**语法分析程序设计**

**一、递归下降分析器**

1.实验目的：通过设计、编制、调试递归下降语法分析程序，对输入的符号串进行分析匹配，观察输入符号串是否为给定文法的句子。

2.实验内容：递归下降分析法是一种自顶向下的分析方法，文法的每个非终结符对应一个递归过程（函数）。分析过程就是从文法开始符出发执行一组递归过程（函数），这样向下推导直到推出句子；或者说从根节点出发，自顶向下为输入串寻找一个最左匹配序列，建立一棵语法树。  
  在不含左递归和每个非终结符的所有候选终结首字符集都两两不相交条件下，我们就可能构造出一个不带回溯的自顶向下的分析程序，这个分析程序是由一组递归过程（或函数）组成的，每个过程（或函数）对应文法的而一个非终结符。这样的一个分析程序称为递归下降分析器。

文法G(E)为：G[E]:E -> E+T | T

T -> T\*F | F

F -> (E) | i

3.实验要求：

(1)输入一以#结束的符号串(包括+—\*/（）i#)：在此位置输入符号串例如：i+i\*i#

(2)输出结果：i+i\*i#为合法符号串

(3)输入一符号串如i+i\*#,要求输出为“非法的符号串”。

4.实验原理及流程图

递归下降分析程序的实现思想是：识别程序由一组子程序组成。每个子程序对应于一个非终结符号。每一个子程序的功能是：选择正确的右部，扫描完相应的字。在右部中有非终结符号时，调用该非终结符号对应的子程序来完成。自上向下分析过程中，如果带回溯，则分析过程是穷举所有可能的推导，看是否能推导出待检查的符号串。分析速度慢。而无回溯的自上向下分析技术，当选择某非终结符的产生时，可根据输入串的当前符号以及各产生式右部首符号而进行，效率高，且不易出错。

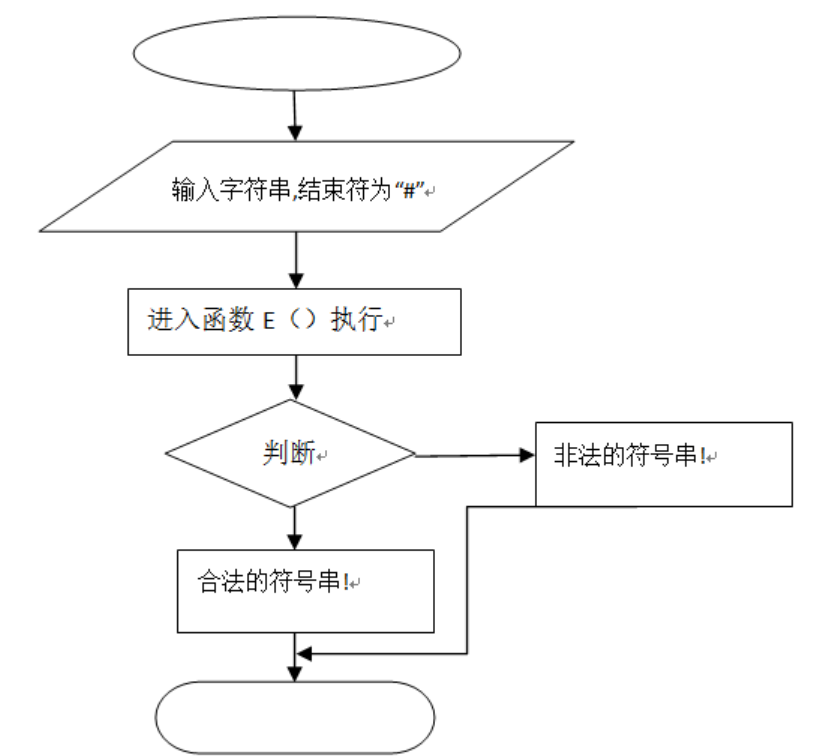
无回溯的自上向下分析技术可用的先决条件是：无左递归和无回溯。

无左递归：既没有直接左递归，也没有间接左递归。

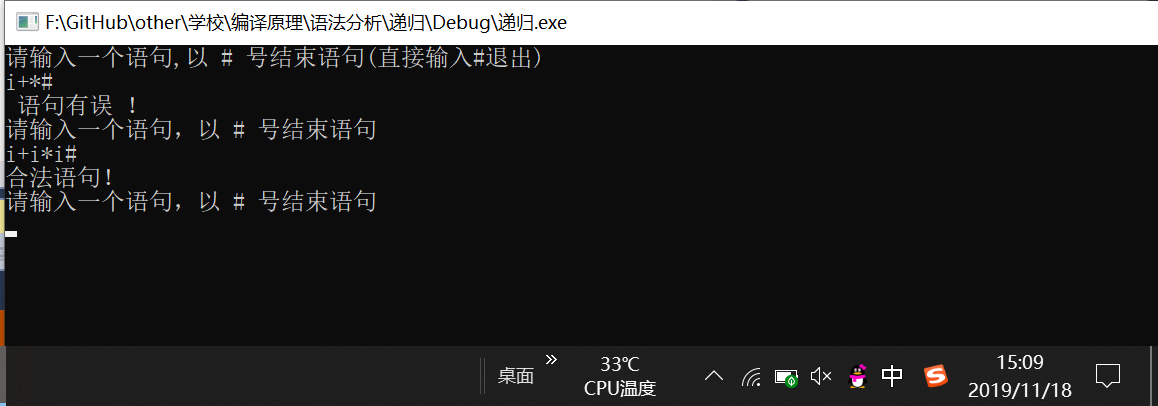
无回溯：对于任一非终结符号U的产生式右部x1|x2|…|xn，其对应的字的首终结符号两两不相交。

如果一个文法不含回路（形如P⇒+ P的推导），也不含以ε为右部的产生式，那么可以通过执行消除文法左递归的算法消除文法的一切左递归（改写后的文法可能含有以ε为右部的产生式）。

流程图：



5.结果截图：



6.源代码：

#define \_CRT\_SECURE\_NO\_WARNINGS

#include<stdio.h>

void E();

void T();

void E1();

void T1();

void F();

char s[100];

int i, SIGN;

int main()

{

printf("请输入一个语句,以 # 号结束语句(直接输入#退出)\n");

while (1)

{

SIGN = 0;

i = 0;

scanf("%s", &s);

if (s[0] == '#')

return 0;

E();

if (s[i] == '#')

printf("合法语句！\n");

printf("请输入一个语句，以 # 号结束语句 \n");

}

return 1;

}

void E()

{

if (SIGN == 0)

{

T();

E1();

}

}

void E1()

{

if (SIGN == 0)

{

if (s[i] == '+')

{

++i;

T();

E1();

}

else if (s[i] != '#'&&s[i] != ')')

{

printf("非法语句\n");

SIGN = 1;

}

}

}

void T()

{

if (SIGN == 0)

{

F();

T1();

}

}

void T1()

{

if (SIGN == 0)

{

if (s[i] == '\*')

{

++i;

F();

T1();

}

else if (s[i] != '#'&&s[i] != ')'&&s[i] != '+')

{

printf(" 语句有误 ！ \n");

SIGN = 1;

}

}

}

void F()

{

if (SIGN == 0)

{

if (s[i] == '(')

{

++i;

E();

if (s[i] == ')')

++i;

else if (s[i] == '#')

{

printf(" 语句有误 ！ \n");

SIGN = 1;

++i;

}

}

else if (s[i] == 'i')

++i;

else

{

printf(" 语句有误 ！ \n");

SIGN = 1;

}

}

}

7.实验总结：通过本次实验对递归下降词法分析器的结构，过程有了更进一步的了解，通过学习书本和试验原理书上的内容，对它的工作原理，具体实行步骤有了进一步的掌握，由于本次试验是测试性试验，所以要求输出的结果是成功与否，输入一个句型，进过分析，判断它是否合法，主要内容在于其判断过程中。本次试验不光提高了自己的编程能力，同时提高了对递归下降的了解。

**二、LL（1）分析法**

1.实验目的：通过设计、开发一个高级语言的 LL（1）语法分析程序，实现对源程序的语法检查和结构分析，加深对相关课堂教学内容（包括自顶向下语法分析、 First 集、 Follow 集、 Select 集、 判断 LL（1）文法的方法、 文法等价变换、 LL(1) 分析表的构造、对某一输入串的分析过程 的理解，提高语法分析方法的实践能力 。

2.实验要求：进行语法分析，判断其是否是合法的句子，给出句子的分析过程。

消除直接左递归前的文法         消除直接左递归后的等价文法  
G [E] ： E→E+T              G’[E] ：E→TE’  
     E→T                   E’→+TE’|ε  
     T→T\*F                  T→FT’  
     T→F                T’→\*FT’|ε  
      F→(E)|i                 F→(E)|i

具体要求如下：

（1）理解语法分析在编译程序中的作用；

（2）理解 LL(1) 语法分析方法对文法的要求 ( 必须是 LL(1) 文法 ) ；

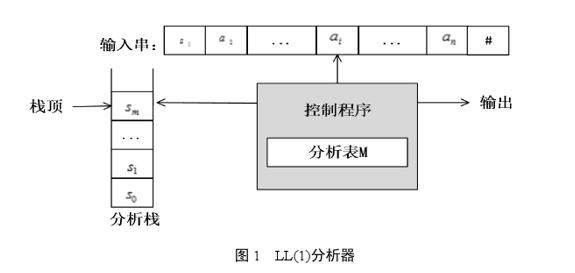
（3）理解 LL(1) 分析器模型；

（4）熟练掌握文法变换方法 ( 消除直接左递归和提取左公共因子 ) 。

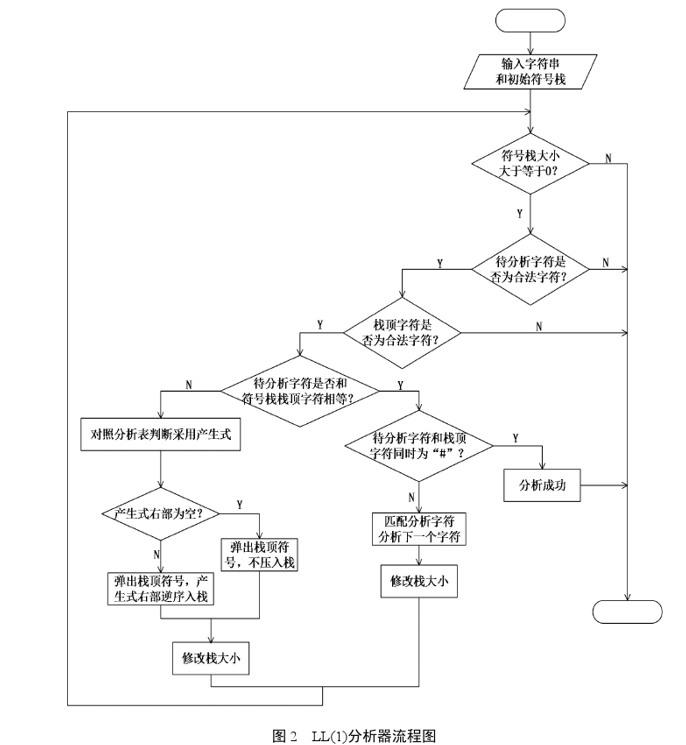
（5）熟练掌握 Select 集合的求解方法和 LL(1) 分析表的构造方法；

3.LL（1）分析法的基本思想是根据输入串的当前输入符号来确定唯一选用某条规则来进行推导：当这个输入符号与推导的第一个符号相同时，再取输入串的下一个符号，继续确定下一个推导应选的规则，直到推导出被分析的输入串为止。

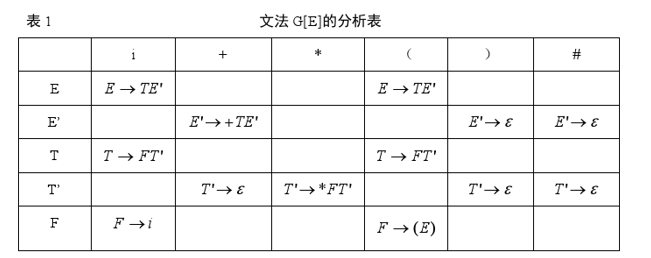
分析器：



分析器流程图：

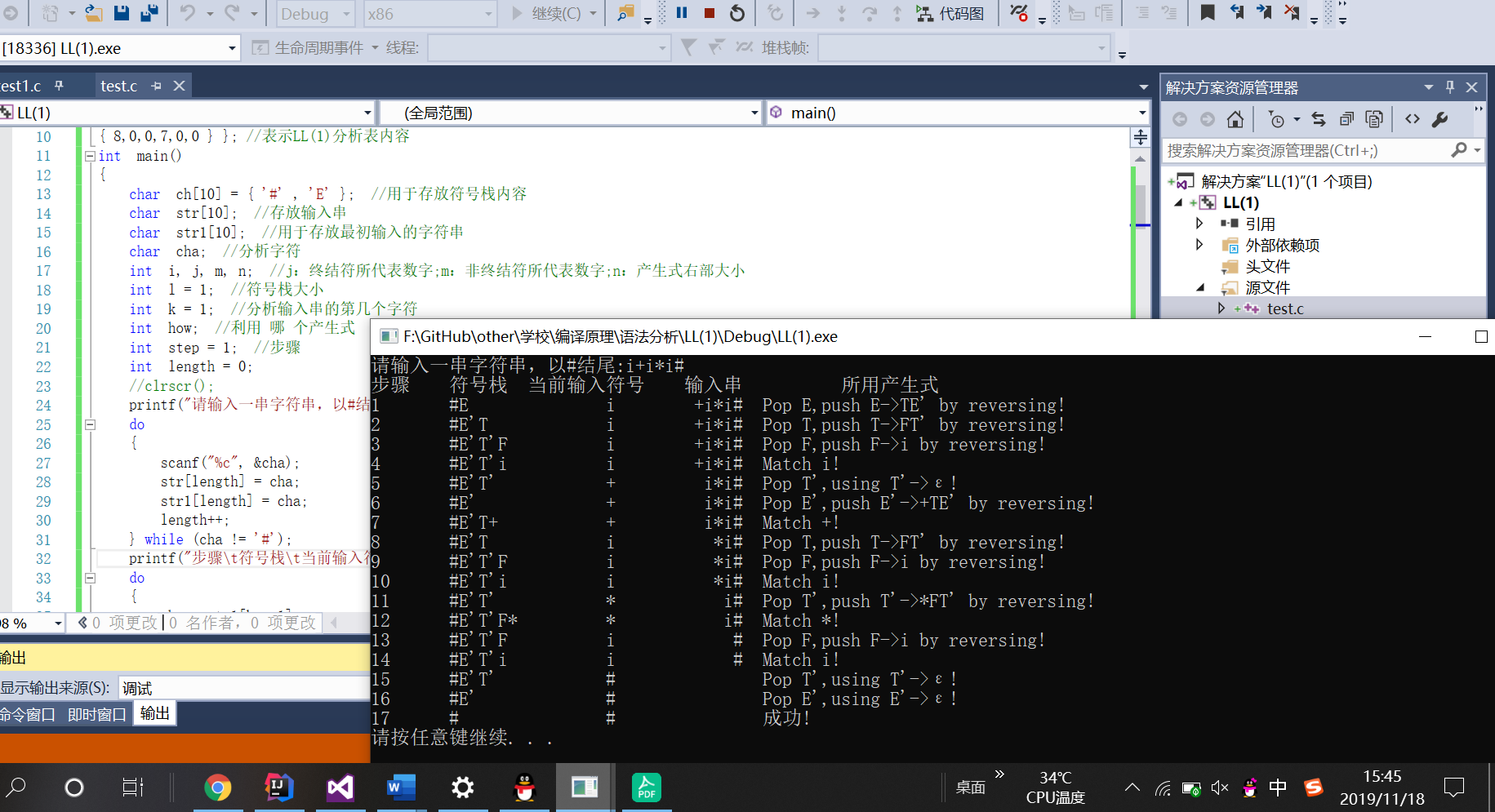


算数表达式文法的分析表，下图为i+i\*i的分析过程：

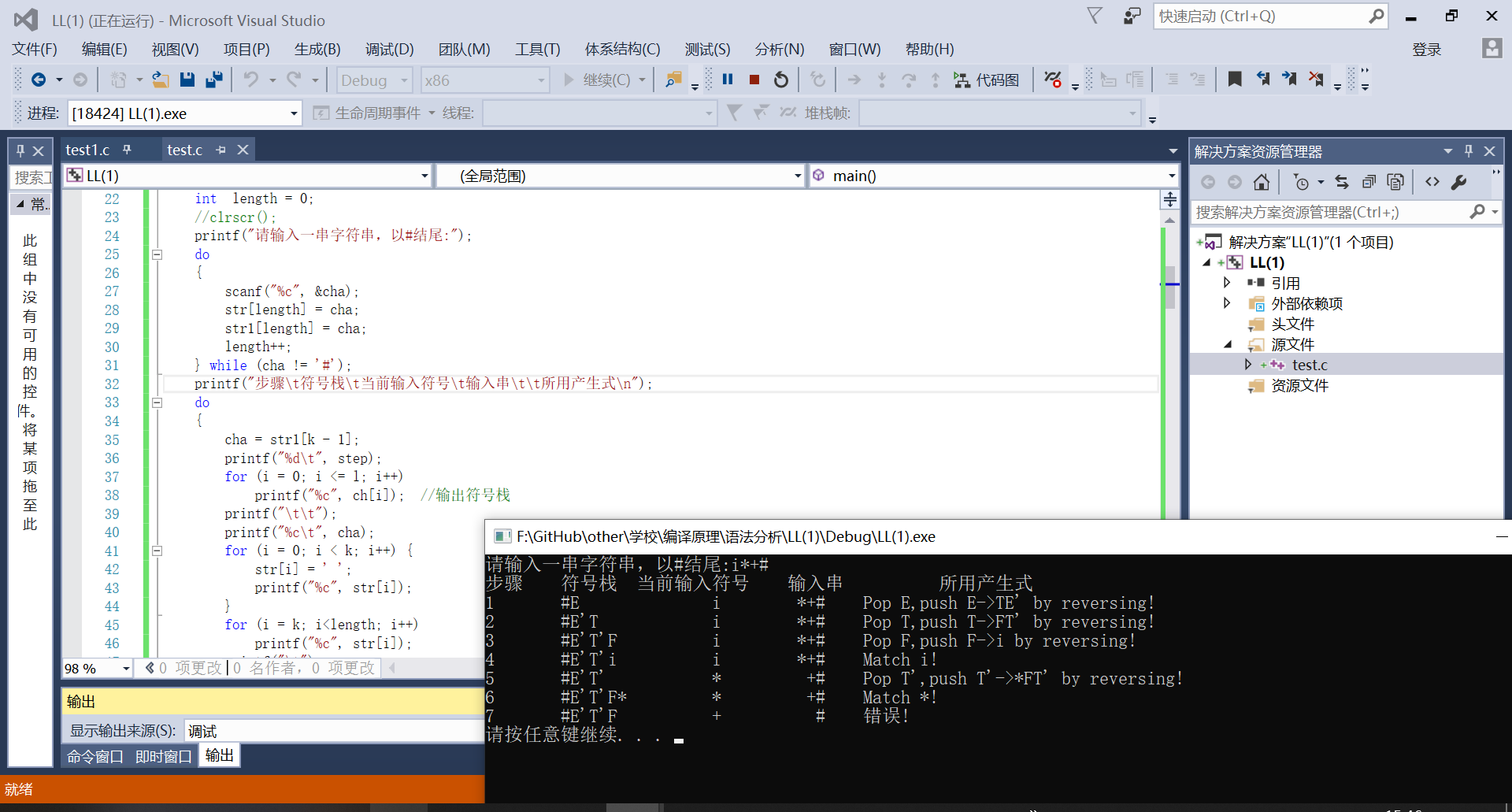


4.结果截图：

语法正确的分析结果：



语法错误的分析结果：



5.源代码：

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <string.h>

#include <stdio.h>

#include <stdlib.h>

#include <conio.h>

int ll1[5][6] = { { 1,0,0,1,0,0 },

{ 0,2,0,0,3,3 },

{ 4,0,0,4,0,0 },

{ 0,6,5,0,6,6 },

{ 8,0,0,7,0,0 } }; //表示LL(1)分析表内容

int main()

{

char ch[10] = { '#' , 'E' }; //用于存放符号栈内容

char str[10]; //存放输入串

char str1[10]; //用于存放最初输入的字符串

char cha; //分析字符

int i, j, m, n;//j：终结符所代表数字;m：非终结符所代表数字;n：产生式右部大小

int l = 1; //符号栈大小

int k = 1; //分析输入串的第几个字符

int how; //利用 哪 个产生式

int step = 1; //步骤

int length = 0;

printf("请输入一串字符串，以#结尾:");

do

{

scanf("%c", &cha);

str[length] = cha;

str1[length] = cha;

length++;

} while (cha != '#');

printf("步骤\t符号栈\t当前输入符号\t输入串\t\t所用产生式\n");

do

{

cha = str1[k - 1];

printf("%d\t", step);

for (i = 0; i <= l; i++)

printf("%c", ch[i]); //输出符号栈

printf("\t\t");

printf("%c\t", cha);

for (i = 0; i < k; i++) {

str[i] = ' ';

printf("%c", str[i]);

}

for (i = k; i<length; i++)

printf("%c", str[i]);

printf("\t");

switch (cha)

{

case 'i':

j = 0; break;

case '+':

j = 1; break;

case '\*':

j = 2; break;

case '(':

j = 3; break;

case ')':

j = 4; break;

case '#':

j = 5; break;

default:

j = -1; break;

}

if (j != -1) //正确的字符

{

if (ch[l] != cha)

{

if (ch[l] != 39)

{

switch (ch[l])

{

case 'E':

m = 0; break;

case 'T':

m = 2; break;

case 'F':

m = 4; break;

default:

m = -1; break;

}

}

else

{

switch (ch[l - 1])

{

case 'E':

m = 1; break;

case 'T':

m = 3; break;

default:

m = -1; break;

}

}

}

if (m != -1)

{

if (ch[l] != cha)

{

how = ll1[m][j];

if (how == 1)

{

printf("Pop %c,push E->TE' by reversing!\n",ch[l]);

n = 3;

l = l + n - 1;

ch[l] = 'T';

ch[l - 1] = 39;

ch[l - 2] = 'E';

step = step + 1;

}

else if (how == 2)

{

printf("Pop %c%c,push E'->+TE' by reversing!\n",ch[l-1],ch[l]);

n = 4;

l = l + n - 2;

ch[l] = '+';

ch[l - 1] = 'T';

ch[l - 2] = 39;

ch[l - 3] = 'E';

step = step + 1;

}

else if (how == 3)

{

printf("Pop %c%c,using E'->ε!\n",ch[l-1],ch[l]);

l = l - 2;

step = step + 1;

}

else if (how == 4)

{

printf("Pop %c,push T->FT' by reversing!\n",ch[l]);

n = 3;

l = l + n - 1;

ch[l] = 'F';

ch[l - 1] = 39;

ch[l - 2] = 'T';

step = step + 1;

}

else if (how == 5)

{

printf("Pop %c%c,push T'->\*FT' by reversing!\n",ch[l-1],ch[l]);

n = 4;

l = l + n - 2;

ch[l] = '\*';

ch[l - 1] = 'F';

ch[l - 2] = 39;

ch[l - 3] = 'T';

step = step + 1;

}

else if (how == 6)

{

printf("Pop %c%c,using T'->ε!\n",ch[l-1],ch[l]);

l = l - 2;

step = step + 1;

}

else if (how == 7)

{

printf("Pop %c,push F->(E) by revesing!\n",ch[l]);

n = 3;

l = l + n - 1;

ch[l] = '(';

ch[l - 1] = 'E';

ch[l - 2] = ')';

step = step + 1;

}

else if (how == 8)

{

printf("Pop %c,push F->i by reversing!\n",ch[l]);

n = 1;

l = l + n - 1;

ch[l] = 'i';

step = step + 1;

}

else

{

printf("错误!\n");

break;

//exit(0);

}

}

else

{

if (cha == '#' && ch[l] == '#')

{

printf("成功!\n");

break;

}

else

{

printf("Match %c!\n",cha);

l = l - 1;

k = k + 1;

step = step + 1;

}

}

}

else

{

printf("错误的字符!");

break;

}

}

else //错误的字符

{

printf("错误的字符!");

break;

}

} while (l >= 0);

system("pause");

return 0;

}