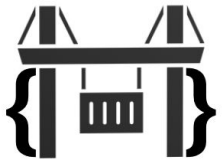
A JSON Data Processing Language

Audrey Copeland, Walter Meyer, Taimur Samee, Rizwan Syed

Introduction

- Language design centered around the programmatic manipulation of JSON data and interacting with HTTP JSON APIs
- C-like syntax and semantics
- Weakly-typed (dynamically-typed) with *most* typing determined at compile-time
- Statically-scoped
- JSON-like data types that are statically typed
- String Library for outputting Objects and Arrays to JSON-encoded Strings
- HTTP Library

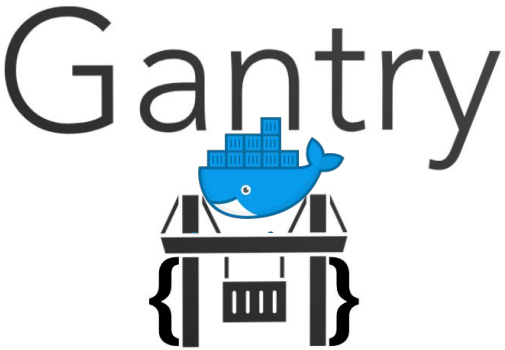
Gantry



Language Idea and Features

What are some of the features provided?

- Nested, Polymorphic, Dynamic Objects
- Static Arrays
- Interaction with the Web APIs (using HTTP Library)
- String Manipulation and Conversion
- Had idea to write a language suitable for systems orchestration (Docker in particular)
- From there realized we could make more generalized language suitable for interacting with web-based JSON APIs.
- “Also, we just preferred the idea of it being statically-typed.”



Language Features

Where did we fall short?

- Objectify and Arrify: Converting/parsing JSON to objects, arrays, and primitives in our language
- Multidimensional and Dynamic Arrays

Data Types

What data types are there?

- Ints
- Floats
- Strings
- Booleans
- Arrays
- Objects

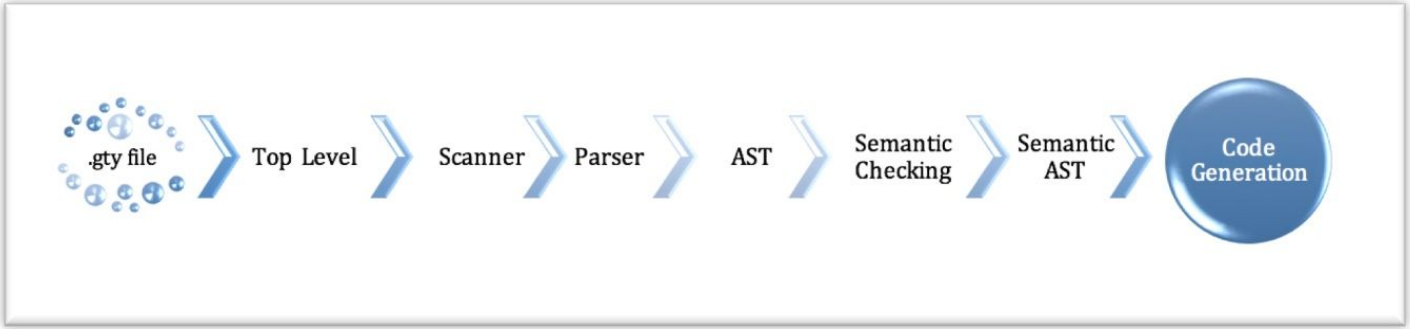
```
int a = 4;
float b = 3.14;
string c = “Gantry!”;
bool d = true;
int array e = [ 4 , 3 , 2 ];

object f = {|
        string s : “foo”,
        int m : 3,
        bool n : true,
        object o = {| int m : 42 |},
        string array o : [“foo”, “bar”]
    |};

object array g = [{| string x: “foo” |}];
```

Control Flow

- if/else
- for
- while
- return

```
int main(){
        int i = 10;
        bool v = true;
        for (i = 0; i < 4; i++){
                if (i == 2){
                        print_i(i);
                }
        }
        bool x = false;
        while(x == false){
                i++;
                if (i > 5 && v == true){
                        x = true;
                }
                else{
                        print_b(x);
                }
        }
        return 0;
}
/*  Output:
 *  2
 */ false
```

System Architecture



Development

Languages

- OCaml
- LLVM
- C
- BASH

Development Environment and Tools

- Google Compute Engine
- Git (GitHub)
- Travis CI
- Ubuntu
- Slack

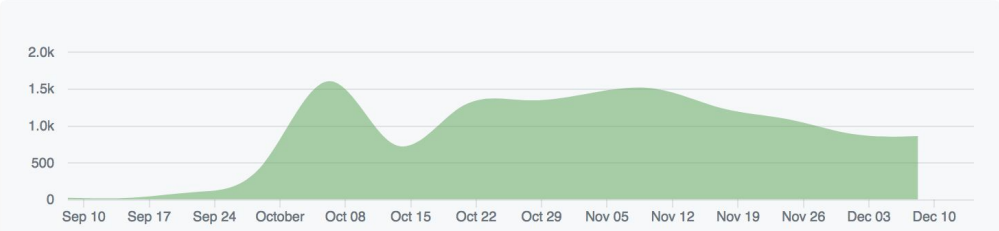
Code Contributions

- We stayed on schedule for the most part

Sep 10, 2017 – Dec 16, 2017

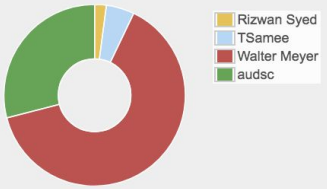
Contributions: **Additions**

Contributions to master, excluding merge commits



The following historical commit information, by author, was found.

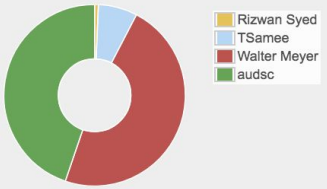
Author	Commits	Insertions	Deletions	% of changes
audsc	99	2529	924	29.01
Rizwan Syed	8	164	83	2.08
TSamee	26	414	183	5.02
Walter Meyer	92	4355	3251	63.90



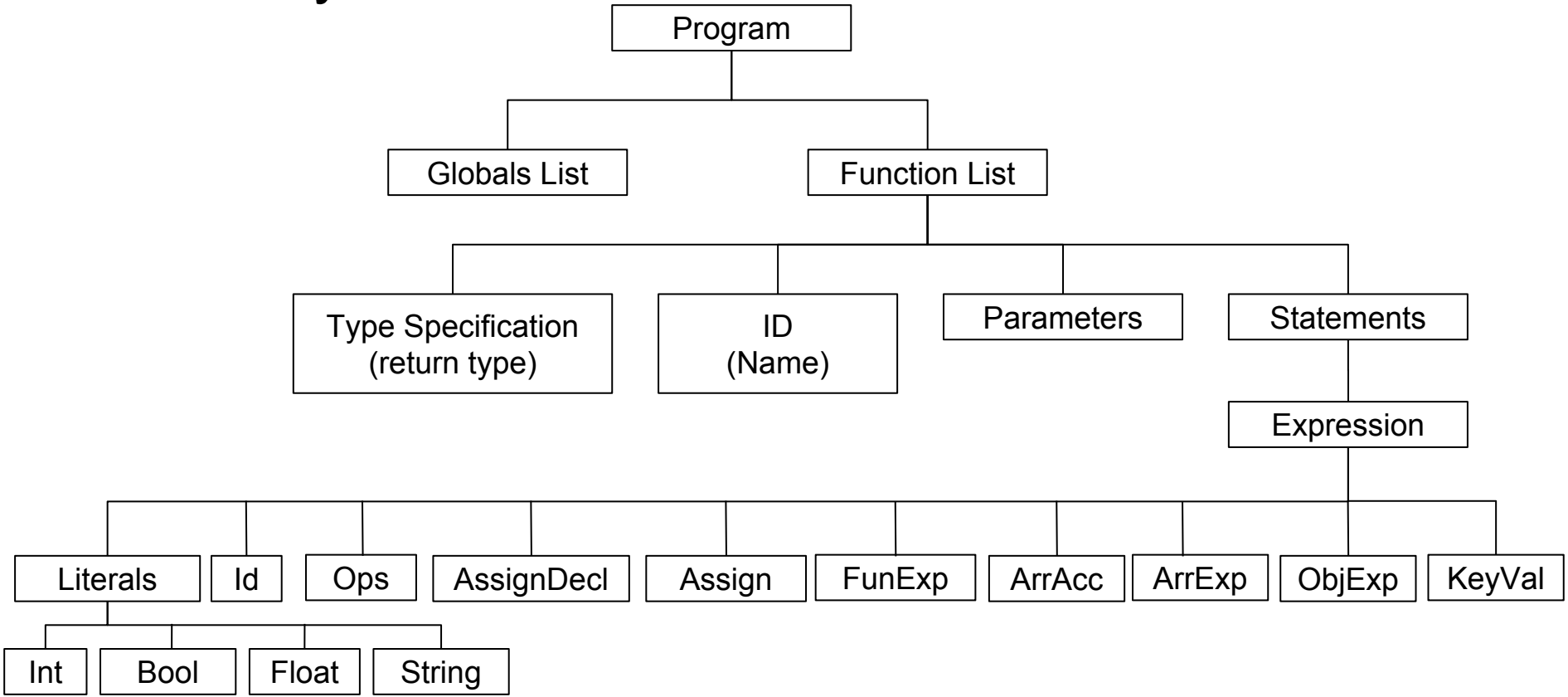
Below are the number of rows from each author that have survived and are still intact in the current revision.

Author	Rows	Stability	Age	% in comments
audsc	1272	50.3	3.5	7.86
Rizwan Syed	19	11.6	6.9	0.00
TSamee	198	47.8	4.0	8.59
Walter Meyer	1352	31.0	4.1	15.16

Hide minor authors (1)



Abstract Syntax Tree



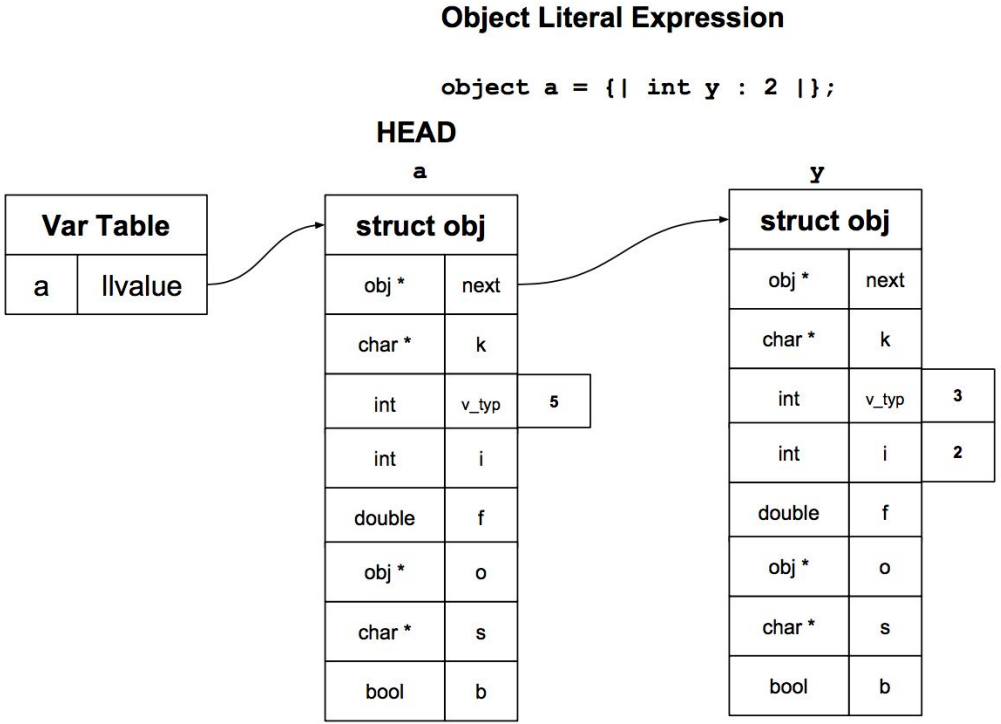
Testing

- Avoided Unit Tests after “Hello World”
- Integration Testing
  - Wrote tests with better code coverage as we built new features
- Test Automation
  - Run all tests at once using test_all script from MicroC
  - Added lines to automatically generate correct output files by using our test format
  - Travis CI
  - All tests must pass before merging into master

```
int main() {
  print_s(“Hello World”);
  return 0;
}

/*
****TEST****
Hello World!
*/
```

Objects and Arrays

- Objects are dynamic, polymorphic, infinitely-nested key-value data types
- Arrays are static, singly-dimensioned, and singly-typed



Objects and Arrays

- Objects are dynamic, polymorphic, infinitely-nested key-value data types
- Arrays are static, singly-dimensioned, and singly-typed

```
/* The Object Struct */
typedef struct obj {
  struct obj *next;
  char *k;
  int  v_typ;

  int i;
  double f;
  struct obj *o;
  char *s;
  bool b;
  struct arr_int *i_a;
  struct arr_flt *f_a;
  struct arr_str *s_a;
  struct arr_bool *b_a;
} obj;
```

```
/* Int Array Struct */
typedef struct arr_int {
  int len;
  int typ;
  int *i_a;
} arr_int;
```

Object Polymorphism and Runtime

- Objects are dynamic, polymorphic, infinitely-nested key-value data structures
- Operations are realized via a small C Object runtime that handles casting data into and out of Struct fields
- Codegen (LLVM) handles casting on return from this library when necessary

```
object f = {|
        string s : “foo”,
        object o = {| int m : 42 |},
    |}

string s = f.s; // runtime handles
f.o.m = “PLT”;  // runtime handles
int x = f.s;    // runtime error
```

Object and Array Stringify

- Produce JSON-encoded Strings
- Object Stringify uses recursion to stringify nested objects
- Array Stringify accepts a void pointer (cast in Codegen) to an array structure

```
int main() {
  object o = {|
           string name: “Joe”,
           int age: 3,
           string array pets: [“Cat”, “Dog”, “Pig”]
           |};
  string to_string = obj_stringify(o);
  print_s(to_string);
}

{ "pets" : [ "Cat" , "Dog" , "Pig" ] , "age" : 3 , "name" : "Joe" }
```

String Library

- Get String, String to Integer, Slice, String Comparison, String Equals, String Concatenation

```
string foo = “foo”;
string bar = “bar”;
string foobar = foo ^ bar;
string f = slice(foobar, 0, 2);
if (foo != bar) {
    print_s(“different strings”);
}
```

HTTP Library

- Support for HTTP1.1 GET and POST methods
- Methods take in target URL, messy stuff under the hood
- Deals purely in strings, relying on other libraries and methods
- Written in C using libcurl


```
string url = "http://192.168.0.9:32000";
string uri = "/v1.19/containers/create";
string response = httpget(url^uri);

object post = {| string image: "centos",
                 string cmd : "[echo,
hi]"
              |};

httppost(url ^ uri, obj_stringify(post));
```

Demo Part I: Blackjack via JSON API

https://deckofcardsapi.com/

Demo Part II: Orchestrating Linux Containers