

# INTRODUCTION TO DATA STRUCTURES

Leetcode: 20. Valid Parentheses & 66. Plus One in C

HW0\_D84099084\_賴澍雨

# 20. Valid Parentheses

## 提交畫面截圖

Status	Language	Runtime	Memory	Notes
Accepted	C	0 ms	5.3 MB	a few seconds ago
Accepted	C	0 ms	5.3 MB	a few seconds ago
Accepted	C	2 ms	5.3 MB	a minute ago
Accepted	C	0 ms	5.7 MB	3 minutes ago
Accepted	C	4 ms	5.8 MB	4 minutes ago

Accepted  
submitted at 10:04:00

Runtime: 0 ms  
Memory: 5.68 MB  
Beats 100.00% of users with C



```
#include <stdio.h> // Include for 'bool' type
#include <string.h> // Include for 'strlen'

bool isValid(char * s){
    char charStack[1000]; // 定義一個字串堆疊
    char *stackTop = charStack; // stackTop 指向堆疊頂端，初始為charStack 首地址
    char *lastPtr = lastPtr; // lastPtr 指向堆疊中最後一個元素，初始為charStack 首地址
    int length = strlen(s); // 計算字串長度
    // ... (rest of the code) ...
}
```

More challenges  
22. Generate Parentheses  
32. Longest Valid Parentheses  
301. Remove Invalid Parentheses

Submit your solution here

Select a problem tag

```
# Code
C v Auto
1 #include <stdio.h> // Include for 'bool' type
2 #include <string.h> // Include for 'strlen'
3
4 // Function to check if the input string 's' contains valid parentheses
5 bool isValid(char * s){
6     char charStack[1000]; // Stack to keep track of opening parentheses
7     int stackTop = 0; // Pointer to the top of the stack
8     int length = strlen(s); // Calculate the length of the input string once to improve efficiency
9
10    // Iterate over each character in the input string
11    for(int i = 0; i < length; i++){ // Using pre-calculated length for efficiency
12        switch(s[i]){ // Switch on the current character
13            case '(': // If it's an opening parenthesis,
14                // opening square bracket,
15                case '[': // opening curly brace,
16                    charStack[stackTop] = s[i]; // push it onto the stack
17                    stackTop++; // and increase the stack pointer
18                    break;
19
20            case ')': // If it's a closing parenthesis
21                if(stackTop == 0 || charStack[stackTop-1] != '(') return false; // Check for matching opening, if not found, return false
22                stackTop--; // If found, pop from stack by decreasing stack pointer
23                break;
24
25            case ']': // Similar check for square brackets
26                if(stackTop == 0 || charStack[stackTop-1] != '[') return false;
27                stackTop--;
28                break;
29
30            case '}': // Similar check for curly braces
31                if(stackTop == 0 || charStack[stackTop-1] != '{') return false;
32                stackTop--;
33                break;
34
35            default:
36                // If input contains characters other than valid parentheses, consider the string invalid
37                return false;
38        }
39    }
40
41    // After processing all characters, if the stack is empty, then the parentheses are valid
42    return stackTop == 0; // Return true if stack is empty (i.e., stackTop is 0), else false
43 }
```

程式碼與註解

Speed to Beat

Testcase: 1 / 3  
Test Result

Accepted Runtime: 6 ms

Case 1 Case 2 Case 3

Input

s =  
"()"

Output

true

Expected

true

## 20. Valid Parentheses

### 程式碼概念說明

#### 1. 使用的資料結構

- 字符棧 ( Stack )：字符棧是這個程式的核心資料結構，用於臨時存儲遇到的開括號 ( 、 [ 、 { ，直到相對應的閉括號 ) 、 ] 、 } 被找到。
- 棧是一種後進先出 ( LIFO ) 的資料結構，最後加入的元素將是第一個被移除。

#### 2. 程式邏輯與演算法

- 程式先計算輸入字符串的長度，以避免在每次迴圈中重複計算。
- 使用 for 迴圈和 pointer 'inStrPtr' 逐字符遍歷輸入字符串。
- 對於每個字符，使用 switch 語句判斷其類型：
  - 如果是開括號 ( ( 、 [ 、 { )，則將其推入棧中，並將棧頂 pointer 'stackPtr' 向前移動。
  - 如果是閉括號 ( ) 、 ] 、 } )，則檢查棧頂元素是否為對應的開括號。如果是，則將棧頂元素彈出 ( 即將 stackPtr 向後移動 )。
  - 如果不是或者棧為空 ( stackPtr == charStack )，則返回 false，表示括號不匹配。
- 循環結束後，如棧不為空 ( 即 stackPtr 不指向初始位置 charStack )，則有未匹配開括號，返回 false。否則所有括號正確配對且棧為空，返回 true。

#### 3. pointer 運作

- pointer 'stackPtr'：這是一個指向棧頂的 pointer，用於添加或移除棧中的元素。當有新的開括號被推入棧時，stackPtr 向前移動；當找到匹配的閉括號時，stackPtr 向後移動，以彈出開括號。
- pointer 'inStrPtr'：這是一個指向當前正在處理的輸入字符串字符的 pointer。使用它來遍歷整個字符串，而無需修改原始字符串 inStr 的內容。

# 66. Plus One

## 提交畫面截圖

Problem List

Accepted

Editorial

Solutions

Submissions

Status	Language	Runtime	Memory	Notes
Accepted	C	0 ms	5.8 MB	
Accepted	C	2 ms	5.6 MB	
Accepted	C	4 ms	5.6 MB	
Accepted	C	0 ms	5.7 MB	

Accepted

494099084 submitted at Mar 03, 2024 21:37

Editorial

Solution

Runtime

0 ms

Beats 100.00% of users with C

Memory

5.78 MB

Beats 58.89% of users with C

Code | C

```
int* plusOne(int* digits, int digitSize, int* returnSize) {  
    // Carry 變量用於表示在加一操作中的進位  
    int carry = 1; // 初始化進位為1，因為我們要對數組最後一位加1  
    int* result = (int*)malloc(sizeof(int)*(digitSize + 1)); // 為結果數組分配記憶體  
    int* resultPtr = result + digitSize; // 初始化指向結果數組result的最後一位  
    int* digitsPtr = digits + digitSize; // 初始化指向輸入數組digits的最後一位  
  
    // 從數組末尾開始向前遍歷，將digits的值加上carry，然後進行進位計算，將每個數字加上 carry。  
    // View more
```

Write your notes here

Select related tags

0/5

Run

Submit

C Code

Auto

```
1 int* plusOne(int* digits, int digitSize, int* returnSize) {  
2     // Carry 變量用於表示在加一操作中的進位  
3     int carry = 1; // 初始化進位為1，因為我們要對數組最後一位加1  
4     int* result = (int*)malloc(sizeof(int)*(digitSize + 1)); // 為結果數組分配記憶體，由於最高位可能進位，所以分配 digitSize + 1 的空間  
5     int* resultPtr = result + digitSize; // 初始化指向結果數組result的最後一位  
6     int* digitsPtr = digits + digitSize; // 初始化指向輸入數組digits的最後一位  
7  
8     // 從數組末尾開始向前遍歷，將digits的值加上carry，然後進行進位計算，將每個數字加上 carry (初始為1，表示進位)  
9     for (int i = 0; i < digitSize; i++) {  
10         digitsPtr--; // 移動pointer到上一位數字  
11         *resultPtr = (*digitsPtr + carry) % 10; // 將每個位加上進位，並取模10得到該位的結果  
12         carry = (*digitsPtr + carry) / 10; // 計算進位，如果進位為1，則將進位值更新為1  
13         resultPtr--; // 移動pointer到上一位數字，準備接收下一位的結果  
14     }  
15  
16     if (carry > 0) { // 如果最高位有進位，需要擴展result數組一位  
17         *returnSize = digitSize + 1; // 更新returnSize為digitSize + 1  
18         *result = 1; // 將最高位設置為1，因為最高位必定是1  
19         for (int i = 1; i < *returnSize; i++) { // 從result的第二位開始將對應位置的數字初始化為0  
20             *(result + i) = 0; // 將其他位設置為0  
21         }  
22     } else { // 如果沒有進位，返回大小不變，不需要擴展result數組  
23         *returnSize = digitSize;  
24         result++; // 將result的pointer前移，以去掉最高位的0  
25     }  
26  
27     return result; // 返回結果數組的pointer  
28 }
```

程式碼與註解

Saved to local

Testcase

Test Result

Accepted

Runtime: 0 ms

Case 1

Case 2

Case 3

Input

digits =

[1,2,3]

Output

[1,2,4]

Expected

[1,2,4]

Contribute a testcase

# 66. Plus One

## 程式碼概念說明

### 1.使用的資料結構

- **整數(int) pointer** : digits 和 result 都是指向整數的 pointer，分別表示輸入的數字和計算結果。
- **動態分配array** : 通過 malloc 函數分配一個動態 array來存儲結果，大小為 digitsSize + 1 以應對可能的進位情況。

### 2.程式邏輯與演算法

- **進位處理** : 通過一個名為 carry 的變量來跟蹤是否需要進位。初始設為 1，因為我們要給最後一位數字加一。
- **反向遍歷** : 從 array最後一位開始遍歷，使用 pointer操作來更新數位值並處理進位。
- **結果擴展** : 如果在遍歷完成後仍有進位（即array的所有數位都是9），則需要將結果 array擴展一位，新的最高位設為 1，其他位設為 0。
- **返回值設置** : 通過 returnSize pointer參數返回結果 array的大小。

### 3.pointer運作

- **pointer算術** : 通過增加或減少 pointer值來移動 pointer，這使我們能夠在不修改原始 array的情況下遍歷 array。
- **賦值與更新** : 使用解引用操作符 \* 來訪問和更新 pointer指向的值。
- **結果pointer前移** : 如果沒有進位，結果 array的實際大小與輸入 array相同，因此將 result pointer前移，以指向正確的開始位置。