name)] -> [BasicBlock] -> LLVM ()
define retty label argtys body = addDefn $
  GlobalDefinition $ functionDefaults {
    name        = AST.Name (fromString label)
  , parameters  = ([Parameter ty nm [] | (ty, nm) <- argtys], False)
  , returnType  = retty
  , basicBlocks = body
  }

external ::  Type -> String -> [(Type, Name)] -> LLVM ()
external retty label argtys = addDefn $
  GlobalDefinition $ functionDefaults {
    name        = AST.Name (fromString label)
  , linkage     = L.External
  , parameters  = ([Parameter ty nm [] | (ty, nm) <- argtys], False)
  , returnType  = retty
  , basicBlocks = []
  }

-------------------------------------------------------------------------------
-- Types
-------------------------------------------------------------------------------

-- IEEE 754 double
double :: Type
double = FloatingPointType DoubleFP
--double = FloatingPointType 64 IEEE

int :: Type
int = IntegerType 32
-------------------------------------------------------------------------------
-- Names
-------------------------------------------------------------------------------

type Names = Map.Map String Int

uniqueName :: String -> Names -> (String, Names)
uniqueName nm ns =
  case Map.lookup nm ns of
    Nothing -> (nm,  Map.insert nm 1 ns)
    Just ix -> (nm ++ show ix, Map.insert nm (ix+1) ns)

-------------------------------------------------------------------------------
-- Codegen State
-------------------------------------------------------------------------------

type SymbolTable = [(String, Operand)]

data CodegenState
  = CodegenState {
    currentBlock :: Name                     -- Name of the active block to
        append to
  , blocks       :: Map.Map Name BlockState  -- Blocks for function
  , symtab       :: SymbolTable              -- Function scope symbol table
  , blockCount   :: Int                      -- Count of basic blocks
  , count        :: Word                     -- Count of unnamed instructions
  , names        :: Names                    -- Name Supply
  } deriving Show

data BlockState
  = BlockState {
    idx   :: Int                            -- Block index
  , stack :: [Named Instruction]            -- Stack of instructions
  , term  :: Maybe (Named Terminator)       -- Block terminator
  } deriving Show

-------------------------------------------------------------------------------
-- Codegen Operations
-------------------------------------------------------------------------------

newtype Codegen a = Codegen { runCodegen :: State CodegenState a }
  deriving (Functor, Applicative, Monad, MonadState CodegenState )

sortBlocks :: [(Name, BlockState)] -> [(Name, BlockState)]
sortBlocks = sortBy (compare `on` (idx . snd))

createBlocks :: CodegenState -> [BasicBlock]
createBlocks m = map makeBlock $ sortBlocks $ Map.toList (blocks m)

makeBlock :: (Name, BlockState) -> BasicBlock
makeBlock (l, (BlockState _ s t)) = BasicBlock l (reverse s) (maketerm t)
  where
    maketerm (Just x) = x
    maketerm Nothing = error $ "Block has no terminator: " ++ (show l)

entryBlockName :: String
entryBlockName = "entry"

emptyBlock :: Int -> BlockState
emptyBlock i = BlockState i [] Nothing

emptyCodegen :: CodegenState
emptyCodegen = CodegenState (AST.Name (fromString entryBlockName)) Map.empty
    [] 1 0 Map.empty

execCodegen :: Codegen a -> CodegenState
execCodegen m = execState (runCodegen m) emptyCodegen

fresh :: Codegen Word
fresh = do
  i <- gets count
  modify $ \s -> s { count = 1 + i }
  return $ i + 1

instr :: Instruction -> Codegen (Operand)
instr ins = do
  n <- fresh
  let ref = (UnName n)
  blk <- current
  let i = stack blk
  modifyBlock (blk { stack = (ref := ins) : i } )
  return $ local ref

terminator :: Named Terminator -> Codegen (Named Terminator)
terminator trm = do
  blk <- current
  modifyBlock (blk { term = Just trm })
  return trm

-------------------------------------------------------------------------------
-- Block Stack
-------------------------------------------------------------------------------


entry :: Codegen Name
entry = gets currentBlock

addBlock :: String -> Codegen AST.Name
addBlock bname = do
  bls <- gets blocks
  ix  <- gets blockCount
  nms <- gets names

  let new = emptyBlock ix
      (qname, supply) = uniqueName bname nms

  modify $ \s -> s { blocks = Map.insert (AST.Name (fromString qname)) new bls
                   , blockCount = ix + 1
                   , names = supply
                   }
  return (AST.Name (fromString qname))

setBlock :: Name -> Codegen Name
setBlock bname = do
  modify $ \s -> s { currentBlock = bname }
  return bname

getBlock :: Codegen Name
getBlock = gets currentBlock

modifyBlock :: BlockState -> Codegen ()
modifyBlock new = do
  active <- gets currentBlock
  modify $ \s -> s { blocks = Map.insert active new (blocks s) }

current :: Codegen BlockState
current = do
  c <- gets currentBlock
  blks <- gets blocks
  case Map.lookup c blks of
```

```
    Just x -> return x
    Nothing -> error $ "No such block: " ++ show c

-------------------------------------------------------------------------------
-- Symbol Table
-------------------------------------------------------------------------------

assign :: String -> Operand -> Codegen ()
assign var x = do
  lcls <- gets symtab
  modify $ \s -> s { symtab = [(var, x)] ++ lcls }

getvar :: String -> Codegen (Maybe Operand)
getvar var = do
  syms <- gets symtab
  return $ lookup var syms

-------------------------------------------------------------------------------

-- References
local ::  Name -> Operand
local = LocalReference double

global ::  Name -> C.Constant
global = C.GlobalReference double

externf :: Name -> Operand
externf = ConstantOperand . C.GlobalReference double

-- Arithmetic and Constants
fadd :: Operand -> Operand -> Codegen Operand
fadd a b = instr $ FAdd NoFastMathFlags a b []

fsub :: Operand -> Operand -> Codegen Operand
fsub a b = instr $ FSub NoFastMathFlags a b []

fmul :: Operand -> Operand -> Codegen Operand
fmul a b = instr $ FMul NoFastMathFlags a b []

fdiv :: Operand -> Operand -> Codegen Operand
fdiv a b = instr $ FDiv NoFastMathFlags a b []

fmod :: Operand -> Operand -> Codegen Operand
fmod a b = instr $ FRem NoFastMathFlags a b []

fcmp :: FP.FloatingPointPredicate -> Operand -> Operand -> Codegen Operand
fcmp cond a b = instr $ FCmp cond a b []

icmp :: Intypoo.IntegerPredicate -> Operand -> Operand -> Codegen Operand
icmp cond a b = instr $ ICmp cond a b []

```

```haskell
cons :: C.Constant -> Operand
cons = ConstantOperand

uitofp :: Type -> Operand -> Codegen Operand
uitofp ty a = instr $ UIToFP a ty []

toArgs :: [Operand] -> [(Operand, [A.ParameterAttribute])]
toArgs = map (\x -> (x, []))

-- Effects
call :: Operand -> [Operand] -> Codegen Operand
call fn args = instr $ Call Nothing CC.C [] (Right fn) (toArgs args) [] []

alloca :: Type -> Codegen Operand
alloca ty = instr $ Alloca ty Nothing 0 []

store :: Operand -> Operand -> Codegen Operand
store ptr val = instr $ Store False ptr val Nothing 0 []

load :: Operand -> Codegen Operand
load ptr = instr $ Load False ptr Nothing 0 []

-- Control Flow
br :: Name -> Codegen (Named Terminator)
br val = terminator $ Do $ Br val []

cbr :: Operand -> Name -> Name -> Codegen (Named Terminator)
cbr cond tr fl = terminator $ Do $ CondBr cond tr fl []

ret :: Maybe Operand -> Codegen (Named Terminator)
ret val = terminator $ Do $ Ret val []

phi :: Type -> [(Operand, Name)] -> Codegen Operand
phi typ res = instr $ Phi typ res []
```

9.8 client.cpp

```cpp
#include <iostream>

#include <cpr/cpr.h>
#include <string>
#include "rapidjson/document.h"
#include "rapidjson/writer.h"
#include "rapidjson/stringbuffer.h"
#include "../jsonlib/jsonlib.h"

using namespace rapidjson;

extern "C" {
  int* post(const char* url, int* json, const char* key, const char* secret,
      const char* header){

    try{
      const char* payload = body_tostring(json);
      std::string urlCpp(url);
      char* ret;
      if(strlen(key) > 0 && strlen(secret) > 0){
        auto r = cpr::Post(cpr::Url{urlCpp}, cpr::Body{payload},cpr::Header{{"
            Content-Type","application/json"}}, cpr::Authentication{key,
            secret});
        ret = (char*) malloc(strlen(r.text.c_str())+1);
        strcpy(ret, r.text.c_str());
      }
      else{
        auto r = cpr::Post(cpr::Url{urlCpp}, cpr::Body{payload},cpr::Header{{"
            Content-Type","application/json"}});
        ret = (char*) malloc(strlen(r.text.c_str())+1);
        strcpy(ret, r.text.c_str());
      }
      return (int *) ret;
    }
    catch(...){
      throw std::runtime_error("Failed post");
    }
  }

  int* exposed_post(int* req){
    Document* d = (Document*) req;
    if ((*d).HasMember("url")){
      const char* url = tostring((int*)(&((*d)["url"])));
      int* body;
            if((*d).HasMember("body")){
        body = (int*) (&((*d)["body"]));
      }
      else if((*d).HasMember("payload")){
        body = (int*) (&((*d)["payload"]));
      }
      else{
        body = (int*) new Document();
      }

      const char* key;
      if((*d).HasMember("key")){
        key = tostring((int*)(&((*d)["key"])));
      }
      else{
        key = "";
      }

      const char* secret;
      if((*d).HasMember("secret")){
        secret = tostring((int*)(&((*d)["secret"])));
      }
      else{
        secret = "";
      }
                  const char* header;
      if((*d).HasMember("header")){
        header = tostring((int*)(&((*d)["header"])));
      }
      else{
        header = "";
      }


      return post(url, body, key, secret, header);
    }
    throw std::runtime_error("Post did not contain URL!");
  }

  int* get(const char* url, int* json, const char* key, const char* secret,
      const char* header) {
    try{
      const char* body = body_tostring(json);
      std::string urlCpp(url);
      auto r = cpr::Get(cpr::Url{urlCpp}, cpr::Payload{{"arg", body}});
      char* ret = (char*) malloc(strlen(r.text.c_str())+1);
      strcpy(ret, r.text.c_str());
      return (int *) ret;
    }
    catch(...){
      throw std::runtime_error("Failed get");
    }
  }
  }
```

9.9 jsonlib.cpp

```
#include "rapidjson/document.h"
#include "rapidjson/writer.h"
#include "rapidjson/stringbuffer.h"
#include "rapidjson/allocators.h"
#include <iostream>
#include <cmath>
#include <sstream>
#include "jsonlib.h"
using namespace rapidjson;
extern "C"{

int* is_json_double(int*);
int* is_json_string(int*);
int* json_double(double);
int* json_string(const char*);
void unflatten(int*,Document::AllocatorType& allo);


Value& getp(int* intdoc, const char* key){
  Document* d = (Document*)intdoc;
  return (*d)[key];
}
```

```

const char* body_tostring(int* tempdoc){
  std::cout.flush();
  if((*((Document*) tempdoc)).IsObject()){
    Value* str = (Value*)tempdoc;
    if(str->IsString())
      return str->GetString();
    if((*((Document*) tempdoc)).HasMember("prim_type")){
      Value& typ = getp(tempdoc, "prim_type");
      if(typ.GetString() == "num"){
        std::ostringstream strdoub;
        strdoub << getp(tempdoc, "prim_val").GetDouble() << '\0';
        char* ret = (char*) malloc(strlen(strdoub.str().c_str())+1);
        strcpy(ret, strdoub.str().c_str());
        return ret;
      }
      else if(typ.GetString() == "str"){
        std::ostringstream strstr;
        Value& pt = getp(tempdoc, "prim_val");
        strstr << "\'" << pt.GetString() << "\'" << '\0';
        char* ret = (char*) malloc(strlen(strstr.str().c_str())+1);
        strcpy(ret, strstr.str().c_str());
        return ret;
      }
      else if(typ.GetString() == "bool"){
        Value& pt = getp(tempdoc, "prim_val");
        if(pt.GetBool())
          return "true";
        else
          return "false";
      }
    }
    else {
      Document* d = (Document *)tempdoc;
      std::ostringstream objstr;
      objstr<<"{";
      for (Value::ConstMemberIterator itr = (*d).MemberBegin(); itr != (*d).
          MemberEnd(); ++itr){
        objstr << "\'" << itr->name.GetString() << "\':" << body_tostring((int
            *)&(itr->value));
        if (itr+1 != (*d).MemberEnd())
          objstr << ",";
      }
      objstr << "}" <<'\0';
      char* ret = (char*) malloc(strlen(objstr.str().c_str())+1);
      strcpy(ret, objstr.str().c_str());
      return ret;
    }
  }
  else if((*((Document*) tempdoc)).IsArray()){
    Document* d = (Document *)tempdoc;
    std::ostringstream objstr;
    objstr << "[";
    for (Value::ConstValueIterator itr = (*d).Begin(); itr != (*d).End(); ++
        itr){
      objstr << body_tostring((int *) itr);
      if (itr+1 !=(*d).End())
        objstr << ",";
    }
    objstr << "]" << '\0';
    char* ret = (char*) malloc(strlen(objstr.str().c_str())+1);
    strcpy(ret, objstr.str().c_str());

    return ret;
  }
  else{
    return (char *) tempdoc;
  }
}

const char* tostring(int* tempdoc){
  std::cout.flush();
  if((*((Document*) tempdoc)).IsObject()){
    Value* str = (Value*)tempdoc;
    if(str->IsString())
      return str->GetString();
    if((*((Document*) tempdoc)).HasMember("prim_type")){
      Value& typ = getp(tempdoc, "prim_type");
      if(typ.GetString() == "num"){
        std::ostringstream strdoub;
        strdoub << getp(tempdoc, "prim_val").GetDouble() << '\0';
        char* ret = (char*) malloc(strlen(strdoub.str().c_str())+1);
        strcpy(ret, strdoub.str().c_str());
        return ret;
      }
      else if(typ.GetString() == "str"){
        std::ostringstream strstr;
        Value& pt = getp(tempdoc, "prim_val");
        strstr << pt.GetString() << '\0';
        char* ret = (char*) malloc(strlen(strstr.str().c_str())+1);
        strcpy(ret, strstr.str().c_str());
        return ret;
      }
      else if(typ.GetString() == "bool"){
        Value& pt = getp(tempdoc, "prim_val");
        if(pt.GetBool())
          return "true";
        else
          return "false";
      }
    }
    else {
      Document* d = (Document *)tempdoc;
      std::ostringstream objstr;
      objstr<<"{";
      for (Value::ConstMemberIterator itr = (*d).MemberBegin(); itr != (*d).
          MemberEnd(); ++itr){
        objstr << itr->name.GetString() << ":" << tostring((int*)&(itr->value)
            );
        if (itr+1 != (*d).MemberEnd())
          objstr << ",";
      }
      objstr << "}" <<'\0';
      char* ret = (char*) malloc(strlen(objstr.str().c_str())+1);
      strcpy(ret, objstr.str().c_str());
      return ret;
    }
  }
  else if((*((Document*) tempdoc)).IsArray()){
    Document* d = (Document *)tempdoc;
    std::ostringstream objstr;
    objstr << "[";
    for (Value::ConstValueIterator itr = (*d).Begin(); itr != (*d).End(); ++
        itr){
      objstr << tostring((int *) itr);
      if (itr+1 !=(*d).End())
        objstr << ",";
    }
    objstr << "]" << '\0';
    char* ret = (char*) malloc(strlen(objstr.str().c_str())+1);
    strcpy(ret, objstr.str().c_str());

    return ret;
  }
  else{
    return (char *) tempdoc;
  }
}

const char* internaltostring(int* tempdoc){
  std::ostringstream strstr;
  Value& pt = getp(tempdoc, "prim_val");
  strstr << pt.GetString() << '\0';
  char* ret = (char*) malloc(strlen(strstr.str().c_str())+1);
  strcpy(ret, strstr.str().c_str());
  return ret;
}

int* json_bool(int b){
  Document *d = new Document();
  (*d).SetObject();
  if(b==0){
    (*d).AddMember("prim_type", "bool", (*d).GetAllocator());
    (*d).AddMember("prim_val", false, (*d).GetAllocator());
  }
  else{
    (*d).AddMember("prim_type", "bool", (*d).GetAllocator());
    (*d).AddMember("prim_val", true, (*d).GetAllocator());
  }
  return (int*)d;
}

```

```
int* is_json_bool(int* intdoc){
  Document *d = (Document *) intdoc;
  if((*d).IsObject() && (*d).HasMember("prim_type")){
    Value& typ = getp(intdoc, "prim_type");
    if (typ.GetString() == "bool")
      return json_bool(1);
    return json_bool(0);
  }
  return json_bool(0);
}

double get_json_bool(int* intdoc){
  Value& pt = getp(intdoc, "prim_type");
  if(pt.GetString() == "bool"){
    if(getp(intdoc, "prim_val").GetBool())
      return 1;
    return 0;
  }
  return 0;
}


int* json_from_string(int* s){
  if(get_json_bool(is_json_object(s))){
    return s;
  }
  const char* str = tostring(s);
  Document* init = new Document();
  (*init).Parse(str);

  unflatten((int*) init, (*init).GetAllocator());
  return (int*) init;
}

void unflatten(int* temp, Document::AllocatorType& allo){
  Document* init = (Document *) temp;
  StringBuffer buffer;
  if((*init).IsObject()){
    for (Value::ConstMemberIterator itr = (*init).MemberBegin(); itr != (*init
        ).MemberEnd(); ++itr){
      if(itr->value.IsNumber()){
        int* tempjdubs = json_double(itr->value.GetDouble());
        Document* jdubs = (Document *) tempjdubs;
        (*init)[itr->name].CopyFrom(*jdubs,allo);
      }
      else if(itr->value.IsString()){
        int* tempjstr = json_string(itr->value.GetString());
        Document* jstr = (Document *) tempjstr;
        (*init)[itr->name].CopyFrom(*jstr,allo);
      }
      else if(itr->value.IsBool()){
        int* tempjbool;
        if (itr->value.GetBool()){
          tempjbool = json_bool(1);
        }
        else {
          tempjbool = json_bool(0);
        }
        Document* jbool = (Document*) tempjbool;
        (*init)[itr->name].CopyFrom(*jbool,allo);
      }
      else{
        unflatten((int *)(&(itr->value)), allo);
      }
    }
  }
  else if ((*init).IsArray()){
    int count = 0;
    for (Value::ConstValueIterator itr = (*init).Begin(); itr != (*init).End()
       ; ++itr){

      if(itr->IsNumber()){
        int* tempjdubs = json_double(itr->GetDouble());
        Document* jdubs = (Document *) tempjdubs;
        (*init)[count].CopyFrom(*jdubs,allo);
      }
      else if(itr->IsString()){
        int* tempjstr = json_string(itr->GetString());
        Document* jstr = (Document *) tempjstr;
        (*init)[count].CopyFrom(*jstr,allo);
      }
      else if(itr->IsBool()){
        int* tempjbool;
        if (itr->GetBool()){
          tempjbool = json_bool(1);
        }
        else {
          tempjbool = json_bool(0);
        }
        Document* jbool = (Document*) tempjbool;
        (*init)[count].CopyFrom(*jbool,allo);
      }
      else{
        unflatten((int*) itr, allo);
      }
      count++;
    }
  }
  else{
    throw std::runtime_error("Attempting to parse a string that did not
       contain an object or array. Terminating.");
  }

}

int* parse_function_arg(int* st){
  const char* str = tostring(st);
  try{
    double dub = std::stod(str);
    return json_double(dub);
  }
  catch(std::invalid_argument){
    //I guess its not a double
  }

  try{
    return json_from_string(st);
  }
  catch(std::runtime_error){

    //Not an arr or obj either
  }

  if(str=="true")
    return json_bool(1);
  else if(str=="false")
    return json_bool(0);

  //Not a bool either? Guess it must be a string!
  return st;
}

int* json_object(int* a[], int num){
  Document *d = new Document();
  (*d).SetObject();
  Document::AllocatorType& allocator = (*d).GetAllocator();
  for(int i = 0; i < num; i++){
    Value tempkey;
    Value tempvalue;
    tempkey.SetString(tostring(a[2*i]), allocator);
    tempvalue.CopyFrom(*((Document *)a[2*i+1]), allocator);
    (*d).AddMember(tempkey, tempvalue, allocator);
  }

  return (int *)d;

}

int* get_json_from_object(int* intdoc, int* key){
  Document* d = (Document*) intdoc;
  const char* skey = tostring(key);
  if((*d).HasMember(skey)){
    return (int*)(&(getp((int*)d, skey)));
  }
  else{
    throw std::runtime_error("Json object did not contain key");
  }
}

int* is_json_object(int* s){
  Document *d = (Document *) s;
  if((*d).IsObject() && !get_json_bool(is_json_double(s))
      && !get_json_bool(is_json_string(s))
      && !get_json_bool(is_json_bool(s)))
    return json_bool(1);
  return json_bool(0);
}

int* add_to_json_object(int *intdoc, int* jkey, int* jvalue){
  const char* key = tostring(jkey);
  const char* value = tostring(jvalue);
  Document* d = (Document*)intdoc;
  Document* findoc = new Document();
  (*findoc).CopyFrom((*d), (*findoc).GetAllocator());
  Value tempkey;
  tempkey.SetString(key, (*findoc).GetAllocator());
  Value tempvalue;
  tempvalue.CopyFrom(*((Document*)jvalue),(*findoc).GetAllocator());

  if ((*d).HasMember(key)){
    (*d)[key] = tempvalue;
    return intdoc;
  }
  else{
    (*findoc).AddMember(tempkey, tempvalue, (*findoc).GetAllocator());
    (*d).CopyFrom((*findoc), (*findoc).GetAllocator());
    return (int*) findoc;
  }
}


//Create a double in json by creating a json object with json_rep_of_num_ts as
    key
int* json_double(double dubs){
  Document *d = new Document();
  (*d).SetObject();
  Value db(dubs);
  (*d).AddMember("prim_type", "num", (*d).GetAllocator());
  (*d).AddMember("prim_val", dubs, (*d).GetAllocator());
  return (int*)d;
}


int* to_json_double(int* intdoc){
  Document *d = new Document();
  Document *old = (Document*) intdoc;
  (*d).SetObject();

  if((*old).IsObject() && (*old).HasMember("prim_type")){
    Value& typ = getp(intdoc, "prim_type");
    if (typ.GetString() == "str"){
      Value& val = getp(intdoc, "prim_val");
      double temp = std::stod(val.GetString());
      return json_double(temp);
    }
  }
  std::runtime_error("TypeMismatch: toNum passed non string");
  return NULL;
}


int * is_json_double(int* intdoc){
  Document *d = (Document *) intdoc;
  if((*d).IsObject() && (*d).HasMember("prim_type")){
    Value& typ = getp(intdoc, "prim_type");
    if (typ.GetString() == "num")
      return json_bool(1);
    return json_bool(0);
  }
  return json_bool(0);
}

//Retrieve a double in json
double get_json_double(int* intdoc){
  Value& pt = getp(intdoc, "prim_type");
  if(pt.GetString() == "num")
    return getp(intdoc, "prim_val").GetDouble();
  else if(pt.GetString() == "bool");
    return get_json_bool(intdoc);
  return pt.GetDouble();
}


//Create a string in json by creating a json object wit json_rep_of_str_ts as
   key
int* json_string(const char* s){
  Document *d = new Document();
  (*d).SetObject();
  Value tempvalue;
  tempvalue.SetString(s, (*d).GetAllocator());
  (*d).AddMember("prim_type", "str", (*d).GetAllocator());
  (*d).AddMember("prim_val", tempvalue.Move(), (*d).GetAllocator());
  return (int*)d;

}

int* is_json_string(int* intdoc){
  Document *d = (Document *) intdoc;
  if((*d).IsObject() && (*d).HasMember("prim_type")){
    Value& typ = getp(intdoc, "prim_type");
    if (typ.GetString() == "str")
      return json_bool(1);
    return json_bool(0);
  }
  return json_bool(0);
}

int* is_string_equal(int* st1, int* st2){
  const char* str1 = body_tostring(st1);
  const char* str2 = body_tostring(st2);
  if(!strcmp(str1, str2)){
    return json_bool(1);
  }
  return json_bool(0);
}

int* concat(int* st1, int* st2){
  const char* str1 = tostring(st1);
  const char* str2 = tostring(st2);
  std::ostringstream retcharst;
  retcharst << str1 << str2;
  return json_string(retcharst.str().c_str());
}

//Create an array in json from json values/docs (by coyping each value/doc
    into a new value and adding that to our new doc
int* json_array(int* a[], int numElements){
  Document *d = new Document();
  (*d).SetArray();
  Document::AllocatorType& allocator = (*d).GetAllocator();
  for(int i = 0; i < numElements; i++){
    Value tempdoc;
          tempdoc.CopyFrom(*((Document *)(a[i])), allocator);
     (*d).PushBack(tempdoc, allocator);
  }
  return (int *)d;
}

int* is_json_array(int* intdoc){
  Document *d = (Document *) intdoc;
  if((*d).IsArray())
    return json_bool(1);
  return json_bool(0);
}

//Return a pointer to the value at the idxth position
int* get_json_from_array(int* arr, int idx){
  Document* d = (Document *) arr;
  return (int *)(&((*d)[idx]));
}

int* push_to_json_array(int* arr, int* add){
  Document* d = (Document*) arr;
  Document* findoc = new Document();
  (*findoc).CopyFrom((*d), (*findoc).GetAllocator());

  Value tempvalue;
  tempvalue.CopyFrom(*((Document*)add), (*findoc).GetAllocator());
  (*findoc).PushBack(tempvalue, (*findoc).GetAllocator());
  return (int*) findoc;
}

int* replace_json_array_element(int* temparr, int* tempel, int* tempidx){
  Document* findoc = new Document();
  int idx = (int) get_json_double(tempidx);
  Document* arr = (Document *) temparr;
  (*findoc).CopyFrom((*arr), (*findoc).GetAllocator());

  Value tempvalue;
  tempvalue.CopyFrom(*((Document*)tempel),(*findoc).GetAllocator());
  (*findoc)[idx] = tempvalue;
  return (int*) findoc;
}

//Create a null in json by creating a json object with json_rep_of_null_ts as
   key
int* json_null(){
  Document *d = new Document();
  (*d).SetObject();
  (*d).AddMember("json_rep_of_null_ts", NULL, (*d).GetAllocator());
  return (int*)d;
}
//Retrieve a null in json
int* get_json_null(int* intdoc){
  Value& pt = getp(intdoc, "json_rep_of_null_ts");
  return (int*)&pt;
}

int test(const char* s){
  std::cout << "HI" << std::endl;
  return 3;
}

int* jgets(int* intdoc, int* key){
  Document* d = (Document*) intdoc;
  if ((*d).IsObject()){
    const char* skey = tostring(key);
    if((*d).HasMember(skey)){
      return (int*)(&(getp((int*)d, skey)));
    }
    else{
      return 0;
    }
  }
  else {
    double dubidx = get_json_double(key);
    int idx = (int) std::round(dubidx);
    if(idx < (*d).Size()){
      return (int *)(&((*d)[idx]));
    }
    return 0;
  }
}


int* create_arr_iter(int* jsonthingie){
  Document* d = (Document*)jsonthingie;
  Value::ConstValueIterator itr = (*d).Begin();
  return (int *) itr;
}

int* arr_next_elem(int* itr, int* intdoc){
  Value::ConstValueIterator iter = (Value::ConstValueIterator) itr;
  int* elem = (int*)(++iter);
  if (elem == ((int *)(*((Document*)(intdoc))).End())){
    return 0;
  }
  else
    return elem;
}
int* create_obj_iter(int* jsonthingie){
  Document *d = (Document*)jsonthingie;
  Value::ConstMemberIterator itr = (*d).MemberBegin();
  return (int *) &(*itr);
}

/*
// Test function
int main(){
  //testing parse
  const char* test = "{\"test\":\"christophe\"}";
<<<<<<< HEAD
=======
  std::cout << sizeof(int) <<std::endl;
>>>>>>> 2819b078101c1be2c044ab36b88749629b683c08
  int* j = json(test);

  //testing adds
  adds(j, "boop", "is");
  adds(j, "test", "w");

  //testing string
  //std::cout << tostring(j) << std::endl;
  //std::cout << (*((Document*)j))["test"].GetString() << std::endl;

  int* d = json_double(3);
  int* s = json_string("waduuuup");
  int* pts[] = {d, s};
  int* pt = json_array(pts, 2);

<<<<<<< HEAD
  //std::cout << get_json_double(d) << std::endl;
  //std::cout << get_json_string(s) << std::endl;
  //std::cout << get_json_double(get_json_from_array(pt,0)) << std::endl;


  int* arr_itr = create_arr_iter(pt);

  print(pt);
  while(arr_itr){
    std::cout << get_json_double(arr_itr) <<std::endl;
    arr_itr = arr_next_elem(arr_itr, pt);
  }

=======
  std::cout << get_json_double(d) << std::endl;
  std::cout << get_json_string(s) << std::endl;
  std::cout << get_json_double(get_json_from_array(pt,0)) << std::endl;
>>>>>>> 2819b078101c1be2c044ab36b88749629b683c08

  return 0;
}*/
}
```

9.10 weblang - Build File

This is the executable used to compile and build a Weblang program.

```
#!/bin/bash
wlfile=$1
length=${#wlfile}
pathwoextension=${wlfile:0:length-3}
filename=${pathwoextension##*/}
updated=false
set -e

echo "Compiling $wlfile to produce executable $filename"
stack build --nix :weblang

if [ ! -d intermediary ]; then
  mkdir intermediary
fi

if [ ! -f intermediary/$filename.ll ] || [ $wlfile -nt intermediary/$filename.
    ll ]; then
  stack exec weblang $wlfile intermediary/$filename.ll
  echo "intermediary/$filename.ll written"
  nix-shell -p llvm --command "llc intermediary/$filename.ll"
  echo "intermediary/$filename.s written"
  nix-shell -p gcc --command "g++ -c intermediary/$filename.s -o intermediary/
      $filename.o"
  echo "intermediary/$filename.o written"
  updated=true
else
  echo "No updates to $wlfile since last compilation; not compiling this
      component"
fi

if [ ! -f jsonlib/jsonlib.o ] || [ jsonlib/jsonlib.cpp -nt jsonlib/jsonlib.o
    ]; then
  nix-shell -p rapidjson gcc --command "g++ \$NIX_CFLAGS_COMPILE -c jsonlib/
      jsonlib.cpp -o jsonlib/jsonlib.o"
  echo "jsonlib/jsonlib.o built"
  updated=true
else
  echo "No updates to jsonlib/jsonlib.cpp since last compilation; not
      compiling this component"
```

```
fi

if [ ! -f client/client.o ] || [ client/client.cpp -nt client/client.o ]; then
  nix-shell -p rapidjson gcc --command "gcc -Wall \$NIX_CFLAGS_COMPILE -c
      client/client.cpp -Iclient/cpr-example/opt/cpr/include -Iclient/cpr-
      example/opt/json/src -Lclient/cpr-example/build/lib -lcpr -lcurl -o
      client/client.o"
  echo "client/client.o built"
  updated=true
else
  echo "No updates to client/client.cpp since last compilation; not compiling
      this component"
fi

if [ ! -f $filename ] || [ $updated ]; then
        nix-shell -p curl gcc --command "g++ intermediary/$filename.o client/
            client.o jsonlib/jsonlib.o -o $filename -Lclient/cpr-example/build
            /lib -lcpr -lcurl"

  echo "Executable has been built: $filename"
else
  echo "No files were updated throughout the process and the executable
      $filename still exists; use that you bozo"
fi
```

9.11 makeserver.hs

```
#!/bin/bash

rm -rf .libs

cd ./libmicrohttpd-0.9.55/src/examples; make;

cd -
echo "copying executable"
cp ./libmicrohttpd-0.9.55/src/examples/post_example ./runWeblangServer

echo "copying libs"
cp -rf ./libmicrohttpd-0.9.55/src/examples/.libs .
```

9.12 runWeblangServer

This is the executable used to run the Weblang server.

```
#! /bin/bash

# post_example - temporary wrapper script for .libs/post_example
# Generated by libtool (GNU libtool) 2.4.6 Debian-2.4.6-2
#
# The post_example program cannot be directly executed until all the libtool
# libraries that it depends on are installed.
#
# This wrapper script should never be moved out of the build directory.
# If it is, it will not operate correctly.

# Sed substitution that helps us do robust quoting.  It backslashifies
# metacharacters that are still active within double-quoted strings.
sed_quote_subst='s|\([`"$\\]\)|\\\1|g'

# Be Bourne compatible
if test -n "${ZSH_VERSION+set}" && (emulate sh) >/dev/null 2>&1; then
  emulate sh
  NULLCMD=:
  # Zsh 3.x and 4.x performs word splitting on ${1+"$@"}, which
  # is contrary to our usage.  Disable this feature.
  alias -g '${1+"$@"}'='"$@"'
  setopt NO_GLOB_SUBST
else
  case `(set -o) 2>/dev/null` in *posix*) set -o posix;; esac
fi
BIN_SH=xpg4; export BIN_SH # for Tru64
DUALCASE=1; export DUALCASE # for MKS sh

# The HP-UX ksh and POSIX shell print the target directory to stdout
# if CDPATH is set.
(unset CDPATH) >/dev/null 2>&1 && unset CDPATH

relink_command=""

# This environment variable determines our operation mode.
if test "$libtool_install_magic" = "%%%MAGIC variable%%%"; then
  # install mode needs the following variables:
  generated_by_libtool_version='2.4.6'
  notinst_deplibs=' ../../src/microhttpd/libmicrohttpd.la'
else
  # When we are sourced in execute mode, $file and $ECHO are already set.
  if test "$libtool_execute_magic" != "%%%MAGIC variable%%%"; then
    file="$0"

# A function that is used when there is no print builtin or printf.
func_fallback_echo ()
{
  eval 'cat <<_LTECHO_EOF
$1
_LTECHO_EOF'
}
    ECHO="printf %s\\n"
  fi

# Very basic option parsing. These options are (a) specific to
# the libtool wrapper, (b) are identical between the wrapper
# /script/ and the wrapper /executable/ that is used only on
# windows platforms, and (c) all begin with the string --lt-
# (application programs are unlikely to have options that match
# this pattern).
#
# There are only two supported options: --lt-debug and
# --lt-dump-script. There is, deliberately, no --lt-help.
#
# The first argument to this parsing function should be the
# script's ../../libtool value, followed by no.
lt_option_debug=
func_parse_lt_options ()
{
  lt_script_arg0=$0
  shift
  for lt_opt
  do
    case "$lt_opt" in
    --lt-debug) lt_option_debug=1 ;;
    --lt-dump-script)
        lt_dump_D=`$ECHO "X$lt_script_arg0" | /bin/sed -e 's/^X//' -e 's%/[^/]*$%%'`
        test "X$lt_dump_D" = "X$lt_script_arg0" && lt_dump_D=.
        lt_dump_F=`$ECHO "X$lt_script_arg0" | /bin/sed -e 's/^X//' -e 's%^.*/%%'`
        cat "$lt_dump_D/$lt_dump_F"
        exit 0
      ;;
    --lt-*)
        $ECHO "Unrecognized --lt- option: '$lt_opt'" 1>&2
        exit 1
      ;;
    esac
  done

  # Print the debug banner immediately:
  if test -n "$lt_option_debug"; then
    echo "post_example:post_example:$LINENO: libtool wrapper (GNU libtool) 2.4.6 Debian-2.4.6-2" 1>&2
  fi
}

# Used when --lt-debug. Prints its arguments to stdout
# (redirection is the responsibility of the caller)
func_lt_dump_args ()
{
  lt_dump_args_N=1;
  for lt_arg
  do
    $ECHO "post_example:post_example:$LINENO: newargv[$lt_dump_args_N]: $lt_arg"
    lt_dump_args_N=`expr $lt_dump_args_N + 1`
  done
}

# Core function for launching the target application
func_exec_program_core ()
{

      if test -n "$lt_option_debug"; then
```

```
        $ECHO "post_example:post_example:$LINENO: newargv[0]: $progdir/
            $program" 1>&2
        func_lt_dump_args ${1+"$@"} 1>&2
      fi
      exec "$progdir/$program" ${1+"$@"}

      $ECHO "$0: cannot exec $program $*" 1>&2
      exit 1
}

# A function to encapsulate launching the target application
# Strips options in the --lt-* namespace from $@ and
# launches target application with the remaining arguments.
func_exec_program ()
{
  case " $* " in
  *\ --lt-*)
    for lt_wr_arg
    do
      case $lt_wr_arg in
      --lt-*) ;;
      *) set x "$@" "$lt_wr_arg"; shift;;
      esac
      shift
    done ;;
  esac
  func_exec_program_core ${1+"$@"}
}

  # Parse options
  func_parse_lt_options "$0" ${1+"$@"}

  # Find the directory that this script lives in.
  thisdir=`$ECHO "$file" | /bin/sed 's%/[^/]*$%%'`
  test "x$thisdir" = "x$file" && thisdir=.

  # Follow symbolic links until we get to the real thisdir.
  file=`ls -ld "$file" | /bin/sed -n 's/.*-> //p'`
  while test -n "$file"; do
    destdir=`$ECHO "$file" | /bin/sed 's%/[^/]*$%%'`

    # If there was a directory component, then change thisdir.
    if test "x$destdir" != "x$file"; then
      case "$destdir" in
      [\\/]* | [A-Za-z]:[\\/]*) thisdir="$destdir" ;;
      *) thisdir="$thisdir/$destdir" ;;
      esac
    fi

    file=`$ECHO "$file" | /bin/sed 's%^.*/%%'`
    file=`ls -ld "$thisdir/$file" | /bin/sed -n 's/.*-> //p'`
  done

  # Usually 'no', except on cygwin/mingw when embedded into
  # the cwrapper.
  WRAPPER_SCRIPT_BELONGS_IN_OBJDIR=no
  if test "$WRAPPER_SCRIPT_BELONGS_IN_OBJDIR" = "yes"; then
    # special case for '.'
    if test "$thisdir" = "."; then
      thisdir=`pwd`
    fi
    # remove .libs from thisdir
    case "$thisdir" in
    *[\\/].libs ) thisdir=`$ECHO "$thisdir" | /bin/sed 's%[\\/][^\\/]*$%%'` ;;
    .libs )   thisdir=. ;;
    esac
  fi

  # Try to get the absolute directory name.
  absdir=`cd "$thisdir" && pwd`
  test -n "$absdir" && thisdir="$absdir"

  program='post_example'
  progdir="$thisdir/.libs"


  if test -f "$progdir/$program"; then
    # Add our own library path to LD_LIBRARY_PATH
    LD_LIBRARY_PATH="/home/jordanvega/plt/libmicrohttpd-0.9.55/src/microhttpd
       /.libs:$LD_LIBRARY_PATH"

    # Some systems cannot cope with colon-terminated LD_LIBRARY_PATH
    # The second colon is a workaround for a bug in BeOS R4 sed
    LD_LIBRARY_PATH=`$ECHO "$LD_LIBRARY_PATH" | /bin/sed 's/::*$//'`

    export LD_LIBRARY_PATH

    if test "$libtool_execute_magic" != "%%%MAGIC variable%%%"; then
      # Run the actual program with our arguments.
      func_exec_program ${1+"$@"}
    fi
  else
    # The program doesn't exist.
    $ECHO "$0: error: '$progdir/$program' does not exist" 1>&2
    $ECHO "This script is just a wrapper for $program." 1>&2
    $ECHO "See the libtool documentation for more information." 1>&2
    exit 1
  fi
fi
```

9.13 stdlib.wl

```
/* Get the average of all numbers in an array.
   Input an array of numbers. */

avg arg : Arr -> Num
```

```
  count = 0
  total = 0
  foreach x in arg
    total = total + arg.[count]
    count = count + 1
  result = (total/count)
  result


/* Concatenate two arrays. Input an array of
   two arrays. */

arrconcat arg : Arr -> Arr
  g = arg.[0]
  q = arg.[1]
  combo = []
  foreach x in g
    combo = push [combo, x]
  foreach y in q
    combo = push [combo, y]
  combo


/* Get the gcd of two numbers. Input is an
   array of two numbers. Not implemented yet.*/

gcd arg : Arr -> Num
  j = arg.[0]
  k = arg.[1]
  max = 0
  final = 0
  if (j < k)
    max = (k + 1)
  else
    max = (j + 1)

  arr = fixedArr max
  foreach i in arr
    if (k%i == 0)
      if (j%i == 0)
        final = i
      else
        final = final
    else
      final = final

  log final
  final


/* This function is used to create an array of
   a particular size. Most practical use case is
   for turning foreach into more of the python
   "for i in range x" by creating a dummy array of
   size x.
*/
fixedArr arg : Num -> Arr
  num = arg
  arr = []
  pass = []
  pass = push [pass, arr]
  pass = push [pass, num]
  pass = push [pass, 0]
  final = []

  if (num == 0)
    final = arr
  else
    final = createArrRec pass

  final


createArrRec arg : Arr -> Arr
  arr = arg.[0]
  num = arg.[1]
  count = arg.[2]
  ret = []

  if ( num == 0 )
    ret = arr
  else
    arr = push [arr, count]
    num = (num - 1)
    count = (count + 1)
    ret = createArrRec [arr, num, count]

  ret


/* Checks if an array contains a string or number. Takes
   in an array with two elements: the array to search and
   the string/number to search for. Returns a bool. */

contains arg : Arr -> Bool
  arr = arg.[0]
  focus = arg.[1]
  final = false
  arr = sort arr

  foreach i in arr
    if (isNum focus)
      if (isNum i)
        if (i == focus)
          final = true
        else
          final = final
      else
```

```
        final = final
    else
      final = final

    if (isString focus)
      if (isString i)
        if (equals [i, focus])
          final = true
        else
          final = final
      else
        final = final
    else
      final = final
  log final
  final


/* Sorts an array of numbers. Takes in an array of two arrays, both being
   all integers, and returns a sorted version of the array */

sort arr : Arr -> Arr
  length = 0
  temp = 0
  foreach i in arr
    length = (length + 1)
  ref = fixedArr length

  foreach i in ref
    inner = fixedArr (length - i - 1)
    foreach j in inner
      if (arr.[j] > arr.[j+1])
        temp = arr.[j]
        arr = update [arr, arr.[j + 1], j]
        arr = update [arr, temp, j + 1]
      else
        arr = arr
  arr
```

Bibliography

[1] Debow, B. Why you need to think about APIs before Websites (https://www.wired.com/insights/2014/11/apis-before-websites/)

[2] IFTTT helps you do more with the services you love. Connect Amazon Alexa, Facebook, Twitter, Instagram, Fitbit, Slack, Skype, and hundreds more. ifttt.com

[3] Zapier makes it easy to automate tasks between web apps. Zapier.com

[4] Automate tasks by integrating your favorite apps with Microsoft Flow. Make repetitive tasks easy with workflow automation. flow.microsoft.com.