# 4.栈与队列

## （c4）栈应用：中缀表达式求值

邓俊辉

deng@tsinghua.edu.cn

知实而不知名，知名而不知实，皆不知也

# 表达式求值

- 给定语法正确的算术表达式S，计算与之对应的数值
- `$ echo $(( 0 + ( 1 + 23 ) / 4 * 5 * 67 - 8 + 9 ))`
- `\>  set /a ( !0 ^<^< ( 1 - 2 + 3 * 4 ) ) - 5 * ( 6 ^| 7 ) / ( 8 ^^ 9 )`
- PostScript

  `GS> 0 1 23 add 4 div 5 mul 67 mul add 8 sub 9 add =`
- `Excel:  = COS(0) + 1 - ( 2 - POWER( ( FACT(3) - 4 ), 5) ) * 67 - 8 + 9`
- `Word:   = NOT(0) + 12 + 34 * 56 + 7 + 89`
- calc:  `0 ! + 12 + 34 * 56 + 7 + 89 =`
- calc:  `0 ! + 1 - ( 2 - ( 3 ! - 4 ) ^ 5 ) * 67 - 8 + 9 =`

  `y`

# 表达式求值

❖ 表达式的求值，可视作 字符串 与对应 数值 交替转换的过程

❖ str(v)：数值v对应的（十进制）字符串（名）

val(S)：符号串S对应的（十进制）数值（实）

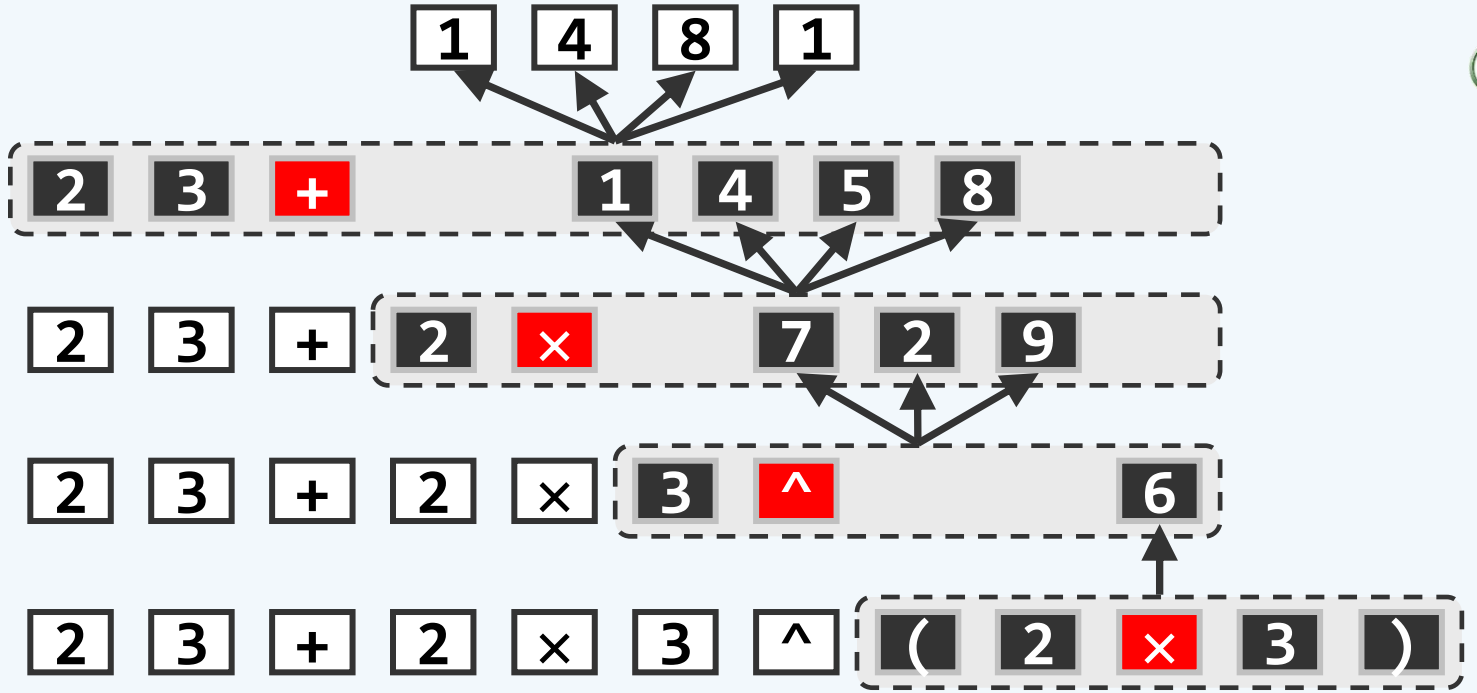
❖ 设表达式：$S = S_L + S_0 + S_R$

1）$S_0$可优先计算，且

2）$val(S_0) = v_0$

❖ 则有递推化简关系

$val(S) = val(S_L + str(v_0) + S_R)$

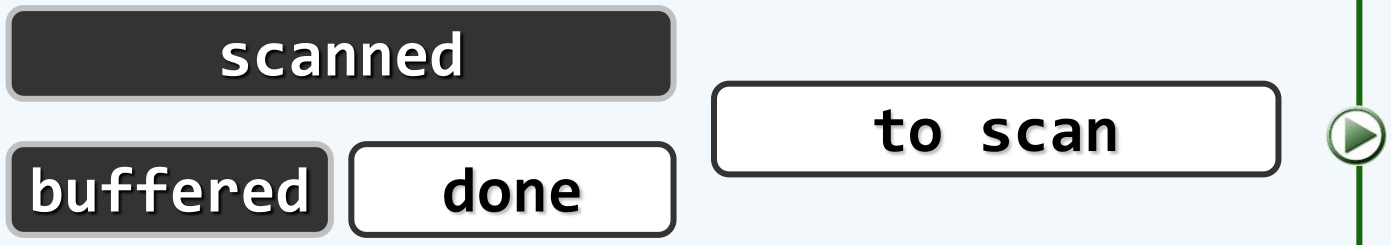


# 延迟缓冲

- 难点：如何高效地找到可优先计算的$s_0$（亦即，其对应的运算符）？
- 与括号匹配等应用不同，不能简单地按“左先右后”次序处理各运算符
- 此时，需要考虑更多因素　　（约定俗成的）优先级：1 + 2 * 3 ^ 4 !

  可强行改变次序的括号：( ( ( 1 + 2 ) * 3 ) ^ 4 ) !
- 仅根据表达式的前缀，不足以确定各运算符的计算次序

  只有在获得足够的后续信息之后，才能确定其中哪些运算符可以执行




- 体现在求值算法的流程上

  为处理某一前缀，必须提前预读并分析更长的前缀
- 为此，需借助某种支持延迟缓冲的机制...

# 求值算法 = 栈 + 线性扫描

❖ 自左向右扫描表达式，用栈记录已扫描的部分（含已执行运算的结果）

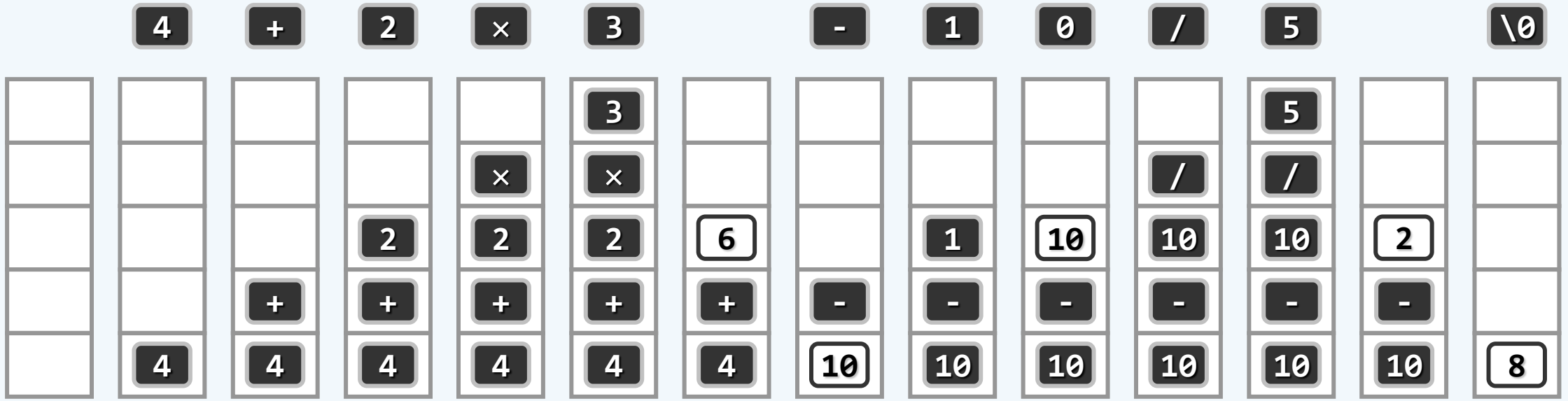
在每一字符处

while（栈的顶部存在可优先计算的子表达式）　　//如何判断？

该子表达式退栈；计算其数值；计算结果进栈

当前字符进栈，转入下一字符

❖ 只要语法正确，则栈内最终应只剩一个元素　　//即表达式对应的数值



# 实现：主算法

```
❖ float evaluate( char* S, char* & RPN ) { //中缀表达式求值
    Stack<float> opnd; Stack<char> optr; //运算数栈、运算符栈
    optr.push('\0'); //尾哨兵'\0'也作为头哨兵首先入栈
    while ( !optr.empty() ) { //逐个处理各字符，直至运算符栈空
       if ( isdigit( *S ) ) //若当前字符为操作数，则
          readNumber( S, opnd ); //读入（可能多位的）操作数
       else //若当前字符为运算符，则视其与栈顶运算符之间优先级的高低
          switch( orderBetween( optr.top(), *S ) ) { /* 分别处理 */ }
    } //while
    return opnd.pop(); //弹出并返回最后的计算结果
 }
```

# 实现：优先级表

```
const char pri[N_OPTR][N_OPTR] = { //运算符优先等级 [栈顶][当前]
   //                  |-------------- 当前运算符 --------------|
   //                    +    -    *    /    ^    !    (    )    \0
   /* --  + */   '>', '>', '<', '<', '<', '<', '<', '>', '>',
   /*  |  - */   '>', '>', '<', '<', '<', '<', '<', '>', '>',
   /* 栈  * */   '>', '>', '>', '>', '<', '<', '<', '>', '>',
   /* 顶  / */   '>', '>', '>', '>', '<', '<', '<', '>', '>',
   /* 运  ^ */   '>', '>', '>', '>', '>', '<', '<', '>', '>',
   /* 算  ! */   '>', '>', '>', '>', '>', '>', ' ', '>', '>',
   /* 符  ( */   '<', '<', '<', '<', '<', '<', '<', '=', ' ',
   /*  |  ) */   ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ',
   /* -- \0 */   '<', '<', '<', '<', '<', '<', '<', ' ', '='
};
```

## 实现：不同优先级处理方法

```
❖ switch( orderBetween( optr.top(), *S ) ) {
     case '<': //栈顶运算符优先级更低
        optr.push( *S ); S++; break; //计算推迟，当前运算符进栈
     case '=': //优先级相等（当前运算符为右括号，或尾部哨兵'\0'）
        optr.pop(); S++; break; //脱括号并接收下一个字符
     case '>': { //栈顶运算符优先级更高，实施相应的计算，结果入栈
        char op = optr.pop(); //栈顶运算符出栈，执行对应的运算
        if ( '!' == op ) opnd.push( calcu( op, opnd.pop() ) ); //一元运算符
        else {   float pOpnd2 = opnd.pop(), pOpnd1 = opnd.pop(); //二元运算符
                 opnd.push( calcu( pOpnd1, op, pOpnd2 ) ); //实施计算，结果入栈
        } //为何不直接：opnd.push( calcu( opnd.pop(), op, opnd.pop() ) )？
        break;
     } //case '>'
  } //switch
```

# 实现：优先级表（理解）

```
const char pri[N_OPTR][N_OPTR] = { //运算符优先等级 [栈顶][当前]
   //              |-------------- 当前运算符 --------------|
   //               +    -    *    /    ^    !    (    )    \0
   /* --  + */   '>', '>', '<', '<', '<', '<', '<', '>', '>',
   /*  |  - */   '>', '>', '<', '<', '<', '<', '<', '>', '>',
   /* 栈  * */   '>', '>', '>', '>', '<', '<', '<', '>', '>',
   /* 顶  / */   '>', '>', '>', '>', '<', '<', '<', '>', '>',
   /* 运  ^ */   '>', '>', '>', '>', '>', '<', '<', '>', '>',
   /* 算  ! */   '>', '>', '>', '>', '>', '>', ' ', '>', '>',
   /* 符  ( */   '<', '<', '<', '<', '<', '<', '<', '=', ' ',
   /*  |  ) */   ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ',
   /* -- \0 */   '<', '<', '<', '<', '<', '<', '<', ' ', '='
};
```

# 实例

| 表达式 | 运算符栈 | 操作数栈 | 注解 |
|---|---|---|---|
| (0!+1)*2^(3!+4)-(5!-67-(8+9))$ | $ | | 表达式起始标识入栈 |
| (0!+1)*2^(3!+4)-(5!-67-(8+9))$ | $ ( | | 左括号入栈 |
| (0!+1)*2^(3!+4)-(5!-67-(8+9))$ | $ ( | 0 | 操作数0入栈 |
| (0!+1)*2^(3!+4)-(5!-67-(8+9))$ | $ ( ! | 0 | 运算符'!'入栈 |
| (0!+1)*2^(3!+4)-(5!-67-(8+9))$ | $ ( | 1 | 运算符'!'出栈执行 |
| (0!+1)*2^(3!+4)-(5!-67-(8+9))$ | $ ( + | 1 | 运算符'+'入栈 |
| (0!+1)*2^(3!+4)-(5!-67-(8+9))$ | $ ( + | 1 1 | 操作数1入栈 |
| (0!+1)*2^(3!+4)-(5!-67-(8+9))$ | $ ( | 2 | 运算符'+'出栈执行 |
| (0!+1)*2^(3!+4)-(5!-67-(8+9))$ | $ | 2 | 左括号出栈 |
| (0!+1)*2^(3!+4)-(5!-67-(8+9))$ | $ * | 2 | 运算符'*'入栈 |
| (0!+1)*2^(3!+4)-(5!-67-(8+9))$ | $ * | 2 2 | 操作数2入栈 |
| (0!+1)*2^(3!+4)-(5!-67-(8+9))$ | $ * ^ | 2 2 | 运算符'^'入栈 |
| (0!+1)*2^(3!+4)-(5!-67-(8+9))$ | $ * ^ ( | 2 2 | 左括号入栈 |
| (0!+1)*2^(3!+4)-(5!-67-(8+9))$ | $ * ^ ( | 2 2 3 | 操作数3入栈 |

# 实例

| 表达式 | 运算符栈 | 操作数栈 | 注解 |
|---|---|---|---|
| (0!+1)*2^(3!+4)-(5!-67-(8+9))$ | $ * ^ ( ! | 2 2 3 | 运算符'!'入栈 |
| (0!+1)*2^(3!+4)-(5!-67-(8+9))$ | $ * ^ ( | 2 2 6 | 运算符'!'出栈执行 |
| (0!+1)*2^(3!+4)-(5!-67-(8+9))$ | $ * ^ ( + | 2 2 6 | 运算符'+'入栈 |
| (0!+1)*2^(3!+4)-(5!-67-(8+9))$ | $ * ^ ( + | 2 2 6 4 | 操作数4入栈 |
| (0!+1)*2^(3!+4)-(5!-67-(8+9))$ | $ * ^ ( | 2 2 10 | 运算符'+'出栈执行 |
| (0!+1)*2^(3!+4)-(5!-67-(8+9))$ | $ * ^ | 2 2 10 | 左括号出栈 |
| (0!+1)*2^(3!+4)-(5!-67-(8+9))$ | $ * | 2 1024 | 运算符'^'出栈执行 |
| (0!+1)*2^(3!+4)-(5!-67-(8+9))$ | $ | 2048 | 运算符'*'出栈执行 |
| (0!+1)*2^(3!+4)-(5!-67-(8+9))$ | $ - | 2048 | 运算符'-'入栈 |
| (0!+1)*2^(3!+4)-(5!-67-(8+9))$ | $ - ( | 2048 | 左括号入栈 |
| (0!+1)*2^(3!+4)-(5!-67-(8+9))$ | $ - ( | 2048 5 | 操作数5入栈 |
| (0!+1)*2^(3!+4)-(5!-67-(8+9))$ | $ - ( ! | 2048 5 | 运算符'!'入栈 |
| (0!+1)*2^(3!+4)-(5!-67-(8+9))$ | $ - ( | 2048 120 | 运算符'!'出栈执行 |
| (0!+1)*2^(3!+4)-(5!-67-(8+9))$ | $ - ( - | 2048 120 | 运算符'-'入栈 |

# 实例

| 表达式 | 运算符栈 | 操作数栈 | 注解 |
|---|---|---|---|
| (0!+1)*2^(3!+4)-(5!-**67**-(8+9))$ | $ - ( - | 2048 120 67 | 操作数67入栈 |
| (0!+1)*2^(3!+4)-(5!-67**-**(8+9))$ | $ - ( | 2048 53 | 运算符'-'出栈执行 |
| (0!+1)*2^(3!+4)-(5!-67**-**(8+9))$ | $ - ( - | 2048 53 | 运算符'-'入栈 |
| (0!+1)*2^(3!+4)-(5!-67-**(**8+9))$ | $ - ( - ( | 2048 53 | 左括号入栈 |
| (0!+1)*2^(3!+4)-(5!-67-(**8**+9))$ | $ - ( - ( | 2048 53 8 | 操作数8入栈 |
| (0!+1)*2^(3!+4)-(5!-67-(8**+**9))$ | $ - ( - ( + | 2048 53 8 | 运算符'+'入栈 |
| (0!+1)*2^(3!+4)-(5!-67-(8+**9**))$ | $ - ( - ( + | 2048 53 8 9 | 操作数9入栈 |
| (0!+1)*2^(3!+4)-(5!-67-(8+9**)**)$ | $ - ( - ( | 2048 53 17 | 运算符'+'出栈执行 |
| (0!+1)*2^(3!+4)-(5!-67-(8+9)**)**$ | $ - ( - | 2048 53 17 | 左括号出栈 |
| (0!+1)*2^(3!+4)-(5!-67-(8+9)**)**$ | $ - ( | 2048 36 | 运算符'-'出栈执行 |
| (0!+1)*2^(3!+4)-(5!-67-(8+9)**)**$ | $ - | 2048 36 | 左括号出栈 |
| (0!+1)*2^(3!+4)-(5!-67-(8+9))**$** | $ | 2012 | 运算符'-'出栈执行 |
| (0!+1)*2^(3!+4)-(5!-67-(8+9))**$** | | 2012 | 表达式起始标识出栈 |
| (0!+1)*2^(3!+4)-(5!-67-(8+9))$ | | | 返回唯一的元素2012 |