

4. 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 ) { //中缀表达式求值：S 语法正确  
    Stack<float> opnd; Stack<char> optr; //运算数栈、运算栈  
    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
```

优先级表

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

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

};

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 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
```

优先级表（理解）

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

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

};