

# 编译器课设报告

## 自动化工具及词法语法分析程序

所属系	计算机系
专业	软件工程
学号	1102610118
姓名	闵大为
指导教师	王永利
起讫日期	2014.5 --- 2014.6
设计地点	南京理工大学

# 目录

1 实验目的 .....	2
2 实验内容及要求 .....	2
2.1 自动生成工具要求 .....	2
2.2 词法分析器设计要求 .....	2
2.3 语法分析器设计要求 .....	2
3 实验步骤 .....	3
3.1 设计词法自动生成工具 myLex .....	3
3.2 设计语法自动生成工具 myYacc .....	5
3.3 设计词法分析程序 .....	7
3.4 设计语法分析程序 .....	8
4 实验结果 .....	10
4.1 词法自动生成工具 myLex 测试 .....	10
4.2 语法自动生成工具 myYacc 测试 .....	12
4.3 词法分析程序测试 .....	14
4.4 语法分析程序测试 .....	15
5 实验总结 .....	17
5.1 完成的工作 .....	17
5.2 收获与体会 .....	17
6 附录 .....	17
附录 1: 程序清单 .....	17
附录 2: 程序源代码——G_to_NFA.cpp .....	18
附录 3: 程序源代码——NFA_to_DFA.cpp .....	20
附录 4: 程序源代码——Grammar.cpp .....	35
附录 5: 程序源代码——Min_DFA.cpp .....	41
附录 6: 程序源代码——myYacc.cpp .....	45
附录 7: 程序源代码——Cifa.cpp .....	72
附录 8: 程序源代码——yufa.cpp .....	85

# 1 实验目的

1. 了解词法分析过程的基本思想。
2. 体会词法分析器及其自动化生成工具的开发过程。
3. 了解语法分析过程的基本思想。
4. 体会语法分析器及其自动化生成工具的开发过程。

# 2 实验内容及要求

## 2.1 自动生成工具要求

1. 能够根据文法自动生成词法分析程序
2. 能够根据文法自动生成语法分析程序

## 2.2 词法分析器设计要求

1. 输入正规文法或者正规式，经过 NFA 到 DFA 的转换，DFA 最小化，递归下降法生成程序等步骤，输出能识别文法表示的句子词法分析程序。
2. 要求至少支持科学计数法常量和标识符识别。
3. 根据输入字符串输出 Token 串。如果是非法字符串则输出错误信息。

## 2.3 语法分析器设计要求

1. 要求使用 LL(1)方法，算符优先分析方法，LR(1)三种方法之一设计语法分析程序。
2. 输入上下文无关文法，输出能进行语法分析的程序。
3. 输入待检测的 Token 串，输出检测结果和出错信息。

## 3 实验步骤

### 3.1 设计词法自动生成工具 myLex

#### 1 实现原理

- 1 正规文法到 NFA p65 上
- 2 NFA 到 DFA 子集构造算法 p59 上
- 3 DFA 去掉多余状态 图的遍历 书 p60
- 4 DFA 最小化 子集划分算法 书 p61 上
- 5 输出程序 根据图的矩阵在代码中写代码

#### 2 输入输出说明

1 输入:

VN1 VT2 VN3

VN4 VT5 VN6

.....

第一个 VN 为开始符号

如  $S \rightarrow bA$  则为 S b A

如  $S \rightarrow a$  则为 S a NULL

如  $S \rightarrow \text{NULL}$  则为 S NULL NULL

最多为 26 个状态

2 输出:

点击 G\_to\_NFA.exe , NFA\_to\_DFA.exe , Min\_DFA.exe, 得到代码文件 test.cpp

#### 3 程序流程图

myLex 程序流程图如图 1 所示:

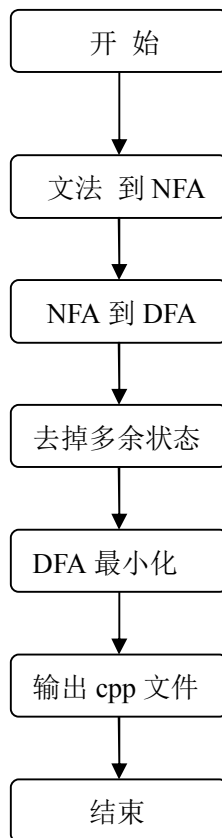


图 1 myLex 程序流程图

## 4 程序文件截图

myLex 程序文件截图如图 2 所示：










 G_to_NFA.exe	2014/5/28 14:59	应用程序
 in.txt	2014/6/4 15:33	文本文档
 Min_DFA.exe	2014/5/30 15:55	应用程序
 NFA_to_DFA.exe	2014/5/29 0:34	应用程序
 out1.txt	2014/6/4 15:33	文本文档
 out2.txt	2014/6/4 15:33	文本文档
 out3.txt	2014/6/4 15:33	文本文档
 test.cpp	2014/6/4 15:33	CPP 文件
 使用说明.txt	2014/6/4 15:29	文本文档

图 2 myLex 程序文件截图

## 3.2 设计语法自动生成工具 myYacc

### 1 实现原理

- 1 读入文法进行标记 1,2,3,4..... (如何记录每一项用字符串解决)
- 2 写 First 函数
- 3 写 First\_str() 入口参数是字符串
- 4 保存项目集, 并编写 CLOSURE(I)函数 书 p144
- 6 GO(I,X)函数, 检测集合是否出现过, 并生成图 (字符串比较)
- 7 设计分析表的数据结构 (矩阵, 负的为 ri, 正的为 Si, acc=140406 为接受态)
- 8 根据对应的图, 遍历节点, 填写相应的表(结合在 Go 函数中)
- 9 写分析的流程, 根据分析表分析, 返回是否正确
- 10 自动生成代码: 总结自己写的过程, 找出变得和不变的, 用代码实现

### 2 输入输出说明

1 输入:

g.txt 中输入拓广文法 (Z 代替 S', \$表示空)

例如  $S \rightarrow aABd$  的话记为 SaABd

ZS

BaB

SBB

Bb

2 输出:

点击 myYacc, 输出代码文件 test2.cpp

### 3 程序流程图

myYacc 程序流程图如图 3 所示:

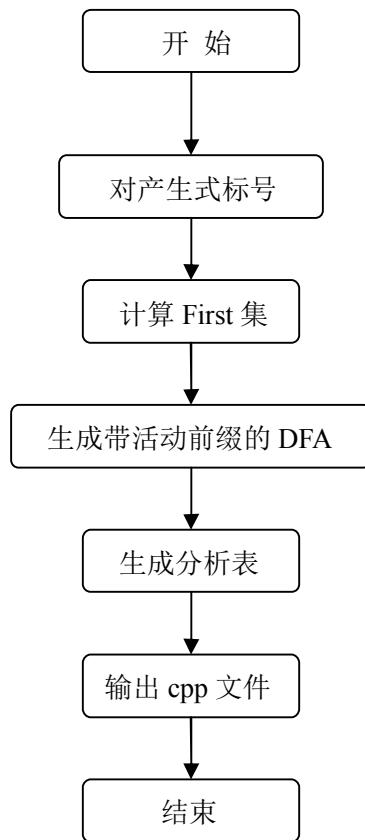


图 3 myYacc 程序流程图

#### 4 程序文件截图

myYacc 程序文件截图如图 4 所示：

	二义性文法	2014/6/8 15:03	文件夹
	非二义性文法	2014/6/8 15:04	文件夹
	g.txt	2014/6/3 16:51	文本文档
	myYacc.exe	2014/6/8 15:07	应用程序
	test2.cpp	2014/6/9 23:29	CPP 文件
	使用说明.txt	2014/6/4 15:48	文本文档

图 4 myYacc 程序文件截图

## 3.3 设计词法分析程序

### 1 实现原理

在词法分析中，利用空格，tab 键，换行，运算符进行分隔，获得字符串，对于简单的字符串可以在主程序中处理，对于复杂的字符串，可以将字符串用 myLex 自动生成的代码进行检测。

### 2 输入输出说明

1 输入：

字符序列 string 字符串

2 输出：

词法分析结果

输出格式：二元式序列（类型，值）

### 3 程序流程图

词法分析程序流程图如图 5 所示：

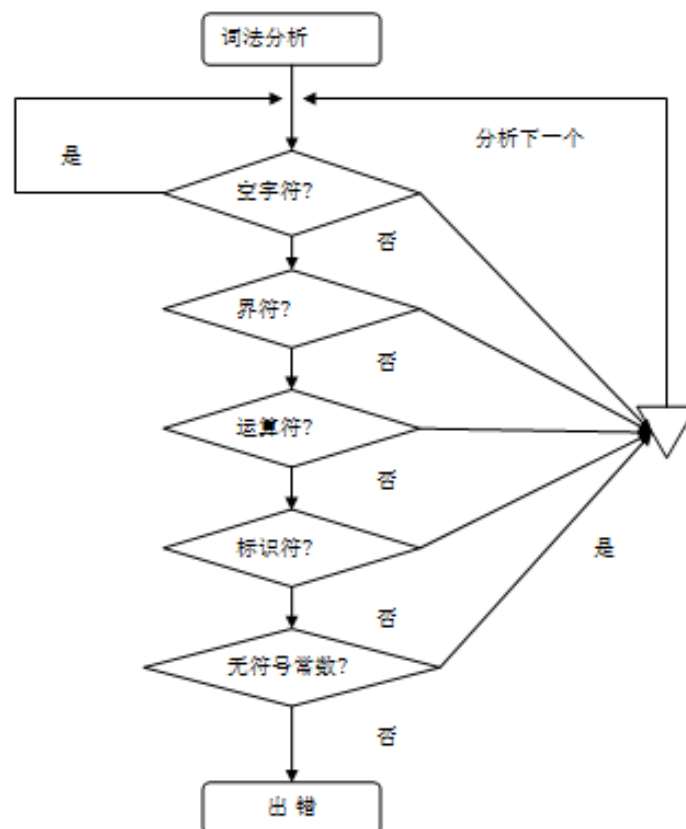
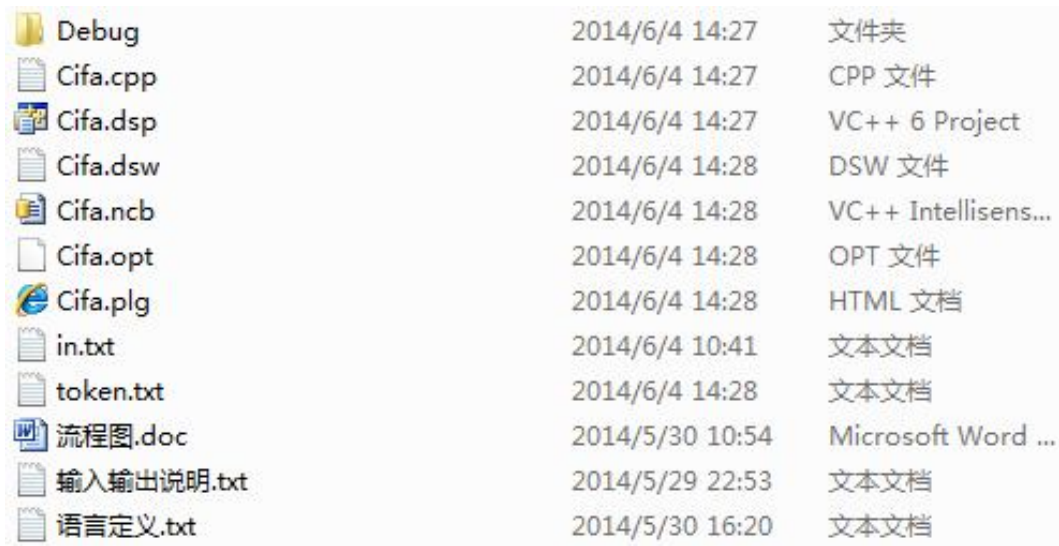


图 5 词法分析程序流程图



## 4 程序文件截图

词法分析程序文件截图如图 6 所示：



Debug	2014/6/4 14:27	文件夹
Cifa.cpp	2014/6/4 14:27	CPP 文件
Cifa.dsp	2014/6/4 14:27	VC++ 6 Project
Cifa.dsw	2014/6/4 14:28	DSW 文件
Cifa.ncb	2014/6/4 14:28	VC++ Intellisens...
Cifa.opt	2014/6/4 14:28	OPT 文件
Cifa.plg	2014/6/4 14:28	HTML 文档
in.txt	2014/6/4 10:41	文本文档
token.txt	2014/6/4 14:28	文本文档
流程图.doc	2014/5/30 10:54	Microsoft Word ...
输入输出说明.txt	2014/5/29 22:53	文本文档
语言定义.txt	2014/5/30 16:20	文本文档

图 6 词法分析程序文件截图

## 3.4 设计语法分析程序

### 1 实现原理

在语法分析中，利用空分号进行分隔，获得处理的 token 串，然后根据文法将每个 token 串换处成文法能处理的字符串，用 myYacc 自动生成的代码进行检测。

### 2 输入输出说明

1 输入：

token 串

2 输出：

yes 或 error

错误情况将输出 token 串中出错的位置。

### 3 程序流程图

语法分析程序流程图如图 7 所示：

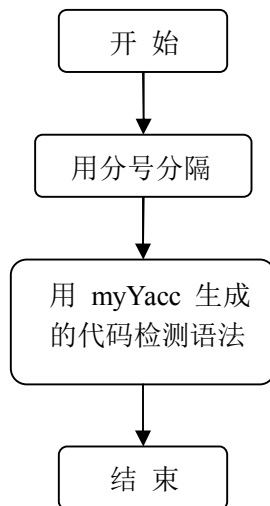


图 7 语法分析程序流程图

## 4 程序文件截图

语法分析程序文件截图如图 8 所示：

Debug	2014/6/8 15:19	文件夹
Cifa.exe	2014/6/4 14:27	应用程序
in.txt	2014/6/4 14:23	文本文档
test2.cpp	2014/6/8 15:12	CPP 文件
test2.dsp	2014/6/4 15:05	VC++ 6 Project
test2.dsw	2014/6/4 15:05	DSW 文件
test2.ncb	2014/6/4 15:05	VC++ Intellisens...
test2.opt	2014/6/4 15:05	OPT 文件
test2.plg	2014/6/4 15:05	HTML 文档
token.txt	2014/6/4 15:06	文本文档
yufa.cpp	2014/6/8 15:19	CPP 文件
yufa.dsp	2014/6/8 15:16	VC++ 6 Project
yufa.dsw	2014/6/8 15:36	DSW 文件
yufa.exe	2014/6/8 15:19	应用程序
yufa.ncb	2014/6/8 15:36	VC++ Intellisens...
yufa.opt	2014/6/8 15:36	OPT 文件
yufa.plg	2014/6/8 15:19	HTML 文档
测试用例.txt	2014/6/4 11:14	文本文档
输入输出说明.txt	2014/6/4 10:44	文本文档
语法定义.txt	2014/6/8 15:13	文本文档
语法分析实现步骤.txt	2014/5/31 16:44	文本文档

图 8 词法分析程序文件截图

## 4 实验结果

### 4.1 词法自动生成工具 myLex 测试

#### 1. 科学计数法

1) 输入文法:

$S \rightarrow dA$

$S \rightarrow .B$

$S \rightarrow eC$

$A \rightarrow dA$

$A \rightarrow .B$

$A \rightarrow eC$

$A \rightarrow \text{NULL NULL}$

$B \rightarrow dD$

$D \rightarrow eC$

$D \rightarrow dD$

$D \rightarrow \text{NULL NULL}$

$C \rightarrow dE$

$C \rightarrow sF$

$F \rightarrow dE$

$E \rightarrow dE$

$E \rightarrow \text{NULL NULL}$

2) 依次点击演示\1 自动生成工具\myLex\tool 中的 G\_to\_NFA.exe , NFA\_to\_DFA.exe , Min\_DFA.exe, 得到代码文件 test.cpp。将其中的 `str[index]=='d'` 替换成 `str[index]>='0'&&str[index]<='9'` , 将其中的 `str[index]=='s'` 替换成 `str[index]=='+'||str[index]=='-'` , 编译得到可执行程序。

3) 在 in.txt 中输入:

3.14e10

3.14e+10

3.14e-10

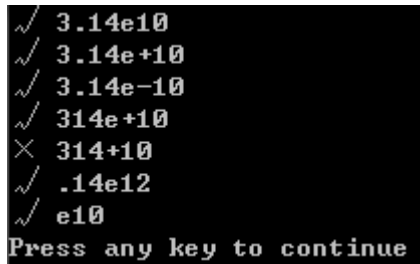
314e+10

314+10

.14e12

e10

4) 程序输出:



```
✓ 3.14e10
✓ 3.14e+10
✓ 3.14e-10
✓ 314e+10
✗ 314+10
✓ .14e12
✓ e10
Press any key to continue
```

## 2. 标识符

1) 输入文法:

S 1 NULL

S 1 A

A 1 NULL

A d NULL

A 1 A

A d A

2) 依次点击演示\1 自动生成工具\myLex\tool 中的 G\_to\_NFA.exe , NFA\_to\_DFA.exe , Min\_DFA.exe, 得到代码文件 test.cpp。将其中的 str[index]=='d' 替换成 str[index]>='0'&&str[index]<='9' , 将其中的 str[index]=='l' 替换成 str[index]>='a'&&str[index]<='z' , 编译得到可执行程序。

3) 在 in.txt 中输入:

abc

a12sd

123

1ab

as+

as2

4) 程序输出:

```

✓ abc
✓ a12sd
✗ 123
✗ 1ab
✗ as+
✓ as2
Press any key to continue

```

## 4.2 语法自动生成工具 myYacc 测试

### 1. 四则运算

1) 输入文法:

ZE

EE+T

ET

TT\*F

TF

F(E)

Fi

2) 点击演示\1 自动生成工具\myYacc\tool 中的 myYacc.exe , 得到代码文件 test2.cpp, 编译得到可执行程序。

3) 在 in.txt 中输入:

i+i

i+(i\*i)

i+i\*)(i

i\*i+(i\*(i+i))

4) 程序输出:

```

i+i Yes
i+(i*i) Yes
i+i*)(i error at 4
i*i+(i*(i+i)) Yes
Press any key to continue

```

### 2. 书 P135 文法

1) 输入文法:

ZE  
EaA  
EbB  
AcA  
Ad  
BcB  
Bd

2) 点击演示\1 自动生成工具\myYacc\tool 中的 myYacc.exe , 得到代码文件 test2.cpp, 编译得到可执行程序。

3) 在 in.txt 中输入:

bccd  
  
bcd  
  
bdc

4) 程序输出:

```
bccd Yes  
bcd Yes  
bdc error at 2  
Press any key to continue
```

### 3. 书 P165 文法

1) 输入文法:

ZA  
AaAd  
AaAb  
A\$

2) 点击演示\1 自动生成工具\myYacc\tool 中的 myYacc.exe , 得到代码文件 test2.cpp, 编译得到可执行程序。

3) 在 in.txt 中输入:

ab  
  
aba

4) 程序输出:

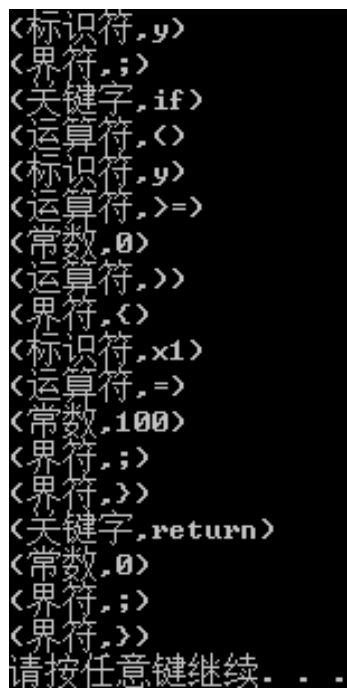
```
ab Yes  
aba error at 2  
Press any key to continue
```

## 4.3 词法分析程序测试

### 1. 测试代码

```
/*
    以下是测试程序
*/
int main(){
    int x1,x2,x3;//变量声明
    int y=0;    /*变量声明 并且赋值*/
    double xyz=3.14e+5;
    y=4+x1-(1-x2+y)/3*y; /*算数表达式*/
    if(y>=0){ /*条件语句*/
        x1=100;
    }
    return 0;
}
```

### 2. 输出截屏



```
<标识符,y>
<界符,;>
<关键字,if>
<运算符,<>
<标识符,y>
<运算符,>=>
<常数,0>
<运算符,>>
<界符,<>
<标识符,x1>
<运算符,=>
<常数,100>
<界符,;>
<界符,>>
<关键字,return>
<常数,0>
<界符,;>
<界符,>>
请按任意键继续. . .
```

## 4.4 语法分析程序测试

### 1. 测试源代码

```
/*
测试程序
*/

double y; //变量定义语句
y=3.1415e-10; //变量赋值语句
double z; //变量定义语句
float a;
a=12312;
a=3.1e+5;
a=.123e-2;
a=4+(a-(1*a+y)/3)*y;
z=(x1-3)*(4-y); //变量赋值语句
if(y>=z) //if 语句
    if(z<=y)
        z=y;
while((z+y*15-7)>10) //while 语句
    if(z<y)
        z=z-100;
```

### 2. 输入 token 流

输入格式： 序号 类型 值

```
1 1 double
2 2 y
3 5 ;
4 2 y
5 4 =
6 3 3.1415e-10
7 5 ;
```



```
8 1 double
```

```
9 2 z
```

```
10 5 ;
```

```
11 1 float
```

```
12 2 a
```

```
.....
```

### 3. 输出截屏

1) 正确时:

```
符合语法!  
请按任意键继续. . .
```

2) 将  $a=4+(a-(1*a+y)/3)*y$ ; 改为  $a=4+(a-1*a+y)/3*y$ ;

```
error at 15  
处理的串出错: a = 4 + ( a - 1 * a + y ) / 3 ) * y ;  
请按任意键继续. . .
```

3) 将  $z=y$ ; 改为  $z<=y$ ;

```
error at 13  
处理的串出错: if < y >= z > if < z <= y > z <= y ;  
请按任意键继续. . .
```

4) 将  $a=12312$ ; 去掉分号

```
error at 3  
处理的串出错: a = 12312 a = 3.1e+5 ;  
请按任意键继续. . .
```

5) 将  $\text{double } z$ ; 改为  $\text{dble } z$ ;

```
error at 1  
处理的串出错: dble z ;  
请按任意键继续. . .
```

## 5 实验总结

### 5.1 完成的工作

1. 设计要检测的语言
2. 设计和编写词法分析自动生成工具 myLex
3. 设计和编写语法分析自动生成工具 myYacc
4. 设计和编写词法分析程序
5. 设计和编写语法分析程序

### 5.2 收获与体会

通过这次课设，我了解了词法分析过程的基本思想，体会了词法分析器及其自动化生成工具的开发过程，了解了语法分析过程的基本思想，体会了语法分析器及其自动化生成工具的开发过程。此外我还提高了编写和调试程序的能力，收获很大。最后感谢王老师的悉心教导。

## 6 附录

### 附录 1：程序清单

源文件	功能
G_to_NFA.cpp	将文法转换成 NFA
NFA_to_DFA.cpp	将 NFA 转换成 DFA，并去掉多余状态
Grammar.cpp	将 DFA 最小化
Min_DFA.cpp	调用 Grammar.cpp 将 DFA 最小化，并输出程序
myYacc.cpp	生成语法自动检测代码
Cifa.cpp	词法分析程序，识别单词
yufa.cpp	语法分析程序，检查语法

## 附录 2：程序源代码——G\_to\_NFA.cpp

```
#include<iostream>
#include<fstream>
#include<string>
#include<vector>
using namespace std;
string intTochar(int n){
    string str="";
    do{
        str=char('0'+(n%10))+str;
        n/=10;
    } while(n>0);
    return str;
}
int main(){
    ifstream cin("in.txt");
    ofstream cout("out1.txt");
    vector<string> vout; //输出字符串
    vector<string> zfb; //字符集
    string start="";//开始状态
    string end="Z";//结束状态
    int a[26];

    int stateNum=1;//状态个数，开始就有结束状态
    int nextIndex=1;//z 下标为 0
    memset(a,0,sizeof(a));

    string s1,s2,s3;
    cin>>s1>>s2>>s3;
```

```

start=s1;
if(s3!="NULL")
    z fj.push_back(s2);

do{
    //状态， 不会超过 25 个
    if(a[s1[0]-'A']==0){
        a[s1[0]-'A']=nextIndex;
        stateNum++;
        nextIndex++;
    }

    string tmp=intTochar(a[s1[0]-'A'])+" ";

    if(s3=="NULL"){
        tmp+=intTochar(0)+" "+s2;
    }else{
        if(a[s3[0]-'A']==0){
            a[s3[0]-'A']=nextIndex;
            stateNum++;
            nextIndex++;
        }
        tmp+=intTochar(a[s3[0]-'A'])+" "+s2;
    }
    vout.push_back(tmp);

    //添加字符集
    if(s2!="NULL"){
        bool needAdd=true;
        for(int j=0;j<z fj.size();++j){

```

```

        if(zfj[j]==s2){
            needAdd=false;
            break;
        }
    }
    if(needAdd)
        zfj.push_back(s2);
}

}while(cin>>s1>>s2>>s3);

//状态个数 开始状态 结束状态
cout<<stateNum<<" "<<a[start[0]-'A']<<" "<<0<<endl;
//字符集个数 字符集
cout<<zfj.size()<<" ";
for(int i=0;i<zfj.size();++i)
    cout<<zfj[i]<<" ";
cout<<endl;
//起点状态 1 目的状态 1 边值 1
for(int j=0;j<vout.size();++j){
    cout<<vout[j]<<endl;
}
return 0;
}

```

### 附录 3：程序源代码——NFA\_to\_DFA.cpp

```

#include<iostream>
#include<fstream>
#include<string>

```

```

#include<vector>
#include<queue>
#include<algorithm>

using namespace std;
#define MaxState 26
#define MaxState2 400
#define MaxStr 150

//个数和开始状态，结束状态
int K,S,Z;

//字符串
int strSize;
string strs[MaxStr];

//定义 NFA 结点
struct NFA_Node{
    int state;//状态 0,1,2,3....
    string str;//到这个状态需要的字符串
    NFA_Node* pNext;
};

//定义 NFA 头结点
struct NFA_Head{
    NFA_Node* pHead;
}NFA_Heads[MaxState];

//输入集合，返回扩展后的集合，以-1 结尾
void closure(int a[]){

```

```

int ta[MaxState];
int gs=0;
for(int i=0;;i++){
    ta[i]=a[i];
    if(ta[i]==-1)
        break;
    else
        gs++;
}

for(int j=0;;j++){
    if(j>=gs)
        break;
    NFA_Node* pNode=NFA_Heads[ta[j]].pHead;
    while(pNode){
        if(pNode->str=="NULL"){
            int index=pNode->state;
            bool isexit=false;
            for(int k=0;k<gs;++k){
                if(ta[k]==index){
                    isexit=true;
                    break;
                }
            }
            if(!isexit){
                ta[gs]=index;
                gs++;
            }
        }
        pNode=pNode->pNext;
    }
}

```

```

    }
}

sort(ta,ta+gs);
ta[gs]=-1;

for(int g=0;g<=gs;++g)
    a[g]=ta[g];
}

//输入集合，返回 Move(mystr)后的集合，以-1 结尾
void move(int a[],string mystr){

    int ta[MaxState];
    //ta[0]=-1;
    int gs=0;

    //对所有的点进行遍历操作
    int index=0;
    while(a[index]!=-1){
        NFA_Node* pNode=NFA_Heads[a[index]].pHead;
        while(pNode){
            if(pNode->str==mystr){//若果需要加入的话
                //是否已经存在
                bool isexit=false;
                for(int k=0;k<gs;k++){
                    if(ta[k]==pNode->state){
                        isexit=true;
                        break;

```



```

        }
    }
    if(!isexit){
        ta[gs]=pNode->state;
        gs++;
    }
}
pNode=pNode->pNext;
}

index++;
}

sort(ta,ta+gs);
ta[gs]=-1;

for(int g=0;g<=gs;++g)
    a[g]=ta[g];
}

//DFA 定义
int Newstate[MaxState2][MaxState];
int MaxStateNum=0;
int IsDeal[MaxState2];
int endState[MaxState2];

//定义 DFA 结点
struct DFA_Node{

```

```

    int state;//状态 0,1,2,3....
    string str;//到这个状态需要的字符串
    DFA_Node* pNext;
};

//定义 DFA 头结点
struct DFA_Head{
    DFA_Node* pHead;
}DFA_Heads[MaxState2];

//判断某个集合是不是已经存在,存在的话返回下标号
int isExitSet(int a[]){
    for(int i=0;i<MaxStateNum;++i){
        bool isfit=true;
        //扫描判断是否在集合中
        for(int j=0;;j++){
            if(a[j]!=Newstate[i][j]){
                isfit=false;
                break;
            }
            if(a[j]==-1)
                break;
        }

        if(isfit)
            return i;
    }
    return -1;
}

```

```

}

//将某个集合加入到新集合中
void addSet(int a[]){
    for(int i=0;;++i){
        Newstate[MaxStateNum][i]=a[i];
        if(a[i]==Z)//终态
            endState[MaxStateNum]=1;
        if(a[i]==-1)
            break;
    }
    MaxStateNum++;
}

//将数字装换成字符串
string intTochar(int n){
    string str="";
    do{
        str=char('0'+(n%10))+str;
        n/=10;
    }while(n>0);
    return str;
}

int visited[MaxState];//记录 DFA 的每个点是否到过
//深度遍历 DFA,标记点是否走过
void dfs_DFA(int index)
{
    visited[index]=1;

```

```

    for(DFA_Node* pNode=DFA_Heads[index].pHead;;pNode=pNode->pNext){
        if(pNode==NULL)
            break;
        if(visited[pNode->state]==0)//点没有访问过
            dfs_DFA(pNode->state);
    }
}

int nowIndex=0; //当前要检测的节点
int visited2[MaxState];//记录 DFA 的每个点是否到过

//深度遍历 DFA,看当前的点能否到达终点
void dfs_DFA2(int index)
{
    visited2[index]=1;
    for(DFA_Node* pNode=DFA_Heads[index].pHead;;pNode=pNode->pNext){
        if(pNode==NULL)
            break;
        if(visited[nowIndex]==1){ //已经确定可以到达终点
            return;
        }
        if(endState[pNode->state]==1){//到达终点
            visited[nowIndex]=1;
            return;
        }else if(visited2[pNode->state]==0){//没访问过
            dfs_DFA2(pNode->state);
        }
    }
}
}

```

```

int main(){
    ifstream cin("out1.txt");
    ofstream cout("out2.txt");

    //读个数和开始状态，结束状态
    cin>>K>>S>>Z;

    //读字符集
    cin>>strSize;
    for(int i0=0;i0<strSize;++i0){
        cin>>strs[i0];
    }

    //初始化邻接矩阵
    int from,to;
    string str;
    while(cin>>from>>to>>str){
        //初始化
        NFA_Node* node=new NFA_Node;
        node->state=to;
        node->str=str;
        //头插法
        node->pNext=NFA_Heads[from].pHead;
        NFA_Heads[from].pHead=node;
    }

    /*for(int i=0;i<statue_num;++i){
        NFA_Node* pNode=NFA_Heads[i].pHead;
        while(pNode){

```

```

        cout<<i<<" "<<pNode->state<<" "<<pNode->str<<endl;
        pNode=pNode->pNext;
    }
}*/

```

//NFA to DFA 算法，子集构造算法，编译原理书 P59 上

```
//string sets[MaxState];
```

```
//int setsSize=0;
```

```
//流程
```

```
/*
```

集合扩展操作

定义一个全局 set 和 vector

在全局 vector 中放入一个初始话集合

然后掉用深度收索，将搜索到的点放入 set 中，点没有存在里面，如果存在，则下一个

Move 操作

根据集合，将所有的点用所有的输入可能 Move 一边，如果产生新的集合，那么就放入队列中

```
*/
```

//1 开始，令 closure(S)为集合中的唯一成员，并且它是未标记的。

```
int a[MaxState];
```

```
a[0]=S;
```

```
a[1]=-1;
```

```
closure(a);
```

```
addSet(a);
```

```

memset(IsDeal,0,sizeof(IsDeal));//全部未标记
memset(endState,0,sizeof(endState));//终态

//循环

//输出字符串向量
vector<int> vfrom;
vector<string> vstr;
vector<int> vto;

while(true){
    int dealIndex=-1;
    for(int i1=0;i1<MaxStateNum;++i1){
        if(IsDeal[i1]==0){
            IsDeal[i1]=1;
            dealIndex=i1;
            break;
        }
    }
    if(dealIndex==-1)
        break;
    //字符串
    for(int i2=0;i2<strSize;++i2){
        string  mystr=strs[i2];
        int ta[MaxState];
        //拷贝数组
        for(int i3=0;i3<MaxState;++i3){
            ta[i3]= Newstate[dealIndex][i3];
        }
    }
}

```

```

move(ta,mystr);
closure(ta);

string str_out="";
int addIndex=isExitSet(ta);
if(addIndex!=-1){

    //保存边
    vfrom.push_back(dealIndex);
    vstr.push_back(mystr);
    vto.push_back(addIndex);

    //把点加入的 DFA 中
    int from=dealIndex;
    int to=addIndex;

    //初始化
    DFA_Node* node=new DFA_Node;
    node->state=to;
    node->str=mystr;

    //头插法
    node->pNext=DFA_Heads[from].pHead;
    DFA_Heads[from].pHead=node;

} else {
    addSet(ta);
    addIndex=isExitSet(ta);
}

```



```

        //保存边
        vfrom.push_back(dealIndex);
        vstr.push_back(mystr);
        vto.push_back(addIndex);

        //把点加入的 DFA 中
        int from=dealIndex;
        int to=addIndex;

        //初始化
        DFA_Node* node=new DFA_Node;
        node->state=to;
        node->str=mystr;
        //头插法
        node->pNext=DFA_Heads[from].pHead;
        DFA_Heads[from].pHead=node;

    }
}
}

```

```

/*

```

消除多余状态 书 P60

无用状态：1 从自动机的开始状（从 0 开始）态出发，任何输入串也不能到达的那个状态

2 从这个状态 没有通路到达终态

```

*/

```

```

memset(visited,0,sizeof(visited));//开始时都不能到达

```

```

//去掉不能到达的状态
dfs_DFA(0);

//遍历能到的状态，看看能不能到达终点
for(int i7=0;i7<MaxStateNum;++i7){
    nowIndex=i7;
    if(visited[nowIndex]==1){//能到达
        if(endState[nowIndex]==0){//终态不用考虑，考虑非终态
            visited[nowIndex]=0;
            memset(visited2,0,sizeof(visited2));//开始时都不能到达
            dfs_DFA2(nowIndex); //如果能到达的话在再设置为 1
        }
    }
}

int stateNums=0;
for(int i8=0;i8<MaxStateNum;++i8){
    stateNums+=visited[i8];
}

//输出个数
cout<<stateNums<<endl;

//输出字符集
cout<<strSize<<endl;
for(int i4=0;i4<strSize;++i4){
    cout<<strs[i4]<<" ";
}

```

```

cout<<endl;

//要偏移的个数
int front[MaxState];//记录 DFA 的每个点是否到过
memset(front,0,sizeof(front));
for(int i9=1;i9<MaxStateNum;++i9){
    front[i9]=front[i9-1];
    if(visited[i9-1]==0)//前面的不能达到
        front[i9]++;
}

string tmp;
int gs=0;//终态个数
//终态
for(int i5=0;i5<MaxStateNum;++i5){
    if( endState[i5]==1&&visited[i5]==1){ //能够到达的终态
        gs++;
        tmp+=intTochar(i5+1-front[i5])+" ";
    }
}
cout<<gs<<endl;
cout<<tmp<<endl;

//输出边
vector<string> vout;
for(int i10=0;i10<vfrom.size();++i10){
    if(visited[vfrom[i10]]==1&&visited[vto[i10]]==1){
        string str_out=intTochar(vfrom[i10]+1-front[vfrom[i10]])+"
+vstr[i10]+" "+intTochar(vto[i10]+1-front[vto[i10]]);
        vout.push_back(str_out);
    }
}

```

```

    }
}

cout<<vout.size()<<endl;
for(int i6=0;i6<vout.size();++i6){
    cout<<vout[i6]<<endl;
}

return 0;
}

```

## 附录 4：程序源代码——Grammar.cpp

```

#include<fstream>
#include<vector>
using namespace std;
struct subset
{
    int id;
    vector<int> set;
};
class Grammar
{
public:
    Grammar(ifstream &in);
    ~Grammar(void);
    int k;//状态个数
    int tc;//终结符个数
    char t[26];//终结符

```

```

int m;//边数
int DFA[32][128];
int belong[32];//某个状态属于哪个子集
bool terminal[32];//终结符表
int terminalCount;//终结符个数
bool Mterminal[32];//最小化后的终结符表
int MDFA[32][128];//最小化的 DFA 表
int MDFAstart;// 最小化的 DFA 的初态
subset DFAsubset[32];//划分的子集
int subsetcount;//子集的个数

void erase();
vector<subset> spilt(subset I);
void min();
bool insameset(int e,subset &v);
};

Grammar::Grammar(ifstream &in)
{
    int i,u,v;
    char a;
    memset(terminal,false,sizeof(terminal));
    memset(Mterminal,false,sizeof(Mterminal));
    for(u=0;u<32;u++)
        for(v=0;v<128;v++)
        {
            DFA[u][v]=-1;
            MDFA[u][v]=-1;
        }
    this->subsetcount=0;
}

```

```

in>>this->k;//状态个数
in>>this->tc;//终结符个数
for(i=0;i<tc;i++)//所有终结符
    in>>t[i];
in>>this->terminalCount;//终态个数
for(i=0;i<this->terminalCount;i++)//标志终态
{
    in>>u;
    terminal[u]=true;
}
in>>this->m;//边数
for(i=0;i<m;i++)//构造邻接矩阵
{
    in>>u>>a>>v;
    DFA[u][a]=v;
}
this->min();
}

void Grammar::min()
{
    subset start,end;
    int i,id=0;
    start.id=id++;
    end.id=id++;
    for(i=1;i<=this->k;i++)
    {
        if(terminal[i])
        {
            end.set.push_back(i);

```

```

        belong[i]=end.id;
    }
    else
    {
        start.set.push_back(i);
        belong[i]=start.id;
    }
}
DFAsubset[subsetcount++]=start;
DFAsubset[subsetcount++]=end;
while(true)
{
    bool flag=false;
    for(i=0;i<subsetcount;i++)
    {
        if(DFAsubset[i].set.size()<2) continue;
        vector<subset> vs=spilt(DFAsubset[i]);
        if(vs.size()>1)
        {
            flag=true;
            vs[0].id=DFAsubset[i].id;
            DFAsubset[i]=vs[0];
            for(unsigned int j=1;j<vs.size();j++)
            {
                DFAsubset[subsetcount]=vs[j];
                DFAsubset[subsetcount].id=id;
                for(unsigned int j=0;j<DFAsubset[subsetcount].set.size();j++)
                    belong[DFAsubset[subsetcount].set[j]]=id;
                subsetcount++;
                id++;
            }
        }
    }
}

```

```

        }
    }
}
if(!flag)
    break;
}
for(i=0;i<subsetcount;i++)
{
    int j;
    for(j=0;j<tc;j++)//确定 MDFA 的状态转换表
        MDFA[i][t[j]]=belong[DFA[DFAsubset[i].set[0]][t[j]]];
    for(j=0;j<DFAsubset[i].set.size();j++)//确定 MDFA 的终态和初态
    {
        if(terminal[DFAsubset[i].set[j]])
            Mterminal[i]=true;
        if(DFAsubset[i].set[j]==1)
            MDFAstart=i;
    }
}
}
vector<subset> Grammar::spilt(subset I)
{
    subset sub;
    vector<subset> U;
    sub.set.push_back(I.set[0]);
    U.push_back(sub);
    for(unsigned int i=1;i<I.set.size();i++)
    {
        bool flag=false;
        for(unsigned int j=0;j<U.size();j++)

```



```

        if(insameset(I.set[i],U[j]))
        {
            flag=true;
            U[j].set.push_back(I.set[i]);
            break;
        }
    if(!flag)
    {
        subset t;
        t.set.push_back(I.set[i]);
        U.push_back(t);
    }
}
return U;
}

bool Grammar::insameset(int e,subset &u)
{
    int s=u.set[0];
    for(int i=0;i<tc;i++)
        if(belong[DFA[e][t[i]]]!=belong[DFA[s][t[i]]])
            return false;
    return true;
}

void Grammar::erase()
{
}

Grammar::~Grammar(void)
{
}

```

## 附录 5：程序源代码——Min\_DFA.cpp

```
#include "Grammar.cpp"
#include <stdio.h>
#include <string>

string intTochar(int n){
    string str="";
    do{
        str=char('0'+(n%10))+str;
        n/=10;
    } while(n>0);
    return str;
}

void main()
{
    ifstream in("out2.txt");
    freopen("out3.txt","w",stdout);

    Grammar G(in);
    int i,k;
    //print DFA
    printf("初始自动机为:\n    ");
    for(i=0;i<G.tc;i++)
        printf("%-4c",G.t[i]);
    printf("\n");
    for(k=1;k<=G.k;k++)
    {
        printf(" %d    ",k);
```

```

        for(i=0;i<G.tc;i++)
        {
            if(G.DFA[k][G.t[i]]!=-1)
                printf("%-4d",G.DFA[k][G.t[i]]);
            else printf("    ");
        }
        printf("\n");
    }
    printf("初始状态为:1\n 接收状态为:");
    for(k=0;k<32;k++)
        if(G.terminal[k])
            printf("%d ",k);
    printf("\n");
    //print MDFA
    printf("最小化之后为:\n    ");
    for(i=0;i<G.tc;i++)
        printf("%-4c",G.t[i]);
    printf("\n");
    for(k=0;k<G.subsetcount;k++)
    {
        printf(" %d  ",k+1);
        for(i=0;i<G.tc;i++)
        {
            if(G.MDFA[k][G.t[i]]!=-1)
                printf("%-4d",G.MDFA[k][G.t[i]]+1);
            else printf("    ");
        }
        printf("\n");
    }
}

```

```

printf("初始状态是:%d\n",G.MDFAstart+1);
printf("接收状态为:");
for(k=0;k<G.subsetcount;k++)
    if(G.Mterminal[k])
        printf("%d ",k+1);
printf("\n");

/*输出代码*/
ofstream cout("test.cpp");

cout<<"#include<iostream>"<<endl;
cout<<"#include<fstream>"<<endl;
cout<<"#include<string>"<<endl;
cout<<"using namespace std;"<<endl;

cout<<endl;

cout<<"//函数声明"<<endl;
for(k=0;k<G.subsetcount;k++)
    cout<<"bool S"<<intTochar(k+1)<<"(string str,int index);"<<endl;
cout<<endl;

cout<<endl;

//输出函数
for(k=0;k<G.subsetcount;k++)
{
    cout<<"bool S"<<intTochar(k+1)<<"(string str,int index){";
    if(k==G.MDFAstart)

```

```

        cout<<" //起始态";
    if(G.Mterminal[k])
        cout<<" //终态";
    cout<<endl;

    cout<<" if(index>=str.length())//超出界限"<<endl;
    if(G.Mterminal[k])//终态
        cout<<"    return true;"<<endl;
    else
        cout<<"    return false;"<<endl;

    for(i=0;i<G.tc;i++)
    {
        if(G.MDFA[k][G.t[i]]!=-1){
            cout<<" if(str[index]==""<<G.t[i]<<""){    // 输 入 字 符 为
""<<G.t[i]<<""<<endl;
            cout<<"    index++;"<<endl;
            cout<<"    return
S"<<intTochar(G.MDFA[k][G.t[i]]+1)<<"(str,index);"<<endl;
            cout<<" }"<<endl;
        }
    }
    cout<<" return false;"<<endl;
    cout<<"}"<<endl<<endl;
}

//输出主函数
cout<<"int main(){"<<endl;
cout<<" ifstream cin(\"in.txt\");"<<endl;
cout<<" string str;    //待检测字符串"<<endl;

```

```

cout<<" int index=0; //待检测字符位置"<<endl;
cout<<" while(cin>>str){"<<endl;
cout<<"     index=0;"<<endl;
cout<<"     if(S1(str,index))"<<endl;
cout<<"         cout<<" √  \ "<<str<<endl;"<<endl;
cout<<"     else"<<endl;
cout<<"         cout<<" ×  \ "<<str<<endl;"<<endl;
cout<<" }"<<endl;
cout<<" return 0;"<<endl;
cout<<"}"<<endl;
}

```

## 附录 6：程序源代码——myYacc.cpp

```

#include<iostream>
#include<vector>
#include<fstream>
#include<string>
#include<iomanip>
#include<algorithm>
#include<stack>
using namespace std;
#define MaxStatue 26 //状态数目最多 26
vector<string> vstr0;//初始文法 S->AB 记为 SAB
vector<string> vstr1;//存放加点的项目 S.AB SA.B SAB.

string X=""; //存放所有的状态
string endStr=""; //存放所有的终态
int CK[MaxStatue]; //能否生成空，0 未知，-1 不能，1 能

```

```

bool isLR1=true;

//求出能推出$的终结符,输入 vstr0 中的表达式, 输出字符串表和是产生空
void FindKong(){
    int i;
    //记录产生式左边的个数
    int gs[MaxStatue];
    memset(gs,0,sizeof(gs));

    //首先获得状态, 通过解析字符串
    for(i=0;i<vstr0.size();++i){
        string tmp=vstr0[i];
        for(int j=0;j<tmp.size();++j){
            if(tmp[j]>='A'&&tmp[j]<='Z'){//是非终态
                bool find=false;//是否已经存在了
                for(int k=0;k<X.size();++k){
                    if(X[k]==tmp[j]){
                        find=true;
                        if(j==0)//以它开头
                            gs[k]++;//产生式增加一个
                        break;
                    }
                }
                //没找到的话就加进去
                if(!find){
                    if(j==0)//以它开头
                        gs[X.size()]++;
                    X+=tmp[j];
                }
            }
        }
    }
}

```

```

    }else{//是终态
        if(tmp[j]!='$'){//不为空
            bool find=false;//是否已经存在了
            for(int k=0;k<endStr.size();++k){
                if(endStr[k]==tmp[j]){
                    find=true;
                    break;
                }
            }
            //没找到的话就加进去
            if(!find)
                endStr+=tmp[j];
        }
    }
}

sort(endStr.begin(),endStr.end());
//初始化产生空的集合
memset(CK,0,sizeof(CK)); //未知

//删除产生式右边所有含有终结符的产生式，若使得
//某一终结符产生式都被删除，则标记为否
//若某一产生式的右部为空（$）,则标记为是，删除产生式
vector<string> v_str;
for(i=0;i<vstr0.size();++i){
    if(vstr0[i][1]>='A'&&vstr0[i][1]<='Z'){//非终结符就保留
        v_str.push_back(vstr0[i]);
    }else if(vstr0[i][1]=='$'){//为空

```



```

        for(int j=0;j<X.size();++j){//删除
            if(X[j]==vstr0[i][0]){
                CK[j]=1;//能产生空
                break;
            }
        }
    }else{//为终结符
        for(int j=0;j<X.size();++j){//删除
            if(X[j]==vstr0[i][0]){
                gs[j]--;//删去此产生式，个数数组并减一
                if(gs[j]==0)
                    CK[j]=-1;//不能产生空
            }
        }
    }
}

//删除能产生空的产生式
vector<string> v_str1;
for(i=0;i<v_str.size();++i){
    bool needDelet=false;
    for(int j=0;j<X.size();++j){
        if(v_str[i][0]==X[j]&&CK[j]==1){
            gs[j]--;
            needDelet=true;
            break;
        }
    }
    if(!needDelet)//如果未确定就加入
        v_str1.push_back(v_str[i]);
}

```

```

}

v_str.clear();//释放资源

//扫描，直到没有再变为止
while(true){
    bool ischange=false;//默认没变
    vector<string> v_str2;
    for(i=0;i<v_str1.size();++i)
        v_str2.push_back(v_str1[i]);
    v_str1.clear(); //存放在 v_str2 中

    for(i=0;i<v_str2.size();++i){//遍历每个表达式

        if(!(v_str2[i][1]>='A'&&v_str2[i][1]<='Z')){//如果是终结符
            for(int j=0;j<X.size();++j){//删除
                if(X[j]==v_str2[i][0]){
                    gs[j]--;//删去此产生式，个数数组并减一
                    if(gs[j]==0)
                        CK[j]=-1;//不能产生空
                }
            }
            ischange=true;
            continue;//下一个
        }

        for(int k=0;k<X.size();++k){//找非终结符

            if(X[k]==v_str2[i][1]){//找到产生式右部字符串

```

```

if(CK[k]==1){//可以产生空
    ischange=true;
    if(v_str2[i].length()==2){//修改标志并删除
        for(int k0=0;k0<X.size();++k0){
            if(X[k0]==v_str2[i][0]){
                CK[k0]=1;
                break;
            }
        }
    }else{
        string tmp="";
        for(int j=0;j<v_str2[i].length();++j){
            if(j==1)
                continue;
            else
                tmp+=v_str2[i][j];
        }
        v_str1.push_back(tmp);//不删除
    }
}else if(CK[k]==-1){//不能产生空
    ischange=true;
    for(int k0=0;k0<X.size();++k0){
        if(X[k0]==v_str2[i][0]){
            gs[k0]--;
            if(gs[k0]==0)
                CK[k0]=-1;
            break;
        }
    }
}
}

```

```

        }else{//未知
            v_str1.push_back(v_str2[i]);
        }

        }//找到产生式右部字符串

    }//X 遍历

} //遍历每个表达式

if(!ischange) //没有改变
    break;
} //while
}

//字符串排序时使用的函数
bool cmp(const char ch1,const char ch2){
    return ch1<ch2;
}

string First_set[MaxStatue];//记录每个状态的集合
int First_finish[MaxStatue];//每个状态是否确认
int Postion[MaxStatue]; //将对应非终态字母快速找到对应位置
//计算每个字符集的 FIRST 集
string First(int index){
    string tmp="";
    int i;
    bool cKong=false;//能产生空
    for(i=0;i<vstr0.size();++i){ //遍历表达式

```

```

if(X[index]==vstr0[i][0]){//找到对应的表达式
    bool canKong=true;//能产生空
    for(int j=1;j<vstr0[i].length();++j){//遍历表达式的每一个字符
        if(!canKong) //不能产生空就退出
            break;
        if(vstr0[i][j]=='$'){ //生成空
            tmp+='$';
        }else if(!(vstr0[i][j]>='A'&&vstr0[i][j]<='Z')){//终结符
            canKong=false;
            tmp+=vstr0[i][j];
        }else{
            int index2=Postion[vstr0[i][j]-'A'];
            if(CK[index2]!=1)//不能产生空
                canKong=false;
            if(First_finish[index2]==1)//已经求出
                tmp+=First_set[index2];
            else
                if(index2!=index) //自身不递归
                    tmp+=First(index2);//递归求解
        }
    }
    }//遍历表达式的每一个字符
    if(canKong)
        cKong=true;
    }//找到对应的表达式
}

//去掉重复的字母，如果不能生成空的话就不生成
string tmp2="";
for(i=0;i<tmp.length();++i){
    if(tmp[i]=='$')

```

```

        continue;
    bool isExit=false;
    for(int j=0;j<tmp2.length();++j){
        if(tmp[i]==tmp2[j]){
            isExit=true;
            break;
        }
    }
    if(!isExit)
        tmp2+=tmp[i];
}
sort(tmp2.begin(),tmp2.begin()+tmp2.length(),cmp);
if(cKong)
    tmp2+='$';

First_set[index]=tmp2;
First_finish[index]=1;//该集合完成
return tmp2;
}

//初始化每个字符集的 FIRST 集
void First_init(){
    memset(First_finish,0,sizeof(First_finish));//初始化都没确认
    int i;
    for(i=0;i<X.length();++i)//将对应非终态字母快速找到对应位置
        Postion[X[i]-'A']=i;

    for(i=0;i<X.length();++i)
        if(First_finish[i]!=1)//未初始化就进行初始化
            First(i);
}

```

```

}

//计算指定字符串的 FIRST 集,算法书 P80, 入口参数 str,返回 string,$表示空
string FIRST_str(string str){
    bool canKong=true;//能产生空
    int i;
    string tmp="";
    for(i=0;i<str.length();++i){//遍历表达式的每一个字符
        if(!canKong) //不能产生空就退出
            break;
        if(str[i]=='$'){ //生成空
            tmp+='$';
        }else if(!(str[i]>='A'&&str[i]<='Z')){//终结符
            canKong=false;
            tmp+=str[i];
        }else{//
            int index2=Postion[str[i]-'A'];
            if(CK[index2]!=1)//不能产生空
                canKong=false;
            tmp+=First_set[index2];
        }
    }
}

//去掉重复的字母, 如果不能生成空的话就不生成
string tmp2="";
for(i=0;i<tmp.length();++i){
    if(tmp[i]=='$')
        continue;
    bool isExit=false;
    for(int j=0;j<tmp2.length();++j){

```

```

        if(tmp[i]==tmp2[j]){
            isExit=true;
            break;
        }
    }
    if(!isExit)
        tmp2+=tmp[i];
}
sort(tmp2.begin(),tmp2.begin()+tmp2.length(),cmp);
if(canKong)
    tmp2+='$';
return tmp2;
}

//CLOSURE(I)函数的数据结构      书 p144
struct Node{ //项目节点
    vector<string> vState;//状态
};
vector<Node> vNode; //存放所有的项目

//对项目集合进行扩展，找到 A->.B， B->.c...
vector<string> vtmp;//存放递归找的项目元素
int added[200];//记录对应产生式是否加入过
void findNext(char ch){
    for(int i=0;i<vstr1.size();++i){
        if(vstr1[i][0]==ch&&vstr1[i][1]=='.'){//找到产生式
            if(vstr1[i].length()==2){//s->.
                vtmp.push_back(vstr1[i]);
            }
        }
    }
}

```



```

        }else if(vstr1[i][2]>='A'&&vstr1[i][2]<='Z'){//非终结符
            if(added[i]==0){
                vtmp.push_back(vstr1[i]);
                added[i]=1;
                findNext(vstr1[i][2]);//递归函数
            }
        }else //终结符
            vtmp.push_back(vstr1[i]);
    }
}

//字符串比较函数
bool cmp2(const string s1,const string s2){
    return s1<s2;
}

//CLOSURE(I)函数
void CLOSURE(Node& node){
    int i;
    for(i=0;i<node.vState.size();++i){
        string tmp=node.vState[i];
        int j=tmp.find('.'); //找到点的位置
        if(tmp[j+1]>='A'&&tmp[j+1]<='Z'){//非终结符
            vtmp.clear();//清空
            memset(added,0,sizeof(added));//初始化，没有使用过
            findNext(tmp[j+1]); //递归进行扩展
            //若有项目 A->aBp,a 属于 CLOSURE(I), B->r 是产生式, b 属于
FIRST(pa),则 B->.r,b
            int j2=tmp.find('.');
            string str_p="";

```

```

int i0;

for(i0=j+2;i0<j2;++i0)
    str_p+=tmp[i0];
string str_pa=str_p+tmp[j2+1];
string str_first_pa=FIRST_str(str_pa);//计算 First(pa)

for(int k=0;k<vtmp.size();++k){ //加入节点中，暂时是 S.AB 的形式
    if(vtmp[k][0]!=tmp[j+1])
        continue;
    for( int k1=0;k1<str_first_pa.size();++k1){
        string addstr=vtmp[k]+' '+str_first_pa[k1];
        bool find=false;
        for(int j=0;j<node.vState.size();++j){
            if(addstr==node.vState[j]){
                find=true;
                break;
            }
        }
        if(!find){
            node.vState.push_back(addstr);
        }
    }
}

}

sort(node.vState.begin(),node.vState.end(),cmp2); //按字符顺序排序
}

//GO(I,x)

```

```

void Go(Node& sNode,char ch,Node& eNode){//起始状态， 字符， 结果状态
    int i;
    for(i=0;i<sNode.vState.size();++i){//遍历集合中每个状态元素 S.aAb
        string tmp=sNode.vState[i];
        int index=tmp.find('.');
        if(tmp[index+1]==ch){//需要这个字符
            //交换
            tmp[index]=ch;
            tmp[index+1]='.';
            //加入
            eNode.vState.push_back(tmp);
        }
    }
}

//判断集合是否存在
int isExit(Node& eNode){
    string str0="";
    int i;
    for(i=0;i<eNode.vState.size();++i)//将集合转换成字符串
        str0+=eNode.vState[i];

    for(i=0;i<vNode.size();++i){//与每个集合比较
        if(vNode[i].vState.size()!=eNode.vState.size()) //个数不同， 集合不同
            continue;
        string tmp="";
        for(int j=0;j<vNode[i].vState.size();++j)
            tmp+=vNode[i].vState[j];
        if(tmp==str0)
            return i;
    }
}

```

```

    }
    return -1;
}

//生成 LR(1)分析表
#define acc 140406
int Action[100][100]; //Action 表, 0 表示出错, 正数表示 Si, 负数表示 ri, 140406 表示
结束
int Goto[100][100]; //Goto 表

//找到非终态符号在 Action 中的下标, 找不到返回-1
int find_endStr(char ch){
    if(ch=='#')
        return endStr.length();
    for(int i=0; i<endStr.length(); ++i)
        if(ch==endStr[i])
            return i;
    return -1;
}

int setIndex=0;
//构造识别活动前缀的 DFA
void CreateDFA(){
    //初始化
    Node node;
    string str=vstr1[0]+"#";
    node.vState.push_back(str);
    CLOSURE(node); //求项目集的闭包

```

```

vNode.push_back(node); //加入队列

int i;
cout<<"I0: "<<endl;
for(i=0;i<node.vState.size();++i)
    cout<<node.vState[i]<<endl;
cout<<endl;

//int i;
for(i=0;i<vNode.size();++i){//处理每个集合
    int j;
    int nowIndex=vNode.size();

    //处理非终态 A,B,C,D...
    for(j=0;j<X.length();++j){
        Node eNode;
        Go(vNode[i],X[j],eNode);    //变换
        if(eNode.vState.size(>0)){
            CLOSURE(eNode);//扩展
            int int_exit=isExit(eNode);
            if(int_exit==-1){//判断集合是否存在
                vNode.push_back(eNode);
                ++setIndex;
                cout<<"I"<<i<<"-> "<<X[j]<<" ->I"<<setIndex<<endl;
                cout<<"I"<<setIndex<<": "<<endl;
                for(int i1=0;i1<eNode.vState.size();++i1)
                    cout<<eNode.vState[i1]<<endl;
                cout<<endl;

                if( Goto[i][j]!=0&&Goto[i][j]!=setIndex){

```

```

        cout<<"出错！ Goto 表位置上有冲突"<<endl;
        isLR1=false;
        return;
    }
    Goto[i][j]=setIndex;// 添 加 到 Goto 表 ,GO(Ik,A)=Ij, 则
Goto[k,A]=j

    }else{//已经存在
        if( Goto[i][j]!=0&&Goto[i][j]!=int_exit){
            cout<<"出错！ Goto 表位置上有冲突"<<endl;
            isLR1=false;
            return;
        }
        Goto[i][j]=int_exit;// 添 加 到 Goto 表 ,GO(Ik,A)=Ij, 则
Goto[k,A]=j

    }

    }//判断集合是否存在
}

}

//处理终态， a,b,c...
for(j=0;j<endStr.length();++j){
    Node eNode;
    Go(vNode[i],endStr[j],eNode); //变换
    if(eNode.vState.size()>0){
        CLOSURE(eNode);//扩展
        int int_exit=isExit(eNode);
        if(int_exit==-1){//判断集合是否存在
            vNode.push_back(eNode);
            ++setIndex;
            cout<<"I"<<i<<"-> "<<endStr[j]<<"->I"<<setIndex<<endl;

```

```

        cout<<"I"<<setIndex<<": "<<endl;
        for(int i1=0;i1<eNode.vState.size();++i1)
            cout<<eNode.vState[i1]<<endl;
        cout<<endl;

        if( Action[i][j]!=0&&Action[i][j]!=setIndex){
            cout<<"出错！ Action 表位置上有冲突"<<endl;
            isLR1=false;
            return;
        }
        Action[i][j]=setIndex;// 添 加 到 Action 表 ,GO(Ik,a)=Ij, 则
Action[k,a]=Sj
    }else{
        if( Action[i][j]!=0&&Action[i][j]!=int_exit){
            cout<<"出错！ Action 表位置上有冲突"<<endl;
            isLR1=false;
            return;
        }
        Action[i][j]=int_exit;// 添 加 到 Action 表 ,GO(Ik,a)=Ij, 则
Action[k,a]=Sj
    }//判断集合是否存在
}

}

//检查规约
for(int k=0;k<vNode[i].vState.size();++k){//若 [A->x.,a]属于 Ik, 则设置
Action[k,a]=rj
    string ss=vNode[i].vState[k];

```

```

if(ss[ss.length()-3]=='.'){
    int ss_index=find_endStr(ss[ss.length()-1]);
    if(ss_index==-1){
        cout<<"出错！出现未识别的终结符"<<endl;
        isLR1=false;
        return;
    }else{
        //找到规约的表达式
        string find_str="";
        int k1=0;
        for(k1=0;k1<ss.length()-3;++k1)
            find_str+=ss[k1];
        if(find_str.length()==1)
            find_str+='$';
        for(k1=0;k1<vstr0.size();++k1)
            if(find_str==vstr0[k1])
                break;

        if( Action[i][ss_index]!=0&&Action[i][j]!=-k1){
            cout<<"出错！Action 表位置上有冲突"<<endl;
            isLR1=false;
            return;
        }

        Action[i][ss_index]=-k1;

        if(ss[ss.length()-1]=='#'&&ss[0]=='Z')
            Action[i][endStr.length()]=acc;
    }
}

```



```

        }//若[A->x.,a]属于 Ik， 则设置 Action[k,a]=rj
    }
}

int main(){
    ifstream cin("g.txt");
    //读入拓广文法
    string str0;
    while(cin>>str0){
        vstr0.push_back(str0);
    }

    //添加原点， 构成项目
    int i;
    for(i=0;i<vstr0.size();++i){
        if(vstr0[i].length()==2&&vstr0[i][1]=='$'){//s->$
            vstr0[i][1]='.';
            vstr1.push_back(vstr0[i]);
            vstr0[i][1]='$';
            continue;
        }
        string tmp="."+vstr0[i];
        for(int j=0;j<vstr0[i].size();++j){//长多少， 插入多少次， 从位置 1 开始，
            //开头为产生式左部
            char t_ch=tmp[j];
            tmp[j]=tmp[j+1];
            tmp[j+1]=t_ch;
            vstr1.push_back(tmp);
        }
    }
}

```

```

    }
}

//标号
cout<<"0 对产生式标号为: "<<endl;
for(i=0;i<vstr0.size();++i){
    cout<<i<<": "<<vstr0[i]<<endl;
}
cout<<endl;

//求出能产生空的非非终结符
FindKong();

cout<<"1 求出能产生空的非非终结符: "<<endl;
for(i=0;i<X.size();++i)
    cout<<X[i]<<" ";
cout<<endl;
for(i=0;i<X.size();++i){
    if(CK[i]==-1)
        cout<<0<<" ";
    else
        cout<<CK[i]<<" ";
}
cout<<endl;
cout<<endl;

//计算每个字符集的 FIRST 集
cout<<"2 计算每个字符集的 FIRST 集: "<<endl;
First_init();
for(i=0;i<X.length();++i)

```

```

        cout<<X[i]<<": "<<First_set[i]<<endl;
    cout<<endl;

    //清空表
    memset(Action,0,sizeof(Action));
    memset(Goto,0,sizeof(Goto));

    //生成带活动前缀的 DFA
    cout<<"3 生成带活动前缀的 DFA:"<<endl;
    CreateDFA();

    //检查是否是 LR1 文法
    if(!isLR1){
        cout<<"该文法不是 LR1 文法！ "<<endl;
        system("pause");
        return 0;
    }
    cout<<endl;

    //生成 LR(1)分析表
    cout<<"4 生成 LR(1)分析表:"<<endl;
    int gs=4;

    for(i=0;i<endStr.length();++i)
        cout<<setw(gs-1)<<endStr[i]<<" ";
    cout<<setw(gs-1)<<"#"<<" ";
    for(i=0;i<X.length();++i)
        cout<<setw(gs)<<X[i]<<" ";
    cout<<endl;

```

```

for(i=0;i<vNode.size();++i){
    int j;
    for(j=0;j<=endStr.length();++j){
        if(Action[i][j]>0){
            if(Action[i][j]==acc)
                cout<<setw(gs-2)<<"acc"<<" ";
            else
                cout<<setw(gs-2-(Action[i][j]/10))<<"S"<<Action[i][j]<<" ";
        }else if(Action[i][j]<0)
            cout<<setw(gs-2-(-Action[i][j])/10)<<"r"<<-Action[i][j]<<" ";
        else
            cout<<setw(gs)<<"0 ";
    }

    for(j=0;j<X.length();++j)
        cout<<setw(gs)<<Goto[i][j]<<" ";

    cout<<endl;
}

//输出 Cpp 文件
/*输出代码*/
ofstream cout("test2.cpp");

cout<<"#include<iostream>"<<endl;
cout<<"#include<fstream>"<<endl;
cout<<"#include<string>"<<endl;
cout<<"#include<stack>"<<endl;
cout<<"#include<vector>"<<endl;
cout<<"using namespace std;"<<endl;
cout<<endl;

```

```

cout<<"#define acc 140406"<<endl;
cout<<endl;
cout<<"//LR(1)分析表"<<endl;
cout<<"int Action[100][100];//Action 表,0 表示出错,正数表示 Si,负数表示
ri,140406 表示结束"<<endl;
    cout<<"int Goto[100][100];//Goto 表"<<endl;
cout<<endl;
cout<<"//相关变量"<<endl;
cout<<"vector<string> vstr0;//初始文法 S->AB 记为 SAB"<<endl;
cout<<"string X=\"\"<<X<<\"\"; //存放所有的状态"<<endl;
cout<<"string endStr=\"\"<<endStr<<\"\"; //存放所有的终态"<<endl;
cout<<endl;
//输出函数
cout<<"//找到非终态符号在 Action 中的下标, 找不到返回-1"<<endl;
cout<<"int find_endStr(char ch){"<<endl;
cout<<" if(ch=='#')"<<endl;
cout<<"    return endStr.length();"<<endl;
cout<<" for(int i=0;i<endStr.length();++i)"<<endl;
cout<<" if(ch==endStr[i])"<<endl;
cout<<"    return i;"<<endl;
cout<<" return -1;"<<endl;
cout<<"}"<<endl;
cout<<endl;
//输出函数
cout<<"void init()//初始化"<<endl;
cout<<" //初始化 Action 表,Goto 表"<<endl;
for(i=0;i<=vNode.size();++i){
    int j;
    for(j=0;j<=endStr.length();++j)
        cout<<" Action["<<i<<"]["<<j<<"]="<<Action[i][j]<<";"<<endl;
}
}

```

```

    }

    for(i=0;i<=vNode.size();++i){
        int j;
        for(j=0;j<=X.length();++j)
            cout<<" Goto["<<i<<"]["<<j<<"]="<<Goto[i][j]<<";"<<endl;
    }

    //初始化 vstr0
    for(i=0;i<vstr0.size();++i){
        cout<<" vstr0.push_back(\""<<vstr0[i]<<"\");"<<endl;
    }
    cout<<"}"<<endl;
    cout<<endl;
    //输出函数
    cout<<"bool Yufa(string inStr){//进行语法分析"<<endl;
    cout<<" stack<int> stack_state;//状态栈"<<endl;
    cout<<" stack<char> stack_fh;//符号栈"<<endl;
    cout<<" //初始化"<<endl;
    cout<<" stack_state.push(0);"<<endl;
    cout<<" stack_fh.push('#');"<<endl;
    cout<<" inStr+='#';"<<endl;
    cout<<" for(int index=0;index<inStr.length();++index){"<<endl;
    cout<<"     char ch=inStr[index];"<<endl;
    cout<<"     int state_index=stack_state.top();//栈顶状态"<<endl;
    cout<<"     if(ch>='A'&&ch<='Z'){//非终态"<<endl;
    cout<<"         int goto_index=X.find(ch);"<<endl;
    cout<<"         if(goto_index<0||Goto[state_index][goto_index]==0){//错误状
态"<<endl;

    cout<<"             cout<<"error at \"<<index<<endl;""<<endl;
    cout<<"             return false;""<<endl;

```

```

cout<<"          }else{//非错误状态"<<endl;
cout<<"          stack_fh.push(ch);//字符进栈"<<endl;
cout<<"          stack_state.push(Goto[state_index][goto_index]);//状态进
栈"<<endl;
cout<<"          }"<<endl;
cout<<"      }else{//终态"<<endl;
cout<<"          int action_index=find_endStr(ch);"<<endl;
cout<<"          if(action_index<0||Action[state_index][action_index]==0){//错
误状态"<<endl;
cout<<"          cout<<"error at \"<<index<<endl;"<<endl;
cout<<"          return false;"<<endl;
cout<<"          }else if(Action[state_index][action_index]==acc){//接受状态
"<<endl;
cout<<"          cout<<"Yes"<<endl;"<<endl;
cout<<"          return true;"<<endl;
cout<<"          }else if(Action[state_index][action_index]>0){//Si"<<endl;
cout<<"          stack_fh.push(ch);//字符进栈"<<endl;
cout<<"          stack_state.push(Action[state_index][action_index]);//状态
进栈"<<endl;
cout<<"          }else{//ri"<<endl;
cout<<"          string t_str=vstr0[-Action[state_index][action_index]];//找
到产生式"<<endl;
cout<<"          int k=t_str.length()-1;"<<endl;
cout<<"          if(t_str[1]=='$')"<<endl;
cout<<"          k=0;//s-A."<<endl;
cout<<"          for(int k0=0;k0<k;++k0){"<<endl;
cout<<"          stack_fh.pop();//字符弹栈"<<endl;
cout<<"          stack_state.pop();//状态弹栈"<<endl;
cout<<"          }"<<endl;
cout<<"          state_index=stack_state.top(); "<<endl;

```

```

    cout<<"          int goto_index=X.find(t_str[0]);"<<endl;
    cout<<"          if(goto_index<0||Goto[state_index][goto_index]==0){// 错
误状态"<<endl;

    cout<<"          cout<<"error3 at \"<<index<<endl;"<<endl;
    cout<<"          return false;"<<endl;
    cout<<"          }else{//非错误状态"<<endl;
    cout<<"          stack_fh.push(t_str[0]);//字符进栈"<<endl;
    cout<<"          stack_state.push(Goto[state_index][goto_index]);// 状
态进栈"<<endl;

    cout<<"          }"<<endl;
    cout<<"          index--;"<<endl;
    cout<<"          }"<<endl;
    cout<<"      }"<<endl;
    cout<<" }"<<endl;
    cout<<"}"<<endl;

//输出主函数
cout<<"int main(){"<<endl;
cout<<" ifstream cin(\"in.txt\");"<<endl;
cout<<" string str;    //待检测字符串"<<endl;
cout<<" init();//初始化    "<<endl;
cout<<" while(cin>>str){"<<endl;
cout<<"      cout<<str<<" \"<<endl;
cout<<"      Yufa(str);"<<endl;
cout<<" }"<<endl;
cout<<" return 0;"<<endl;
cout<<"}"<<endl;

system("pause");

```



```
    return 0;
}
```

## 附录 7：程序源代码——Cifa.cpp

```
#include<iostream>
#include<fstream>
#include<string>
using namespace std;

string str;
int index=0;
int token_index=1;
ofstream out("token.txt");

//获得界符，是：返回结束的位置，不是：返回-1
int getJieFu(){
    if(index>=str.length()) //扫描完后返回
        return -1;
    if(str[index]=='||str[index]=='|'||str[index]=='{'||str[index]=='}')
        return index;
    else
        return -1;
}

//获得运算符，是：返回结束的位置，不是：返回-1
int getYunSuanFu(){
    if(index>=str.length()) //扫描完后返回
        return -1;
    if(str[index]=='+'||str[index]=='-'||str[index]=='*'||str[index]=='/'
        ||str[index]=='%'||str[index]=='='||str[index]=='('||str[index]==')')
```

```

        return index;
    if(str[index]=='>'||str[index]=='<'){
        if(index+1>=str.length()) //扫描完后返回
            return index;
        else{
            if(str[index+1]=='=') //>=或者<=
                return index+1;
            else //>或者<
                return index;
        }
    }
    return -1;
}

//判断是否是标识符 2
bool isBiaozhifu2(string str,int index){ //终态
    if(index>=str.length())//超出界限
        return true;
    if(str[index]>='a'&&str[index]<='z'){ //输入字符为'l'
        index++;
        return isBiaozhifu2(str,index);
    }
    if(str[index]>='0'&&str[index]<='9'){ //输入字符为'd'
        index++;
        return isBiaozhifu2(str,index);
    }
    return false;
}
}

```

```

//判断是否是标识符
bool isBiaozhifu(string str,int index){ //起始态
    if(index>=str.length())//超出界限
        return false;
    if(str[index]>='a'&&str[index]<='z'){ //输入字符为'l'
        index++;
        return isBiaozhifu2(str,index);
    }
    return false;
}

string gjz_strs[]={//关键字表
    "bool",
    "break",
    "case",
    "catch",
    "char",
    "class",
    "const",
    "continue",
    "default",
    "delete",
    "do",
    "double",
    "else",
    "false",
    "float",
    "for",
    "if",
    "int",

```

```
"long",
"main",
"namespace",
"new",
"private",
"protected",
"public",
"return",
"short",
"sizeof",
"static",
"struct",
"switch",
"this",
"throw",
"true",
"try",
"void",
"while",
"mdw"};
//判断是否是关键字
bool isGuanJiZi(string dealStr){
    for(int i=0;;i++){
        if(gjz_strs[i]=="mdw") //查完了
            return false;
        if(gjz_strs[i]==dealStr)//查到了
            return true;
    }
}
```

```

//是否是常数函数声明
bool S1(string str,int index);
bool S2(string str,int index);
bool S3(string str,int index);
bool S4(string str,int index);
bool S5(string str,int index);
bool S6(string str,int index);
bool S7(string str,int index);

bool S1(string str,int index){ //起始态
    if(index>=str.length())//超出界限
        return false;
    if(str[index]>='0'&&str[index]<='9'){ //输入字符为'd'
        index++;
        return S2(str,index);
    }
    if(str[index]=='.'){ //输入字符为'!'
        index++;
        return S3(str,index);
    }
    if(str[index]=='e'){ //输入字符为'e'
        index++;
        return S4(str,index);
    }
    return false;
}
}

```

```

bool S2(string str,int index){ //终态
    if(index>=str.length())//超出界限
        return true;
    if(str[index]>='0'&&str[index]<='9'){ //输入字符为'd'
        index++;
        return S2(str,index);
    }
    if(str[index]=='.'){ //输入字符为'.'
        index++;
        return S3(str,index);
    }
    if(str[index]=='e'){ //输入字符为'e'
        index++;
        return S4(str,index);
    }
    return false;
}

```

```

bool S3(string str,int index){
    if(index>=str.length())//超出界限
        return false;
    if(str[index]>='0'&&str[index]<='9'){ //输入字符为'd'
        index++;
        return S5(str,index);
    }
    return false;
}

```

```

bool S4(string str,int index){

```

```

    if(index>=str.length())//超出界限
        return false;
    if(str[index]>='0'&&str[index]<='9'){ //输入字符为'd'
        index++;
        return S6(str,index);
    }
    if(str[index]=='+'||str[index]=='-'){ //输入字符为's'
        index++;
        return S7(str,index);
    }
    return false;
}

```

```

bool S5(string str,int index){ //终态
    if(index>=str.length())//超出界限
        return true;
    if(str[index]>='0'&&str[index]<='9'){ //输入字符为'd'
        index++;
        return S5(str,index);
    }
    if(str[index]=='e'){ //输入字符为'e'
        index++;
        return S4(str,index);
    }
    return false;
}

```

```

bool S6(string str,int index){ //终态
    if(index>=str.length())//超出界限
        return true;

```

```

        if(str[index]>='0'&&str[index]<='9'){ //输入字符为'd'
            index++;
            return S6(str,index);
        }
        return false;
    }

bool S7(string str,int index){
    if(index>=str.length())//超出界限
        return false;
    if(str[index]>='0'&&str[index]<='9'){ //输入字符为'd'
        index++;
        return S6(str,index);
    }
    return false;
}

//是否是常数
bool isChangShu(string dealStr){
    //return false;
    return S1(dealStr,0);
}

void cifa(){
    while(index<str.length()){

        bool isDealed=false; //记录是否处理过

        while(str[index]==' '||str[index]=='9'||str[index]=='10'||str[index]=='14'){ //过滤

```



空格,Tab,换行符

```
        index++;

        if(index>=str.length()) //扫描完后返回
            return;
    }

    if(str[index]=='/'){//去掉注释
        if(index+1<str.length()&&str[index+1]=='*'){//去掉 /**/注释
            index+=2;
            while(true){
                if(index+1>=str.length()) //扫描完后返回
                    return;
                if(str[index]=='*'&&str[index+1]=='/') //结束符号 */
                    break;
                index+=1; //扫描下面两个字符
            }
            isDealed=true;
            index+=2;
        }else if(index+1<str.length()&&str[index+1]=='/'){//去掉 //注释
            index+=2;
            while(true){
                if(index>=str.length()) //扫描完后返回
                    return;
                if(str[index]==10){ //换行符
                    //index++; //去掉 10 和 14
                    break;
                }
                index++; //扫描下面一个字符
            }
        }
    }
```

```

        isDealed=true;

        index+=1;
    }
}

while(str[index]==' '||str[index]==9||str[index]==10||str[index]==14){ //过滤
空格,Tab,换行符

    index++;

    if(index>=str.length()) //扫描完后返回

        return;

}

int srt_index=0;
//是否是界符
int index1=getJieFu();
if(index1!=-1){//找到界符

    string tmp="";

    for(srt_index=index;srt_index<=index1;++srt_index)

        tmp+=str[srt_index];

    cout<<"(界符,"<<tmp<<)"<<endl;

    out<<token_index++<<" "<<5<<" "<<tmp<<endl;

    isDealed=true;

    index=index1+1;

}

//是否是运算符

index1=getYunSuanfFu());

```

```

if(index1!=-1){//找到运算符
    string tmp="";
    for(srt_index=index;srt_index<=index1;++srt_index)
        tmp+=str[srt_index];
    if(!(index1+1<str.length()&&(tmp=="/"&&str[index+1]=='/'))){//如果不是//
        cout<<"(运算符,"<<tmp<<")"<<endl;
        out<<token_index++<<" "<<4<<" "<<tmp<<endl;
        index=index1+1;
    }
    isDealed=true;
}

if(index>=str.length()) //扫描完后返回
    return;

if(str[index]>='a'&&str[index]<='z'){//是否是标识符、关键字
    string dealStr="";//要处理的字符串
    dealStr+=str[index];//加入待处理的串
    index++;
    while(index<str.length()){
        if(str[index]==' '||str[index]==9||str[index]==10||str[index]==14||
            getJieFu()!=-1||getYunSuanFu()!=-1)
            break;//遇到分隔符
        else {
            dealStr+=str[index];//加入待处理的串
            index++;
        }
    }
}

```

```

        if(isBiaozhifu(dealStr,0)){ //是否是标识符

            if(isGuanJiZi(dealStr)){ //查询关键字表，判断是否是关键字

                cout<<"(关键字,"<<dealStr<<)"<<endl;

                out<<token_index++<<" "<<1<<" "<<dealStr<<endl;

            }else{

                cout<<"(标识符,"<<dealStr<<)"<<endl;

                out<<token_index++<<" "<<2<<" "<<dealStr<<endl;

            }

        }else{

            cout<<"非法标识符："<<dealStr<<endl;

            out<<token_index++<<" "<<0<<" "<<dealStr<<endl;

        }

    }else if((str[index]>='0'&&str[index]<='9')||str[index]=='.'||str[index]=='e'){ //
    是否是无符号常数

        string dealStr="";//要处理的字符串

        dealStr+=str[index];//加入待处理的串

        index++;

        while(index<str.length()){

            if(str[index]==' '||str[index]==9||str[index]==10||str[index]==14||

                getJieFu()!=-1||getYunSuanFu()!=-1){

                if((str[index]=='+'||str[index]=='-')&&

                    str[index-1]=='e'&&str[index-2]>='0'&&str[index-2]<='9'){//3.14e+5 防止+被当
                    做加减

                        dealStr+=str[index];//加入待处理的串

                        index++;

                    }else

                        break;//遇到分隔符

                }else {

```

```
dealStr+=str[index]; //加入待处理的串
index++;
}
}
if(isChangShu(dealStr)){ //是否是常数
    cout<<"(常数,"<<dealStr<<)"<<endl;
    out<<token_index++<<" "<<3<<" "<<dealStr<<endl;
}else{
    cout<<"非法常数： "<<dealStr<<endl;
    out<<token_index++<<" "<<0<<" "<<dealStr<<endl;
}

}else{//其他未定义
    if(!isDealed){
        cout<<"非法字符： "<<str[index]<<endl;
        out<<token_index++<<" "<<0<<" "<<str[index]<<endl;
        index++;
    }
}

}

}while 循环
}

int main(){
    ifstream cin("in.txt");
    string tmp;
    while(getline(cin,tmp)){
        str+=tmp+'\n';
    }
    cifa();
```

```
system("pause");  
return 0;  
}
```

## 附录 8：程序源代码——yufa.cpp

```
#include<iostream>  
#include<fstream>  
using namespace std;  
#include "test2.cpp"  
  
int main(){  
    ifstream cin("token.txt");  
    init();//初始化  
    //ofstream cout("result.txt");  
  
    int type;  
    string val;  
    bool isFit=false;  
    string dealStr="";  
    bool findfh=false;  
    int token_index;  
    string out_str="";  
    string last_str="";  
    while(cin>>token_index>>type>>val){  
        last_str=val;  
        findfh=false;  
        isFit=false;  
        out_str+=val+" ";  
        //cout<<val<<endl;
```

```

//以分号为单位处理串

if(type==0){//错误串

    cout<<"token.txt 中第"<<token_index<<"行存在错误串"<<endl;

    isFit=false;

    break;

}else{//非错误串

    if(val==";"){

        findfh=true;

        isFit=Yufa(dealStr);

        if(!isFit){

            cout<<"处理的串出错： "<<out_str<<endl;

            break;

        }

        out_str="";

        dealStr="";//清空

    }else{

        bool find=false;

        if(type==1){//关键字

            if(val=="int"||val=="float"||val=="double"){//a 表示变量类型

                dealStr+='a';

                find=true;

            }

            if(val=="if"||val=="while"){//g 表示 if, while

                dealStr+='g';

                find=true;

            }

        }

    }

    if(type==2){//标识符

        dealStr+='b';
    }
}

```

```

        find=true;
    }

    if(type==3){//常数
        dealStr+='e';
        find=true;
    }

    if(type==4){//运算符符
        if(val=="+"||val=="-"||val=="*"||val=="/"||val=="%"){//d 表示
运算符
            dealStr+='d';
            find=true;
        }
        if(val=="<"||val==">"||val=="<="||val==">="){//f 表示关系运
算符
            dealStr+='f';
            find=true;
        }
    }

    if(val=="="||val=="("||val=="){
        dealStr+=val;
        find=true;
    }

    if(!find){
        cout<<"精简语法不支持 token 串: "<<val<<endl;
        isFit=false;
    }

```



```
                break;
            }
        }
    }//非错误串
} //while

if(isFit)
    cout<<"符合语法！"<<endl;

if(last_str!=";")
    cout<<"结尾缺少分号！"<<endl;

system("pause");

return 0;
}
```