南京信息工程大学滨江学院
实验（实习）报告

实验（实习）名称 编译原理判断LL（1）文法 **实验（实习）日期** 9月25日 **得分** ____ **指导教师** 林美华

软件工程 专业 2018 **年级** 2 **班次 姓名** 毛济洲 **学号** 20182344050

一．实验目的

通过完成预测分析法的语法分析程序，了解预测分析法和递归子程序法的区别和联系。熟悉判断LL（1）文法的方法及对某一输入串的分析过程。学会构造表达式文法的预测分析表。

二．实验设备及仪器

PC电脑、visual studio c++

三．涉及的知识

LL（1）分析法，就是指从左到右扫描输入串（源程序），同时采用最左推导，且对每次直接推导只需向前看一个输入符号，便可确定当前所应当选择的规则。实现LL（1）分析的程序又称为LL（1）分析程序或LL（1）分析器。

一个文法要能进行LL（1）分析，那么这个文法应该满足：无二义性，无左递归，无左公因子。当文法满足条件后，再分别构造文法每个非终结符的FIRST和FOLLOW集合，然后根据FIRST和FOLLOW集合构造LL（1）分析表，最后利用分析表，根据LL(1)语法分析构造一个分析器。

LL（1）的语法分析程序包含了三个部分：控制程序，预测分析表函数，先进先出的语法分析栈。

LL（1）预测分析程序的总控程序在判断STACK栈顶符号X和当前的输入符号a做哪种过程的。对于任何（X，a），总控程序每次都执行下述三种可能的动作之一：

（1）若X = a =‘#’，则宣布分析成功，停止分析过程。

（2）若X = a ！ =‘#’，则把X从STACK栈顶弹出，让a指向下一个输入符号。

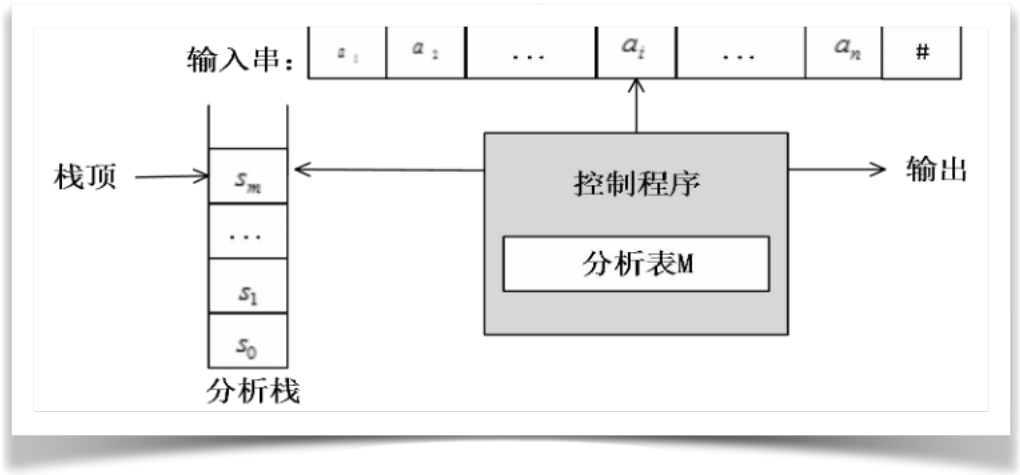
（3）若X是一个非终结符，则查看预测分析表M。若M[A，a]中存放着关于X的一个产生式，那么，首先把X弹出STACK栈顶，然后，把产生式的右部符号串按反序一一弹出STACK栈（若右部符号为ε，则不推什么东西进STACK栈）。若M[A，a]中存放着“出错标志”，则调用出错诊断程序ERROR。

1 LL(1)分析法

LL(1)分析法又称预测分析法，是一种不带回溯的非递归自顶向下分析法。

LL(1)的含义是：第一个 L 表明自顶向下分析是从左至右扫描输入串的；第二个 L 表明分析过程中将用最左推导；“1”表明只需向右查看一个符号就可以决定如 何推导（即可知用哪一个产生式进行推导）。

LL(1)分析器结构如图：



使用 LL(1)分析法将会涉及到 LL(1)分析表，而分析表又会涉及到 FIRST 集 和 FOLLOW 集。

FIRST 集构造

对文法中的每一个非终结符 X 构造 FIRST(X)，其方法是连续使用以下规则， 直到每个集合的 FIRST 不再增大为止。

?若有产生式 $X \rightarrow a...$，且 $a \in V_T$，则把 a 加入到 FIRST(X)中；若存在 $X \rightarrow \varepsilon$， 则将ε也加入到 FIRST(X)中。

?若有 $X \rightarrow Y$ ...且 $Y \in V_N$，则将 FIRST(Y)中的所有非ε元素（记为"\{ε}"）都 加入到 FIRST(X)中；若有 $X \rightarrow Y_1Y_2...Y_K$，且 $Y_1 \sim Y_i$ 都是非终结符，而 $Y_1 \sim Y_i$ 的候 选式都有ε存在，则把 FIRST(Y_j)(j=1,2,...i)的所有非ε元素都加入到 FIRST(X)中； 特别是当 $Y_1 \sim Y_k$ 均含有ε产生式时，应把ε也加入到 FIRST(X)中。

FOLLOW 集构造

对文法 G[S]的每一个非终结符 A 构造 FOLLOW(A)，其方法是连续使用以下 规则，直到每个集合的 FOLLOW 不再增大为止。

?对文法开始符号 S，置#于 FOLLOW(S)中（由语句括号"#S#"中的 S#得到）。

?若有 $A \rightarrow \alpha B\beta$ （α可为空），则将 FIRST(β)\{ε}加入到 FOLLOW(B)中。

*

?若有 $A \rightarrow \alpha B$ 或 $A \rightarrow \alpha B\beta$，且 $\beta \Rightarrow \varepsilon$ （即$\varepsilon \in$ FIRST (β) ），则把 FOLLOW(A)

加到 FOLLOW(B)中（此处α也可为空）。

四．实验任务

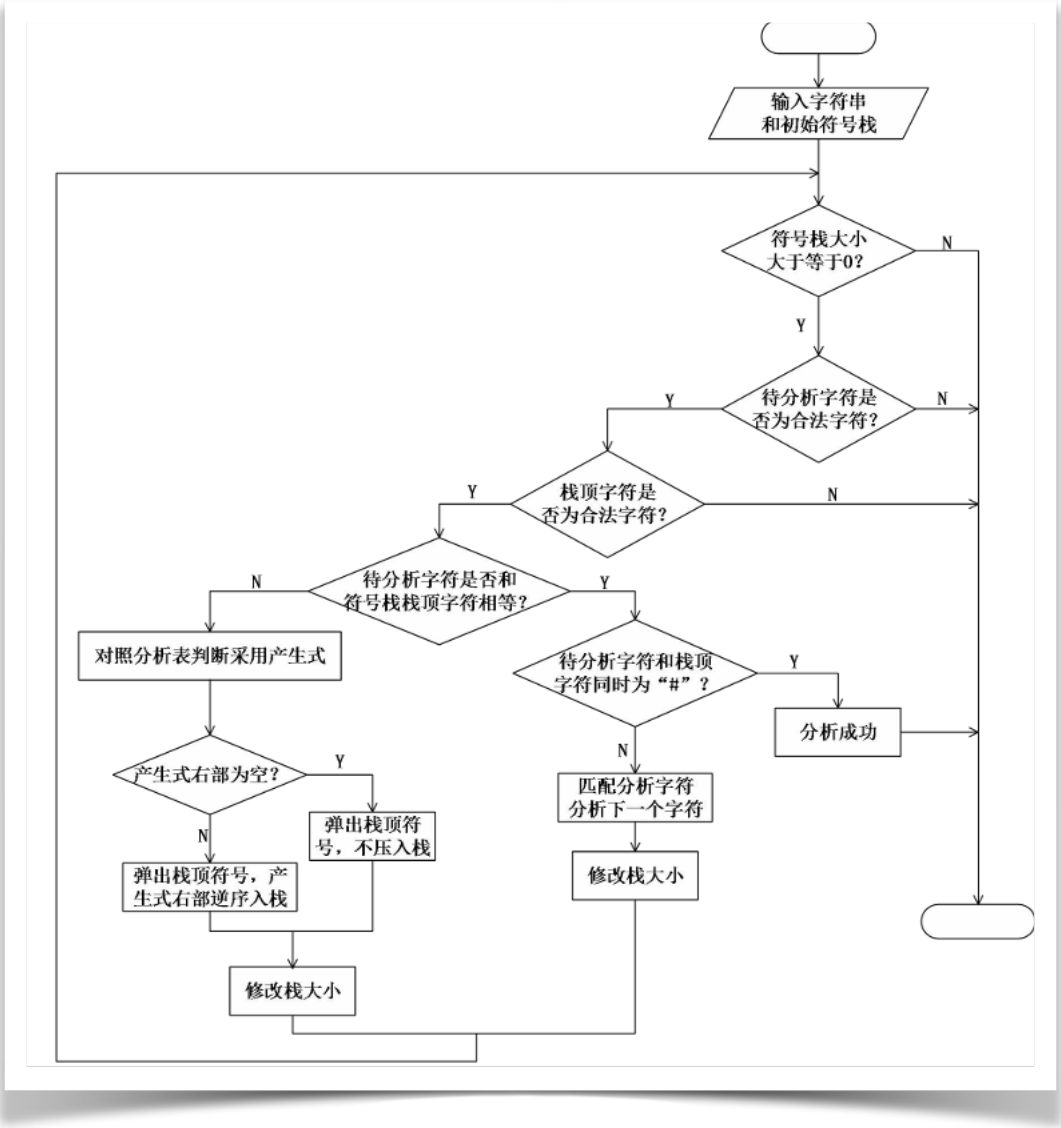
1.从键盘读入某个文法所有产生式。

2.判断该文法是否是LL（1）文法。若不是则做消除左递归和提取左公因子。若仍不是LL（1）文法则该文法不能用预测分析法来分析。

3.识别所有终结符和非终结符，构造所有非终结符的First和Follow集，由程序自动构造LL（1）分析表 。

4. 从键盘读入要识别的符号串。

5. 由程序根据分析表进行预测分析得出结果。



五．实验程序及结果

1. 消除左递归

```
void recur( char *point )
{ /*完整的产生式在 point[] 中*/
 int    j, m = 0, n = 3, k;
 char temp[20], ch;
 ch          = c(); /*得到一个非终结符 */
 k           = strlen( non_ter );
 non_ter[k]= ch;
 non_ter[k + 1]  = '\0';
 for ( j = 0; j <= strlen( point ) – 1; j++ )
```

```
{
    if ( point[n] == point[0] )
    {               /* 如果 ?| ‘ 后的首符号和左部相同 */
        for ( j = n + 1; j <= strlen( point ) – 1; j++ )
        {
            while ( point[j] != '|' && point[j] != '\0' )
                temp[m++] = point[j++];
            left[count] = ch;
            memcpy( right[count], temp, m );
            right[count][m]     = ch;
            right[count][m + 1]  = '\0';
            m                   = 0;
            count++;
            if ( point[j] == '|' )
            {
                n = j + 1;
                break;
            }
        }
    }else { /*如果 ?| ‘ 后的首符号和左部不同 */
        left[count]     = ch;
        right[count][0] = '^';
        right[count][1] = '\0';
        count++;
        for ( j = n; j <= strlen( point ) – 1; j++ )
        {
            if ( point[j] != '|' )
                temp[m++] = point[j];
            else{
                left[count] = point[0];
                memcpy( right[count], temp, m );
                right[count][m]     = ch;
                right[count][m + 1]  = '\0';
                printf( " count=%d ", count );
                m = 0;
                count++;
            }
        }
        left[count] = point[0];
        memcpy( right[count], temp, m );
        right[count][m]     = ch;
        right[count][m + 1]  = '\0';
        count++;
        m = 0;
    }
}
}
```

2. 构建FIRST和FOLLOW集

```
void First( int U )
{
 int i, j;
 for ( i = 0; i < PNum; i++ )
 {
    if ( P[i].lCursor == U )
    {
        struct pRNode* pt;
        pt    = P[i].rHead;
        j     = 0;
        while ( j < P[i].rLength )
        {
            if ( 100 > pt->rCursor )
            {
/*注:此处因编程出错 ,使空产生式时
 * rlength 同样是 1,故此处同样可处理空产生式 */
                AddFirst( U, pt->rCursor );
```

```
                        break;
                    }else  {
                        if ( NULL == first[pt->rCursor - 100] )
                        {
                            First( pt->rCursor );
                        }
                        AddFirst( U, pt->rCursor );
                        if ( !HaveEmpty( pt->rCursor ) )
                        {
                            break;
                        }else  {
                            pt = pt->next;
                        }
                    }
                    j++;
                }
                if ( j >= P[i].rLength ) /* 当产生式右部都能推出空时 */
                    AddFirst( U, -1 );
            }
        }
    }

/*加入 first 集*/
void AddFirst( int U, int nCh )        /* 当数值小于 100 时 nCh 为 Vt*/
/*当处理非终结符时 ,AddFirst 不添加空项 (-1)*/
{
    struct collectNode    *pt, *qt;
    int                ch;     /* 用于处理 Vn*/
    pt    = NULL;
    qt    = NULL;
    if ( nCh < 100 )
    {
        pt = first[U - 100];
        while ( NULL != pt )
        {
            if ( pt->nVt == nCh )
                break;
            else{
                qt    = pt;
                pt    = pt->next;
            }
        }
        if ( NULL == pt )
        {
            pt          = (struct collectNode *) malloc( sizeof(struct collectNode) );
            pt->nVt          = nCh;
            pt->next   = NULL;
            if ( NULL == first[U - 100] )
            {
                first[U - 100] = pt;
            }else  {
                qt->next = pt; /*qt 指向 first 集的最后一个元素 */
            }
            pt = pt->next;
        }
    }else  {
        pt = first[nCh - 100];
        while ( NULL != pt )
        {
            ch = pt->nVt;
            if ( -1 != ch )
            {
                AddFirst( U, ch );
            }
            pt = pt->next;
```

```
        }
    }
}


/*判断 first 集中是否有空 (-1)*/
bool HaveEmpty( int nVn )
{
    if ( nVn < 100 ) /* 为终结符时 (含-1)，在 follow 集中用到 */
        return(false);
    struct collectNode *pt;
    pt = first[nVn - 100];
    while ( NULL != pt )
    {
        if ( -1 == pt->nVt )
            return(true);
        pt = pt->next;
    }
    return(false);
}


/*计算 follow 集,例： U->xVy,U->xV.( 注：初始符必含 #—— "-1")*/
void Follow( int V )
{
    int         i;
    struct pRNode   *pt;
    if ( 100 == V )                                 /* 当为初始符时 */
        AddFollow( V, -1, 0 );
    for ( i = 0; i < PNum; i++ )
    {
        pt = P[i].rHead;
        while ( NULL != pt && pt->rCursor != V )        /* 注此不能处理： U->xVyVz 的情况 */
            pt = pt->next;
        if ( NULL != pt )
        {
            pt = pt->next;                          /*V 右侧的符号 */
            if ( NULL == pt )                         /* 当 V 后为空时 V->xV ，将左符的 follow 集并入 V 的 follow
集中 */
            {
                if ( NULL == follow[P[i].lCursor - 100] && P[i].lCursor != V )
                {
                    Follow( P[i].lCursor );
                }
                AddFollow( V, P[i].lCursor, 0 );
            }else { /* 不为空时 V->xVy,( 注意： y->)，调用 AddFollow 加入 Vt 或 y 的 first 集*/
                while ( NULL != pt && HaveEmpty( pt->rCursor ) )
                {
                    AddFollow( V, pt->rCursor, 1 ); /*y 的前缀中有空时，加如 first 集*/
                    pt = pt->next;
                }
                if ( NULL == pt )                       /* 当后面的字符可以推出空时 */
                {
                    if ( NULL == follow[P[i].lCursor - 100] && P[i].lCursor != V )
                    {
                        Follow( P[i].lCursor );
                    }
                    AddFollow( V, P[i].lCursor, 0 );
                }else { /* 发现不为空的字符时 */
                    AddFollow( V, pt->rCursor, 1 );
                }
            }
        }
    }
}
```

```
/*当数值小于 100 时 nCh 为 Vt*/


/*#用-1 表示 ,kind 用于区分是并入符号的 first 集，还是 follow 集
 * kind = 0 表加入 follow 集， kind = 1 加入 first 集*/
void AddFollow( int V, int nCh, int kind )
{
 struct collectNode    *pt, *qt;
 int                ch;     /* 用于处理 Vn*/
 pt     = NULL;
 qt     = NULL;
 if ( nCh < 100 )              /* 为终结符时 */
 {
       pt = follow[V - 100];
       while ( NULL != pt )
       {
             if ( pt->nVt == nCh )
                   break;
             else{
                   qt     = pt;
                   pt     = pt->next;
             }
       }
       if ( NULL == pt )
       {
             pt          = (struct collectNode *) malloc( sizeof(struct collectNode) );
             pt->nVt           = nCh;
             pt->next    = NULL;
             if ( NULL == follow[V - 100] )
             {
                   follow[V - 100] = pt;
             }else  {
                   qt->next = pt; /*qt 指向 follow 集的最后一个元素 */
             }
             pt = pt->next;
       }
 }else  { /* 为非终结符时，要区分是加 first 还是 follow*/
       if ( 0 == kind )
       {
             pt = follow[nCh - 100];
             while ( NULL != pt )
             {
                   ch = pt->nVt;
                   AddFollow( V, ch, 0 );
                   pt = pt->next;
             }
       }else  {
             pt = first[nCh - 100];
             while ( NULL != pt )
             {
                   ch = pt->nVt;
                   if ( -1 != ch )
                   {
                         AddFollow( V, ch, 1 );
                   }
                   pt = pt->next;
             }
       }
 }
```

4.分析表进行预测分析

```
void Identify( char *st )
{
 int current, step, r; /*r 表使用的产生式的序号 */
```

```
 printf( "\n%s 的分析过程：  \n", st );
 printf( " 步骤 \t 分析符号栈 \t 当前指示字符 \t 使用产生式序号 \n" );
 step  = 0;
 current = 0; /* 符号串指示器 */
 printf( "%d\t", step );
 ShowStack();
 printf( "\t\t%c\t\t- -\n", st[current] );
 while ( '#' != st[current] )
 {
       if ( 100 > analyseStack[topAnalyse] ) /* 当为终结符时 */
       {
             if ( analyseStack[topAnalyse] == IndexCh( st[current] ) )
             {
/*匹配出栈，指示器后移 */
                   Pop();
                   current++;
                   step++;
                   printf( "%d\t", step );
                   ShowStack();
                   printf( "\t\t%c\t\t 出栈、后移 \n", st[current] );
             }else {
                   printf( "%c-%c 不匹配！  ", analyseStack[topAnalyse], st[current] );
                   printf( " 此串不是此文法的句子！  \n" );
                   return;
             }
       }else { /* 当为非终结符时 */
             r = analyseTable[analyseStack[topAnalyse] - 100][IndexCh( st[current] )];
             if ( -1 != r )
             {
                   Push( r ); /* 产生式右部代替左部，指示器不移动 */
                   step++;
                   printf( "%d\t", step );
                   ShowStack();
                   printf( "\t\t%c\t\t%d\n", st[current], r );
             }else {
                   printf( " 无可用产生式，此串不是此文法的句子！  \n" );
                   return;
             }
       }
 }
 if ( '#' == st[current] )
 {
       if ( 0 == topAnalyse && '#' == st[current] )
       {
             step++;
             printf( "%d\t", step );
             ShowStack();
             printf( "\t\t%c\t\t 分析成功！  \n", st[current] );
             printf( "%s 是给定文法的句子！  \n", st );
       }else {
             while ( topAnalyse > 0 )
             {
                   if ( 100 > analyseStack[topAnalyse] ) /* 当为终结符时 */
                   {
                         printf( " 无可用产生式，此串不是此文法的句子！  \n" );
                         return;
                   }else {
                         r = analyseTable[analyseStack[topAnalyse] - 100][vtNum];
                         if ( -1 != r )
                         {
                               Push( r ); /* 产生式右部代替左部，指示器不移动 */
                               step++;
                               printf( "%d\t", step );
                               ShowStack();
                               if ( 0 == topAnalyse && '#' == st[current] )
                               {
```

```
                        printf( "\t\t%c\t\t 分析成功！  \n", st[current] );
                        printf( "%s 是给定文法的句子！  \n", st );
                    }else
                        printf( "\t\t%c\t\t%d\n", st[current], r );
                }else {
                    printf( " 无可用产生式，此串不是此文法的句子！  \n" );
                    return;
                }
            }
        }
    }
}
}
```

六、实验结论

本实验程序较好地完成了预测分析法分析程序的设计与实现，能够对所给字符串进行识别，判断是否是给定文法的句子。

七、总结及心得体会

通过这次的实验，我了解了LL(1)文法预测分析法设计和实现，加深了对LL（1）文法的理解和认识，使我的编译原理的知识更加巩固，而且可以使理论与实践相结合，更好的掌握所学知识。