NAME:	Jinyi Xia
STUDENT ID:	2021212057
CLASS NUMBER:	2021211802
CONTAINER NUMBER:	df492137f3eca5844e1f8c7dce8b55741165f498417aafe88d30c2be6816f3d1

REPORT ON LAB 6

1 Implementation of Json's Data Structure

Json's data structure is implemented in `json.h` as follows.

```
1  typedef enum {
2      JSON_NULL,
3      JSON_BOOLEAN,
4      JSON_INTEGER,
5      JSON_NUMBER,
6      JSON_STRING,
7      JSON_ARRAY,
8      JSON_OBJECT
9  } JsonType;
10
11 typedef struct JsonValue JsonValue;
12
13 typedef struct {
14     size_t size;
15     size_t capacity;
16     JsonValue *values;
17 } JsonArray;
18
19 typedef struct {
20     char *key;
21     JsonValue *value;
22 } JsonObjectMember;
23
24 typedef struct {
25     size_t size;
26     size_t capacity;
27     JsonObjectMember *members;
28 } JsonObject;
29
30 typedef union {
31     int boolean;
32     long long integer;
33     double number;
34     char *string;
35     JsonArray array;
36     JsonObject object;
37 } JsonUnion;
```

```
38
39 struct JsonValue {
40     JsonType type;
41     JsonUnion value;
42 };
```

JsonValue is the basic data structure of Json. It contains a `JsonType` and a `JsonUnion`.

JsonType is an enumeration of all possible types of Json.

JsonUnion is a union of all possible values of Json.

JsonArray is a dynamic array of `JsonValues`.

JsonObjectMember is a key-value pair of `JsonValues`.

JsonObject is a dynamic array of `JsonObjectMembers`.

2 Test availability of Json's Data Structure

The availability of Json's data structure is tested in `main.c` as follows.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include "json.h"
4
5  int main() {
6      // Create a JSON object
7      JsonValue json;
8      json.type = JSON_OBJECT;
9      json.value.object.size = 0;
10     json.value.object.capacity = 10;
11     // Allocate memory for members
12     json.value.object.members = malloc(sizeof(JsonObjectMember) *
           json.value.object.capacity);
13     // Add a member of type string
14     json.value.object.members[json.value.object.size].key = "Name";
15     json.value.object.members[json.value.object.size].value =
           malloc(sizeof(JsonValue));
16     json.value.object.members[json.value.object.size].value->type = JSON_STRING;
17     json.value.object.members[json.value.object.size].value->value.string = "Jinyi
           Xia";
18     json.value.object.size++;
19     // Add a member of type integer
20     json.value.object.members[json.value.object.size].key = "Student ID";
21     json.value.object.members[json.value.object.size].value =
           malloc(sizeof(JsonValue));
22     json.value.object.members[json.value.object.size].value->type = JSON_INTEGER;
23     json.value.object.members[json.value.object.size].value->value.integer =
           2021212057;
24     json.value.object.size++;
```

```
25      // Add another member of type integer
26      json.value.object.members[json.value.object.size].key = "Class";
27      json.value.object.members[json.value.object.size].value =
            malloc(sizeof(JsonValue));
28      json.value.object.members[json.value.object.size].value->type = JSON_INTEGER;
29      json.value.object.members[json.value.object.size].value->value.integer =
            2021211802;
30      json.value.object.size++;
31      // Print the JSON object
32      json_print(json);
33      puts("");
34      return 0;
35  }
```

The output is as follows.

```
root@df492137f3ec:/mnt/Workspace/lab6# make
cc -o json main.c json.c
root@df492137f3ec:/mnt/Workspace/lab6# ./json
{"Name": "Jinyi Xia", "Student ID": 2021212057, "Class": 2021211802}
root@df492137f3ec:/mnt/Workspace/lab6#
```

Figure 1: Output of `main.c`