|  |  |  |
| --- | --- | --- |
| ТИПЫ И СТРУКТУРЫ ДАННЫХ  Лабораторная работа № 3  Json-parser | Студент | Хуинь Минь Туен |
| Группа | ПрИн – 366 |
| Преподавателя | Романенко Р.Р |
| Дата выполнения |  |
| Дата отчета |  |
| Оценка (баллы) |  |

**Задача:**

Реализовать Json-parser

**1. Краткая терия**

JSON (JavaScript Object Notation) — [текстовый формат](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D0%BA%D1%81%D1%82%D0%BE%D0%B2%D1%8B%D0%B9_%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82) [обмена данными](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D0%BC%D0%B5%D0%BD_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D0%BC%D0%B8), основанный на [JavaScript](https://ru.wikipedia.org/wiki/JavaScript) и обычно используемый именно с этим языком. Как и многие другие текстовые форматы, JSON легко читается людьми. Несмотря на происхождение от JavaScript (точнее, от подмножества языка стандарта [ECMA-262](https://ru.wikipedia.org/wiki/ECMA-262) [1999 года](https://ru.wikipedia.org/wiki/1999_%D0%B3%D0%BE%D0%B4)), формат считается независимым от языка и может использоваться практически с любым [языком программирования](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F). Для многих языков существует готовый код для создания и обработки данных в формате JSON.

**Синтаксис**

JSON-текст представляет собой (в закодированном виде) одну из двух структур:

Набор пар ключ:значение. В различных языках это реализовано как [объект](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)), [запись](https://ru.wikipedia.org/wiki/%D0%97%D0%B0%D0%BF%D0%B8%D1%81%D1%8C_(%D1%82%D0%B8%D0%BF_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85)), [структура](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85), [словарь](https://ru.wikipedia.org/wiki/%D0%90%D1%81%D1%81%D0%BE%D1%86%D0%B8%D0%B0%D1%82%D0%B8%D0%B2%D0%BD%D1%8B%D0%B9_%D0%BC%D0%B0%D1%81%D1%81%D0%B8%D0%B2), [хэш-таблица](https://ru.wikipedia.org/wiki/%D0%A5%D1%8D%D1%88-%D1%82%D0%B0%D0%B1%D0%BB%D0%B8%D1%86%D0%B0), [список](https://ru.wikipedia.org/wiki/%D0%A1%D0%BF%D0%B8%D1%81%D0%BE%D0%BA_(%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)) с ключом или [ассоциативный массив](https://ru.wikipedia.org/wiki/%D0%90%D1%81%D1%81%D0%BE%D1%86%D0%B8%D0%B0%D1%82%D0%B8%D0%B2%D0%BD%D1%8B%D0%B9_%D0%BC%D0%B0%D1%81%D1%81%D0%B8%D0%B2). Ключом может быть только строка, значением — любая форма.

Упорядоченный набор значений. Во многих языках это реализовано как [массив](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%81%D1%81%D0%B8%D0%B2_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)), [вектор](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D0%B4%D0%B5%D0%BA%D1%81%D0%BD%D1%8B%D0%B9_%D0%BC%D0%B0%D1%81%D1%81%D0%B8%D0%B2), список или [последовательность](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D1%81%D0%BB%D0%B5%D0%B4%D0%BE%D0%B2%D0%B0%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D0%BE%D1%81%D1%82%D1%8C).

Это универсальные структуры данных: как правило, любой современный язык программирования поддерживает их в той или иной форме. Они легли в основу JSON, так как он используется для обмена данными между различными языками программирования.

В качестве значений в JSON используются структуры:

- Объект — это неупорядоченное множество пар ключ:значение, заключённое в фигурные скобки «{ }». Ключ описывается строкой, между ним и значением стоит символ «:». Пары ключ-значение отделяются друг от друга запятыми.

- Массив (одномерный) — это упорядоченное множество значений. Массив заключается в квадратные скобки «[ ]». Значения разделяются запятыми.

-Значение может быть [строкой](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D1%80%D0%BE%D0%BA%D0%BE%D0%B2%D1%8B%D0%B9_%D1%82%D0%B8%D0%BF) в двойных кавычках, [числом](https://ru.wikipedia.org/wiki/%D0%A7%D0%B8%D1%81%D0%BB%D0%BE), объектом, массивом, одним из [литералов](https://ru.wikipedia.org/wiki/%D0%9B%D0%B8%D1%82%D0%B5%D1%80%D0%B0%D0%BB_(%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)): true, false или null. Т.о. структуры могут быть вложены друг в друга.

Строка — это упорядоченное множество из нуля или более символов [юникода](https://ru.wikipedia.org/wiki/%D0%AE%D0%BD%D0%B8%D0%BA%D0%BE%D0%B4), заключенное в двойные кавычки. Символы могут быть указаны с использованием [escape-последовательностей](https://ru.wikipedia.org/wiki/Escape-%D0%BF%D0%BE%D1%81%D0%BB%D0%B5%D0%B4%D0%BE%D0%B2%D0%B0%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D0%BE%D1%81%D1%82%D1%8C), начинающихся с [обратной косой черты](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%80%D0%B0%D1%82%D0%BD%D0%B0%D1%8F_%D0%BA%D0%BE%D1%81%D0%B0%D1%8F_%D1%87%D0%B5%D1%80%D1%82%D0%B0) «\».

Строка очень похожа на одноимённый тип данных в языках [С](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)) и [Java](https://ru.wikipedia.org/wiki/Java). Число тоже очень похоже на С- или Java-число, за исключением того, что используется только десятичный формат. Пробелы могут быть вставлены между любыми двумя синтаксическими элементами.

Следующий пример показывает JSON-представление объекта, описывающего человека. В объекте есть строковые поля имени и фамилии, объект, описывающий адрес, и массив, содержащий список телефонов.

{

"firstName": "Иван",

"lastName": "Иванов",

"address": {

"streetAddress": "Московское ш., 101, кв.101",

"city": "Ленинград",

"postalCode": 101101

},

"phoneNumbers": [

"812 123-1234",

"916 123-4567"

]

}

**2. Результат работы программы**

****

**3. Программа**

**//**myjson.h

#ifndef MYJSON\_H

#define MYJSON\_H

#define MAX\_ARR\_SIZE 30000

#define MAX\_NUM\_SIZE 30

enum JsonType {

Number =0, /\*INTEGER, BOOL\*/

True = 1,

False = 2,

String = 3,

Object =4,

Array = 5,

None = 6,

Null = 7

};

struct Json{

Json\* prevs, \*next;

Json\* child;

JsonType type;

char\* stringValue; //string in case item is a string

char\* stringName; // key name in case item is subitem of an object

int intValue;

double doubleValue;

} ;

class MyJson

{

public:

char\* parseString(char\* json, Json\* item);

char\* parseNumber(char\* json, Json\* item);

char\* parseObject(char\* json, Json\* item);

char\* parseValue(char\* json, Json\* item);

char\* parseArray(char\* json, Json\* item);

char\* skipSpace(char\* json);

void printString(char\* str);

void printArray(Json\* arr);

void printObject(Json\* obj);

void printValue(Json\* val);

void printNumber(Json\* num);

char\* serializeString(char\* str) ;

char\* serializeNumber(double value );

char\* serializeBoolean(bool value);

char\* serializeNull ();

char\* serializeArray(Json\* arr);

char\* serializeValue(Json\* value);

char\* serializeObject(Json\* json);

void getArrayValues(Json\* object, Json values[], int& length);

void getObjectValues(Json\* object, Json values[], int& length);

Json\* getFieldOfObject(Json\* object, char\* fieldName);

MyJson(void);

~MyJson(void);

};

#endif MYJSON\_H

**//**myjson.cpp

#include "StdAfx.h"

#include "MyJson.h"

#include <iostream>

using namespace std;

MyJson::MyJson()

{

}

char\* MyJson::parseString(char\* json, Json\* item){

if(\*json != '\"'){

cout << "parse string error";

return 0;

}

char\* out, \*ptr = json+1;

int length = 0 ;

while(\*ptr!= '\"' && \*ptr){

++length;

++ptr;

}

//validate string

if(\*ptr == 0) {

return 0 ;

}

out = new char[length+1];

strncpy(out, json+1, length);

out[length] ='\0';

item->type= String;

item->stringValue = out;

return ptr+1;

}

char\* MyJson::serializeString(char\*str ){

int size = strlen(str) ;

char\* out = new char[size +3] ;

char\*ptr = out;

out[0] = '\"';

strncpy(out +1, str, size);

out[size +1] ='\"';

out[size +2] ='\0';

return out;

}

char\* MyJson::parseNumber(char\* json, Json\* item) {

double value = 0;

if(\*json !='-' && \*json< '0' || \*json > '9'){

cout << "parse number error";

return 0;

}

bool isNegative = \*json == '-';

if(isNegative) {

json++;

}

while(\*json >= '0' && \* json <= '9'){

value = value\*10 + \*json-48;

++json ;

}

if(\*json == '.'){

++json;

if(\*json >= '0' && \*json < '9') {

double t = 10;

while(\*json >= '0' && \* json <= '9'){

value+=(\*json -48)/t;

t\*=10;

++json;

}

}else {

//invalid rational number

return 0 ;

}

}

//support rational number type 1.234e5, 1.234e+5, 1.234e-5

if(\*json == 'e' || \*json == 'E') {

++json;

bool isNegative = false;

if(\*json == '-'){

isNegative = true;

++json;

}else if(\*json =='+'){

++json;

}

double exp = 0;

if(\*json>='0' && \*json <='9'){

while (\*json>='0' && \*json <='9'){

exp = exp\*10 + \*json - 48;

++json;

}

exp = isNegative ? exp \* -1 : exp;

value = value\*pow(10.0, exp);

}else { //invalid json number, eg 1.23e

return 0;

}

}

value = isNegative ? value \* -1: value;

item->type= Number;

item->doubleValue = value;

item->intValue = int(value);

return json ;

}

//functionally, this is not safe operation, for example 1e100 yell an error

char\* MyJson::serializeNumber(double value) {

char tmp[MAX\_NUM\_SIZE];

char\* out;

sprintf(tmp, "%f", value);

int len = strlen(tmp);

out = new char[len+1];

strncpy(out, tmp, len);

out[len] ='\0';

return out;

}

char\* MyJson::serializeBoolean(bool value ) {

if(value) {

return "true";

}

return "false";

}

char\* MyJson::serializeNull () {

return "null";

}

char\* MyJson::parseArray(char\* json, Json\* item) {

//must check for valid next token in array [a, b, ]

if(\*json != '['){

cout << "parse array error";

return 0;

}

json = skipSpace(json+1);

if(\*json == ']'){

item->type= Array;

json = skipSpace(json+1);

return json;

}

Json\* child;

item->child = child = new Json();

json = parseValue(json, child);

json = skipSpace(json);

if(!json ){

cout << "parse array error";

return 0;

}

while(\*json == ','){

Json\* newItem;

newItem = new Json();

child->next = newItem;

newItem->prevs = child;

child = newItem;

json = skipSpace(json+1);

json = parseValue(json, child);

//invalid value ["a", ab] or [1,]

if(json == 0) {

return 0;

}

json = skipSpace(json);

}

if(\*json == ']'){

item->type= Array;

return json+1 ;

}else {

return 0;

}

}

void MyJson::printArray(Json\* arr) {

//print arr to json

//check for valid next token in array [a, b, ]

if(arr->child == 0) {

printf("[]\n");

} else {

printf("[\n");

Json\* child = arr->child;

while(child != 0){

printValue(child);

child = child->next;

if(child!= 0){

printf(",");

}

}

printf("\n]\n");

}

}

char\* MyJson::serializeArray(Json\* arr) {

if(arr == 0 ){

return 0 ;

}

char\* out;

int entryCount = 0 ;

Json\* ptr = arr->child;

int len[MAX\_ARR\_SIZE];

char\* items[MAX\_ARR\_SIZE];

int arr\_len = 3;

while(ptr != 0 ){

entryCount ++;

int i = entryCount - 1;

items[i] = serializeValue(ptr);

len[i] = strlen(items[i]);

arr\_len+= len[i];

if(ptr->next != 0){

ptr = ptr->next;

}else {

ptr = 0;

}

}

if(entryCount == 0) {

return "[]";

}

arr\_len += entryCount - 1;

out = new char[arr\_len] ;

out[0] ='[';

char\* currPos = out+1;

for( int i = 0 ; i < entryCount; ++ i ) {

strncpy (currPos, items[i], len[i]);

currPos+=len[i];

\*currPos = ',';

++currPos;

delete [] items[i];

items[i] = 0;

}

\*(currPos - 1)= ']';

\*currPos = '\0';

return out;

}

char\* MyJson::parseObject(char\* json, Json\* item){

if(\*json != '{'){

cout << "parse object error";

return 0;

}

json = skipSpace(json+1);

if(\*json == '}'){

item->type = Object;

return skipSpace(json+1) ;

}

Json\* child = new Json();

item->child = child;

json = skipSpace(json);

json = parseString(json, child);

if(json == 0) {

return 0;

}

json = skipSpace(json);

if(\*json != ':') {

return 0;

}

child->stringName = child->stringValue;

child->stringValue = 0;

json = skipSpace(json +1);

//???

json = parseValue(json, child);

//check for stringName to ensure that the

while(json && \*json == ','){

Json\* newItem = new Json();

newItem->prevs = child;

child->next = newItem;

child = newItem;

json = skipSpace(json+1);

//check for null

json = parseString(json, child);

child->stringName = child->stringValue;

child->stringValue = 0;

json = skipSpace(json);

if(\*json ==':'){

json = skipSpace(json+1);

//parse value to child

json = parseValue(json, child);

}

skipSpace(json);

}

char\*tmp = json;

json = skipSpace(json);

if(\*json !='}'){

return 0;

}

item->type = Object;

return json+1;

}

void MyJson::printObject(Json\* object){

Json\* child = object->child;

printf("\n{\n");

while(child != 0){

printString(child->stringName);

printf(":");

printValue(child);

child = child->next;

if(child != 0){

printf(",");

}

}

printf("\n}\n");

}

char\* MyJson::serializeObject(Json\* object) {

if(!object) {

return 0 ;

}

char\* out;

int entryCount = 0 ;

Json \* ptr = object->child;

char\* items[MAX\_ARR\_SIZE] ;

int len[MAX\_ARR\_SIZE];

int objectLen = 3;

while (ptr != 0 ) {

entryCount++ ;

int i = entryCount - 1;

char\* key = serializeString(ptr->stringName);

char\* value = serializeValue(ptr);

int keyLen = strlen(key);

int valLen = strlen(value );

len[i] = keyLen + valLen + 2;

items[i] = new char[len[i]] ;

char\* pos = items[i];

strncpy (pos, key, keyLen);

delete [] key;

pos+= keyLen ;

\*pos = ':';

++pos;

strncpy ( pos, value, valLen );

delete [] value;

pos += valLen;

\*pos = '\0';

objectLen+= len[i]; // ,

ptr = ptr->next ;

}

if(entryCount == 0) {

return "{}";

}

objectLen+= entryCount - 1;

out = new char[objectLen];

out[0] = '{';

char\* pos = out+1;

for( int i = 0 ; i < entryCount ; ++ i ) {

strncpy(pos, items[i], len[i]);

//TODO why len[i] -1, not len[i] ??????

pos += len[i]-1;

\*pos = ',';

++pos;

delete [] items[i];

items[i] = 0;

}

\*(pos -1) ='}';

\*(pos) = '\0';

return out;

}

char\* MyJson::parseValue(char\* json, Json\* item){

switch(\*json){

case '[':

return parseArray(json, item);

case '{':

return parseObject(json, item);

case '\"':

return parseString(json, item);

case '-':case '0':case '1': case '2': case '3': case '4':case '5':

case '6': case '7': case '8': case '9':

return parseNumber(json, item);

case 't':

if(strlen(json) >=4 && \*(json+1) =='r' && \*(json+2) =='u' && \*(json+3) =='e'){

item->type = True;

return json+4;

}

case 'f':

if(strlen(json) >= 5 && \*(json+1) =='a' && \*(json+2) =='l' && \*(json+3) =='s' && \*(json+4) =='e'){

item->type = False;

return json + 5;

}

case 'n':

if(strlen(json) > 4 && \*(json+1)=='u' && \*(json+2) =='l' && \*(json+3) == 'l'){

item->type = Null;

return json+4;

}

default:

return 0;

}

}

void MyJson::printString(char\* str){

printf("\"%s\"", str);

}

void MyJson::printNumber(Json\* number){

printf("%f", number->doubleValue);

}

void MyJson::printValue(Json\* value){

switch(value->type){

case Number:

printNumber(value);

return ;

case String:

printString(value->stringValue);

return ;

case True:

printf("true");

return ;

case False:

printf("false");

return ;

case Array:

printArray(value);

return ;

case Object:

printObject(value);

return ;

case Null:

printf("null");

return ;

}

}

char\* MyJson::serializeValue (Json\* value) {

switch(value->type) {

case Number:

return serializeNumber(value->doubleValue);

case String:

return serializeString(value->stringValue);

case True:

return serializeBoolean(true);

case False:

return serializeBoolean(false );

case Array:

return serializeArray(value );

case Object:

return serializeObject(value);

case Null:

return serializeNull();

default:return 0 ;

}

}

void MyJson::getArrayValues(Json\* array, Json values[], int& length){

length= 0 ;

Json\* child = array->child;

while(child != 0){

values[length++] =\*child;

child = child->next;

}

}

void MyJson::getObjectValues(Json\* object, Json values[], int& length){

length= 0 ;

Json\* child = object->child;

while(child != 0){

values[length++] =\*child;

if(child->next != 0 ){

child = child->next;

}else {

child = 0 ;

}

}

}

Json\* MyJson::getFieldOfObject(Json\* object, char\* fieldName){

if(object != 0) {

Json\* child = object->child;

while(child != 0 && strcmp(child->stringName, fieldName)){

child = child->next;

}

if(child != 0 && strcmp(child->stringName, fieldName) == 0) {

return child;

}

return 0;

}

return 0;

}

char\* MyJson::skipSpace(char\* json){

while(\*json == ' ' || \*json == '\t' || \*json == '\n'){

json ++;

}

return json;

}

MyJson::~MyJson()

{}

#include "stdafx.h"

#include <iostream>

#include <fstream>

#include <stdio.h>

#include <random>

#include <windows.h>

#include "MyJson.h"

using namespace std;

char\* createBigJsonObject();

char\* readFile(char\* filename);

void writeFile(char\*filename, char\* data);

int main(int argc, char \*argv[])

{

char\*str ="{\"\":\"sjfhsjdhfjsdf\"}";

MyJson parser;

Json\* result = new Json();

parser.parseObject(str, result);

char\* serialized = parser.serializeObject(result);

Json\* value = parser.getFieldOfObject(result, "");

cout << value->stringValue;

system("pause");

return 0 ;

}

char\* readFile(char\* filename) {

FILE\* fp;

fp = fopen(filename,"r");

char c;

char\* text = new char[12\*1024\*2024];

int pos = 0 ;

while(1)

{

c = fgetc(fp);

if(c == '\\') {

c = fgetc(fp);

if(c=='n') {

c = '\n';

}else if(c == 't') {

c = '\t';

}else if(c == '\\'){

}else if(c =='\"') {

c = '\'';

}

}

text[pos++] = c;

if( feof(fp))

{

break ;

}

}

text[pos] = '\0';

fclose(fp);

return text;

}

void writeFile(char\*filename, char\*data){

ofstream myfile;

myfile.open (filename);

myfile << data;

myfile.close();

}