Санкт-Петербургский государственный политехнический университет

Институт компьютерных наук и технологий

**Кафедра «Компьютерные системы и программные технологии»**

**КУРСОВОЙ ПРОЕКТ**

**Парсер JSON (DOM)**

по дисциплине «Прикладное программирование»

Выполнил

студент гр.23531/1 И.К. Сергеев

Руководитель М.А.Петров

«\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2018 г.

Санкт-Петербург

2018

Санкт-Петербургский государственный политехнический университет

**ЗАДАНИЕ**

**НА ВЫПолнение курсовоГО ПРОЕКТА**

студенту группы 23531/1 Сергееву Илье Константиновичу

***1. Тема проекта (работы):*** Парсер JSON (DOM)

***2. Срок сдачи законченного проекта (работы)*** г.

***3. Исходные данные к проекту (работе)***:

Разработка кода и документации должна вестись в репозитории git.

Разработанное ПО должно собираться с помощью команды make компилятором gcc с опциями -std=c11 -pedantic -Wall -Wextra без предупреждений компилятора.

***4. Содержание пояснительной записки***: введение, основная часть (раскрывается структура основной части), заключение, список использованных источников, приложения.

***Дата получения задания***: «10» мая 2018 г.

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ М.А.Петров

*(подпись)*

Задание принял к исполнению \_\_\_\_\_\_\_\_\_\_\_\_\_ И.К.Сергеев

*(подпись студента)*

05.04.2018г.

**СОДЕРЖАНИЕ**

|  |  |
| --- | --- |
| Введение. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . …………….. | 4 |
|  |  |
| 1. Код программы, реализующей симулятор……………………................... | 5 |
| Заключение .................................................................................................................. | 13 |
| Список использованных источников  ........................................................................ | 14 |
|  |  |

**ВВЕДЕНИЕ**

В данном курсовом проекте мною написана программа, являющаяся парсером JSON модели DOM, что соответствует моему индивидуальному заданию по прикладному программированию.

В ней будут реализованы основные структуры данных, такие как массивы и объекты. А так же некоторые функции для проверки целостности и правильности полученных данных.

**1. Код программы, реализующей ПАРСЕР**

**Cсылка на гитхаб: https://github.com/tiger31/json\_parser**

**Makefile:**

\_\_\_\_\_\_\_\_\_\_\_\_\_

all : json\_parser

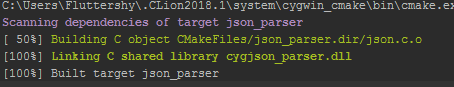
json\_parser : json.o

gcc -pedantic -Wall -Wextra -shared -o json\_parser json.o

json.o : json.c json.h

gcc -std=c11 -pedantic -Wall -Wextra -c json.c

clean : json.o



**json.h**

**typedef enum** \_types {  
 *JSON\_OBJ*,  
 *JSON\_ARR*,  
 *JSON\_STR*,  
 *JSON\_NUM*,  
 *JSON\_BOOL*,  
 *JSON\_NULL*,  
 *JSON\_ERR*} json\_type;  
  
**typedef enum** \_states {  
 *INIT*,  
 *EXP\_KEY*,  
 *EXP\_COLON*,  
 *EXP\_VALUE*,  
 *EXP\_COMMA*,  
 *ENDED*} states;  
**typedef struct** \_json\_value {  
 json\_type type;  
 **void** \* value;  
} json\_value;  
**typedef struct** \_json\_pair {  
 **char** \* key;  
 json\_value \* value;  
} json\_pair;  
**typedef struct** \_json\_object {  
 json\_pair \*\* pairs;  
 size\_t length;  
} json\_object;  
**typedef struct** \_json\_array {  
 json\_value \*\* values;  
 size\_t length;  
} json\_array;  
**int** parse\_object(json\_object \* object, **char** \* string);  
**int** parse\_array(json\_array \* array, **char** \* string);  
  
**void** json\_object\_free(json\_object \* object);  
**void** json\_array\_free(json\_array \* array);

**json.c**

#include <stdio.h>  
#include <string.h>  
#include <regex.h>  
#include <stdlib.h>  
#include "json.h"  
  
**const char** \* num\_reg = "^(-?([1-9][0-9]\*|0)[ ]\*(.[ ]\*[0-9]+)?[ ]\*([eE][ ]\*[+-]?[ ]\*[0-9]+)?)$";  
**const char** \* str\_reg = "^((\\\\[\\\\\"bfnrt/])|(\\\\u[0-9a-fA-F]{4})|([^\"\\\\]))\*$";  
  
**int** check\_number(**char** \* number) {  
 regex\_t regex;  
 regcomp(&regex, num\_reg, REG\_EXTENDED);  
 **int** result = regexec(&regex, number, 0, NULL, 0);  
 regfree(&regex);  
 **return** !result;  
}  
**int** check\_string(**char** \* string) {  
 regex\_t regex;  
 regcomp(&regex, str\_reg, REG\_EXTENDED);  
 **int** result = regexec(&regex, string, 0, NULL, 0);  
 regfree(&regex);  
 **return** !result;  
}  
**int** isspace(**char** c) {  
 **if** (c != ' ' && c != '\t' && c != '\n' && c != '\v'  
 && c != '\f' && c != '\r')  
 **return** 0;  
 **return** 1;  
}  
**int** skip\_spaces(**char** \* string, **int** from) {  
 **for** (**int** i = from; i < strlen(string); ++i) {  
 **if** (!isspace(string[i]))  
 **return** i;  
 }  
 **return** -1;  
}  
**int** pair\_index(**char** \* string, **char** c, **int** pos) {  
 **char** pair;  
 **int** inc\_obj = 0;  
 **int** inc\_arr = 0;  
 **int** in\_quotes = 0;  
 **int** escape\_next = 0;  
 **switch** (c) {  
 **case** 34:  
 pair = 34;  
 in\_quotes = 1;  
 **break**;  
 **case** 91:  
 pair = 93;  
 **break**;  
 **case** 123:  
 pair = 125;  
 **break**;  
 **default**:  
 **return** 0;  
 }  
 **int** i;  
 **for** (i = pos; i < strlen(string); i++) {  
 **switch** (string[i]) {  
 **case** 92:  
 //Следующий символ будет пропущен  
 escape\_next = !escape\_next;  
 **break**;  
 **case** 34:  
 //Если не симовл не пропускается, то обрабатываем  
 **if** (!escape\_next) {  
 //Выходим из кавычек, если в них были и наоборот  
 in\_quotes = !in\_quotes;  
 //Если вышли, и нужно найти кавычки - возвращаем индекс  
 **if** (!in\_quotes && pair == 34) {  
 **return** i;  
 }  
 **break**;  
 } **else** escape\_next = 0;  
 **case** 91:  
 //Если символ не пропускается(вроде такое не возможно, но проверить стоит) и не внутри строки  
 **if** (!escape\_next && !in\_quotes)  
 inc\_arr++; //Инкапсуляция внутрь массива  
 **break**;  
 **case** 93:  
 **if** (!escape\_next && !in\_quotes) {  
 //Если нет инкапсуляции (тот же уровень) - возвращаем индекс  
 **if** (inc\_arr == 0 && inc\_obj == 0)  
 **return** i;  
 inc\_arr--;  
 }  
 **break**;  
 **case** 123:  
 **if** (!escape\_next && !in\_quotes)  
 inc\_obj++;  
 **break**;  
 **case** 125:  
 **if** (!escape\_next && !in\_quotes) {  
 **if** (inc\_arr == 0 && inc\_obj == 0)  
 **return** i;  
 inc\_obj--;  
 }  
 **break**;  
 **default**:  
 **if** (escape\_next)  
 escape\_next = 0;  
 **break**;  
 }  
 }  
 **return** -1;  
}  
**int** num\_index(**char** \* string, **int** pos) {  
 size\_t len = strlen(string);  
 **int** i = 0;  
 **int** space\_resist = 0;  
 **int** space\_index = 0;  
 **for** (i = pos; i < len; i++) {  
 **if** (isspace(string[i])) {  
 **if** (!space\_resist)  
 space\_index = i;  
 space\_resist = 1;  
 **continue**;  
 }  
 **if** (string[i] == 44 || string[i] == 93 || string[i] == 125) //"," может не быть, будет "]" или "}", потратил часа пол чтобы найти этот баг  
 **return** ((space\_resist) ? space\_index : i);  
 space\_resist = 0;  
 }  
 **return** -1;  
}  
**char** \* strpaired(**char** \* string, **int** from, **int** to, **int** include\_c) {  
 size\_t len = (size\_t)to - (from + ((include\_c) ? -1 : 1));  
 **char** \* str = malloc((len + 1) \* **sizeof**(**char**));  
 strncpy(str, string + (from + 1 - include\_c), len);  
 str[len] = '\0';  
 **return** str;  
}  
**char** \* get\_number(**char** \* string, **int** from, **int** to) {  
 size\_t len = (size\_t)to - from;  
 **char** \* str = malloc((len + 1) \* **sizeof**(**char**));  
 strncpy(str, string + from, len);  
 str[len] = '\0';  
 **return** str;  
}  
json\_type estimate\_type(**const char** \* string, **int** pos) {  
 **const char** c = string[pos];  
 **if** (c == 34) // === '"'  
 **return** JSON\_STR;  
 **else if** (c == 91) // === '['  
 **return** JSON\_ARR;  
 **else if** (c == 123) // === '{'  
 **return** JSON\_OBJ;  
 **else if** (c == 45 || (c > 47 && c < 58)) // === '-' или [0;9]  
 **return** JSON\_NUM;  
 **else if** (c == 110) // === 'n'  
 **return** JSON\_NULL;  
 **else if** (c == 102 || c == 116) // === 'f' или 't'  
 **return** JSON\_BOOL;  
 **else  
 return** JSON\_ERR;  
}  
**int** parse\_value(json\_value \* value, **char** \* string, **int** pos) {  
 value->type = estimate\_type(string, pos);  
 **switch** (value->type) {  
 **case** JSON\_OBJ: { //Без скобок запрещает объявлять переменную, так как не в начале блока  
 //Копируем кусок в новую строку  
 **int** index = pair\_index(string, 123, pos + 1);  
 **char** \* obj\_str = strpaired(string, pos, index, 1);  
 json\_object \* obj = (json\_object \*)malloc(**sizeof**(json\_object));  
 **if** (parse\_object(obj, obj\_str))  
 **return** -1;  
 value->value = obj;  
 free(obj\_str);  
 **return** index;  
 }  
 **case** JSON\_ARR: {  
 **int** index = pair\_index(string, 91, pos + 1);  
 **char** \* arr\_str = strpaired(string, pos, index, 1);  
 json\_array \* arr = (json\_array \*)malloc(**sizeof**(json\_array));  
 **if** (parse\_array(arr, arr\_str))  
 **return** -1;  
 value->value = arr;  
 free(arr\_str);  
 **return** index;  
 }  
 **case** JSON\_STR: {  
 **int** index = pair\_index(string, 34, pos + 1);  
 **char** \* str = strpaired(string, pos, index, 0);  
 **if** (!check\_string(str)) {  
 free(str);  
 **return** -1;  
 }  
 value->value = str;  
 **return** index;  
 }  
 **case** JSON\_NUM: {  
 **int** index = num\_index(string, pos);  
 **char** \* num = get\_number(string, pos, index);  
 **if** (!check\_number(num)) {  
 free(num);  
 **return** -1;  
 }  
 value->value = num;  
 **return** index - 1;  
 }  
 **case** JSON\_BOOL: {  
 **char** \* true = "true";  
 **char** \* false = "false";  
 **char** \* seq;  
 **if** (string[pos] == 102)  
 seq = false;  
 **else** seq = true;  
 **int** i;  
 **for** (i = 0; i < strlen(seq); i++)  
 **if** (string[pos + i] != seq[i])  
 **return** -1;  
 //Не теряем локальную переменную из памяти  
 **int** \* result = malloc(**sizeof**(**int**));  
 \*result = ((seq == true) ? 1 : 0);  
 value->value = result;  
 **return** pos + i - 1;  
 }  
 **case** JSON\_NULL: {  
 **char** \* null = "null";  
 **int** i = 0;  
 **for** (i = 0; i < 4; i++)  
 **if** (string[pos + i] != null[i])  
 **return** -1;  
 //value оставим null-pointer`ом  
 **return** pos + i - 1;  
 }  
 **case** JSON\_ERR:  
 **return** -1;  
 }  
}  
**int** parse\_object(json\_object \* object, **char** \* string) {  
 states state = INIT;  
 size\_t length = strlen(string);  
 json\_pair \*\* pairs = malloc(**sizeof**(json\_pair\*));  
 **int** members = 0;  
 **int** i = 0;  
 **for** (i; i < length; ++i) {  
 **switch** (state) {  
 //Стадия инициализации объекта - поиск {  
 **case** INIT:  
 **if** (string[i] != 123)  
 **return** 1;  
 state = EXP\_KEY;  
 **break**;  
 **case** EXP\_KEY:  
 i = skip\_spaces(string, i); //Пропускаем все пробелы до следующего ключа  
 **if** (string[i] == 125) { //Может быть не ключ, а "}", что тоже верно  
 state = ENDED;  
 } **else if** (string[i] == 34) { //Ключ всегда начинается с ", так как ключом может быть только строка  
 **int** index = pair\_index(string, 34, i + 1);  
 **char** \* key = strpaired(string, i, index, 0);  
 pairs = (json\_pair \*\*)realloc(pairs, ++members \* **sizeof**(json\_pair\*)); //Увеличиваем массив ключей на один  
 json\_pair \* p = malloc(**sizeof**(json\_pair));  
 pairs[members - 1] = p;  
 pairs[members - 1]->key = key;  
 state = EXP\_COLON;  
 i = index;  
 }  
 **break**;  
 **case** EXP\_COLON:  
 i = skip\_spaces(string, i);  
 **if** (string[i] != 58)  
 **return** 1;  
 i = skip\_spaces(string, ++i);  
 i--;  
 state = EXP\_VALUE;  
 **break**;  
 **case** EXP\_VALUE: {  
 json\_value \* value = (json\_value \*)malloc(**sizeof**(json\_value));  
 **int** result = parse\_value(value, string, i);  
 **if** (result == -1)  
 **return** 1;  
 i = result;  
 pairs[members - 1]->value = value;  
 state = EXP\_COMMA;  
 **break**;  
 }  
 **case** EXP\_COMMA:  
 i = skip\_spaces(string, i); //Пропускаем все пробелы до следующего ключа  
 **if** (string[i] == 125) { //Может быть не ключ, а "}", что тоже верно  
 state = ENDED;  
 } **else if** (string[i] == 44) {  
 state = EXP\_KEY;  
 } **else** {  
 **return** 1;  
 }  
 **break**;  
 **case** ENDED:  
 **return** 1;  
 }  
 }  
 object->pairs = pairs;  
 object->length = (size\_t)members;  
 **return** 0;  
}  
**int** parse\_array(json\_array \* array, **char** \* string) {  
 states state = INIT;  
 size\_t length = strlen(string);  
 json\_value \*\* values = '\0';  
 **int** members = 0;  
 **int** i = 0;  
 **for** (i = 0; i < length; i++) {  
 **switch** (state) {  
 **case** INIT:  
 **if** (string[i] != 91)  
 **return** 1;  
 state = EXP\_VALUE;  
 **break**;  
 **case** EXP\_VALUE: {  
 i = skip\_spaces(string, i); //Пропускаем все пробелы до следующего значения  
 **if** (string[i] == 93) { //Может быть не значение, а "]", что тоже верно  
 state = ENDED;  
 **break**;  
 }  
 json\_value \* value = malloc(**sizeof**(json\_value));  
 **int** result = parse\_value(value, string, i);  
 **if** (result == -1)  
 **return** 1;  
 i = result;  
 values = (json\_value \*\*)realloc(values, ++members \* **sizeof**(json\_value\*));  
 values[members - 1] = value;  
 state = EXP\_COMMA;  
 **break**;  
 }  
 **case** EXP\_COMMA:  
 i = skip\_spaces(string, i);  
 **if** (string[i] == 93) { //Может быть не ключ, а "}", что тоже верно  
 state = *ENDED*;  
 } **else** {  
 state = *EXP\_VALUE*;  
 }  
 **break**;  
 **default**:  
 **return** 1;  
 }  
 }  
 array->values = values;  
 array->length = (size\_t)members;  
 **return** 0;  
}  
  
**void** json\_object\_free(json\_object \* object) {  
 **for** (**int** i = 0; i < object->length; i++) {  
 json\_pair \* p = object->pairs[i];  
 **switch** (p->value->type) {  
 **case** JSON\_OBJ:  
 json\_object\_free(p->value->value);  
 free(p->value->value);  
 free(p->value);  
 **break**;  
 **case** JSON\_ARR:  
 json\_array\_free(p->value->value);  
 free(p->value->value);  
 free(p->value);  
 **break**;  
 **case** JSON\_STR:  
 **case** JSON\_BOOL:  
 **case** JSON\_NUM:  
 free(p->value->value);  
 free(p->value);  
 **break**;  
 **default**:  
 **break**;  
 }  
 free(p->key);  
 free(p);  
 }  
 free(object->pairs);  
}  
**void** json\_array\_free(json\_array \* array) {  
 **for** (**int** i = 0; i < array->length; ++i) {  
 json\_value \* value = array->values[i];  
 **switch** (value->type) {  
 **case** JSON\_OBJ:  
 json\_object\_free(value->value);  
 free(value->value);  
 free(value);  
 **break**;  
 **case** JSON\_ARR:  
 json\_array\_free(value->value);  
 free(value->value);  
 free(value);  
 **break**;  
 **case** JSON\_STR:  
 **case** JSON\_BOOL:  
 **case** JSON\_NUM:  
 free(value->value);  
 free(value);  
 **break**;  
 **default**:  
 **break**;  
 }  
 }  
 free(array->values);  
}

**ЗАКЛЮЧЕНИЕ**

В данном курсовом проекте мною была написана программа, реализующая парсер JSON модели DOM

Она поддерживает базовый функционал, необходимый для корректной работы программы, а также полностью соответствует заданным мне в индивидуальном задании требованиям.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Описание стандарта JSON (https://www.json.org/json-ru.html)

# Язык С (https://en.cppreference.com/w/)