C语言词法分析程序的设计与实现实验报告

1、实验题目及要求：

完成C语言词法分析程序的设计与实现

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.可以统计源程序中的语句行数、各类单词的个数、以及字符总数，并输出统计结果。

4.检查源程序中存在的词法错误，并报告错误所在的位置。

5.对源程序中出现的错误进行适当的恢复，使词法分析可以继续进行，对源程序进行一次扫描，即可检查并报告源程序中存在的所有词法错误。

2、程序设计说明

2.1程序总体说明：

首先逐字符(包含\n,\t等)读取测试文件内的C语言代码，并将读取到的字符标记类型。如果读取到的为制表符及空格，类型值为0；字母的类型值为1，数字的类型值为2，运算符的类型值为3，界符为4；若为预处理，如#include<stdio.h>,则分配为5，对其的处理不是词法分析阶段的任务，故跳过。

然后根据字符的类型值，结合状态转换图，进行词法分析。

2.2全局变量及宏说明：

int* cur：当前指针

int line：存储当前行数

int state：存储当前识别的状态

long ch_num：存储字符总数

ifstream demo：测试文件流

string token:存储token

#define BUFFSIZE 1024:缓冲区

```
#define LINEPLUS if(cur_ch == '\n'&& isForward)\
                 {\
```

```
                    Line++;
                }
```

当读取到换行符且要继续向前读取时,行数+1.

#define TEST_FILE "test.txt" 测试文件路径

#define ERROR_FILE "error.txt" 错误信息文件路径

#define DATA_FILE "data.txt" 输出信息文件路径

2.3 数据结构说明：

```
set<string> op = {
    "+","-","*","/","%","++","--",//算数运算
    ">","<","==",">=","<=","!=",//关系运算
    "&&","||","!",//逻辑运算
    "&","|","~","^","<<",">>",//位操作
    "=","+=","-=","*=","/=","%=","&=","|=","^=",">>=","<<=",//赋值运算
    ".",":","\\"
};//运算符

set<string> delimiter{
    "(",")","[","]","{","}",";",",","\"","\'",
};//界符

set<string> keyword{
    "auto","short","int","long","float","double","char",
    "struct","union","enum","typedef","const","unsigned",
    "signed","extern","register","static","void","if","else",
    "switch","case","for","do","while","goto",
    "continue","break","default","sizeof","return",
};//关键字
```

使用 set 集合分别存储运算符，界符，以及关键字。

```
int left_buf[BUFFSIZE] = { 0 };//左输入缓冲
int right_buf[BUFFSIZE] = { 0 };//右输入缓冲
```

左右输入缓冲区

```
string error_type[1] = { "Error input" };
string data_type[8] = { "identifier","constant","op","delimiters","keyword","string","character" };
```

两个数组分别存储错误类型，和字符的类型。

2.4 输入缓冲区说明：

为了得到某一单词符号的确切性质，只从该的单词本身所含有的字符不能做判定，需要超前扫描若干字符之后才能做出确定的分析。因此有必要设置一个缓冲区来保存输入符

号串。

2.5 记号文法说明：

1.标识符：用 id,letter,digit 分别表示标识符、字母+下划线、数字。则其正规表达式为：

letter(letter|digit)*

将子表达式(letter|digit)*命名为 rid，则其文法为：

id→letter rid

rid→ ε |letter rid|digit rid

2.常数:以 digits 表示$(\boldsymbol{digit})^{+}$,remainder 表示$(\boldsymbol{digit})^{*}$。

整数正规文法: digits→digit remainder

Remainder→ ε |digit remainder

无符号数: num→digit num1

num1→digit num1|.num2|E num4|

num2→digit num3

num3→digit num3|E num4| ε

num4→+digits|-digits|digit num5

digits→digit num5

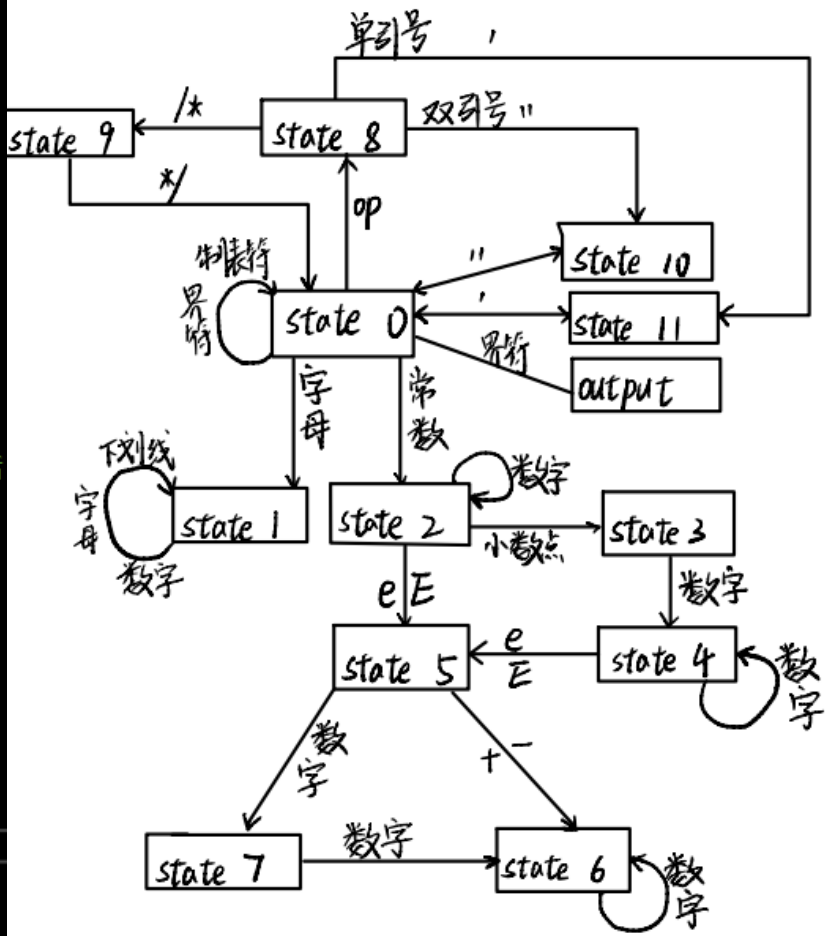
num5→digit num5| ε

3.运算符正规定义式：

relop→<|<=|=|<>|>=|>

2.6 状态转换图说明：

```
switch (state)
{
case 0://初始状态
{ ... }
case 1://标识符、关键字
{ ... }
case 2://常数
{ ... }
case 3://小数点后第一个数字
{ ... }
case 4://小数点后的其余数字
{ ... }
case 5://识别指数
{ ... }
case 6://识别指数E之后的符号之后
{ ... }
case 7://识别指数次数
{ ... }
case 8://运算符和界符
{ ... }
case 9://注释
{ ... }
case 10://识别字符串
{ ... }
case 11://识别字符
{ ... }
default:
    exit(1);
```



2.6 主要函数说明

1.void type_assign()

作用:为缓冲区的每一个字符分配类型，并用 cha[i]数组存储该类型。制表符和空格 cha[i]=0;字母和下划线 cha[i]=1;数字 cha[i]=2;运算符 cha[i]=3;界符 cha[i]=4;预处理 cha[i]=5。这样首先分配类型有利于之后状态转换函数的进行。

2.int get_ch()

作用：在文件没有读取完成之前，使用 file >> noskipws >> ch,不跳过任何字符地逐字符读取。返回值为当前字符在缓冲区地下标，当返回值为-1 时代表文件结束。

3.void read_code()

作用：根据 type_assign()函数和 get_ch()函数的结果确定词法分析的状态，并根据状态转换图进行词法分析。

4. void out_data()

作用:输出当前的记号在第几行,以及其数据类型,如常数,字符等。

5. void out_error()

作用:输出当前错误信息的行数、错误信息及其错误类型。

3. 测试说明:

测试文件:

error.txt test.txt data.txt LexicalAnalysis.cpp sig

```
#include<stdio.h>
//test
int main()
{
    float num1;
    int _num2, 4flag;
    _num2 -= (int) num1;
    num1 = 6E-2;
    printf("right sentence");//test file
    a = "false sentence;
    /*jjjjjjjjjj
    aaaaaaaa*/
    num1 -= 1.354;
    return 0;
}
```

结果说明:

```
C:\Users\艾桐\Desktop\LexicalAnalysis\Debug\LexicalAnalysis.exe
详细信息参见输出文件
字符总数: 220
行数: 15
标识符数量: 10
常数数量: 4
运算符数量: 5
界符数量: 16
关键字数量: 5
字符串数量: 1
请按任意键继续. . .
```

错误说明:

error.txt　test.txt　data.txt　LexicalAnalysis.cpp　sign.h　File.h

```
Error in Line 6:   f    Error type: Error input
Error in Line 11:   "false sentence;    Error type: 未识别结束符
```

第六行处，标识符不能以数字开头，成功检错

第十行处，双引号没有成对出现，成功检错

单词记号说明：

```
Line 3:   int            Notation: keyword
Line 3:   main           Notation: identifier
Line 3:   (              Notation: delimiters
Line 3:   )              Notation: delimiters
Line 4:   {              Notation: delimiters
Line 5:   float          Notation: keyword
Line 5:   num1           Notation: identifier
Line 5:   ;              Notation: delimiters
Line 6:   int            Notation: keyword
Line 6:   _num2          Notation: identifier
Line 6:   ,              Notation: delimiters
Line 6:   4              Notation: constant
Line 6:   lag            Notation: identifier
Line 6:   ;              Notation: delimiters
Line 7:   _num2          Notation: identifier
Line 7:   -              Notation: op
Line 7:   =              Notation: op
Line 7:   (              Notation: delimiters
Line 7:   int            Notation: keyword
Line 7:   )              Notation: delimiters
Line 7:   num1           Notation: identifier
Line 7:   ;              Notation: delimiters
Line 8:   num1           Notation: identifier
Line 8:   =              Notation: op
Line 8:   6E-2           Notation: constant
Line 8:   ;              Notation: delimiters
Line 9:   printf         Notation: identifier
Line 9:   (              Notation: delimiters
Line 9:   "right sentence"  Notation: string
Line 9:   )              Notation: delimiters
Line 9:   ;              Notation: delimiters
Line 10:   a             Notation: identifier
Line 10:   =             Notation: op
```

```
Line 13:   num1        Notation: identifier
Line 13:   -=          Notation: op
Line 13:   1.354       Notation: constant
Line 13:   ;           Notation: delimiters
Line 14:   return      Notation: keyword
Line 14:   0           Notation: constant
Line 14:   ;           Notation: delimiters
Line 15:   }           Notation: delimiters
```

由此可见，可以识别出源程序中的每个单词记号，并以记号的形式输出每个单词符号。

可识别并跳过源程序中的注释(2 行，11-12 行)。

可识别标识符与关键字(5，6 行)。

可识别指数与常数(8,13 行)。

可以统计源程序中的语句行数，各类单词的个数，以及字符总数，并输出统计结果。

同时可以检查源程序中存在的语法错误，并报告错误所在的位置，遇到错误时可以适当的修复问题，使词法分析继续。

附源程序：

Sign.h:

```
#pragma once
#include<set>
#include<string>
using namespace std;

int cha[1024] = { 0 };
set<string> op = {
    "+","-","*","/","%","++","--",//算数运算
    ">","<","==",">=","<=","!=",//关系运算
    "&&","||","!",//逻辑运算
    "&","|","~","^","<<",">>",//位操作
    "=","+=","-=","*=","/=","%=","&=","|=","^=",">>=","<<=",//赋值
运算
    ".",":","\\"
};//运算符

set<string> delimiter{
    "(",")","[","]","{","}",";",",","\"","\'",
```

```
};//界符

set<string> keyword{
    "auto","short","int","long","float","double","char",
    "struct","union","enum","typedef","const","unsigned",
    "signed","extern","register","static","void","if","else",
    "switch","case","for","do","while","goto",
    "continue","break","default","sizeof","return",
};//关键字

//标记字符类型
void type_assign()
{
    int i = 0;
    for (i = 0; i < 1024; i++)
    {
        char ch = (char)i;
        string tmp = "";
        tmp += ch;//转换为string便于后续操作
        while (ch == '=')
            break;
        if (ch == '\t' || ch == '\r' || ch == '\n' || ch == ' ')
            cha[i] = 0;//跳过
        else if ((ch >= 'a' && ch <= 'z') || (ch >= 'A' && ch <=
'Z') || ch == '_')
            cha[i] = 1;//字母
        else if (ch >= '0' && ch <= '9')
            cha[i] = 2;//数字
        else if (op.find(tmp) != op.end())
            cha[i] = 3;//运算符
        else if (delimiter.find(tmp) != delimiter.end())
            cha[i] = 4;//界符
        else if (ch == '#')
            cha[i] = 5;//预处理
        else
            cha[i] = -1;
    }
}
```

File.h:

```
#pragma once
#define TEST_FILE "test.txt"
#define ERROR_FILE "error.txt"
#define DATA_FILE "data.txt"
```

LexcicalAnalysis.cpp:

```
#include<iostream>
#include<fstream>
#include<sstream>
#include<iomanip>
#include"sign.h"
#include"File.h"
#define BUFFSIZE 1024
#define  LINEPLUS  if(cur_ch == '\n' && isForward)\
                  {\
                      line++;\
                  }


int get_ch();//利用配对缓冲区，读取缓冲区字符
void read_code(ofstream& dtxt, ofstream& etxt);//读取源程序，并进行词法分析
void out_data(string data, int data_sign, ofstream& txt);//数据类型信息
void out_error(string data, int error_sign, ofstream& txt);//错误信息
void print_inf();//输出语句行数,各类单词、字符个数

int left_buf[BUFFSIZE] = { 0 };//左输入缓冲
int right_buf[BUFFSIZE] = { 0 };//右输入缓冲
int* left_end = &left_buf[1024];//左缓冲区结尾
int* right_end = &right_buf[1024];//右缓冲区结尾
int* cur = right_end;//向前指针

string token = "";//存储字符串
int sign_num[7] = { 0 };//存储运算符、字符等数量
string error_type[2] = { "Error input","未识别结束符"};
string data_type[8] =
{ "identifier","constant","op","delimiters","keyword","string","character" };

string result = "";//记录当前识别结果
bool isForward = true;
int length = 1;
int line = 1;//记录当前行数
int line_num = 1;
int state = 0;//标识符状态，初始为 0
long ch_num = 0;//字符总数
```

```
ifstream demo;
int main()
{
    ofstream dtxt(DATA_FILE, ios::trunc);
    ofstream etxt(ERROR_FILE, ios::trunc);
    demo.open(TEST_FILE);

    type_assign();
    read_code(dtxt, etxt);
    print_inf();

    demo.close();
    dtxt.close();
    etxt.close();
    system("pause");
    return 0;
}

void read_code(ofstream& dtxt, ofstream& etxt)//读取源程序，并进行词法分析
{
    int cur_ch = get_ch();//读取字符
    LINEPLUS;
    int ch_type = cha[cur_ch];//判断类型
    int isFlag = 2;
    while (cur_ch != -1)
    {
        if (cur_ch == '#')
        {
            while (cur_ch != '\n')
            {
                cur_ch = get_ch();
                LINEPLUS;
            }
        }
        ch_type = cha[cur_ch];

        isForward = false;
        switch (state)
        {
        case 0://初始状态
        {
            switch (ch_type)
            {
```

```
        case 0:case 4:// ' ' ,'\n', '\t', '\r', 界符
        {
            state = 0;
            if ((char)cur_ch == '\"')//双引号
            {
                token += "\"";
                state = 10;
                isFlag = 1;
                break;
            }
            else if ((char)cur_ch == '\')//单引号
            {
                token += "\'";
                state = 11;
                break;
            }
            else if (ch_type == 4)
            {
                string tmp = "";
                tmp += (char)cur_ch;
                out_data(tmp, 3, dtxt);
            }
            cur_ch = get_ch();
            LINEPLUS;
            break;
        }
        case 1://进入识别标识符、关键字阶段
        {
            token += (char)cur_ch;
            state = 1;
            cur_ch = get_ch();
            LINEPLUS;
            break;
        }
        case 2://识别常数
        {
            token += (char)cur_ch;
            state = 2;
            cur_ch = get_ch();
            LINEPLUS;
            break;
        }
        case 3://运算符
        {
```

```
            token += (char)cur_ch;
            state = 8;
            cur_ch = get_ch();
            if (cur_ch == '*')
                state = 9;
            LINEPLUS;
            break;
        }
        default://发生错误，识别到了非字符集的输入，进行忽略
        {
            string tmp = "";
            state = 0;
            tmp += (char)cur_ch;
            out_error(tmp, 0, etxt);
            cur_ch = get_ch();
            LINEPLUS;
            break;
        }
        }
        break;
    }
    case 1://标识符、关键字
    {
        switch (ch_type)
        {
        case 1:case 2://标识符可有任意个 _ 和数字
        {
            token += (char)cur_ch;
            state = 1;
            cur_ch = get_ch();//从输入流读取一个字符
            LINEPLUS;
            break;
        }
        case 0:case 3:case 4://类型：界符，标识符识别结束
        {
            if (keyword.find(token) != keyword.end())//查找成功
            {
                out_data(token, 4, dtxt);
            }
            else//不是关键字，则是标识符
            {
                out_data(token, 0, dtxt);
            }
```

```
                token = "";
                state = 0;

                break;
            }
            default://发生错误，识别到非法字符，则将其视为空白符跳过
            {
                state = 1;
                string tmp = "";
                tmp += (char)cur_ch;
                out_error(tmp, 0, etxt);
                cur_ch = get_ch();//从输入流读取一个字符
                LINEPLUS;
                break;
            }

            }
            break;
        }
        case 2://常数
        {
            if ((char)cur_ch == '.')//识别小数状态
            {
                token += (char)cur_ch;

                state = 3;
                cur_ch = get_ch();
                LINEPLUS;
            }
            else if ((char)cur_ch == 'e' || (char)cur_ch == 'E')//
识别指数状态
            {
                token += (char)cur_ch;

                state = 5;
                cur_ch = get_ch();//从输入流读取一个字符
                LINEPLUS;
            }
            else if (ch_type == 2)//仍为此状态
            {
                token += (char)cur_ch;

                state = 2;
                cur_ch = get_ch();//从输入流读取一个字符
```

```
            LINEPLUS;
        }
        else if (ch_type == 4 || ch_type == 3)//界符或运算符
        {
            out_data(token, 1, dtxt);
            token = "";
            state = 0;
        }
        else//在识别常数时出现了错误字符，如字母，则将其当作界符
        {
            out_data(token, 1, dtxt);
            token = "";
            string tmp = "";
            tmp += (char)cur_ch;
            out_error(tmp, 0, etxt);

            state = 0;
            cur_ch = get_ch();//从输入流读取一个字符
            LINEPLUS;
        }
        break;
    }
    case 3://小数点后第一个数字
    {
        if (ch_type == 2)//识别到数字
        {
            token += (char)cur_ch;
            state = 4;
            cur_ch = get_ch();
            LINEPLUS;
        }
        else//小数点后没有识别到数字，则默认小数以 0 结尾继续识别
        {
            token += '0';
            state = 4;

            string tmp = "";
            tmp += (char)cur_ch;
            out_error(tmp, 0, etxt);
            cur_ch = get_ch();
            LINEPLUS;
        }
        break;
    }
```

```
        case 4://小数点后的其余数字
        {
            if ((char)cur_ch == 'e' || (char)cur_ch == 'E')//转到识别指数状态
            {
                token += (char)cur_ch;
                state = 5;
                cur_ch = get_ch();
                LINEPLUS;
            }
            else if (ch_type == 2)//继续识别数字
            {
                token += (char)cur_ch;
                state = 4;
                cur_ch = get_ch();
                LINEPLUS;
            }
            else if (ch_type == 4 || ch_type == 3)//运算符或界符
            {
                out_data(token, 1, dtxt);
                token = "";
                state = 0;
            }
            else//错误处理
            //视为界符
            {
                out_data(token, 1, dtxt);
                token = "";
                string tmp = "";
                tmp += (char)cur_ch;
                out_error(tmp, 0, etxt);
                state = 0;
                cur_ch = get_ch();//从输入流读取一个字符
                LINEPLUS;
            }
            break;
        }
        case 5://识别指数
        {
            if ((char)cur_ch == '+' || (char)cur_ch == '-')
            {
                token += (char)cur_ch;

                state = 6;
```

```
            cur_ch = get_ch();
            LINEPLUS;
        }
        else if (ch_type == 2)
        {
            token += (char)cur_ch;

            state = 7;
            cur_ch = get_ch();
            LINEPLUS;
        }
        else//未识别到有效的正负号和常数,则视为识别到+号
        {
            token += '+';
            string temp = "";
            temp += (char)cur_ch;
            out_error(temp, 0, etxt);

            state = 6;
            cur_ch = get_ch();
            LINEPLUS;
        }
        break;
    }
    case 6://识别指数 E 之后的符号之后
    {
        if (ch_type == 2)//继续识别数字
        {
            token += (char)cur_ch;

            state = 6;
            cur_ch = get_ch();
            LINEPLUS;
        }
        else if (ch_type == 4 || ch_type == 3)//遇到界符、运算符
        {
            out_data(token, 1, dtxt);
            token = "";
            state = 0;
        }
        else//读取到 非数字 字符，视为界符
        {
            out_data(token, 1, dtxt);
            token = "";
```

```
                string temp = "";
                temp += (char)cur_ch;
                out_error(temp, 0, etxt);
                state = 0;
                cur_ch = get_ch();
                LINEPLUS;
            }
            break;
        }
        case 7://识别指数次数
        {
            if (ch_type == 2)//遇到了常数
            {
                token += (char)cur_ch;

                state = 6;
                cur_ch = get_ch();
                LINEPLUS;
            }
            else//未识别到有效数字,则视为识别到 0
            {
                token += '0';

                string temp = "";
                temp += (char)cur_ch;
                out_error(temp, 0, etxt);

                state = 6;
                cur_ch = get_ch();//从输入流读取一个字符
                LINEPLUS;
            }
            break;
        }
        case 8://运算符和界符
        {
            length++;
            if (token == "/")//注释开始标志
            {
                if ((char)cur_ch == '*')//识别为注释
                {
                    state = 9;
                    cur_ch = get_ch();
                    LINEPLUS;
                    break;
```

```
                }
                else if ((char)cur_ch == '/')//第二类注释则去寻找换行符
                {
                    cur_ch = get_ch();
                    LINEPLUS;
                    while ((cur_ch != -1) && ((char)cur_ch != 
'\n'))
                    {
                        cur_ch = get_ch();
                        LINEPLUS;
                    }
                    //mark
                    token = "";
                    state = 0;
                    break;
                }
            }
            else if (token == "\"")//准备识别字符串
            {
                state = 10;
                isFlag = 1;
                cur_ch = get_ch();
                LINEPLUS;
                break;
            }
            else if (token == "\'")//准备识别字符
            {
                state = 11;
                cur_ch = get_ch();
                LINEPLUS;
                break;
            }

            if (length > 1 && ch_type != 0)
            {
                ch_type = 2;
            }

            switch (ch_type)
            {
            case 0:
            {
                state = 8;
                cur_ch = get_ch();//从输入流读取一个字符
```

```
                LINEPLUS;
                break;
            }
            case 3://类型：运算符，进入运算符识别模式
            {
                token += (char)cur_ch;
                state = 8;
                cur_ch = get_ch();//从输入流读取一个字符
                LINEPLUS;
                break;
            }
            case 1:case 2:case 4://遇到数字、标识符、界符均可认为识别已经
结束
            {
                length = 0;
                if (op.find(token) != op.end())
                {
                    out_data(token, 2, dtxt);
                }
                else if (delimiter.find(token) != delimiter.end())
                {
                    out_data(token, 3, dtxt);
                }
                else
                {
                    string tmp1 = "";
                    tmp1 += token[0];
                    string tmp2 = "";
                    tmp2 += token[1];
                    if (op.find(tmp1) != op.end() &&
op.find(tmp2) != op.end())
                    {
                        out_data(tmp1, 2, dtxt);
                        out_data(tmp2, 2, dtxt);
                    }
                }
                token = "";
                state = 0;

                break;
            }
            default://遇到非法字符 忽略
            {
                string tmp = "";
```

```
                tmp += (char)cur_ch;
                out_error(tmp, 0, etxt);
                state = 8;
                cur_ch = get_ch();
                LINEPLUS;
                break;
            }
            }
            break;
        }
        case 9://注释
        {
            while (1)
            {
                cur_ch = get_ch();
                LINEPLUS;
                if (cur_ch == -1)
                    break;
                else if ((char)cur_ch == '*')
                {
                    cur_ch = get_ch();
                    LINEPLUS;
                    if (cur_ch == -1)//文件结束
                        break;
                    else if ((char)cur_ch == '/')//注释结束
                    {
                        cur_ch = get_ch();
                        LINEPLUS;
                        ch_type = cha[cur_ch];
                        state = 0;
                        token = "";
                        break;
                    }
                    else
                        break;
                }
            }
            break;
        }
        case 10://识别字符串
        {
            while (1)
            {
                cur_ch = get_ch();
```

```
            LINEPLUS;
            if (cur_ch == -1)
                break;
            else if ((char)cur_ch == '\"')//识别到了另一半引号
            {
                token += (char)cur_ch;
                out_data(token, 5, dtxt);
                token = "";

                state = 0;
                cur_ch = get_ch();
                LINEPLUS;
                break;
            }
            else
            {
                if ((char)cur_ch == '\n'&&isFlag==1)
                {
                    isFlag = 2;
                     line--;
                    out_error(token, 1, etxt);
                    state = 0;
                    break;
                }
                token += (char)cur_ch;
            }
        }
        break;
    }
    case 11://识别字符
    {
        cur_ch = get_ch();
        LINEPLUS;
        token += cur_ch;
        if (cur_ch == '\\')
        {
            cur_ch = get_ch();
            token += cur_ch;
            LINEPLUS;
        }
        cur_ch = get_ch();
        LINEPLUS;
        if ((char)cur_ch == '\'')
        {
```

```
                token += '\'';
                out_data(token, 6, dtxt);
                token = "";
                state = 0;
                cur_ch = get_ch();
                LINEPLUS;
            }
            else//字符识别错误,抛出异常
            {
                out_error(token, 0, etxt);
                token += '\';//补救措施
                out_data(token, 6, dtxt);
                token = "";
                state = 0;
            }
            break;
        }
        default:
            exit(1);
        }
    }
}
int get_ch()//读取一个字符
{
    isForward = true;
    if (demo.peek() == EOF)
        ch_num++;
    if (cur == left_end)
    {
        cur = right_buf;
        while (!demo.eof() && cur != right_end)
        {
            char ch;
            demo >> noskipws >> ch;//读入所有字符
            if (demo.fail())
                break;
            *cur = ch;
            *cur++;
            if (ch == '\n' && isForward)
            {
                line_num++;
            }
        }
        if (cur != right_end)
```

```
            *cur = -1;
        cur = right_buf;
    }
    else if (cur == right_end)
    {
        cur = left_buf;
        while (!demo.eof() && cur != left_end)
        {
            char ch;
            demo >> noskipws >> ch;//读入所有字符
            if (demo.fail())
                break;
            if (ch == '\n'&&isForward)
                line_num++;
            *cur = ch;
            *cur++;
        }
        if (cur != left_end)
            *cur = -1;
        cur = left_buf;
    }
    return (int)(*cur++);
}
void out_data(string data, int data_sign, ofstream& txt)//数据类型
信息
{
    txt << "Line " << line << ":   ";
    txt << left << setw(10) << data;
    txt << "    Notation: " << data_type[data_sign] << endl;
    sign_num[data_sign]++;
}
void out_error(string data, int error_sign, ofstream& txt)//错误信
息
{
    txt << "Error in Line " << line << ":   ";
    txt << data;
    txt << "    Error type: " << error_type[error_sign] << endl;
}
void print_inf()//输出语句行数,各类单词、字符个数
{
    cout << "详细信息参见输出文件" << endl;
    cout << "字符总数: " << ch_num << endl;
    cout << "行数: " << line_num << endl;
    cout << "标识符数量: " << sign_num[0] << endl;
```

```
    cout << "常数数量：" << sign_num[1] << endl;
    cout << "运算符数量：" << sign_num[2] << endl;
    cout << "界符数量：" << sign_num[3] << endl;
    cout << "关键字数量：" << sign_num[4] << endl;
    cout << "字符串数量：" << sign_num[5] << endl;
    }
```