**词法分析程序实验报告**

班级：2014211302

学号：2014211168

姓名：周尧棋

**一、实验要求**

题目：词法分析程序的设计与实现。

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）、可以识别并读取源程序中的注释。

（3）、可以统计源程序汇总的语句行数、单词个数和字符个数，其中标点和空格不计算为单词，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行列位置。

（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，通过一次词法分析处理，可以检查并报告源程序中存在的所有错误。

实验要求：

方法1：采用C/C++作为实现语言，手工编写词法分析程序。

方法2：通过编写LEX源程序，利用LEX软件工具自动生成词法分析程序。

**二、程序设计说明**

1、使用输入缓冲区来保存输入符号串，输入缓冲区有两个指针，单词符号开始指针和向前指针。开始时，两个指针都指向下一个单词符号的第一个字符，向前指针向前扫描，直到单词符号可以被确定为止，一但单词被确定，向前指针置于该单词符号的右端。在处理完该单词符号后，让两个指针都指向下一个字符。

2、记号有下列种类：

(1)标识符

(2)C语言关键字

(3)符号数

(4)运算符

(5)标点符号

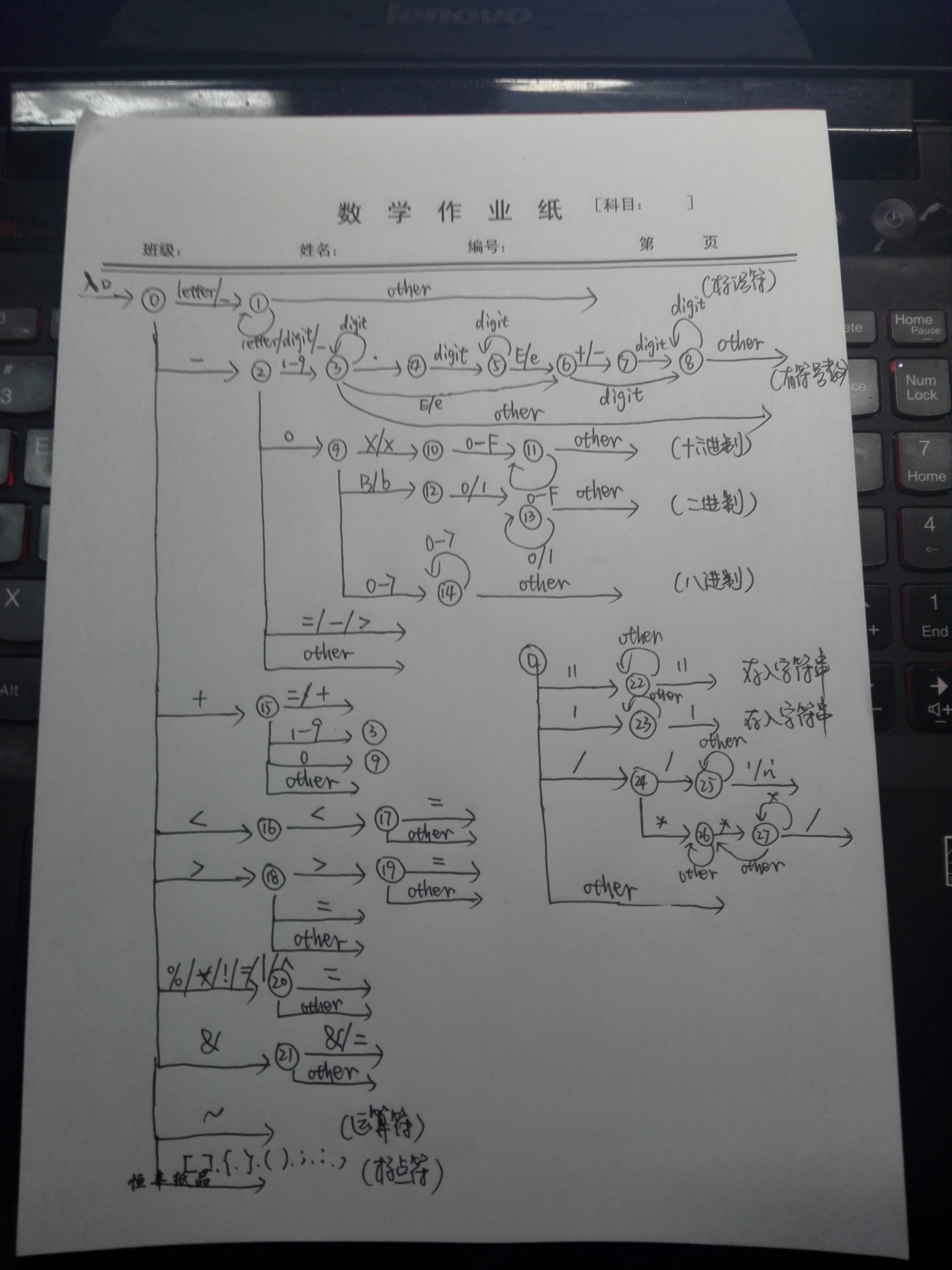
(6)注释

(7)字符串

3、程序输出

程序输出为<记号符，属性>对，如果发生错误则输出错误发生的行数。

4、程序的状态转换图



**三、源程序**

#include<iostream>

#include<cstdlib>

#include<cstring>

#include<vector>

#include<fstream>

#include<sstream>

#define KEYWORD 0 //关键字

#define ID 1 //标识符

#define RELOP 2 //运算符

#define NUM 3 //常数

#define LITERAL 4 //字符串

#define PUNC 5 //标点

#define ANNOTATION 6 //注释

#define MAXBUF 82 //缓冲区长度

#define L\_END 40 //缓冲区左半边终止位

#define R\_END 81 //缓冲区右半边终止位

#define START 0 //开始指针

using namespace std;

vector<string> key; //C语言的关键字表

class Lex

{

public:

string file; //需要词法分析的文件名

ifstream in;

char buffer[MAXBUF]; //输入缓冲区

int l\_end, r\_end; //左右半区的终点

int forward; //前进指针

bool flag\_l,flag\_r;

vector<string> id; //自己定义的标识符表

vector<string> keyword;

vector<string> num;

vector<string> literal; //""中的字符串

char C; //读入当前字符

int linenum, wordnum, charnum; //行数，单词数，字符数

string token; //存放当前的字符串

string change\_to\_string(char c); //将char->string

string change\_to\_string(int c); //将int->string

void get\_char(); //从输入缓冲区中读一个字符，放入C中，forward指向下一个

void get\_nbc(); //检查C中字符是否为空格，是则调用get\_char()，直到C进入非空格字符

void retract(); //向前指针后退一位

void initial(); //初始化

void fillBuffer(int a); //填充缓冲区，0左1右

void function(); //分析过程

void output(int type,string out); //以<记号，属性>的形式输出

void error(); //检查出错误，打印错误的行数

void operation(string f); //调用main()函数

};

void init\_key()

{

key.clear();

key.push\_back("auto"); key.push\_back("double"); key.push\_back("int"); key.push\_back("struct");

key.push\_back("break"); key.push\_back("else"); key.push\_back("long"); key.push\_back("switch");

key.push\_back("case"); key.push\_back("enum"); key.push\_back("register"); key.push\_back("typdef");

key.push\_back("char"); key.push\_back("extern"); key.push\_back("return"); key.push\_back("union");

key.push\_back("const"); key.push\_back("float"); key.push\_back("short"); key.push\_back("unsigned");

key.push\_back("continue"); key.push\_back("for"); key.push\_back("signed"); key.push\_back("void");

key.push\_back("default"); key.push\_back("goto"); key.push\_back("sizeof"); key.push\_back("volatile");

key.push\_back("do"); key.push\_back("if"); key.push\_back("static"); key.push\_back("while");

}

//判断字符是否为字母

bool isletter(char c)

{

if(c >= 'a' && c <= 'z' || c >= 'A' && c <= 'Z')

{

return true;

}

else

{

return false;

}

}

//判断字符是否为数字

bool isdigit(char c)

{

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

{

return true;

}

else

{

return false;

}

}

//判断标识符是否在C语言关键字表中

int iskey(string token)

{

vector<string>::iterator it;

for(it = key.begin(); it != key.end(); it++)

{

if(token == (\*it))

{

return it-key.begin();

}

}

return -1;

}

string Lex::change\_to\_string(char c)

{

string str;

stringstream stream;

stream << c;

str = stream.str();

return str;

}

string Lex::change\_to\_string(int c)

{

string str;

stringstream stream;

stream << c;

str = stream.str();

return str;

}

//初始化

void Lex::initial()

{

id.clear(); //清空标识符表

l\_end = L\_END; //初始化缓冲区

r\_end = R\_END;

forward = 0;

flag\_l = flag\_r = false;

buffer[l\_end] = buffer[r\_end] = EOF;

fillBuffer(0);

linenum = wordnum = charnum = 0; //初始化行数、单词数、字符数

}

void Lex::fillBuffer(int a)

{

if(a == 0) //填充缓冲区左半边

{

if(flag\_l == false)

{

in.read(buffer,l\_end);

if(in.gcount() != l\_end)

{

buffer[in.gcount()] = EOF;

}

}

else

{

flag\_l = false;

}

}

else

{

if(flag\_r == false)

{

in.read(buffer+l\_end+1,l\_end);

if(in.gcount() != l\_end)

{

buffer[in.gcount()+l\_end+1] = EOF;

}

}

else

{

flag\_r = false;

}

}

}

void Lex::get\_char()

{

C = buffer[forward];

if(C == EOF)

{

return;

}

if(C == '\n')

{

linenum++;

charnum++;

}

else

{

charnum++;

}

forward++;

if(buffer[forward] == EOF)

{

if(forward == l\_end)

{

fillBuffer(1);

forward++;

}

else if(forward == r\_end)

{

fillBuffer(0);

forward = START;

}

}

}

void Lex::get\_nbc()

{

while(C == ' ' || C == '\t' || C == '\n')

{

get\_char();

}

}

void Lex::retract()

{

if(forward == 0)

{

flag\_l = true; //已经读取过文件，不要再读取

forward = l\_end - 1;

}

else

{

forward--;

if(forward == l\_end)

{

flag\_r = true;

forward--;

}

}

}

//按<记号，属性>的格式输出

void Lex::output(int type,string out)

{

switch(type)

{

case KEYWORD:

cout << "<keyword," << atoi(out.c\_str()) << ">" << endl;

break;

case ID:

cout << "<id," << atoi(out.c\_str()) << ">" << endl;

break;

case NUM:

cout << "<num," << out << ">" << endl;

break;

case RELOP:

cout << "<" << out << ", >" << endl;

break;

case LITERAL:

cout << "<literal," << out << ">" << endl;

break;

case PUNC:

cout << "<" << out << ", >" << endl;

break;

case ANNOTATION:

cout << "<annotation," << out << ">" << endl;

break;

}

wordnum++;

}

void Lex::error()

{

cout << "Line: " << linenum+1 << " error!" << endl;

}

void Lex::function()

{

int state = 0;

do

{

switch(state)

{

case 0:

token.clear();

get\_char();

get\_nbc();

if(isletter(C) || C == '\_')

{

state = 1;

}

else if(C == '-')

{

state = 2;

}

else if(C >= '1' && C <= '9')

{

state = 3;

}

else if(C == '0')

{

state = 9;

}

else if(C == '+')

{

state = 15;

}

else if(C == '<')

{

state = 16;

}

else if(C == '>')

{

state = 18;

}

else if(C == '\*' || C == '&' || C == '%' || C == '!' || C == '=' || C == '|' || C == '^')

{

state = 20;

}

else if(C == '~' || C == '.')

{

output(RELOP,change\_to\_string(C));

state = 0;

}

else if(C == '{' || C == '}' || C == '[' || C == ']' || C == '(' || C == ')' || C == ';' || C == ':' || C == ',')

{

output(PUNC,change\_to\_string(C));

state = 0;

}

else if(C == '"')

{

state = 22;

}

else if(C == '\'')

{

state = 23;

}

else if(C == '/')

{

state = 24;

}

else if(C == '#')

{

state = 28;

}

else

{

state = 29;

}

break;

case 1:

token.push\_back(C);

get\_char();

if(isletter(C) || isdigit(C) || C == '\_')

{

state = 1;

}

else

{

retract();

state = 0;

if(iskey(token) != -1)

{

output(KEYWORD,change\_to\_string(iskey(token)));

}

else

{

id.push\_back(token);

int local = id.size() - 1;

output(ID,change\_to\_string(local));

state = 0;

}

}

break;

case 2:

token.push\_back(C);

get\_char();

if(C >= '1' && C <= '9')

{

state = 3;

}

else if(C == '0')

{

state = 9;

}

else if(C == '.')

{

state = 4;

}

else if(C == '=' || C == '-' || C == '>')

{

token.push\_back(C);

output(RELOP,token);

state = 0;

}

else

{

retract();

output(RELOP,token);

state = 0;

}

break;

case 3:

token.push\_back(C);

get\_char();

if(isdigit(C))

{

state = 3;

}

else if(C == '.')

{

state = 4;

}

else if(C == 'e' || C == 'E')

{

state = 6;

}

else

{

retract();

output(NUM,token);

state = 0;

}

break;

case 4:

token.push\_back(C);

get\_char();

if(isdigit(C))

{

state = 5;

}

else

{

error();

state = 0;

}

break;

case 5:

token.push\_back(C);

get\_char();

if(isdigit(C))

{

state = 5;

}

else if(C == 'e' || C == 'E')

{

state = 6;

}

else

{

retract();

output(NUM,token);

state = 0;

}

break;

case 6:

token.push\_back(C);

get\_char();

if(C == '+' || C == '-')

{

state = 7;

}

else if(isdigit(C))

{

state = 8;

}

else

{

retract();

output(NUM,token);

state = 0;

}

break;

case 7:

token.push\_back(C);

get\_char();

if(isdigit(C))

{

state = 8;

}

else

{

retract();

error();

state = 0;

}

break;

case 8:

token.push\_back(C);

get\_char();

if(isdigit(C))

{

state = 8;

}

else

{

retract();

output(NUM,token);

state = 0;

}

break;

case 9:

token.push\_back(C);

get\_char();

if(C == '.')

{

state = 4;

}

else if(C == 'X' || C == 'x')

{

state = 10;

}

else if(C == 'B' || C == 'b')

{

state = 12;

}

else if(C >= '0' && C <= '7')

{

state = 14;

}

else

{

retract();

output(NUM,token);

state = 0;

}

break;

case 10:

token.push\_back(C);

get\_char();

if((C >= '0' && C <= '9') || (C >= 'A' && C <= 'F') || (C >= 'a' && C <= 'f') )

{

state = 11;

}

else

{

retract();

error();

state = 0;

}

break;

case 11:

token.push\_back(C);

get\_char();

if((C >= '0' && C <= '9') || (C >= 'A' && C <= 'F') || (C >= 'a' && C <= 'f') )

{

state = 11;

}

else

{

retract();

output(NUM,token);

state = 0;

}

break;

case 12:

token.push\_back(C);

get\_char();

if(C == '0' || C == '1')

{

state = 13;

}

else

{

retract();

error();

state = 0;

}

break;

case 13:

token.push\_back(C);

get\_char();

if(C == '0' || C == '1')

{

state = 13;

}

else

{

retract();

output(NUM,token);

state = 0;

}

break;

case 14:

token.push\_back(C);

get\_char();

if(C >= '0' && C <= '7')

{

state = 14;

}

else

{

retract();

output(NUM,token);

state = 0;

}

break;

case 15:

token.push\_back(C);

get\_char();

if(C >= '1' && C <= '9')

{

state = 3;

}

else if(C == '0')

{

state = 9;

}

else if(C == '=' || C == '+')

{

token.push\_back(C);

output(RELOP,token);

state = 0;

}

else

{

retract();

output(RELOP,token);

state = 0;

}

break;

case 16:

token.push\_back(C);

get\_char();

if(C == '<')

{

state = 17;

}

else if(C == '=')

{

token.push\_back(C);

output(RELOP,token);

state = 0;

}

else

{

retract();

output(RELOP,token);

state = 0;

}

break;

case 17:

token.push\_back(C);

get\_char();

if(C == '=')

{

token.push\_back(C);

output(RELOP,token);

state = 0;

}

else

{

retract();

output(RELOP,token);

state = 0;

}

break;

case 18:

token.push\_back(C);

get\_char();

if(C == '>')

{

state = 19;

}

else if(C == '=')

{

token.push\_back(C);

output(RELOP,token);

state = 0;

}

else

{

retract();

output(RELOP,token);

state = 0;

}

break;

case 19:

token.push\_back(C);

get\_char();

if(C == '=')

{

token.push\_back(C);

output(RELOP,token);

state = 0;

}

else

{

retract();

output(RELOP,token);

state = 0;

}

break;

case 20:

token.push\_back(C);

get\_char();

if(C == '=')

{

token.push\_back(C);

output(RELOP,token);

state = 0;

}

else

{

retract();

output(RELOP,token);

state = 0;

}

break;

case 21:

token.push\_back(C);

get\_char();

if(C == '&')

{

token.push\_back(C);

output(RELOP,token);

state = 0;

}

else if(C == '=')

{

token.push\_back(C);

output(RELOP,token);

state = 0;

}

else

{

retract();

output(RELOP,token);

state = 0;

}

break;

case 22:

get\_char();

if(C == '"')

{

output(LITERAL,token);

state = 0;

}

else

{

token.push\_back(C);

state = 22;

}

break;

case 23:

get\_char();

if(C == '\'')

{

output(LITERAL,token);

state = 0;

}

else

{

token.push\_back(C);

state = 23;

}

break;

case 24:

get\_char();

if(C == '/')

{

state = 25;

}

else if(C == '\*')

{

state = 26;

}

else if(C == '=')

{

token.push\_back(C);

output(RELOP,token);

state = 0;

}

else

{

retract();

error();

state = 0;

}

break;

case 25:

get\_char();

if(C == '\n')

{

output(ANNOTATION,token);

state = 0;

}

else

{

token.push\_back(C);

state = 25;

}

break;

case 26:

get\_char();

if(C == '\*')

{

state = 27;

}

else

{

token.push\_back(C);

state = 26;

}

break;

case 27:

get\_char();

if(C == '\*')

{

token.push\_back('\*');

state = 27;

}

else if(C == '/')

{

output(ANNOTATION,token);

state = 0;

}

else

{

token.push\_back('\*');

token.push\_back(C);

state = 26;

}

break;

case 28:

while(C != '\n')

{

get\_char();

}

state = 0;

break;

case 29:

cout << "Line: " << linenum << " error!" << endl;

// system("pause");

state = 0;

break;

}

}while(C != EOF);

}

void Lex::operation(string f)

{

char filename[20];

file = f;

strcpy(filename,file.c\_str());

in.open(filename);

if(!in)

{

cout << "文件不存在！" << endl;

}

else

{

initial();

function();

cout << endl;

cout << "linenum:" << linenum << endl;

cout << "wordnum:" << wordnum << endl;

cout << "charnum:" << charnum << endl;

}

}

int main()

{

init\_key();

string filename;

cout << "输入文件名：" ;

cin >> filename;

Lex test;

test.operation(filename);

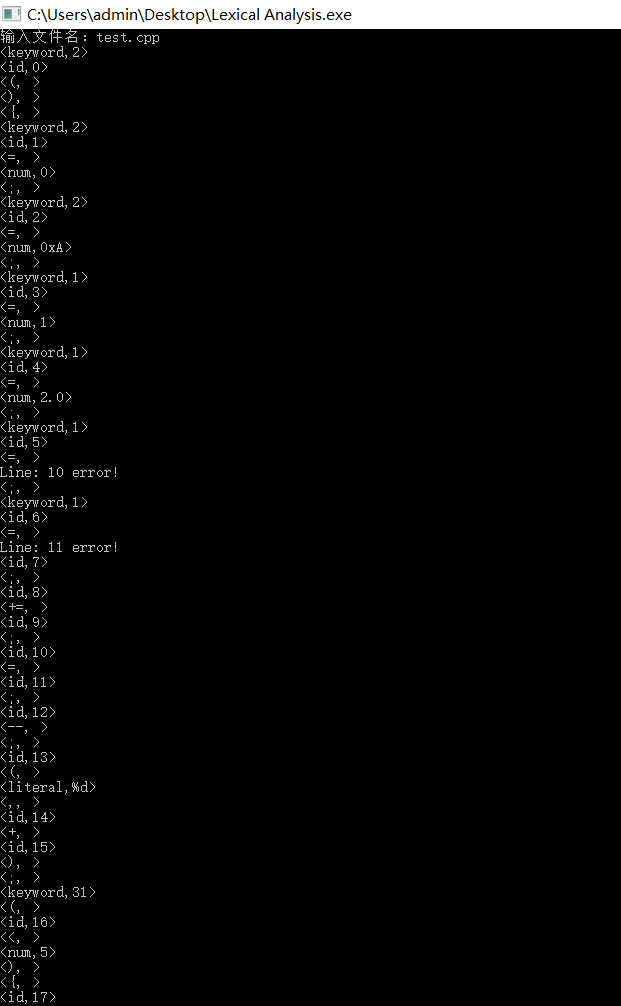
system("pause");

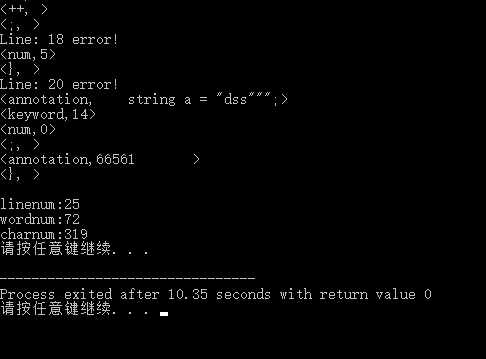
return 0;

}

**四、程序测试说明**

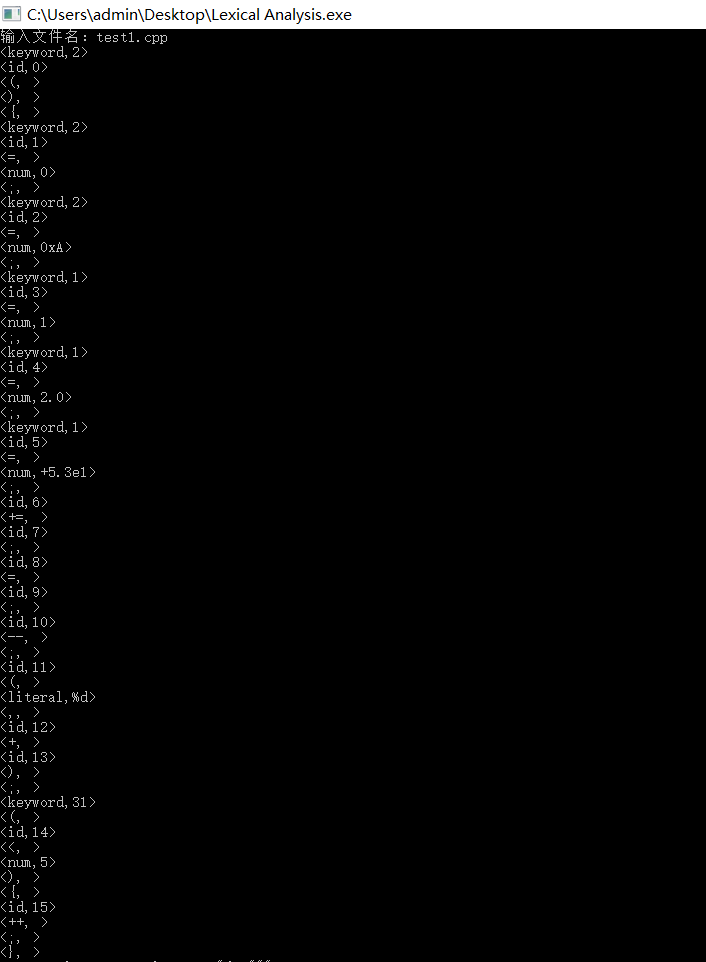
test.cpp：

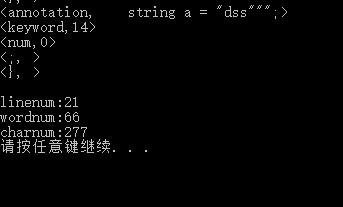




在test.cpp中第10行中，有符号数中e后面没有数字，故提示错误，第11行还是有符号数中-的后面应该接数字，但是却接了字母，所以分析器报错，在第20行出现了非法字符，报错。输出则为<记号名，属性>的形式，将读取的单词输出，此外，""、''和注释中的内容单独输出。

test1.cpp：





在test1.cpp中，在编译器中能够编译运行，在自己的程序中也没有提示错误，程序正确。