编译器专题实验报告

**实验二：词法分析（独立模式）**

**一、实验目的**

构建词法分析程序能将源语言程序作为输入，并输出词法记号串到文件中。

包含功能：

（1）单词设计包含主文法中所有词法单位；

（2）一遍扫描或含预处理遍，能删除注解，并允许空白字符串作为分隔；

（3）分析框架scanner()允许连续调用直到输入串被扫描完毕，每调用一次返回一个记号；

（4）有联合DFA设计结果以及超前搜索功能；

（5）（可选）词法错误处理。

编写某语言的词法分析器的源程序并调试通过。

**二、实验内容（必做）**

**1.实验要求**

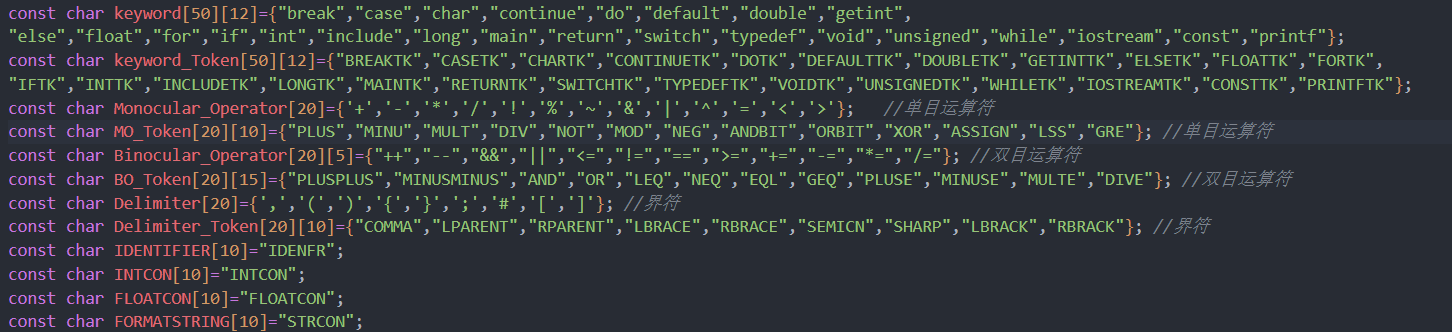
根据给定的文法设计并实现词法分析程序，从源程序中识别出单词，记录其单词类别和单词值，输入输出及处理要求如下：

（1）单词类别码 单词的字符/字符串形式(中间仅用一个空格间隔)。

（2）单词的类别码统一按如下形式定义：

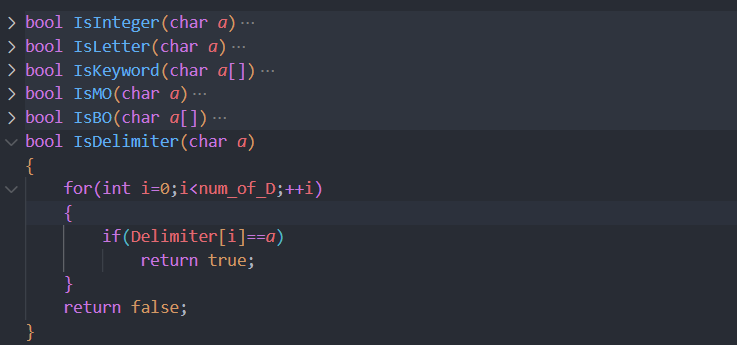
| **单词名称** | **类别码** | **单词名称** | **类别码** | **单词名称** | **类别码** | **单词名称** | **类别码** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Ident** | IDENFR | ! | NOT | \* | MULT | = | ASSIGN |
| **IntConst** | INTCON | && | AND | / | DIV | ; | SEMICN |
| **FormatString** | STRCON | || | OR | % | MOD | , | COMMA |
| main | MAINTK | while | WHILETK | < | LSS | ( | LPARENT |
| const | CONSTTK | getint | GETINTTK | <= | LEQ | ) | RPARENT |
| int | INTTK | printf | PRINTFTK | > | GRE | [ | LBRACK |
| break | BREAKTK | return | RETURNTK | >= | GEQ | ] | RBRACK |
| continue | CONTINUETK | + | PLUS | == | EQL | { | LBRACE |
| if | IFTK | - | MINU | != | NEQ | } | RBRACE |
| else | ELSETK | void | VOIDTK |  |  |  |  |

**2.定义关键字及Token**



本实验中使用到的关键字及其对应的Token如图所示，对于保留字、单目多目运算符等含有多种情况的关键字，我们使用一个二维数组进行存储，在存储关键字之后，我们将关键字对应的Token也用对应的二维数组进行存储。对于ID等可变的项目，这里我们先只存储其Token。

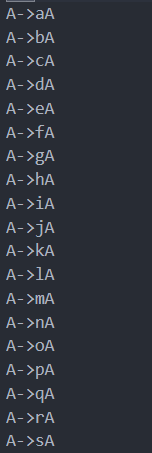
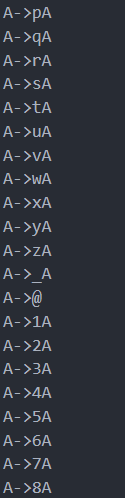
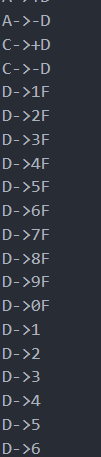
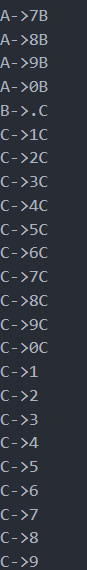
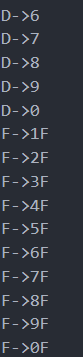
**3.定义基本函数**



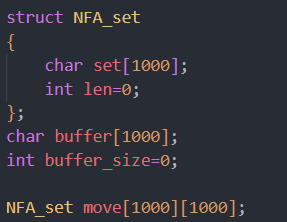
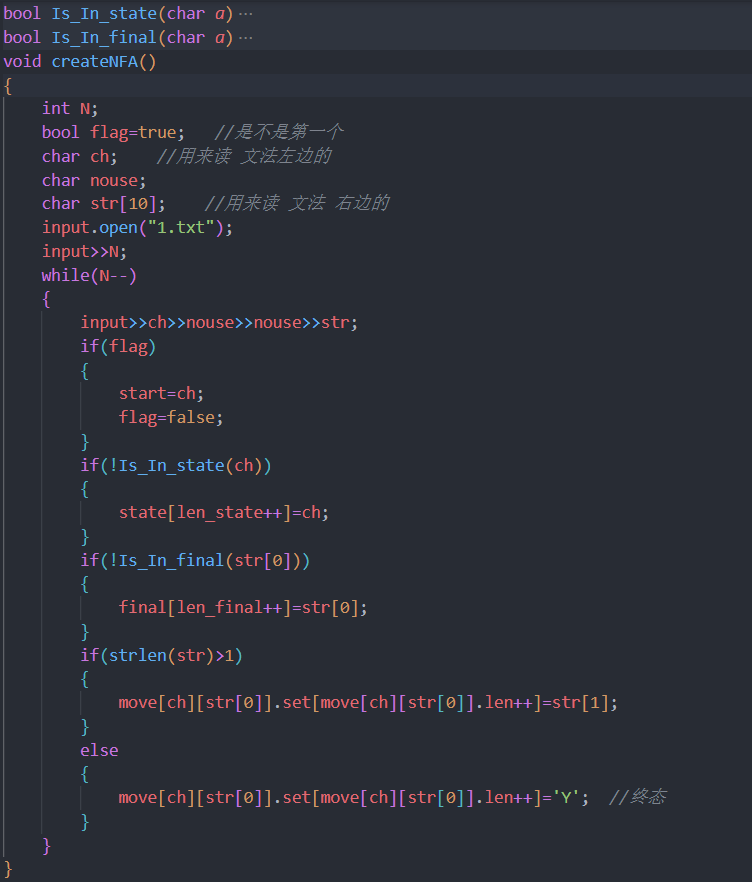
这里我们定义了一些后续需要使用的函数，包括判断输入的字符是否为数字，是否为字母，是否为单目运算符，是否为分隔符，输入的字符串是否为双目运算符，是否为保留字等。以判断是否为分隔符为例，这里就是在第一步中的数组中遍历，判断是否为其中的一项以判断是否为分隔符。

**4.自定义NFA**

对于上面的关键字，进行匹配判断的逻辑较为简单，只需要读入文字，而后判断文字是否属于其中的关键字即可，但是对于ID，数字（包括整数、浮点数）等需要进行判断就不像这么容易，因为ID，数字类型的具体形式不固定，所以我们这里打算使用NFA进行ID和数字的判断，程序员在一个文件中写好NFA的规则，词法分析程序读入其规则并将其转化为DFA，如果读入的字符串能够被DFA所接受，那么就判断该字符串为ID或者数字，我们使用A->bC表示A状态接收到b字符后变为了C状态，这里词法分析器中使用的部分NFA定义如下所示（附录中有全部定义），可以用于接收下划线数字字母组成的ID，整型数据，浮点数字包括科学计数法，一共104条规则。

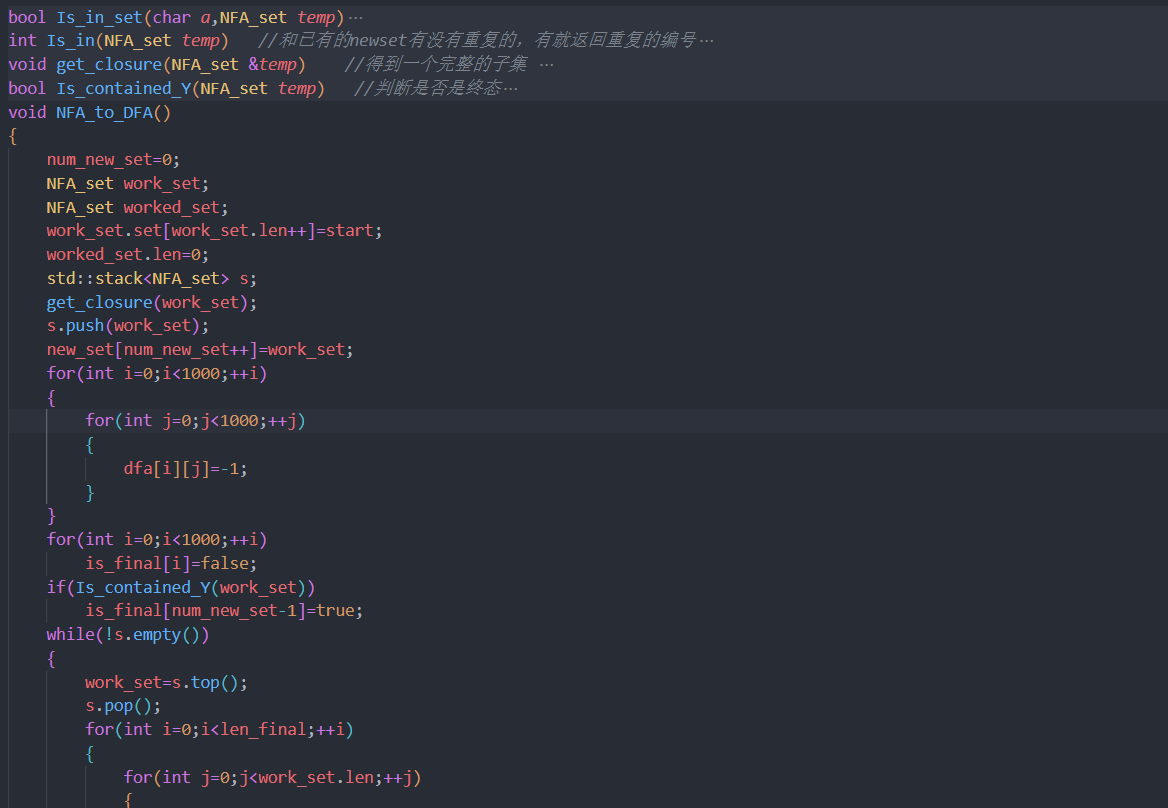
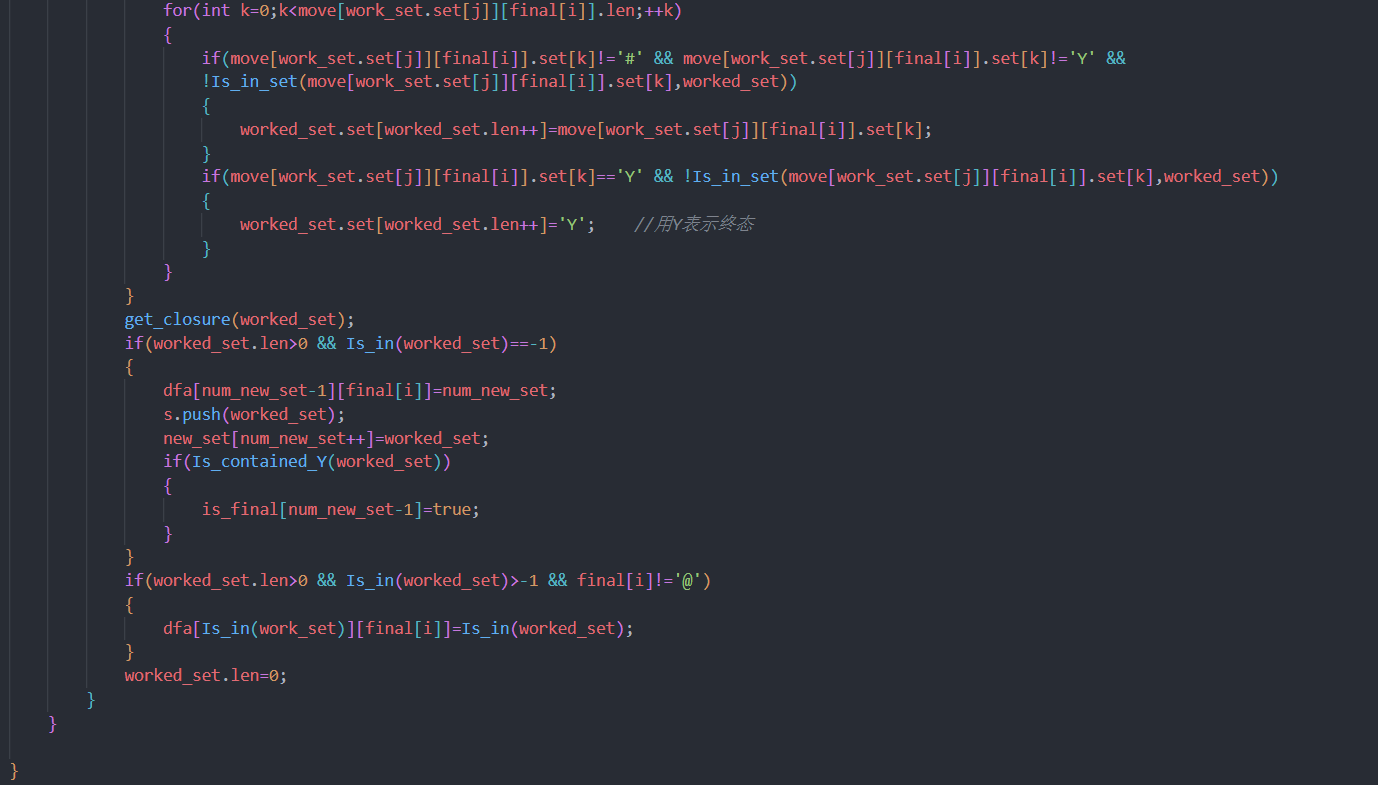
   

代码中使用如下代码段进行NFA的定义，读入和创建，每次读入规则都会判断读入的状态之前是否出现，是否为新的状态，并根据输入规则判断状态是否为终态。

**5.NFA转DFA**

读入并创建上面的NFA之后，需要将其转化为DFA后才能用于输入字符串的类型判断，在代码中，我们使用以下三个函数进行NFA到DFA的转化。

Is\_in\_set函数检查字符a是否在NFA中。如果在，返回true，否则返回false。Is\_in函数检查NFA是否已经存在于new\_set中。如果存在，返回对应的索引，否则返回-1。get\_closure函数用于获取NFA的闭包。Is\_contained\_Y函数检查NFA是否包含终态Y。如果包含，返回true，否则返回false。

主函数NFA\_to\_DFA首先初始化了一些变量，然后将起始状态添加到工作集work\_set中，并获取其闭包。然后，它将work\_set添加到new\_set中，并将其推入栈s中。接下来，初始化了DFA的转移表dfa和终态标记数组is\_final。如果work\_set包含终态Y，则将其标记为终态。然后，进入一个循环，直到栈s为空。在每次循环中，它从栈顶取出一个工作集，然后对于每个可能的输入符号，它计算出通过该符号可以到达的所有状态，并将其添加到worked\_set中。然后，获取worked\_set的闭包。如果worked\_set不为空，并且不在new\_set中，那么它将worked\_set添加到new\_set中，并将其推入栈s中。同时，它更新DFA的转移表dfa。如果worked\_set包含终态Y，则将其标记为终态。如果worked\_set不为空，并且在new\_set中，那么它只更新DFA的转移表dfa。最后，它清空worked\_set，准备下一次循环。通过这个过程可以得到DFA。

**6.输出处理**

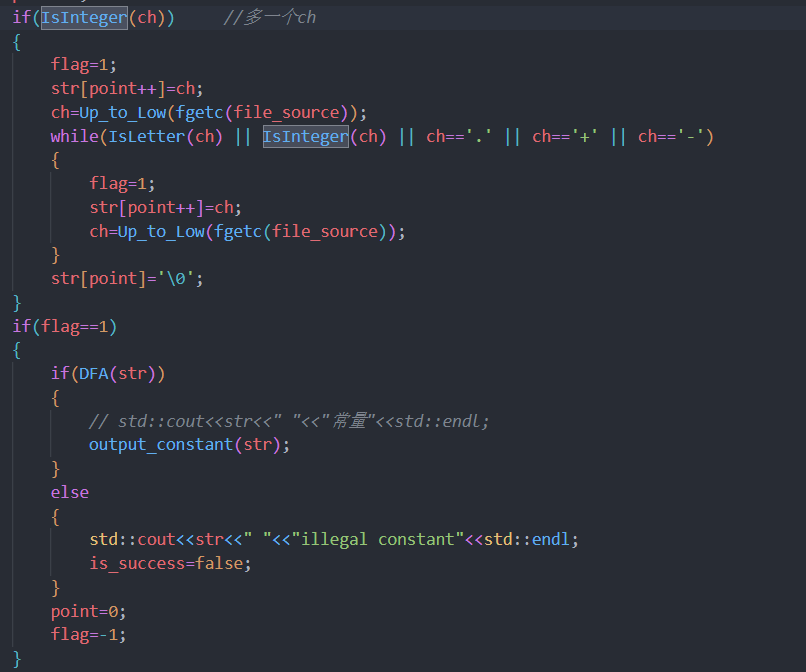


用于输出处理的函数定义如上图所示，根据不同类型（数字，保留字，ID等）进行不同的输出操作。以输出数字常量为例，当判断出读入的字符串为数字（整型或浮点型）时，根据不同情况，输出整型或者浮点型的Token并输出其对应的数字。其他类型的输出操作类似，根据不同的关键字进行不同的输出。

**7.Scanner**

将以上编写的函数进行汇总可以编写为Scanner进行词法分析，Scanner有如下几个部分（这里我们不区分字母大小写，大写字母转为小写分析）。

（1）数字处理



如果发现读入的字符为数字，则认为后续可能会继续读入数字，于是开启超前搜索，往后继续读，直到无法读入数字的组成部分。随后使用DFA判断读入的数字是否满足要求，能被DFA接收，可以被接受则写入文件并输出。

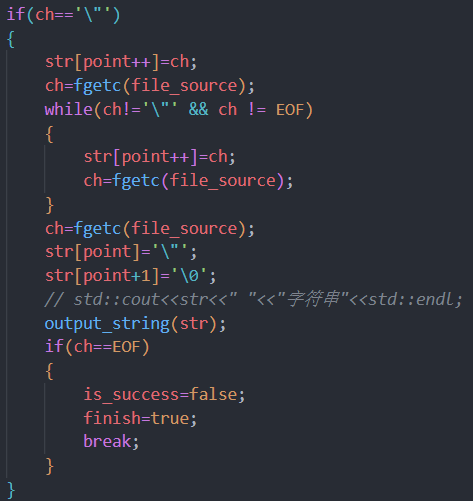
（2）ID处理

和处理数字类似也可以用超前搜索和DFA判断ID是否能被接受，只不过在判断是否为ID之前要先判断是否为保留字。



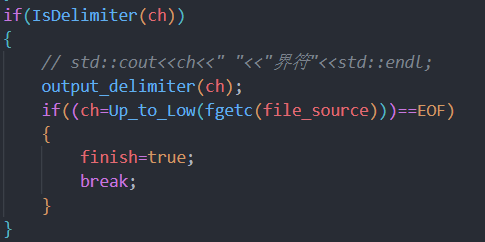
（3）字符串处理

这里认为两个引号之间的内容均为字符串的内容，于是发现读入引号时即认为开始读入字符串，直到读到另一个引号认为字符串结束。



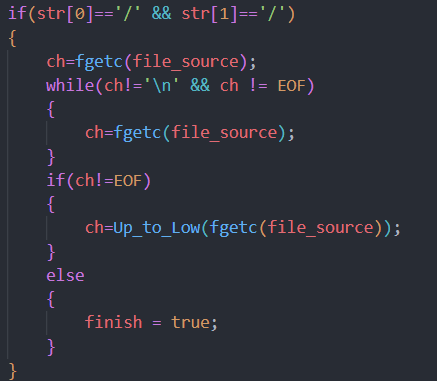
（4）分隔符处理

分隔符处理比较简单，就是读入分隔符时直接根据不同类型输出即可。

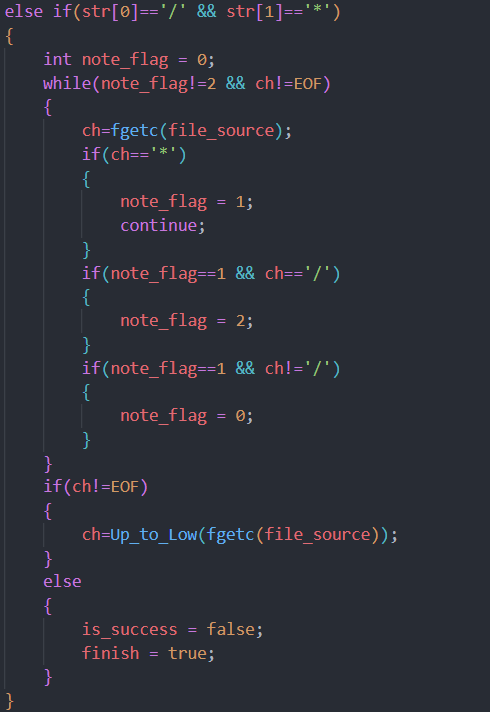


（5）单目双目运算符处理及注释处理

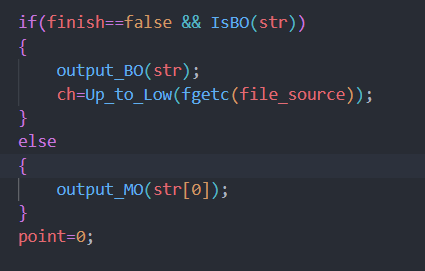
把单目双目运算符处理和注释处理放在一起是因为“/”是一个单目运算符同时也是注释开始的第一个字符，也就是说读入“/”时要判断是读入了运算符还是将要读入注释了。当读入“/”时会超前搜索一个字符，并首先开始注释判断，首先是//的注释，发现为注释则读到这行结束。



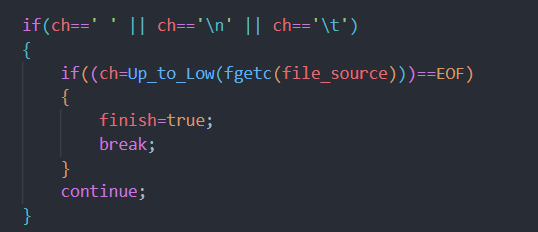
其次为/\*\*/的注释，发现/\*之后读到出现\*/结束，如果到最后都没有出现\*/则认为词法分析出错了。



最后才是单目和双目运算符的处理，根据读入内容进行运算符的匹配操作并进行对应的输出。



（6）其他字符的处理



当读入换行空格回车等操作时不需要进行额外操作，需要判断文件是否已经读完。

**8.实验结果分析**

本实验使用简单的C语言程序进行验证，程序代码中包含注释，代码如下所示：

const int array[2] = {1,2};

int main(){

int c;

int b = 1;

/\*

123456

\*/

c = getint();

printf("output is %d",c);

if (c > 1 || b>1)

{

printf("output is %d",c);

}

if (c > 1 && b>1)

{

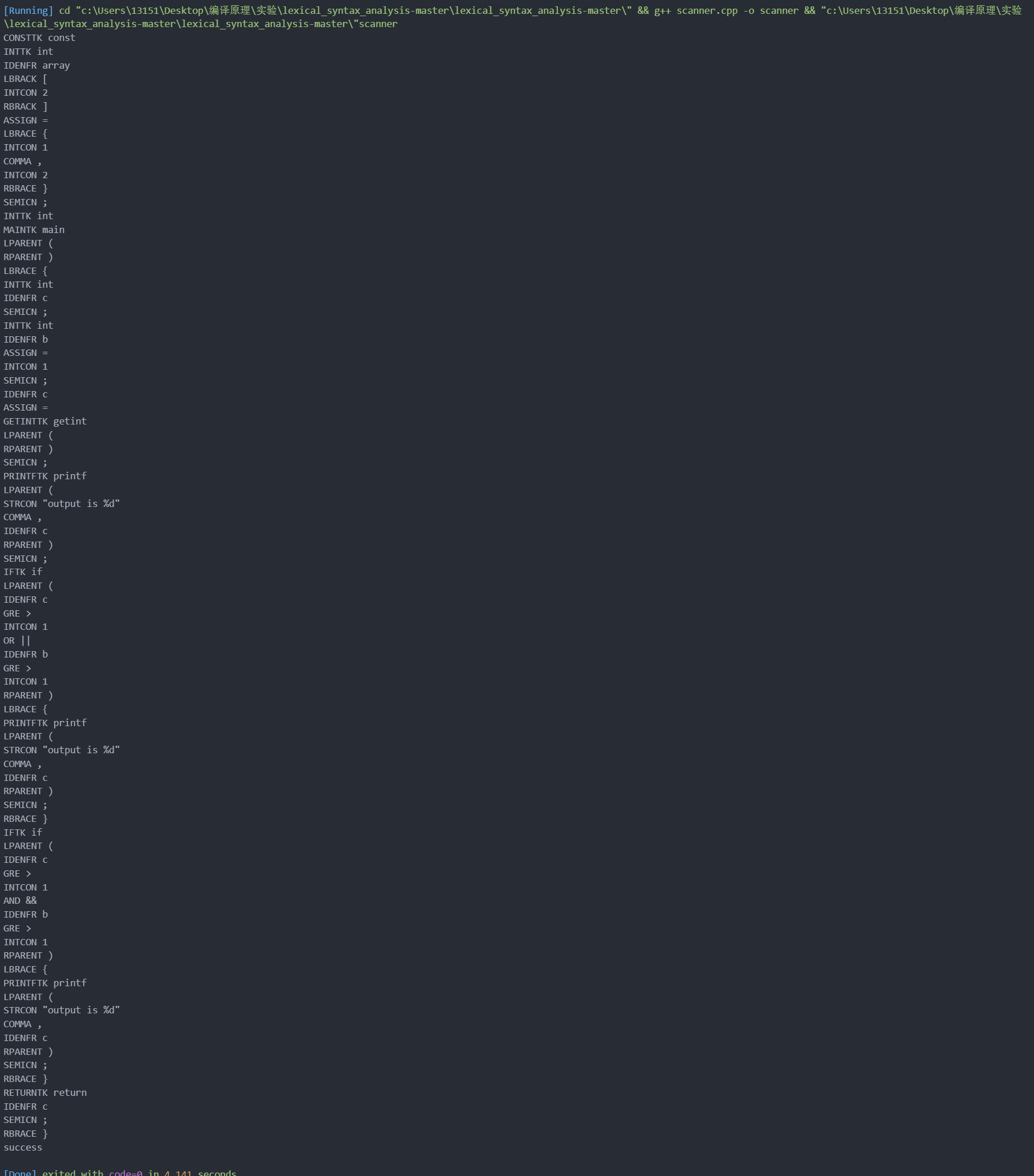
printf("output is %d",c);

}//123456

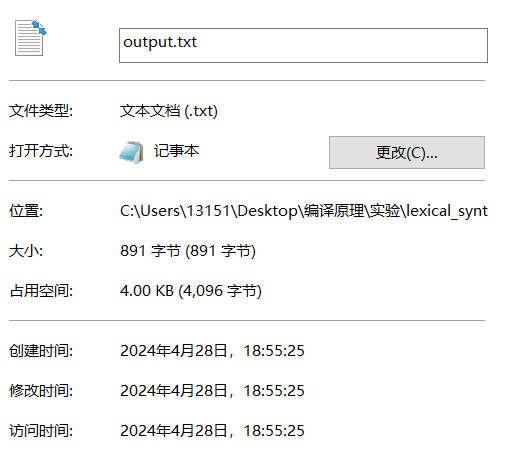
return c;

}

运行该词法分析文件后可以得到词法分析结果并输出在output.txt中。



最后输出success说明分析成功，同时，output.txt中也会输出相同的内容，这里不重复放了，这里用output.txt的修改时间作为实验的完成时间。

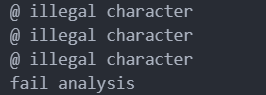


**三、实验内容（选做）**

**错误检查：** 在上一节的Scanner中，我们对于无法用DFA识别的数字进行了报错处理，对无法用DFA输出别的且不属于保留字的字符串进行了报错处理，代码中也有对应的体现（输出illegal constant和illegal identity）。

除此之外，对于上一节描述的Scanner无法识别的字符，即上一节Scanner的六种情况之外的情况也要进行错误处理。思路如下：每读入一个字符，设置其处理flag为false，如果该字符能被上述Scanner六种情况之一进行处理，则将flag置为true，如果最后发现flag为false则报错，出现了illegal character。

在上述测试代码最后加上@@@字符串进行词法分析，输出的部分结果如下：



说明读入了非法字符，最后输出的fail analysis指的是词法分析有报错。如果出现了illegal constant/identity/character，则会出现词法分析报错。

**四、遇到的问题**

1. 不知道NFA如何转化为DFA，通过查阅CSDN，Github知道了变化过程，并研究了相应代码。

**五、附件：代码**

NFA规则：

104

A->aA

A->bA

A->cA

A->dA

A->eA

A->fA

A->gA

A->hA

A->iA

A->jA

A->kA

A->lA

A->mA

A->nA

A->oA

A->pA

A->qA

A->rA

A->sA

A->tA

A->uA

A->vA

A->wA

A->xA

A->yA

A->zA

A->\_A

A->@

A->1A

A->2A

A->3A

A->4A

A->5A

A->6A

A->7A

A->8A

A->9A

A->0A

A->1B

A->2B

A->3B

A->4B

A->5B

A->6B

A->7B

A->8B

A->9B

A->0B

B->.C

C->1C

C->2C

C->3C

C->4C

C->5C

C->6C

C->7C

C->8C

C->9C

C->0C

C->1

C->2

C->3

C->4

C->5

C->6

C->7

C->8

C->9

C->0

C->eA

A->+D

A->-D

C->+D

C->-D

D->1F

D->2F

D->3F

D->4F

D->5F

D->6F

D->7F

D->8F

D->9F

D->0F

D->1

D->2

D->3

D->4

D->5

D->6

D->7

D->8

D->9

D->0

F->1F

F->2F

F->3F

F->4F

F->5F

F->6F

F->7F

F->8F

F->9F

F->0F

代码：

#include <iostream>

#include <cstring>

#include <cstdio>

#include <algorithm>

#include <stack>

#include <fstream>

const int num\_of\_keyword=25;

const int num\_of\_MO=13;

const int num\_of\_BO=12;

const int num\_of\_D=9;

const char keyword[50][12]={"break","case","char","continue","do","default","double","getint",

"else","float","for","if","int","include","long","main","return","switch","typedef","void","unsigned","while","iostream","const","printf"};

const char keyword\_Token[50][12]={"BREAKTK","CASETK","CHARTK","CONTINUETK","DOTK","DEFAULTTK","DOUBLETK","GETINTTK","ELSETK","FLOATTK","FORTK",

"IFTK","INTTK","INCLUDETK","LONGTK","MAINTK","RETURNTK","SWITCHTK","TYPEDEFTK","VOIDTK","UNSIGNEDTK","WHILETK","IOSTREAMTK","CONSTTK","PRINTFTK"};

const char Monocular\_Operator[20]={'+','-','\*','/','!','%','~','&','|','^','=','<','>'}; //单目运算符

const char MO\_Token[20][10]={"PLUS","MINU","MULT","DIV","NOT","MOD","NEG","ANDBIT","ORBIT","XOR","ASSIGN","LSS","GRE"}; //单目运算符

const char Binocular\_Operator[20][5]={"++","--","&&","||","<=","!=","==",">=","+=","-=","\*=","/="}; //双目运算符

const char BO\_Token[20][15]={"PLUSPLUS","MINUSMINUS","AND","OR","LEQ","NEQ","EQL","GEQ","PLUSE","MINUSE","MULTE","DIVE"}; //双目运算符

const char Delimiter[20]={',','(',')','{','}',';','#','[',']'}; //界符

const char Delimiter\_Token[20][10]={"COMMA","LPARENT","RPARENT","LBRACE","RBRACE","SEMICN","SHARP","LBRACK","RBRACK"}; //界符

const char IDENTIFIER[10]="IDENFR";

const char INTCON[10]="INTCON";

const char FLOATCON[10]="FLOATCON";

const char FORMATSTRING[10]="STRCON";

FILE\* file\_source=NULL;

std::ifstream input;

std::ofstream output;

char state[1000];

int len\_state;

char start;

char final[1000];

int len\_final;

bool is\_final[1000];

struct NFA\_set

{

char set[1000];

int len=0;

};

char buffer[1000];

int buffer\_size=0;

NFA\_set move[1000][1000];

char N\_state[1000];

int N\_len\_state;

char N\_start;

char N\_final[1000];

int N\_len\_final;

bool N\_is\_final[1000];

NFA\_set new\_set[1000];

int num\_new\_set=0;

int dfa[1000][1000];

bool IsInteger(char a)

{

if(a>='0' && a<='9')

return true;

return false;

}

bool IsLetter(char a)

{

if(a>='a' && a<='z')

return true;

if(a>='A' && a<='Z')

return true;

return false;

}

bool IsKeyword(char a[])

{

int len=strlen(a);

for(int j=0;j<num\_of\_keyword;++j)

{

if(strlen(keyword[j])==len)

{

if(strcmp(keyword[j],a)==0)

return true;

}

}

return false;

}

bool IsMO(char a)

{

for(int i=0;i<num\_of\_MO;++i)

{

if(Monocular\_Operator[i]==a)

return true;

}

return false;

}

bool IsBO(char a[])

{

for(int i=0;i<num\_of\_BO;++i)

{

if(strcmp(Binocular\_Operator[i],a)==0)

return true;

}

return false;

}

bool IsDelimiter(char a)

{

for(int i=0;i<num\_of\_D;++i)

{

if(Delimiter[i]==a)

return true;

}

return false;

}

bool Is\_In\_state(char a)

{

for(int i=0;i<len\_state;++i)

{

if(a==state[i])

return true;

}

return false;

}

bool Is\_In\_final(char a)

{

for(int i=0;i<len\_final;++i)

{

if(a==final[i])

return true;

}

return false;

}

void createNFA()

{

int N;

bool flag=true; //是不是第一个

char ch; //用来读 文法左边的

char nouse;

char str[10]; //用来读 文法 右边的

input.open("1.txt");

input>>N;

while(N--)

{

input>>ch>>nouse>>nouse>>str;

if(flag)

{

start=ch;

flag=false;

}

if(!Is\_In\_state(ch))

{

state[len\_state++]=ch;

}

if(!Is\_In\_final(str[0]))

{

final[len\_final++]=str[0];

}

if(strlen(str)>1)

{

move[ch][str[0]].set[move[ch][str[0]].len++]=str[1];

}

else

{

move[ch][str[0]].set[move[ch][str[0]].len++]='Y'; //终态

}

}

}

bool Is\_in\_set(char a,NFA\_set temp)

{

for(int i=0;i<temp.len;++i)

{

if(a==temp.set[i])

return true;

}

return false;

}

int Is\_in(NFA\_set temp) //和已有的newset有没有重复的，有就返回重复的编号

{

bool flag[1000];

bool flag1;

for(int i=0;i<temp.len;++i)

{

flag[i]=false;

}

for(int i=0;i<num\_new\_set;++i)

{

for(int k=0;k<temp.len;++k)

{

for(int j=0;j<new\_set[i].len;++j)

{

if(temp.set[k]==new\_set[i].set[j])

{

flag[k]=true;

}

}

}

flag1=true;

for(int m=0;m<temp.len;++m)

{

if(flag[m]==false)

{

flag1=false;

break;

}

}

if(flag1==true)

return i;

for(int m=0;m<temp.len;++m)

{

flag[m]=false;

}

}

return -1;

}

void get\_closure(NFA\_set &temp) //得到一个完整的子集

{

for(int i=0;i<temp.len;++i)

{

for(int j=0;j<move[temp.set[i]]['@'].len;++j)

{

if(!Is\_in\_set(move[temp.set[i]]['@'].set[j],temp))

{

temp.set[temp.len++]=move[temp.set[i]]['@'].set[j];

}

}

}

}

bool Is\_contained\_Y(NFA\_set temp) //判断是否是终态

{

for(int i=0;i<temp.len;++i)

{

if(temp.set[i]=='Y')

return true;

}

return false;

}

void NFA\_to\_DFA()

{

num\_new\_set=0;

NFA\_set work\_set;

NFA\_set worked\_set;

work\_set.set[work\_set.len++]=start;

worked\_set.len=0;

std::stack<NFA\_set> s;

get\_closure(work\_set);

s.push(work\_set);

new\_set[num\_new\_set++]=work\_set;

for(int i=0;i<1000;++i)

{

for(int j=0;j<1000;++j)

{

dfa[i][j]=-1;

}

}

for(int i=0;i<1000;++i)

is\_final[i]=false;

if(Is\_contained\_Y(work\_set))

is\_final[num\_new\_set-1]=true;

while(!s.empty())

{

work\_set=s.top();

s.pop();

for(int i=0;i<len\_final;++i)

{

for(int j=0;j<work\_set.len;++j)

{

for(int k=0;k<move[work\_set.set[j]][final[i]].len;++k)

{

if(move[work\_set.set[j]][final[i]].set[k]!='#' && move[work\_set.set[j]][final[i]].set[k]!='Y' &&

!Is\_in\_set(move[work\_set.set[j]][final[i]].set[k],worked\_set))

{

worked\_set.set[worked\_set.len++]=move[work\_set.set[j]][final[i]].set[k];

}

if(move[work\_set.set[j]][final[i]].set[k]=='Y' && !Is\_in\_set(move[work\_set.set[j]][final[i]].set[k],worked\_set))

{

worked\_set.set[worked\_set.len++]='Y'; //用Y表示终态

}

}

}

get\_closure(worked\_set);

if(worked\_set.len>0 && Is\_in(worked\_set)==-1)

{

dfa[num\_new\_set-1][final[i]]=num\_new\_set;

s.push(worked\_set);

new\_set[num\_new\_set++]=worked\_set;

if(Is\_contained\_Y(worked\_set))

{

is\_final[num\_new\_set-1]=true;

}

}

if(worked\_set.len>0 && Is\_in(worked\_set)>-1 && final[i]!='@')

{

dfa[Is\_in(work\_set)][final[i]]=Is\_in(worked\_set);

}

worked\_set.len=0;

}

}

}

bool DFA(char str[])

{

char now\_state=0;

for(int i=0;i<strlen(str);++i )

{

now\_state=dfa[now\_state][str[i]];

if(now\_state==-1)

return false;

}

if(is\_final[now\_state]==true)

return true;

return false;

}

char Up\_to\_Low(char ch) // 不区分大小写，如果是大写转变为小写

{

if (ch >= 'A' && ch <= 'Z')

{

return ch+32;

}

return ch;

}

void output\_constant(char buffer[])

{

// 如果是整型数字，输出INTTK buffer

bool is\_int = true;

for(int i=0;i<strlen(buffer);++i)

{

if(!IsInteger(buffer[i]))

{

is\_int = false;

break;

}

}

if(is\_int)

{

std::cout<< INTCON <<" "<<buffer<<std::endl;

output<< INTCON <<" "<<buffer<<std::endl;

}

else

{

std::cout<< FLOATCON <<" "<<buffer<<std::endl;

output<< FLOATCON <<" "<<buffer<<std::endl;

}

}

void output\_key(char buffer[])

{

for(int i=0;i<num\_of\_keyword;++i)

{

if(strcmp(buffer,keyword[i])==0)

{

std::cout<<keyword\_Token[i]<<" "<<buffer<<std::endl;

output<<keyword\_Token[i]<<" "<<buffer<<std::endl;

return;

}

}

}

void output\_iden(char buffer[])

{

std::cout<<IDENTIFIER<<" "<<buffer<<std::endl;

output<<IDENTIFIER<<" "<<buffer<<std::endl;

}

void output\_string(char buffer[])

{

std::cout<<FORMATSTRING<<" "<<buffer<<std::endl;

output<<FORMATSTRING<<" "<<buffer<<std::endl;

}

void output\_delimiter(char buffer)

{

for(int i=0;i<num\_of\_D;++i)

{

if(buffer==Delimiter[i])

{

std::cout<<Delimiter\_Token[i]<<" "<<buffer<<std::endl;

output<<Delimiter\_Token[i]<<" "<<buffer<<std::endl;

return;

}

}

}

void output\_BO(char buffer[])

{

for(int i=0;i<num\_of\_BO;++i)

{

if(strcmp(buffer,Binocular\_Operator[i])==0)

{

std::cout<<BO\_Token[i]<<" "<<buffer<<std::endl;

output<<BO\_Token[i]<<" "<<buffer<<std::endl;

return;

}

}

}

void output\_MO(char buffer)

{

for(int i=0;i<num\_of\_MO;++i)

{

if(buffer==Monocular\_Operator[i])

{

std::cout<<MO\_Token[i]<<" "<<buffer<<std::endl;

output<<MO\_Token[i]<<" "<<buffer<<std::endl;

return;

}

}

}

void scan()

{

char str[100000];

char ch;

int i,j;

int point;

int flag;

bool is\_success=true;

output.open("output.txt");

ch=Up\_to\_Low(fgetc(file\_source));

bool finish=false;

while(!finish)

{

bool is\_process = false;

flag=-1;

point=0;

if(IsInteger(ch)) //多一个ch

{

is\_process = true;

flag=1;

str[point++]=ch;

ch=Up\_to\_Low(fgetc(file\_source));

while(IsLetter(ch) || IsInteger(ch) || ch=='.' || ch=='+' || ch=='-')

{

flag=1;

str[point++]=ch;

ch=Up\_to\_Low(fgetc(file\_source));

}

str[point]='\0';

}

if(flag==1)

{

if(DFA(str))

{

// std::cout<<str<<" "<<"常量"<<std::endl;

output\_constant(str);

}

else

{

std::cout<<str<<" "<<"illegal constant"<<std::endl;

is\_success=false;

}

point=0;

flag=-1;

}

if(IsLetter(ch))

{

is\_process = true;

flag=2;

str[point++]=ch;

ch=Up\_to\_Low(fgetc(file\_source));

while(IsLetter(ch) || IsInteger(ch))

{

flag=2;

str[point++]=ch;

ch=Up\_to\_Low(fgetc(file\_source));

}

str[point]='\0';

}

if(flag==2)

{

if(IsKeyword(str))

{

// std::cout<<str<<" "<<"关键字"<<std::endl;

output\_key(str);

}

else

{

if(DFA(str))

{

// std::cout<<str<<" "<<"标识符"<<std::endl;

output\_iden(str);

}

else

{

std::cout<<str<<" "<<"illegal identity"<<std::endl;

is\_success=false;

}

}

point=0;

flag=-1;

}

if(ch=='\"')

{

is\_process = true;

str[point++]=ch;

ch=fgetc(file\_source);

while(ch!='\"' && ch != EOF)

{

str[point++]=ch;

ch=fgetc(file\_source);

}

ch=fgetc(file\_source);

str[point]='\"';

str[point+1]='\0';

// std::cout<<str<<" "<<"字符串"<<std::endl;

output\_string(str);

if(ch==EOF)

{

is\_success=false;

finish=true;

break;

}

}

if(IsDelimiter(ch))

{

is\_process = true;

// std::cout<<ch<<" "<<"界符"<<std::endl;

output\_delimiter(ch);

if((ch=Up\_to\_Low(fgetc(file\_source)))==EOF)

{

finish=true;

break;

}

}

if(IsMO(ch))

{

is\_process = true;

str[point++]=ch;

if((ch=Up\_to\_Low(fgetc(file\_source)))==EOF)

{

finish=true;

}

str[point++]=ch;

str[point]='\0';

// 注释处理

if(str[0]=='/' && str[1]=='/')

{

ch=fgetc(file\_source);

while(ch!='\n' && ch != EOF)

{

ch=fgetc(file\_source);

}

if(ch!=EOF)

{

ch=Up\_to\_Low(fgetc(file\_source));

}

else

{

finish = true;

}

}

else if(str[0]=='/' && str[1]=='\*')

{

int note\_flag = 0;

while(note\_flag!=2 && ch!=EOF)

{

ch=fgetc(file\_source);

if(ch=='\*')

{

note\_flag = 1;

continue;

}

if(note\_flag==1 && ch=='/')

{

note\_flag = 2;

}

if(note\_flag==1 && ch!='/')

{

note\_flag = 0;

}

}

if(ch!=EOF)

{

ch=Up\_to\_Low(fgetc(file\_source));

}

else

{

is\_success = false;

finish = true;

}

}

else

{

if(finish==false && IsBO(str))

{

output\_BO(str);

ch=Up\_to\_Low(fgetc(file\_source));

}

else

{

output\_MO(str[0]);

}

point=0;

}

}

if(ch==' ' || ch=='\n' || ch=='\t')

{

is\_process = true;

if((ch=Up\_to\_Low(fgetc(file\_source)))==EOF)

{

finish=true;

break;

}

continue;

}

if(is\_process == false)

{

std::cout<<ch<<" "<<"illegal character"<<std::endl;

is\_success=false;

ch=Up\_to\_Low(fgetc(file\_source));

}

}

output.close();

if(is\_success)

{

std::cout<<"success analysis"<<std::endl;

}

else

{

std::cout<<"fail analysis"<<std::endl;

}

}

void init()

{

len\_final=0;

len\_state=0;

for(int i=0;i<1000;++i)

{

//is\_final[i]=false;

for(int j=0;j<1000;++j)

for(int k=0;k<1000;++k)

move[i][j].set[k]='#';

}

}

int main()

{

init();

len\_final=0;

len\_state=0;

createNFA();

NFA\_to\_DFA();

file\_source=fopen("test.c","r+");

scan();

fclose(file\_source);

return 0;

}