Санкт-Петербургский государственный политехнический университет

Институт компьютерных наук и технологий

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

**Отчет**

**по курсовому проекту**

**по курсу «Прикладное программирование»**

**на тему «SAX Parser формата JSON»**

Выполнил

студент гр.23531/3 \_\_\_\_\_\_\_\_\_ И.В. Долгушев

Руководитель \_\_\_\_\_\_\_\_\_ М.А. Беляев

«\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2018 г.

Санкт-Петербург

2018

Содержание

[1. Техническое задание 3](#_Toc517782145)

[2. Метод решения 5](#_Toc517782146)

[3. Листинг программы 6](#_Toc517782147)

[3.1. Parser.h 6](#_Toc517782148)

[3.2. Parser.c 7](#_Toc517782149)

[3.3. Main.c 20](#_Toc517782150)

[3.4. Makefie 22](#_Toc517782150)

[4. Коды ошибок 22](#_Toc517782154)

[5. Моделирование ошибок. Тестирование 23](#_Toc517782155)

[6. Заключение 34](#_Toc517782155)

[7. Список использованных источников информации 34](#_Toc517782155)

# 1. Техническое задание

В данном курсовом проекте будет разработан SAX-парсер JSON формата на языке программирования С, что соответствует индивидуальному заданию по прикладному программированию. На основе разработанного парсера будет разработана программа, отображающая работоспособность написанного парсера. Входные данные будут представлять собой файл JSON, а выходные данные будут печататься в консоль, по мере того как парсер SAX будет присылать события. Рассмотрим каждый аспект подробнее.

1. **Формат командной строки.**

Вызов программы из командной строки будет осуществляться следующим образом:

Parser\_JSON input.json

1. **Формат входного файла**

Формат входного файла: файл формата JSON в котором содержится корневой элемент любого из следующих типов («Объект», «Массив», «null», «true», «false») (текст, соответствующий формату JSON)

1. **Формат выходных данных**

Выходными данными являются выполнение переданных в парсер функций:

* startDocument();
* endDocument();
* startElement(TokenType);
* endElement(TokenType);
* characters(Token);

Реализация данных функций должна содержать вывод строк в консоль:

* “startDocument “
* “endDocument “
* “startElement + TokenType”
* “endElement + TokenType”
* “characters >>> «текстовое содержимое токена»”

1. **Обработка ошибок формата JSON**

Ошибки в формате JSON должны завершать программу с кодом 1 и   
выводить краткое описание сути ошибки.

Таким образом, разработанная программа SAX – парсер JSON формата должна сообщать, когда она находит в JSON-файле разные вещи (начало/конец элемента/документа и передавать при нахождении токена (элемента) его содержимое).

В частности:

* SAX не должен строить в памяти дерево JSON-элементов.
* Парсер SAX должен посылать события по мере того, как он находит в JSON-файле разные элементы. На усмотрение пользователя парсера решать, как (и если) он хочет сохранять эти данные.
* SAX должен посылать события немедленно. Пользователь не должен ожидать, пока парсер закончит чтение документа.

**Тестирование**. Для тестирования будут взяты проверенные на корректность JSON файлы и файлы с ошибкой в JSON. Проверка будет происходить с помощью передачи абсолютного пути тестируемого файла через аргументы к прикладной программе, которая отображает работоспособность написанного парсера. Для тестирования ошибок в формате JSON, требуется создать набор тестов, отображающих корректность завершения программы с описанием сути ошибки JSON формата.

# 2. Метод решения

Программа состоит из 3 файлов. Два файла относятся непосредственно к парсеру (Parcer.c Parcer.h) и файл прикладной программы, использующей парсер Main.c.

Заголовочный файл («header») – Parcer.h – содержит в себе объявление 2 структур: Токена (Token) и Парсера (Parser) и 3 перечислений: перечисление основных типов ошибок при работе парсера, перечисление основных типов JSON, перечисление состояний при парсинге объекта. Также данный файл содержит 4 публичных метода для взаимосвязи с парсером.

Начинается программа с файла Main.c. В него передаются аргументы командной строки для дальнейшей обработки. Если строка запуска удовлетворяет заданному формату, то происходит передача строки, содержащей путь до json файла в аргументы функции getJSON(), которая также принимает в аргументы переопределенные нами функции обратного вызова. Далее работа происходит в файле Parse.c.

Следующий этап – за первый проход подсчитывается кол-во токенов, за второй проход происходит проверка токенов на корректность и проброс через метод обратного вызова, а также запись в массив токена.

В программе используются только стандартные подключаемые библиотеки и методы. Все действие основано на обработке строк и символов.

# 3. Листинг программы

## 3.1. Parser.h

|  |
| --- |
| #ifndef **\_\_PARSER\_H\_** #define **\_\_PARSER\_H\_** #include **<stddef.h>** #define **N** 80  **typedef enum** {  ***NUMBER*** = 0,  ***OBJECT*** = 1,  ***ARRAY*** = 2,  ***STRING*** = 3,  ***TRUE\_VALUE*** = 4,  ***FALSE\_VALUE*** = 5,  ***NULL\_VALUE*** = 6 } TokenType;  **typedef enum** {  ***ERROR\_ROOT\_TOKEN*** = -1,  ***ERROR\_NUMBER*** = -2,  ***ERROR\_BACKSLASH*** = -3,  ***ERROR\_OBJECT\_PART*** = -4,  ***ERROR\_CHARACTER*** = -5,  ***ERROR\_STRING*** = -6,  ***ERROR\_NOMEM*** = -7,  ***ERROR\_INVAL*** = -8,  ***ERROR\_PART*** = -9,  ***ERROR\_UNDEFINED*** = -10 } TokenError;  **typedef struct** {  TokenType type;  **int** start;  **int** end;  **int** size; } Token;  **typedef enum** {  ***UNDEFINED*** = 0,  ***KEY*** = 1,  ***COLUMN*** = 2,  ***COMMA*** = 3 } HelpState;  **typedef struct** {  **int** position;  **int** toknext;  **int** toksuper; } Parser;  **void** freeParserJSON();  Token \*getNextToken(Token token);  **void** printToken(Token token);  Token getJSON(**char** \*JSON, **void** (\*startDocument)(), **void** (\*endDocument)()  , **void** (\*startElement)(TokenType), **void** (\*endElement)(TokenType), **void** (\*characters)(Token));   #endif |

## 3.2. Parser.c

|  |
| --- |
| #include **<stdlib.h>** #include **<mem.h>** #include **<stdio.h>** #include **<stdbool.h>** #include **"Parser.h"  static** Token \*tokensJSON;  **static char** \*document;  **static void** (\*startDocumentCallback)();  **static void** (\*endDocumentCallback)();  **static void** (\*startElementCallback)(TokenType tokenType);  **static void** (\*endElementCallback)(TokenType tokenType);  **static void** (\*charactersCallback)(Token token);  **static** Token \*alloc\_token(Parser \*parser,  Token \*tokens, size\_t num\_tokens) {  Token \*tok;  **if** (parser->toknext >= (**signed**) num\_tokens) {  **return NULL**;  }  tok = &tokens[parser->toknext++];  tok->start = tok->end = -1;  tok->size = 0;  **return** tok; }  **static void** fill\_token(Token \*token, TokenType type,  **int** start, **int** end) {  token->type = type;  token->start = start;  token->end = end;  token->size = 0;  charactersCallback(\*token); }  **static void** subString(**const char** \*string, **int** offset, **int** length, **char** \*\*dst) {  \*dst = calloc((size\_t) (length + 1), **sizeof**(**char**));  **for** (**int** i = 0; i < length; ++i) {  (\*dst)[i] = string[offset + i];  } }   **static int** parse\_different(Parser \*parser, **const char** \*js,  size\_t len, Token \*tokens, size\_t num\_tokens) {  Token \*token;  **int** start = parser->position;  TokenType tokenType;  **for** (; parser->position < (**signed**) len && js[parser->position] != **'\0'**; parser->position++) {  **switch** (js[parser->position]) {  *// определяем что токен не являющийся массиовм или объектом или  // строкой завершился с помощью одного из следующих символов* **case '\t'** :  **case '\r'** :  **case '\n'** :  **case ' '** :  **case ','** :  **case ']'** :  **case '}'** :  **goto** found;  **default**:  **break**;  }   *// бросаем ошибку потому что нашли символ который не входит в таблицу ASCII* **if** (js[parser->position] < 32 || js[parser->position] >= 127) {  parser->position = start;  **return *ERROR\_INVAL***;  }  }   *//сюда попадем если нашли токен, знаем его старт и конец и валидность символов внутри токена* found:   *//если режим подсчета токенов то ничего не делаем откатываем позицию и прерываем обработку токена* **if** (!tokens) {  parser->position--;  **return** (TokenError) 0;  }   *//Находим тип* **char** \*tokenString;  **int** length = parser->position - start;  subString(js, start, length, &tokenString);   **if** (strcmp(tokenString, **"null"**) == 0) tokenType = ***NULL\_VALUE***;  **else if** (strcmp(tokenString, **"true"**) == 0) tokenType = ***TRUE\_VALUE***;  **else if** (strcmp(tokenString, **"false"**) == 0) tokenType = ***FALSE\_VALUE***;  **else** tokenType = ***NUMBER***;  startElementCallback(tokenType);  **if** (tokenType == ***NUMBER***) {  *//проверяем валидность числа  // 1) начинается с цифры или со знака минус* **if** (!(js[start] < 0x30 || js[start] > 0x39) || js[start] == **'-'**) {  *// 2) если начинается с цифры 0 и если все число не является 0, то обязательно должна быть плавающая точка* **if** (js[start] == **'0'** && start + 1 < parser->position && js[start + 1] != **'.'**) **return *ERROR\_NUMBER***;  *//объявляем переменыые для уникальности одной точки в числе и експоненциальной формы* **int** i = 0;  **bool** hasFloat = **false**;  **bool** hasExp = **false**;  *// смещаемся если был знак '-'* **if** (js[start] == **'-'**) {  i = start + 1;  } **else** {  i = start;  }  *// проверяем остальные числа* **for** (; i < parser->position; ++i) {  *// если это не последний символ и он равен точке - проверяем что точки еще не было и экспоненциалной формы,  // так как точка не может стоять после экспоненциальной формы, смещаемся если все норм :)* **if** (i < parser->position - 1 && js[i] == **'.'**) {  **if** (!hasFloat && !hasExp) {  i++;  hasFloat = **true**;  } **else return *ERROR\_NUMBER***;  }  *//проверка экспоненциальной формы* **if** (i < parser->position - 1 && (js[i] == **'e'** || js[i] == **'E'**)) {  **int** offset = 1;  **if** (js[i + 1] == **'-'** || js[i + 1] == **'+'**) offset++;  **if** (!hasExp) {  i += offset;  hasExp = **true**;  } **else return *ERROR\_NUMBER***;  }  *//не является цифрой* **if** (js[i] < 0x30 || js[i] > 0x39) {  **return *ERROR\_NUMBER***;  }  }  } **else return *ERROR\_INVAL***;   }   *//выделяем память под токен* token = alloc\_token(parser, tokens, num\_tokens);   *//если выделенная память на кол-во токенов заполнена (исчерпали все кол-во токенов в аргументах парсера)* **if** (!token) {  parser->position = start;  **return *ERROR\_NOMEM***;  }   *//если все норм, то заполняем токен* fill\_token(token, tokenType, start, parser->position);  *// откатываем позицию назад* endElementCallback(tokenType);  parser->position--;   **return** 0; }  **static int** parse\_string(Parser \*parser, **const char** \*js,  size\_t len, Token \*tokens, size\_t num\_tokens) {  *//токен строки* Token \*token;  **int** start = parser->position;  parser->position++;  **if** (tokens) {  startElementCallback(***STRING***);  }  **for** (; parser->position < (**signed**) len && js[parser->position] != **'\0'**; parser->position++) {  *//символ строки* **char** c = js[parser->position];  **if** (c == **'\t'** || c == **'\n'**) {  **return *ERROR\_STRING***;  }  *//нашли кавычку* **if** (c == **'\"'**) {  *//если режим подсчета токенов, ниче не делаем возвращаем нолик :)* **if** (!tokens) {  **return** 0;  }  *// выделяем память под токен стринга* token = alloc\_token(parser, tokens, num\_tokens);  *// если исчерпали ресурс токенов (заданное кол-во)* **if** (!token) {  parser->position = start;  **return *ERROR\_NOMEM***;  }  *//заполняем токен* fill\_token(token, ***STRING***, start + 1, parser->position);  **return** 0;  }  *// если нашли обратный слэш и после него есть какой то символ* **if** (c == **'\\'** && parser->position + 1 < (**signed**) len) {  **int** i;  parser->position++;  **switch** (js[parser->position]) {  **case '\"'**:  **case '/'** :  **case '\\'** :  **case 'b'** :  **case 'f'** :  **case 'r'** :  **case 'n'** :  **case 't'** :  **break**;  **case 'u'**:  parser->position++;  **for** (i = 0; i < 4 && parser->position < (**signed**) len && js[parser->position] != **'\0'**; i++) {  **if** (!((js[parser->position] >= 48 && js[parser->position] <= 57) || */\* проверяем 0-9 \*/* (js[parser->position] >= 65 && js[parser->position] <= 70) || */\* проверяем A-F \*/* (js[parser->position] >= 97 && js[parser->position] <= 102))) { */\* проверяем a-f \*/* parser->position = (**unsigned int**) start;  **return *ERROR\_INVAL***;  }  parser->position++;  }  parser->position--;  **break**;  **default**:  parser->position = start;  **return *ERROR\_BACKSLASH***;  }  }  }  parser->position = (**unsigned int**) start;  **return *ERROR\_PART***; }   **static int** parse(Parser \*parser, **const char** \*js, size\_t len,  Token \*tokens, **unsigned int** countTokens) {  HelpState helpState = ***UNDEFINED***;  TokenError r;  **int** i;  Token \*tokenTemp;  *// Подсчет кол-ва токенов* **int** count = 0;  **if** (tokens)  startDocumentCallback();  *//пока не уткнулись в конец строки или не превысили указанный диапазон строки передвигаемся посимвольно* **for** (; parser->position < (**signed**) len && js[parser->position] != **'\0'**; parser->position++) {  *//определяем тип токена* TokenType type;  *//текущий символ* **char** c = js[parser->position];   **switch** (c) {  *// Если начало токена массива или объекта* **case '{'**:  **case '['**:  *// Мы нашли новый токен* count++;  *// Если режим подсчета токенов, то посчитали и хватит :)* **if** (!tokens) {  **break**;  }  **if** (c == **'{'**)startElementCallback(***OBJECT***);  **else** startElementCallback(***ARRAY***);  *// выделяем токен* tokenTemp = alloc\_token(parser, tokens, countTokens);  *// если превысили допустимое кол-во токенов* **if** (!tokenTemp)  **return *ERROR\_NOMEM***;  **if** (c == **'{'** && helpState != ***UNDEFINED***) helpState = ***UNDEFINED***;  *// если обрабатываем какой-то токен и нашли токен внутри него, то увеличиваем его кол-во детей* **if** (parser->toksuper != -1) {  tokens[parser->toksuper].size++;  }  *// определяем что же ве таки объект или массив* tokenTemp->type = (c == **'{'** ? ***OBJECT*** : ***ARRAY***);  *// устанавливаем токену метку начала токена в строке* tokenTemp->start = parser->position;  *// ставим родителя предпоследний токен* parser->toksuper = parser->toknext - 1;  **break**;  *// Если закончился токен (массив/объект)* **case '}'**:  **case ']'**:  *// Если был режим подсчета, то ниче не делаем* **if** (!tokens)  **break**;  *// определяем что тип токена* type = (c == **'}'** ? ***OBJECT*** : ***ARRAY***);  **if** (c == **'}'**) {  **if** (helpState == ***COLUMN*** || helpState == ***UNDEFINED***) {  helpState = ***COLUMN***;  } **else return *ERROR\_OBJECT\_PART***;  }  *// пробегаемся по всем найденым токенам* **for** (i = parser->toknext - 1; i >= 0; i--) {  tokenTemp = &tokens[i];  *// если начало существует а конец еще не определен* **if** (tokenTemp->start != -1 && tokenTemp->end == -1) {  *// если найденный тип не соответсвует Пример: {[ }] скобка закончилась раньше* **if** (tokenTemp->type != type) {  **return *ERROR\_PART***;  }  *// сбрасываем текущий обрабатываемый токен* parser->toksuper = -1;  *// устанавливаем конец токена в строке* tokenTemp->end = parser->position + 1;  charactersCallback(\*tokenTemp);  **break**;  }  }  *// если не нашли обрабатываемый токен (когда кол-во закрывающих скобок больше начинающих)* **if** (i == -1) **return *ERROR\_PART***;  *// доходим до конца массива токенов для того чтобы узнать позицию следующего обрабатываемого токена* **for** (; i >= 0; i--) {  tokenTemp = &tokens[i];  *// если не обработаный в плане границ токен то устанавливаем индекс текущего обрабатываемого токена* **if** (tokenTemp->start != -1 && tokenTemp->end == -1) {  *// текущий индекс токена в массиве токенов который мы обрабатываем* parser->toksuper = i;  **break**;  }  }  endElementCallback(***ARRAY***);  **break**;  *// Если нашли кавычку* **case '\"'**:   *// проверяем строку* r = (TokenError) parse\_string(parser, js, len, tokens, countTokens);  **if** (r < 0)**return** r;   *// если поймали ошибку в формате строки то возвращаем код ошибки  // токен строки считаем* count++;  **if** (!tokens) **break**;  **if** (parser->toksuper == -1)  **return *ERROR\_ROOT\_TOKEN***;  **if** (helpState == ***UNDEFINED*** && tokens[parser->toksuper].type == ***OBJECT***) helpState = ***COMMA***;  **if** (tokens[parser->toksuper].type == ***OBJECT***) {  **if** (helpState == ***COMMA***) helpState = ***KEY***;  **else if** (helpState != ***COLUMN***)  **return *ERROR\_OBJECT\_PART***;  }  *// если режим не подсчета токенов и предок есть то увеличиваем кол-во чайлдов* **if** (parser->toksuper != -1)  tokens[parser->toksuper].size++;  endElementCallback(***STRING***);  **break**;  *// если управляющие символы или пробел то ничего не делаем* **case '\t'** :  **case '\r'** :  **case '\n'** :  **case ' '**:  **break**;  *// нашли разделитель между ключем и значением* **case ':'**:  *//ставим родителя как предпоследний токен  // parser->toksuper = parser->toknext - 1;* **if** (!tokens)**break**;  **if** (tokens[parser->toksuper].type != ***OBJECT***) **return *ERROR\_CHARACTER***;  **if** (helpState == ***KEY***) helpState = ***COLUMN***;  **else return *ERROR\_OBJECT\_PART***;  **break**;  *// если нашли разграничитель между токенами* **case ','**:  *// переопределяем ToSuper так как мы перечисляем токены в Toksuper  // если режим не подсчета токенов и мы находимся не в массиве и не в объекте* **if** (tokens && tokens[parser->toksuper].type == ***OBJECT***) {  **if** (helpState == ***COLUMN***) helpState = ***COMMA***;  **else** {  **return *ERROR\_OBJECT\_PART***;  }  }  **if** (tokens && tokens[parser->toksuper].type != ***ARRAY*** && tokens[parser->toksuper].type != ***OBJECT***) {  *// идем по всем токенам и находим самый близжайший массив или объект* **for** (i = parser->toknext - 1; i >= 0; i--) {  *// если токен равен объекту или массиву* **if** (tokens[i].type == ***ARRAY*** || tokens[i].type == ***OBJECT***) {  *// если найденный токен не закрыт, то указываем родителя на него* **if** (tokens[i].start != -1 && tokens[i].end == -1) {  parser->toksuper = i;  **break**;  }  }  }  }  **break**;  *//если нашли что то другое* **default**:  *//пробуем определить что это число/true/false/null* r = (TokenError) parse\_different(parser, js, len, tokens, countTokens);  **if** (r < 0) **return** r;  *//увеличиваем кол-во токенов если режим подсчета* count++;  *//увеличиваем кол-во чайлдов если режим не подсчета и мы обрабатываем какой то токен* **if** (parser->toksuper != -1 && tokens)  tokens[parser->toksuper].size++;  **break**;  }  }  *// если режим не поиска кол-ва токенов то проверяем корневой токен на тип* **if** (tokens && (tokens[0].type != ***OBJECT*** && tokens[0].type != ***ARRAY*** && tokens[0].type != ***TRUE\_VALUE*** && tokens[0].type != ***FALSE\_VALUE*** && tokens[0].type != ***NULL\_VALUE***)) {  **return *ERROR\_ROOT\_TOKEN***;  }  *// пробегаемся по всем токенам если они есть и проверяем на завершенность обработки  // токенов* **for** (i = parser->toknext - 1; i >= 0; i--) {  **if** (tokens[i].start != -1 && tokens[i].end == -1) {  **return *ERROR\_PART***;  }  }  **if** (tokens)  endDocumentCallback();  **return** count; }   **static void** trim(**const char** \*line, **char** \*y, **int** \*startName) {  **int** newSize = **N**;  **for** (**int** i = 0; i < **N**; ++i) {  **if** (line[i] != **'\n'** || \*startName == 1) {  **if** (line[i] == **' '**) {  **if** (\*startName == 1) {  y[i - (**N** - newSize)] = line[i];  } **else** newSize--;  } **else** {  y[i - (**N** - newSize)] = line[i];  }  **if** (line[i] == **'\"'**) {  **if** (\*startName == 0) \*startName = 1;  **else** \*startName = 0;  }  } **else** {  **break**;  }  } }  **static char** \*concat(**char** \*s1, **char** \*s2) {   size\_t len1 = strlen(s1);  size\_t len2 = strlen(s2);  **char** \*result = calloc((size\_t) len1 + len2 + 1, **sizeof**(**char**));  **if** (!result) {  fprintf(**stderr**, **"calloc() failed: insufficient memory!\n"**);  **return NULL**;  }  memcpy(result, s1, len1);  memcpy(result + len1, s2, len2 + 1);  **return** result; }  **static char** \*parseFromFile(**char** name[]) {  FILE \*fp;  **char** line[**N**];  **char** refactorLine[**N**];  **char** \*first = **""**;  memset(refactorLine, 0, **sizeof** refactorLine);  **int** startName = 0;  fp = fopen(name, **"r"**);  **if** (fp == **NULL**) {  perror(**"File not found"**);  exit(1);  }  **while** (fgets(line, **N**, fp) != **NULL**) {  trim(line, refactorLine, &startName);  first = concat(first, refactorLine);  memset(refactorLine, 0, **sizeof** refactorLine);  }  fclose(fp);  **return** first; }  **static char** \*getJsonInline(**char** \*path) {  **return** parseFromFile(path); *//create String JSON without space and \n* }   Token \*getNextToken(Token token) {  **for** (**int** i = 0; i < (**signed**) (\_msize(tokensJSON) / **sizeof**(Token)); ++i) {  **if** (tokensJSON[i].start == token.start && tokensJSON[i].end == token.end) {  **return** &tokensJSON[i + 1];  }  }  **return NULL**; }  **void** printToken(Token token) {  **char** \*result;  **int** size = token.end - token.start;  subString(document, token.start, size, &result);  printf(**"%s"**, result);  free(result); }  **static void** init(Parser \*parser) {  parser->position = 0;  parser->toknext = 0;  parser->toksuper = -1; }  **static void** throwError(**int** error) {  **switch** (error) {  **case *ERROR\_ROOT\_TOKEN***:  perror(**"Undefined JSON ROOT. A JSON payload should be an object or array"**);  exit(***ERROR\_ROOT\_TOKEN***);  **break**;  **case *ERROR\_NUMBER***:  perror(**"The number must be in the correct form"**);  exit(***ERROR\_NUMBER***);  **break**;  **case *ERROR\_BACKSLASH***:  perror(**"Undefined control character"**);  exit(***ERROR\_BACKSLASH***);  **break**;  **case *ERROR\_NOMEM***:  perror(**"cant allocate memory for tokens"**);  exit(***ERROR\_NOMEM***);  **break**;  **case *ERROR\_INVAL***:  perror(**"Illegal Arguments!"**);  exit(***ERROR\_INVAL***);  **break**;  **case *ERROR\_PART***:  perror(**"Incorrect part"**);  exit(***ERROR\_PART***);  **break**;  **case *ERROR\_CHARACTER***:  perror(**"Undefined character ':'"**);  exit(***ERROR\_CHARACTER***);  **break**;  **case *ERROR\_OBJECT\_PART***:  perror(**"Object in JSON should be {\"key\": value, \"key\":value}"**);  exit(***ERROR\_OBJECT\_PART***);  **break**;  **case *ERROR\_STRING***:  perror(**"String is broken"**);  exit(***ERROR\_STRING***);  **break**;;  **default**:  perror(**"Unsupported Error"**);  exit(***ERROR\_UNDEFINED***);  **break**;  } }  */\*\*  \* Parse JSON String & convert to Tokens  \* @return root Token (JSON)  \*/* **static** Token \*getJsonTokens() {  **char** \*jsonLine = document;  **int** count; *//count of Token* **int** err; *//value Error* Parser p; *//parser* init(&p);  *// за первый проход считаем кол-во токенов для того чтобы выделить память* count = parse(&p, jsonLine, strlen(jsonLine), **NULL**, 10);  **if** (count < 0) {  throwError(count);  }  *// выделяем память для массива токенов* tokensJSON = calloc((size\_t) count, **sizeof**(Token));  **if** (!tokensJSON) {  throwError(***ERROR\_NOMEM***);  }  init(&p);  err = parse(&p, jsonLine, strlen(jsonLine), tokensJSON, (**unsigned int**) count);  **if** (err < 0) {  throwError(err);  }  **return** tokensJSON; }  */\*\*  \* Convert JSON text in file to inline text  \* Start parse JSON String & convert to Tokens  \* @return root Token (JSON)  \*/* Token getJSON(**char** \*JSON, **void** (\*startDocument)(), **void** (\*endDocument)(), **void** (\*startElement)(TokenType),  **void** (\*endElement)(TokenType), **void** (\*characters)(Token)) {  startDocumentCallback = startDocument;  endDocumentCallback = endDocument;  startElementCallback = startElement;  endElementCallback = endElement;  charactersCallback = characters;  document = getJsonInline(JSON);  **return** getJsonTokens()[0]; }  **void** freeParserJSON(){  free(tokensJSON);  free(document); } |

## 3.3. Main.c

|  |
| --- |
| #include **<stdio.h>** #include **<stdlib.h>** #include **"Parser.h"** */\*\*  \* Require for give string value tokenType  \* \*/* **static char** \*getStringType(TokenType tokenType) {  **switch** (tokenType) {  **case *ARRAY***:  **return "ARRAY"**;  **case *OBJECT***:  **return "OBJECT"**;  **case *NUMBER***:  **return "NUMBER"**;  **case *STRING***:  **return "STRING"**;  **case *NULL\_VALUE***:  **return "NULL"**;  **case *TRUE\_VALUE***:  **return "TRUE"**;  **case *FALSE\_VALUE***:  **return "FALSE"**;  **default**:  perror(**"UNDEFINED TOKEN TYPE"**);  exit(1);  } }  */\*\*  \* Прерываем программу с помощью ввода данных  \* \*/* **static void** interrupt() {  **int** c = getchar();  **while** (c != **'\n'**) {  c = getchar();  } }  */\*\*  \* implements require for callback notify event  \*/* **static void** startDocument() {  printf(**"starDocument"**);  interrupt(); }  **static void** endDocument() {  printf(**"endDocument"**);  interrupt(); }  **static void** startElement(TokenType tokenType) {  printf(**"startElement. Type = "**);  printf(**"%s"**, getStringType(tokenType));  interrupt(); }  **static void** endElement(TokenType tokenType) {  getStringType(tokenType);  printf(**"endElement. Type = "**);  printf(**"%s"**, getStringType(tokenType));  interrupt(); }  **static void** characters(Token token) {  printf(**"characters >>> "**);  printToken(token);  interrupt(); }  **int** main(**int** argc, **char** \*argv[]) {   */\*\*  \* SAX ParserJSON  \*\*/* **if** (argc != 2) {  perror(**"Does not match the format of the input command. \n For example: \"parser C:\\Users\\ilya\\Desktop\\testForJson\\pass1.json\""**);  **return** 1;  }  Token token = getJSON(argv[1], startDocument,  endDocument, startElement, endElement, characters);  printToken(\*getNextToken(token));  freeParserJSON();  **return** 0; } |

## 3.4. Makefile

|  |
| --- |
| **CC**=gcc **CFLAG**=-pedantic -Wall -Wextra **CFLAGS**=$(CFLAG) -c -o  Parser\_JSON : main.o parser.o  $(CC) $(CFLAGS) -o Parser\_JSON main.o parser.o  main.o : Main.c  $(CC) $(CFLAGS) main.o Main.c  parser.o : Parser.c  $(CC) $(CFLAGS) parser.o Parser.c |

# 4. Коды ошибок

|  |  |  |
| --- | --- | --- |
| **Код ошибки** | **Сообщение** | **Комментарий** |
| 0 | - | Успешное завершение |
| 1 | - | Остановка программы пользователем. |
| -1 | Undefined JSON ROOT. A JSON payload should be an object, array, null true, false | Не определен корневой элемент. Корневым элементом могут являться: массив, объект, null, true, false. |
| -2 | The number must be in the correct form | Запись числа JSON записана не в корректной форме |
| -3 | Undefined control character | Управляющий символ не соответствует стандарту POSIX. |
| -4 | Object in JSON should be {"key": value, "key":value} | Нарушена структура описания содержимого объекта. (см. ECMA-404) |
| -5 | Undefined character ':' | Использование данного символа разрешено только внутри объекта. (см. ECMA-404) |
| -6 | String is broken | В строке содержится перевод каретки на следующую строку или табуляция |
| -7 | Can’t allocate memory for tokens | Произошла проблема выделения памяти. Требуется больше свободного места. Рекомендуется ограничить входные данные |
| -8 | Illegal Arguments! | Присутствует некорректный символ при встрече Unicode или отсутствует символ в кодировке ASCII (при этом не входит в состав строки) |
| -9 | Incorrect part | Не соблюдена структура оформления элементов JSON |
| -10 | Unsupported Error | Неопределенный код ошибки |

# 5. Моделирование ошибок. Тестирование

**Ошибка с кодом -1:** Пустой файл с расширением JSON.

|  |  |
| --- | --- |
| Входные данные | Результат |

**Ошибка с кодом -2:** Неправильный формат числа в экспоненциальной форме.

|  |  |
| --- | --- |
| Входные данные | Результат |

**Ошибка с кодом -3:** После одного бэкслеша стоит символ, такой, что образуемая комбинация с бэкслешем не присутствует в системе POSIX.

|  |  |
| --- | --- |
| Входные данные | Результат |

**Ошибка с кодом -4:** Не правильное состояние при завершении объекта. Состояние перечисления элементов в объекте.

|  |  |
| --- | --- |
| Входные данные | Результат |

**Ошибка с кодом -5:** Использование символа двоеточие помимо строки возможно только в разделении ключа и значения в объекте.

|  |  |
| --- | --- |
| Входные данные | Результат |

**Ошибка с кодом -6:** Строка разбита ‘\n’.

|  |  |
| --- | --- |
| Входные данные | Результат |

**Ошибка с кодом -8:** Некорректный символ в составе обозначения юникода.

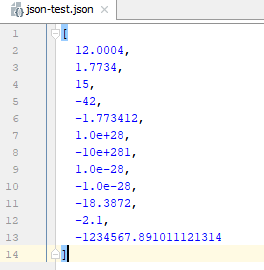
|  |  |
| --- | --- |
| Входные данные | Результат |

**Ошибка с кодом -9:** Структура объекта объявлена, но нет завершающего символа ‘}’.

|  |  |
| --- | --- |
| Входные данные | Результат |

Теперь продемонстрируем работоспособность парсера на следующих примерах:

Входные данные:



Выходные данные:

starDocument

startElement. Type = ARRAY

startElement. Type = NUMBER

characters >>> 12.0004

endElement. Type = NUMBER

startElement. Type = NUMBER

characters >>> 1.7734

endElement. Type = NUMBER

startElement. Type = NUMBER

characters >>> 15

endElement. Type = NUMBER

startElement. Type = NUMBER

characters >>> -42

endElement. Type = NUMBER

startElement. Type = NUMBER

characters >>> -1.773412

endElement. Type = NUMBER

startElement. Type = NUMBER

characters >>> 1.0e+28

endElement. Type = NUMBER

startElement. Type = NUMBER

characters >>> -10e+281

endElement. Type = NUMBER

startElement. Type = NUMBER

characters >>> 1.0e-28

endElement. Type = NUMBER

startElement. Type = NUMBER

characters >>> -1.0e-28

endElement. Type = NUMBER

startElement. Type = NUMBER

characters >>> -18.3872

endElement. Type = NUMBER

startElement. Type = NUMBER

characters >>> -2.1

endElement. Type = NUMBER

startElement. Type = NUMBER

characters >>> -1234567.891011121314

endElement. Type = NUMBER

characters >>> [12.0004,1.7734,15,-42,-1.773412,1.0e+28,-10e+281,1.0e-28,-1.0e-28,-18.3872,-2.1,-1234567.891011121314]

endElement. Type = ARRAY

endDocument

Тест2:

Входные данные:

[  
 **null**,  
 {  
 **"integer"**: 1234567890,  
 **"real"**: -9876.543210,  
 **"e"**: 0.123456789e-12,  
 **"E"**: 1.234567890E+34,  
 **""**: 23456789012E66,  
 **"zero"**: 0,  
 **"one"**: 1,  
 **"space"**: **" "**,  
 **"quote"**: **"\""**,  
 **"backslash"**: **"\\"**,  
 **"controls"**: **"\b\f\n\r\t"**,  
 **"slash"**: **"/ & \/"**,  
 **"alpha"**: **"abcdefghijklmnopqrstuvwyz"**,  
 **"ALPHA"**: **"ABCDEFGHIJKLMNOPQRSTUVWYZ"**,  
 **"digit"**: **"0123456789"**,  
 **"0123456789"**: **"digit"**,  
 **"special"**: **"`1~!@#$%^&\*()\_+-={':[,]}|;.</>?"**,  
 **"hex"**: **"\u0123\u4567\u89AB\uCDEF\uabcd\uef4A"**,  
 **"true"**: **true**,  
 **"false"**: **false**,  
 **"null"**: **null**,  
 **"array"**:[],  
 **"object"**:{},  
 **"address"**: **"50 St. James Street"**,  
 **"url"**: **"http://www.JSON.org/"**,  
 **"comment"**: **"// /\* <!-- --"**,  
 **"# -- --> \*/"**: **" "**,  
 **" s p a c e d "** :[1,2 , 3  
  
,  
  
4 , 5 , 6 ,7 ],**"compact"**:[1,2,3,4,5,6,7],  
 **"jsontext"**: **"{\"object with 1 member\":[\"array with 1 element\"]}"**,  
 **"quotes"**: **"&#34; \u0022 %22 0x22 034 &#x22;"**,  
 **"\/\\\"\uCAFE\uBABE\uAB98\uFCDE\ubcda\uef4A\b\f\n\r\t`1~!@#$%^&\*()\_+-=[]{}|;:',./<>?"**: **"A key can be any string"** },  
 0.5 ,98.6  
,  
99.44  
,  
  
1066,  
1e1,  
0.1e1,  
1e-1,  
1e00,2e+00,2e-00  
,**"rosebud"**]

Выходные данные:

starDocument

startElement. Type = ARRAY

startElement. Type = NULL

characters >>> null

endElement. Type = NULL

startElement. Type = OBJECT

startElement. Type = STRING

characters >>> integer

endElement. Type = STRING

startElement. Type = NUMBER

characters >>> 1234567890

endElement. Type = NUMBER

startElement. Type = STRING

characters >>> real

endElement. Type = STRING

startElement. Type = NUMBER

characters >>> -9876.543210

endElement. Type = NUMBER

startElement. Type = STRING

characters >>> e

endElement. Type = STRING

startElement. Type = NUMBER

characters >>> 0.123456789e-12

endElement. Type = NUMBER

startElement. Type = STRING

characters >>> E

endElement. Type = STRING

startElement. Type = NUMBER

characters >>> 1.234567890E+34

endElement. Type = NUMBER

startElement. Type = STRING

characters >>>

endElement. Type = STRING

startElement. Type = NUMBER

characters >>> 23456789012E66

endElement. Type = NUMBER

startElement. Type = STRING

characters >>> zero

endElement. Type = STRING

startElement. Type = NUMBER

characters >>> 0

endElement. Type = NUMBER

startElement. Type = STRING

characters >>> one

endElement. Type = STRING

startElement. Type = NUMBER

characters >>> 1

endElement. Type = NUMBER

startElement. Type = STRING

characters >>> space

endElement. Type = STRING

startElement. Type = STRING

characters >>>

endElement. Type = STRING

startElement. Type = STRING

characters >>> quote

endElement. Type = STRING

startElement. Type = STRING

characters >>> \"

endElement. Type = STRING

startElement. Type = STRING

characters >>> backslash

endElement. Type = STRING

startElement. Type = STRING

characters >>> \\

endElement. Type = STRING

startElement. Type = STRING

characters >>> controls

endElement. Type = STRING

startElement. Type = STRING

characters >>> \b\f\n\r\t

endElement. Type = STRING

startElement. Type = STRING

characters >>> slash

endElement. Type = STRING

startElement. Type = STRING

characters >>> /&\/

endElement. Type = STRING

startElement. Type = STRING

characters >>> alpha

endElement. Type = STRING

startElement. Type = STRING

characters >>> abcdefghijklmnopqrstuvwyz

endElement. Type = STRING

startElement. Type = STRING

characters >>> ALPHA

endElement. Type = STRING

startElement. Type = STRING

characters >>> ABCDEFGHIJKLMNOPQRSTUVWYZ

endElement. Type = STRING

startElement. Type = STRING

characters >>> digit

endElement. Type = STRING

startElement. Type = STRING

characters >>> 0123456789

endElement. Type = STRING

startElement. Type = STRING

characters >>> 0123456789

endElement. Type = STRING

startElement. Type = STRING

characters >>> digit

endElement. Type = STRING

startElement. Type = STRING

characters >>> special

endElement. Type = STRING

startElement. Type = STRING

characters >>> `1~!@#$%^&\*()\_+-={':[,]}|;.</>?

endElement. Type = STRING

startElement. Type = STRING

characters >>> hex

endElement. Type = STRING

startElement. Type = STRING

characters >>> \u0123\u4567\u89AB\uCDEF\uabcd\uef4A

endElement. Type = STRING

startElement. Type = STRING

characters >>> true

endElement. Type = STRING

startElement. Type = TRUE

characters >>> true

endElement. Type = TRUE

startElement. Type = STRING

characters >>> false

endElement. Type = STRING

startElement. Type = FALSE

characters >>> false

endElement. Type = FALSE

startElement. Type = STRING

characters >>> null

endElement. Type = STRING

startElement. Type = NULL

characters >>> null

endElement. Type = NULL

startElement. Type = STRING

characters >>> array

endElement. Type = STRING

startElement. Type = ARRAY

characters >>> []

endElement. Type = ARRAY

startElement. Type = STRING

characters >>> object

endElement. Type = STRING

startElement. Type = OBJECT

characters >>> {}

endElement. Type = ARRAY

startElement. Type = STRING

characters >>> address

endElement. Type = STRING

startElement. Type = STRING

characters >>> 50 St. James Street

endElement. Type = STRING

startElement. Type = STRING

characters >>> url

endElement. Type = STRING

startElement. Type = STRING

characters >>> http://www.JSON.org/

endElement. Type = STRING

startElement. Type = STRING

characters >>> comment

endElement. Type = STRING

startElement. Type = STRING

characters >>> // /\* <!-- --

endElement. Type = STRING

startElement. Type = STRING

characters >>> # -- --> \*/

endElement. Type = STRING

startElement. Type = STRING

characters >>>

endElement. Type = STRING

startElement. Type = STRING

characters >>> s p a c e d

endElement. Type = STRING

startElement. Type = ARRAY

startElement. Type = NUMBER

characters >>> 1

endElement. Type = NUMBER

startElement. Type = NUMBER

characters >>> 2

endElement. Type = NUMBER

startElement. Type = NUMBER

characters >>> 3

endElement. Type = NUMBER

startElement. Type = NUMBER

characters >>> 4

endElement. Type = NUMBER

startElement. Type = NUMBER

characters >>> 5

endElement. Type = NUMBER

startElement. Type = NUMBER

characters >>> 6

endElement. Type = NUMBER

startElement. Type = NUMBER

characters >>> 7

endElement. Type = NUMBER

characters >>> [1,2,3,4,5,6,7]

endElement. Type = ARRAY

startElement. Type = STRING

characters >>> compact

endElement. Type = STRING

startElement. Type = ARRAY

startElement. Type = NUMBER

characters >>> 1

endElement. Type = NUMBER

startElement. Type = NUMBER

characters >>> 2

endElement. Type = NUMBER

startElement. Type = NUMBER

characters >>> 3

endElement. Type = NUMBER

startElement. Type = NUMBER

characters >>> 4

endElement. Type = NUMBER

startElement. Type = NUMBER

characters >>> 5

endElement. Type = NUMBER

startElement. Type = NUMBER

characters >>> 6

endElement. Type = NUMBER

startElement. Type = NUMBER

characters >>> 7

endElement. Type = NUMBER

characters >>> [1,2,3,4,5,6,7]

endElement. Type = ARRAY

startElement. Type = STRING

characters >>> jsontext

endElement. Type = STRING

startElement. Type = STRING

characters >>> {\"objectwith1member\":[\"arraywith1element\"]}

endElement. Type = STRING

startElement. Type = STRING

characters >>> quotes

endElement. Type = STRING

startElement. Type = STRING

characters >>> &#34; \u0022 %22 0x22 034 &#x22;

endElement. Type = STRING

startElement. Type = STRING

characters >>> \/\\\"\uCAFE\uBABE\uAB98\uFCDE\ubcda\uef4A\b\f\n\r\t`1~!@#$%^&\*()\_+-=[]{}|;:',./<>?

endElement. Type = STRING

startElement. Type = STRING

characters >>> Akeycanbeanystring

endElement. Type = STRING

characters >>> {"integer":1234567890,"real":-9876.543210,"e":0.123456789e-12,"E":1.234567890E+34,"":23456789012E66,"zero

":0,"one":1,"space":" ","quote":"\"",

"backslash": "\\",

"controls": "\b\f\n\r\t",

"slash": "/&\/",

"alpha":"abcdefghijklmnopqrstuvwyz","ALPHA":"ABCDEFGHIJKLMNOPQRSTUVWYZ","digit":"0123456789","0123456789":"digit","speci

al":"`1~!@#$%^&\*()\_+-={':[,]}|;.</>?","hex":"\u0123\u4567\u89AB\uCDEF\uabcd\uef4A","true":true,"false":false,"null":null

,"array":[],"object":{},"address":"50 St. James Street","url":"http://www.JSON.org/","comment":"// /\* <!-- --","# -- -->

\*/":" "," s p a c e d ":[1,2,3,4,5,6,7],"compact":[1,2,3,4,5,6,7],"jsontext":"{\"objectwith1member\":[\"arraywith1eleme

nt\"]}","quotes":"&#34; \u0022 %22 0x22 034 &#x22;","\/\\\"\uCAFE\uBABE\uAB98\uFCDE\ubcda\uef4A\b\f\n\r\t`1~!@#$%^&\*()\_+

-=[]{}|;:',./<>?"

: "Akeycanbeanystring"

}

endElement. Type = ARRAY

startElement. Type = NUMBER

characters >>> 0.5

endElement. Type = NUMBER

startElement. Type = NUMBER

characters >>> 98.6

endElement. Type = NUMBER

startElement. Type = NUMBER

characters >>> 99.44

endElement. Type = NUMBER

startElement. Type = NUMBER

characters >>> 1066

endElement. Type = NUMBER

startElement. Type = NUMBER

characters >>> 1e1

endElement. Type = NUMBER

startElement. Type = NUMBER

characters >>> 0.1e1

endElement. Type = NUMBER

startElement. Type = NUMBER

characters >>> 1e-1

endElement. Type = NUMBER

startElement. Type = NUMBER

characters >>> 1e00

endElement. Type = NUMBER

startElement. Type = NUMBER

characters >>> 2e+00

endElement. Type = NUMBER

startElement. Type = NUMBER

characters >>> 2e-00

endElement. Type = NUMBER

startElement. Type = STRING

characters >>> rosebud

endElement. Type = STRING

characters >>> [null,{"integer":1234567890,"real":-9876.543210,"e":0.123456789e-12,"E":1.234567890E+34,"":23456789012E66

,"zero":0,"one":1,"space":" ","quote":"\"",

"backslash": "\\",

"controls": "\b\f\n\r\t",

"slash": "/&\/",

"alpha":"abcdefghijklmnopqrstuvwyz","ALPHA":"ABCDEFGHIJKLMNOPQRSTUVWYZ","digit":"0123456789","0123456789":"digit","speci

al":"`1~!@#$%^&\*()\_+-={':[,]}|;.</>?","hex":"\u0123\u4567\u89AB\uCDEF\uabcd\uef4A","true":true,"false":false,"null":null

,"array":[],"object":{},"address":"50 St. James Street","url":"http://www.JSON.org/","comment":"// /\* <!-- --","# -- -->

\*/":" "," s p a c e d ":[1,2,3,4,5,6,7],"compact":[1,2,3,4,5,6,7],"jsontext":"{\"objectwith1member\":[\"arraywith1eleme

nt\"]}","quotes":"&#34; \u0022 %22 0x22 034 &#x22;","\/\\\"\uCAFE\uBABE\uAB98\uFCDE\ubcda\uef4A\b\f\n\r\t`1~!@#$%^&\*()\_+

-=[]{}|;:',./<>?"

: "Akeycanbeanystring"

},

0.5,98.6,99.44,1066,1e1,0.1e1,1e-1,1e00,2e+00,2e-00,"rosebud"]

endElement. Type = ARRAY

endDocument

null

Process finished with exit code 0

# 6. Заключение

В ходе выполнения курсовой работы были закреплены знания о формате JSON. Также более детально были изучены особенности программирования на языке C.

Была реализована поставленная задача работы – реализация SAX парсера JSON формата, а затем проведена экспериментальная проверка правильности работы созданного парсера за счет написанной прикладной программы.

# 7. Список использованных источников информации

1.Standard ECMA-404 2nd Edition / December 2017 «The JSON Data Interchange Syntax» <http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf>

2.Пост с Хабра про сборку через Makefile <https://habr.com/post/155201/>

3. Wikipedia – Управляющие символы. <https://ru.wikipedia.org/wiki/%D0%A3%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D1%8F%D1%8E%D1%89%D0%B8%D0%B5_%D1%81%D0%B8%D0%BC%D0%B2%D0%BE%D0%BB%D1%8B>