

# 编译原理实验

## 实验二：将算术表达式转换成语法树形式

姓名：陈家豪

班级：2018 级计科 1 班

学号：18308013

# 1 实验要求

请用 C++ 编写程序，将表达式转化为语法树形式。语法定义如下：

- 表达式:= 表达式 + 项
- 表达式:= 表达式 - 项
- 表达式:= 项
- 项:= 项 \* 因子
- 项:= 项 / 因子
- 项:= 因子
- 因子:= 正整数
- 因子:= (表达式)

## 2 算法原理与描述

本次试验我们需要将算术表达式转换为语法树形式。我们可以发现语法树根据算术表达式的运算顺序进行组织。在上次实验中我们实现了从算术表达式到逆波兰表达式的转换，而逆波兰表达式是对语法树的后序遍历。因此，在得到算术表达式后，我们可以先把算术表达式转换成逆波兰表达式，然后使用栈把后序遍历形式的逆波兰表达式转换成语法树形式。

在本次实验中，我将对算术表达式的相关操作封装成类 *Expression*，并在算术表达式到逆波兰表达式的转换过程中添加了对负数的支持和非法输入的检测。具体来说，对于负数，我通过在负号前添加 0 来保持兼容性（如  $-2 * 3$  变为  $0 - 2 * 3$ ）；而当识别到非法输入时会停止转换并输出错误信息。算法具体描述如下：

1. 获取算术表达式；
2. 划分算术表达式，形式为 *vector < string >*。每个 *string* 代表一个操作数（变量、常数等）或一个操作符，按顺序排列；
3. 使用实验 1 的算法将划分后的算术表达式转换成逆波兰表达式，形式为 *vector < string >*。在转换过程中完成对正负号的处理和表达式合法性的判定，若合法，进行下一步，否则输出错误信息并退出；
4. 创建空栈，然后从左到右遍历逆波兰表达式的每个元素：
  - (a) 如果当前元素是操作数，那么创建二叉树节点，值为该操作数，左子树和右子树指针均为空。然后该节点直接压栈；
  - (b) 如果当前元素是运算符，那么创建二叉树节点，值为该运算符，右子树指针和左子树指针分别指向栈顶第一、二个节点。被指向的栈顶两个节点出栈，新节点压栈；
5. 遍历完成后，栈中只剩下一个节点，弹出即可得到表达式对应的语法树根节点；

### 3 关键实现代码

语法树的节点结构体 *node* 如下所示，是一个二叉结构：

```
1 struct node { // 树节点
2     node *left = nullptr;
3     node *right = nullptr;
4     string value;
5 };
```

*Expression* 类结构如下所示：

```
1 class Expression {
2     private:
3         int GetPriority(char c); // 获取字符c的优先级
4         int GetPriority(string str); // 获取字符串str的优先级
5         void SplitExpression(); // 对表达式字符串expression划分
6         bool GetPostfixExpression(); // 检测表达式合法性并转换成逆波兰表达式
7         bool JudgeOperator(string str); // 判断str是否为运算符
8     public:
9         string expression; // 算术表达式
10        vector<string> expVec; // 划分后的算术表达式
11        vector<string> posexpVec; // 逆波兰表达式
12        bool isLegal; // 表达式是否合法
13        Expression(string input) {
14            expression=input;
15        }
16        vector<string> transfer() { // 算术表达式→逆波兰表达式
17            SplitExpression();
18            if (GetPostfixExpression())
19                printf("Sussecfully Done!\n");
20            else
21                printf("Error: Incorrect Format!\n");
22            return posexpVec;
23        }
24        node *BuildTree(); // 获取语法树树根
25};
```

获取到原始算术表达式后，构建语法树的函数 *BuildTree()* 代码如下所示：

```
1 // 将后缀表达式vec转换为语法树
2 node* Expression::BuildTree() {
3     SplitExpression(); // 划分算术表达式
4     // 将算术表达式转换为逆波兰表达式，判断是否合法
5     if (!GetPostfixExpression()) {
6         printf("Error: Incorrect Format!\n");
7         return nullptr;
8     }
9     stack<node*> s;
10    for (int i=0; i<posexpVec.size(); i++) {
```

```

11     node *np = new node;
12     np->value=posexpVec[i];
13     // 如果为操作符
14     if (JudgeOperator(np->value)) {
15         np->right = s.top();
16         s.pop();
17         np->left = s.top();
18         s.pop();
19     }
20     s.push(np);
21 }
22 return s.top();
23 }

```

其中，函数 *SplitExpression()* 对原始算术表达式进行划分，代码如下所示：

```

1 // 按照操作数和操作符的形式划分算术表达式
2 void Expression::SplitExpression() {
3     for (int i=0; i<expression.length(); i++) {
4         int priority = GetPriority(expression[i]);
5         // 若为空白字符
6         if (priority== -1)
7             continue;
8         // 若为操作数
9         else if (priority == 0) {
10             string temp;
11             // 获取整个操作数
12             while (i<expression.length() && GetPriority(expression[i])==0) {
13                 temp.push_back(expression[i]);
14                 i++;
15             }
16             expVec.push_back(temp);
17             i--;
18         }
19         // 若为操作符
20         else {
21             string temp;
22             temp.push_back(expression[i]);
23             expVec.push_back(temp);
24         }
25     }
26 }

```

而函数 *GetPostfixExpression()* 将划分好的算术表达式转换成逆波兰表达式，并进行正负号的处理和合法性的检测，代码如下所示：

```

1 // 将expVec转换为后缀表达式vector
2 bool Expression::GetPostfixExpression() {
3     stack<string> s;

```

```

4   isLegal = true;
5   int preType;
6   for (int i=0; i<expVec.size(); i++) {
7       int len = expVec[i].length();
8       // 若长度为0，不合法
9       if (len==0) {
10          isLegal = false;
11          break;
12      }
13      // 若长度大于1，则为操作数
14      else if (len>1) {
15          // 若上一个也为操作数，不合法
16          if (i>=1 && preType==VAL) {
17              isLegal = false;
18              break;
19          }
20          else
21              preType = VAL;
22          bool flag = true;
23          // 判断操作数是否为数字开头的变量，若是则不合法
24          if (expVec[i][0]>='0'&&expVec[i][0]<='9') {
25              for (int j=1; j<len; j++) {
26                  if (expVec[i][j]>='0'&&expVec[i][j]<='9')
27                      continue;
28                  else {
29                      flag=false;
30                      break;
31                  }
32              }
33          }
34          // 合法，加入队列
35          if (flag)
36              posexpVec.push_back(expVec[i]);
37          // 不合法，over
38          else {
39              isLegal=false;
40              break;
41          }
42      }
43      else {
44          int prior = GetPriority(expVec[i][0]);
45          // 若为操作符 '+', '-', '*', '/' 且在最后一位，不合法
46          if (prior>0 && prior<4 && i==expVec.size()-1) {
47              isLegal = false;
48              break;
49          }
50          // 若为操作数
51          if (prior==0) {

```

```

52 // 若上一个也是操作数，不合法
53 if (i>=1 && preType==VAL) {
54     isLegal = false;
55     break;
56 }
57 preType = VAL;
58 posexpVec.push_back(expVec[i]);
59 }
60 // 若为 '('
61 else if (prior==1) {
62     // 若上一个为操作数或 ')', 不合法
63     if (i>=1 && (preType==VAL||preType==RB))
64     {
65         isLegal = false;
66         break;
67     }
68     preType = LB;
69     s.push(expVec[i]);
70 }
71 // 若为 '+' 或 '-'
72 else if (prior==2) {
73     // 若为操作数的正/负号，添0
74     if (i==0 || (i>0&&preType==LB))
75         posexpVec.push_back("0");
76     // 若上一个为 '+', '-', '*', 或 '/', 不合法
77     else if (preType==PLUS||preType==MINUS||preType==MUL||preType==DIV) {
78         isLegal = false;
79         break;
80     }
81     if (expVec[i][0]=='+')
82         preType = PLUS;
83     else if (expVec[i][0]=='-')
84         preType = MINUS;
85     // 弹出优先级大的操作符，再压栈
86     while (!s.empty() && GetPriority(s.top())>=GetPriority(expVec[i])) {
87         posexpVec.push_back(s.top());
88         s.pop();
89     }
90     s.push(expVec[i]);
91 }
92 // 若为 '*' 或 '/'
93 else if (prior==3) {
94     // 若在第一位出现，或者前一位不是操作数或 ')', 不合法
95     if (i==0 || (preType!=VAL&&preType!=RB)) {
96         isLegal = false;
97         break;
98     }
99     if (expVec[i][0]=='*')

```

```

100         preType = MUL;
101     else if (expVec[i][0]=='/')
102         preType = DIV;
103     // 弹出优先级大的操作符，再压栈
104     while (!s.empty() && GetPriority(s.top())>=GetPriority(expVec[i])) {
105         posexpVec.push_back(s.top());
106         s.pop();
107     }
108     s.push(expVec[i]);
109 }
110 // 若为')'
111 else if (prior==4) {
112     // 若前一位不是操作数，不合法
113     if (preType!=VAL) {
114         isLegal = false;
115         break;
116     }
117     preType = RB;
118     bool isFinish = false;
119     // 不断弹栈直到'('
120     while (!s.empty()) {
121         if (s.top()=="(") {
122             s.pop();
123             isFinish = true;
124             break;
125         }
126         posexpVec.push_back(s.top());
127         s.pop();
128     }
129     // 栈中没有匹配的'(', 不合法
130     if (!isFinish) {
131         isLegal = false;
132         break;
133     }
134 }
135 // 出若为其他符号，不合法
136 else {
137     isLegal = false;
138     break;
139 }
140 }
141 }
142 // 清空栈
143 while (isLegal&&!s.empty()) {
144     // 有多余'('时不合法
145     if (s.top()=="(")
146         isLegal = false;
147     posexpVec.push_back(s.top());

```

```

148     s.pop();
149 }
150 // 不合法时清空逆波兰表达式
151 if (!isLegal)
152     posexpVec.clear();
153 return isLegal;
154 }

```

其余代码与注释可见附件 code，在此不再描述展示。

## 4 实验结果

输入合法算数表达式时，运行结果如图1、图2、图3和图4所示，可见程序运行正确。



图 1: 题目测试样例 1



图 2: 题目测试样例 2

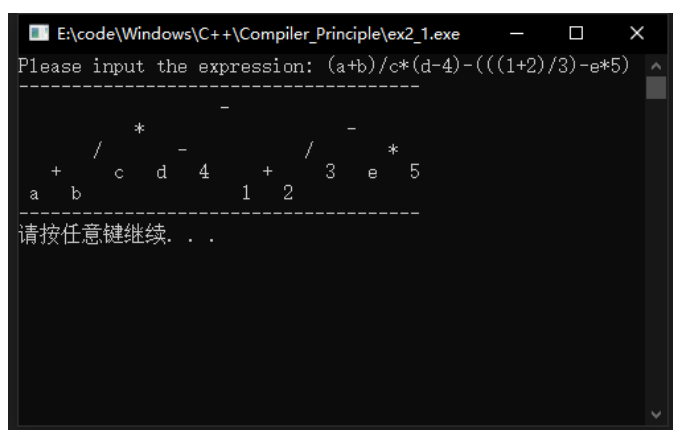



图 3: 多变量测试样例



图 4: 正负数测试样例

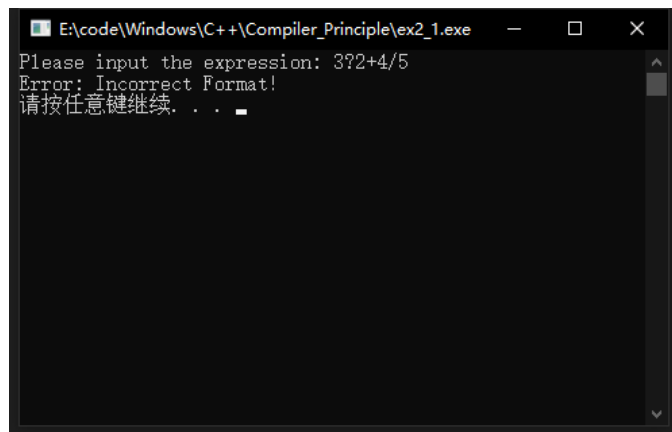
输入非法表达式时，运行结果如图5、图6、图7和图8所示，可见程序运行正确。





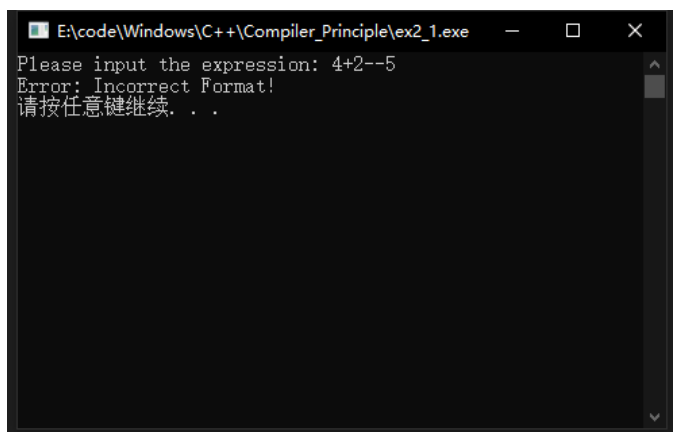
```
E:\code\Windows\C++\Compiler_Principle\ex2_1.exe
Please input the expression: ((-3+2)/5
Error: Incorrect Format!
请按任意键继续. . .
```

图 5: 非法样例 1



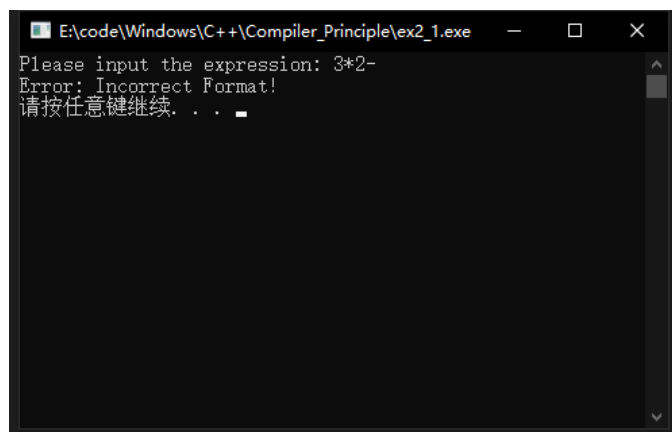
```
E:\code\Windows\C++\Compiler_Principle\ex2_1.exe
Please input the expression: 3?2+4/5
Error: Incorrect Format!
请按任意键继续. . .
```

图 6: 非法样例 2



```
E:\code\Windows\C++\Compiler_Principle\ex2_1.exe
Please input the expression: 4+2--5
Error: Incorrect Format!
请按任意键继续. . .
```

图 7: 非法样例 3



```
E:\code\Windows\C++\Compiler_Principle\ex2_1.exe
Please input the expression: 3*2-
Error: Incorrect Format!
请按任意键继续. . .
```

图 8: 非法样例 4