



# 【华为OD机考 统一考试机试C卷】最长合法表达式 (C++ Java Python javaScript)

## 华为OD机考:统一考试 C卷 + D卷 + B卷 +A卷

2023年11月份, 华为官方已经将 华为OD机考: OD统一考试 (A卷 / B卷) 切换到 OD统一考试 (C卷) 和 OD统一考试 (D卷) 。根据考友反馈: 目前抽到的试卷为B卷或C卷/D卷, 其中C卷居多, 按照之前的经验C卷D卷部分考题会复用A卷/B卷题, 博主正积极从考过的同学收集C卷和D卷真题。可以先继续刷B卷, C卷和D卷的题目会放在现在大家购买的专栏内, 不需要重新购买, 请大家放心。

**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 华为OD机考B卷C卷华为OD机考华为OD机考B卷华为OD机试B卷华为OD机试C卷华为OD机考C卷华为OD机考D卷题目华为OD机考C卷/D卷答案华为OD机考C卷/D卷解析华为

OD机考C卷和D卷真题华为OD机考C卷和D卷题解

## 题目描述: 最长合法表达式 (本题分值200)

提取字符串中的最长合法简单数学表达式字符串长度最长的, 并计算表达式的值。如果没有返回 0

简单数学表达式只能包含以下内容

0-9 数字, 符号 $\pm^*$

说明:

- 1.所有数字, 计算结果都不超过 long
- 2.如果有多个长度一样的, 请返回第一个表达式的结果
- 3.数学表达式, 必须是最长的, 合法的
- 4.操作符不能连续出现, 如  $\pm+1$  是不合法的

### 输入描述

字符串

### 输出描述

表达式值

### 示例一

输入

1-2abcd

输出

-1

## 输入描述

字符串

## 输出描述

表达式值

## 用例

输入

1 | 1-2abcd

输出

1 | -1

## 解题思路

1. 首先，我们需要从输入的字符串中提取出所有的合法数学表达式。在这里，一个合法的数学表达式是指由数字和操作符（+、-、\*）组成的字符串，且操作符不能连续出现。我们通过遍历输入字符串的每个字符，并使用一个变量 `start` 来记录当前表达式的开始索引，来实现这一步。当我们遇到一个数字字符时，如果 `start` 为-1（表示我们还没有开始记录一个新的表达式），我们就将 `start` 设置为当前索引；当我们遇到一个操作符字符时，如果前一个字符不是操作符，我们就将当前索引加1；否则，我们就提取从 `start` 到当前索引的子字符串作为一个表达式，并将 `start` 重置为-1。如果我们遇到一个既不是数字也不是操作符的字符，并且 `start` 不为-1，我们也会提取从 `start` 到当前索引的子字符串作为一个表达式，并将 `start` 重置为-1。最后，如果 `start` 不为-1，我们会提取从 `start` 到字符串末尾的子字符串作为一个表达式。
2. 然后，我们需要对提取出的所有表达式按长度进行排序，并选择长度最长的表达式。我们使用Python的 `sorted` 函数来实现这一步，其中 `key=len` 表示按长度排序，`reverse=True` 表示按降序排序。
3. 接下来，我们需要计算选出的表达式的结果。我们首先使用正则表达式来分割表达式，得到一个由数字和操作符组成的列表。然后，我们遍历这个列表，先进行所有的乘法操作，然后进行所有的加法和减法操作。在这个过程中，我们使用一个变量 `result` 来记录当前的计算结果。
4. 最后，我们打印出计算结果。如果没有提取出任何表达式，我们就打印0。

**C++**

```
1  #include <iostream>
2  #include <string>
3  #include <vector>
4  #include <algorithm>
5  #include <cctype>
6  using namespace std;
7  // 提取合法表达式的函数
8  vector<string> extractExpressions(const string &line) {
9      vector<string> expressions;
10     int start = -1;
11
12     for (size_t i = 0; i < line.length(); ++i) {
13         char cur = line[i];
14
15         // 如果当前字符是数字
16         if (isdigit(cur)) {
17             // 如果开始索引为-1, 设置开始索引为当前索引
18             if (start == -1) {
19                 start = i;
20             }
21         } else if (start != -1 && (cur == '+' || cur == '-' || cur == '*')) {
22             // 如果当前字符是操作符, 并且前一个字符不是操作符
23             // 则将当前索引加1
24             if (i == 0 || !(line[i - 1] == '+' || line[i - 1] == '-' || line[i - 1] == '*')) {
25                 ++i;
26             } else {
27                 // 否则, 提取从开始索引到当前索引的子字符串作为一个表达式
28                 // 并将开始索引重置为-1
29                 expressions.push_back(line.substr(start, i - start));
30                 start = -1;
31             }
32         } else {
33             // 如果当前字符既不是数字也不是操作符
34             // 并且开始索引不为-1, 则提取从开始索引到当前索引的子字符串作为一个表达式
35             // 并将开始索引重置为-1
36             if (start != -1) {
37                 expressions.push_back(line.substr(start, i - start));
38                 start = -1;
39             }
40         }
41     }
```

```
41     }
42
43 // 如果开始索引不为-1, 则提取从开始索引到字符串末尾的子字符串作为一个表达式
44 if (start != -1) {
45     expressions.push_back(line.substr(start));
46 }
47
48 // 返回提取的表达式列表
49 return expressions;
50 }
51
52 // 计算表达式结果的函数
53 int calc(const string &str) {
54     vector<string> tokens;
55     string number;
56
57     for (char ch : str) {
58         // 如果当前字符是数字, 则将其添加到当前数字的末尾
59         if (isdigit(ch)) {
60             number += ch;
61         } else {
62             // 否则, 将当前数字和当前操作符添加到元素列表中
63             // 并开始构建新的数字
64             tokens.push_back(number);
65             tokens.push_back(string(1, ch));
66             number.clear();
67         }
68     }
69     // 将最后一个数字添加到元素列表中
70     tokens.push_back(number);
71
72     // 遍历元素列表, 先进行所有的乘法操作
73     for (size_t i = 0; i < tokens.size(); ++i) {
74         if (tokens[i] == "*" || tokens[i] == "/") {
75             int result = stoi(tokens[i - 1]) * stoi(tokens[i + 1]);
76             tokens[i - 1] = to_string(result);
77             tokens.erase(tokens.begin() + i, tokens.begin() + i + 2);
78             --i;
79         }
80     }
81 }
```

```
82
83 // 初始化结果为元素列表的第一个数字
84 int result = stoi(tokens[0]);
85 // 遍历元素列表, 进行所有的加法和减法操作
86 for (size_t i = 1; i < tokens.size(); i += 2) {
87     if (tokens[i] == "+") {
88         result += stoi(tokens[i + 1]);
89     } else {
90         result -= stoi(tokens[i + 1]);
91     }
92 }
93
94 // 返回计算结果
95 return result;
96 }
97
98 int main() {
99     // 创建一个字符串来处理输入
100     string line;
101     // 读取一行输入
102     getline(cin, line);
103
104     // 调用extractExpressions函数提取输入中的合法表达式
105     vector<string> expressions = extractExpressions(line);
106
107     // 使用C++的sort函数对表达式按长度进行排序
108     sort(expressions.begin(), expressions.end(), [](const string &s1, const string &s2) {
109         return s2.length() < s1.length();
110     });
111
112     // 如果表达式列表不为空, 则计算并打印最长表达式的结果
113     // 否则, 打印0
114     if (!expressions.empty()) {
115         cout << calc(expressions[0]) << endl;
116     } else {
117         cout << 0 << endl;
118     }
119
120     return 0;
}
```

## Java

```
1 import java.util.ArrayList;
2 import java.util.List;
3 import java.util.Scanner;
4
5 class Main {
6     public static void main(String[] args) {
7         // 创建一个Scanner对象来处理输入
8         Scanner in = new Scanner(System.in);
9         // 读取一行输入
10        String line = in.nextLine();
11        // 关闭Scanner对象
12        in.close();
13
14        // 调用extractExpressions方法提取输入中的合法表达式
15        List<String> expressions = extractExpressions(line);
16
17        // 使用Java 8的Lambda表达式对表达式按长度进行排序
18        expressions.sort((s1, s2) -> Integer.compare(s2.length(), s1.length()));
19
20        // 如果表达式列表不为空, 则计算并打印最长表达式的结果
21        // 否则, 打印0
22        if (!expressions.isEmpty()) {
23            System.out.println(calc(expressions.get(0)));
24        } else {
25            System.out.println(0);
26        }
27    }
28
29    // 提取合法表达式的方法
30    public static List<String> extractExpressions(String line) {
31        // 创建一个列表来存储提取的表达式
32        List<String> expressions = new ArrayList<>();
33        // 初始化开始索引为-1
34        int start = -1;
35
36        // 遍历输入字符串的每个字符
37        for (int i = 0; i < line.length(); i++) {
38            char cur = line.charAt(i);
39
```

```
40
41 // 如果当前字符是数字
42 if (Character.isDigit(cur)) {
43     // 如果开始索引为-1, 设置开始索引为当前索引
44     if (start == -1) {
45         start = i;
46     }
47 } else if (start != -1 && "+-*".contains(String.valueOf(cur))) {
48     // 如果当前字符是操作符, 并且前一个字符不是操作符
49     // 则将当前索引加1
50     if (!"+-*".contains(String.valueOf(line.charAt(i - 1)))) {
51         i++;
52     } else {
53         // 否则, 提取从开始索引到当前索引的子字符串作为一个表达式
54         // 并将开始索引重置为-1
55         expressions.add(line.substring(start, i));
56         start = -1;
57     }
58 } else {
59     // 如果当前字符既不是数字也不是操作符
60     // 并且开始索引不为-1, 则提取从开始索引到当前索引的子字符串作为一个表达式
61     // 并将开始索引重置为-1
62     if (start != -1) {
63         expressions.add(line.substring(start, i));
64         start = -1;
65     }
66 }
67
68 // 如果开始索引不为-1, 则提取从开始索引到字符串末尾的子字符串作为一个表达式
69 if (start != -1) {
70     expressions.add(line.substring(start));
71 }
72
73 // 返回提取的表达式列表
74 return expressions;
75 }
76
77 // 计算表达式结果的方法
78 public static int calc(String str) {
79
80
```



```
81 // 创建一个列表来存储表达式的每个元素 (数字或操作符)
82 List<String> tokens = new ArrayList<>();
83 // 创建一个StringBuilder对象来构建数字
84 StringBuilder sb = new StringBuilder();
85
86 // 遍历表达式的每个字符
87 for (char ch : str.toCharArray()) {
88     // 如果当前字符是数字, 则将其添加到当前数字的末尾
89     if (Character.isDigit(ch)) {
90         sb.append(ch);
91     } else {
92         // 否则, 将当前数字和当前操作符添加到元素列表中
93         // 并开始构建新的数字
94         tokens.add(sb.toString());
95         tokens.add(String.valueOf(ch));
96         sb.setLength(0);
97     }
98 }
99 // 将最后一个数字添加到元素列表中
100 tokens.add(sb.toString());
101
102 // 遍历元素列表, 先进行所有的乘法操作
103 for (int i = 0; i < tokens.size(); i++) {
104     if ("*".equals(tokens.get(i))) {
105         int result = Integer.parseInt(tokens.get(i - 1)) * Integer.parseInt(tokens.get(i + 1));
106         tokens.set(i - 1, String.valueOf(result));
107         tokens.remove(i);
108         tokens.remove(i);
109         i--;
110     }
111 }
112
113 // 初始化结果为元素列表的第一个数字
114 int result = Integer.parseInt(tokens.get(0));
115 // 遍历元素列表, 进行所有的加法和减法操作
116 for (int i = 1; i < tokens.size(); i += 2) {
117     if ("+".equals(tokens.get(i))) {
118         result += Integer.parseInt(tokens.get(i + 1));
119     } else {
120         result -= Integer.parseInt(tokens.get(i + 1));
121     }
122 }
```

```
121         }
122     }
123
124     // 返回计算结果
125     return result;
}
}
```

## javaScript

```
1  const readline = require('readline');
2
3  // 创建一个readline.Interface实例来处理输入
4  const rl = readline.createInterface({
5      input: process.stdin,
6      output: process.stdout
7  });
8
9  rl.on('line', (line) => {
10     // 调用extractExpressions函数提取输入中的合法表达式
11     let expressions = extractExpressions(line);
12
13     // 使用JavaScript的sort函数对表达式按长度进行排序
14     expressions.sort((s1, s2) => s2.length - s1.length);
15
16     // 如果表达式列表不为空, 则计算并打印最长表达式的结果
17     // 否则, 打印0
18     if (expressions.length > 0) {
19         console.log(calc(expressions[0]));
20     } else {
21         console.log(0);
22     }
23
24     rl.close();
25 });
26
27 // 提取合法表达式的函数
28 function extractExpressions(line) {
29     let expressions = [];
30     let start = -1;
```

```
31
32
33 for (let i = 0; i < line.length; i++) {
34     let cur = line.charAt(i);
35
36     // 如果当前字符是数字
37     if (!isNaN(cur)) {
38         // 如果开始索引为-1, 设置开始索引为当前索引
39         if (start === -1) {
40             start = i;
41         }
42     } else if (start !== -1 && "+-*.includes(cur)) {
43         // 如果当前字符是操作符, 并且前一个字符不是操作符
44         // 则将当前索引加1
45         if (!"+-*.includes(line.charAt(i - 1))) {
46             i++;
47         } else {
48             // 否则, 提取从开始索引到当前索引的子字符串作为一个表达式
49             // 并将开始索引重置为-1
50             expressions.push(line.substring(start, i));
51             start = -1;
52         }
53     } else {
54         // 如果当前字符既不是数字也不是操作符
55         // 并且开始索引不为-1, 则提取从开始索引到当前索引的子字符串作为一个表达式
56         // 并将开始索引重置为-1
57         if (start !== -1) {
58             expressions.push(line.substring(start, i));
59             start = -1;
60         }
61     }
62 }
63
64 // 如果开始索引不为-1, 则提取从开始索引到字符串末尾的子字符串作为一个表达式
65 if (start !== -1) {
66     expressions.push(line.substring(start));
67 }
68
69 // 返回提取的表达式列表
70 return expressions;
71 }
```

```
72
73 // 计算表达式结果的函数
74 function calc(str) {
75     let tokens = [];
76     let number = '';
77
78     for (let ch of str) {
79         // 如果当前字符是数字, 则将其添加到当前数字的末尾
80         if (!isNaN(ch)) {
81             number += ch;
82         } else {
83             // 否则, 将当前数字和当前操作符添加到元素列表中
84             // 并开始构建新的数字
85             tokens.push(number);
86             tokens.push(ch);
87             number = '';
88         }
89     }
90     // 将最后一个数字添加到元素列表中
91     tokens.push(number);
92
93     // 遍历元素列表, 先进行所有的乘法操作
94     for (let i = 0; i < tokens.length; i++) {
95         if (tokens[i] === "*") {
96             let result = parseInt(tokens[i - 1]) * parseInt(tokens[i + 1]);
97             tokens[i - 1] = String(result);
98             tokens.splice(i, 2);
99             i--;
100         }
101     }
102
103     // 初始化结果为元素列表的第一个数字
104     let result = parseInt(tokens[0]);
105     // 遍历元素列表, 进行所有的加法和减法操作
106     for (let i = 1; i < tokens.length; i += 2) {
107         if (tokens[i] === "+") {
108             result += parseInt(tokens[i + 1]);
109         } else {
110             result -= parseInt(tokens[i + 1]);
111         }
112     }
```

```
113     }
114
115     // 返回计算结果
    return result;
}
```

## Python

```
1 # 导入需要的模块
2 import re
3
4 # 提取合法表达式的函数
5 def extract_expressions(line):
6     # 创建一个列表来存储提取的表达式
7     expressions = []
8     # 初始化开始索引为-1
9     start = -1
10
11     # 遍历输入字符串的每个字符
12     for i in range(len(line)):
13         cur = line[i]
14
15         # 如果当前字符是数字
16         if cur.isdigit():
17             # 如果开始索引为-1, 设置开始索引为当前索引
18             if start == -1:
19                 start = i
20         elif start != -1 and cur in "+-*":
21             # 如果当前字符是操作符, 并且前一个字符不是操作符
22             # 则将当前索引加1
23             if line[i - 1] not in "+-*":
24                 i += 1
25         else:
26             # 否则, 提取从开始索引到当前索引的子字符串作为一个表达式
27             # 并将开始索引重置为-1
28             expressions.append(line[start:i])
29             start = -1
30     else:
31         # 如果当前字符既不是数字也不是操作符
32         # 并且开始索引不为-1, 则提取从开始索引到当前索引的子字符串作为一个表达式
33         expressions.append(line[start:i])
34         start = -1
35
36     return expressions
```

```
33     # 并将开始索引重置为-1
34     if start != -1:
35         expressions.append(line[start:i])
36         start = -1
37
38     # 如果开始索引不为-1, 则提取从开始索引到字符串末尾的子字符串作为一个表达式
39     if start != -1:
40         expressions.append(line[start:])
41
42     # 返回提取的表达式列表
43     return expressions
44
45 # 计算表达式结果的函数
46 def calc(expression):
47     # 使用正则表达式分割字符串, 得到数字和操作符
48     tokens = re.split('([-+*])', expression)
49     tokens = [token for token in tokens if token]
50
51     # 遍历元素列表, 先进行所有的乘法操作
52     while '*' in tokens:
53         index = tokens.index('*')
54         result = int(tokens[index - 1]) * int(tokens[index + 1])
55         tokens = tokens[:index - 1] + [str(result)] + tokens[index + 2:]
56
57     # 初始化结果为元素列表的第一个数字
58     result = int(tokens[0])
59     # 遍历元素列表, 进行所有的加法和减法操作
60     for i in range(1, len(tokens), 2):
61         if tokens[i] == '+':
62             result += int(tokens[i + 1])
63         else:
64             result -= int(tokens[i + 1])
65
66     # 返回计算结果
67     return result
68
69 # 读取一行输入
70 line = input()
71
72 # 调用extract_expressions函数提取输入中的合法表达式
73
```

```
74 | expressions = extract_expressions(line)
75 |
76 | # 使用Python的sorted函数对表达式按长度进行排序
77 | expressions = sorted(expressions, key=len, reverse=True)
78 |
79 | # 如果表达式列表不为空, 则计算并打印最长表达式的结果
80 | # 否则, 打印0
81 | if expressions:
82 |     print(calc(expressions[0]))
83 | else:
    |     print(0)
```

## 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: 最长合法表达式 (本题分值200)

输入描述

输出描述

用例

解题思路

C++

Java

javaScript

Python

