
北京邮电大学

编译原理实验报告



题目: LL(1)语法分析程序的实现

姓 名: 谢蔚钦

学 院: 计算机科学与技术学院

专 业: 网络工程

班 级: 2020211315

学 号: 2020211451

指导老师: 王雅文

2022 年 11 月 10 日

一、实验内容及要求

实验内容：编写语法分析程序，实现对算术表达式的语法分析。要求所分析算数表达式由如下的文法产生：

$$E \rightarrow E+T \mid E-T \mid T$$
$$T \rightarrow T * F \mid T / F \mid F$$
$$F \rightarrow (E) \mid \text{num}$$

实验要求：在对输入的算术表达式进行分析的过程中，依次输出所采用的产生式。

编写 LL(1) 语法分析程序，要求如下：

(1) 编程实现预测分析表的构造方法，为给定文法自动构造预测分析表。

(2) 编程实现非递归预测分析方法，构造 LL(1) 预测分析程序。

二、实验环境

1. 编程语言：c 语言
2. 操作系统：windows11
3. 编译器：gcc 12.2.0

三、实验流程

观察所给出的文法，发现其存在左递归，要对其进行改写，消除左递归，将其改写为 LL(1) 文法，如下所示：

@表示空串

$E \rightarrow TP$

$P \rightarrow +TP \mid -TP \mid @$

$T \rightarrow FM$

$M \rightarrow *FM \mid /FM \mid @$

$F \rightarrow (E) \mid n$

然后需要计算出改写后的文法中的非终结字符的 FIRST 集和 FOLLOW 集，如下所示：

$\text{FIRST}(E) = \{ "(", "n" \}$

$\text{FOLLOW}(E) = \{ ")", "$" \}$

$\text{FIRST}(P) = \{ "+", "-", "@" \}$

$\text{FOLLOW}(P) = \{ ")", "$" \}$

$\text{FIRST}(T) = \{ "(", "n" \}$

$\text{FOLLOW}(T) = \{ "+", "-", "$" \}$

$\text{FIRST}(M) = \{ "*", "/", "@" \}$

$\text{FOLLOW}(M) = \{ "+", "-", ")", "$" \}$

$\text{FIRST}(F) = \{ "(", "n" \}$

$\text{FOLLOW}(F) = \{ "*", "/", "$" \}$

改写后的文法以及 FIRST 集和 FOLLOW 集均使用链表进行存储，结构体代码如下所示：

```
typedef struct {  
    char cur;
```

```
generator *fol;
```

```
}nonterminal;
```

这个结构体用于存储上下文无关文法中的生成式的左半部分，其中 `cur` 存储当前的非终结符；

```
typedef struct generator {
```

```
    char *formual;
```

```
    struct generator *next;
```

```
}generator;
```

这个结构体用于存储生成式的右半部分，其中 `formula` 指向当前的字符串。

对于求 FIRST 集和 FOLLOW 集，构造了两个函数，分别返回非终结符对应 FIRST 集和 FOLLOW 的符号串，如下所示：

```
char *FIRST(char c) {  
    if (c == 'E') {  
        return "(n";  
    } else if (c == 'P') {  
        return "+-@";  
    } else if (c == 'T') {  
        return "(n";  
    } else if (c == 'M') {  
        return "*/@";  
    } else {
```

```

        return "(n";
    }
}

char *FOLLOW(char c) {
    if (c == 'E') {
        return ")$";
    } else if (c == 'P') {
        return ")$";
    } else if (c == 'T') {
        return "+-$";
    } else if (c == 'M') {
        return "+-$";
    } else {
        return "*/$";
    }
}

```

接下来构造预测分析表，构造分析表的伪代码如下所示：

输入：文法 G

输出：文法 G 的预测分析表 M 方法：

for (文法 G 的每个产生式 $A \rightarrow \alpha$) {

for (每个终结符号 $a \in \text{FIRST}(\alpha)$)

把 $A \rightarrow \alpha$ 放入 $M[A, a]$ 中;

if ($\epsilon \in \text{FIRST}(\alpha)$)

for (任何 $b \in \text{FOLLOW}(A)$)

把 $A \rightarrow \alpha$ 放入 $M[A, b]$ 中;

};

for (所有无定义的 $M[A, a]$) 标上错误标志。

用于存储预测分析表的结构体如下所示:

```
typedef struct {
```

```
    char begin;
```

```
    table *next;
```

```
}form;
```

其中 `begin` 的值的是当前的非终结字符, `next` 指向的是对应的终结符的生成式, `table` 结构体代码如下所示:

```
typedef struct table {
```

```
    char cur;
```

```
    char *process;
```

```
    struct table *next;
```

```
}table;
```

其中 `cur` 存储的是当前的终结字符, `process` 指向的是对应的生成式。

在构造完预测分析表后, 开始预测分析程序, 其中需要使用到栈, 代码如下所示:

```
typedef struct Stack{
    char item[BUFFER_SIZE];
    int len;
} Stack;
```

BUFFER_SIZE 的大小为 100（可根据需要进行修改）。

预测分析程序的工作流程如下所示：

输入：输入符号串 ω ，文法 G 的一张预测分析表 M 。输出：

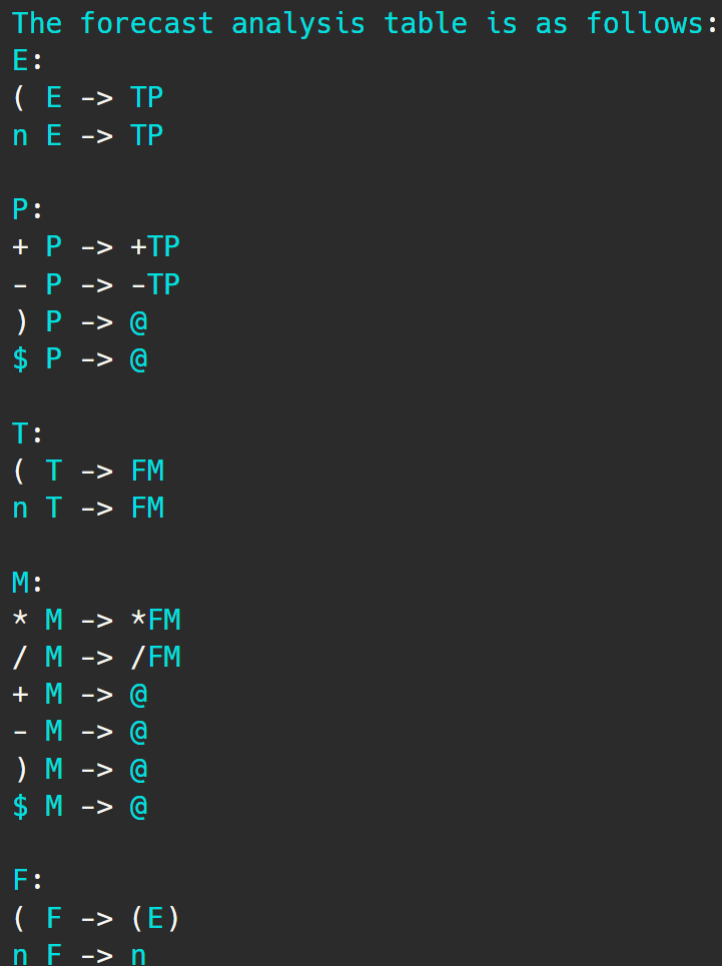
若 ω 在 $L(G)$ 中，则输出 ω 的最左推导，否则报告错误。方法：
分析开始时， $\$$ 在栈底，文法开始符号 S 在栈顶， $\omega \$$ 在输入缓冲区中置 ip 指向 $\omega \$$ 的第一个符号；

```
do {
    令  $X$  是栈顶符号， $a$  是  $ip$  所指向的符号； if ( $X$  是终结符
    号或 $\$$ ) {
        if ( $X==a$ ) {
            从栈顶弹出  $X$ ；  $ip$  前移一个位置；
        };
        else error();
        else /*  $X$  是非终结符号 */
        if ( $M[X, a]=X\omega Y_1Y_2...Y_k$ 
        ) {
            从栈顶弹出  $X$ ；
            把  $Y_k$ 、 $Y_{k-1}$ 、 $\dots$ 、 $Y_2$ 、 $Y_1$  压入栈， $Y_1$  在栈顶；
```

```
    输出产生式  $X \omega Y_1 Y_2 \dots Y_k$ ;  
};  
else error();  
} while(X!= $\$$ ) /* 栈不空, 继续 */
```

四、实验结果

对源代码进行编译, 结果如下所示:



```
The forecast analysis table is as follows:  
E:  
( E -> TP  
n E -> TP  
  
P:  
+ P -> +TP  
- P -> -TP  
) P -> @  
$ P -> @  
  
T:  
( T -> FM  
n T -> FM  
  
M:  
* M -> *FM  
/ M -> /FM  
+ M -> @  
- M -> @  
) M -> @  
$ M -> @  
  
F:  
( F -> (E)  
n F -> n
```

对结果进行验证, 发现所构造的预测分析表不存在冲突, 表明该文法是 LL(1)型文法。

接下来输入字符串 $(n*n-(n-n)*n/n)+(n/n+n*n)*n$ ，对预测分析程序进行验证：

```

Please enter the string you want to analyze:
(n*n-(n-n)*n/n)+(n/n+n*n)*n

Stack  input  output
$E      (n*n-(n-n)*n/n)+(n/n+n*n)*n$  @
$PT      (n*n-(n-n)*n/n)+(n/n+n*n)*n$  E -> TP
$PMF      (n*n-(n-n)*n/n)+(n/n+n*n)*n$  T -> FM
$PM)E(    (n*n-(n-n)*n/n)+(n/n+n*n)*n$  F -> (E)
$PM)E      n*n-(n-n)*n/n)+(n/n+n*n)*n$  @
$PM)PT      n*n-(n-n)*n/n)+(n/n+n*n)*n$  E -> TP
$PM)PMF      n*n-(n-n)*n/n)+(n/n+n*n)*n$  T -> FM
$PM)PMn      n*n-(n-n)*n/n)+(n/n+n*n)*n$  F -> n
$PM)PM      *n-(n-n)*n/n)+(n/n+n*n)*n$  @
$PM)PMF*      *n-(n-n)*n/n)+(n/n+n*n)*n$  M -> *FM
$PM)PMF      n-(n-n)*n/n)+(n/n+n*n)*n$  @
$PM)PMn      n-(n-n)*n/n)+(n/n+n*n)*n$  F -> n
$PM)PM      -(n-n)*n/n)+(n/n+n*n)*n$  @
$PM)P      -(n-n)*n/n)+(n/n+n*n)*n$  M -> @
$PM)PT-      -(n-n)*n/n)+(n/n+n*n)*n$  P -> -TP
$PM)PT      (n-n)*n/n)+(n/n+n*n)*n$  @
$PM)PMF      (n-n)*n/n)+(n/n+n*n)*n$  T -> FM
$PM)PM)E(    (n-n)*n/n)+(n/n+n*n)*n$  F -> (E)
$PM)PM)E      n-n)*n/n)+(n/n+n*n)*n$  @
$PM)PM)PT      n-n)*n/n)+(n/n+n*n)*n$  E -> TP
$PM)PM)PMF      n-n)*n/n)+(n/n+n*n)*n$  T -> FM
$PM)PM)PMn      n-n)*n/n)+(n/n+n*n)*n$  F -> n
$PM)PM)PM      -n)*n/n)+(n/n+n*n)*n$  @
$PM)PM)P      -n)*n/n)+(n/n+n*n)*n$  M -> @
$PM)PM)PT-      -n)*n/n)+(n/n+n*n)*n$  P -> -TP
$PM)PM)PT      n)*n/n)+(n/n+n*n)*n$  @
$PM)PM)PMF      n)*n/n)+(n/n+n*n)*n$  T -> FM
$PM)PM)PMn      n)*n/n)+(n/n+n*n)*n$  F -> n
$PM)PM)PM      )*n/n)+(n/n+n*n)*n$  @
$PM)PM)P      )*n/n)+(n/n+n*n)*n$  M -> @
$PM)PM)      )*n/n)+(n/n+n*n)*n$  P -> @
$PM)PM      *n/n)+(n/n+n*n)*n$  @
$PM)PMF*      *n/n)+(n/n+n*n)*n$  M -> *FM
$PM)PMF      n/n)+(n/n+n*n)*n$  @
$PM)PMn      n/n)+(n/n+n*n)*n$  F -> n
$PM)PM      /n)+(n/n+n*n)*n$  @
$PM)PMF/      /n)+(n/n+n*n)*n$  M -> /FM
$PM)PMF      n)+(n/n+n*n)*n$  @
$PM)PMn      n)+(n/n+n*n)*n$  F -> n
$PM)PM      )+(n/n+n*n)*n$  @
$PMF*      *n$  M -> *FM
$PMF      n$  @
$PMn      n$  F -> n
$PM      $  @
$P      $  M -> @
$      $  P -> @
success to match
```

发现能够识别出该字符串，进行多次类似实验后，该程序总能识别出相应的字符串，表明该程序正确无误，能够识别出包含加减乘除的带括号的计算式。（其中 n 表明任意一个数字）

六、实验心得与总结

本次实验给出的文法比较简单，适用于识别加减乘除简单计算式，首先要对文法进行改写，消除左递归，接着构造出非终结字符对应的 FIRST 集和 FOLLOW 集，然后构造出预测分析表，再利用预测分析表，构造出一个栈，这个栈根据输入的字符串和预测分析表的内容来载入或者弹出字符串，最终当栈中只剩\$符号时，表示该字符串识别成功。

通过本次实验，让我明白了如何通过自顶向下的方法构造出非递归预测分析程序，使我对其中的一些构造细节有了更明确的认知，同时让我对 C 语言中指针的应用更加得心应手，有利于我在编译原理的学习中更进一步。