编译原理与技术

词法分析程序设计报告

姓　　名: 刘立敏

学　　号: 2018211398

学　　院: 计算机学院

专　　业: 计算机科学与技术

班　　级: 2018211308

指导老师: 张玉洁

目录

1、实验内容

设计并实现C语言的词法分析程序，要求如下。

1) 可以识别出用C语言编写的源程序中的每个单词符号，并以记号的形式输出每个单词符号。

2) 可以识别并跳过源程序中的注释。

3) 可以统计源程序汇总的语句行数、各类单词个数和字符个数，并输出统计结果。

4) 检查源程序中存在的错误，并可以报告错误所在的位置。

5) 发现源程序中存在的错误后，进行适当的恢复，使词法分析可以继续进行，对源程序进行一次扫描，即可检查并报告出源程序中存在的词法错误。

2、实验环境

编程语言：C++

集成开发环境: Visual Studio 2019, Qt Creator 4.11

3、总体设计

总体设计分为三个模块：统计模块、token识别模块和错误处理模块。

3.1、统计模块

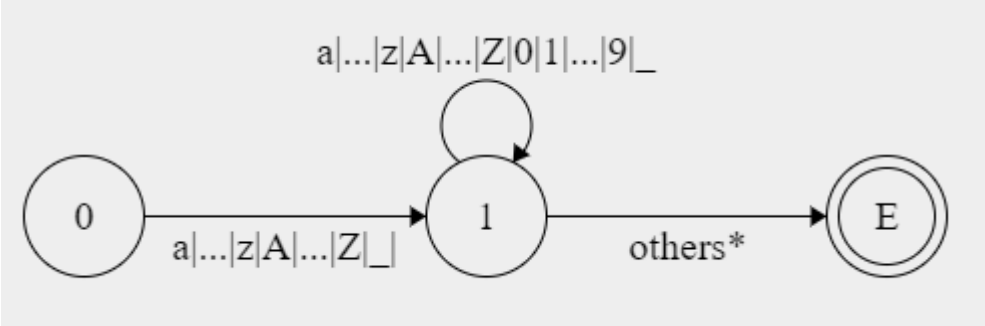
该模块需要统计出源程序的语句行数、单词个数和字符个数，并输出统计结果。该模块的实现较为简单，只需要三个变量来保存所要统计的量，在扫描源程序时即可进行统计。

3.2、token 识别模块

C语言有各种单词记号（token），对于每种token，都有相应的识别方法。具体分析如下：

3.2.1、标识符

C语言的标识符由字母（a-z,A-Z）、数字（0-9）、下划线(_)组成，且首字符必须是字母或下划线。其用DFA表示如下：



注：在图示里，DFA的终态用状态E表示。但在实际代码中，当到达终态表示完成一个token的识别时，我们立即将其转到初态（0），从而进行下一个token的识别。换言之，相当于存在一个终态E到初态0的空转换。

3.2.2、关键字

C语言定义了37个关键字（C99标准）：

auto	break	case	char	const	continue	default	do
double	else	enum	extern	float	for	goto	if
inline	int	long	register	restrict	return	short	signed
sizeof	static	struct	switch	typedef	union	unsigned	void
volatile	while	_Bool	_Complex	_Imaginary			

因为所有的关键字都是标识符，因此不再单独采用DFA方式来识别关键字。我采用的方法是：将这些关键字全部放入一个set中，每次识别出一个标识符时，我们使用set类的count函数来检测其是不是关键字。

3.2.3、运算符

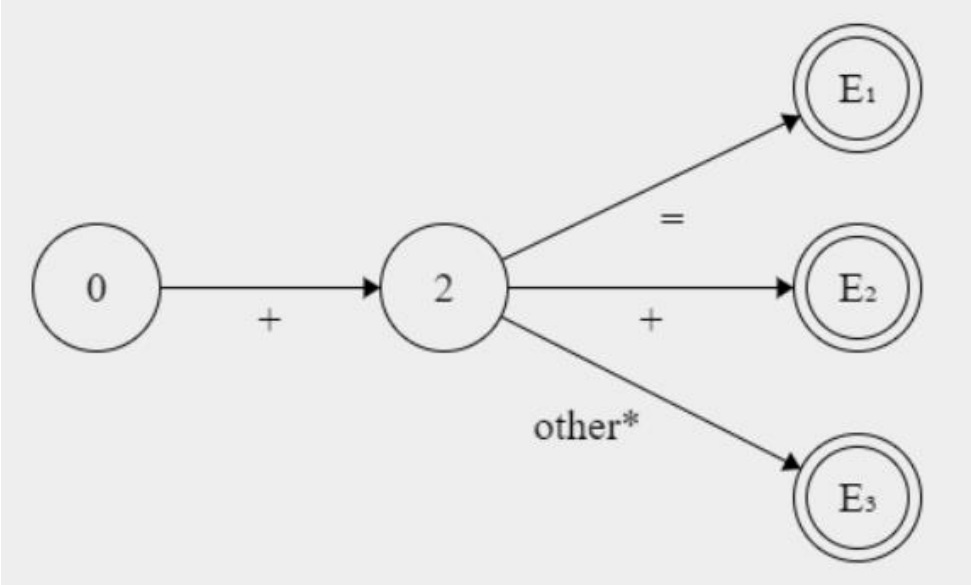
对于运算符的识别，我们采取一符一种方法。

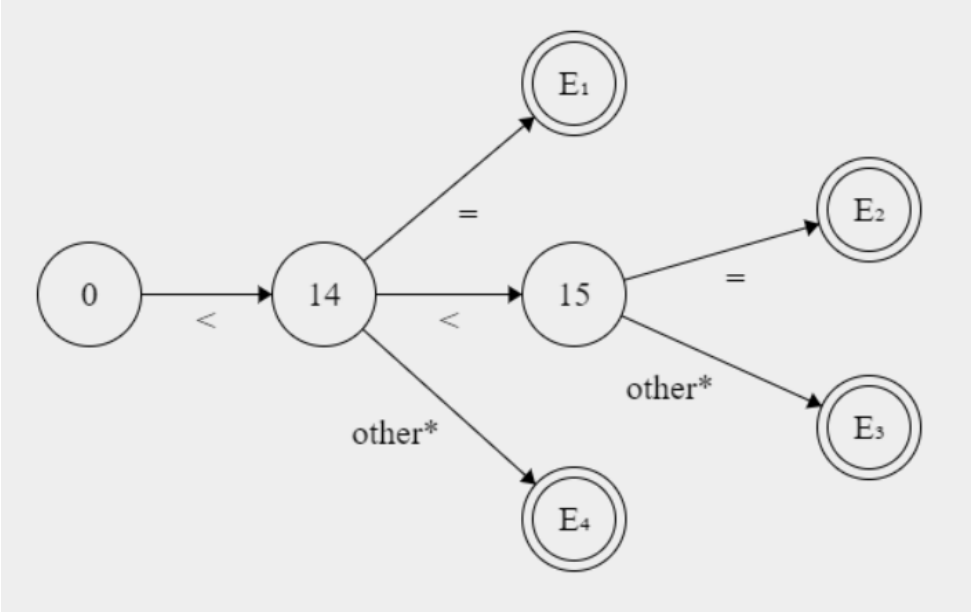
类名	符号	含义

ADD	+	加
SUB	-	减
MUL	*	乘
DIV	/	除
MOD	%	模
INC	++	自增
DEC	--	自减
ASSIGN	=	赋值
ADD_ASSIGN	+=	加赋值
SUB_ASSIGN	-=	减赋值
MUL_ASSIGN	*=	乘赋值
DIV_ASSIGN	/=	除赋值
MOD_ASSIGN	%=	模赋值
BIT_AND_ASSIGN	&=	按位与赋值
BIT_OR_ASSIGN	|=	按位或赋值
BIT_XOR_ASSIGN	^=	按位异或赋值
SHL_ASSIGN	<<=	左移赋值
SHR_ASSIGN	>>=	右移赋值
LESS	<	小于
GREATER	>	大于
LESS_EQUAL	<=	小于等于
GREATER_EQUAL	>=	大于等于
EQUAL	==	等于
NOT_EQUAL	!=	不等于
AND	&&	与
OR	||	或
NOT	!	非
BIT_AND	&	按位与
BIT_OR	|	按位或

BIT_XOR	^	按位异或
BIT_NOT	~	按位取反
SHL	<<	左移
SHR	>>	右移
DOT	.	点
ARROW	->	箭头

由于运算符种类过于繁多，这里不再一一画出识别各运算符的DFA，而仅仅举出几个例子如下所示：





3.2.4、分界符

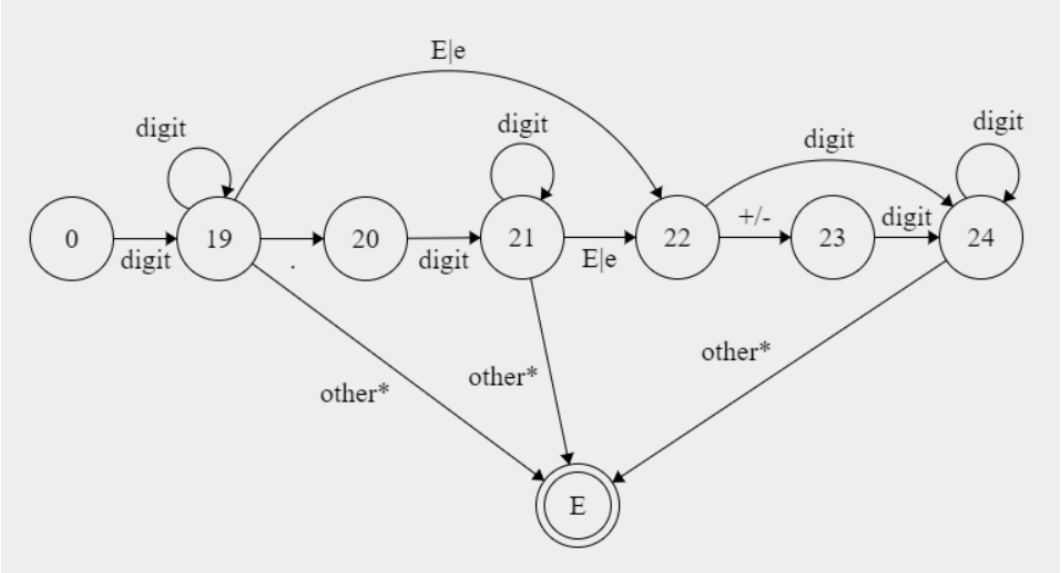
分界符包括括号、分号、冒号、问号（冒号和问号组成三目运算符?:）、逗号等。

对分界符的处理与运算符相似。

类名	符号	含义
L_PARENTHESIS	(	左圆括号
R_PARENTHESIS	)	右圆括号
L_BRACKET	[	左方括号
R_BRACKET	]	右方括号
L_BRACE	{	左花括号
R_BRACE	}	右花括号
SEMICOLON	;	分号
COLON	:	冒号
COMMA	,	逗号
QUESTION	?	问号

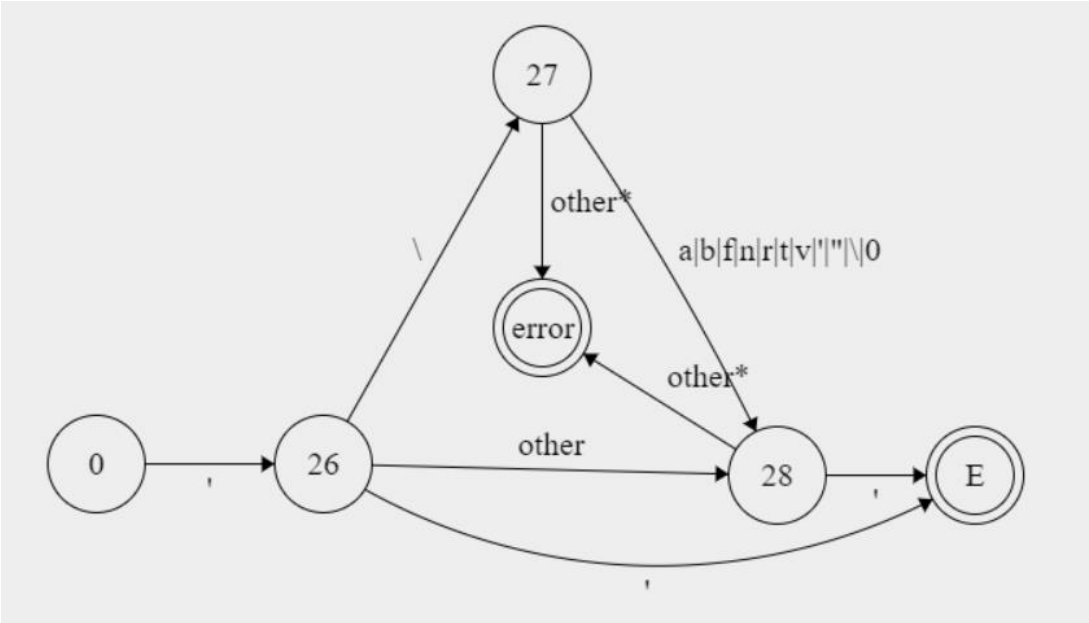
3.2.5、数字常量

识别数字常量的DFA如下



3.2.6、字符常量

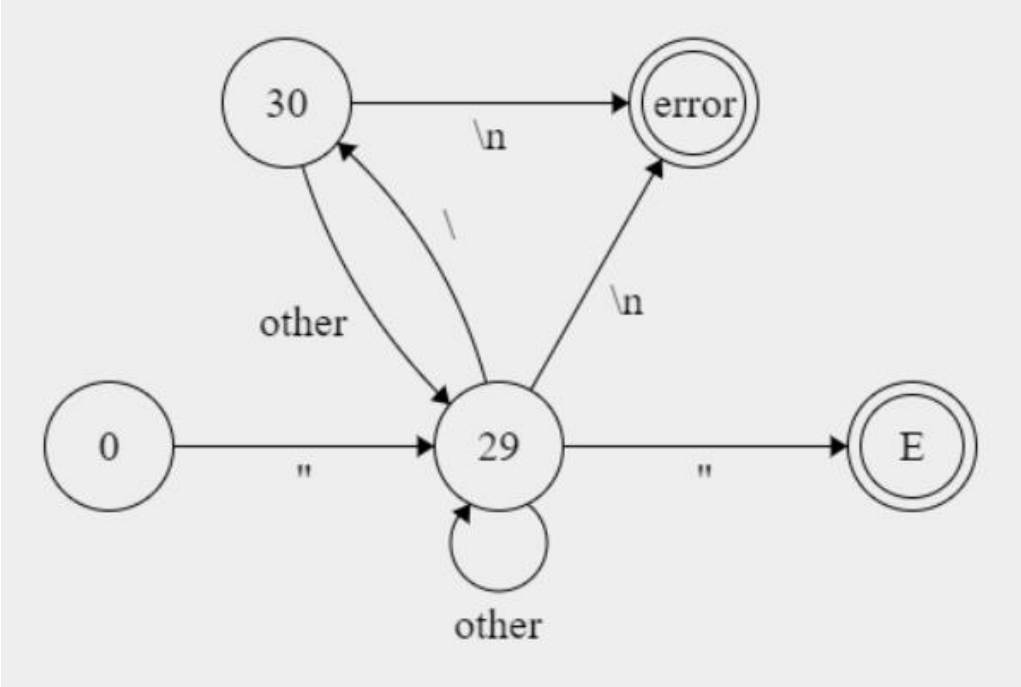
识别字符常量（含转义字符）的 DFA 如下：



3.2.7、字符串字面量

字符串字面量由一对双引号括起，但是如果出现了转义字符\”，该引号则不能视为字符串字面量的结束标志。

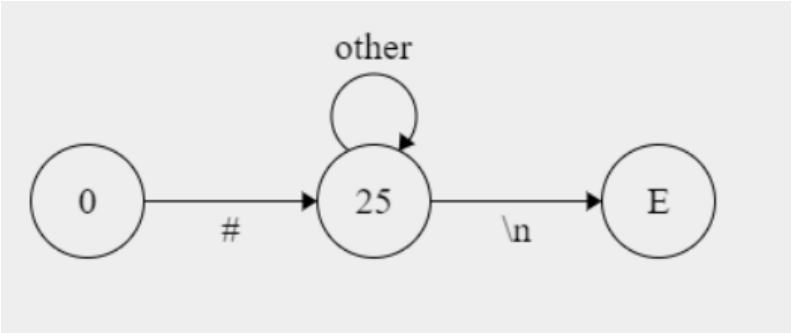
识别字符串字面量的 DFA 如下所示：



3.2.8、预处理命令

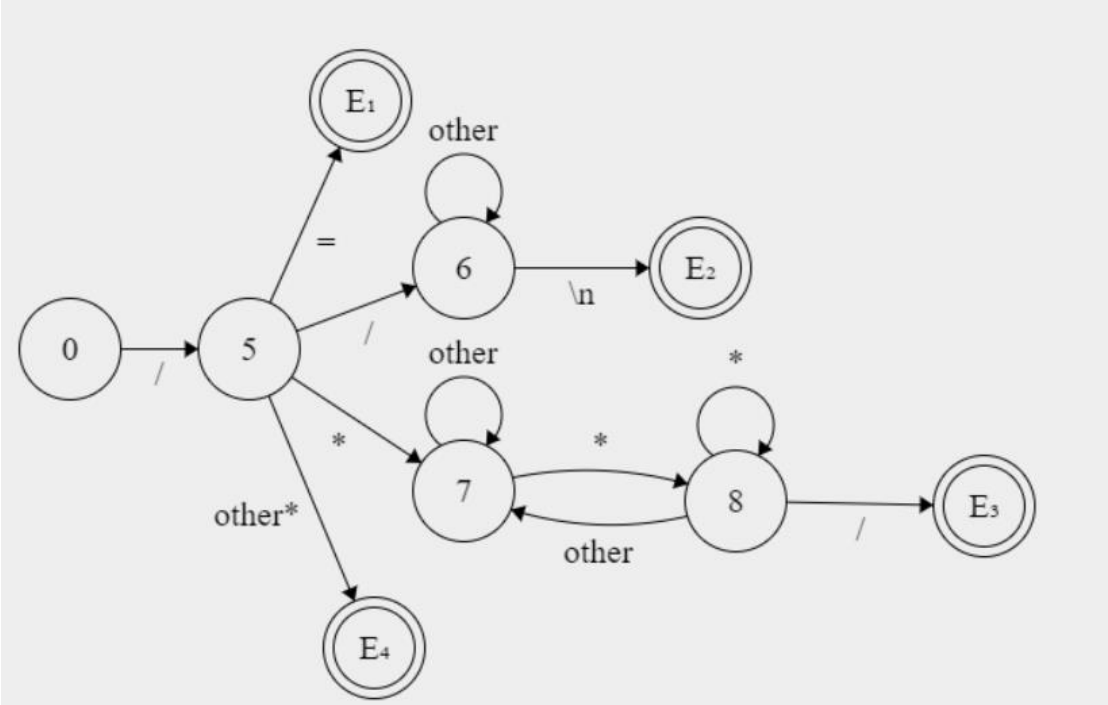
一般而言，预处理操作本应是在词法分析之前进行的，但是本次的实验只要求做词法分析，而输入里往往会有没有经过处理的预处理命令，所以只能将预处理的过程合并到词法分析中去。

C语言的预处理命令有三种，分别为文件包含、宏定义和条件编译。所有的预处理命令均以‘#’开头。因为词法分析程序无需处理预处理命令，所以我们只需将‘#’后的一行字符视作预处理命令，跳过即可。



3.2.9、注释

C语言的注释分为行注释和块注释，其中行注释的形式为（//…）,块注释的形式为（/*…*/）。识别注释的DFA（也包含了/=和/的识别）如下所示：



3.3、错误处理模块

3.3.1、数字常量错误

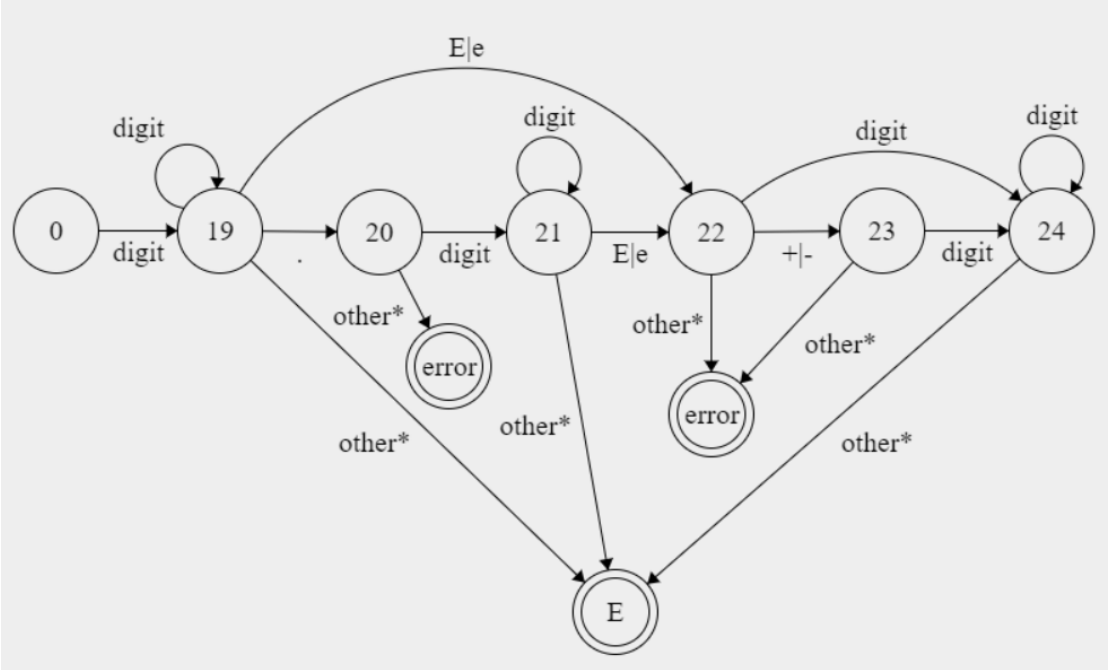
数字常量错误有以下几种情况：

①用科学计数法表示时没有给出指数，即E或e后没有数字，如13.14Eet，此时识别出错误数字常量13.14E和标识符et。

②用科学计数法表示时+号或-号后面没有数字，如13.14E+et,此时识别出错误数字常量13.14E+和标识符et。

③数字常量不完整，未识别完就遇到文件尾eof。

考虑错误处理后，识别数字常量的DFA修改如下：



输入：

```
13.14Eet        //error 13.14E and identifier et
13.14E+et       //error 13.14E+ and identifier et
13.14E          //error 13.14E (meet eof)
```

输出：

```
************************************************************
Token Stream:

<          IDENTIFIER ,          et      >
<          IDENTIFIER ,          et      >

************************************************************

************************************************************
Error Stream:

Row: 1
Error Type: NUMERIC_ERROR
13.14E

Row: 2
Error Type: NUMERIC_ERROR
13.14E+

Row: 3
Error Type: NUMERIC_ERROR
13.14E


************************************************************
```

3.3.2、字符常量错误

字符常量未闭合将导致错误。

```
1  '';
2  '\''
3  'a      //error 1: lack the right quote
4  'aaa'   //error 2: too much character
5  '       //error 3: eof
```

输入如上所示，前两行没有错误，第三行缺失右引号；第四行一对引号之间有三个字符，词法分析器在读完第一个字母a时，期待看到右引号，然而读入的下一个字符还是a，因此发现错误，并回退一个字符a到输入流中，接下来读到两个a，词法分析器将其视作标识符aa，然后右引号再次被发现未闭合的引号；在第五行读到文件尾仍未找到右引号匹配，再次报错。因此，总共有四个错误。

输出如下：

```
************************************************************
Token Stream:

<      CHAR_CONSTANTS ,          ''      >
<          SEMICOLON ,           ;       >
<      CHAR_CONSTANTS ,        '\''      >
<          IDENTIFIER ,          aa      >

************************************************************
```

```
************************************************************
Error Stream:

Row: 3
Error Type: CHAR_ERROR
'a

Row: 4
Error Type: CHAR_ERROR
'a

Row: 4
Error Type: CHAR_ERROR
'


Row: 5
Error Type: CHAR_ERROR
'



************************************************************
```

3.3.3、字符串字面量错误

字符串字面量的错误也是出在未闭合上。

输入如下：

```
1  "abc\n\"def"
2  "abc
3  def"
4  "abc
```

第一行字符串中使用了转义字符，因此中间的引号并不被识别为字符串结束标志，没有错误。

第二、三行本意是将一个字符串分到两行写，但是不符合 C 的语法。C 语言规定如果读到换行符时还没有匹配上左引号，将视之为未闭合错误。因此，二、三行各有一个错误，且 def 被视作为标识符。

第四行到文件尾仍未出现右引号，同样是字符串未闭合错误。

输出如下：

```
************************************************************
Token Stream:

<      STRING_LITERAL , "abc\n\"def"    >
<          IDENTIFIER ,         def      >

************************************************************

************************************************************
Error Stream:

Row: 2
Error Type: STRING_ERROR
"abc

Row: 3
Error Type: STRING_ERROR
"


Row: 4
Error Type: STRING_ERROR
"abc


************************************************************
```

3.3.4、注释错误

块注释在文件末尾未闭合错误。

```
/*abc
def*/
////////
/*cdvfdb
ss*

```

```
**********************************************************
Token Stream:


**********************************************************

**********************************************************
Error Stream:

Row: 4
Error Type: COMMENT_ERROR
/*cdvfdb
ss*


**********************************************************
```

3.3.5、非法字符错误

当自动机读到非法字符（例如$）时，词法分析器将发现该错误。

```
a$=$3.1415@
@_@$>//12$

```

注：第二行双斜线后视作注释

```
**********************************************************
Error Stream:

Row: 1
Error Type: ILLEGAL_CHAR
$

Row: 1
Error Type: ILLEGAL_CHAR
$

Row: 1
Error Type: ILLEGAL_CHAR
@

Row: 2
Error Type: ILLEGAL_CHAR
@

Row: 2
Error Type: ILLEGAL_CHAR
@

Row: 2
Error Type: ILLEGAL_CHAR
$


**********************************************************
```

```
************************************************************
Token Stream:

<          IDENTIFIER ,           a      >
<              ASSIGN ,           =      >
<   NUMERIC_CONSTANTS ,      3.1415      >
<          IDENTIFIER ,           _      >
<             GREATER ,           >      >

************************************************************
```

4、详细设计

4.1、TokenType 枚举类型

该数据结构指明了识别的 token 类型。

```
enum TokenType {
    NUMERIC_ERROR,          //数字常量错误
    CHAR_ERROR,             //字符常量错误
    STRING_ERROR,       //字符串字面量错误
    COMMENT_ERROR,          //注释错误
    ILLEGAL_CHAR,       //非法字符错误
    UNKNOWN,            //未知
    IDENTIFIER,             //标识符
    KEYWORD,            //关键字
    ADD,                //加
    SUB,                //减
    ADD_ASSIGN,             //加赋值
    INC,                //自增
    SUB_ASSIGN,             //减赋值
    DEC,                //自减
    ARROW,                  //箭头
    MUL,                //乘法
    MUL_ASSIGN,             //乘赋值
    DIV,                //除
    DIV_ASSIGN,             //除赋值
    LINE_COMMENT,       //行注释
    BLOCK_COMMENT,          //块注释
    MOD,                //模
    MOD_ASSIGN,             //模赋值
    EQUAL,                  //等于
    ASSIGN,                 //赋值
    BIT_AND_ASSIGN,         //位与赋值
    BIT_AND,            //位与
    AND,                //与
    BIT_OR_ASSIGN,          //位或赋值
```

```
    BIT_OR,                 //位或
    OR,                     //或
    BIT_XOR_ASSIGN,         //位异或赋值
    BIT_XOR,            //位异或
    XOR,                //异或
    LESS,                   //小于
    GREATER,            //大于
    LESS_EQUAL,             //小于等于
    GREATER_EQUAL,          //大于等于
    NOT_EQUAL,              //不等于
    SHL,                //左移
    SHR,                //右移
    SHL_ASSIGN,             //左移赋值
    SHR_ASSIGN,             //右移赋值
    NOT,                //非
    BIT_NOT,            //按位取反
    DOT,                //点
    L_PARENTHESIS,          //左圆括号
    R_PARENTHESIS,          //右圆括号
    L_BRACKET,              //左方括号
    R_BRACKET,              //右方括号
    L_BRACE,            //左花括号
    R_BRACE,            //右花括号
    SEMICOLON,              //分号
    COLON,                  //冒号
    COMMA,                  //逗号
    QUESTION,               //问号
    NUMERIC_CONSTANTS,      //数字常量
    CHAR_CONSTANTS,         //字符常量
    PREPROCESSOR,       //预处理命令
    STRING_LITERAL          //字符串字面量
};
```

4.2、tokenName 向量

该数据结构存储了对应的 token 类型名，用于输出。

```
const vector<string>tokenName = {
    "NUMERIC_ERROR",
    "CHAR_ERROR",
    ……
    ……
    "PREPROCESSOR",
    "STRING_LITERAL"
};
```

4.3、errorName 向量

该数据结构存储了错误类型名，用于输出。

```
const vector<string>errorName = {
    "NUMERIC_ERROR",
    "CHAR_ERROR",
    "STRING_ERROR",
    "COMMENT_ERROR",
    "ILLEGAL_CHAR"
};
```

4.4、keywordList 向量

该数据结构用于存储 C 语言关键字。(详见 3.2.2、关键字)

```
const set<string> keywordList = {
    "auto",
    "break",
    ……
    ……
    "_Complex",
    "_Imaginary"
};
```

4.5、Token 结构

结构成员由token类型（tokenType）和值(str）构成。

```
typedef struct
{
    TokenType tokenType;
    string str;
}Token;
```

4.6、Error 结构

结构成员由error类型（errorType）、值(str）、错误所在行(rowPos）构成。

```
typedef struct
{
    TokenType errorType;
    string str;
    int rowPos;
}Error;
```

4.7、Lexer 类

词法分析器类，详见注释。

```
class Lexer
{
public:
    Lexer();                    //构造函数
    ~Lexer() {};                //析构函数
    void scan();                //扫描程序
    void printInfo();           //打印统计信息
    void printTokenStream();    //打印记号流
    void printErrorStream();    //打印错误流
    void printSymbolTable();    //打印符号表

private:
    DFA dfa;                    //DFA
    vector<Token>tokenStream;   //记号流
    vector<Error>errorStream;   //错误流
    set<string>symbolTable;     //符号表
    array<int, 100>symbolCnt;   //各类符号计数
    int rowCnt;                 //行计数
    int tokenCnt;               //单词计数
    int charCnt;                //字符计数
    int rowPos;                 //错误定位行
};
```

4.8、DFA 类

```
class DFA
{
public:
    //构造函数
    DFA() :state(0), putbackFlag(false) {};
    //析构函数
    ~DFA() {};
    //状态转移函数，eol(end-of-line)用于标记是否读到一行的末尾
    bool transition(char ch,bool &eol);
    //返回自动机当前识别的token类型
    TokenType getTokenType() { return tokenType; }
    //返回当前自动机是否需要回退的标志
    bool getPutbackFlag() { return putbackFlag; }
    //返回当前自动机的状态
    int getState() { return state; }
    //状态转移过程中用于修改状态、是否回退标志以及识别的token类型
```

```
    inline void setArg(int s, bool p, TokenType t);

private:
    int state;                      //当前状态
    bool putbackFlag;               //标记是否需要回退
    TokenType tokenType;            //token类型
};
```

5、测试样例

程序的输入是从文件中读取，输出则有控制台和文件两种方式。（下面的截图会包含这两种形式）

输出结果包含统计信息(StatisticalInformation.txt)、记号流（TokenStream.txt)、错误流(ErrorStream.txt)以及符号表(SymbolTable.txt)，它们全部位于源文件同目录下的 output 目录下。

5.1、测试样例 1（正确程序）

这是该词法分析程序的 main 函数

```
#include<iostream>
#include"Lexer.h"

int main()
{
    Lexer lexer;
    lexer.scan();
    lexer.printInfo();
    lexer.printTokenStream();
    lexer.printErrorStream();
    lexer.printSymbolTable();
}
```

统计信息（含总行数、总字符数、总 token 数、各类 token 数）：

```
************************************************************
Statistical Information:
Row Count: 13
Character Count: 163
Token Count: 39

IDENTIFIER: 13
KEYWORD: 1
DOT: 5
L_PARENTHESIS: 6
R_PARENTHESIS: 6
L_BRACE: 1
R_BRACE: 1
SEMICOLON: 6

************************************************************
```

记号流：

```
****************************************************
Token Stream:

<             KEYWORD ,         int      >
<          IDENTIFIER ,        main      >
<       L_PARENTHESIS ,           (      >
<       R_PARENTHESIS ,           )      >
<             L_BRACE ,           {      >
<          IDENTIFIER ,       Lexer      >
<          IDENTIFIER ,       lexer      >
<           SEMICOLON ,           ;      >
<          IDENTIFIER ,       lexer      >
<                 DOT ,           .      >
<          IDENTIFIER ,        scan      >
<       L_PARENTHESIS ,           (      >
<       R_PARENTHESIS ,           )      >
<           SEMICOLON ,           ;      >
<          IDENTIFIER ,       lexer      >
<                 DOT ,           .      >
<          IDENTIFIER ,   printInfo      >
<       L_PARENTHESIS ,           (      >
<       R_PARENTHESIS ,           )      >
<           SEMICOLON ,           ;      >
<          IDENTIFIER ,       lexer      >
<                 DOT ,           .      >
<          IDENTIFIER , printTokenStream        >
<       L_PARENTHESIS ,           (      >
<       R_PARENTHESIS ,           )      >
<           SEMICOLON ,           ;      >
<          IDENTIFIER ,       lexer      >
<                 DOT ,           .      >
<          IDENTIFIER , printErrorStream        >
<       L_PARENTHESIS ,           (      >
<       R_PARENTHESIS ,           )      >
<           SEMICOLON ,           ;      >
<          IDENTIFIER ,       lexer      >
<                 DOT ,           .      >
<          IDENTIFIER , printSymbolTable        >
<       L_PARENTHESIS ,           (      >
<       R_PARENTHESIS ,           )      >
<           SEMICOLON ,           ;      >
<             R_BRACE ,           }      >

****************************************************
```

错误流（为空，因为是正确程序）：

```
****************************************************************
Error counts: 0
Error Stream:


****************************************************************
```

符号表：

```
****************************************************
Symbol Table:

Lexer
lexer
main
printErrorStream
printInfo
printSymbolTable
printTokenStream
scan

****************************************************
```

5.2、测试样例 2（正确程序）

```
//This is a correct program.

#include<stdio.h>

void swap(int &    a,int &b)
{
    /* The function is to swap the value of
        integer variable a and b*/
    int tmp  =  a;
    a=b;
    b=  tmp;
}

int   main()
{
    int a =4;
    int b=6;
    printf("a=%d, b=%d\n",a,b);
    swap(a,b);
    printf("a=%d, b=%d\n",a,b);
    return 0;
}
```

统计信息：

```
**************************************************************
Statistical Information:
Row Count: 22
Character Count: 248
Token Count: 70

IDENTIFIER: 21
KEYWORD: 8
ASSIGN: 5
BIT_AND: 2
L_PARENTHESIS: 5
R_PARENTHESIS: 5
L_BRACE: 2
R_BRACE: 2
SEMICOLON: 9
COMMA: 6
NUMERIC_CONSTANTS: 3
STRING_LITERAL: 2

**************************************************************
```

记号流：

```
**********************************************************
Token Stream:

<            KEYWORD ,        void      >
<         IDENTIFIER ,        swap      >
<      L_PARENTHESIS ,           (      >
<            KEYWORD ,         int      >
<            BIT_AND ,           &      >
<         IDENTIFIER ,           a      >
<              COMMA ,           ,      >
<            KEYWORD ,         int      >
<            BIT_AND ,           &      >
<         IDENTIFIER ,           b      >
<      R_PARENTHESIS ,           )      >
<            L_BRACE ,           {      >
<            KEYWORD ,         int      >
<         IDENTIFIER ,         tmp      >
<             ASSIGN ,           =      >
<         IDENTIFIER ,           a      >
<          SEMICOLON ,           ;      >
<         IDENTIFIER ,           a      >
<             ASSIGN ,           =      >
<         IDENTIFIER ,           b      >
<          SEMICOLON ,           ;      >
<         IDENTIFIER ,           b      >
<             ASSIGN ,           =      >
<         IDENTIFIER ,         tmp      >
<          SEMICOLON ,           ;      >
<            R_BRACE ,           }      >
<            KEYWORD ,         int      >
<         IDENTIFIER ,        main      >
<      L_PARENTHESIS ,           (      >
<      R_PARENTHESIS ,           )      >
<            L_BRACE ,           {      >
```

```
<            KEYWORD ,         int      >
<         IDENTIFIER ,           a      >
<             ASSIGN ,           =      >
<   NUMERIC_CONSTANTS ,           4      >
<          SEMICOLON ,           ;      >
<            KEYWORD ,         int      >
<         IDENTIFIER ,           b      >
<             ASSIGN ,           =      >
<   NUMERIC_CONSTANTS ,           6      >
<          SEMICOLON ,           ;      >
<         IDENTIFIER ,      printf      >
<      L_PARENTHESIS ,           (      >
<     STRING_LITERAL , "a=%d, b=%d\n"  >
<              COMMA ,           ,      >
<         IDENTIFIER ,           a      >
<              COMMA ,           ,      >
<         IDENTIFIER ,           b      >
<      R_PARENTHESIS ,           )      >
<          SEMICOLON ,           ;      >
<         IDENTIFIER ,        swap      >
<      L_PARENTHESIS ,           (      >
<         IDENTIFIER ,           a      >
<              COMMA ,           ,      >
<         IDENTIFIER ,           b      >
<      R_PARENTHESIS ,           )      >
<          SEMICOLON ,           ;      >
<         IDENTIFIER ,      printf      >
<      L_PARENTHESIS ,           (      >
<     STRING_LITERAL , "a=%d, b=%d\n"  >
<              COMMA ,           ,      >
<         IDENTIFIER ,           a      >
<              COMMA ,           ,      >
<         IDENTIFIER ,           b      >
<      R_PARENTHESIS ,           )      >
<          SEMICOLON ,           ;      >
<            KEYWORD ,      return      >
<   NUMERIC_CONSTANTS ,           0      >
<          SEMICOLON ,           ;      >
<            R_BRACE ,           }      >

**********************************************************
```

错误流：

```
**********************************************************
Error counts: 0
Error Stream:


**********************************************************
```

符号表：

```
**********************************************************
Symbol Table:

a
b
main
printf
swap
tmp

**********************************************************
```

5.3、测试样例 3（错误程序）

```
#include<stdio.h>
#include<string.h>
#define df

int main()
{
    #ifdef df

    char a[10]=″asdfgh
    char $[10e+]
    strcpy(b,a)
    printf(″%s″,b)

    #endif
}
/*
Unterminated
comment
```

统计信息：

StatisticalInformation.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
Statistical Information:
Row Count: 20
Character Count: 143
Token Count: 27

IDENTIFIER: 7
KEYWORD: 3
ASSIGN: 1
L_PARENTHESIS: 3
R_PARENTHESIS: 3
L_BRACKET: 2
R_BRACKET: 2
L_BRACE: 1
R_BRACE: 1
COMMA: 2
NUMERIC_CONSTANTS: 1
STRING_LITERAL: 1
```

记号流：

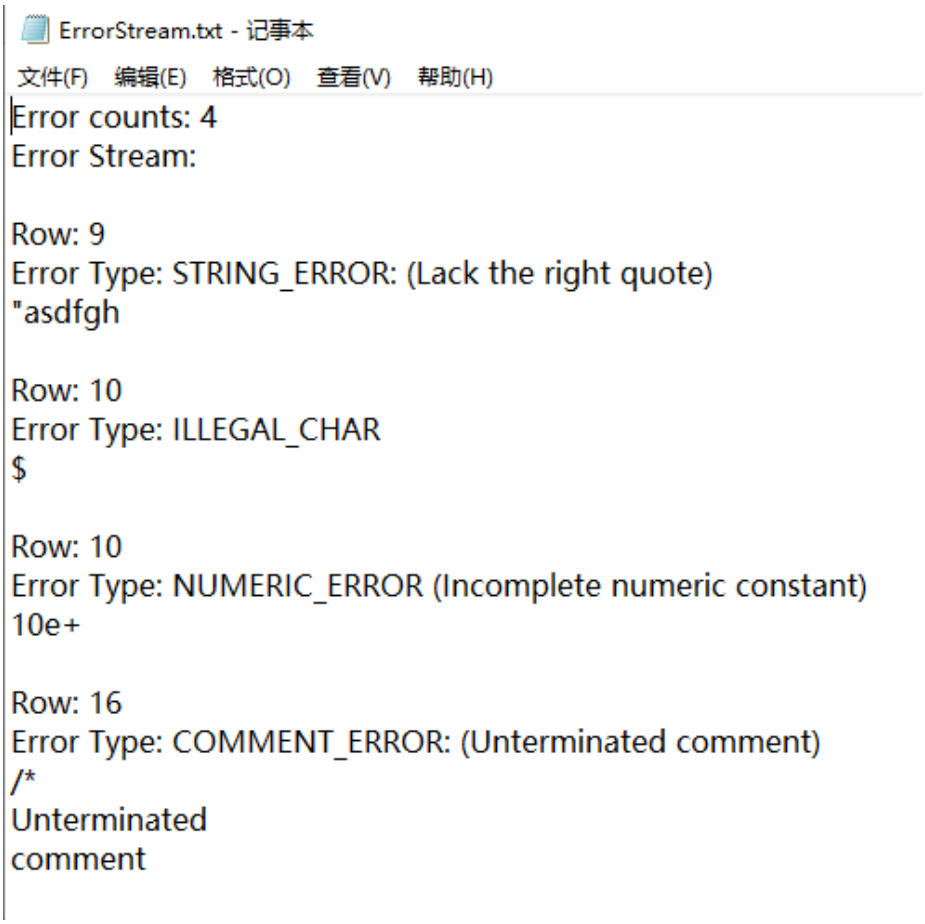
TokenStream.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

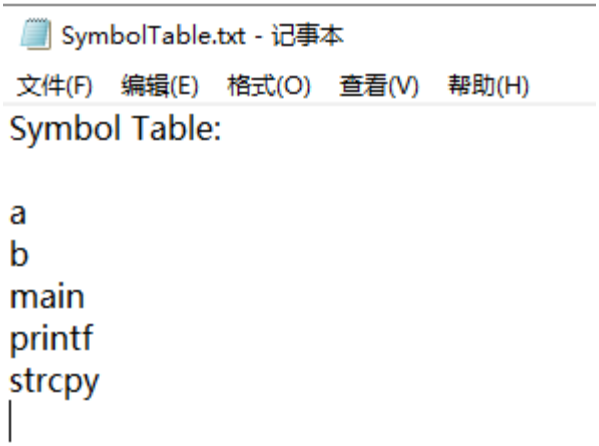
```
Token Stream:

<          KEYWORD ,       int  >
<       IDENTIFIER ,      main  >
<    L_PARENTHESIS ,         (  >
<    R_PARENTHESIS ,         )  >
<          L_BRACE ,         {  >
<          KEYWORD ,      char  >
<       IDENTIFIER ,         a  >
<        L_BRACKET ,         [  >
< NUMERIC_CONSTANTS ,        10  >
<        R_BRACKET ,         ]  >
<           ASSIGN ,         =  >
<          KEYWORD ,      char  >
<        L_BRACKET ,         [  >
<        R_BRACKET ,         ]  >
<       IDENTIFIER ,     strcpy  >
<    L_PARENTHESIS ,         (  >
<       IDENTIFIER ,         b  >
<            COMMA ,         ,  >
<       IDENTIFIER ,         a  >
<    R_PARENTHESIS ,         )  >
<       IDENTIFIER ,     printf  >
<    L_PARENTHESIS ,         (  >
<   STRING_LITERAL ,      "%s"  >
<            COMMA ,         ,  >
<       IDENTIFIER ,         b  >
<    R_PARENTHESIS ,         )  >
<          R_BRACE ,         }  >
```

错误流：



符号表：



5.4、测试样例 4（错误程序）

```
//This is a wrong program.

#include<stdio.h>
```

```
//ILLEGAL CHAR: @
void @swap    (int& a, int& b)
{
    /* The function is to swap the value of
        integer variable a and b*/
    int tmp = a;
    a = b;
    b = tmp;
}

int   main()
{
    //NUMERIC_ERROR: Incomplete numeric constant
    float a = 4Ew;

    //NUMERIC_ERROR: Incomplete numeric constant
    float b = 6.13e-;

    //CHAR_ERROR: lack the right quote
    char c = '

    //CHAR_ERROR: to much character
    char c = 'aaaa';

    //STRING_ERROR: lack the right quote
    printf("a=%d, b=%d\n, a, b);

    //ILLEGAL_CHAR: $
    swap($a, b);

    printf("a=%d, b=%d\n", a, b);
    return 0;
}

/*vbfbngfn
*
```

统计信息：

StatisticalInformation.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
Statistical Information:
Row Count: 40
Character Count: 493
Token Count: 69

IDENTIFIER: 23
KEYWORD: 10
ASSIGN: 7
BIT_AND: 2
L_PARENTHESIS: 5
R_PARENTHESIS: 4
L_BRACE: 2
R_BRACE: 2
SEMICOLON: 8
COMMA: 4
NUMERIC_CONSTANTS: 1
STRING_LITERAL: 1
```

记号流：

TokenStream.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
Token Stream:

<         KEYWORD ,     void          >
<      IDENTIFIER ,     swap          >
<   L_PARENTHESIS ,        (          >
<         KEYWORD ,      int >
<         BIT_AND ,        & >
<      IDENTIFIER ,        a >
<           COMMA ,        , >
<         KEYWORD ,      int >
<         BIT_AND ,        & >
<      IDENTIFIER ,        b >
<   R_PARENTHESIS ,        )          >
<         L_BRACE ,        { >
<         KEYWORD ,      int >
<      IDENTIFIER ,      tmp >
<          ASSIGN ,        = >
<      IDENTIFIER ,        a >
<       SEMICOLON ,        ; >
<      IDENTIFIER ,        a >
<          ASSIGN ,        = >
<      IDENTIFIER ,        b >
<       SEMICOLON ,        ; >
```

```
<       IDENTIFIER,       b  >
<         ASSIGN,         =  >
<       IDENTIFIER,     tmp  >
<        SEMICOLON,       ;  >
<         R_BRACE,        }  >
<         KEYWORD,      int  >
<       IDENTIFIER,     main           >
<     L_PARENTHESIS,       (           >
<     R_PARENTHESIS,       )           >
<         L_BRACE,        {  >
<         KEYWORD,    float           >
<       IDENTIFIER,       a  >
<         ASSIGN,         =  >
<       IDENTIFIER,       w  >
<        SEMICOLON,       ;  >
<         KEYWORD,    float           >
<       IDENTIFIER,       b  >
<         ASSIGN,         =  >
<        SEMICOLON,       ;  >
<         KEYWORD,     char           >
<       IDENTIFIER,       c  >
<         ASSIGN,         =  >
<         KEYWORD,     char           >
<       IDENTIFIER,       c  >
<         ASSIGN,         =  >
<       IDENTIFIER,     aaa  >
<       IDENTIFIER,  printf  >
<     L_PARENTHESIS,       (           >
<       IDENTIFIER,    swap           >
<     L_PARENTHESIS,       (           >
<       IDENTIFIER,       a  >
<          COMMA,         ,  >
<       IDENTIFIER,       b  >
<     R_PARENTHESIS,       )           >
<        SEMICOLON,       ;  >
<       IDENTIFIER,  printf  >
<     L_PARENTHESIS,       (           >
<    STRING_LITERAL, "a=%d, b=%d\n"
<          COMMA,         ,  >
<       IDENTIFIER,       a  >
<          COMMA,         ,  >
<       IDENTIFIER,       b  >
<     R_PARENTHESIS,       )           >
<        SEMICOLON,       ;  >
<         KEYWORD,  return           >
<  NUMERIC_CONSTANTS,      0     >
<        SEMICOLON,       ;  >
<         R_BRACE,        }  >
```

错误流：

ErrorStream.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
Error counts: 9
Error Stream:

Row: 6
Error Type: ILLEGAL_CHAR
@

Row: 18
Error Type: NUMERIC_ERROR (Incomplete numeric constant)
4E

Row: 21
Error Type: NUMERIC_ERROR (Incomplete numeric constant)
6.13e-

Row: 24
Error Type: CHAR_ERROR: (Lack the right quote)
'

Row: 27
Error Type: CHAR_ERROR: (Lack the right quote)
'a

Row: 27
Error Type: CHAR_ERROR: (Lack the right quote)
';

Row: 30
Error Type: STRING_ERROR: (Lack the right quote)
"a=%d, b=%d\n, a, b);

Row: 33
Error Type: ILLEGAL_CHAR
$

Row: 39
Error Type: COMMENT_ERROR: (Unterminated comment)
/*vbfbngfn
*
```

符号表：

SymbolTable.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助

```
Symbol Table:

a
aaa
b
c
main
printf
swap
tmp
w
```

5.5、测试样例 5（错误程序）

```
//This is a wrong program.

#include<stdio.h>
#define pi 3.14

int main(int argc,int *argv[])
{
    //test operator
    //result:+ - ++ = += -= -- . -> * *= % %= || & ^= >>= <= != ~ [ ]
{ } ( ) ; ? :
    //error:@,$
    +-++=@+=-=--.->**=%%=||&^=$>>=<=!=~[]{}();?:

    //test numeric
    6e+
    .15E-

    //test string
    "abc\n\t\'\"\a\b"
    "abcdef
    894574"

    //test char
    //error: lack right quote
    '\n'
    '\"'
    '\\\'

    /*
    this is a comment
    */

    #this is a preproccesor

    //keyword
    int char unsigned return true false if switch case while;

    //identifier
    id1 i_d My_i5d;
}
```

统计信息：

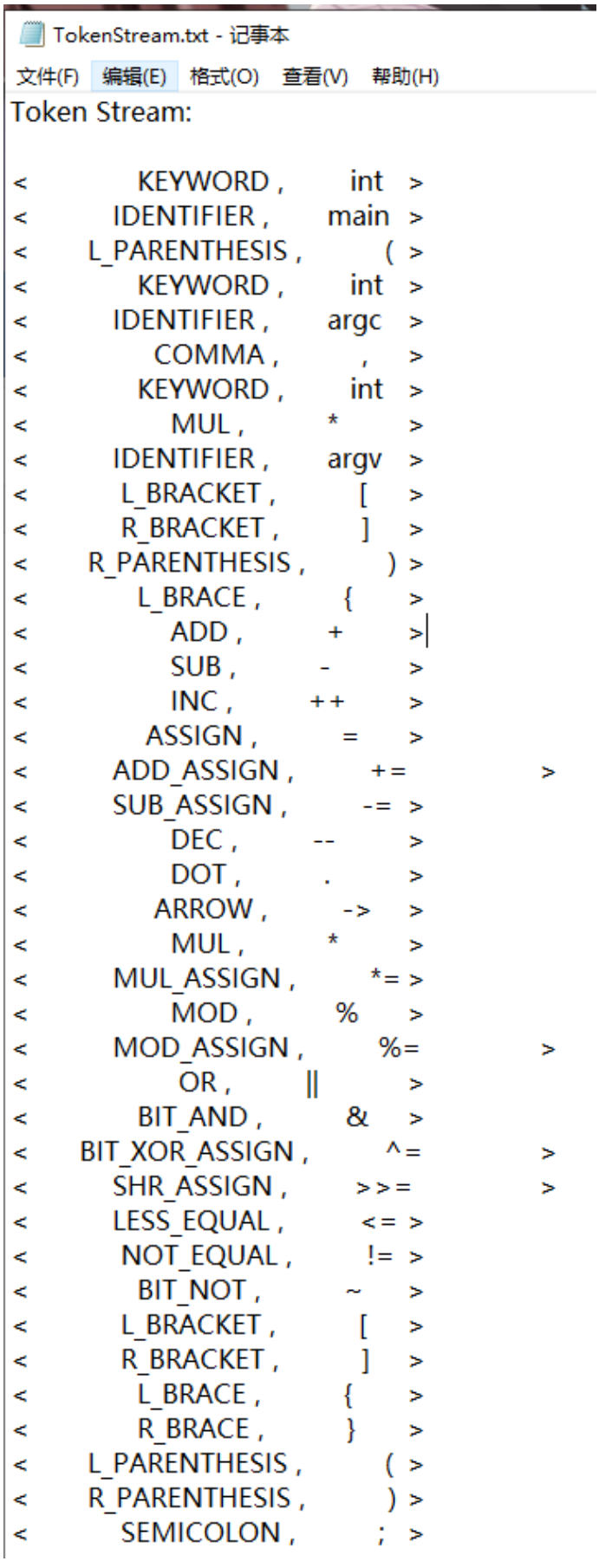
StatisticalInformation.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
Statistical Information:
Row Count: 40
Character Count: 438
Token Count: 63

IDENTIFIER: 8
KEYWORD: 11
ADD: 1
SUB: 1
ADD_ASSIGN: 1
INC: 1
SUB_ASSIGN: 1
DEC: 1
ARROW: 1
MUL: 2
MUL_ASSIGN: 1
MOD: 1
MOD_ASSIGN: 1
ASSIGN: 1
BIT_AND: 1
OR: 1
BIT_XOR_ASSIGN: 1
LESS_EQUAL: 1
NOT_EQUAL: 1
SHR_ASSIGN: 1
BIT_NOT: 1
DOT: 2
L_PARENTHESIS: 2
R_PARENTHESIS: 2
L_BRACKET: 2
R_BRACKET: 2
L_BRACE: 2
R_BRACE: 2
SEMICOLON: 3
COLON: 1
COMMA: 1
QUESTION: 1
NUMERIC_CONSTANTS: 1
CHAR_CONSTANTS: 2
STRING_LITERAL: 1
```

记号流：


TokenStream.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

Token Stream:

```
<          KEYWORD ,      int  >
<       IDENTIFIER ,     main  >
<     L_PARENTHESIS ,        (  >
<          KEYWORD ,      int  >
<       IDENTIFIER ,     argc  >
<            COMMA ,        ,  >
<          KEYWORD ,      int  >
<              MUL ,        *  >
<       IDENTIFIER ,     argv  >
<        L_BRACKET ,        [  >
<        R_BRACKET ,        ]  >
<     R_PARENTHESIS ,        )  >
<          L_BRACE ,        {  >
<              ADD ,        +  >
<              SUB ,        -  >
<              INC ,       ++  >
<           ASSIGN ,        =  >
<       ADD_ASSIGN ,       +=  >
<       SUB_ASSIGN ,       -=  >
<              DEC ,       --  >
<              DOT ,        .  >
<            ARROW ,       ->  >
<              MUL ,        *  >
<       MUL_ASSIGN ,       *=  >
<              MOD ,        %  >
<       MOD_ASSIGN ,       %=  >
<               OR ,       ||  >
<          BIT_AND ,        &  >
<    BIT_XOR_ASSIGN ,       ^=  >
<       SHR_ASSIGN ,      >>=  >
<       LESS_EQUAL ,       <=  >
<        NOT_EQUAL ,       !=  >
<          BIT_NOT ,        ~  >
<        L_BRACKET ,        [  >
<        R_BRACKET ,        ]  >
<          L_BRACE ,        {  >
<          R_BRACE ,        }  >
<     L_PARENTHESIS ,        (  >
<     R_PARENTHESIS ,        )  >
<        SEMICOLON ,        ;  >
```

```
<          QUESTION ,        ?   >
<             COLON ,        :   >
<               DOT ,        .   >
<    STRING_LITERAL , "abc\n\t\'\"\a\b"  >
<  NUMERIC_CONSTANTS ,    894574   >
<    CHAR_CONSTANTS ,      '\n'       >
<    CHAR_CONSTANTS ,      '\"'       >
<          KEYWORD ,       int   >
<          KEYWORD ,      char   >
<          KEYWORD ,  unsigned          >
<          KEYWORD ,    return   >
<       IDENTIFIER ,      true   >
<       IDENTIFIER ,     false   >
<          KEYWORD ,        if   >
<          KEYWORD ,    switch   >
<          KEYWORD ,      case   >
<          KEYWORD ,     while   >
<        SEMICOLON ,         ;   >
<       IDENTIFIER ,       id1   >
<       IDENTIFIER ,       i_d   >
<       IDENTIFIER ,    My_i5d   >
<        SEMICOLON ,         ;   >
<          R_BRACE ,        }   >
```

错误流：

ErrorStream.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
Error counts: 9
Error Stream:

Row: 11
Error Type: ILLEGAL_CHAR
@

Row: 11
Error Type: ILLEGAL_CHAR
$

Row: 14
Error Type: NUMERIC_ERROR (Incomplete numeric constant)
6e+

Row: 15
Error Type: NUMERIC_ERROR (Incomplete numeric constant)
15E-

Row: 19
Error Type: STRING_ERROR: (Lack the right quote)
"abcdef

Row: 20
Error Type: STRING_ERROR: (Lack the right quote)
"

Row: 26
Error Type: CHAR_ERROR: (Lack the right quote)
'\\

Row: 26
Error Type: ILLEGAL_CHAR
\

Row: 26
Error Type: CHAR_ERROR: (Lack the right quote)
'
```

符号表

6、图形界面

在开发词法分析器的过程中，我发现前面提到的两种输出方式都有缺陷：控制台输出往往过长，可视性不是很好，也不方便对程序正确性的检验。文件方式虽然在可视性上有所改善，但是由于我将输出分在了多个文件，查看的时候需要一个个打开，也很麻烦。另外，如果改变了输入文件名，则需要在源代码上加以修改（当然，也可以将文件名作为命令行参数传给程序）。

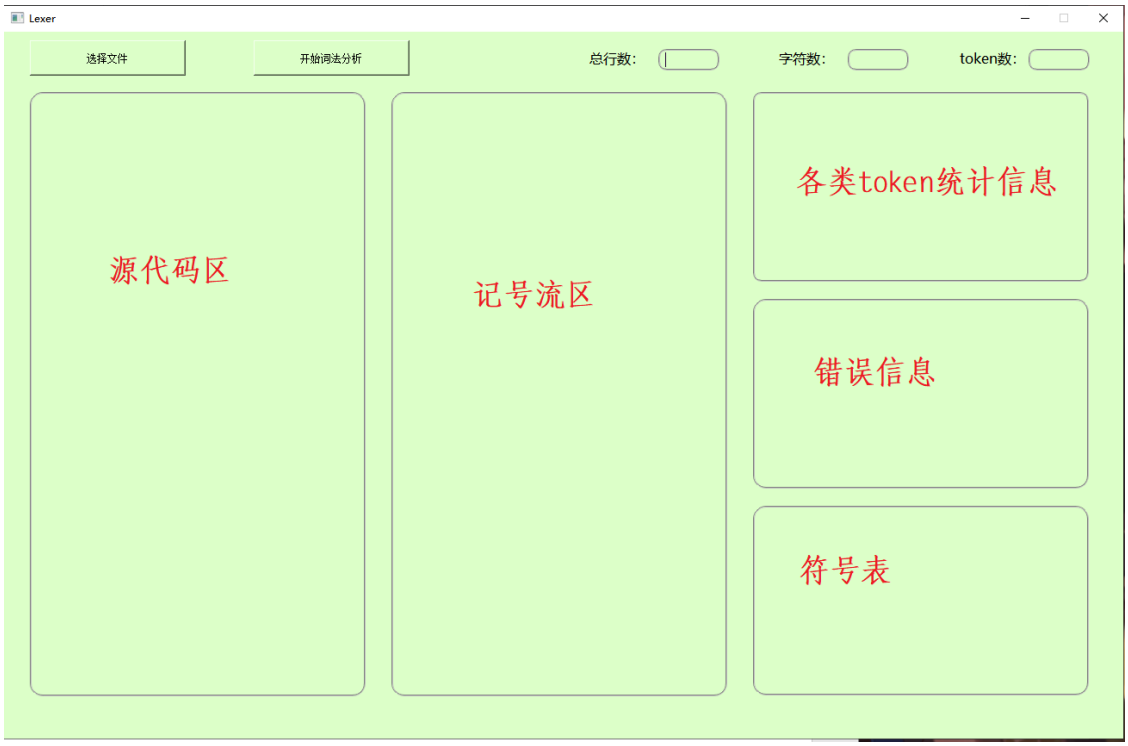
正是基于以上几点原因，再加上我曾经做过的课程设计的经验，让我想到可以用图形界面来解决这个问题。图形界面的好处在于，一个大界面可以分成好几个部分，同时展示记号流、错误以及统计数据等信息，并且可以将源程序与词法分析得到的token流进行直观的对比。此外，还可以直接选择不同的文件，而不需考虑路径问题。

初始的图形界面如下所示：

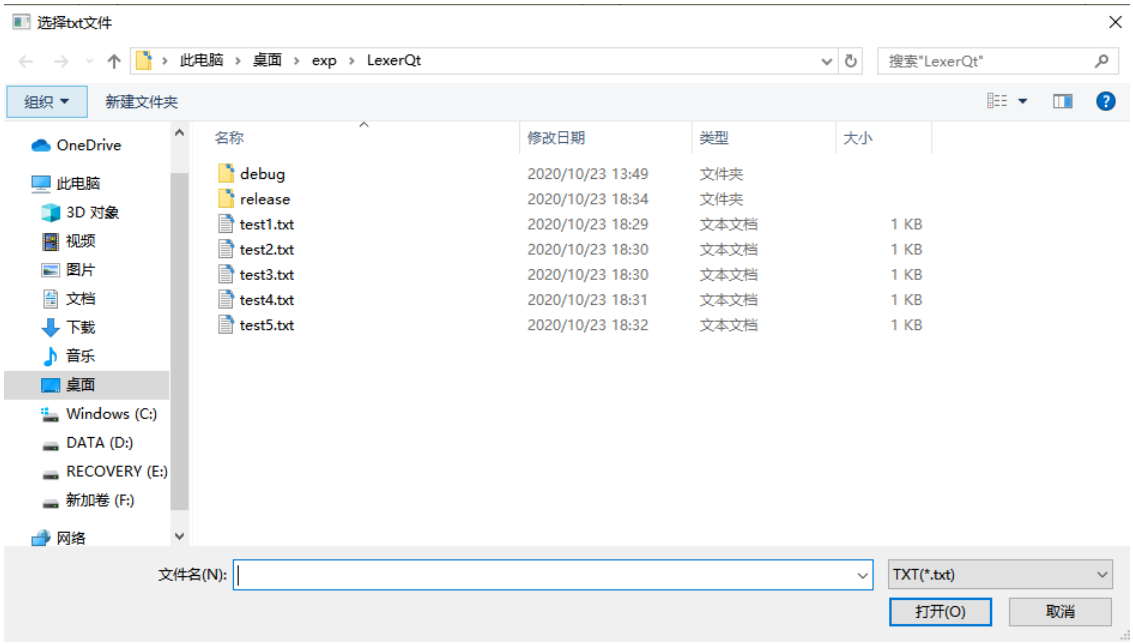
其中左上角有两个按钮，分别为“选择文件”和“开始词法分析”。

右上角分别展示了该源程序的总行数、字符数和token数。

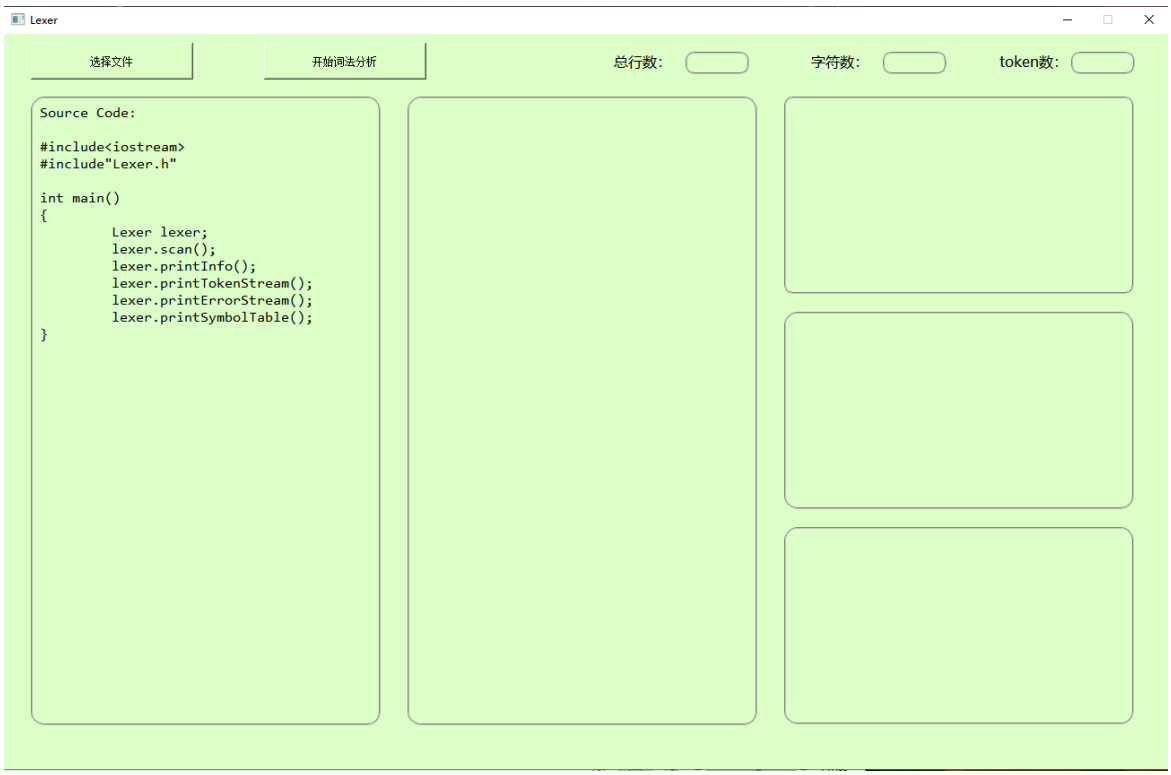
剩余的部分被划分为5个区域，具体内容参见下图。



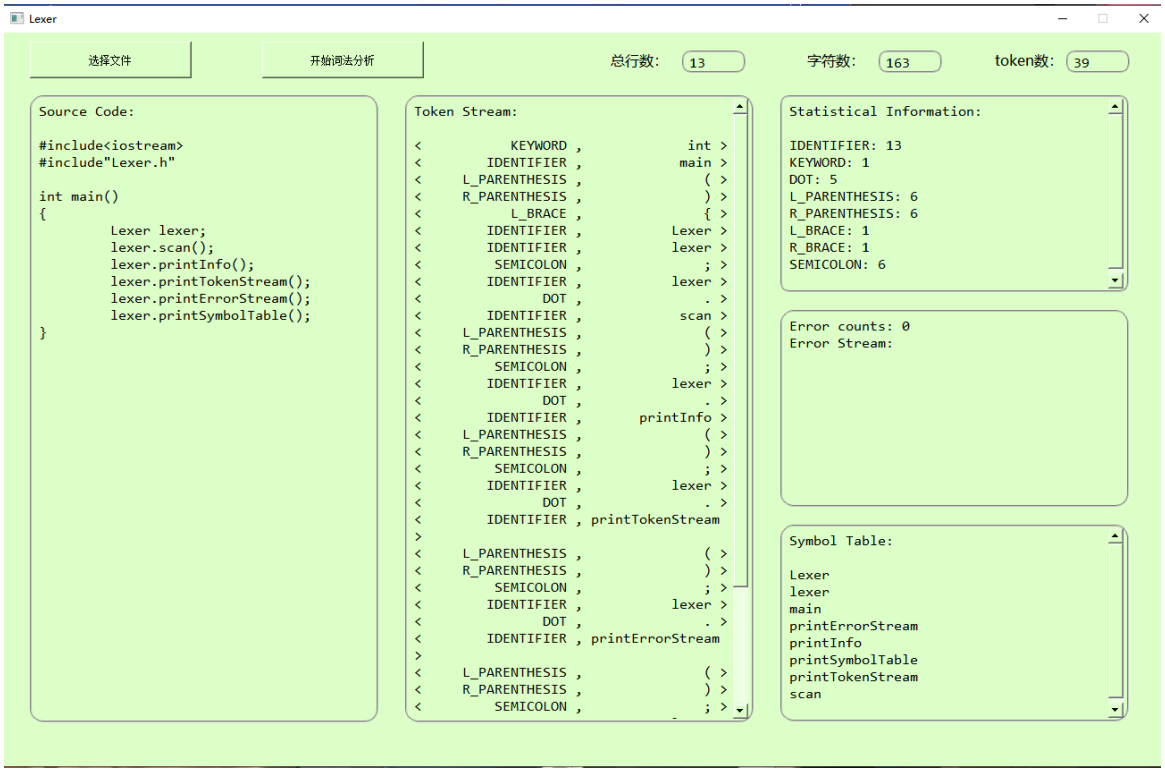
点击“选择文件”按钮后，可以选择所需分析的文本文件。



选择指定文件并打开后，程序将会读取文件中的信息，并将其显示在源代码区。

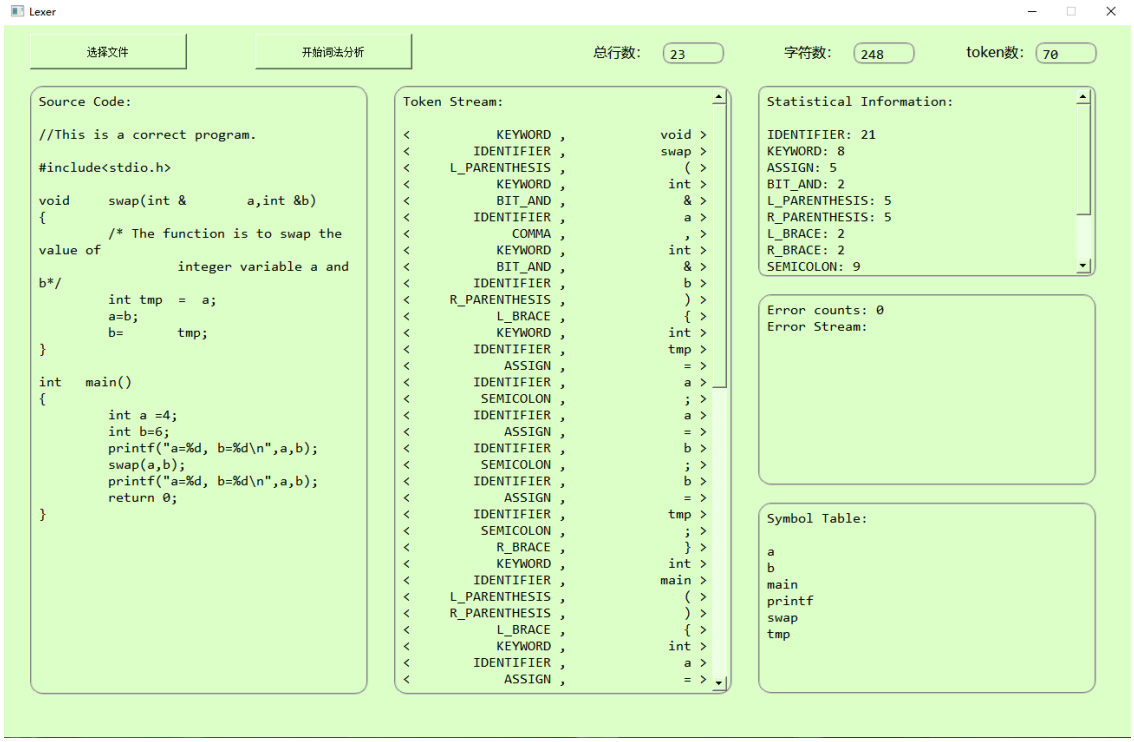


接下来点击“开始词法分析”，即可输出分析的各项结果。

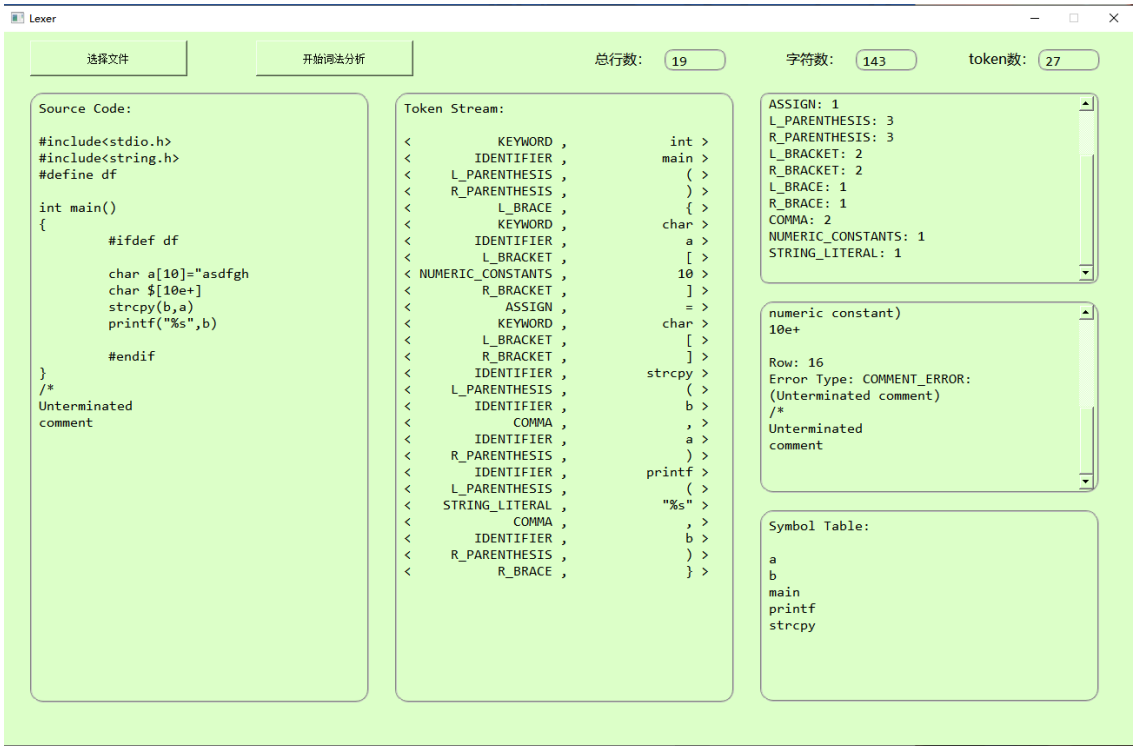


以下是其它四个程序的分析结果：

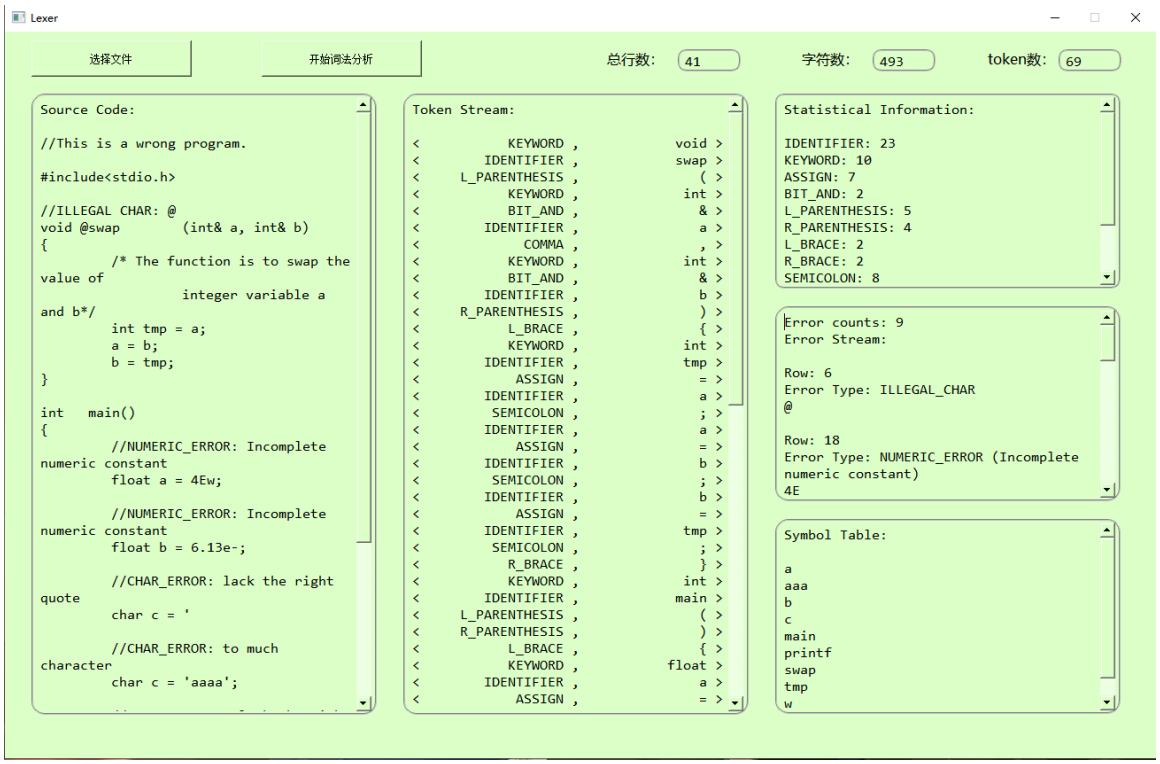
测试样例2：



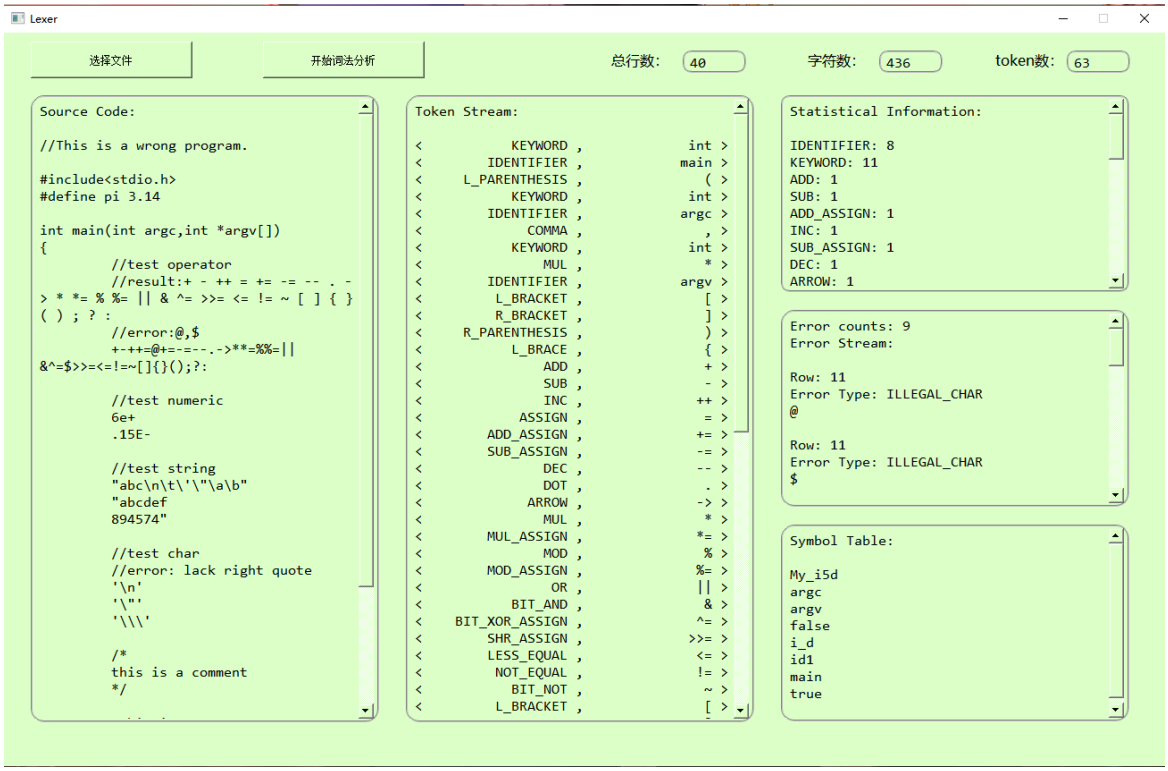
测试样例3：



测试样例4：


Lexer
选择文件
开始词法分析
总行数: 41
字符数: 493
token数: 69
Source Code:
//This is a wrong program.
#include<stdio.h>
//ILLEGAL CHAR: @
void @swap (int& a, int& b)
{
/* The function is to swap the value of
integer variable a and b*/
int tmp = a;
a = b;
b = tmp;
}
int main()
{
//NUMERIC_ERROR: Incomplete numeric constant
float a = 4Ew;
//NUMERIC_ERROR: Incomplete numeric constant
float b = 6.13e-;
//CHAR_ERROR: lack the right quote
char c = '
//CHAR_ERROR: to much character
char c = 'aaaa';
Token Stream:
< KEYWORD , void >
< IDENTIFIER , swap >
< L_PARENTHESIS , (>
< KEYWORD , int >
< BIT_AND , & >
< IDENTIFIER , a >
< COMMA , , >
< KEYWORD , int >
< BIT_AND , & >
< IDENTIFIER , b >
< R_PARENTHESIS ,) >
< L_BRACE , { >
< KEYWORD , int >
< IDENTIFIER , tmp >
< ASSIGN , = >
< IDENTIFIER , a >
< SEMICOLON , ; >
< IDENTIFIER , a >
< ASSIGN , = >
< IDENTIFIER , b >
< SEMICOLON , ; >
< IDENTIFIER , b >
< ASSIGN , = >
< IDENTIFIER , tmp >
< SEMICOLON , ; >
< R_BRACE , } >
< KEYWORD , int >
< IDENTIFIER , main >
< L_PARENTHESIS , (>
< R_PARENTHESIS ,) >
< L_BRACE , { >
< KEYWORD , float >
< IDENTIFIER , a >
< ASSIGN , = >
Statistical Information:
IDENTIFIER: 23
KEYWORD: 10
ASSIGN: 7
BIT_AND: 2
L_PARENTHESIS: 5
R_PARENTHESIS: 4
L_BRACE: 2
R_BRACE: 2
SEMICOLON: 8
Error counts: 9
Error Stream:
Row: 6
Error Type: ILLEGAL_CHAR
@
Row: 18
Error Type: NUMERIC_ERROR (Incomplete numeric constant)
4E
Symbol Table:
a
aaa
b
c
main
printf
swap
tmp
w

测试样例5：


Lexer
选择文件
开始词法分析
总行数: 40
字符数: 436
token数: 63
Source Code:
//This is a wrong program.
#include<stdio.h>
#define pi 3.14
int main(int argc,int *argv[])
{
//test operator
//result:+ - ++ = += -= -- . -> * *= % %= || & ^= >>= <= != ~ [ ] { } ( ) ; ? :
//error:@,$
+-++=@+=-=--.->**=%%=|| &^=$>>=<=!=~[]{}();?:
//test numeric
6e+
.15E-
//test string
"abc\n\t\'\"\a\b"
"abcdef
894574"
//test char
//error: lack right quote
'\n'
'\"'
'\\\'
/*
this is a comment
*/
Token Stream:
< KEYWORD , int >
< IDENTIFIER , main >
< L_PARENTHESIS , (>
< KEYWORD , int >
< IDENTIFIER , argc >
< COMMA , , >
< KEYWORD , int >
< MUL , * >
< IDENTIFIER , argv >
< L_BRACKET , [>
< R_BRACKET ,] >
< R_PARENTHESIS ,) >
< L_BRACE , { >
< ADD , + >
< SUB , - >
< INC , ++ >
< ASSIGN , = >
< ADD_ASSIGN , += >
< SUB_ASSIGN , -= >
< DEC , -- >
< DOT , . >
< ARROW , -> >
< MUL , * >
< MUL_ASSIGN , *= >
< MOD , % >
< MOD_ASSIGN , %= >
< OR , || >
< BIT_AND , & >
< BIT_XOR_ASSIGN , ^= >
< SHR_ASSIGN , >>= >
< LESS_EQUAL , <= >
< NOT_EQUAL , != >
< BIT_NOT , ~ >
< L_BRACKET , [>
Statistical Information:
IDENTIFIER: 8
KEYWORD: 11
ADD: 1
SUB: 1
ADD_ASSIGN: 1
INC: 1
SUB_ASSIGN: 1
DEC: 1
ARROW: 1
Error counts: 9
Error Stream:
Row: 11
Error Type: ILLEGAL_CHAR
@
Row: 11
Error Type: ILLEGAL_CHAR
$
Symbol Table:
My_i5d
argc
argv
false
i_d
id1
main
true

7、总结

7.1、程序实现的效果

该词法分析程序设计完成了题目所要求的全部功能。它接受C语言程序代码作为输入，分析后得出记号流（以二元组的形式）输出给用户。该程序能够正确识别C语言的各种单词记号，跳过源程序中的注释和预处理命令，输出对源程序进行分析后的各种统计结果。该程序还能够对部分词法错误进行识别和处理，并报告给用户错误所处的位置，以便用户对程序作出修改。对于C程序中的词法错误，该分析器能够进行一些简单的错误恢复，以便词法分析能够继续下去。

7.2、实验遇到的问题

实验中碰到的最大的麻烦是对换行符和文件尾的处理。比如，行注释和预处理命令分别为//和#号之后的一行，当他们遇到换行符时就表示成功识别；而常量（数字常量、字符常量、字符串字面量）又要求必须在同一行，否则就会被视作未闭合或不完整错误。再如，块注释如果到达文件尾都没有找到*/，以及字符串到文件尾都没有右引号匹配，都应被发现为未闭合错误。

对这些问题，必须非常小心地进行处理和设计，否则轻则遗漏很多本该发现的错误，重则直接导致程序崩溃。在整个实验过程中，我一开始有很多细节没有考虑好，之后就只能通过单步调试的方法一个个地去修复这些bug，好在最后顺利完成了实验要求。

7.3、设计亮点和缺点

该词法分析器完全基于DFA实现，所有种类的token（除了关键字是先用DFA识别为标识符然后再判断是否在关键字列表中）都由DFA来识别，这样做的好处是处理起来更加统一和简单，当然缺点就是增加了代码量规模。

该词法分析器能够将词法错误局限在一个很小的范围内，即一个错误只可能会对其临近的代码造成影响，使得词法分析过程能够完整而正确地运行下去，而不会出现一个小错误导致后面的识别全部出错或中断的情况。这与现代

编译器的设计理念是完全相符的。

但是，由于C语言本身是相当复杂的，而我对词法错误的考虑也必然不可能做到面面俱到（实际上要完成符号识别和错误处理远远不是我这里的一千多行代码所能做到的），所以这个程序对错误的处理终究是有限的，还有一些符号我也可能未能正确识别，这有待我后续继续学习和改进。

7.4、设计的创新点

该程序的创新之处主要在于图形界面的设计，通过图形界面可以将输入和输出进行对比展示，使得分析结果变得更加直观，对用户来说比较友好，也便于设计者发现bug进行修改。

7.5、实验心得

在本次实验中，通过对数据结构、DFA、扫描程序、类接口、性能优化等各方面细节的仔细研究，大大加深了我对词法分析过程和自动机理论的理解。之前的学习都是只停留在理论阶段，虽然可能会做题，但未必就真正掌握课程里的核心内容；而在亲自动手实践，尤其是不断发现BUG、调试BUG、修复BUG的过程中，我才算是真正理解了词法分析。总之，这次实验给我带来了很大的收获，也让我期待接下来的语法分析和语义分析。