Gantry Proposal

Audrey Copeland (asc2182), Walter Meyer (wgm2110),
Taimur Samee (ts2903), Rizwan Syed (rms2241)

September 25, 2017

1. Introduction

The Gantry Language is designed to make algorithmic processing of JSON data simpler. Gantry will allow for the programmatic manipulation of JSON data by implementing C-like syntax and semantics along with JSON-like[1] data types and structures.

1.1 Implementation

Gantry will be designed to interface with the Docker container runtime via its API, which uses JSON as a data-exchange format.

1.2 Use Cases

Gantry's JSON-centric features will make it easy to develop programs that interact with a multitude of JSON-based APIs. Specifically, Gantry can be used to facilitate the orchestration of Linux Containers using the Docker[2] JSON API. Containers are a form of Operating System Virtualization typically used to deploy applications (Web Servers, Database Servers, etc.) into lightweight, isolated, and immutable objects. For example, a Gantry program will be able to create, destroy, start and stop Containers using the Docker JSON API based on real-time Container properties returned from the Docker API.

2. Lexical Conventions

2.1 Comments

Comments are lines beginning with two forward slashes, or blocks beginning with /* and ending with */

```
// This is a comment
```

[1]http://www.ietf.org/rfc/rfc4627.txt
[2]https://www.docker.com/what-docker

```
/*
This is a comment
in block format
*/
```

2.2 Identifiers

An identifier (a variable) is a sequence of alphanumeric characters that cannot begin with a number.

2.3 Keywords

func	int	float
string	char	arr
true	false	null
if	elif	else
for	while	object
continue	break	return

2.4 Operators

Operator	Syntax	Operands
Arithmetic	a + b, a - b, a * b, a / b	int, float
Assignment	a = b	int, float, bool, char, string
Equal	a == b	int, float, bool, char, string
Not Equal	a != b	int, float, bool, char, string
Comparison	a <= b, a <b , a >= b, a >b	int, float
Logical AND	a && b	bool
Logical OR	a||b	bool
Logical NOT	!a	bool
Concatenation	aˆb	string

2.5 Built-In Functions

Gantry includes built-in functions to handle some fundamental operations that are useful for parsing JSON-formatted data.

2.5.1 jsonify()

The jsonify() function will take an object as a parameter and will convert the object into a

JSON-formatted string. e.g.:

Listing 1: jsonify()

```
string course = "PLT";
int students = 125;
string location = "NWC";
object course_obj;
string course_obj.course = course;
int course_obj.students = students;
string course_obj.location = location;
string course_str = jsonify(course_obj);
print(course_str);
// prints {course="PLT",students=125,location="NWC"}
```

2.5.2 objectify()

The objectify() function will take a string as a parameter and will attempt to produce a representation of that JSON-formatted string as an object with its nested component data types. e.g:

Listing 2: objectify()

```
string str = "{course=\"PLT\",students=125,location=\"NWC\"}";
object course_obj = objectify(str);
string course_name = course_obj.course;
int course_enrollment = course_obj.students;
string course_location = course_obj.location;
print(course_name);
// prints "PLT"
```

2.5.3 arrify()

The arrify() function will take a string as a parameter and will attempt to produce a representation of that JSON-formatted string as an array. e.g:

Listing 3: arrify()

```
string str = "[{course=\"PLT\",students=125,location=\"NWC\"},
{course=\"CS Theory\",students=200,location=\"NWC\"}]";
arr courses_arr = arrify(str);
object first_course = courses_arr[0];
object second_course = courses_arr[1];
string first_course_name = first_course.course;
string second_course_name = second_course.course;
string output_string = first_course_name ^ " and " ^ second_course_name;
```

```
print(output_string);
// prints "PLT and CS Theory"
```

2.5.4 length()

The length() function will take an array as a parameter and will return the number of elements in the array. This function can also take a string as a parameter since we implement a string as an array of characters.

Listing 4: length()

```
arr student_arr = ["Joe", "Bob", "Alan"];
int arr_length = length(student_arr);
print(arr_length);
// prints 3
```

2.5.5 print()

The print() function will take a parameter of any type defined in our language and print its string representation. *Print* requires text to be bound by "" and can be concatenated with strings or integers using the hat symbol.

Listing 5: print()

```
string course_name = "PLT";
print("This is the course name: "^ course_name);
// prints "This is the course name : PLT"
```

2.5.6 to_string()

The to_string() function will take a parameter of any type defined in our language and return it as a string. This method will be called by print.

Listing 6: to_string()

```
int course_enrollment = 3;
string course_enrollment_string = to_string(course_enrollment);
```

2.5.7 http_get()

The http_get() function will take a server and port as a parameter along with a URI to perform an HTTP GET request to.

Listing 7: http_get()

```
/*
  Returns a json object of containers running on
  a particular Docker engine.
*/
string uri = "/v1.19/containers/json"
string cons = http_get("192.168.0.9", 80, uri);
object cons_arr = arrify(cons);
print(cons_arr);
```

2.5.8 http_post()

The http_post() function will take a server, port, URI, and POST data as parameters to form an HTTP POST request.

Listing 8: http_post()

```
/*
  Returns a json object of a newly created container
  running on a particular Docker engine.
*/
string post_data = "{"Image": "centos", "Cmd": ["echo", "hello world"]}"
string uri = "/v1.19/containers/create"
string con = http_post("192.168.0.9", 80, uri, post_data);
object con_obj = objectify(con);
print(con_obj);
```

2.5.9 sort()

The sort() function will take an array as an argument, and will by default sort it in ascending order. The function will optionally accept sort order (asc or dsc) and a key value to sort on. For example, 'sort_key=Size' would search inside the objects in the array and sort based on the values associated with the 'Size' key. If no 'Size' key exists in one or more of the objects present in the array, the sort function will fail to sort the array.

Listing 9: sort()

```
/*
  Returns a sorted list.
*/
```

```
arr my_number_list = [3, 9, 2, 1];
arr my_sorted_number_list = sort(my_number_list, asc);
print(my_sorted_number_list);
// prints '[1, 2, 3, 9]'
```

2.6 Constants

2.6.1 Integers

Integers are a sequence of numeric characters [0-9] in decimal notation. They are 32-bit signed integers in the range of -2,147,483,648 to 2,147,483,647.

2.6.2 Floats

Floats represent real numbers with integer and decimal parts separated by a decimal point. They are 32-bit unsigned values in the range $1.3 * 10^{-38}$ to $3.4 * 10^{38}$, with precision up to 6 decimal places.

2.6.3 Characters

Characters are standard ASCII characters with associated integer values from 0 to 127.

2.6.3.1 Character Constants

Character constants have special meaning and specific functionality. In Gantry, the following character constants are defined:

- '\n' will be used for line break.
- '\' will be used as an escape character.

2.6.4 Strings

Strings are immutable null-terminated arrays of characters.

2.7 Control Flow

Gantry supports basic control flow features, including if-else, for loops, and while loops.

2.7.1 If-Else

Listing 10: If-Else

```
int number = 10;
if(number > 5) {
    print("Number is greater than 5");
} else {
    print("Number is less than 5");
}

// prints "Number is greater than 5"
```

2.7.2 *Loops*

2.7.2.1 *For Loop*

Listing 11: For Loop

```
int number = 10;
for(int example = 1; example < 6; example++)  {
    number = number + 1;
}
print(number);

// prints 15
```

2.7.2.2 *While Loop*

Listing 12: While Loop

```
int example = 1;
int number = 10;
while(example < 6) {
    number = number + 1;
    example = example + 1;
}
print(number);

// prints 15
```

2.8 *Composite Types*

2.8.1 *Arrays*

Arrays are an ordered list of values of a single type. The elements of an array can be indexed using bracket notation.

Listing 13: Arrays

```
arr nums = [101, 346, 1032, 2, 25];
int i = nums[0];
print(i); // prints 101

arr strs = ["hello", "world"];
string str = strs[1];
print(str); // prints "world"
```

2.8.2 Objects

Objects are mutable collections of key value pairs. Values of an object can be accessed and modified using dot notation.

Listing 14: Sample Code

```
object class = {
    name:   "PLT",
    students: "125"
};

print(class.name); // prints PLT

class.students = 100; // change # of students
print(class.students); // prints 100
```

3. Sample Code

3.1 Function to query the Docker API for running Containers

Listing 15: Sample Code

```
object host = {
    ip:   "192.168.1.3",
    port: "443",
    ver:  "v1.19"
};

arr func listCont(object host) {
    string uri = "/" ^ host.ver ^ "/containers/json";
    arr cons = http_get(host.ip, host.port, uri);
    return cons;
}
```

3.2 Connect to the Docker Engine and Get Container Information

The following example program establishes a connection with a Docker engine on a running host at http://192.168.1.3/. First, the program gets the number of containers running on the engine and sorts them based on particular container object keys. Then it prints the container names in sorted orders. If there are no containers running on the engine it reports that instead.

Listing 16: Sample Code

```
object myhost = {
    ip:   "192.168.1.3",
    port: "443",
    ver:  "v1.19"
};

arr container_list = listCont(myhost);
if (length(container_list) > 0) {

    // sort container list by given key and order
    arr sorted_list = sort(container_list, asc, sort_key="size");
    arr sorted_list2 = sort(container_list, dsc, sort_key="name");

    // prints names of containers from smallest to largest size
    for(int i = 0 ; i < length(sorted_list); i++){
        print(sorted_list[i].name);
    }
}
else {
    print("There are no containers running on this engine.");
}
```