**《词法分析程序》实验报告**

**一、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，对源程序进行一次扫描，既可检查并报告源程序中存在的所有语法错误。

**二、实验分析**

1. 语言说明

根据C语言标准，可将一个C源程序分为以下记号及单词：

1. 标识符：以字母开头的、后跟字母或数字组成的字符串
2. 关键字：标识符集合的子集，C语言的关键字有32个



1. 无符号数：由整数部分、可选的小数部分和可选的指数部分构成
2. 关系运算符：<、<=、==、>、>=、!=
3. 算术运算符：+、-、\*、/。
4. 标点符号：(、)、{、}、[、]、;、,、\、\’、’
5. 赋值号:=
6. 注释标记：以“/\*”开始，以“\*/”结束。
7. 单词符号间的分隔符：空格、\t
8. 文法产生式
9. 标识符文法

id->letter rid

rid-> | letter rid | digit rid

1. 无符号整数的文法

digits->digit remainder

remainder-> | digit remainder

1. 无符号数的文法

num->digit num 1

num1->digit num 1 |. num2 | E num4 |

num2->digit num3

num3->digit num3 | E num4

num4->+digits | -digits | digits num5

digits->digit num 5

num5->digit num 5 |

1. 关系运算符的文法

relop-> < | < equal | = |< greater | > | >equal

greater-> >

equal-> =

1. 赋值号的文法

assign\_op-> =

1. 算术运算符及标点符号的文法

single->+|-|\*|/|(|)|[|]|{|}|;|,|\|\’|’

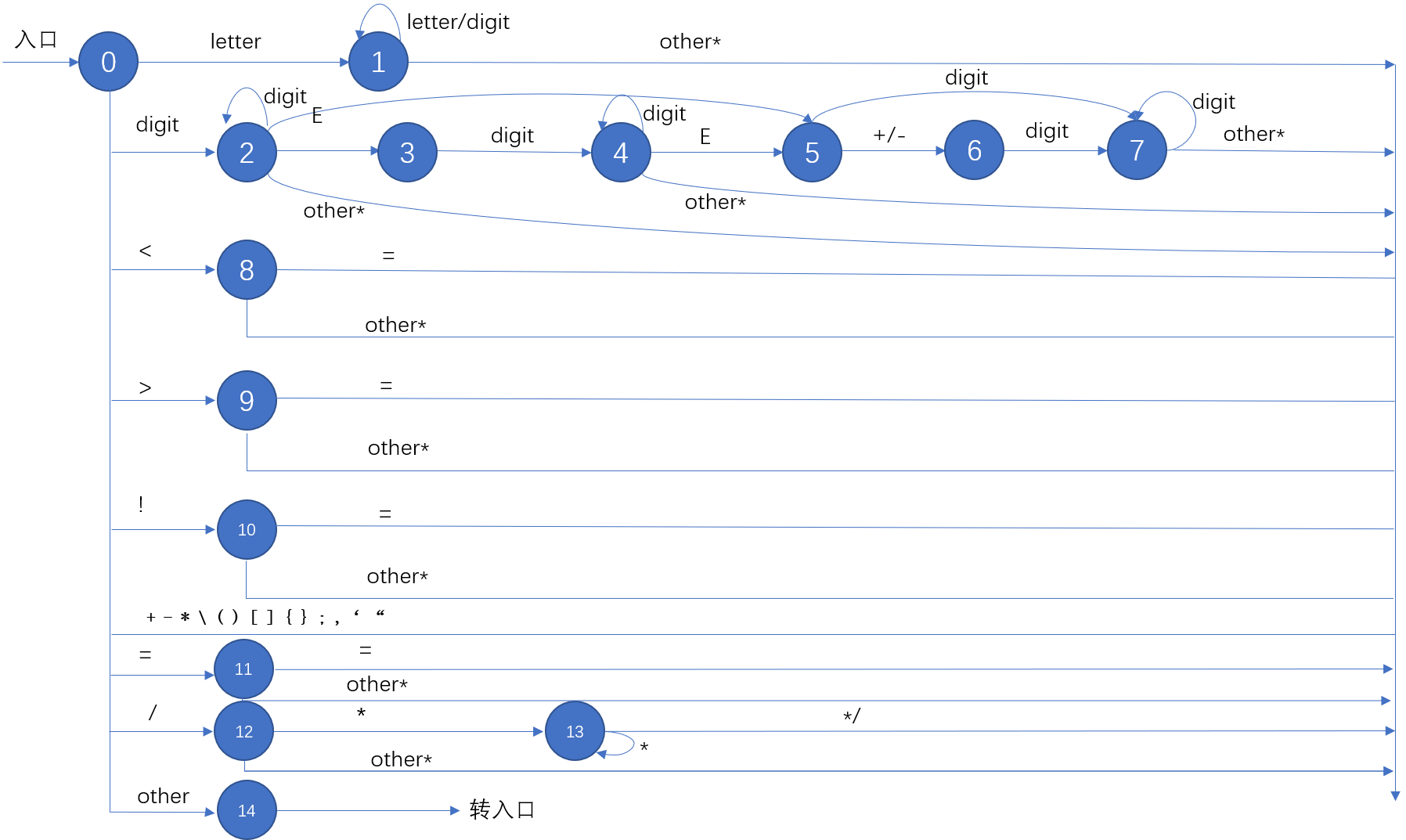
1. 注释头符号的文法

note->/star

star->\*

1. 状态转移图

根据(二)中的文法，我们可以构造出如下的状态转移图：



状态0是初始状态，若此时读入的字符是字母，则转换到状态1，进入标识符识别过程，如果此时读入的字符是数字，则转换到状态2，进入无符号数识别过程，在状态2时，若读入字符是”.”，则转入状态3，而若读入其他，则转到状态14出口，状态3读入字符为数字时，转入状态4，此时读入“e”或“E”时，转入状态5，若读入其他则转入状态14出口，状态5读入数字则转入状态7，读入+|-则转入状态6，状态6读入数字转入状态7，状态7若接着读入数字则循环状态7，否则转入状态14出口。

初始时读入字符是“<”，转入状态8

初始时读入字符是“>”，转入状态9

初始时读入字符是“！”，转入状态10

初始时读入字符是“=”，转入状态11

初始时读入字符是“/”，转入状态12,状态12再次读入字符“\*”将转入状态13，状态13为注释处理状态

状态14为错误处理状态

**三、程序设计**

1. 输出形式

本词法分析程序使用下表所给出的翻译表，在分离出一个单词之后，对识别出的记号以二元式的形式加以输出，其形式为<记号，属性>

|  |  |  |
| --- | --- | --- |
| 正则表达式 | 记号 | 属性 |
| 32个关键字 | 关键字本身 | Keyword |
| id | id | 符号表入口指针 |
| num | num | 常数值 |
| ( | ( | symbol |
| ) | ) | symbol |
| [ | [ | symbol |
| ] | ] | symbol |
| { | { | symbol |
| } | } | symbol |
| , | , | symbol |
| ; | ; | symbol |
| ‘ | ‘ | symbol |
| “ | “ | symbol |
| \ | \ | symbol |
| < | relop | LT |
| > | relop | GT |
| <= | relop | LE |
| >= | relop | GE |
| == | relop | EQ |
| != | relop | NE |
| = | = | assign\_op |
| + | + | -- |
| - | - | -- |
| \* | \* | -- |
| : | : | -- |
| & | & | -- |
| % | % | -- |
|  |  |  |

1. 全局变量和过程
2. kw类

class kw

{

public:

string re\_exp;//正规表达式

string mark;//记号

string nature;//属性

kw()//无参数构造函数

{

re\_exp = "";

mark = "";

nature = "";

}

kw(string a, string b, string c)//有参数构造函数

{

re\_exp = a;

mark = b;

nature = c;

}

};

kw类用于定义字符串，kw类中有三个参数，re\_exp代表正则表达式，mark代表记号，nature代表属性，kw类中定义了一个默认构造函数，一个带参构造函数，根据上述输出形式中定义的表，可给出每个识别字符串对应的kw类实例，程序伊始定义了C语言中的32个关键字：

kw kw\_ex[32] = { kw("auto", "auto", "Z"),

kw("short", "stort", "Z"),

kw("int", "int", "Z"),

kw("long", "long", "Z"),

kw("float", "float", "Z"),

kw("double", "double", "Z"),

kw("char", "char", "Z"),

kw("struct", "struct", "Z"),

kw("union", "union", "Z"),

kw("enum", "enum", "Z"),

kw("typedef", "typedef", "Z"),

kw("const", "const", "Z"),

kw("unsigned", "unsigned", "Z"),

kw("signed", "signed", "Z"),

kw("extern", "extern", "Z"),

kw("register", "register", "Z"),

kw("static", "static", "Z"),

kw("volatile", "volatile", "Z"),

kw("void", "void", "Z"),

kw("if", "if", "Z"),

kw("else", "else", "Z"),

kw("switch", "switch", "Z"),

kw("case", "case", "Z"),

kw("for", "for", "Z"),

kw("do", "do", "Z"),

kw("while", "while", "Z"),

kw("goto", "goto", "Z"),

kw("continue", "continue", "Z"),

kw("break", "break", "Z"),

kw("default", "default", "Z"),

kw("sizeof", "sizeof", "Z"),

kw("return", "return", "Z")

};

32个关键字的记号mark值均为原关键字，属性nature值为Z，另外定义了kw类的实例数组kw tag[100]用于存储用于自定义的标识符，int型变量i\_tag用于表示用户自定义标识符的指针，当识别到新定义的标识符时，将标识符添加进数组，并将i\_tag后移。

1. fw类

class fw

{

public:

int row;

int line;

fw()

{

row = 0;

line = 0;

}

};//指针forward类

fw是forward的简写，代表字符指针，向前指针，fw中定义了两个变量row，line。分别代表当前指针在源文件中所处的行数和列数，fw默认构造函数将row和line都置0，在本次实验中，我们定义了fw的两个实例f和lf，分别代表当前字符的指针位置和之前指针的字符位置(用于回退)

1. ch字符串数组

string ch[11] = { "{", "[", "(", ")", "]", "}", ";", ",", "\'", "\"","\\"};

ch数组是所有标点符号数组的集合，对于标点符号正则式，其记号定义为本身，属性定义为symbol

1. 错误数err， 总行数all\_line，单词个数word\_num，字符总数all\_ch

int err = 0;//错误数

int all\_line = 0;//语句行数

int word\_num = 0;//单词个数

int all\_ch = 0;//字符总数

1. 其他变量

char C;

string token;

int state = 0;//当前状态指示

string iskey = "";

int line = 0;

int row = 0;//指针

C为字符变量，存放当前读入的字符

token为字符数组，存放当前正在识别的单词字符串

state为整型变量，为当前状态指示

iskey为字符串型变量，若值为“-1”则表示识别出的单词是用户自定义标识符，否则为所定义的关键字之一，其值为关键字的记号

line为整型变量，当前指针所指向的字符所在列数

row为整型变量，当前指针所指向的字符所在行数

1. 文件读入函数readTxt()

void readTxt(string file)

{

ifstream infile;

infile.open(file); //将文件流对象与文件连接起来

assert(infile.is\_open()); //若失败,则输出错误消息,并终止程序运行

string s;

int i = 0;

while (getline(infile, s))

{

code[i] = s;

all\_ch += s.length();

code[i] += '\n';

i++;

std::cout << s << endl;

}

all\_line = i;

infile.close(); //关闭文件输入流

code[i] = "#";

for (auto a : ch) {

cout << a << " ";

}

cout << endl;

}

用于从指定的txt文件中读入代码字符串并处理，在这里我们按行读入并将每一行中的所有字符都依次存入数组code中，每一行末尾添加’\n’，用于字符指针的更新和处理，统计当前代码文件所有行数，再整个code数组后中添加“#”代表整个代码的结束。

1. 字符读入函数get\_char()

void get\_char()

{

rf.row = f.row;

rf.line = f.line;

if (code[f.row][f.line] != '\n')

{

C = code[f.row][f.line];

f.line++;

}

else

{

f.row++;

f.line = 0;

C = code[f.row][f.line];

f.line++;

}

}//根据向前指针forward的指示从输入缓冲区中读出一个字符并放入C中

在本实验中的get\_char()主要用于从已经存储好的字符数组code中读入字符并将其放入C中，当未读取到‘\n’时，将当前字符指针f的line值加一，row值不变。当读取到‘\n’时，将f.row值加一，f.line值置为0，取出字符C

1. 空格跳过函数get\_nbc()

void get\_nbc()

{

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

{

get\_char();

}

}//检查空格和\t

每次调用时，检查C中的字符是否为空格，若是，则反复调用过程get\_char(),直到C中进入一个非空字符为止

1. 字符拼接函数cat

void cat()

{

token += C;

}//将C中字符拼接在token中的字符串后面

1. 判断字母函数letter()

bool letter()

{

if (((C >= 'a') && (C <= 'z')) || ((C >= 'A') && (C <= 'Z')) || (C == '\_'))//字符

return true;

else

return false;

}//判断字母或下划线

布尔函数，判断C中的字符是否为字母，若是则返回true，否则返回false

1. 判断数字函数digit()

bool digit()

{

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

return true;

else

return false;

}//判断数字

布尔函数，判断C中的字符是否为字母，若是则返回true，否则返回false

1. 指针回退函数restract()

void restract()

{

f.row = rf.row;

f.line = rf.line;

}//指针forward后退一个字符

向前指针forward(f)后退一个字符

1. 关键字匹配函数reverse()

string reserve()

{

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

{

if (token == kw\_ex[i].re\_exp)

return kw\_ex[i].mark;

}

return "-1";

}

//根据token中的单词查关键字表，若token中的单词是关键字，则返回值该关键字的记号，否则返回-1

1. 自定义标识符插入函数table\_insert()

int table\_insert(){

for (int i = 0; tag[i].re\_exp != ""; i++){

if (token == tag[i].re\_exp){

return i;

}

}

tag[i\_tag].re\_exp = token;

tag[i\_tag].mark = "id";

tag[i\_tag].nature = i\_tag;

i\_tag++;

return (i\_tag - 1);

}//将识别出来的用户自定义标识符插入符号表，返回该单词在符号表中的位置指针

将传入的字符串进行识别，若已经在符号表中找到，则返回自定义字符在符号表中的入口地址，否则将字符串插入符号表中(新增一个自定义标识符),并将该位置返回

1. 输出函数ret()

void ret(string A, string B)

{

int flag = -1;

for (int i = 0; i < 11; i++)

if (ch[i] == A)

flag = i;

if (flag >= 0)

std::cout << "<" << A << " , " << "symbol" << ">" << endl;

else if (A == "relop")

std::cout << "<" << "relop" << " , " << B << ">" << endl;

else if (B == "Z")

std::cout << "<" << A << " , " << "keyword" << ">" << endl;

else

std::cout << "<" << A << " , " << B << ">" << endl;

}//输出

void ret(string A, int B)

{

std::cout << "<" << A << " , " << B << ">" << endl;

}

void ret(string A, float B)

{

std::cout << "<" << A << " , " << B << ">" << endl;

}

根据传入参数的不同，ret函数有三种不同的重载，分别对应三种不用的输出形式

1. 错误输出函数error()

void error(int row, int line)

{

std::cout << "第" << row + 1 << "行" << "第" << line + 1 << "列出现错误" << endl;

err++;

}//对发现的错误进行相应的处理

输出错误所在的指针位置

1. 具体状态分析

状态0是初始状态，将token置空，将变量row和line置为当前字符所在位置指针f.row和f.line，通过get\_char()函数读入字符并赋值给C，通过get\_nbc()跳过所有的空格和\t，之后根据C的值更新状态值state

1. 状态 1

case 1:

//标识符状态

cat();

get\_char();

if (letter() || digit())

state = 1;

else

{

restract();

state = 0;

iskey = reserve();

word\_num++;

if (iskey != "-1")

ret(iskey, "Z");

else {

int identry = table\_insert(); //返回标识符在符号表的入口指针

ret("id", identry);

}

}

break;

当状态0读入的字符C为字母时，当前状态转换到状态1，状态1一直读入字符并将其拼接到token后，知道读入的字符不为字母或数字其一，此时转入处理。首先将字符指针回退，状态回退0，之后置iskey的值为reverse(),若iskey的值为”-1”表明当前识别到的标识符为关键字，通过ret()函数输出，否则，说明当前识别到的标识符为用户自定义标识符，执行table\_insert()函数，返回符号表的入口指针地址或插入新的自定义标识符进入表中，通过ret()函数输出。

1. 状态2

case 2:

//常数状态

cat();

get\_char();

switch (C) {

case '0': case '1':

case '2': case '3':

case '4': case '5':

case '6': case '7':

case '8': case '9':

state = 2;

break;

case '.':

state = 3;

break;

case 'E':

case 'e':

state = 5;

break;

default:

if (((C >= 'a') && (C <= 'z')) || ((C >= 'A') && (C <= 'Z')) || (C == '\_'))

{

while ((C != ' ') && (C != ',') && (C != ';'))

get\_char();

state = 14;

restract();

}

else

{

word\_num++;

restract();

state = 0;

ret("num", stoi(token));//返回整数

}

break;

}

break;

当状态0读入的字符C为数字时，当前状态转换到状态2，之后接着读入字符，若字符一直为数字则一直循环状态2，若此时读入字符’.’，说明可能是小数，转入状态3，若此时读入字符’e’或’E’，说明可能是科学计数法表示数，转入状态5，当读入字符为字母或下划线时，此时词法出错，返回状态14，输出错误信息，否则说明仅有一个整数，指针回退，状态置0，通过ret()函数输出该整数

1. 状态3

case 3:

//小数点状态

cat();

get\_char();

if (digit())

state = 4;

else

{

error(row, line);

state = 0;

}

break;

当状态0读入的字符C为数字时，当前状态转换到状态2，之后接着读入字符，若字符一直为数字则一直循环状态2，若此时读入字符’.’，说明可能是小数，转入状态3，若此时读入字符’e’或’E’，说明可能是科学计数法表示数，转入状态5，当读入字符为字母或下划线时，此时词法出错，返回状态14，输出错误信息，否则说明仅有一个整数，指针回退，状态置0，通过ret()函数输出该整数