# 程序设计 2

**题目：**语法分析程序的设计与实现。

**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。要求所分析算术表达式由如下的文法产生。

E->E+T|E-T|T

T->T*F|T/F|F

F->(E)|num

**实现要求：**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方法 3：编写语法分析程序实现自底向上的分析，要求如下。

（1） 构造识别该文法所有活前缀的 DFA

（2） 构造该文法的 LR 分析表。

（3） 编程实现算法 4.3，构造 LR 分析程序。

**实验目的：**根据源语言的语法规则从源程序记号序列中识别出各种语法成分，同时进行语法检查，为语义分析和代码生成做准备。

**输入：**文法的产生式以及输入串

**输出：**验证输入串是不是符合该语言语法规则的一个程序，若是，则输出其分析树，若不是，则表明输入串中存在语法错误，需要报告错误的性质和位置。

**设计过程：**

1. 通过 LR 分析方法进行分析，其主要由以下几个部分构成：

（1） 输入缓冲区：存放待分析的输入符号串，并以$作为符号串的结束标志。

（2） 输出：是 LR 分析控制程序分析输入符号串的过程中所采用的动作序列。

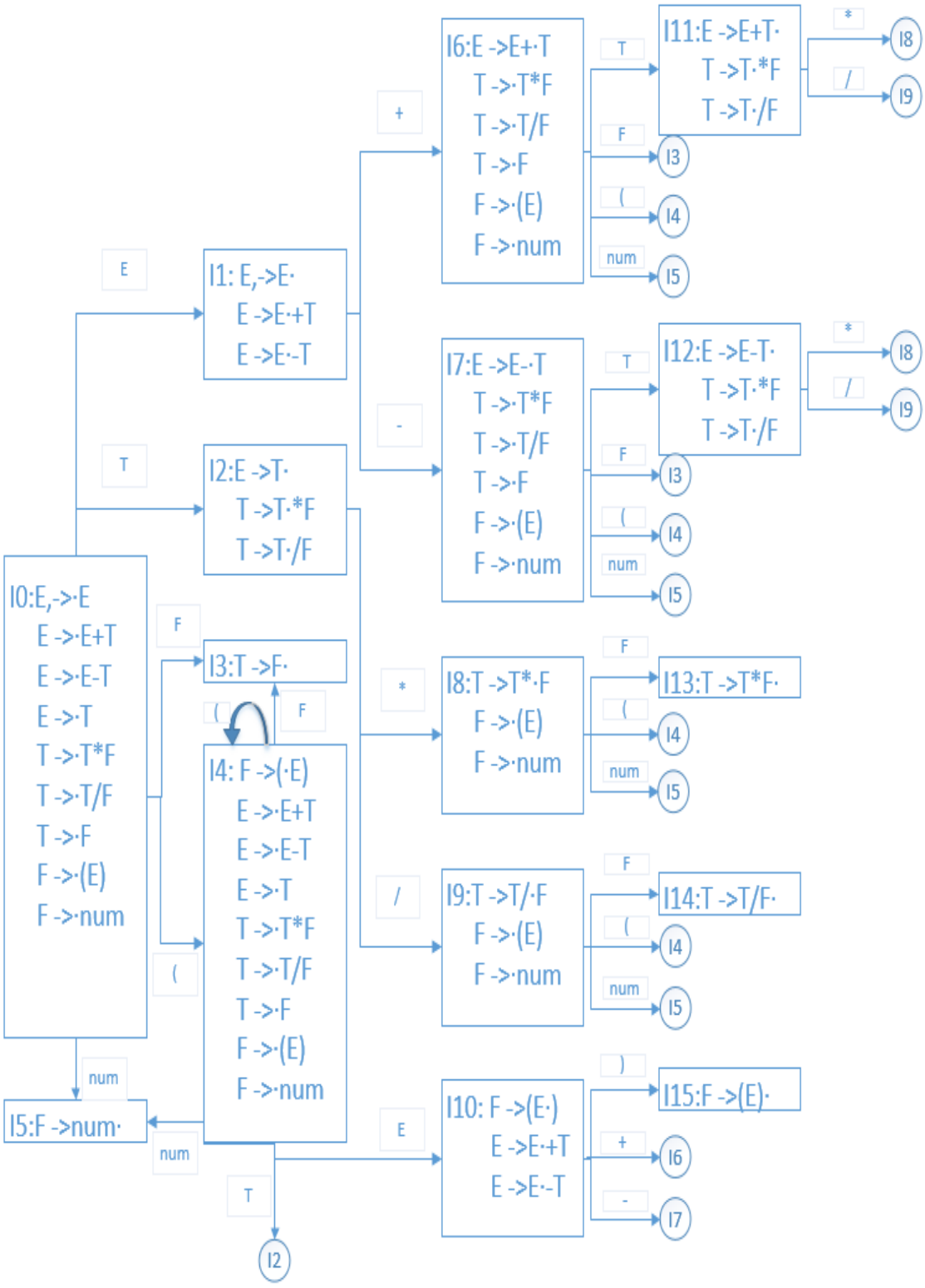
（3） 栈：由状态栈和符号栈构成（State 和 Symbol）

（4） 分析表：分析表是一个确定有限自动机的状态转移表，该自动机的字母表即由全部文法符号构成的集合。其中终结符号及$对应的列构成动作（Action）表，非终结符好对应的列构成状态转移（goto）表

Goto[Sm,A]保存了当前状态 Sm 相对于非终结符号 A 的后继状态。

Action[Sm,ai]规定了当前状态 Sm 面临输入符号 ai 时应采取的分析动作，主要有移进，归约，接收，出错几个动作。

2. 构造识别该文法的所有活前缀的 DFA：


I0:E,->·E
E ->·E+T
E ->·E-T
E ->·T
T ->·T*F
T ->·T/F
T ->·F
F ->·(E)
F ->·num
I1: E,->E·
E ->E·+T
E ->E·-T
I2:E ->T·
T ->T·*F
T ->T·/F
I3:T ->F·
I4: F ->(·E)
E ->·E+T
E ->·E-T
E ->·T
T ->·T*F
T ->·T/F
T ->·F
F ->·(E)
F ->·num
I5:F ->num·
I6:E ->E+·T
T ->·T*F
T ->·T/F
T ->·F
F ->·(E)
F ->·num
I7:E ->E-·T
T ->·T*F
T ->·T/F
T ->·F
F ->·(E)
F ->·num
I8:T ->T*·F
F ->·(E)
F ->·num
I9:T ->T/·F
F ->·(E)
F ->·num
I10: F ->(E·)
E ->E·+T
E ->E·-T
I11:E ->E+T·
T ->T·*F
T ->T·/F
I12:E ->E-T·
T ->T·*F
T ->T·/F
I13:T ->T*F·
I14:T ->T/F·
I15:F ->(E)·
E
T
F
(
num
+
-
*
/
)
I2
I3
I4
I5
I6
I7
I8
I9

3. 构造 LR 分析表：

| 状态 | Action | | | | | | | | goto | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | + | - | * | / | ( | ) | num | $ | E | T | F |
| 0 | e1 | e1 | e1 | e1 | S4 | e2 | S5 | e1 | 1 | 2 | 3 |
| 1 | S6 | S7 | e5 | e5 | e3 | e2 | e3 | ACC | | | |
| 2 | R3 | R3 | S8 | S9 | e3 | R3 | e3 | R3 | | | |
| 3 | R6 | R6 | R6 | R6 | e6 | R6 | e6 | R6 | | | |
| 4 | e1 | e1 | e1 | e1 | S4 | e2 | S5 | e1 | 10 | 2 | 3 |
| 5 | R8 | R8 | R8 | R8 | e6 | R8 | e6 | R8 | | | |
| 6 | e1 | e1 | e1 | e1 | S4 | e2 | S5 | e1 | | 11 | 3 |
| 7 | e1 | e1 | e1 | e1 | S4 | e2 | S5 | e1 | | 12 | 3 |
| 8 | e1 | e1 | e1 | e1 | S4 | e2 | S5 | e1 | | | 13 |
| 9 | e1 | e1 | e1 | e1 | S4 | e2 | S5 | e1 | | | 14 |
| 10 | S6 | S7 | e5 | e5 | e3 | S15 | e3 | e4 | | | |
| 11 | R1 | R1 | S8 | S9 | e3 | R1 | e3 | R1 | | | |
| 12 | R2 | R2 | S8 | S9 | e3 | R2 | e3 | R2 | | | |
| 13 | R4 | R4 | R4 | R4 | e6 | R4 | e6 | R4 | | | |
| 14 | R5 | R5 | R5 | R5 | e6 | R5 | e6 | R5 | | | |
| 15 | R7 | R7 | R7 | R7 | e6 | R7 | e6 | R7 | | | |

4. LR 分析程序源代码：

```
#include<iostream>
#include<vector>
#include<sstream>
#include<stack>
#include<string>
using namespace std;
vector<char> VN;  //非终结符表
vector<string> VT;   //终结符表
vector<string> P; //产生式表
stack<int> State; //状态栈
stack<string> Symbol;//符号栈
//Action 表
string Action[16][8] = {
    { "e1","e1","e1","e1","S4","e2","S5","e1" },
    { "S6","S7","e5","e5","e3","e2","e3","ACC" },
    { "R3","R3","S8","S9","e3","R3","e3","R3" },
    { "R6","R6","R6","R6","e6","R6","e6","R6" },
    { "e1","e1","e1","e1","S4","e2","S5","e1" },
    { "R8","R8","R8","R8","e6","R8","e6","R8" },
    { "e1","e1","e1","e1","S4","e2","S5","e1" },
    { "e1","e1","e1","e1","S4","e2","S5","e1" },
    { "e1","e1","e1","e1","S4","e2","S5","e1" },
    { "e1","e1","e1","e1","S4","e2","S5","e1" },
    { "S6","S7","e5","e5","e3","S15","e3","e4" },
    { "R1","R1","S8","S9","e3","R1","e3","R1" },
    { "R2","R2","S8","S9","e3","R2","e3","R2" },
```

```
    { "R4","R4","R4","R4","e6","R4","e6","R4" },
    { "R5","R5","R5","R5","e6","R5","e6","R5" },
    { "R7","R7","R7","R7","e6","R7","e6","R7" }
};
//goto 表
int Goto[16][3] = {
    { 1,2,3 },
    { -1,-1,-1 },
    { -1,-1,-1 },
    { -1,-1,-1 },
    { 10,2,3 },
    { -1,-1,-1 },
    { -1,11,3 },
    { -1,12,3 },
    { -1,-1,13 },
    { -1,-1,14 },
    { -1,-1,-1 },
    { -1,-1,-1 },
    { -1,-1,-1 },
    { -1,-1,-1 },
    { -1,-1,-1 },
    { -1,-1,-1 },
};
//字符类型转换为字符串类型
string charToString(char c) {
    string s;
    stringstream stream;
    stream << c;
    s = stream.str();
    return s;
}
//定位非终结符
int findVN(vector<char> _VN, char c) {
    vector<char>::iterator it;
    for ( it = _VN.begin(); it != _VN.end(); it++)
    {
        if ((*it)==c)
        {
            return it - _VN.begin();
        }
    }
    return -1;
}
//定位终结符
```

```
int findVT(vector<string> _VT, string s) {
    vector<string>::iterator it;
    for ( it = _VT.begin(); it != _VT.end(); it++)
    {
        if ((*it)==s)
        {
            return it - _VT.begin();
        }
    }
    return -1;
}
//加入终结符表
void addToVT(string s) {
    if (s.length() > 0) {
        if (s=="num")
        {
            if (findVT(VT, s) < 0) {
                VT.push_back(s);
            }
        }
        else
        {
            for (int j = 0; j < s.length(); j++)
            {
                if (findVT(VT,charToString(s[j]))<0)
                {
                    VT.push_back(charToString(s[j]));
                }
            }
        }
    }
}
//输入产生式
void inputP() {
    string s;
    string temp;//临时字符串变量
    string vt;//终结符字符串
    cout << "输入产生式:(键入 ok 表示输入结束) " << endl;
    getline(cin, s);
    while (s!="ok")
    {
        temp.clear();
        vt.clear();
        if (s.length()>1)
```

```
{
    for (int i = 0; i < s.length(); i++) {
        if (s[i]>='A'&&s[i]<='Z')
        {
            if (findVN(VN, s[i]) < 0) {
                VN.push_back(s[i]);
            }
            if (i!=0)
            {
                temp += s[i];
                addToVT(vt);
                vt.clear();
            }
        }
        else if ((s[i]=='-
'&&s[i+1]=='>')||(s[i]=='>'&&s[i-1]=='-'))
        {
            continue;
        }
        else if (s[i]=='|')
        {
            addToVT(vt);
            vt.clear();
            string j = "->";
            P.push_back(s[0] + j + temp);
            temp.clear();
        }
        else
        {
            vt += s[i];
            temp += s[i];
        }
        if (i==s.length()-1)
        {
            addToVT(vt);
            vt.clear();
            string j = "->";
            P.push_back(s[0] + j + temp);
            temp.clear();
        }
    }
}
cout << "输入产生式:(键入 ok 表示输入结束) " << endl;
getline(cin, s);
```

```
    }
    vector<char>::iterator it_n;    //输出非终结符迭代器
    vector<string>::iterator it;  //输出终结符和产生式迭代器
    cout << "VN:";
    for ( it_n = VN.begin(); it_n != VN.end(); it_n++)
    {
        cout << *(it_n) << " ";
    }
    cout << endl;
    cout << "VT:";
    for ( it = VT.begin(); it !=VT.end() ; it++)
    {
        cout << *it << " ";
    }
    cout << endl;
    for ( it = P.begin(); it != P.end(); it++)
    {
        cout << *it << endl;
    }
    cout << endl;
}
//输出栈中内容(flag 为 0 输出状态栈, 1 输出符号栈)
void output(int flag) {
    //利用两个栈反转输出栈中的内容
    stack<int> _state;
    stack<string> _symbol;
    int t_state;
    string t_symbol;
    if (flag==0)
    {
        while (!State.empty())
        {
            t_state = State.top();
            State.pop();
            _state.push(t_state);
        }
        while (!_state.empty())
        {
            t_state = _state.top();
            cout << t_state << " ";
            _state.pop();
            State.push(t_state);
        }
    }
```

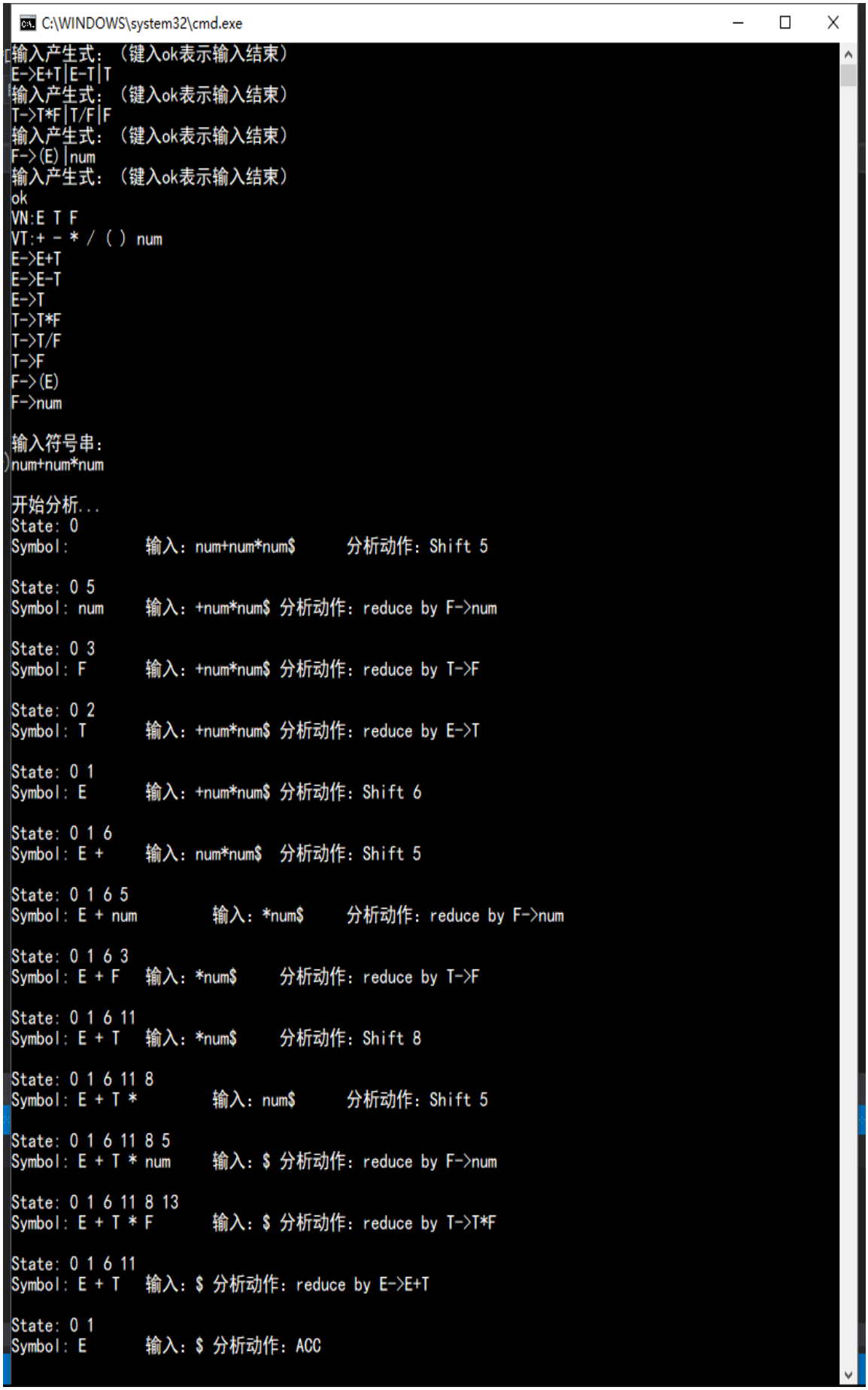
```
    else
    {
        while (!Symbol.empty())
        {
            t_symbol = Symbol.top();
            Symbol.pop();
            _symbol.push(t_symbol);
        }
        while (!_symbol.empty())
        {
            t_symbol = _symbol.top();
            cout << t_symbol << " ";
            _symbol.pop();
            Symbol.push(t_symbol);
        }
    }
}
void function() {
    string input;
    cout << "输入符号串: " << endl;
    getline(cin, input);
    cout << endl;
    input += "$";
    State.push(0);//初始化状态栈
    int i = 0;
    int j;
    int S;       //S 为状态栈栈顶元素
    string temp;
    VT.push_back("$");
    cout << "开始分析..." << endl;
    while (true)
    {
        S = State.top();
        j = i;
        while (findVT(VT,temp)<0)
        {
            temp += input[j];
            j++;
        }
        cout << "State: ";
        output(0);
        cout << endl;
        cout << "Symbol: ";
        output(1);
```

```
            cout << '\t' << "输入：" << input.substr(i) << '\t' <<
"分析动作：";
            //移进
            if (Action[S][findVT(VT,temp)][0]=='S')
            {
                State.push(Action[S][findVT(VT, temp)][1]-48);
                Symbol.push(temp);//移进符号
                cout << "Shift " << Action[S][findVT(VT, temp)][1]
<< endl << endl;
                temp.clear();
                i = j;//指向输入串的下一个符号
            }
            //归约
            else if (Action[S][findVT(VT,temp)][0]=='R')
            {
                int r = Action[S][findVT(VT, temp)][1] - 48;//取得
归约产生式的下标
                int p_length = 0;                        //用于存取产
生式箭头后的长度
                string str;
                str = P[r - 1].substr(3);                  //从箭头
之后开始计数
                int c = 0;
                string t1;                                //计算 A->β
中 |β|长度
                while (c < str.length())
                {
                    t1 += str[c];
                    if (findVN(VN,str[c])>=0||findVT(VT,t1)>=0)
                    {
                        p_length++;
                        t1.clear();
                    }
                    c++;
                }
                t1.clear();
                for (int i = 0; i < p_length; i++)        //从栈顶
弹出|β|（即这里的 p_length）个符号
                {
                    State.pop();
                    Symbol.pop();
                }
                int n_S = State.top();          //n_S 为新的栈顶
                string A = P[r - 1].substr(0, 1);//归约后的 A 压入
```

```
Symbol 栈中
                Symbol.push(A);
                State.push(Goto[n_S][findVN(VN, A[0])]);//定位 A 在
非终结符表中的位置进而查 GOTO 表将下一状态压栈
                cout << "reduce by " << P[r - 1] << endl << endl;
            }
            //接受
            else if (Action[S][findVT(VT,temp)][0]=='A')
            {
                cout << "ACC" << endl;
                break;
            }
            //错误处理
            else if (Action[S][findVT(VT, temp)][0] == 'e')
            {
                cout << "error: " << endl;
                if (Action[S][findVT(VT, temp)][1]=='1')
                {//状态 0 6 7 8 9 期待输入符号为运算对象的首字符却输
入+、-、*、/和$
                 //缺少运算对象，把假想的 num 压入栈，并转移到状态 3
                    Symbol.push("num");
                    State.push(5);
                    cout << "缺少运算对象，将 num 压入栈" << endl;
                }
                else if (Action[S][findVT(VT, temp)][1] == '2')
                {//状态 0 1 4 6 7 8 9 期待输入符号为运算符号却输入)
                 //括号不匹配，直接略过
                    i++;
                    cout << "括号不匹配，指向下一个符号" << endl;
                }
                else if (Action[S][findVT(VT, temp)][1] == '3')
                {//状态 1 2 10 11 12 期待输入符号为运算符号或)却输入
（或者 num
                 //缺少运算符，将假想的"+"压入栈，并转移到状态 6
                    Symbol.push("+");
                    State.push(6);
                    cout << "缺少运算符号，将+压入栈" << endl;
                }
                else if (Action[S][findVT(VT, temp)][1] == '4')
                {//状态 10 期待输入符号为运算符号或)却输入$
                 //缺少右括号，将假想的")"压入栈，并转移到状态 15
                    Symbol.push(")");
                    State.push(15);
                    cout << "缺少有括号，将）压入栈" << endl;
```

```
                }
                else if (Action[S][findVT(VT, temp)][1] == '5')
                {//状态 1 10 期待输入+或-却输入*或/
                 //运算符号错误，将假想的"+"压入栈，并转移到状态 6
                    Symbol.push("+");
                    State.push(6);
                    cout << "缺少+/-，将+压入栈" << endl;
                }
                else
                {//归约产生式时输入符号为（或者 num,直接跳过相关符号
                    if (temp == "(")
                        i++;
                    else
                        i = i + 3;
                    cout << "归约产生多余符号，直接跳过" << endl;
                }
            }
            temp.clear();
        }
        cout << endl;
        cout << "...分析结束..." << endl;
    }
    void main() {
        inputP();
        function();
        system("pause");
        return;
}
```

**实验结果：**


C:\WINDOWS\system32\cmd.exe
输入产生式：（键入ok表示输入结束）
E->E+T|E-T|T
输入产生式：（键入ok表示输入结束）
T->T*F|T/F|F
输入产生式：（键入ok表示输入结束）
F->(E)|num
输入产生式：（键入ok表示输入结束）
ok
VN:E T F
VT:+ - * / ( ) num
E->E+T
E->E-T
E->T
T->T*F
T->T/F
T->F
F->(E)
F->num
输入符号串：
num+num*num
开始分析...
State: 0
Symbol:  输入：num+num*num$  分析动作：Shift 5
State: 0 5
Symbol: num  输入：+num*num$ 分析动作：reduce by F->num
State: 0 3
Symbol: F  输入：+num*num$ 分析动作：reduce by T->F
State: 0 2
Symbol: T  输入：+num*num$ 分析动作：reduce by E->T
State: 0 1
Symbol: E  输入：+num*num$ 分析动作：Shift 6
State: 0 1 6
Symbol: E +  输入：num*num$  分析动作：Shift 5
State: 0 1 6 5
Symbol: E + num  输入：*num$  分析动作：reduce by F->num
State: 0 1 6 3
Symbol: E + F  输入：*num$  分析动作：reduce by T->F
State: 0 1 6 11
Symbol: E + T  输入：*num$  分析动作：Shift 8
State: 0 1 6 11 8
Symbol: E + T *  输入：num$  分析动作：Shift 5
State: 0 1 6 11 8 5
Symbol: E + T * num  输入：$ 分析动作：reduce by F->num
State: 0 1 6 11 8 13
Symbol: E + T * F  输入：$ 分析动作：reduce by T->T*F
State: 0 1 6 11
Symbol: E + T  输入：$ 分析动作：reduce by E->E+T
State: 0 1
Symbol: E  输入：$ 分析动作：ACC