Санкт-Петербургский политехнический университет

Институт компьютерных наук и технологий

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

**КУРСОВОЙ ПРОЕКТ**

**Разработка парсера JSON (DOM)**

по дисциплине «Прикладное программирование»

Выполнила

студентка гр.23531/1 <*подпись*> М.А. Кузина

Руководитель

<*подпись*> М.А. Петров

«\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2018 г.

Санкт-Петербург

2018

Санкт-Петербургский политехнический университет

**ЗАДАНИЕ**

**НА ВЫПолнение курсового проекта**

студентке группы 23531/1 Кузиной Марии Александровне

***1. Тема проекта (работы):*** разработка парсера JSON (DOM)

***2. Срок сдачи законченного проекта (работы)*** 02.06.2018

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

Парсер должен посимвольно считывать полученный на вход файл в формате json и формировать внутреннее представление в виде древовидной структуры

Для проверки работы должен использоваться вывод в cmd.

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

***Дата получения задания***: «\_\_\_». \_\_\_\_\_\_\_\_\_\_\_\_ 2018 г.

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

Задание принял к исполнению \_\_\_\_\_\_\_\_\_\_\_\_\_ М.А. Кузина

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

*(дата)*

**СОДЕРЖАНИЕ**

|  |  |
| --- | --- |
| Введение. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . ……………4. |  |
| Методика решения…………………………………………………...........................5 |  |
| Код программы, реализующей парсер…………………..………………….………6 |  |
| Пример работы программы…………………………………………………………11 |  |
| Заключение ...................................................................................................................12 |  |
| Список использованных источников  .......................................................................12 |  |
|  |  |
|  |  |

**ВВЕДЕНИЕ**

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

В ней будут реализовано чтение файла в формате JSON посимвольно, формирование древовидного представления информации в памяти компьютера, а также вывод полученной структуры в cmd интерпретатор командной строки для проверки правильности работы программы.

**МЕТОДИКА РЕШЕНИЯ**

Программа берет входной файл в формате json из файла input.json. Все данные считаются корректными. Посимвольно считывает информацию из входного файла и передает ее в основную функцию process\_char. Для анализа формата json у парсера есть два состояния:

* number;
* string;
* пустое состояние

От состояния парсера и от текущего символа зависит распознавание информации. Так, если встречается кавычка, то устанавливается тип строка, если встречается цифра, то число. Далее заполняется древовидная структура и подсчитываются дети объекта.

В зависимости от состояния и от полученного символа функция анализирует полученные данные и формирует древовидную структуру данных.

Для контроля работы используются функции печати print\_node и print\_input. Вывод древовидной структуры в cmd интерпретатор командной строки в соответствующих форматах: для типа число (number) – число, для типа строка (string) – строка в кавычках, для типа список (array) – перечисление отформатированных текстовых представлений содержащихся в списке узлов, для типа набора пар ключ-значение (hash) так же перечисление ключей и текстовых представлений узлов.

**struct Node{**

**char type[STRMAX]; \\** тип узла

**char name[STRMAX];** \\ имя узла ([ключ](file:///\\ключ) узла, если он содержится в hash)

**char value[STRMAX]; \\** значение простого узла

**int children\_count; \\** кол-во детей

**struct Node \*parent;**

**struct Node \*children[MAXCHILDREN];**

**};**

Код программы, реализующей парсер, приложен (Приложение 1).

**ПРИМЕР РАБОТЫ ПРОГРАММЫ**

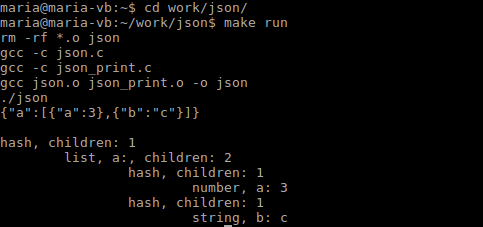
****

Рис1. Печать древовидной структуры в cmd

**ЗАКЛЮЧЕНИЕ**

В данном курсовом проекте мною была написана программа, реализующая парсер формата JSON.

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

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Герберт Шилдт. Полный справочник по C, 4-е издание. — Москва: Вильямс, 2004. — 800 с.
2. Интернет-ресурсы по программированию на C/C++, такие как stackoverflow (<https://stackoverflow.com/>)

**ПРИЛОЖЕНИЕ 1**

**Код программы, реализующей парсер**

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <string.h>**

**#define STRMAX 100**

**#define MAXCHILDREN 100**

**struct Node{**

**char type[STRMAX];**

**char name[STRMAX];**

**char value[STRMAX];**

**int children\_count;**

**struct Node \*parent;**

**struct Node \*children[MAXCHILDREN];**

**};**

**struct State{**

**char ch;**

**FILE \*input\_file;**

**char name[STRMAX];**

**char previous[STRMAX];**

**char key[STRMAX];**

**char token[STRMAX];**

**struct Node \*current;**

**struct Node \*parent;**

**struct Node \*root;**

**char last\_key[STRMAX];**

**int token\_length;**

**bool escape;**

**};**

**void print\_node( char indent[STRMAX], struct Node \*node );**

**void free\_node( struct Node \*node );**

**void print\_self\_node( struct Node \*node );**

**void print\_input( struct State \*machine );**

**void init\_state( struct State \*machine );**

**void end\_state( struct State \*machine );**

**void process( struct State \*machine );**

**void process\_char( struct State \*machine );**

**typedef void (\*StateHandler)(struct State \*);**

**typedef char String[STRMAX];**

**typedef struct State \*Machine;**

**typedef struct Node \*NodePtr;**

**typedef const char \*StringPtr;**

**int main(){**

**struct State \*machine;**

**machine =(struct State \*) malloc (sizeof(struct State));**

**print\_input( machine );**

**printf("\n\n");**

**init\_state( machine );**

**process( machine );**

**end\_state( machine );**

**return 0;**

**}**

**bool strings\_equal( StringPtr str1, StringPtr str2 ){**

**return strcmp( str1, str2 ) ==0; }**

**void print\_node( char indent[STRMAX], struct Node \*node ){**

**printf( "%s%s", indent, node->type );**

**if( !strings\_equal( node->name, "" ) ) printf( ", %s:", node->name );**

**if( !strings\_equal( node->value, "" ) ) printf( " %s", node->value );**

**if( node->children\_count >0 ) printf( ", children: %d", node->children\_count );**

**printf( "\n" );**

**char indent2[STRMAX];**

**strncpy( indent2, indent, STRMAX );**

**strcat( indent2, " " );**

**for( int i =0; i < node->children\_count; i++ ){**

**print\_node( indent2, node->children[ i ] );**

**}**

**}**

**void free\_node( struct Node \*node ){**

**for( int i =0; i < node->children\_count; i++ ){**

**free\_node( node->children[ i ] );**

**}**

**free( node );**

**}**

**void print\_input( struct State \*machine ){**

**machine->input\_file = fopen("input.json", "r"); // read mode**

**while((machine->ch = fgetc(machine->input\_file)) != EOF){**

**printf("%c", machine->ch);**

**}**

**fclose(machine->input\_file);**

**}**

**void init\_state( struct State \*machine ){**

**machine->input\_file = fopen("input.json", "r"); // read mode**

**strncpy( machine->name, "", STRMAX );**

**strncpy( machine->key, "", STRMAX );**

**strncpy( machine->token, "", STRMAX );**

**strncpy( machine->last\_key, "", STRMAX );**

**machine->token\_length =0;**

**machine->current =0;**

**machine->parent =0;**

**machine->escape =false;**

**}**

**void end\_state( struct State \*machine ){**

**char indent[STRMAX] ="";**

**print\_node( indent, machine->root );**

**free\_node( machine->root );**

**fclose(machine->input\_file);**

**free( machine );**

**}**

**void process( struct State \*machine ){**

**while((machine->ch = fgetc(machine->input\_file)) != EOF){**

**//printf("%c\n", ch);**

**process\_char( machine );**

**}**

**}**

**bool check\_state( Machine machine, StringPtr state ){**

**return strings\_equal( machine->name, state ); }**

**void string\_copy( char \*dest, StringPtr src ){**

**strncpy( dest, src, STRMAX ); }**

**bool check\_char( Machine machine, char ch ){**

**return machine->ch ==ch; }**

**bool is\_number\_char( Machine machine ){**

**return machine->ch =='.' || ( machine->ch >= '0' && machine->ch <= '9' ); }**

**void set\_type( Machine machine, StringPtr type ){**

**string\_copy( machine->current->type, type ); }**

**bool is\_token\_state( StringPtr state ){**

**return strings\_equal( state, "string" ) || strings\_equal( state, "number" ); }**

**void set\_key( NodePtr node, StringPtr key ){**

**string\_copy( node->name, key ); }**

**void empty\_token( Machine machine ){**

**string\_copy( machine->token, "" );**

**machine->token\_length =-1; }**

**void new\_node( Machine machine, StringPtr type ){**

**machine->current=(NodePtr) malloc (sizeof(struct Node));**

**machine->current->children\_count =0;**

**set\_type( machine, type );**

**if( machine->parent !=0 ){**

**machine->parent->children[ machine->parent->children\_count ] =machine->current;**

**machine->parent->children\_count++;**

**machine->current->parent =machine->parent;**

**set\_key( machine->current, machine->key );**

**}else{**

**machine->parent = machine->current;**

**machine->root = machine->current;**

**}**

**string\_copy( machine->key, "" );**

**}**

**void state\_change\_handler( Machine machine, StringPtr old\_state, StringPtr new\_state ){**

**if( machine->parent != 0 ){**

**if( strings\_equal( machine->parent->type, "list" )**

**&& is\_token\_state( old\_state ) ){**

**string\_copy( machine->key, "" );**

**new\_node( machine, old\_state ); }**

**if( strings\_equal( machine->parent->type, "hash" )**

**&& strings\_equal( old\_state, "string" ) ){**

**string\_copy( machine->key, machine->token ); }**

**if( is\_token\_state( new\_state ) ){**

**if( machine->current != 0 ) set\_type( machine, new\_state ); } }**

**if( machine->current != 0 ){**

**if( is\_token\_state( old\_state ) ){**

**if( strings\_equal( machine->current->type, "string" )**

**|| strings\_equal( machine->current->type, "number" ) ){**

**string\_copy( machine->current->value, machine->token );**

**} } }**

**empty\_token( machine );**

**if( strings\_equal( new\_state, "number" ) ){**

**machine->token[0] =machine->ch;**

**machine->token[1] =0;**

**machine->token\_length =1; }**

**if( strings\_equal( old\_state, "number" ) ){**

**process\_char( machine ); } }**

**void set\_state( Machine machine, StringPtr state ){**

**String old\_state ="";**

**string\_copy( old\_state, machine->name );**

**if( !check\_state( machine, state ) ){**

**string\_copy( machine->name, state );**

**state\_change\_handler( machine, old\_state, state ); } }**

**void append\_char( Machine machine ){**

**if( machine->token\_length <0 ) machine->token\_length =0;**

**machine->token[ machine->token\_length ] =machine->ch;**

**machine->token[ machine->token\_length +1 ] =0;**

**machine->token\_length++; }**

**void process\_char( Machine machine ){**

**if( machine->ch == '\n' || machine->ch == ' ' )**

**return;**

**if( check\_state( machine, "string" ) ){**

**if( machine->escape ){**

**append\_char( machine );**

**machine->escape =false; }**

**else if( check\_char( machine, '\\' ) ){**

**machine->escape =true; }**

**else if( check\_char( machine, '"' ) ){**

**set\_state( machine, "" ); }**

**else{**

**append\_char( machine ); } }**

**else if( check\_state( machine, "number" ) ){**

**if( is\_number\_char( machine ) ){**

**append\_char( machine ); }**

**else{**

**set\_state( machine, "" ); } }**

**else{**

**if( check\_char( machine, '{' ) || check\_char( machine, '[' ) ){**

**String type ="hash";**

**if( check\_char( machine, '[' ) ) string\_copy( type, "list" );**

**if( machine->root ==0 ){**

**new\_node( machine, type ); }**

**else{**

**if( strings\_equal( machine->parent->type, "list" ) ){**

**new\_node( machine, type ); }**

**else{**

**set\_type( machine, type );**

**}**

**machine->parent =machine->current; }**

**machine->current =0; }**

**else if( check\_char( machine, '"' ) ){**

**set\_state( machine, "string" ); }**

**else if( is\_number\_char( machine ) ){**

**set\_state( machine, "number" ); }**

**else if( check\_char( machine, ':' ) ){**

**new\_node( machine, "" ); }**

**else if( check\_char( machine, ',' ) ){**

**machine->current =0;**

**if( !strings\_equal( machine->parent->type, "list" ) ){**

**machine->current =0; } }**

**else if( check\_char( machine, '}' ) || check\_char( machine, ']' ) ){**

**if( strings\_equal( machine->parent->type, "list" )**

**&& machine->token\_length >= 0 ){**

**}**

**machine->parent =machine->parent->parent;**

**machine->current =0; }**

**}**

**strncpy( machine->previous, machine->name, STRMAX );**

**}**