

定义

逆波兰式，又称**后缀表达式**，指的是操作符在其所控制的操作数后面的表达式。

举个例子， $1 + 2 * 3 - 4$ 这个表达式是我们熟悉的中缀表达式，那么其所对应的后缀表达式为： $1 2 3 * + 4 -$ 。

再来个复杂的例子： $1 * (2 + 3) / 5 - 4 / 2$ 其对应的后缀表达式为： $1 2 3 + * 5 / 4 2 / -$ （其中括号由于只是提升表达式**优先级**的作用，因此不放入后缀表达式中）。

二.逆波兰式的意义

为什么要将看似简单的中缀表达式转换为复杂的逆波兰式，原因就在于这个简单是相对我们人类的思维结构来说的，对计算机而言中序表达式是非常复杂的结构。相对的，逆波兰式在计算机看来却是比较简单易懂的结构。因为计算机普遍采用的内存结构是栈式结构，它执行先进后出的顺序。

三.逆波兰式的实现

1.方法

(1) 中缀表达式转化为后缀表达式

对于给出的中缀表达式，如何将其转化为后缀表达式呢？

第一，若遇到操作数则直接输出/存储。

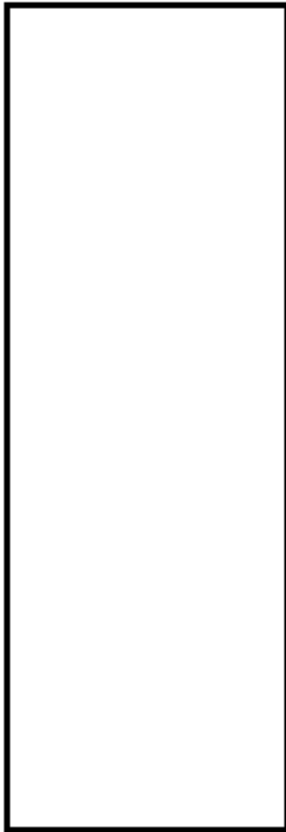
第二，遇到操作符，若此时栈为空或者操作符优先级高于栈顶，则入栈。

第三，若操作符的优先级低于或者等于栈顶，则出栈直至栈空或者优先级低于该操作符。

第四，遇到'(',其后的所有操作符（直至遇到'）'）按上述操作入栈或出栈；当遇到'）'时，将'('顶上的所有操作符出栈。

临时存放操作符
的栈

$1 * (2 + 3) / 5 - 4 / 2$



存放后缀表达式



(2) 由后缀表达式计算结果

第一，遇到操作数则入栈。

第二，遇到操作符则将栈顶的两个操作数出栈，其中第一个数为右操作数，第二个数为左操作数。

第三，计算结果并将计算的结果入栈。

第四，最后栈顶的结果即为所计算的结果。

1 2 3 + * 5 / 4 2 / -

临时存放
操作数的栈



2.代码实现

```
1  #include <iostream>
2  #include <string>
3  #include <stack>
4  #include <vector>
5  using namespace std;
6
7  string trans(string& s)
8  {
9      string operand;
10     stack<char> Operator;
11     int flag = 0; // 记录括号优先级
12     for (const auto& e : s)
13     {
14         if (e == '(')
15         {
16             Operator.push(e);
17             flag = 1;
18             continue;
19         }
20         if (e == ')')
```

```

21         flag = 0;
22         while (Operator.top() != '(')
23         {
24             operand.push_back(Operator.top());
25             Operator.pop();
26         }
27         Operator.pop();
28         continue;
29     }
30     //操作符
31     if (e == '+' || e == '-' || e == '*' || e == '/')
32     {
33         if (flag == 1)
34         {
35             if (Operator.top() == '(')
36             {
37                 Operator.push(e);
38             }
39             else if ((e == '*' || e == '/') && (Operator.top() == '*' || Operator.top() == '/'))
40             {
41                 Operator.push(e);
42             }
43             else //操作符的优先级低于或等于栈顶操作符则出栈，直至
44             {
45                 while (Operator.top() != '(')
46                 {
47                     operand.push_back(Operator.top());
48                     Operator.pop();
49                 }
50                 Operator.push(e);
51             }
52         }
53     }
54     else if (Operator.empty()) //栈空就入栈
55     {
56         Operator.push(e);
57     }
58     //操作符的优先级高于栈顶操作符，入栈
59     else if ((e == '*' || e == '/') && (Operator.top() == '+' || Operator.top() == '-'))
60     {
61         Operator.push(e);
62     }
63     else //操作符的优先级低于或等于栈顶操作符则出栈，直至栈空或遇到 '('
64     {
65         while (!Operator.empty())
66         {
67             operand.push_back(Operator.top());
68         }

```

```

69         Operator.pop();
70     }
71     Operator.push(e);
72 }
73 }
74 //操作数
75 else
76 {
77     operand.push_back(e);
78 }
79 }
80 while (!Operator.empty())
81 {
82     operand.push_back(Operator.top());
83     Operator.pop();
84 }
85 return operand;
86 }
87 int evalRPN(const string& s)
88 {
89     stack<char> operand;
90     int left = 0, right = 0;
91     for (const auto& e : s)
92     {
93         if (e == '+' || e == '-' || e == '*' || e == '/')
94         {
95             switch (e)
96             {
97             case '+':
98                 right = operand.top();
99                 operand.pop();
100                 left = operand.top();
101                 operand.pop();
102                 operand.push(left + right);
103                 break;
104             case '-':
105                 right = operand.top();
106                 operand.pop();
107                 left = operand.top();
108                 operand.pop();
109                 operand.push(left - right);
110                 break;
111             case '*':
112                 right = operand.top();
113                 operand.pop();
114                 left = operand.top();
115                 operand.pop();

```

```

116         operand.push(left * right);
117         break;
118     case '/':
119         right = operand.top();
120         operand.pop();
121         left = operand.top();
122         operand.pop();
123         operand.push(left / right);
124         break;
125     }
126 }
127 else//操作数
128 {
129     operand.push(e - '0');
130 }
131 }
132 return operand.top();
133 }
134
135 int RPN(const string& str)
136 {
137     //1. 中缀表达式转化为后缀表达式
138     string s(str);
139     s = trans(s);
140     //2. 后缀表达式计算答案
141     return evalRPN(s);
142 }
143
144 int main()
145 {
146     string s("1*(2*3+5)/5-4/2");
147     int ret = RPN(s);
148     cout << "ret:" << ret << endl;
149     return 0;
150 }
151

```

结果:

```

string s("1*(2*3+5)/5-4/2");
int ret = RPN(s);
cout << "ret:" << ret << endl;
return 0;

```

C:\WINDOWS\system32\cmd.exe

ret:0
请按任意键继续. . .