

【华为OD机考 统一考试机试C卷】火星文计算 (C++ Java JavaScript Python)

华为OD机考:统一考试 C卷 + D卷 + B卷 +A卷

目前在考C卷，经过两个月的收集整理，**C卷真题已基本整理完毕**

抽到原题的概率为2/3到3/3，**也就是最少抽到两道原题。请注意：大家刷完C卷真题，最好要把B卷的真题刷一下，因为C卷的部分真题来自B卷。**

另外订阅专栏还可以联系笔者开通在线 **OJ** 进行刷题，提高刷题效率。

真题目录：华为OD机考机试 真题目录 (C卷 + D卷 + B卷 + A卷) + 考点说明

专栏：2023华为OD机试(B卷+C卷+D卷) (C++JavaJSPy)

华为OD面试真题精选：华为OD面试真题精选

在线OJ：点击立即刷题，模拟真实机考环境

题目描述

已知火星使用的运算符为#、\$，其与地球人的等价公式如下：

$$x\#y = 2*x+3*y+4$$

$$x\$y = 3*x+y+2$$

1. 其中x、y是无符号整数
2. 地球人公式按C语言规则计算
3. 火星公式中，\$的优先级高于#，相同的运算符，按从左到右的顺序计算

现有一段火星人的字符串报文，请你来翻译并计算结果。

输入描述

火星字符串表达式（结尾不带回车换行）

输入的字符串说明：字符串为仅由无符号整数和操作符（#、\$）组成的计算表达式。

例如：123#4\$5#67\$78。

- 1. 用例保证字符串中，操作数与操作符之间没有任何分隔符。
- 2. 用例保证操作数取值范围为32位无符号整数。
- 3. 保证输入以及计算结果不会出现整型溢出。
- 4. 保证输入的字符串为合法的求值报文，例如：123#4\$5#67\$78
- 5. 保证不会出现非法的求值报文，例如类似这样字符串：

#4\$5 //缺少操作数

4\$5# //缺少操作数

4#\$5 //缺少操作数

4 \$5 //有空格

3+4-5*6/7 //有其它操作符

12345678987654321\$54321 //32位整数计算溢出

输出描述

根据输入的火星人字符串输出计算结果（结尾不带回车换行）。

用例

输入	7#6\$5#12
输出	226
说明	<div>7#6\$5#12</div> <div>=7#(3*6+5+2)#12</div> <div>=7#25#12</div>

$$=(2*7+3*25+4)\#12$$

$$=93\#12$$

$$=2*93+3*12+4$$

$$=226$$

解题思路：

1. **处理输入**：首先，读取用户输入的火星字符串表达式。
2. **遍历字符串**：遍历输入的字符串，根据字符类型（数字或运算符）执行相应的操作。
3. **处理数字**：当遇到数字字符时，提取完整的数字并将其转换为长整型。然后将数字压入栈中。这样可以确保在处理运算符时，操作数已经准备好。
4. **处理运算符**：当遇到运算符时，根据运算符类型执行相应的操作：
 - 如果遇到 `$` 运算符，首先弹出栈顶元素作为 `y`，然后提取紧跟在 `$` 后面的数字作为 `x`。接着计算 `x$y` 的值（`3 * y + x + 2`），并将结果压入栈中。这样可以确保 `$` 运算符的优先级高于 `#` 运算符。
 - 如果遇到 `#` 运算符，不执行任何操作，只将索引向后移动一位。这是因为在最后处理 `#` 运算符时，我们会按照从左到右的顺序计算。
5. **反转栈**：遍历字符串完成后，将栈中的元素反转。这是为了确保从栈底部开始计算 `#` 运算符。
6. **计算结果**：从反转后的栈底部开始计算 `x#y` 的值。首先弹出栈底元素作为初始结果。然后，当栈不为空时，弹出栈顶元素作为 `x`，并计算 `x#y` 的值（`2 * result + 3 * x + 4`），更新结果。最后返回计算结果。

C++

```
1 | #include <iostream> // 导入 iostream 库, 用于输入输出
2 | #include <stack> // 导入 stack 库, 用于存储操作数
3 | #include <string> // 导入 string 库, 用于处理字符串
4 | #include <cctype> // 导入 cctype 库, 用于处理字符
5 |
6 | long operate(const std::string& str) {
7 |
```

```

8 std::stack<long> stack; // 创建一个栈用于存储操作数
9 int i = 0; // 初始化遍历字符串的索引
10 while (i < str.length()) { // 遍历输入的字符串
11     char ch = str[i]; // 获取当前字符
12     if (isdigit(ch)) { // 如果当前字符是数字
13         int start = i; // 记录数字的起始位置
14         while (i < str.length() && isdigit(str[i])) { // 提取完整的数字
15             i++;
16         }
17         long num = std::stol(str.substr(start, i - start)); // 将提取到的数字字符串转换为长整型
18         stack.push(num); // 将数字压入栈中
19     } else {
20         if (ch == '$') { // 如果当前字符是 $ 运算符
21             long y = stack.top(); // 弹出栈顶元素作为 y
22             stack.pop();
23             i++; // 索引向后移动一位
24             int start = i; // 记录数字的起始位置
25             while (i < str.length() && isdigit(str[i])) { // 提取完整的数字
26                 i++;
27             }
28             long x = std::stol(str.substr(start, i - start)); // 将提取到的数字字符串转换为长整型
29             stack.push(3 * y + x + 2); // 计算 x$y 的值并压入栈中
30         } else if (ch == '#') { // 如果当前字符是 # 运算符
31             i++; // 索引向后移动一位
32         }
33     }
34 }

35 std::stack<long> reversedStack; // 创建一个新的栈, 用于反转操作数栈中的元素
36 while (!stack.empty()) { // 当操作数栈不为空时
37     reversedStack.push(stack.top()); // 将操作数栈的元素弹出并压入新栈中
38     stack.pop();
39 }

40
41 long result = reversedStack.top(); // 弹出新栈的栈顶元素作为初始结果
42 reversedStack.pop();

43
44 while (!reversedStack.empty()) { // 当新栈不为空时
45     long x = reversedStack.top(); // 弹出新栈的栈顶元素作为 x
46     reversedStack.pop();
47 }
48

```

```

49     result = 2 * result + 3 * x + 4; // 计算 x#y 的值并更新结果
50 }
51
52 return result; // 返回计算结果
53 }
54
55 int main() {
56     std::string str; // 创建一个字符串变量, 用于存储输入的火星字符串表达式
57     std::cin >> str; // 读取输入的火星字符串表达式
58     std::cout << operate(str) << std::endl; // 计算并输出结果
59     return 0;
60 }

```

JavaScript

```

1  const readline = require('readline'); // 导入 readline 模块, 用于读取输入
2
3  const rl = readline.createInterface({
4      input: process.stdin,
5      output: process.stdout,
6  });
7
8  // 读取输入的火星字符串表达式
9  rl.on('line', (str) => {
10     console.log(operate(str)); // 计算并输出结果
11     rl.close();
12 });
13
14 function operate(str) {
15     const stack = []; // 创建一个栈用于存储操作数
16     let i = 0; // 初始化遍历字符串的索引
17
18     while (i < str.length) { // 遍历输入的字符串
19         const ch = str.charAt(i); // 获取当前字符
20
21         if (/^\d/.test(ch)) { // 如果当前字符是数字
22             const start = i; // 记录数字的起始位置
23             while (i < str.length && /^\d/.test(str.charAt(i))) { // 提取完整的数字
24

```

```

24     i++;
25 }
26 const num = parseInt(str.substr(start, i), 10); // 将提取到的数字字符串转换为整型
27 stack.push(num); // 将数字压入栈中
28 } else {
29     if (ch === '$') { // 如果当前字符是 $ 运算符
30         const y = stack.pop(); // 弹出栈顶元素作为 y
31         i++; // 索引向后移动一位
32         const start = i; // 记录数字的起始位置
33         while (i < str.length && /\d/.test(str.charAt(i))) { // 提取完整的数字
34             i++;
35         }
36         const x = parseInt(str.substr(start, i), 10); // 将提取到的数字字符串转换为整型
37         stack.push(3 * y + x + 2); // 计算 x$y 的值并压入栈中
38     } else if (ch === '#') { // 如果当前字符是 # 运算符
39         i++; // 索引向后移动一位
40     }
41 }
42 }
43
44 const reversedStack = []; // 创建一个新的栈，用于反转操作数栈中的元素
45 while (stack.length > 0) { // 当操作数栈不为空时
46     reversedStack.push(stack.pop()); // 将操作数栈的元素弹出并压入新栈中
47 }
48
49 let result = reversedStack.pop(); // 弹出新栈的栈顶元素作为初始结果
50
51 while (reversedStack.length > 0) { // 当新栈不为空时
52     const x = reversedStack.pop(); // 弹出新栈的栈顶元素作为 x
53     result = 2 * result + 3 * x + 4; // 计算 x#y 的值并更新结果
54 }
55
56 return result; // 返回计算结果
57 }
58
59
60

```

```

1  import java.util.Scanner;
2  import java.util.Stack;
3
4  public class Main {
5      public static void main(String[] args) {
6          Scanner sc = new Scanner(System.in); // 创建 Scanner 对象, 用于读取输入
7          String str = sc.next(); // 读取输入的火星入字符串表达式
8          System.out.println(operate(str)); // 计算并输出结果
9      }
10
11     public static long operate(String str) {
12         Stack<Long> stack = new Stack<>(); // 创建一个栈用于存储操作数
13         int i = 0; // 初始化遍历字符串的索引
14         while (i < str.length()) { // 遍历输入的字符串
15             char ch = str.charAt(i); // 获取当前字符
16             if (Character.isDigit(ch)) { // 如果当前字符是数字
17                 int start = i; // 记录数字的起始位置
18                 while (i < str.length() && Character.isDigit(str.charAt(i))) { // 提取完整的数字
19                     i++;
20                 }
21                 long num = Long.parseLong(str.substring(start, i)); // 将提取到的数字字符串转换为长整型
22                 stack.push(num); // 将数字压入栈中
23             } else {
24                 if (ch == '$') { // 如果当前字符是 $ 运算符
25                     long y = stack.pop(); // 弹出栈顶元素作为 y
26                     i++; // 索引向后移动一位
27                     int start = i; // 记录数字的起始位置
28                     while (i < str.length() && Character.isDigit(str.charAt(i))) { // 提取完整的数字
29                         i++;
30                     }
31                     long x = Long.parseLong(str.substring(start, i)); // 将提取到的数字字符串转换为长整型
32                     stack.push(3 * y + x + 2); // 计算 x$y 的值并压入栈中
33                 } else if (ch == '#') { // 如果当前字符是 # 运算符
34                     i++; // 索引向后移动一位
35                 }
36             }
37         }
38
39         Stack<Long> reversedStack = new Stack<>(); // 创建一个新的栈, 用于反转操作数栈中的元素
40         while (!stack.isEmpty()) { // 当操作数栈不为空时
41

```

```

41         reversedStack.push(stack.pop()); // 将操作数栈的元素弹出并压入新栈中
42     }
43
44     long result = reversedStack.pop(); // 弹出新栈的栈顶元素作为初始结果
45
46     while (!reversedStack.isEmpty()) { // 当新栈不为空时
47         long x = reversedStack.pop(); // 弹出新栈的栈顶元素作为 x
48         result = 2 * result + 3 * x + 4; // 计算 x#y 的值并更新结果
49     }
50
51     return result; // 返回计算结果
52 }
53 }
54
55
56
57

```

Python

```

1  def operate(s):
2      stack = [] # 创建一个栈用于存储操作数
3      i = 0 # 初始化遍历字符串的索引
4      while i < len(s): # 遍历输入的字符串
5          ch = s[i] # 获取当前字符
6          if ch.isdigit(): # 如果当前字符是数字
7              start = i # 记录数字的起始位置
8              while i < len(s) and s[i].isdigit(): # 提取完整的数字
9                  i += 1
10             num = int(s[start:i]) # 将提取到的数字字符串转换为整型
11             stack.append(num) # 将数字压入栈中
12         else:
13             if ch == '$': # 如果当前字符是 $ 运算符
14                 y = stack.pop() # 弹出栈顶元素作为 y
15                 i += 1 # 索引向后移动一位
16                 start = i # 记录数字的起始位置
17                 while i < len(s) and s[i].isdigit(): # 提取完整的数字
18                     i += 1
19                 x = int(s[start:i]) # 将提取到的数字字符串转换为整型
20                 stack.append(3 * y + x + 2) # 计算 x$y 的值并压入栈中
21

```



```

41         elif ch == '#': # 如果当前字符是 # 运算符
42             i += 1 # 索引向后移动一位
43
44     reversed_stack = [] # 创建一个新的栈, 用于反转操作数栈中的元素
45     while stack: # 当操作数栈不为空时
46         reversed_stack.append(stack.pop()) # 将操作数栈的元素弹出并压入新栈中
47
48     result = reversed_stack.pop() # 弹出新栈的栈顶元素作为初始结果
49
50     while reversed_stack: # 当新栈不为空时
51         x = reversed_stack.pop() # 弹出新栈的栈顶元素作为 x
52         result = 2 * result + 3 * x + 4 # 计算 x#y 的值并更新结果
53
54     return result # 返回计算结果
55
56 # 读取输入的火星字符串表达式
57 input_str = input()
58 # 计算并输出结果
59 print(operate(input_str))
60
61
62

```

文章目录

华为OD机考:统一考试 C卷 + D卷 + B卷 +A卷

题目描述

输入描述

输出描述

用例

解题思路：

C++

JavaScript

Java

Python

机考真题 华为OD



CSDN @算法大师