

04-F2

栈与队列

中缀表达式求值：算法

邓俊辉

deng@tsinghua.edu.cn

主算法

```
❖ float evaluate( char* S, char* & RPN ) { //S保证语法正确

    Stack<float> opnd; Stack<char> optr; //运算数栈、运算栈

    optr.push( '\0' ); //铺垫

    while ( ! optr.empty() ) { //逐个处理各字符，直至运算符栈空

        if ( isdigit( *S ) ) //若为操作数（可能多位、小数），则

            readNumber( S, opnd ); //读入

        else //若为运算符，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

}
```

优先级表

```
const char pri[N_OPTR][N_OPTR] = { //运算符优先等级 [栈顶][当前]

    /* -- + */ '>', '>', '<', '<', '<', '<', '<', '>', '>',
    /* | - */ '>', '>', '<', '<', '<', '<', '<', '>', '>',
    /* 栈 * */ '>', '>', '>', '>', '<', '<', '<', '>', '>',
    /* 顶 / */ '>', '>', '>', '>', '<', '<', '<', '>', '>',
    /* 运 ^ */ '>', '>', '>', '>', '>', '<', '<', '>', '>',
    /* 算 ! */ '>', '>', '>', '>', '>', '>', ' ', '>', '>',
    /* 符 ( */ '<', '<', '<', '<', '<', '<', '<', '=', ' ',
    /* | ) */ ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ',
    /* -- \0 */ '<', '<', '<', '<', '<', '<', '<', ' ', '=',

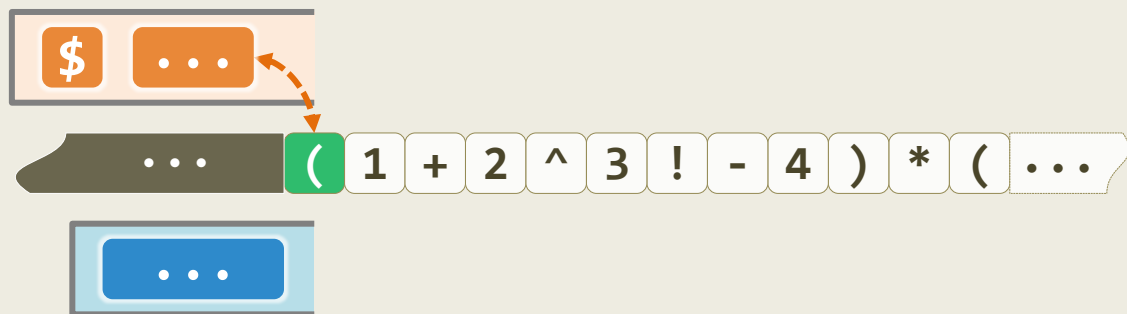
    //      +      -      *      /      ^      !      (      )      \0
    //      |----- 当前运算符 -----|

};
```

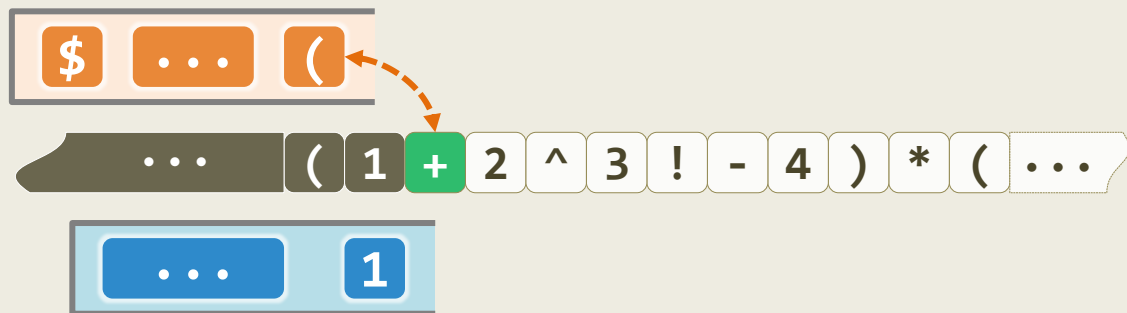
'<'：静待时机：算法

```
❖ 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 '=':  
        /* ..... */  
  
    case '>': {  
        /* ..... */  
  
        break;  
    } //case '>'  
} //switch
```

'<' : 静待时机 : 实例

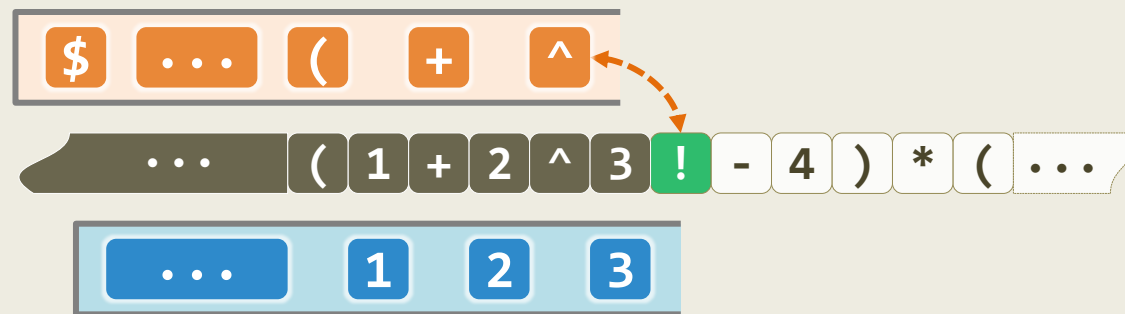


$\text{pri}[' ']['('] = '<'$

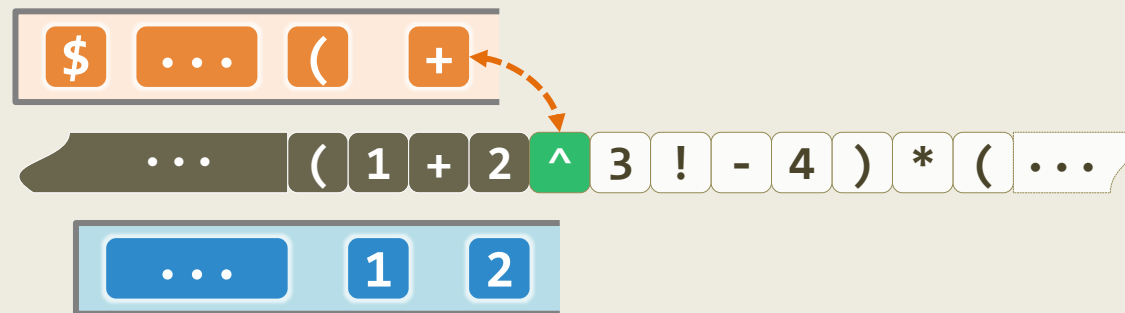


$\text{pri}['(']['+'] = '<'$

$\text{pri}['^']['!'] = '<'$



$\text{pri}['+']['^'] = '<'$

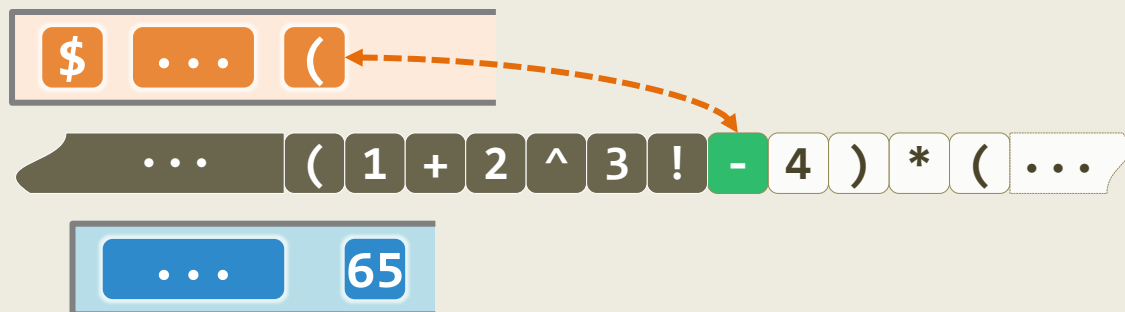


'>' : 时机已到 : 算法

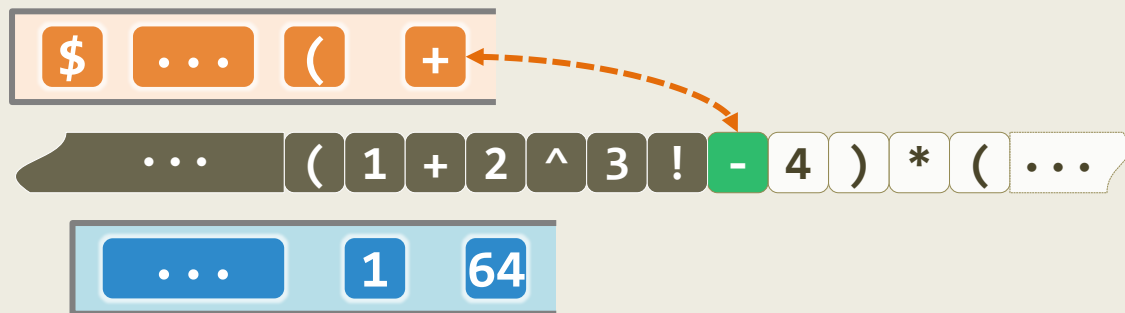
```
❖ switch( orderBetween( optr.top(), *S ) ) {  
    /* ..... */  
    case '>': { //栈顶运算符优先级更高，实施相应的计算，结果入栈  
        char op = optr.pop(); //栈顶运算符出栈，执行对应的运算  
        if ( '!' == op ) opnd.push( calcu( op, opnd.pop() ) ); //一元运算符  
        else { float p0pnd2 = opnd.pop(), p0pnd1 = opnd.pop(); //二元运算符  
            opnd.push( calcu( p0pnd1, op, p0pnd2 ) ); //实施计算，结果入栈  
        } //为何不直接 : opnd.push( calcu( opnd.pop(), op, opnd.pop() ) ) ?  
        break;  
    } //case '>'  
}  
} //switch
```

'>' : 时机已到 : 实例

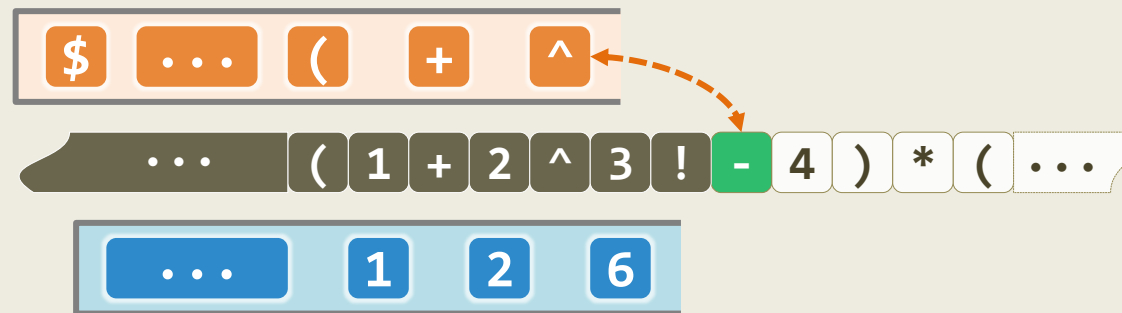
`pri['(']['-'] = '<'`



`pri['+']['-'] = '>'`



`pri['!']['-'] = '>'`

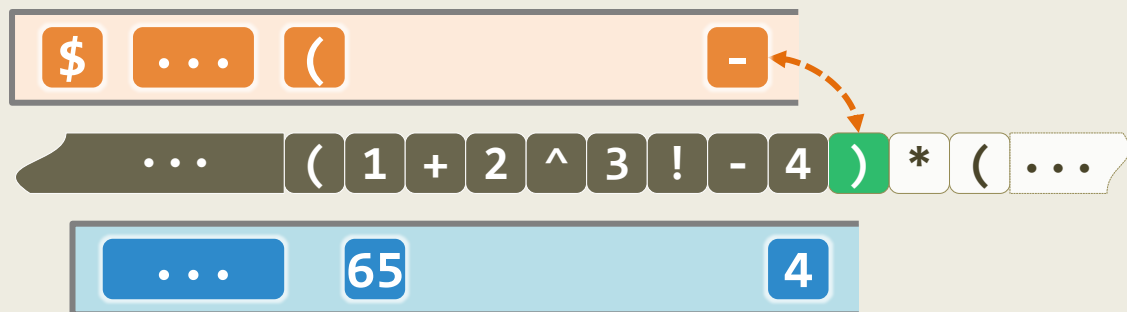


`pri['^']['-'] = '>'`

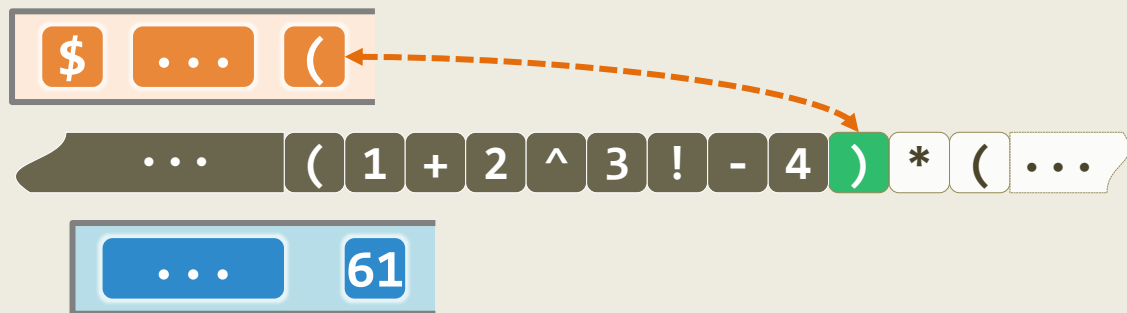
'='：终须了断：算法

```
❖ switch( orderBetween( optr.top(), *S ) ) {  
    case '<':  
        /* ..... */  
    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 '>': {  
        /* ..... */  
        break;  
    } //case '>'  
} //switch
```


'=' : 终须了断 : 实例

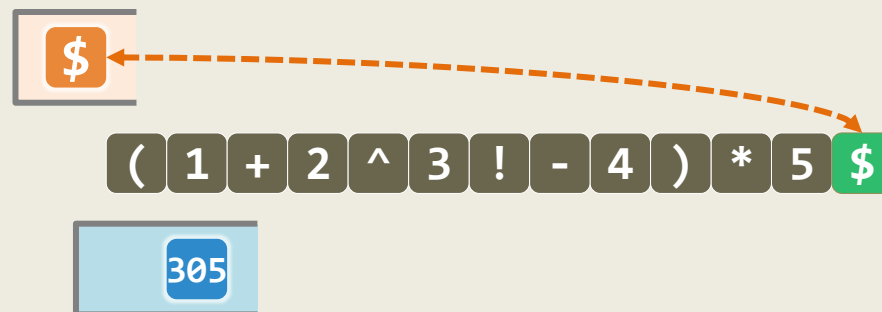


`pri['-'][')'] = '>'`

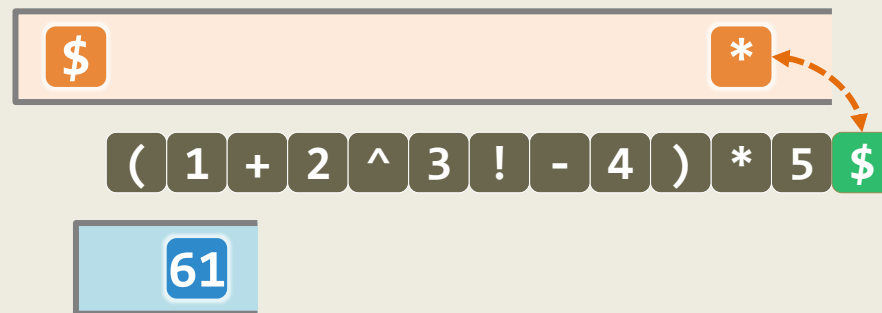


`pri['('][')'] = '='`

`pri['$']['$'] = '='`



`pri['*']['$'] = '>'`



+ - * / ! : 芸芸众生

const char pri[N_OPTR][N_OPTR] = { //运算符优先等级 [栈顶][当前]

/* -- + */ '>', '>', '<', '<', '<', '<', '<', '>', '>',

/* | - */ '>', '>', '<', '<', '<', '<', '<', '>', '>',

/* 栈 * */ '>', '>', '>', '>', '<', '<', '<', '>', '>',

/* 顶 / */ '>', '>', '>', '>', '<', '<', '<', '>', '>',

/* 运 ^ */ '>', '>', '>', '>', '>', '<', '<', '>', '>',

/* 算 ! */ '>', '>', '>', '>', '>', '>', ' ', '>', '>',

/* 符 (*/ '<', '<', '<', '<', '<', '<', '<', '=', ' ',

/* |) */ ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ',

/* -- \0 */ '<', '<', '<', '<', '<', '<', '<', ' ', '=',

// + - * / ^ ! () \0

// |-----当前运算符-----|

};

'(' : 我不下地狱，谁下地狱

const char pri[N_OPTR][N_OPTR] = { //运算符优先级 [栈顶][当前]

/* -- + */ '>', '>', '<', '<', '<', '<', '<', '>', '>',

/* | - */ '>', '>', '<', '<', '<', '<', '<', '>', '>',

/* 栈 * */ '>', '>', '>', '>', '<', '<', '<', '>', '>',

/* 顶 / */ '>', '>', '>', '>', '<', '<', '<', '>', '>',

/* 运 ^ */ '>', '>', '>', '>', '>', '>', '<', '>', '>',

/* 算 ! */ '>', '>', '>', '>', '>', '>', '>', '>', '>',

/* 符 (*/ '<', '<', '<', '<', '<', '<', '<', '=', ' ',

/* |) */ ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ',

/* -- \0 */ '<', '<', '<', '<', '<', '<', '<', ' ', '='

// + - * / ^ ! () \0

// |-----当前运算符-----|

};

') ' : deadline has arrived (then full blood resurrection)

const char pri[N_OPTR][N_OPTR] = { //运算符优先等级 [栈顶][当前]

/* -- + */ '>', '>', '<', '<', '<', '<', '<', '>', '>',

/* | - */ '>', '>', '<', '<', '<', '<', '<', '>', '>',

/* 栈 * */ '>', '>', '>', '>', '<', '<', '<', '>', '>',

/* 顶 / */ '>', '>', '>', '>', '<', '<', '<', '>', '>',

/* 运 ^ */ '>', '>', '>', '>', '>', '>', '<', '<', '<',

/* 算 ! */ '>', '>', '>', '>', '>', '>', '>', '>', '>',

/* 符 (*/ '<', '<', '<', '<', '<', '<', '<', '=', ' ',

/* |) */ ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ',

/* -- \0 */ '<', '<', '<', '<', '<', '<', '<', ' ', '=',

// + - * / ^ ! () \0

// |-----当前运算符-----|

};

'\0' : 从创世纪, 到世界末日

const char pri[N_OPTR][N_OPTR] = { //运算符优先等级 [栈顶][当前]

/* -- + */ '>', '>', '<', '<', '<', '<', '<', '>', '>',

/* | - */ '>', '>', '<', '<', '<', '<', '<', '>', '>',

/* 栈 * */ '>', '>', '>', '>', '<', '<', '<', '>', '>',

/* 顶 / */ '>', '>', '>', '>', '<', '<', '<', '>', '>',

/* 运 ^ */ '>', '>', '>', '>', '>', '<', '<', '>', '>',

/* 算 ! */ '>', '>', '>', '>', '>', '>', '>', '>', '>',

/* 符 (*/ '<', '<', '<', '<', '<', '<', '<', '=', ' ',

/* |) */ ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ',

/* -- \0 */ '<', '<', '<', '<', '<', '<', '<', ' ', '='

// + - * / ^ ! () \0

// |-----当前运算符-----|

};