**词法分析程序的设计与实现**

**实**

**验**

**报**

**告**

**课程名称 编译原理与技术**

**指导老师 刘辰**

**班 级 2016211310**

**姓 名 张绍磊**

**学 号 2016211392**

目录

[一、实验目的 3](#_Toc528858508)

[二、实验内容 3](#_Toc528858509)

[三、实验环境 4](#_Toc528858510)

[四、程序编写思路及原理 4](#_Toc528858511)

[（1）单词符号的类别： 4](#_Toc528858512)

[（2）记号的正规表达式及属性 5](#_Toc528858513)

[（3）词法分析器的输出形式 6](#_Toc528858514)

[（4）程序运行流程 6](#_Toc528858515)

[五、状态转移图 7](#_Toc528858516)

[六、词法分析程序流程图 8](#_Toc528858517)

[七、具体实现 9](#_Toc528858518)

[1、变量定义及声明 9](#_Toc528858519)

[2、识别程序字符的读取 9](#_Toc528858520)

[3、读取字符的函数 10](#_Toc528858521)

[4、过滤空格的函数 11](#_Toc528858522)

[5、读取单个字符是数字或字母 11](#_Toc528858523)

[6、连接函数 12](#_Toc528858524)

[7、错误处理函数 12](#_Toc528858525)

[8、关键字的识别 13](#_Toc528858526)

[9、数字的识别 13](#_Toc528858527)

[10、单词的识别 17](#_Toc528858528)

[11、注释的识别 18](#_Toc528858529)

[12、运算符的识别 20](#_Toc528858530)

[13、转义字符的识别 24](#_Toc528858531)

[14、输出 26](#_Toc528858532)

[八、程序测试与分析 27](#_Toc528858533)

[九、实验总结 30](#_Toc528858534)

[附：源代码 31](#_Toc528858535)

# 一、实验目的

通过本实验的编程实践，使学生了解词法分析的任务，掌握词法分析程序设计的原理和构造方法，使学生对编译的基本概念、原理和方法有完整的和清楚的理解，并能正确地、熟练地运用。

# 二、实验内容

设计并实现C语言的词法分析程序，要求实现如下功能：

(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v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。

编写一个词法分析程序，要求从左到右进行扫描和分解，根据词法规则，统计语句行数、单词个数、总字符个数，识别出一个个具有独立意义的单词符号以供语法分析之用。若发现词法错误，则返回出错信息。

# 三、实验环境

Windows 10 PC机；

Microsoft Visual Studio 2017；

# 四、程序编写思路及原理

## （1）单词符号的类别：

单词符号是程序语言最基本的语法符号，为了便于语法分析，通常将单词符号分为五类。

**1、 标识符**

用来命名程序中出现的变量、数组、函数、过程、标号等，通常是一个字母开头的字母数字串，如length，nextch等。

**2、 基本字**

也可以成为关键字或保留字。如if，while，for，do，goto等。他们具有标识符的形式，但他们不是由用户而是由语言定义的，其意义是约定的。多数语言中规定，他们不能作为标识符或者标识符的前缀，即用户不能使用它们来定义用户使用的名字，故我们称它为保留字，这些语言如Pascal和C等。但也有的语言允许将基本字作为标识符或者标识符的前缀，这类语言如Fortran等。

**3、 常数**

包括各种类型的常数，如整型、实型、字符型、布尔型等。如：5、3.1415926、’a’、TRUE等都是常数。

**4、 运算符**

算术运算符+、-、×、÷；关系运算符<,<=,>,>=,==,!=以及逻辑运算符&&，()，||或者!等。

**5、 界符**

如”，”、”；”等单字界符和/,/,//等双字界符，空白符等。

对于一个程序语言来说，基本字、运算符、界符的数目是确定的，通常在几十个到几百个之间。标识符，常数则由用户定义，如何指定，指定多少，程序语言未加限制，但规定了他们应满足的构词规则。

## （2）记号的正规表达式及属性

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 正规表达式 | 记号 | 属性 | 正规表达式 | 记号 | | 属性 |
|  |  |
| C语言的32个保留字 | iskey | - | >= | relop | | GE |
| 用户标识符 | ID | 符号表入口指针 | / | ARITH | | - |
| 数字 | NUM | 数值 | + | ARITH | | - |
| < | relop | LT | - | ARITH | | - |
| <= | relop | LE | \* | ARITH | | - |
| = | assign\_op | - | /,(,),&,%,$,#,[,],{,}等标点符号 | PUNC | | - |
| != | relop | NE |  |  | |  |
| > | relop | GT |  |  | |  |
| == | relop | EQ |  |  | |  |

## （3）词法分析器的输出形式

识别出来的单词应该采用某种中间表示形式，以便为编译后续阶段方便地引用。通常一个单词用一个二元式来表示：

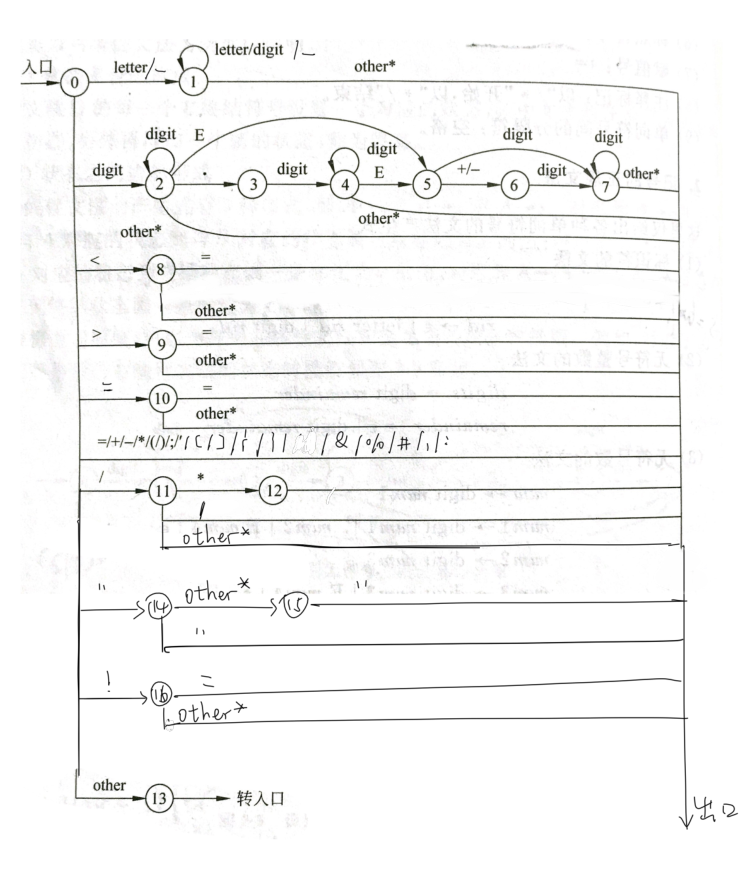
第一元用于区分单词所属的类别，以整数编码表示。第二元用于区分该类别中的哪一个单词符号，即单词符号的值。

单词的编码随类别不同而不同。由于基本字、运算符、界符的数目是确定的，一般每个单词可以定义一个类别码，单词与它的类别码为一一对应的关系，即一字一码。这时它的第二元就没有识别意义了，显然对这类单词的识别很简单。也可以将关系运算符全部归为一类，用第二元的值来区分是哪一个关系运算符，这种分类在一定程度上可以简化以后的语法分析。常数可归为一类，也可按整型，实型，字符型，布尔型等分类，标识符类似处理。在这种情况下，每一类别中的常数或标识符将由第二元单词的属性值来区别。通常将常数在常数表中的位置编号作为常数的属性值，从而将标识符在符号表中的位置编号作为标识符的属性值。

## （4）程序运行流程

每次读取一个字符，若读到的是空格，则跳过，继续读字符，直到读到非空字符为止，根据读到的字符，选择下一个标记与其相匹配的下一个状态，选择相应的程序段进行处理和识别。

# 五、状态转移图

****

其中，状态0是初始状态，若此时读人的字符是字母，则转换到状态1，进入标识符识別过程；如果读入的字符是数字，则转换到状态2，逬入无符号数识别过程；……；若读入的字符是“/”，转换到状态11，再读入下一个字符，如果读入的是“\*”则转换到状态12，进入注释处理状态；如果在状态0读入的字符不是语言所定义的单词符号的首字符，则转换到状态13，进入错误处理状态。

# 六、词法分析程序流程图

词法分析程序流程图如下图所示：



# 七、具体实现

## 1、变量定义及声明

ifstream file;*//源文件*

ofstream ofile;*//识别文件*

ofstream errorlist;*//错误分析文件*

*//行数、当前行数、当前列数、字符数*

**int** rownum = 1, nowrow = 1, column = 0, characternum = 0;

*//token数组的长度 allword数组的长度 allnum数组的长度*

**int** tokennum, wordnum = 0, digitnum = 0, searchwordnum, searchnnum;

*//向前指针 注释数目*

**int** forword = 0, notenum=0;

**char** ch = ' ', buffer[4095];

*//关键字数组*

**char** keyword[32][20] =

{ "include","main","int","float","double","char","long","bool","short",

"if","else","for","while","do","struct","typedef","const","default",

"return","case","switch","break","continue","enum","goto","sizeof",

"static","void","union","unsigned","signed","extern" };

*//所有单词的数组 所有数字的数组*

**char** allword[512][32], allnum[512][32];

*//字符数组 注释数组*

**char** token[512], note[1024], getstring[1024];

## 2、识别程序字符的读取

设计了void inbuffer(int pointer)函数，读取字符放入buffer()数组，每次都读取缓冲区容量的一半，便于识别单词。

*//读取字符放入buffer()数组，每次都读取缓冲区容量的一半*

**void** inbuffer(**int** pointer)

{

**int** i = 0;

**char** ch1;

**while** (!file.eof() && i<2048) {

file.read(&ch1, 1);*//读入一个字符*

buffer[pointer+i] = ch1;

**if** (ch1 != ' ') {

**if** (ch1 == '\n')

rownum++; *//行数+1*

**else**

characternum++; *//字符+1*

}

i++;

}

**if** (file.eof())

buffer[pointer+i] = '\0'; *//源程序读取完毕*

}

## 3、读取字符的函数

void get\_char()，读取函数，每次从缓冲区中读出当前指针对应的字符。

*//读取函数，每次从缓冲区中读出当前指针对应的字符*

**void** get\_char()

{

ch = buffer[forword];

column++;*//当前列数增加*

**if** (forword == 2047)

{

inbuffer(2048);

forword++;

}

**else** **if** (forword == 4095)

{

inbuffer(0);

forword = 0;

}

**else**

forword++;

}

## 4、过滤空格的函数

void deletespace()，对要识别的程序进行预处理，将空格全部过滤掉。

*//过滤掉读到的空格*

**void** deletespace()

{

**if** (ch == ' ')

{

ch = buffer[forword];

column++;

**if** (forword == 2047)

{

inbuffer(2048);

forword++;

}

**else** **if** (forword == 4095)

{

inbuffer(0);

forword = 0;

}

**else**

forword++;

deletespace();

}

}

## 5、读取单个字符是数字或字母

通过int digit()判断字符是否为数字。

*//判断数字的函数*

**int** digit()

{

**if** ((ch >= 48 && ch <= 57))

**return** 1;

**else**

**return** 0;

}

通过int letter()判断字符是否为字母。

*//判断字母的函数*

**int** letter()

{

**if** ((ch >= 97 && ch <= 122) || (ch >= 65 && ch <= 90))

**return** 1;

**else**

**return** 0;

}

## 6、连接函数

void cat()，连接函数，把当前字符与token中的字符串连接起来

*//连接函数，把当前字符与token中的字符串连接起来*

**void** cat()

{

token[tokennum] = ch;

tokennum++;

}

## 7、错误处理函数

void error()，可以处理的错误有：

（1）非法数字的识别，比如5.E；

（2）单个引号的识别，比如printf(“错误处理)；

（3）除了数字、单词、关键字、转义字符、注释之外，不符合以上合理字符的所有错误；

*//错误处理程序*

**void** error()

{

ofile << "词法分析程序发现错误：位于第" << nowrow << "行，第" << column << "列" << endl;

cout << endl;

cout << "词法分析程序发现错误：位于第" << nowrow << "行，第" << column << "列" << endl;

errorlist << "词法分析程序发现错误：位于第" << nowrow << "行，第" << column << "列" << endl;

}

## 8、关键字的识别

本程序可以识别关键字，建立了一个关键字的二维数组keyword[32][20]，存储了所有关键字，调用int reserve()函数，可以识别是否为关键字，若是返回1，不是则返回0。

*//判断当前单词是否为保留的关键字，若是返回1，不是则返回0*

**int** reserve()

{

**for** (**int** i = 0; i<32; i++)

{

**if** (strcmp(token, keyword[i]) == 0)

**return** 1;

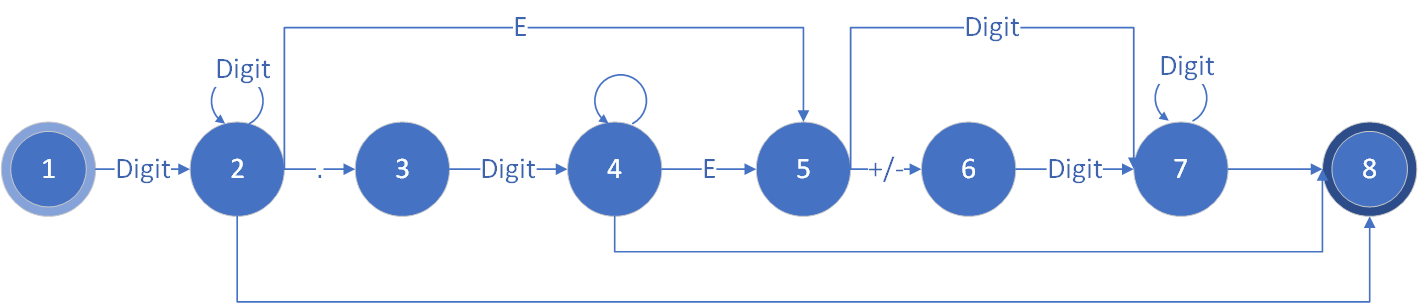
}

**return** 0;

}

## 9、数字的识别

因为合法数字判断条件太多，因此画出其状态转移图：



*//单词处理*

**case**'a':**case**'b':**case**'c':**case**'d':**case**'e':**case**'f':**case**'g':

**case**'h':**case**'i':**case**'j':**case**'k':**case**'l':**case**'m':**case**'n':

**case**'o':**case**'p':**case**'q':**case**'r':**case**'s':**case**'t':**case**'u':

**case**'v':**case**'w':**case**'x':**case**'y':**case**'z':**case**'A':**case**'B':

**case**'C':**case**'D':**case**'E':**case**'F':**case**'G':**case**'H':**case**'I':

**case**'J':**case**'K':**case**'L':**case**'M':**case**'N':**case**'O':**case**'P':

**case**'Q':**case**'R':**case**'S':**case**'T':**case**'U':**case**'V':**case**'W':

**case**'X':**case**'Y':**case**'Z':**case**'\_':

tokennum = 0;

**while** (letter() || digit() || ch == '\_')

{

cat();

get\_char();

**if** (ch == '\n') nowrow--;

}

token[tokennum] = '\0';*//添加末尾符号否则出现乱码*

forword--;

result = reserve();

**if** (result == 0)

{

iskey = findidword();*//看看当前单词是否已经被放入单词数组*

**if** (iskey == 0)*// 如果没放入*

{

insertallword();

ofile << "< id , " << wordnum - 1 << " >:" << token << endl;

*//cout << "<id," << wordnum - 1 << ">:" << token << endl;*

}

**else**

{

ofile << "< id , " << searchwordnum << " >:" << token << endl;

*//cout << "<id," << searchwordnum << ">:" << token << endl;*

}

}

**else**

{

ofile << "关键字：" << token << endl;

*//cout << "关键字:" << token << endl;*

}

**break**;

调用int digit()可以识别是否是数字，然后通过如下代码段，能判断出是否是合法数字：

*//数字处理 ，依循书上的处理方法*

**case** '0':**case**'1':**case**'2':**case**'3':**case**'4':**case**'5':

**case** '6':**case**'7':**case**'8':**case**'9':

tokennum = 0;

**while** (digit())

{

cat();

get\_char();

}

**if** (ch == '.')

{

cat();

get\_char();

**if** (!digit())

{

error();*//判断2.E这种类型的错误*

**break**;

}

**else**

{

**while** (digit())

{

cat();

get\_char();

}

}

}

**if** (ch == 'E')*//2.3E这种可以被识别*

{

cat();

get\_char();

**if** (ch == '+' || ch == '-')*//E的后面可加+号也可以加-号*

{

cat();

get\_char();

**if** (!digit())

{

error();

**break**;

}

**else** {

**while** (digit())

{

cat();

get\_char();

}

}

}

**else** **if** (digit())

{

**while** (digit())

{

cat();

get\_char();

}

}

}

token[tokennum] = '\0';

forword--;

iskey = findnumword();

**if** (iskey == 0)

{

insertallnum();

ofile << "< num ," << digitnum - 1 << " >:" << token << endl;

*//cout << "<num," << digitnum - 1 << ">:" << token << endl;*

}

**else**

{

ofile << "< num ," << searchnnum << " >:" << token << endl;

*//cout << "<num," << searchnnum << ">:" << token << endl;*

}

**break**;

## 10、单词的识别

调用int letter()来识别出是否是字母，然后通过如下代码段识别是否是单词：

*//单词处理*

**case**'a':**case**'b':**case**'c':**case**'d':**case**'e':**case**'f':**case**'g':

**case**'h':**case**'i':**case**'j':**case**'k':**case**'l':**case**'m':**case**'n':

**case**'o':**case**'p':**case**'q':**case**'r':**case**'s':**case**'t':**case**'u':

**case**'v':**case**'w':**case**'x':**case**'y':**case**'z':**case**'A':**case**'B':

**case**'C':**case**'D':**case**'E':**case**'F':**case**'G':**case**'H':**case**'I':

**case**'J':**case**'K':**case**'L':**case**'M':**case**'N':**case**'O':**case**'P':

**case**'Q':**case**'R':**case**'S':**case**'T':**case**'U':**case**'V':**case**'W':

**case**'X':**case**'Y':**case**'Z':**case**'\_':

tokennum = 0;

**while** (letter() || digit() || ch == '\_')

{

cat();

get\_char();

**if** (ch == '\n') nowrow--;

}

token[tokennum] = '\0';*//添加末尾符号否则出现乱码*

forword--;

result = reserve();

**if** (result == 0)

{

iskey = findidword();*//看看当前单词是否已经被放入单词数组*

**if** (iskey == 0)*// 如果没放入*

{

insertallword();

ofile << "< id , " << wordnum - 1 << " >:" << token << endl;

*//cout << "<id," << wordnum - 1 << ">:" << token << endl;*

}

**else**

{

ofile << "< id , " << searchwordnum << " >:" << token << endl;

*//cout << "<id," << searchwordnum << ">:" << token << endl;*

}

}

**else**

{

ofile << "关键字：" << token << endl;

*//cout << "关键字:" << token << endl;*

}

**break**;

## 11、注释的识别

由于合法注释有两种，第一种是//开头的单行注释，另外一种是以/\* …….\*/中间内容为主的注释，识别方法不同，因此有两段不同的代码用以识别不同类型的注释：

（1）识别//类型的注释

**case** '/':

get\_char();

**if** (ch == '/')

{

ofile << "< '//' , note >" << endl;

notenum = 0;

**char** note1[1024];

**while** (ch != '\n')

{

get\_char();

note1[notenum++] = ch;

}

note1[notenum++] = '\0';

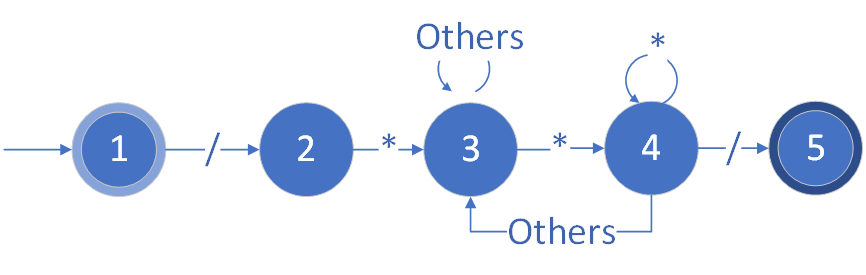
ofile << "注释如下：" << note1;

**break**;

}

（2）识别/\*……\*/

过程比较繁琐，画出状态转移图如下所示：



由状态转移图写出代码如下：

**case** '/':

get\_char();

**if** (ch == '\*')

{

ofile << "< '/\*' , note >" << endl;

notenum = 0;

takingnote();

**while** (ch != '/') takingnote();

note[notenum - 1] = ' ';

note[notenum - 2] = ' ';

ofile << "注释如下：" << note << endl;

ofile << "< '\*/' , note >" << endl;

**break**;

}

识别/\*……\*/这种类型注释的时候，难以处理的是/\*…….\*/中的\*，因此得循环处理，代码段如下：

*//识别/\*开头的注释，需要循环处理*

**void** takingnote()

{

get\_char();

note[notenum++] = ch;

**while** (ch != '\*')

{

get\_char();

note[notenum++] = ch;

}

get\_char();

note[notenum++] = ch;

}

## 12、运算符的识别

本程序可识别C语言中所有合法运算符，包括==，+=，&&，||等字符，具体识别过程如下：

（1）识别以<开头的所有符号

*//识别以<开头的所有符号*

**case** '<':

get\_char();

**if** (ch == '=')

{

ofile << "< relop , LE(<=) >" << endl;

*//cout << "<relop,LE(<=)>" << endl;*

}

**else** **if** (ch == '>')

{

ofile << "< relop , NE(<>) >" << endl;

*//cout << "<relop,NE(<>)>" << endl;*

}

**else**

{

forword--;

ofile << "< relop , LT(<) >" << endl;

*//cout << "<relop,LT(<)>" << endl;*

}

**break**;

（2）识别以=开头的所有符号

*//识别以=开头的所有符号*

**case** '=':

get\_char();

**if** (ch == '=')

{

ofile << "< relop, EQ(==) >" << endl;

*//cout << "<relop,EQ(==)>" << endl;*

}

**else**

{

forword--;

ofile << "< relop , ass(=) >" << endl;

*//cout << "<relop,ass(=)>" << endl;*

}

**break**;

（3）识别以>开头的所有符号

*//识别以>开头的所有符号*

**case** '>':

get\_char();

**if** (ch == '=')

{

ofile << "< relop , GE(>=) >" << endl;

*//cout << "<relop,GE(>=)>" << endl;*

}

**else**

{

forword--;

ofile << "< relop , GT(>) >" << endl;

*//cout << "<relop,GT(>)>" << endl;*

}

**break**;

（4）识别以:开头的所有符号

**case** ':':

get\_char();

**if** (ch == '=')

{

ofile << "< assign-op , -(:=) >" << endl;

*//cout << "<assign-op,-(:=)>" << endl;*

}

**else**

{

forword--;

ofile << "< ':' , - >" << endl;

*//cout << "<':',->" << endl;*

}

**break**;

（5）识别以+开头的所有符号

**case** '+':

get\_char();

**if** (ch == '=')

{

ofile << "< assignop , assad(+=) >" << endl;

*//cout << "<assignop,assad(+=)>" << endl;*

}

**else** **if** (ch == '+')

{

ofile << "< '++' , - >" << endl;

*//cout << "<'++',->" << endl;*

}

**else**

{

forword--;

ofile << "< '+' , - >" << endl;

*//cout << "<'+',->" << endl;*

}

**break**;

（6）识别以-开头的所有符号

**case** '-':

get\_char();

**if** (ch == '=')

{

ofile << "< assignop , assub(-=) >" << endl;

*//cout << "<assignop,assub(-=)>" << endl;*

}

**else** **if** (ch == '-')

{

ofile << "< '--' , - >" << endl;

*//cout << "<'--',->" << endl;*

}

**else**

{

forword--;

ofile << "< '-' , - >" << endl;

*//cout << "<'-',->" << endl;*

}

**break**;

（7）识别其他符号

**case** '\*':

get\_char();

**if** (ch == '=')

ofile << "< assignop , assmul(\*=) >" << endl;

**else**

{

forword--;

ofile << "< '\*' , - >" << endl;

}

**break**;

**case** '(':

ofile << "< '(' , - >" << endl;

**break**;

**case** ')':

ofile << "< ')' , - >" << endl;

**break**;

**case** ';':

ofile << "< ';' , - >" << endl;

**break**;

**case** '%':

ofile << "< '%' , - >" << endl;

**break**;

**case** '[':

ofile << "< '[' , - >" << endl;

**break**;

**case** ']':

ofile << "< ']' , - >" << endl;

**break**;

**case** '{':

ofile << "< '{' , - >" << endl;

**break**;

**case** '}':

ofile << "< '}' , - >" << endl;

**break**;

**case** ',':

ofile << "< ',' , - >" << endl;

**break**;

**case** '#':

ofile << "< '#' , - >" << endl;

**break**;

**case** '.':

ofile << "< '.' , - >" << endl;

**break**;

**case** '?':

ofile << "< '?' , - >" << endl;

**break**;

**case** '!':

get\_char();

**if** (ch == '=')

ofile << "< relop , (!=) >" << endl;

**else**

{

forword--;

ofile << "< logicop , not(!) >" << endl;

}

**break**;

**case** '|':

get\_char();

**if** (ch == '|')

ofile << "< logicop , or(||) >" << endl;

**break**;

**case** '&':

get\_char();

**if** (ch == '&')

ofile << "< logicop , and(&&) >" << endl;

**else**

{

forword--;

ofile << "< '&' , - >" << endl;

}

**break**;

## 13、转义字符的识别

本程序能够识别所有的转义字符，具体实现如下所示：

**case** '\\':

ofile << "< esc , \ >" << endl;

**break**;

**case** '\a':

ofile << "< esc , BEl()响铃 >" << endl;

**break**;

**case** '\b':

ofile << "< esc , BS(退格) >" << endl;

**break**;

**case** '\f':

ofile << "< esc , FF(换页) >" << endl;

**break**;

**case** '\n':

ofile << "< esc , LF(换行) >" << endl;

nowrow++;

column = 0;

**break**;

**case** '\r':

ofile << "< esc , CR(回车) >" << endl;

**break**;

**case** '\t':

ofile << "< esc , HT(水平制表) >" << endl;

**break**;

**case** '\v':

ofile << "< esc , VT(垂直制表) >" << endl;

**break**;

**case** '\"':

ofile << "< esc , 双引号字符 >" << endl;

get\_char();

iskey = 0, result = 1;

getstring[128];

**while** (ch != '\"')

{

getstring[iskey] = ch;

**if** (ch == '\n')

{

error();

forword--;

result = 0;

**break**;

}

get\_char();

iskey++;

}

**for** (**int** i = iskey; i <= 100; i++) getstring[i] = ' ';

**if** (result == 1)*//判断是否为字符串*

ofile << "字符串：" << getstring << endl;

**else**

ofile << "出错字符串：" << getstring << endl;

ofile << "< esc, 双引号字符 >" << endl;

**break**;

**case** '\0':

ofile << "< esc , NULL(空字符) >" << endl;

**break**;

**case** '\'':

ofile << "< esc , '''(单引号) >" << endl;

**break**;

## 14、输出

本词法分析程序将语句行数、单词个数、字符个数输出到屏幕。

具体分析过程输出至文件“程序分析结果.txt”中。

错误分析输出至文件“错误分析报告.txt”中。

ofile << "分析程序中的语句行数为：" << rownum << endl;

cout << "分析程序中的语句行数为：" << rownum << endl;

ofile << "分析程序中的单词个数为：" << wordnum << endl;

cout << "分析程序中的单词个数为：" << wordnum << endl;

ofile << "源程序中字符个数为：" << characternum << endl;

cout << "源程序中字符个数为：" << characternum << endl;

cout << endl;

cout << "具体分析过程，已输出到文件‘程序分析结果.txt’中，请查看。" << endl;

cout << "错误分析，已输出到文件‘错误分析报告.txt’中，请查看。" << endl;

cout << endl;

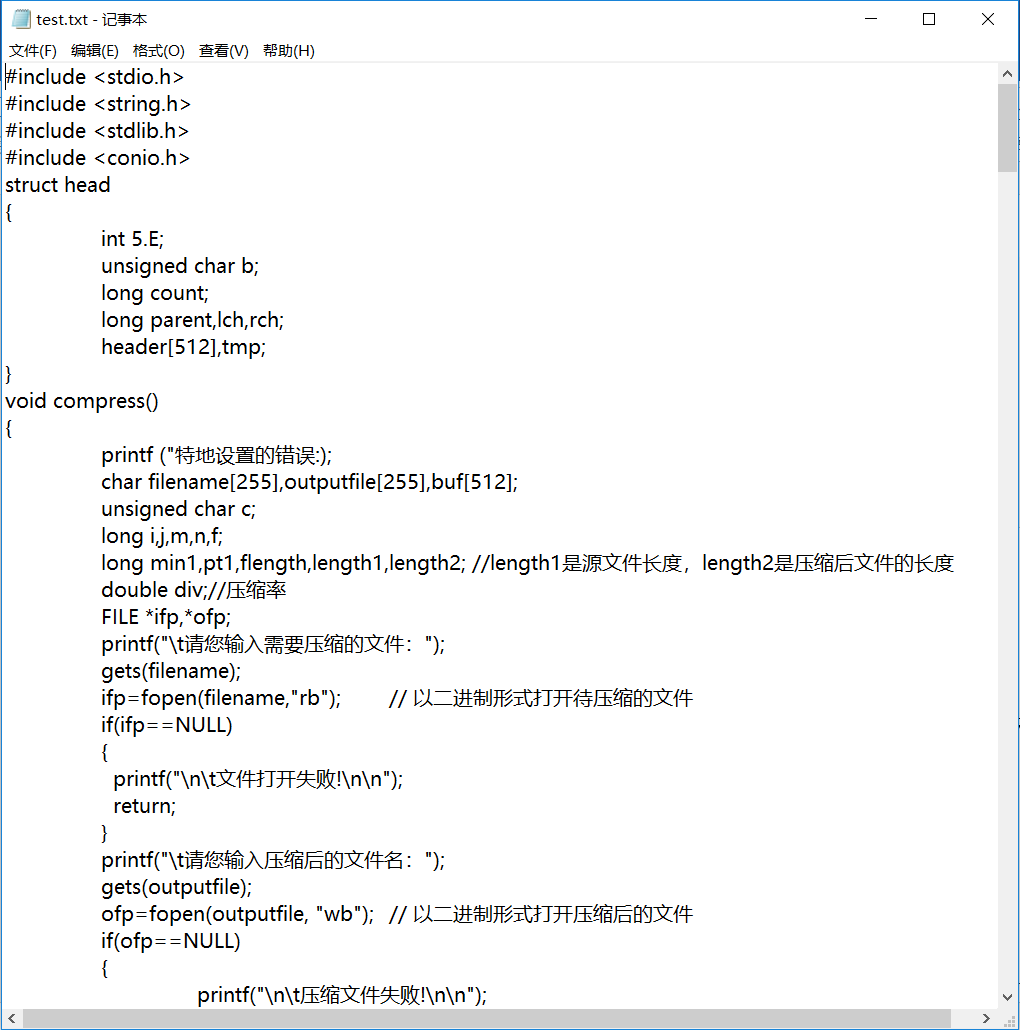
file.close();

ofile.close();

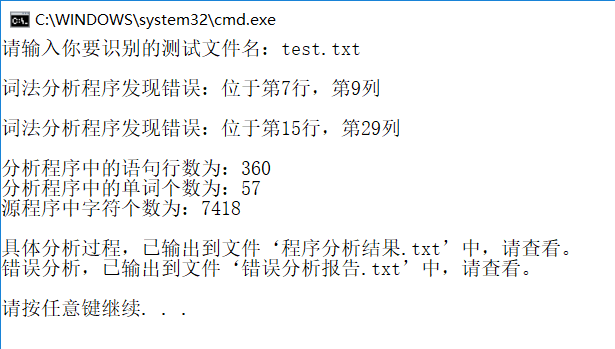
# 八、程序测试与分析

1、本程序可以打开对应的程序文件进行词法分析，本人提供了一个测试程序test.txt。

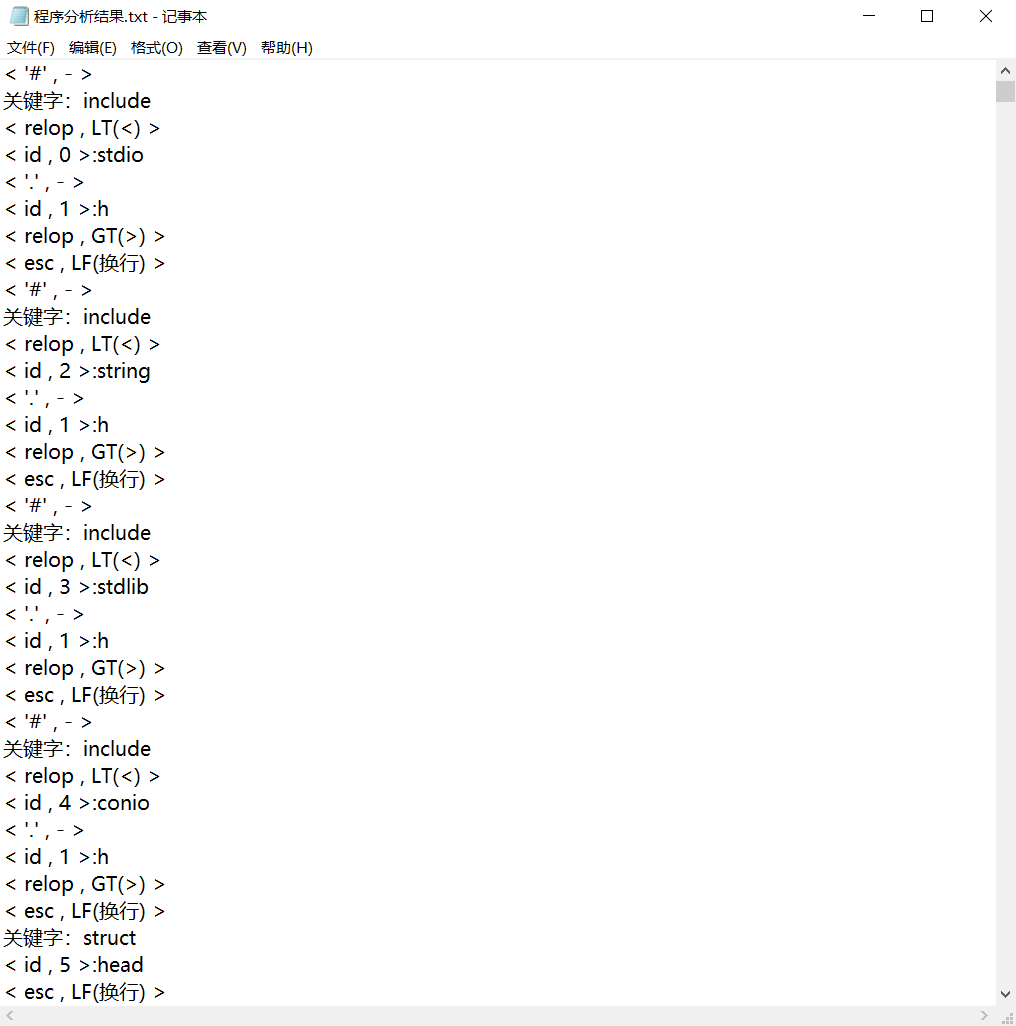
源文件为任意一段复杂C语言代码，包含注释、几乎全部标识符、运算符、转译符。部分如下图所示：



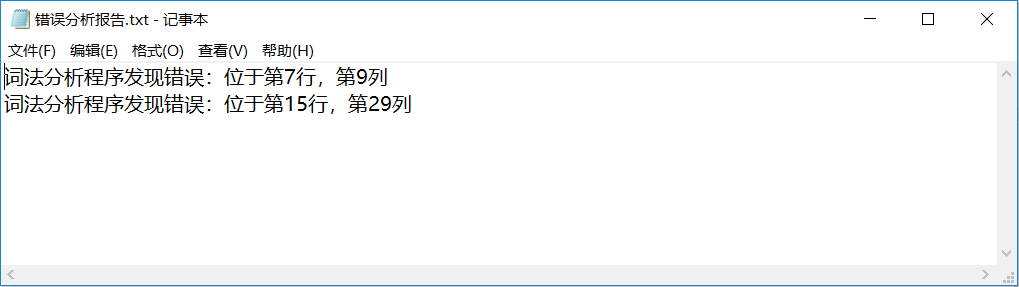
运行词法分析程序，结果如下图所示：



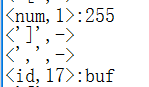
生成文件“程序分析结果.txt”如下图所示：



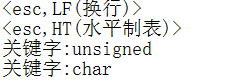
生成文件“错误分析报告.txt”如下图所示：



2、本程序还将识别出来的数字存入数字数组allnum[512][32]，将识别出的所有单词存入单词数组allword[512][32]，并且都有对应编号，输出时均以（记号，属性）表示。



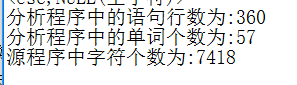
3、关键字和换行的识别如下：



4、注释的识别：



5、统计结果



6、错误处理



# 九、实验总结

共两天左右，一天花了大概5小时讨论程序结构以及函数的调用，一天完成编程并调试，一晚上写文档。

本次实验中遇到了几个问题，第一个是识别/\*…..\*/中多个\*的情况一开始没有考虑，后面画出状态转移图之后，成功识别注释当中的\*。第二个问题是每次识别注释时，后缀总是一堆乱码，我一开始的时候找不出bug，但是后面察觉到有可能是申请的数组空间没有限定终结符号，所以数组的后面的元素显示了乱码，因此我将数组的最后一个非空元素的后一个元素置为“\0”,问题成功解决。

通过本次程序设计实验，我进一步加深了对词法分析的理解，更加明白了词法分析的过程，更让我感觉到计算机世界的博大精深，获益匪浅。

# 附：源代码

#include *<iostream>*

#include *<fstream>*

#include *<string>*

**using** **namespace** std;

ifstream file;*//源文件*

ofstream ofile;*//识别文件*

ofstream errorlist;*//错误分析文件*

*//行数、当前行数、当前列数、字符数*

**int** rownum = 1, nowrow = 1, column = 0, characternum = 0;

*//token数组的长度 allword数组的长度 allnum数组的长度*

**int** tokennum, wordnum = 0, digitnum = 0, searchwordnum, searchnnum;

*//向前指针 注释数目*

**int** forword = 0, notenum=0;

**char** ch = ' ', buffer[4095];

*//关键字数组*

**char** keyword[32][20] =

{ "include","main","int","float","double","char","long","bool","short",

"if","else","for","while","do","struct","typedef","const","default",

"return","case","switch","break","continue","enum","goto","sizeof",

"static","void","union","unsigned","signed","extern" };

*//所有单词的数组 所有数字的数组*

**char** allword[512][32], allnum[512][32];

*//字符数组 注释数组*

**char** token[512], note[1024], getstring[1024];

*//判断数字的函数*

**int** digit()

{

**if** ((ch >= 48 && ch <= 57))

**return** 1;

**else**

**return** 0;

}

*//判断字母的函数*

**int** letter()

{

**if** ((ch >= 97 && ch <= 122) || (ch >= 65 && ch <= 90))

**return** 1;

**else**

**return** 0;

}

*//读取字符放入buffer()数组，每次都读取缓冲区容量的一半*

**void** inbuffer(**int** pointer)

{

**int** i = 0;

**char** ch1;

**while** (!file.eof() && i<2048) {

file.read(&ch1, 1);*//读入一个字符*

buffer[pointer+i] = ch1;

**if** (ch1 != ' ') {

**if** (ch1 == '\n')

rownum++; *//行数+1*

**else**

characternum++; *//字符+1*

}

i++;

}

**if** (file.eof())

buffer[pointer+i] = '\0'; *//源程序读取完毕*

}

*//过滤掉读到的空格*

**void** deletespace()

{

**if** (ch == ' ')

{

ch = buffer[forword];

column++;

**if** (forword == 2047)

{

inbuffer(2048);

forword++;

}

**else** **if** (forword == 4095)

{

inbuffer(0);

forword = 0;

}

**else**

forword++;

deletespace();

}

}

*//读取函数，每次从缓冲区中读出当前指针对应的字符*

**void** get\_char()

{

ch = buffer[forword];

column++;*//当前列数增加*

**if** (forword == 2047)

{

inbuffer(2048);

forword++;

}

**else** **if** (forword == 4095)

{

inbuffer(0);

forword = 0;

}

**else**

forword++;

}

*//连接函数，把当前字符与token中的字符串连接起来*

**void** cat()

{

token[tokennum] = ch;

tokennum++;

}

*//判断当前单词是否已经被放入单词数组*

**int** findidword()

{

**for** (**int** i = 0; i<wordnum; i++)

{

**if** (strcmp(token, allword[i]) == 0)

{

searchwordnum = i;

**return** 1;

}

}

**return** 0;

}

*//将新单词插入单词数组*

**void** insertallword()

{

strcpy\_s(allword[wordnum], token);

wordnum++;

}

*//判断当前数字是否已经放入数字数组*

**int** findnumword()

{

**for** (**int** i = 0; i<digitnum; i++)

{

**if** (strcmp(token, allnum[i]) == 0)

{

searchnnum = i;

**return** 1;

}

}

**return** 0;

}

*//将新数字插入数字数组*

**void** insertallnum()

{

strcpy\_s(allnum[digitnum], token);

digitnum++;

}

*//判断当前单词是否为保留的关键字，若是返回1，不是则返回0*

**int** reserve()

{

**for** (**int** i = 0; i<32; i++)

{

**if** (strcmp(token, keyword[i]) == 0)

**return** 1;

}

**return** 0;

}

*//识别/\*开头的注释，需要循环处理*

**void** takingnote()

{

get\_char();

note[notenum++] = ch;

**while** (ch != '\*')

{

get\_char();

note[notenum++] = ch;

}

get\_char();

note[notenum++] = ch;

}

*//错误处理程序*

**void** error()

{

ofile << "词法分析程序发现错误：位于第" << nowrow << "行，第" << column << "列" << endl;

cout << endl;

cout << "词法分析程序发现错误：位于第" << nowrow << "行，第" << column << "列" << endl;

errorlist << "词法分析程序发现错误：位于第" << nowrow << "行，第" << column << "列" << endl;

}

*//主处理函数*

**int** main()

{

errorlist.open("错误分析报告.txt");

ofile.open("程序分析结果.txt");

string mytestname;

cout << "请输入你要识别的测试文件名：";

cin >> mytestname;

file.open(mytestname, ios::in);

**if** (!file)

{

cout << "未能正确打开测试文件" << endl;

**return** 1;

}

inbuffer(0);

**int** iskey, result;

**while** (ch != '\0')

{

get\_char();

deletespace();

**switch** (ch)

{

*//单词处理*

**case**'a':**case**'b':**case**'c':**case**'d':**case**'e':**case**'f':**case**'g':

**case**'h':**case**'i':**case**'j':**case**'k':**case**'l':**case**'m':**case**'n':

**case**'o':**case**'p':**case**'q':**case**'r':**case**'s':**case**'t':**case**'u':

**case**'v':**case**'w':**case**'x':**case**'y':**case**'z':**case**'A':**case**'B':

**case**'C':**case**'D':**case**'E':**case**'F':**case**'G':**case**'H':**case**'I':

**case**'J':**case**'K':**case**'L':**case**'M':**case**'N':**case**'O':**case**'P':

**case**'Q':**case**'R':**case**'S':**case**'T':**case**'U':**case**'V':**case**'W':

**case**'X':**case**'Y':**case**'Z':**case**'\_':

tokennum = 0;

**while** (letter() || digit() || ch == '\_')

{

cat();

get\_char();

**if** (ch == '\n') nowrow--;

}

token[tokennum] = '\0';*//添加末尾符号否则出现乱码*

forword--;

result = reserve();

**if** (result == 0)

{

iskey = findidword();*//看看当前单词是否已经被放入单词数组*

**if** (iskey == 0)*// 如果没放入*

{

insertallword();

ofile << "< id , " << wordnum - 1 << " >:" << token << endl;

*//cout << "<id," << wordnum - 1 << ">:" << token << endl;*

}

**else**

{

ofile << "< id , " << searchwordnum << " >:" << token << endl;

*//cout << "<id," << searchwordnum << ">:" << token << endl;*

}

}

**else**

{

ofile << "关键字：" << token << endl;

*//cout << "关键字:" << token << endl;*

}

**break**;

*//数字处理 ，依循书上的处理方法*

**case** '0':**case**'1':**case**'2':**case**'3':**case**'4':**case**'5':

**case** '6':**case**'7':**case**'8':**case**'9':

tokennum = 0;

**while** (digit())

{

cat();

get\_char();

}

**if** (ch == '.')

{

cat();

get\_char();

**if** (!digit())

{

error();*//判断2.E这种类型的错误*

**break**;

}

**else**

{

**while** (digit())

{

cat();

get\_char();

}

}

}

**if** (ch == 'E')*//2.3E这种可以被识别*

{

cat();

get\_char();

**if** (ch == '+' || ch == '-')*//E的后面可加+号也可以加-号*

{

cat();

get\_char();

**if** (!digit())

{

error();

**break**;

}

**else** {

**while** (digit())

{

cat();

get\_char();

}

}

}

**else** **if** (digit())

{

**while** (digit())

{

cat();

get\_char();

}

}

}

token[tokennum] = '\0';

forword--;

iskey = findnumword();

**if** (iskey == 0)

{

insertallnum();

ofile << "< num ," << digitnum - 1 << " >:" << token << endl;

*//cout << "<num," << digitnum - 1 << ">:" << token << endl;*

}

**else**

{

ofile << "< num ," << searchnnum << " >:" << token << endl;

*//cout << "<num," << searchnnum << ">:" << token << endl;*

}

**break**;

*//识别以<开头的所有符号*

**case** '<':

get\_char();

**if** (ch == '=')

{

ofile << "< relop , LE(<=) >" << endl;

*//cout << "<relop,LE(<=)>" << endl;*

}

**else** **if** (ch == '>')

{

ofile << "< relop , NE(<>) >" << endl;

*//cout << "<relop,NE(<>)>" << endl;*

}

**else**

{

forword--;

ofile << "< relop , LT(<) >" << endl;

*//cout << "<relop,LT(<)>" << endl;*

}

**break**;

*//识别以=开头的所有符号*

**case** '=':

get\_char();

**if** (ch == '=')

{

ofile << "< relop, EQ(==) >" << endl;

*//cout << "<relop,EQ(==)>" << endl;*

}

**else**

{

forword--;

ofile << "< relop , ass(=) >" << endl;

*//cout << "<relop,ass(=)>" << endl;*

}

**break**;

*//识别以>开头的所有符号*

**case** '>':

get\_char();

**if** (ch == '=')

{

ofile << "< relop , GE(>=) >" << endl;

*//cout << "<relop,GE(>=)>" << endl;*

}

**else**

{

forword--;

ofile << "< relop , GT(>) >" << endl;

*//cout << "<relop,GT(>)>" << endl;*

}

**break**;

**case** ':':

get\_char();

**if** (ch == '=')

{

ofile << "< assign-op , -(:=) >" << endl;

*//cout << "<assign-op,-(:=)>" << endl;*

}

**else**

{

forword--;

ofile << "< ':' , - >" << endl;

*//cout << "<':',->" << endl;*

}

**break**;

**case** '+':

get\_char();

**if** (ch == '=')

{

ofile << "< assignop , assad(+=) >" << endl;

*//cout << "<assignop,assad(+=)>" << endl;*

}

**else** **if** (ch == '+')

{

ofile << "< '++' , - >" << endl;

*//cout << "<'++',->" << endl;*

}

**else**

{

forword--;

ofile << "< '+' , - >" << endl;

*//cout << "<'+',->" << endl;*

}

**break**;

**case** '-':

get\_char();

**if** (ch == '=')

{

ofile << "< assignop , assub(-=) >" << endl;

*//cout << "<assignop,assub(-=)>" << endl;*

}

**else** **if** (ch == '-')

{

ofile << "< '--' , - >" << endl;

*//cout << "<'--',->" << endl;*

}

**else**

{

forword--;

ofile << "< '-' , - >" << endl;

*//cout << "<'-',->" << endl;*

}

**break**;

**case** '\*':

get\_char();

**if** (ch == '=')

ofile << "< assignop , assmul(\*=) >" << endl;

**else**

{

forword--;

ofile << "< '\*' , - >" << endl;

}

**break**;

**case** '(':

ofile << "< '(' , - >" << endl;

**break**;

**case** ')':

ofile << "< ')' , - >" << endl;

**break**;

**case** ';':

ofile << "< ';' , - >" << endl;

**break**;

**case** '%':

ofile << "< '%' , - >" << endl;

**break**;

**case** '[':

ofile << "< '[' , - >" << endl;

**break**;

**case** ']':

ofile << "< ']' , - >" << endl;

**break**;

**case** '{':

ofile << "< '{' , - >" << endl;

**break**;

**case** '}':

ofile << "< '}' , - >" << endl;

**break**;

**case** ',':

ofile << "< ',' , - >" << endl;

**break**;

**case** '#':

ofile << "< '#' , - >" << endl;

**break**;

**case** '.':

ofile << "< '.' , - >" << endl;

**break**;

**case** '?':

ofile << "< '?' , - >" << endl;

**break**;

**case** '!':

get\_char();

**if** (ch == '=')

ofile << "< relop , (!=) >" << endl;

**else**

{

forword--;

ofile << "< logicop , not(!) >" << endl;

}

**break**;

**case** '|':

get\_char();

**if** (ch == '|')

ofile << "< logicop , or(||) >" << endl;

**break**;

**case** '&':

get\_char();

**if** (ch == '&')

ofile << "< logicop , and(&&) >" << endl;

**else**

{

forword--;

ofile << "< '&' , - >" << endl;

}

**break**;

**case** '/':

get\_char();

**if** (ch == '\*')

{

ofile << "< '/\*' , note >" << endl;

notenum = 0;

takingnote();

**while** (ch != '/') takingnote();

note[notenum - 1] = ' ';

note[notenum - 2] = ' ';

ofile << "注释如下：" << note << endl;

ofile << "< '\*/' , note >" << endl;

**break**;

}

**else** **if** (ch == '/')

{

ofile << "< '//' , note >" << endl;

notenum = 0;

**char** note1[1024];

**while** (ch != '\n')

{

get\_char();

note1[notenum++] = ch;

}

note1[notenum++] = '\0';

ofile << "注释如下：" << note1;

**break**;

}

**else** **if** (ch == '=')

ofile << "< assignop , assdiv'/=' >" << endl;

**else** {

forword--;

ofile << "< '/' , - >" << endl;

}

**break**;

**case** '\\':

ofile << "< esc , \ >" << endl;

**break**;

**case** '\a':

ofile << "< esc , BEl()响铃 >" << endl;

**break**;

**case** '\b':

ofile << "< esc , BS(退格) >" << endl;

**break**;

**case** '\f':

ofile << "< esc , FF(换页) >" << endl;

**break**;

**case** '\n':

ofile << "< esc , LF(换行) >" << endl;

nowrow++;

column = 0;

**break**;

**case** '\r':

ofile << "< esc , CR(回车) >" << endl;

**break**;

**case** '\t':

ofile << "< esc , HT(水平制表) >" << endl;

**break**;

**case** '\v':

ofile << "< esc , VT(垂直制表) >" << endl;

**break**;

**case** '\"':

ofile << "< esc , 双引号字符 >" << endl;

get\_char();

iskey = 0, result = 1;

getstring[128];

**while** (ch != '\"')

{

getstring[iskey] = ch;

**if** (ch == '\n')

{

error();

forword--;

result = 0;

**break**;

}

get\_char();

iskey++;

}

**for** (**int** i = iskey; i <= 100; i++) getstring[i] = ' ';

**if** (result == 1)*//判断是否为字符串*

ofile << "字符串：" << getstring << endl;

**else**

ofile << "出错字符串：" << getstring << endl;

ofile << "< esc, 双引号字符 >" << endl;

**break**;

**case** '\0':

ofile << "< esc , NULL(空字符) >" << endl;

**break**;

**case** '\'':

ofile << "< esc , '''(单引号) >" << endl;

**break**;

**default**:

error();

**break**;

}

}

cout << endl;

ofile << "分析程序中的语句行数为：" << rownum << endl;

cout << "分析程序中的语句行数为：" << rownum << endl;

ofile << "分析程序中的单词个数为：" << wordnum << endl;

cout << "分析程序中的单词个数为：" << wordnum << endl;

ofile << "源程序中字符个数为：" << characternum << endl;

cout << "源程序中字符个数为：" << characternum << endl;

cout << endl;

cout << "具体分析过程，已输出到文件‘程序分析结果.txt’中，请查看。" << endl;

cout << "错误分析，已输出到文件‘错误分析报告.txt’中，请查看。" << endl;

cout << endl;

file.close();

ofile.close();

system("pause");

**return** 0;

}