# Improved demo on testing functions

Jiahao Cai

Oct, 8, 2018

## 0. Overview

Now I have annotated 4 functions in cJSON, they are:

- parse_string: https://github.com/DaveGamble/cJSON/blob/master/cJSON.c#L698
- parse_object: https://github.com/DaveGamble/cJSON/blob/master/cJSON.c#L1497
- parse_array: https://github.com/DaveGamble/cJSON/blob/master/cJSON.c#L1341
- parse_number: https://github.com/DaveGamble/cJSON/blob/master/cJSON.c#L266

Also, I write some test cases, and a Perl script to do test, log generation and simple log processing automatically.

## 1. Annotate the function

Here is an example of how I annotated it:

```c
static FILE *log_file;
static const char log_dir[] = "./temp/";
static char path[100] = "./temp/";
#define LOG(msg) \
    do {\
        strcat(path, msg);\
        log_file = fopen(path, "w");\
        memset(path + sizeof(log_dir) - 1, 0, sizeof(path) - sizeof(log_dir));\
        fclose(log_file);\
    } while(0)
/* Build an array from input text. */
static cJSON_bool parse_array(cJSON * const item, parse_buffer * const input_buffer)
{
    ...
    if (input_buffer->depth >= CJSON_NESTING_LIMIT)
    {
        LOG("parse_array: return false");
        return false; /* to deeply nested */
    }
    if (buffer_at_offset(input_buffer)[0] != '[')
    {
        LOG("parse_array: != \'[\'");
        goto fail; /* not an array */
    }
    LOG("parse_array: == \'[\'");
    ...
    if (can_access_at_index(input_buffer, 0) && (buffer_at_offset(input_buffer)[0] == ']'))
    {
        LOG("parse_array: == \']\'");
        /* empty array */
        goto success;
    }
    LOG("parse_array: != \']\'");
    while (can_access_at_index(input_buffer, 0) && (buffer_at_offset(input_buffer)[0] == ','

    if (cannot_access_at_index(input_buffer, 0) || buffer_at_offset(input_buffer)[0] != ']'
    {
        LOG("parse_array: != \']\'");
        goto fail; /* expected end of array */
    }
    LOG("parse_array: == \']\'");

success:
    LOG("parse_array: return true");
    return true;

fail:
    LOG("parse_array: return false");
    return false;
}
```

## 2. Test cases

I have come up with some test cases, below are the tests I used in the script

```
my @parse_string_test =
('{"normal": "whatever"}',
'{"empty": ""}',
'{"not correctly quoted: "whatever"}',
'{"escapes": "\n\r\t\b\f\"\\\\\/"}', # By now, most predicates of the function has been cove
'{"numerical": "123.456"}', # Some random strings
'{"I": "say,"}',
'{"\"Perl": "is"}',
'{"very": "convenient"}',
'{"in": "such"}',
'{"scenario": "right?\""}',
'{"Yeah!": "It is tota11y: r$ght"}'
);
my @parse_object_test =
('{"normal": "obj"}',
'{"nested": "John", "outside": {"inside": "Bob"}}',
'{"deeper nested": {"first": {"second": {"third": {"fourth": {"fifth": "treasure"}}}}}}"',
'{"numerical": 1234, {"inside": 5678}}',
'{"num_array": [1, 2, 3, 4], "str_array": ["Monday", "Tuesday", "Wednesday",
"Thursday", "Friday"]}'
);
my @parse_array_test =
('{"simple_num_arr": [1, 2, 3, 4]}',
'{"simple_str_arr": ["1", "2", "3", "4"]}',
'{"simple_mixed_arr": [1, "2", 3, "4"]}',
'{"num_arrs": [1, 2, 3, 4], "arr1": [1234, 5678], "arr2": [7493, 0912]}',
'{"str_arrs": ["1", "2", "3"], "arr1": ["123", "456"], "arr2": ["abc", "def"]}',
'{"mixed_arrs": [1, "2", 3, "4"], "arr1": [42, "is", "the", "answer],
"arr2": [13, "is", "unlucky"]}',
'{"nested_mixed_arrs": ["outside", ["middle": ["inside": "yo", "heart": "broken"],
"skeleton": [13, "not cool"]',
'{"nested_mixed_arrs": ["outside", ["middle", [{"inside": "yo"}, {"heart": "broken"}]],
"skeleton", [13, "not cool"]]}',
'{"empty: [[],[],[],[[],[],[]]]"}',
'{"invalid": [1234]}',
);
my @parse_number_test =
('{"normal": 123456789, "another": 434227}',
'{"scientific": 1e5, "an0ther": 1E5, "plus": 1E+5, "minus": 1e-5}',
'{"decimal": [1.1, 2.4e100, 9.3244921234323232323, 5.396E-432]}',
);
```

## 3. Generated logs after simple processed.

This is a sample log of processing `{"normal": "obj"}`, with my comments:

```
open("./temp/*******", O_WRONLY|O_CREAT|O_TRUNC, 0666) = 3
open("./temp/parse_object: start", O_WRONLY|O_CREAT|O_TRUNC, 0666) = 3
# a valid JSON object should start with {
open("./temp/parse_object: == '{'", O_WRONLY|O_CREAT|O_TRUNC, 0666) = 3
# if it is an empty object?
open("./temp/parse_object: != '}'", O_WRONLY|O_CREAT|O_TRUNC, 0666) = 3
# it's a string, *first overestimate its length*
open("./temp/parse_string: start", O_WRONLY|O_CREAT|O_TRUNC, 0666) = 3
# first character should be "
open("./temp/parse_string: == '\"'", O_WRONLY|O_CREAT|O_TRUNC, 0666) = 3
# if find another ", the string ends here
open("./temp/parse_string: != '\"'", O_WRONLY|O_CREAT|O_TRUNC, 0666) = 3
# if there is an escape character?
open("./temp/parse_string: != '\\'", O_WRONLY|O_CREAT|O_TRUNC, 0666) = 3
# keep looking for the ending " and escape character
open("./temp/parse_string: != '\"'", O_WRONLY|O_CREAT|O_TRUNC, 0666) = 3
open("./temp/parse_string: != '\\'", O_WRONLY|O_CREAT|O_TRUNC, 0666) = 3
open("./temp/parse_string: != '\"'", O_WRONLY|O_CREAT|O_TRUNC, 0666) = 3
open("./temp/parse_string: != '\\'", O_WRONLY|O_CREAT|O_TRUNC, 0666) = 3
open("./temp/parse_string: != '\"'", O_WRONLY|O_CREAT|O_TRUNC, 0666) = 3
open("./temp/parse_string: != '\\'", O_WRONLY|O_CREAT|O_TRUNC, 0666) = 3
open("./temp/parse_string: != '\"'", O_WRONLY|O_CREAT|O_TRUNC, 0666) = 3
open("./temp/parse_string: != '\\'", O_WRONLY|O_CREAT|O_TRUNC, 0666) = 3
open("./temp/parse_string: != '\"'", O_WRONLY|O_CREAT|O_TRUNC, 0666) = 3
open("./temp/parse_string: != '\\'", O_WRONLY|O_CREAT|O_TRUNC, 0666) = 3
# get the ending "
open("./temp/parse_string: == '\"'", O_WRONLY|O_CREAT|O_TRUNC, 0666) = 3
# loop through the string literal, not care about the boundary this time
open("./temp/parse_string: != '\\'", O_WRONLY|O_CREAT|O_TRUNC, 0666) = 3
open("./temp/parse_string: != '\\'", O_WRONLY|O_CREAT|O_TRUNC, 0666) = 3
open("./temp/parse_string: != '\\'", O_WRONLY|O_CREAT|O_TRUNC, 0666) = 3
open("./temp/parse_string: != '\\'", O_WRONLY|O_CREAT|O_TRUNC, 0666) = 3
open("./temp/parse_string: != '\\'", O_WRONLY|O_CREAT|O_TRUNC, 0666) = 3
open("./temp/parse_string: != '\\'", O_WRONLY|O_CREAT|O_TRUNC, 0666) = 3
open("./temp/parse_string: return true", O_WRONLY|O_CREAT|O_TRUNC, 0666) = 3
open("./temp/parse_object: == ':'", O_WRONLY|O_CREAT|O_TRUNC, 0666) = 3
# another string, parse similarly
open("./temp/parse_string: start", O_WRONLY|O_CREAT|O_TRUNC, 0666) = 3
open("./temp/parse_string: == '\"'", O_WRONLY|O_CREAT|O_TRUNC, 0666) = 3
open("./temp/parse_string: != '\"'", O_WRONLY|O_CREAT|O_TRUNC, 0666) = 3
open("./temp/parse_string: != '\\'", O_WRONLY|O_CREAT|O_TRUNC, 0666) = 3
open("./temp/parse_string: != '\"'", O_WRONLY|O_CREAT|O_TRUNC, 0666) = 3
open("./temp/parse_string: != '\\'", O_WRONLY|O_CREAT|O_TRUNC, 0666) = 3
open("./temp/parse_string: != '\"'", O_WRONLY|O_CREAT|O_TRUNC, 0666) = 3
open("./temp/parse_string: != '\\'", O_WRONLY|O_CREAT|O_TRUNC, 0666) = 3
open("./temp/parse_string: == '\"'", O_WRONLY|O_CREAT|O_TRUNC, 0666) = 3
open("./temp/parse_string: != '\\'", O_WRONLY|O_CREAT|O_TRUNC, 0666) = 3
open("./temp/parse_string: != '\\'", O_WRONLY|O_CREAT|O_TRUNC, 0666) = 3
open("./temp/parse_string: != '\\'", O_WRONLY|O_CREAT|O_TRUNC, 0666) = 3
open("./temp/parse_string: return true", O_WRONLY|O_CREAT|O_TRUNC, 0666) = 3
# any more elements?
open("./temp/parse_object: != ','", O_WRONLY|O_CREAT|O_TRUNC, 0666) = 3
open("./temp/parse_object: == '}'", O_WRONLY|O_CREAT|O_TRUNC, 0666) = 3
open("./temp/parse_object: return true", O_WRONLY|O_CREAT|O_TRUNC, 0666) = 3
open("./temp/*******", O_WRONLY|O_CREAT|O_TRUNC, 0666) = 3
fstat(1, {st_mode=S_IFIFO|0600, st_size=0, ...}) = 0
write(1, "{\n\t\"normal\":\t\"obj\"\n}\n", 21{
    "normal":   "obj"
}
) = 21
exit_group(0)                          = ?
+++ exited with 0 +++
```

## Future work

- I think I should come up with some proper test case, then input the *same* test cases to the 4 different functions above, and observe their behaviors.
- Need to do more processing of the logs.
    - If we have discussed and determined the log format, then I think we can start to think about the algorithm to compare the signatures.

What do you think?