栈与队列

中缀表达式求值：算法

04-F2

邓俊辉

deng@tsinghua.edu.cn

主算法

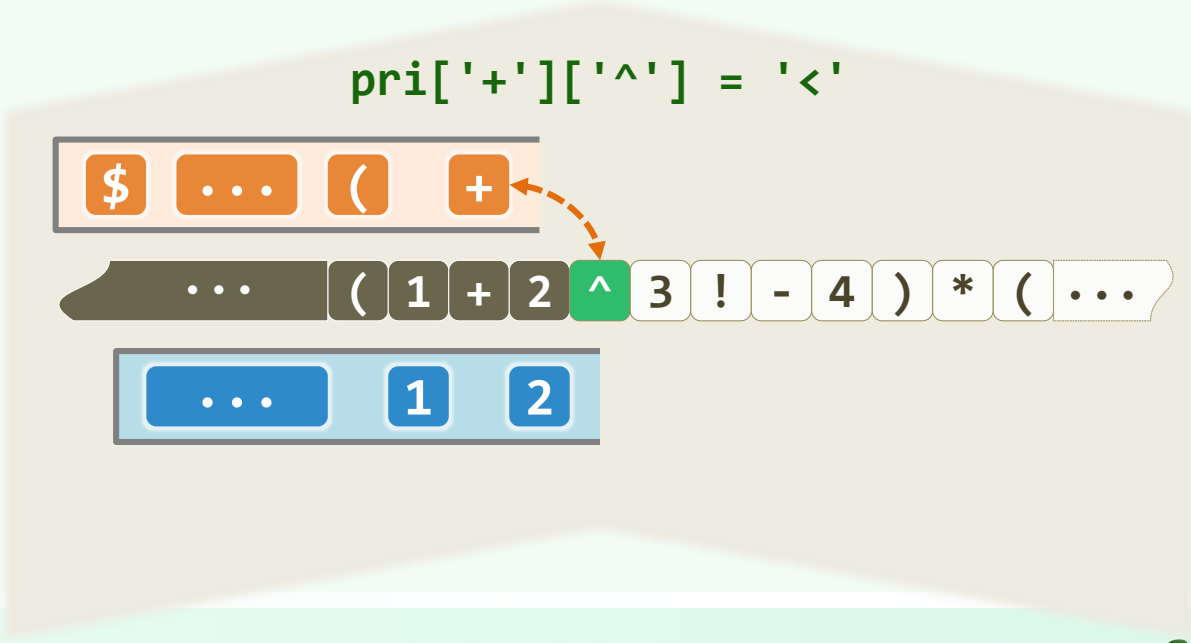
```
double evaluate( char* S, char* RPN ) { //S保证语法正确
   Stack<double> opnd; Stack<char> optr; //运算数栈、运算符栈
   optr.push('\0'); //哨兵
   while ( ! optr.empty() ) { //逐个处理各字符，直至运算符栈空
      if ( isdigit( *S ) ) //若为操作数（可能多位、小数），则
         readNumber( S, opnd ); //读入
      else //若为运算符，则视其与栈顶运算符之间优先级的高低
         switch( priority( 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

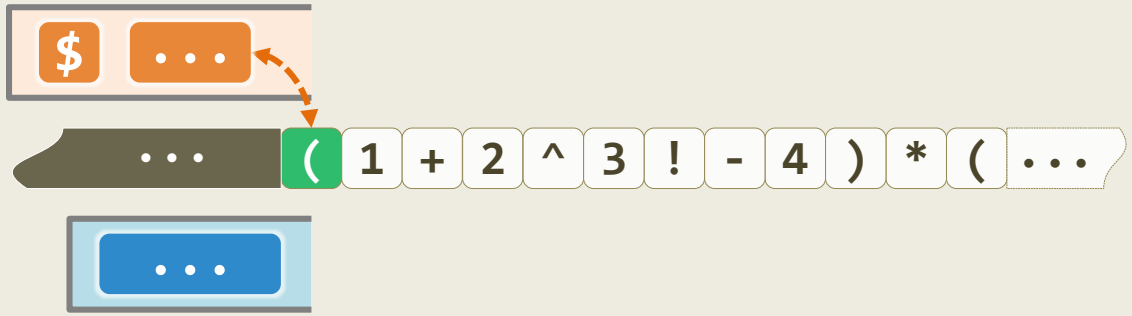
```
const char pri[N_OPTR][N_OPTR] = { //运算符优先等级 [栈顶][当前]
   /* --  + */  '>', '>', '<', '<', '<', '<', '<', '>', '>',
   /*  |  - */  '>', '>', '<', '<', '<', '<', '<', '>', '>',
   /* 栈  * */  '>', '>', '>', '>', '<', '<', '<', '>', '>',
   /* 顶  / */  '>', '>', '>', '>', '<', '<', '<', '>', '>',
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

'<': 静待时机: 算法

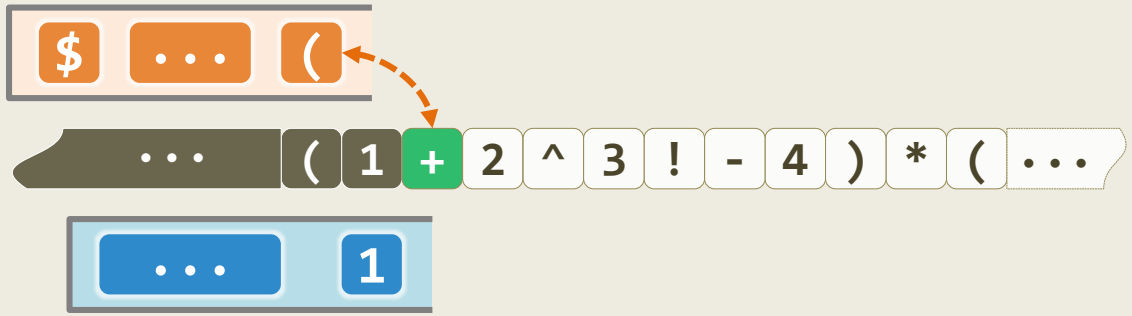
```
switch( priority( optr.top(), *S ) ) {
   case '<': //栈顶运算符优先级更低
      optr.push( *S ); S++; break; //计算推迟，当前运算符进栈
   case '=':
      /* ..... */
   case '>': {
      /* ..... */
      break;
   } //case '>'
} //switch
```



'<'：静待时机：实例

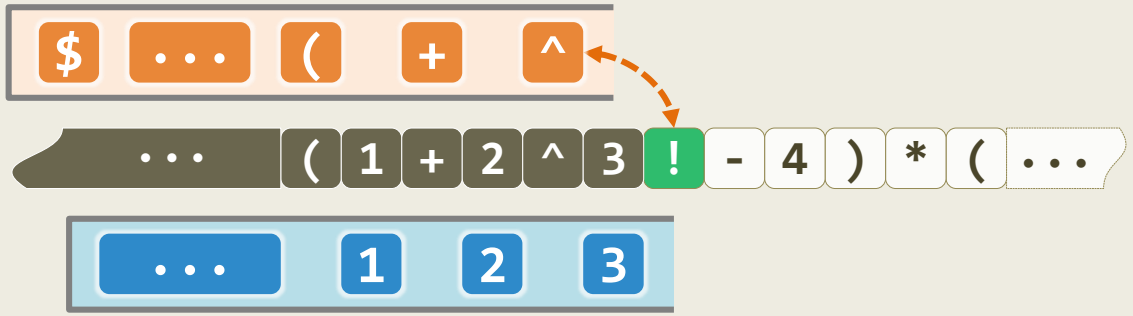


pri[ ]['('] = '<'

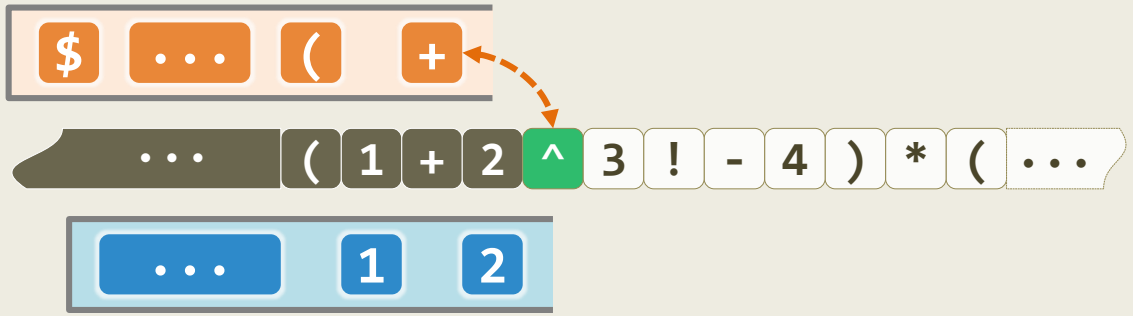


pri['('][+'] = '<'

pri['^']['!'] = '<'

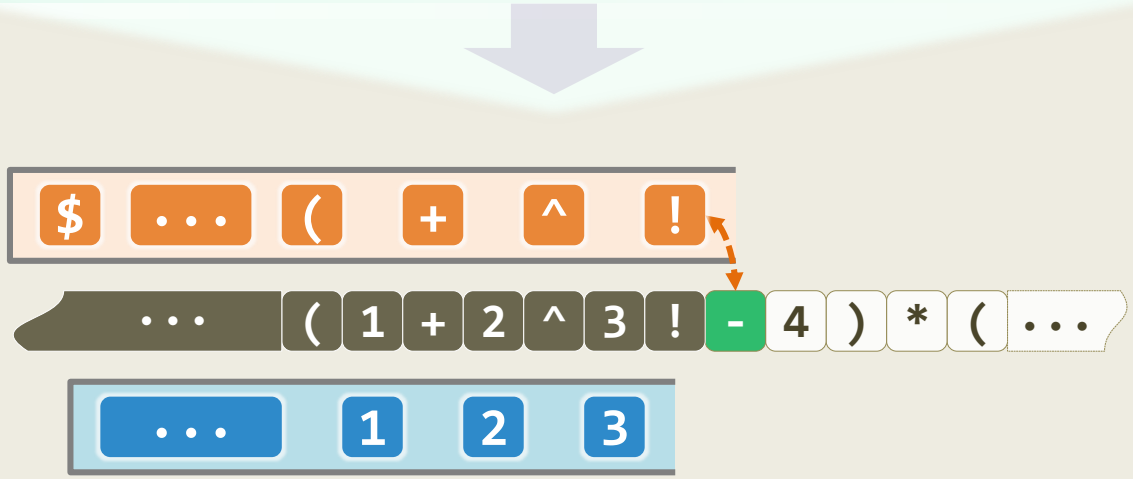


pri['+']['^'] = '<'



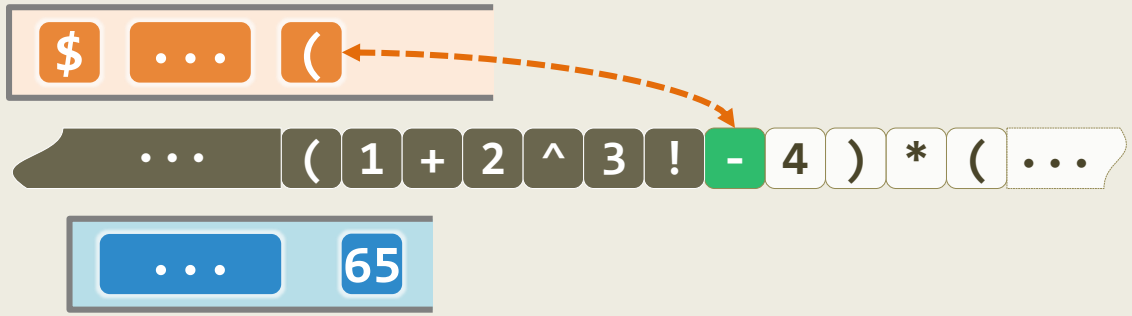
'>': 时机已到：算法

```
switch( priority( optr.top(), *S ) ) {
   /* ..... */
   case '>': {
      char op = optr.pop();
      if ( '!' == op ) opnd.push( calcu( op, opnd.pop() ) ); //一元运算符
      else { double opnd2 = opnd.pop(), opnd1 = opnd.pop(); //二元运算符
             opnd.push( calcu( opnd1, op, opnd2 ) ); //实施计算，结果入栈
      } //为何不直接：opnd.push( calcu( opnd.pop(), op, opnd.pop() ) )?
      break;
   } //case '>'
} //switch
```

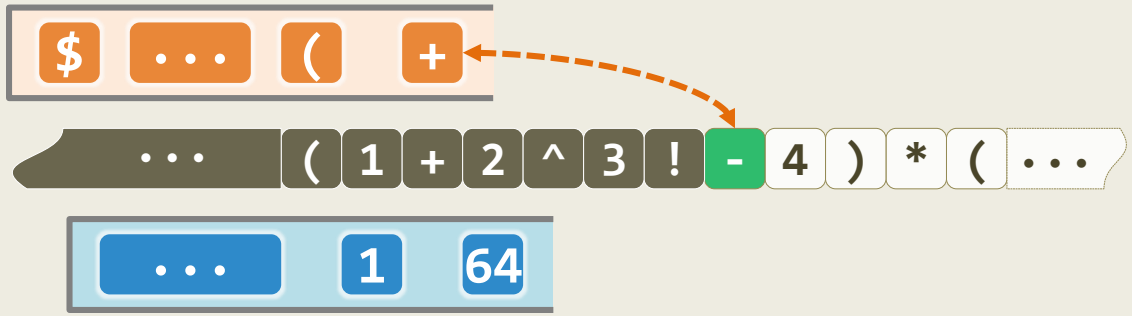


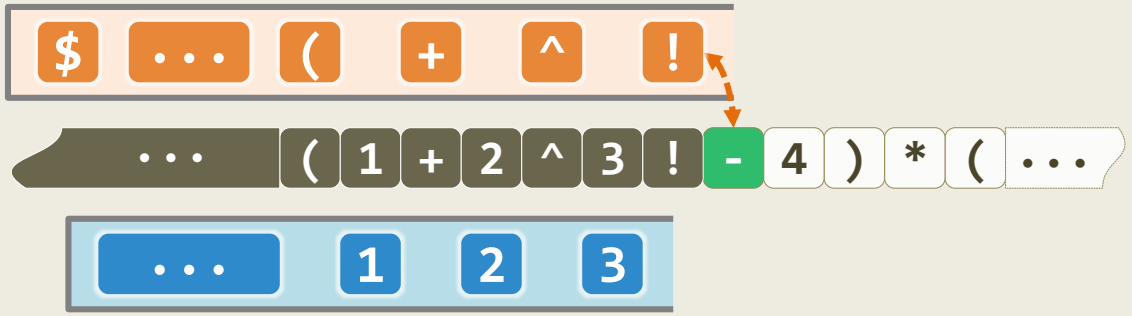
'>': 时机已到：实例

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

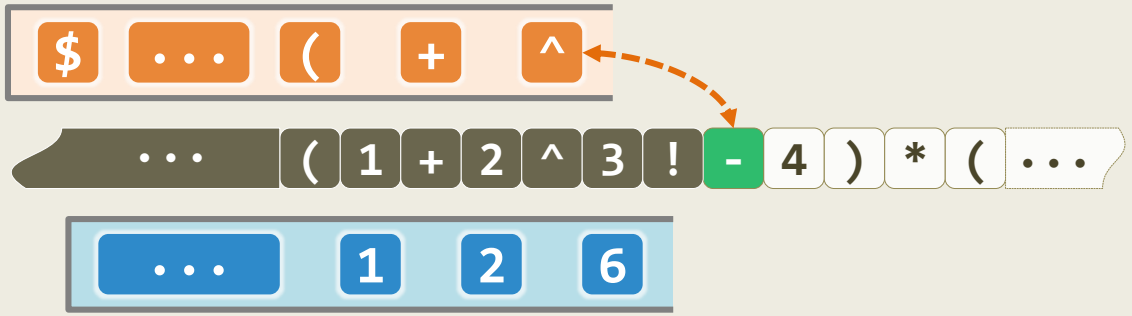


pri['+']['-'] = '>'





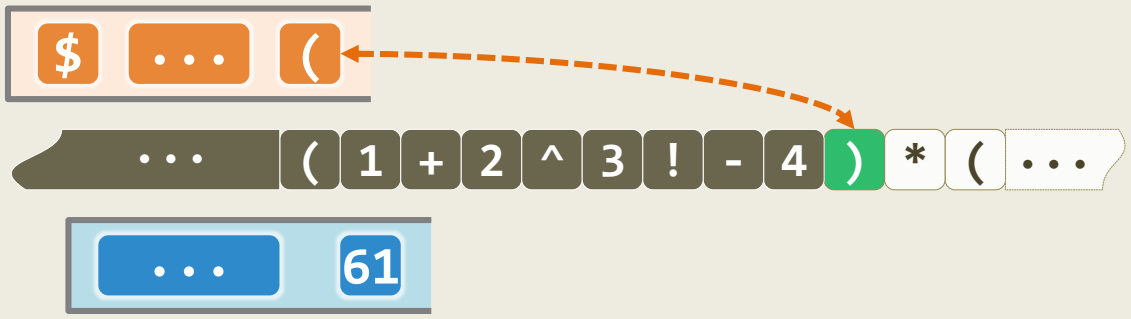
pri['!']['-'] = '>'



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

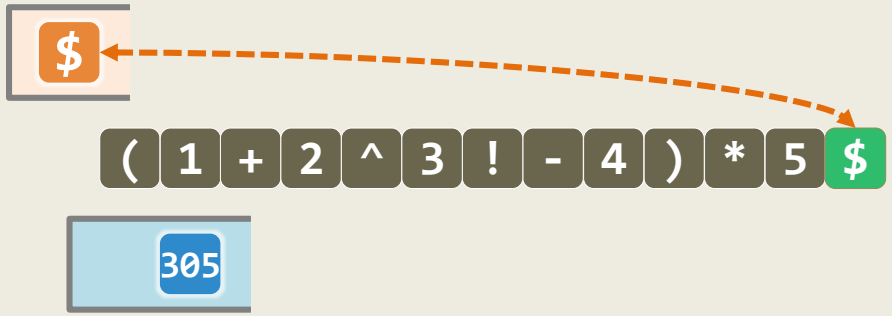
'=': 终须了断：算法

```
switch( priority( 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
```

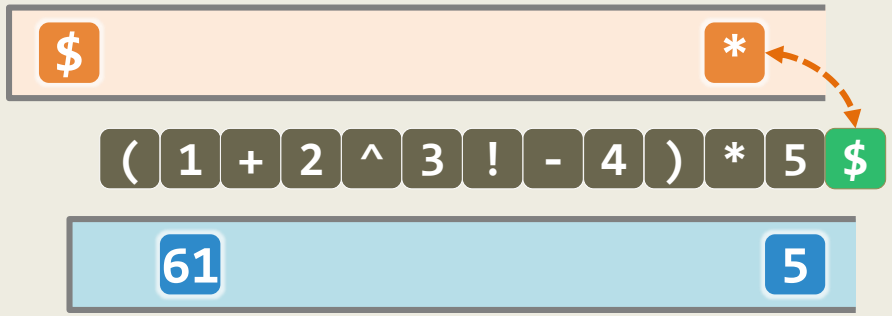


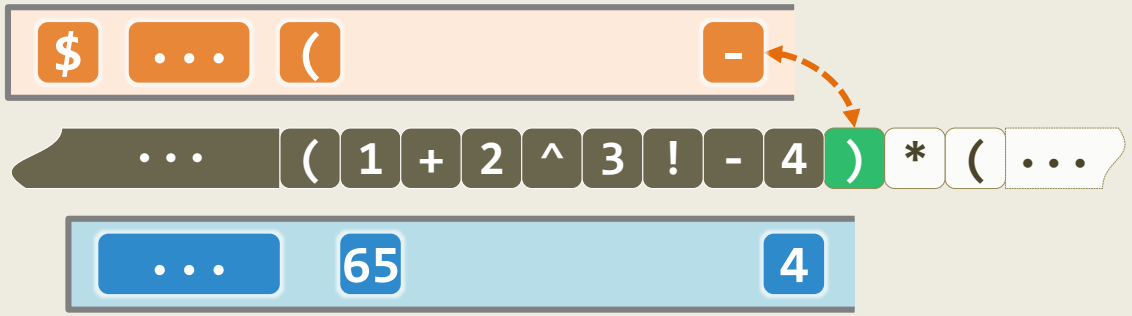
'=': 终须了断: 实例

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

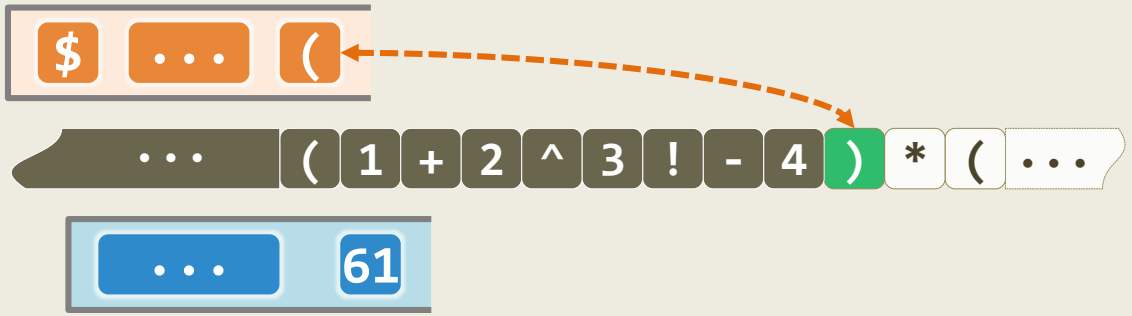


pri['*']['$'] = '>'





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

')': 死线已至（然后满血复活）

```
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 --------------|
};
```