栈的应用

- 栈的特点：**后进先出**
- 常用来处理具有递归结构的数据
  - 深度优先搜索
  - **表达式求值**
  - 子程序 / 函数调用的管理
  - **消除递归**

计算表达式的值

- 表达式的递归定义
  - 基本符号集：{0，1，...，9，+，-，*，/，（，）}
  - 语法成分集：{<表达式>，<项>，<因子>，<常数>，<数字>}
- 中缀表达式　23+(34*45)/(5+6+7)
- 后缀表达式　23 34 45 * 5 6 + 7 + / +

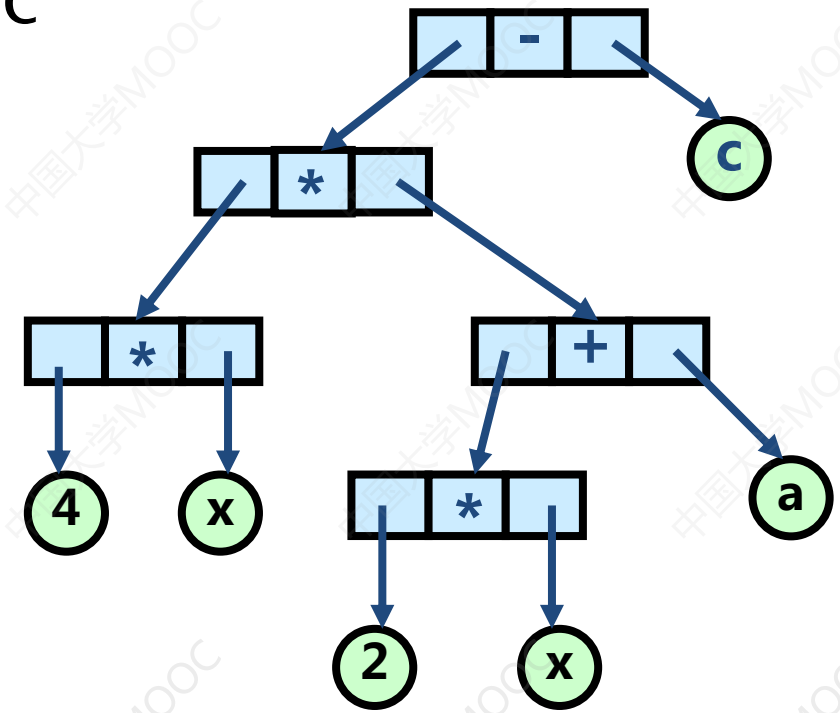
中缀表达式

- **中缀表达式**

  4 * x * (2 * x + a) – c

  – 运算符在中间

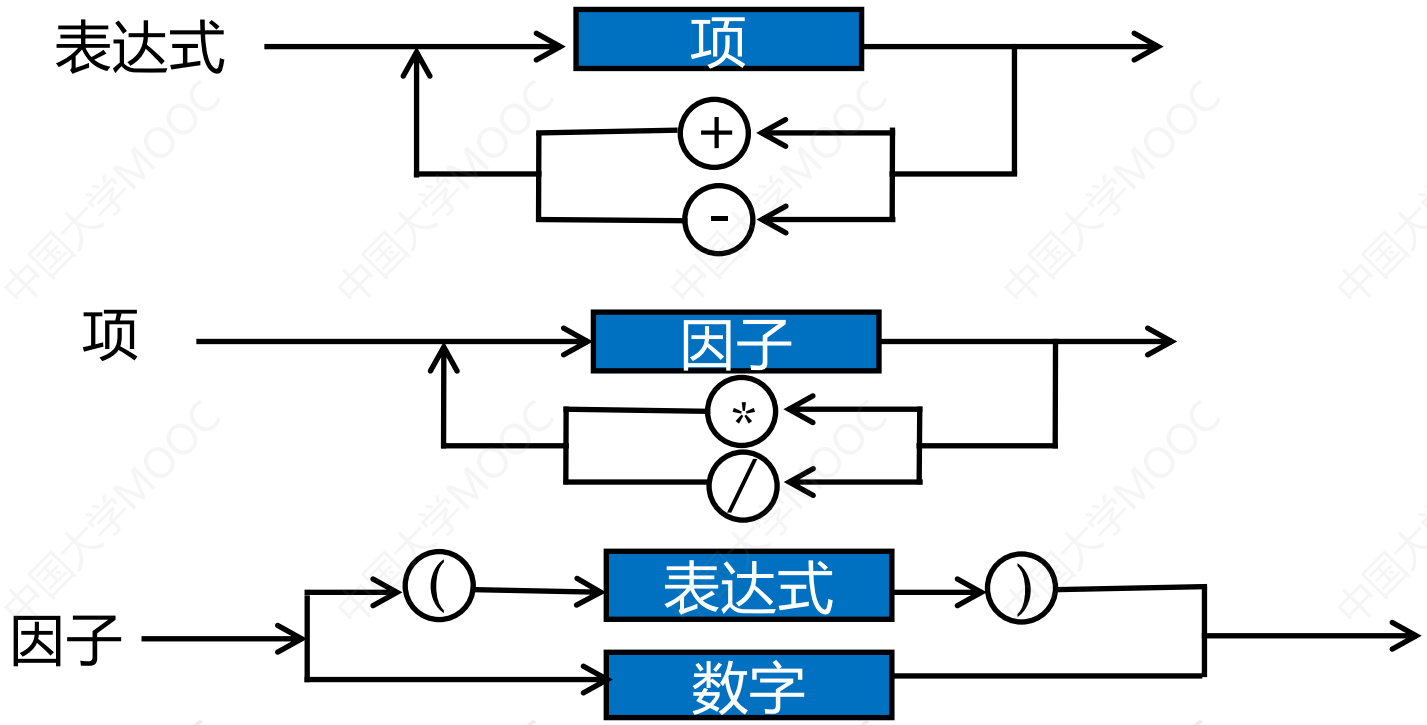
  – 需要括号改变优先级



中缀表达法的语法公式

<表达式> ::= <项> + <项>
| <项> - <项>
| <项>
<项> ::= <因子> * <因子>
| <因子> / <因子>
| <因子>
<因子> ::= <常数>
| (<表达式>)
<常数> ::= <数字>
| <数字> <常数>
<数字> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

表达式的递归图示

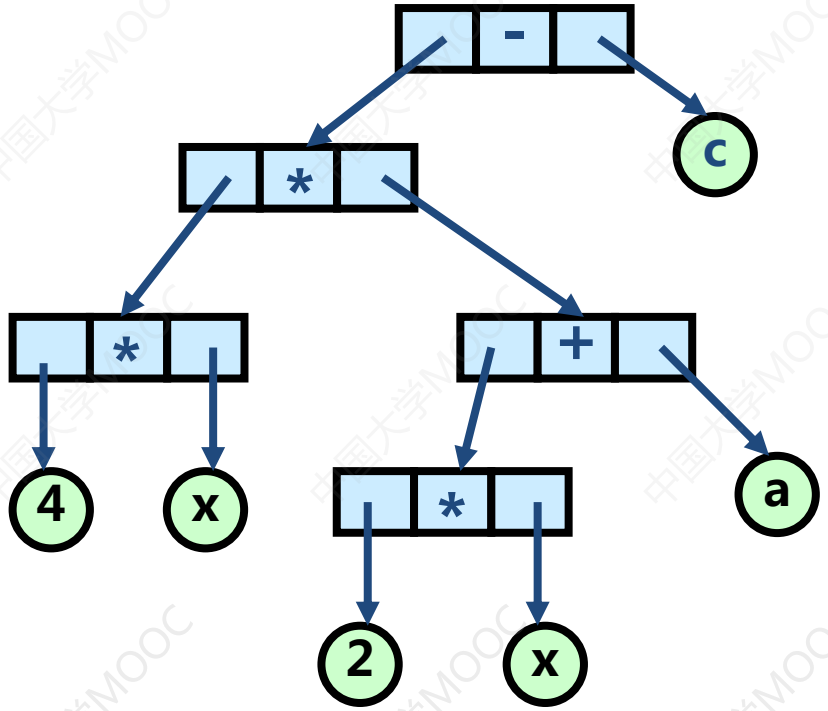


后缀表达式

- **后缀表达式**

  4 x * 2 x * a + * c -

  - 运算符在后面
  - 完全不需要括号



后缀表达式

<表达式> ::= <项> <项> +
| <项> <项> -
| <项>
<项> ::= <因子> <因子> *
| <因子> <因子> /
| <因子>
<因子> ::= <常数>
<常数> ::= <数字>
| <数字> <常数>
<数字> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

- 中缀表达式

  (23 + 34 * 45 / (5 + 6 + 7))

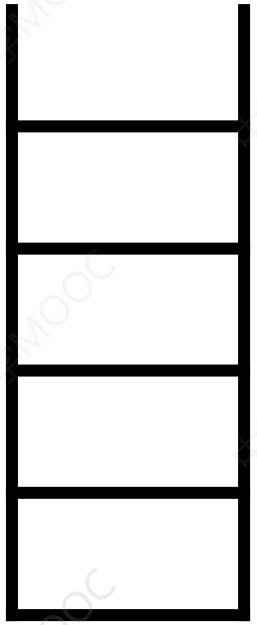
- 后缀表达式

  23 34 45 * 5 6 + 7 + / +

待处理后缀表达式：

23 34 45 * 5 6 + 7 + / +

栈状态的变化



	计算	结果

后缀表达式求值

- 循环：依次顺序读入表达式的符号序列（假设以 = 作为输入序列的结束），并根据读入的元素符号逐一分析
  1. 当遇到的是一个操作数，则压入栈顶
  2. 当遇到的是一个运算符，就从栈中两次取出栈顶，按照运算符对这两个操作数进行计算。然后将计算结果压入栈顶
- 如此继续，直到遇到符号 = ，这时栈顶的值就是输入表达式的值

后缀计算器的类定义

```
class Calculator {
private:
    Stack<double> s;              // 这个栈用于压入保存操作数
    // 从栈顶弹出两个操作数opd1和opd2
    bool GetTwoOperands(double& opd1,double& opd2);
    // 取两个操作数，并按op对两个操作数进行计算
    void Compute(char op);
public:
    Calculator(void){} ;          // 创建计算器实例，开辟一个空栈
    void Run(void);               // 读入后缀表达式，遇“=”符号结束
    void Clear(void);             // 清除计算器，为下一次计算做准备
};
```

后缀计算器的类定义

```
template <class ELEM>
bool Calculator<ELEM>::GetTwoOperands(ELEM& opnd1, ELEM& opnd2) {
    if (S.IsEmpty()) {
        cerr << "Missing operand!" <<endl;
        return false;
    }
    opnd1 = S.Pop(); // 右操作数
    if (S.IsEmpty()) {
        cerr << "Missing operand!" <<endl;
        return false;
    }
    opnd2 = S.Pop(); // 左操作数
    return true;
}
```

后缀计算器的类定义

```
template <class ELEM> void Calculator<ELEM>::Compute(char op) {
    bool result;  ELEM operand1, operand2;
    result = GetTwoOperands(operand1, operand2);
    if (result == true)
        switch(op) {
            case '+' : S.Push(operand2 + operand1); break;
            case '-' : S.Push(operand2 - operand1); break;
            case '*' : S.Push(operand2 * operand1); break;
            case '/' : if (operand1 == 0.0) {
                       cerr << "Divide by 0!" << endl;
                       S.ClearStack();
                   } else S.Push(operand2 / operand1);
                   break;
        }
    else S.ClearStack();
}
```

后缀计算器的类定义

```
template <class ELEM> void Calculator<ELEM>::Run(void) {
    char c; ELEM newoperand;
    while (cin >> c, c != '=') {
        switch(c) {
            case '+': case '-': case '*': case '/':
                Compute(c);
                break;
            default:
                cin.putback(c); cin >> newoperand;
                S.Push(newoperand);
                break;
        }
    }
    if (!S.IsEmpty())
        cout << S.Pop() << endl;          // 印出求值的最后结果
}
```

中缀到后缀表达式的转换

思路：利用栈来记录表达式中的运算符

- 当输入是操作数，直接输出到后缀表达式序列
- 当输入的是左括号时，也把它压栈
- 当输入的是运算符时
  - While (以下循环)
    - If（栈非空 *and* 栈顶不是左括号 *and* 输入运算符的优先级 “≤”栈顶运算符的优先级）时
      - 将当前栈顶元素弹栈，放到后缀表达式序列中（此步反复循环，直到上述if条件不成立）；将输入的运算符压入栈中。
    - Else 把输入的运算符压栈（>当前栈顶运算符才压栈！）

- 中缀表达式

  (23 + 34 * 45 / (5 + 6 + 7))

- 后缀表达式

  23 34 45 * 5 6 + 7 + / +

中缀到后缀表达式的转换

- 当输入的是右括号时，先判断栈是否为空
  - 若为空（括号不匹配），异常处理，清栈退出；
  - 若非空，则把栈中的元素依次弹出
    - 遇到第一个左括号为止，将弹出的元素输出到后缀表达式的序列中（弹出的开括号不放到序列中）
    - 若没有遇到开括号，说明括号也不匹配，做异常处理，清栈退出
- 最后，当中缀表达式的符号序列全部读入时，若栈内仍有元素，把它们全部依次弹出，都放到后缀表达式序列尾部。
  - 若弹出的元素遇到开括号时，则说明括号不匹配，做错误异常处理，清栈退出

中缀表达式→后缀表达式

输入中缀表达式： 23 + (34 * 45) / (5 + 6 + 7)

栈的状态

输出后缀表达式：