JSON 库程序

C 语言课程项目

[Type the author name]

12/1/2014

同济大学软件学院 C 语言课程项目要求

目录

1. 基本需求

个人项目：编写一个库程序能够提供指定接口对 JSON 数据进行操作。

2. 具体需求

本项目将提供一个.h 头文件，里面定义了 JSON 的数据结构以及需要实现的函数。各位同学需要实现这些接口（**可自行增加需要的函数，但是不能删除给定的头文件里的函数**），之后评测程序将来验证这些接口的是否符合需求。

2.1 数据结构

```
/* cJSON Types: */
#define JSON_FALSE 0
#define JSON_TRUE 1
#define JSON_NULL 2
#define JSON_NUMBER 3
#define JSON_STRING 4
#define JSON_ARRAY 5
#define JSON_OBJECT 6


/* The cJSON structure: */
typedef struct JSON {
    int type;                   /* The type of the item, as above. */

    char *valuestring;          /* The item's string, if type==JSON_STRING */
    int valueint;               /* The item's number, if type==JSON_TRUE||JSON_FALSE */
    double valuedouble;         /* The item's number, if type==JSON_NUMBER  */
} JSON;
```

大家需要**严格按照上面的声明来定义基本数据结构（上面的这些成员及 type 定义是必须的）**，以方便最终的测试。上述 JSON 定义只能覆盖到除 Array 和 Object 外的基本类型，其他 JSON 内成员可以根据自己实现的需要**自由添加**。

2.2 接口

函数声明	函数说明	参数说明
Parse & Print		
JSON *ParseJSON(const char *value);	解析 JSON 字符串	value 为需要解析的字符串，函数返回指向解析后的 JSON 的指针。
JSON *ParseJSONFromFile(const char *file_name);	解析 JSON 文件	file_name 为需要解析的文件名，函数返回指向解析后的 JSON 的指针。
void PrintJSON(JSON *item);	将 JSON **无格式**输出到屏幕。	item 指向需要输出的 JSON。

void PrintJSONToFile(JSON *item, const char *file_name);	将 JSON 按**指定格式**输出到文件中。	Item 指向需要输出的 JSON，file_name 为文件名。
Create		
JSON *CreateNULL(void);	创建一个 NULL。	返回一个指向创建后的 JSON 的指针。
JSON *CreateTrue(void);	创建一个 True。	返回一个指向创建后的 JSON 的指针。
JSON *CreateFalse(void);	创建一个 False。	返回一个指向创建后的 JSON 的指针。
JSON *CreateBool(int b);	创建一个 True 或 False。	b 为 0 则创建 False，否则创建 True。返回一个指向创建后的 JSON 的指针。
JSON *CreateNumber(double num);	创建一个 Number。	num 为值，返回一个指向创建后的 JSON 的指针。
JSON *CreateString(const char *string);	创建一个 String。	string 为值，返回一个指向创建后的 JSON 的指针。
JSON *CreateArray(void);	创建一个 Array。	返回一个指向创建后的 JSON 的指针。
JSON *CreateObject(void);	创建一个 Object。	返回一个指向创建后的 JSON 的指针。
Append		
void AddItemToArray(JSON * array, JSON *item);	将一个 JSON 添加到 Array 中。	将 item 指向的 JSON 添加到 array 指向的 JSON 中。如原来 array 为[1, 2, 3]，添加进一个 type 为 JSON_TRUE 的 item，则 array 变为[1, 2, 3, true]。
void AddItemToObject(JSON *object, const char *key, JSON *value);	讲一个 JSON 添加到 Object 中。	将 item 指向的 JSON 添加到 object 指向的 JSON 中，并且 key 为指定值。如原来 object 为{ “name”: “c”}，添加进一个 type 为 JSON_TRUE 的 item，key 为”is_girl”，则 object 变为{“name”: “c”, “is_girl”: “true”}
Update		
void ReplaceItemInArray(JSON *array, int which, JSON *new_item);	替换 array 中的一个 item。	which 为需要替换的 item 是 array 中第几个，从 0 开始。
void ReplaceItemInObject(JSON *object, const char *key, JSON *new_value);	替换 object 中的一个 item。	key 为需要替换的键，值替换为 new_value 指向的 JSON。
Remove/Delete		
JSON *DetachItemFromArray(JSON *array, int which);	将 array 中的某个元素删除，不释放其内存。	array 指向一个 JSON Array，which 表示要删除的元素的下标，从 0 开始。返回指向被删

		除元素的指针。
void *DeleteItemFromArray(JSON *array, int which);	将 array 中的某个元素删除，释放其内存。	array 指向一个 JSON Array，which 表示要删除的元素的下标，从 0 开始。
JSON *DetachItemFromObject(JSON *object, const char *key);	将 object 中的某个元素删除，不释放其内存。	object 指向一个 JSON Object，key 表示要删除的元素的 key。返回指向被删除元素的指针。
void *DeleteItemFromObject(JSON *object, const char *key);	将 object 中的某个元素删除，释放其内存。	object 指向一个 JSON Object，key 表示要删除的元素的 key。
void DeleteJSON(JSON *item);	删除一个 JSON，释放内存。	Item 指向需要删除的 JSON。
Duplicate		
JSON *Duplicate(JSON *item, int recurse);	复制一个 JSON。	Item 指向需要被复制的 JSON，recurse 表示是否**深度复制**。
Read		
JSON *GetItemInArray(JSON *array, int which);	获取 array 中某个元素。	Which 表示需要获取的元素的下标，从 0 开始。
JSON *GetItemInObject(JSON *object, const char *key);	获取 object 中某个元素。	Key 表示需要获取的元素的键。

JSON *GetItemInJSON(JSON *json, const char *path);	按路径获取 json 中的某个元素。	Path 表示路径，如下面这个 JSON
		{
		“school”: “Tongji”,
		“properties”: {
		“age”: 18,
		“courses”: [
		“C language”,
		“C++ and Java”
		]
		}
		}
		那么，/school 的值为”Tongji”，/properties/courses/1 的值为”C++ and Java”

2.3 说明

- **输出**：所有 key 需要加上双引号，value 格式参考 JSON 标准。JSON_TRUE 和 JSON_FALSE 类型分别输出 true 和 false（小写）,JSON_NULL 输出 null（小写）。JSON_STRING 要求有双引号，JSON_NUMBER 不需要双引号。
- **无格式**：除 string 中的空格之外不能存在其他空格。
  如：
  [true,false,"111",{"fff":"fff"}]

{"age":20}

- **指定格式**：如果 JSON 为 object 或者 array 类型，那么首尾的 '{' 或 '[' 需要各占一行，其余的 '{' 或 '[' 需要在 ':' 后一个空格输出。所有缩进统一为**四个空格**。例子如下：

```
{
    "school": "Tongji",
    "properties": {
        "age": 18,
        "courses": [
            "C language",
            "C++ and Java"
        ]
    }
}
```

- **JSON 文件举例**（给定的 JSON 文件不一定符合指定格式，如样例文件中 age 就没有引号，但这样的读入是合法的）：

```
{
    "school": "Tongji",
    "properties": {
        age: 18,
        "courses": [
            "C language",
            "C++ and Java"
        ]
    }
}
```

- **路径**：从最外层 JSON 开始，以’/’分隔，类似于文件路径，数组中元素用下标定位。如上述 JSON，/school 的值为”Tongji”，/properties/age 的值为 18，/properties/courses/1 的值为”C++ and Java”。
- **深度复制**：递归复制，即创建一个全新的 JSON，不引用原来 JSON 中的任何部分。

3. 项目提交

3.1 提交内容

- JSON.h 包含了自己修改后的 struct JSON 和 要求提供的函数
- JSON.c 实现 JSON.h 中的函数及其他自定义内容
- 其他头文件及 c 文件
- 项目说明文档（大概介绍一下开发过程以及收获，如果有实现其他头文件及 c 文件请说明用法）

3.2 提交方式

3.3 截止时间

4. 评测

4.1 基本要求

实现头文件中指定接口，并且程序能够运行。

4.2 自动化评测

我们将通过程序调用你实现好的函数，通过与预期结果对比实现自动化评测部分。

举例如下（实际测试时测试函数不会这么大，粒度会更小）：

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "JSON.h"

int TestSomeFuncs() {

    int score = 10;

    // 调用同学们实现的接口
    JSON *root = CreateObject();
    // 评测1 是否能正确创建Object
    if (root->type != JSON_OBJECT) { // 类型不对
        score -= 2;
    }

    AddItemToObject(root, "name", CreateString("Hello World"));
    // 评测2 是否能正确AddItemToObject并且读取
    JSON *value = GetItemInObject(root, "name");
    // 类型不对或者值不对
    if (value->type != JSON_STRING || strcmp(value->valuestring, "Hello World")) {
        score -= 2;
    }

    JSON *array = CreateArray();
    AddItemToArray(array, CreateBool(0));
    AddItemToArray(array, CreateNumber(2.3));
    // 评测3 是否能正确AddItemToArray并且读取
    JSON *item = GetItemInArray(array, 1);
    if (item->type != JSON_NUMBER || item->valuedouble != 2.3) {
        score -= 2;
    }
```

```
    AddItemToObject(root, "array", courses);
    // 现在root的状态
    /*
    {
        "name": "Hello Wrold",
        "array": [
            false,
            2.3
        ]
    }
    */

    // 评测4 是否能正确地根据路径读取值
    item = GetItemInJSON(root, "/array/0");
    if (item->type != JSON_FALSE) {
        score -= 2;
    }

    PrintJSONToFile(root, "test.json");
    // 评测5 是否与标准答案文件相同
    if (!IsSame("test.json", "test.json.ans")) {
        score -= 2;
    }
    return score;
}

int main() {
    printf("Score: %d\n", TestSomeFuncs());
    return 0;
}
```

4.3 代码质量

- 代码风格，变量命名、函数命名、缩进等等。
- 是否有大量冗余、重复
- 注释

4.4 其他

- 内存使用及管理情况
- *健壮性

5. 附录

- JSON 标准
- JSON 在线格式化
- JSON.h
- example.json

- test.c