编译原理与技术程序设计三

——LR 分析程序的设计与实现

实

验

报

告

班 级 2016211310

姓 名 张绍磊

学 号 2016211392

一、实验目的

让同学们更加深刻理解 LR 文法在自底向上分析程序中的具体应用。

二、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->E+T|E-T|T
T->T*F|T/F|F
F->id|(E)|num
```

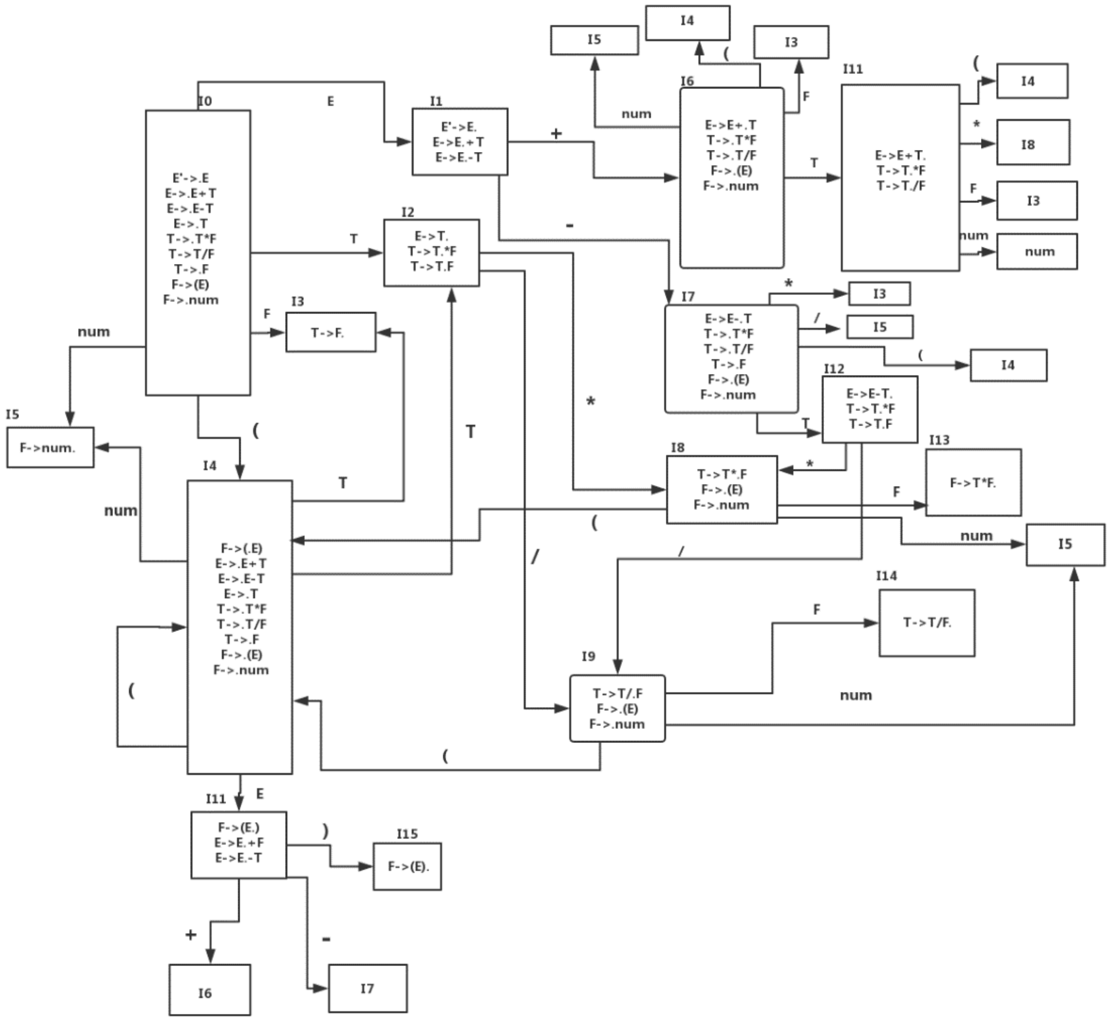
（1） 构造识别该文法所有活前缀的 DFA

（2） 构造该文法的 SLR 分析表

（3） 要求编程实现算法 4.3，构造 SLR 分析程序。

三、实验步骤

1. 该文法所有活前缀的 DFA



2. 构造该文法的 SLR 分析表

所有的产生式如下所示:

1：E->E+T

2：E->E-T

3：E->T

4：T->T*F

5：T->T/F

6：T->F

7：F->(E)

8：F->num

	action								goto		
状态	(	)	+	-	*	/	num	$	E	T	F
0	S4						S5		1	2	3
1			S6	S7				ACC			
2		R3	R3	R3	S8	S9		R3			
3		R6	R6	R6	R6	R6		R6			
4	S4						S5		10	2	3
5		R8	R8	R8	R8	R8		R8			
6	S4						S5			11	3
7	S4						S5			12	3
8	S4						S5				13
9	S4						S5				14
10		S15	S6	S7							
11		R1	R1	R1	S8	S9		R1			
12		R2	R2	R2	S8	S9		R2			
13		R4	R4	R4	R4	R4		R4			
14		R5	R5	R5	R5	R5		R5			
15		R7	R7	R7	R7	R7		R7			

3. SLR 分析程序

（1）生成式的数据结构

```
struct createRule

{
    string rule;//产生式本身
    string leftsymbol;//生成式的左侧符号
    int length;//产生式右部的字符串的长度
};
typedef createRule mycreateRule;
```

(2)符号表

```
        string endsymbol[8] = { "(",")","+","-","*","/","num","$" };//终结符符号表
```

(3)预测分析表

```
      string actionAnalyseMap[16][11] =
      {
        { "S4","error","error","error","error","error","S5","error","1","2","3" },
        { "error","error","S6","S7","error","error","error","ACC","error","error","error" },
        { "error","R3","R3","R3","S8","S9","error","R3","error","error","error" },
        { "error","R6","R6","R6","R6","R6","error","R6","error","error","error" },
        { "S4","error","error","error","error","error","S5","error","10","2","3" },
        { "error","R8","R8","R8","R8","R8","error","R8","error","error","error" },
        { "S4","error","error","error","error","error","S5","error","error","11","3"
      },
        { "S4","error","error","error","error","error","S5","error","error","12","3"
      },
        { "S4","error","error","error","error","error","S5","error","error","error","13" },
        { "S4","error","error","error","error","error","S5","error","error","error","14" },
        { "error","S15","S6","S7","error","error","error","error","error","error","error" },
        { "error","R1","R1","R1","S8","S9","error","R1","error","error","error",},
        { "error","R2","R2","R2","S8","S9","error","R2","error","error","error" },
        { "error","R4","R4","R4","R4","R4","error","R4","error","error","error" },
        { "error","R5","R5","R5","R5","R5","error","R5","error","error","error" },
        { "error","R7","R7","R7","R7","R7","error","R7","error","error","error" },
      };
```

（4）数据成员

序号	数据成员	说明
01	vector<string> stateStack	状态栈
02	vector<string> symbolStack	符号栈
03	string input	输入分析串

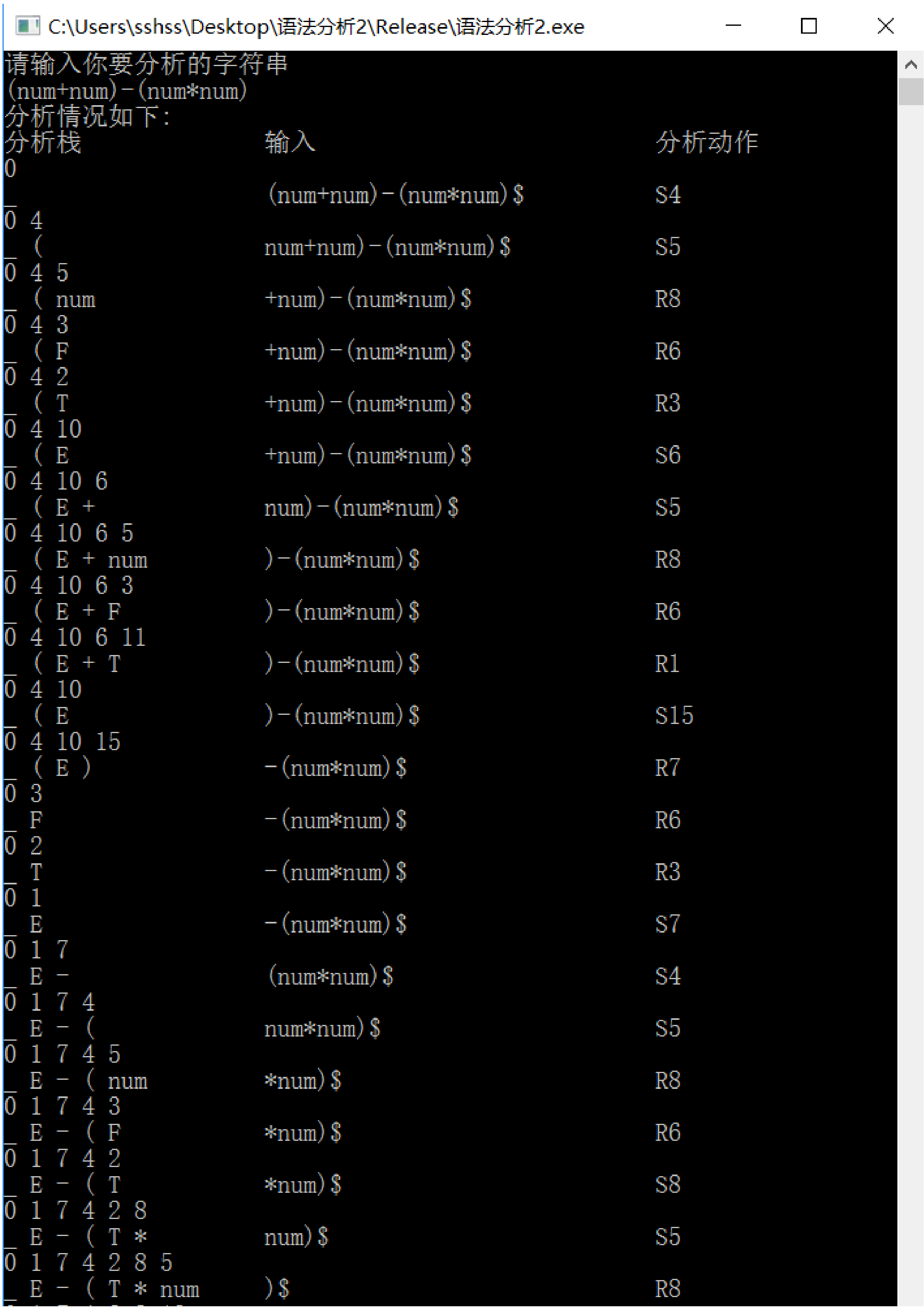
（5）子函数模块设计

序号	子函数名称	说明

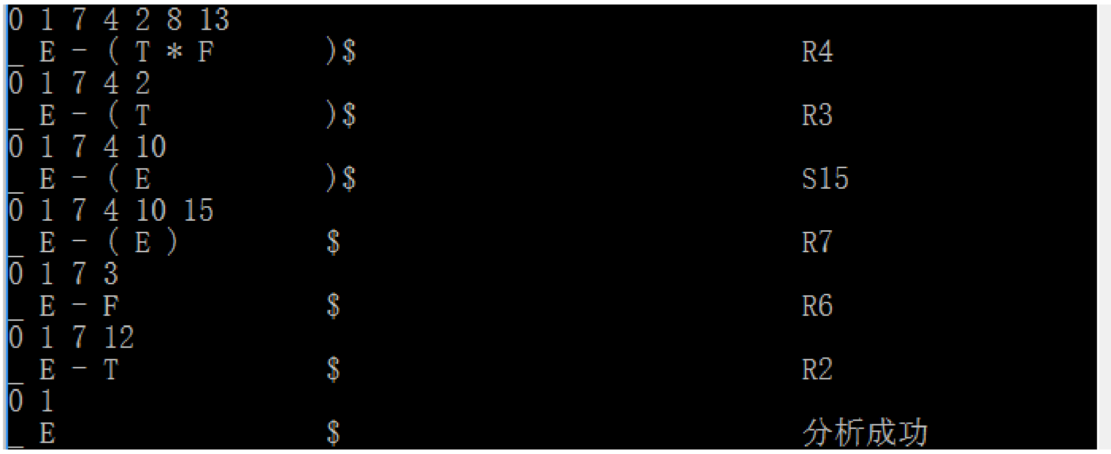
01	int findendsymbol(string a)	找出终结符在符号表的位置
02	void error()	错误分析

四、实验结果

1. 样例 1：（num+num）-(num*num)



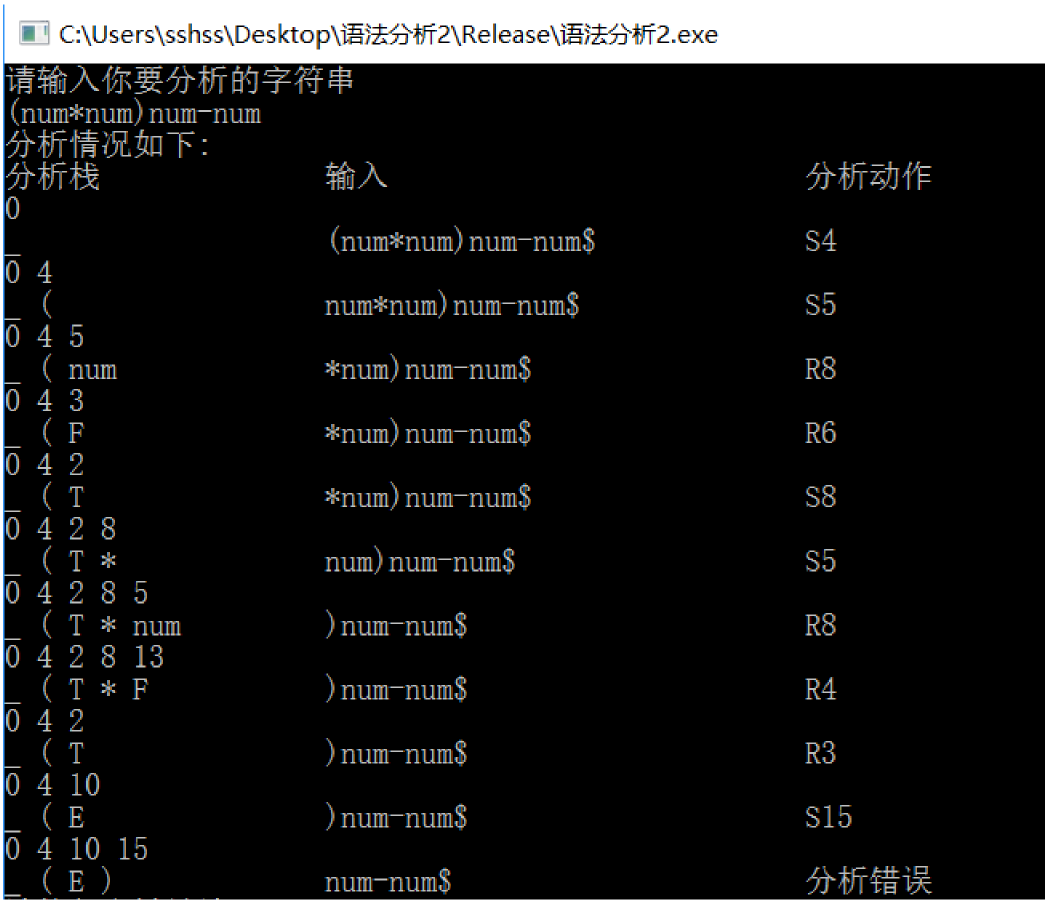
```
请输入你要分析的字符串
(num+num)-(num*num)
分析情况如下:
分析栈                  输入                          分析动作
0
_                       (num+num)-(num*num)$          S4
0 4
_ (                     num+num)-(num*num)$           S5
0 4 5
_ ( num                 +num)-(num*num)$              R8
0 4 3
_ ( F                   +num)-(num*num)$              R6
0 4 2
_ ( T                   +num)-(num*num)$              R3
0 4 10
_ ( E                   +num)-(num*num)$              S6
0 4 10 6
_ ( E +                 num)-(num*num)$               S5
0 4 10 6 5
_ ( E + num             )-(num*num)$                  R8
0 4 10 6 3
_ ( E + F               )-(num*num)$                  R6
0 4 10 6 11
_ ( E + T               )-(num*num)$                  R1
0 4 10
_ ( E                   )-(num*num)$                  S15
0 4 10 15
_ ( E )                 -(num*num)$                   R7
0 3
_ F                     -(num*num)$                   R6
0 2
_ T                     -(num*num)$                   R3
0 1
_ E                     -(num*num)$                   S7
0 1 7
_ E -                   (num*num)$                    S4
0 1 7 4
_ E - (                 num*num)$                     S5
0 1 7 4 5
_ E - ( num             *num)$                        R8
0 1 7 4 3
_ E - ( F               *num)$                        R6
0 1 7 4 2
_ E - ( T               *num)$                        S8
0 1 7 4 2 8
_ E - ( T *             num)$                         S5
0 1 7 4 2 8 5
_ E - ( T * num         )$                            R8
```



```
0 1 7 4 2 8 13
_ E - ( T * F       )$                              R4
0 1 7 4 2
_ E - ( T           )$                              R3
0 1 7 4 10
_ E - ( E           )$                              S15
0 1 7 4 10 15
_ E - ( E )         $                               R7
0 1 7 3
_ E - F             $                               R6
0 1 7 12
_ E - T             $                               R2
0 1
_ E                 $                               分析成功
```

分析成功，该句子属于该 LR 文法。

2. 样例 2：(num*num)num-num



C:\Users\sshss\Desktop\语法分析2\Release\语法分析2.exe

```
请输入你要分析的字符串
(num*num)num-num
分析情况如下:
分析栈              输入                          分析动作
0
_                   (num*num)num-num$             S4
0 4
_ (                 num*num)num-num$              S5
0 4 5
_ ( num             *num)num-num$                 R8
0 4 3
_ ( F               *num)num-num$                 R6
0 4 2
_ ( T               *num)num-num$                 S8
0 4 2 8
_ ( T *             num)num-num$                  S5
0 4 2 8 5
_ ( T * num         )num-num$                     R8
0 4 2 8 13
_ ( T * F           )num-num$                     R4
0 4 2
_ ( T               )num-num$                     R3
0 4 10
_ ( E               )num-num$                     S15
0 4 10 15
_ ( E )             num-num$                      分析错误
```

出错，无状态 15 遇到 num 时的分析动作。

五、总结与体会

通过本次对语法分析程序的设计和编写，自己获得了很大的收获，语法分析程序的功能有了更进一步认识，也更加理解了 LR 分析方法的精髓，加深了对课本知识的了解与应用。

虽然在程序的设计和编写过程中出现了一些错误，但是经过同学的帮助和指导，顺利地将程序中存在的错误顺利解决，从而顺利完成了本程序的设计和编写，获益匪浅。

六、实验源代码

```
#include<iostream>
#include<string>
#include<vector>
#include <iomanip>
using namespace std;

//产生式结构
struct createRule
{
    string rule;//产生式本身
    string leftsymbol;//生成式的左侧符号
    int length;//产生式右部的字符串的长度
};
typedef createRule mycreateRule;

//生成式的数组
mycreateRule nowCreateRule[8] =
{
    { { "E->E+T" },{ "E" },{ 3 } },
    { { "E->E-T" },{ "E" },{ 3 } },
    { { "E->T"   },{ "E" },{ 1 } },
    { { "T->T*F" },{ "T" },{ 3 } },
    { { "T->T/F" },{ "T" },{ 3 } },
    { { "T->F"   },{ "T" },{ 1 } },
    { { "F->(E)" },{ "F" },{ 3 } },
    { { "F->num" },{ "F" },{ 1 } },
};
string endsymbol[8] = { "(",")","+","-","*","/","num","$" };//终结符符号表

//分析表
string actionAnalyseMap[16][11] =
{
    { "S4","error","error","error","error","error","S5","error","1","2","3" },
    { "error","error","S6","S7","error","error","error","ACC","error","error","error" },
    { "error","R3","R3","R3","S8","S9","error","R3","error","error","error" },
    { "error","R6","R6","R6","R6","R6","error","R6","error","error","error" },
    { "S4","error","error","error","error","error","S5","error","10","2","3" },
    { "error","R8","R8","R8","R8","R8","error","R8","error","error","error" },
    { "S4","error","error","error","error","error","S5","error","error","11","3" },
    { "S4","error","error","error","error","error","S5","error","error","12","3" },
    { "S4","error","error","error","error","error","S5","error","error","error","13" },
    { "S4","error","error","error","error","error","S5","error","error","error","14" },
    { "error","S15","S6","S7","error","error","error","error","error","error","error" },
    { "error","R1","R1","R1","S8","S9","error","R1","error","error","error", },
    { "error","R2","R2","R2","S8","S9","error","R2","error","error","error" },
    { "error","R4","R4","R4","R4","R4","error","R4","error","error","error" },
    { "error","R5","R5","R5","R5","R5","error","R5","error","error","error" },
    { "error","R7","R7","R7","R7","R7","error","R7","error","error","error" },
};

```

```
//错误处理程序
void error()
{
    cout << "分析错误" << endl;
}

//找出终结符号在哪个位置
int findendsymbol(string a)
{
    int i;
    for (i = 0; i < 8; i++)
    {
        if (a == endsymbol[i])
        {
            return i;
        }
    }
    return -1;
}


void main()
{
    vector<string> stateStack;//状态栈
    vector<string> symbolStack;//符号栈
    stateStack.push_back("0");//状态栈的初始状态
    symbolStack.push_back("_");//符号栈的初始状态
    string input;//分析串
    cout << "请输入你要分析的字符串" << endl;
    cin >> input;
    input += "$";
    int ip = 0;
    string a;
    string s;
    cout << "分析情况如下:" << endl;
    cout.setf(ios::left);//设置左对齐
    cout << setw(20) << "分析栈" << setw(30) << "输入" << setw(20) << "分析动作" << endl;
    do
    {
        string string_content = "";
        for (int k = 0; k < stateStack.size(); k++)//输出状态栈内容
            string_content += stateStack[k]+" ";
        cout << setw(20) << string_content << endl;

        string_content = "";
        for (int k = 0; k < symbolStack.size(); k++)//输出符号栈内容
            string_content += symbolStack[k]+" ";
        cout << setw(20) << string_content;
        cout << setw(30) << input.substr(ip);//取子串输出分析符号
        s = stateStack.back();
        a = input.at(ip);
        if (a == "n")//处理“num”
        {
            a = "num";
            ip = ip + 2;
        }
        int index=findendsymbol(a);
        if (actionAnalyseMap[stoi(stateStack.back())][index].at(0) == 'S')//遇到分析动作是移进
        {

            string i =actionAnalyseMap[stoi(stateStack.back())][index].substr(1) ;
            cout << setw(20) << actionAnalyseMap[stoi(stateStack.back())][index] << endl;
            symbolStack.push_back(a);
            stateStack.push_back(i);
            ip = ip + 1;
        }
        else if (actionAnalyseMap[stoi(stateStack.back())][index].at(0) == 'R')//遇到分析动作是规约
        {
            int i = stoi(actionAnalyseMap[stoi(stateStack.back())][index].substr(1));
            cout << setw(20) << actionAnalyseMap[stoi(stateStack.back())][index] << endl;
            for (int j = 0; j < nowCreateRule[i - 1].length; j++)//出栈当前生成式左边符号长度个符号
            {
                symbolStack.pop_back();
                stateStack.pop_back();
            }
            string s1;
            s1 = stateStack.back();

            symbolStack.push_back(nowCreateRule[i - 1].leftsymbol);

            int s2;
            if (nowCreateRule[i - 1].leftsymbol == "E")//遇到“E”时转移的状态
            {
                s2 = 8;
            }
            else if (nowCreateRule[i - 1].leftsymbol == "T")//遇到“T”时转移的状态
            {
                s2 = 9;
            }
            else if (nowCreateRule[i - 1].leftsymbol == "F")//遇到“F”时转移的状态
            {
                s2 = 10;
            }

            if (actionAnalyseMap[stoi(s1)][s2] != "error")//若不是错误，则进栈
            {
                stateStack.push_back(actionAnalyseMap[stoi(s1)][s2]);

            }
            else
            {
                error();
                break;
            }
        }
        else if (actionAnalyseMap[stoi(stateStack.back())][index] == "ACC")
        {
            cout << "分析成功" << endl;
            break;
        }
        else
        {
            error();
            break;
        }

    } while (1);
    system("pause");
}
```