資料結構 HW4 report 徐松廷 110501521

1. 作業目標:

   延續先前作業的程式，增加支援 Tree(nested data type)的操作，樹的每一個節點 (node) 儲存 key 資訊，節點內使用指位器指向 value 資料。指令自行設計，功能為可以存取多棵樹，對任一棵樹新增/搜尋/刪除節點。我們這邊主要存取 json 的資料格式，也就是對於一個 json object 而言，每個 node 都可以是 subtree(nested data type 的性質)。

2. 指令設計與說明:

   a. Json tree 的建立:當輸入指令為: create json_name {M:B,T:E,W:Q,{Q:R,R:Q,{W:Q,P:Z},D:A,B:A},A:T,M:N}
   建立出的 tree 如下，以{}區分一個 level，當{}中出現{}，表示被包住的大括號為其 sub tree:

```
Object_key: Object
  M: "B"
  T: "E"
  W: "Q"
  Object_key: Object
    Q: "R"
    R: "Q"
    Object_key: Object
      W: "Q"
      P: "Z"
    D: "A"
    B: "A"
  A: "T"
  M: "N"
```

   b. 對 json tree 中的資料以 key 搜尋對應 value: find json_name key1，表示查找名稱為 json_name 的 tree 中 Key 為 key1 的 pair 其對應的 value 是多少。

   c. 新增 key-value pair: insert json_name key1 value1 key2 value2，表示在名稱為 json_name 的 tree 中，在 key1-value1 這個 pair 的後方插入 key2-value2 這個 pair。

   d. 刪除 key-value pair: delete json_name key1，表示刪除名稱為 json_name 的 tree 中 Key 為 key1 的 pair。

   e. 印出整個 json tree: show json_name，表示印出名稱為 json_name 的 tree 的所有內容。

3. Code review

   a. Definition of the data structure: 在我們的定義中，JsonNode 可以有兩種型態，一種是 string，也就是 json tree 中的 leaf node，一種則是 Object，表示該 node 有子樹(這邊是用 array 來存放指向 child node 的 pointer)。而定義 struct json_tree_node 的部分則是因為每個 json tree 都有自己的名稱，故開一個 struct 額外處理。

```c
typedef enum {
    STRING,
    OBJECT
} JsonNodeType;
typedef struct JsonNode {
    JsonNodeType type;
    char* key;
    union {
        char* stringValue;
        struct JsonNode** objectValue;
    } value;
    int valid_bool;
} JsonNode;
struct json_tree_node{
    char tree_name[200];
    JsonNode* tree_obj;
};
```

b. 建立 string_node 與 object_node:基本上就是動態配置記憶體的 function，後續會持續使用。

```
JsonNode* createStringNode(const char* key, const char* value) {
    JsonNode* node = (JsonNode*)malloc(sizeof(JsonNode));
    node->type = STRING;
    node->key = strdup(key);
    node->value.stringValue = strdup(value);
    node->valid_bool=1;
    return node;
}

JsonNode* createObjectNode(const char* key, JsonNode** objectValue) {
    JsonNode* node = (JsonNode*)malloc(sizeof(JsonNode));
    node->type = OBJECT;
    node->key = strdup(key);
    node->value.objectValue = objectValue;
    node->valid_bool=1;
    return node;
}
```

c. 建立 json tree 的 function: Input 是 json 格式(ex: {A:B,C:E,{Q:R,R:Q,{W:Q,P:Z}},D:A,A:T})的字串，使用 recursion 實現。以逗號切割並且以{}判定字串的分界(for loop 中的一堆 if else)，判定完成之後再次放入 partition()，持續以遞迴的方式切割到完全無{}後建立成 string node。最後 for loop 跑完之後表示存取 object 成員的 array 建立完成，最後回傳該由 array 建立的 object node。

```
JsonNode* partition(char *buffer){
    if(countBraces(buffer,'}')==0&&countBraces(buffer,'{')==0){
        //printf("finally: %s\n",buffer);
        char key[20], val[20];
        sscanf(buffer, "%[^:]:%s",key,val);
        return createStringNode(key,val);
    }
    else{
        //printf("before trimmed: %s\n",buffer);
        trimChars(buffer,'{','}');
        //printf("after trimmed: %s\n",buffer);
        JsonNode** Obj_level_0=(JsonNode**)malloc(100* sizeof(JsonNode*));
        //---------初始化temp----------------
        char *temp = (char *)malloc(strlen(buffer) * sizeof(char));
        temp[0] = '\0';
        int obj_ct_level_0=0;
        int inside_brackets = 0;
        int count=0;
        int length = strlen(buffer);
        int start = 0;
        //---------------------------------
        for (int i = 0; i <= length; i++) {
            if (buffer[i] == ',' || buffer[i] == '\0') {
                int tokenLength = i - start;
                char *token = (char *)malloc((tokenLength + 1) * sizeof(char));
                strncpy(token, &buffer[start], tokenLength);
                token[tokenLength] = '\0';
                //-------------------------------------------------
                if(strstr(token, "{")&&strstr(token, "}")){//,{A:B},...
                    //printf("case1 The token %s\n", token);
                    Obj_level_0[obj_ct_level_0]=partition(token);
                    obj_ct_level_0++;
                    count++;
                }
                else if (strstr(token, "{") != NULL) {
                    inside_brackets+=countBraces(token,'{');
                    strcat(temp, token);
                    strcat(temp, ",");
                    //printf("case2 The temp %s\n", temp);
                }
                else if (strstr(token, "}") != NULL&&inside_brackets>1) {
                    inside_brackets-=countBraces(token,'}');
                    strcat(temp, token);
                    if(inside_brackets==1){
                        strcat(temp, ",");
                    }
                    else if(inside_brackets==0){
                        //printf("case3 the temp %s\n", temp);
                        //---------------------
                        Obj_level_0[obj_ct_level_0]=partition(temp);
                        obj_ct_level_0++;
                        count++;
                        //---------------------
                        temp[0] = '\0';
                    }
                }
```

```
            else if (strstr(token, "}") != NULL&&inside_brackets==1) {
                inside_brackets--;
                strcat(temp, token);
                //printf("case6 the temp %s\n", temp);
                Obj_level_0[obj_ct_level_0]=partition(temp);
                obj_ct_level_0++;
                count++;
                temp[0] = '\0';
            }
            else if (inside_brackets) {
                strcat(temp, token);
                strcat(temp, ",");
                //printf("case4 The temp %s\n", temp);
            }
            else {
                //printf("case5 The token %s\n", token);
                Obj_level_0[obj_ct_level_0]=partition(token);
                obj_ct_level_0++;
                count++;
            }
            //-----------------------------------------------------
            start = i + 1;
        }
    }
    Obj_level_0[obj_ct_level_0] = NULL; // Mark the end of the object array
    JsonNode* node=createObjectNode("Object_key", Obj_level_0);
    return node;
}
}
```

d. Insert function:以遞迴方式查找到要插入的 key-value pair 後將欲插入的 key-value pair 放入。

```
void insert(JsonNode* node,char *tar_key,char *tar_value,char *key_to_insert,char *value_to_insert,int *found_bool,JsonNode* pre_level_obj,int cur_level
    switch (node->type) {
        case STRING:
            if(strcmp(node->key,tar_key)==0&&strcmp(node->value.stringValue,tar_value)==0&&node->valid_bool==1){
                (*found_bool)=1;
                int i_inserted=cur_level_i+1;
                JsonNode** obj_arr=(JsonNode**)malloc(100* sizeof(JsonNode*));

                for(int j=0;j<i_inserted;j++){
                    obj_arr[j]=pre_level_obj->value.objectValue[j];
                }
                obj_arr[i_inserted]=createStringNode(key_to_insert,value_to_insert);

                for(int j=cur_level_i+1;pre_level_obj->value.objectValue[j] != NULL;j++){
                    obj_arr[j+1]=pre_level_obj->value.objectValue[j];
                }
                pre_level_obj->value.objectValue =obj_arr;
                //printJsonTree(pre_level_obj,0);
            }
            break;
        case OBJECT:
            for (int i = 0; node->value.objectValue[i] != NULL; i++) {
                insert(node->value.objectValue[i],tar_key,tar_value,key_to_insert,value_to_insert,found_bool,node,i);
            }
            break;
    }
}
```

e. printJsontree function:基本上也是用遞迴方式把所有 node 都印出來

```
void printJsonTree(JsonNode* node, int level) {
    int obj_valid=0;
    if(node->valid_bool==1){
        for (int i = 0; i < level; i++) {
            printf("  ");
        }
        printf("%s: ", node->key);//key就直接印出來
    }

    switch (node->type) {
        case STRING:
            if(node->valid_bool==1){
                printf("\"%s\"\n", node->value.stringValue);
            }
            break;
        case OBJECT:
            for(int i = 0; node->value.objectValue[i] != NULL; i++){
                if(node->value.objectValue[i]->valid_bool==1)obj_valid=1;
            }
            if(obj_valid==1)printf("Object\n");
            for (int i = 0; node->value.objectValue[i] != NULL; i++) {
                printJsonTree(node->value.objectValue[i], level + 1);
            }
            break;
    }
}
```

f. find function:以遞迴方式走過所有 node，並印出對應傳入 key 的 value:

```c
void get_value(JsonNode* node,char *key,int *found_bool){
    switch (node->type) {
        case STRING:
            if(strcmp(node->key,key)==0&&node->valid_bool==1){
                printf("the value to %s is \"%s\"\n", node->key, node->value.stringValue);
                (*found_bool)=1;
            }
            break;
        case OBJECT:
            for (int i = 0; node->value.objectValue[i] != NULL; i++) {
                get_value(node->value.objectValue[i],key,found_bool);
            }
            break;
    }
}
```

g. delete function:以遞迴方式走過所有 node，並刪除對應傳入 key 的 key-value pair:

```c
void delete_pair(JsonNode* node,char *key,int *found_bool){
    switch (node->type) {
        case STRING:
            if(strcmp(node->key,key)==0&&node->valid_bool==1){
                node->valid_bool=0;
                (*found_bool)=1;
            }
            break;
        case OBJECT:
            for (int i = 0; node->value.objectValue[i] != NULL; i++) {
                delete_pair(node->value.objectValue[i],key,found_bool);
            }
            break;
    }
}
```

4. 編譯與執行結果

編譯方式:

gcc -c redis_dll.c -o redis_dll.o

gcc -c redis_str.c -o redis_str.o

gcc -c redis_ss.c -o redis_ss.o

gcc -o main main.c redis_dll.o redis_str.o redis_ss.o

在 WSL Ubuntu 上執行的結果

```
eason@LAPTOP-Q69P3FAE:~$ cd /mnt/c/Users/user/DS_HW4
eason@LAPTOP-Q69P3FAE:/mnt/c/Users/user/DS_HW4$ gcc main.c -o main
eason@LAPTOP-Q69P3FAE:/mnt/c/Users/user/DS_HW4$ ./main
Please input command :
create json_1 {A:B,C:E,{Q:R,R:Q,{W:Q,P:Z}},D:A,A:T}

Please input command :
show json_1
json_1:
Object_key: Object
  A: "B"
  C: "E"
  Object_key: Object
    Q: "R"
    R: "Q"
    Object_key: Object
      W: "Q"
      P: "Z"
  D: "A"
  A: "T"
```

使用 create 建立 json_1

使用 show 查看 json_1 的內容，json tree 內容與結構正確。

```
Please input command :
show json_3
json_3 doesn't exist
```

使用 show 查看 json_3 的內容，json_3 未被建立，故不存在。

```
Please input command :
insert json_1 A : B U : V
inserted successfully

Please input command :
show json_1
json_1:
Object_key: Object
  A: "B"
  U: "V"
  C: "E"
  Object_key: Object
    Q: "R"
    R: "Q"
    Object_key: Object
      W: "Q"
      P: "Z"
  D: "A"
  A: "T"
```

使用 insert 在 json_1 中 pair: A-B 的後面插入 key 為 U，value 為 V 的 pair

使用 show 查看 json_1 中插入的結果是否正確，執行結果顯示正確。

```
Please input command :
find json_1 U
the value to U is "V"

Please input command :
delete json_1 Q
deleted successfully
```

使用 find 查看 json_1 中 key 為 U 的 pair 對應的 value 為 V，也就是我們剛插入的 pair，故功能正確。

刪除 json_1 中 key 為 Q 的 pair

```
Please input command :
show json_1
json_1:
Object_key: Object
  A: "B"
  U: "V"
  C: "E"
  Object_key: Object
    R: "Q"
    Object_key: Object
      W: "Q"
      P: "Z"
  D: "A"
  A: "T"

Please input command :
create json_2 {M:B,T:E,W:Q,{Q:R,R:Q,{W:Q,P:Z},D:A,B:A},A:T,M:N}
```

使用 show 查看 json_1 中 key 為 Q 的 pair 是否正確被刪除:結果正確

使用 create 建立 json_2

```
Please input command :
show json_2
json_2:
Object_key: Object
  M: "B"
  T: "E"
  W: "Q"
  Object_key: Object
    Q: "R"
    R: "Q"
    Object_key: Object
      W: "Q"
      P: "Z"
    D: "A"
    B: "A"
  A: "T"
  M: "N"
```

使用 show 查看建立的 json_2，結果正確。

使用 insert 在 json_2 中 pair: T-E 的後面插入 key 為 PP，value 為 MN 的 pair

```
Please input command :
insert json_2 T:E PP:MN
inserted successfully
```

```
Please input command :
show json_2
json_2:
Object_key: Object
  M: "B"
  T: "E"
  PP: "MN"
  W: "Q"
  Object_key: Object
    Q: "R"
    R: "Q"
    Object_key: Object
      W: "Q"
      P: "Z"
    D: "A"
    B: "A"
  A: "T"
  M: "N"
```

使用 find 查看 json_2 中 key 為 PP，value 為 MN 的 pair 是否被正確插入在對的位置，結果正確。

使用 insert 在 json_2 中 pair: Q-R 的後面插入 key 為 Inserted，value 為 HIHIHI 的 pair

```
Please input command :
insert json_2 Q:R Inserted:HIHIHI
inserted successfully
```

```
Please input command :
show json_2
json_2:
Object_key: Object
  M: "B"
  T: "E"
  PP: "MN"
  W: "Q"
  Object_key: Object
    Q: "R"
    Inserted: "HIHIHI"
    R: "Q"
    Object_key: Object
      W: "Q"
      P: "Z"
    D: "A"
    B: "A"
  A: "T"
  M: "N"
```

使用 find 查看 json_2 中 key 為 Inserted，value 為 HIHIHI 的 pair 是否被正確插入在對的位置，結果正確。

```
Please input command :
delete json_2 A
deleted successfully
```

刪除 json_1 中 key 為 A 的 pair

```
Please input command :
show json_2
json_2:
Object_key: Object
  M: "B"
  T: "E"
  PP: "MN"
  W: "Q"
  Object_key: Object
    Q: "R"
    Inserted: "HIHIHI"
    R: "Q"
    Object_key: Object
      W: "Q"
      P: "Z"
    D: "A"
    B: "A"
  M: "N"
```

使用 show 查看 json_1 中 key 為 A 的 pair 是否正確被刪除:結果正確

```
Please input command :
find json_1 R
the value to R is "Q"

Please input command :
find json_2 R
the value to R is "Q"

Please input command :
find json_2 A
no value corresponds to A
```

使用 find 查看 json_1 和 json_2 中 key 為 R 的 pair 對應的 value 為 Q，故功能正確。

使用 find 查看 json_2 中 key 為 A 的 pair 對應的 value，但由之前的結果可知該 pair 不存在，故功能正確

```
Please input command :
create json_3 {II:B,VQ:E,BN:Q,{EA:R,UY:Q,{M:Q,Z:DC},YT:bv},{O:U,B:FD,{R:U,Q:PO},A:B},A:T,M:
N}

Please input command :
show json_3
json_3:
Object_key: Object
  II: "B"
  VQ: "E"
  BN: "Q"
  Object_key: Object
    EA: "R"
    UY: "Q"
    Object_key: Object
      M: "Q"
      Z: "DC"
    YT: "bv"
```

使用 create 建立一個更複雜的 json_3 並印出，功能正確

```
  Object_key: Object
    O: "U"
    B: "FD"
    Object_key: Object
      R: "U"
      Q: "PO"
    A: "B"
  A: "T"
  M: "N"
```

5. 說明:

基本上這個作業真的蠻不容易的，我寫了好久 QQ，基本上每個功能都牽扯到 recursion 的使用，後來發現這次的 task 如果完全只使用 linked list 而不要用 array 存取，並且在 node 的定義中多加入一個 parrent pointer 指向父節點的話可能會再容易一些，尤其是建立 json tree 的地方，不過再 printJsontree 的部分就會比較困難就是，所以這份作業的難度還是有的。

這個作業最困難的地方是 json 格式的解析(有一堆{}和逗號要一直判斷和去除，strtok 不知道為何在遞迴使用時跑一跑會 segmentation error，後來只好參考別人手刻好的字串切割 function),C 語言的字串切割真的很不容易，又時常會有一些很怪的 bug。

感謝助教批改，祝助教新年快樂!