**C语言词法分析器实验报告**

**付彬 2017211307班 2017211357**

1. **运行/开发环境及版本号**

OS：Windows 10

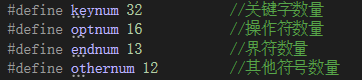
IDE：Visual Studio 2019

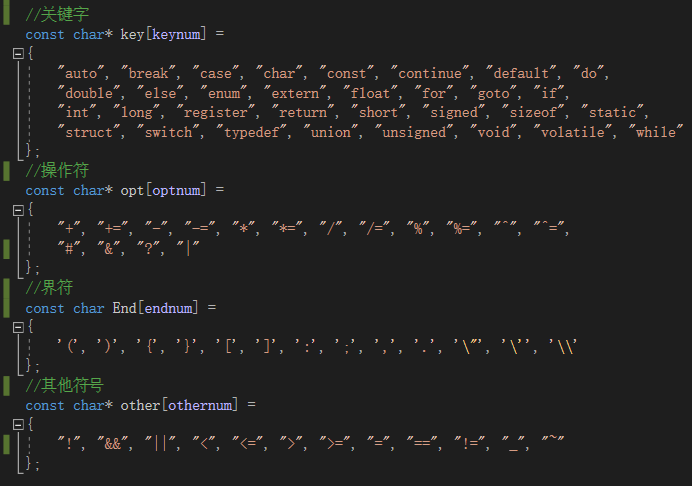
1. **C语言词法分析器设计**
2. **状态转换图**
3. **实现**
4. **数据结构**
5. **双缓存区**

****

****

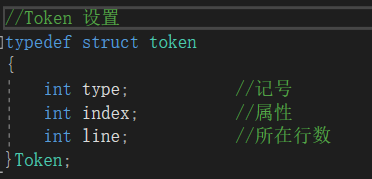
1. **关键字和符号**

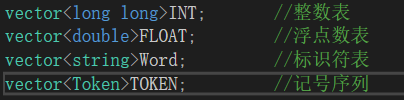
****

****

1. **记号序列、标识符表和数字表**

**记号的设计：**

****

****

1. **主要功能**
2. **识别单词符号，并以记号的形式输出每个单词符号；**

**输出格式：**

**< 记号类别 属性（打印时用单词展示） 行号 >**

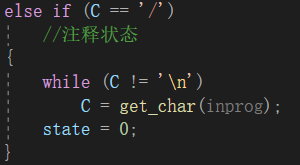
**这里的单词是源程序中的单词符号，记号中实际存储的是属性值，即标识符表、数字表或符号表的索引。**

****

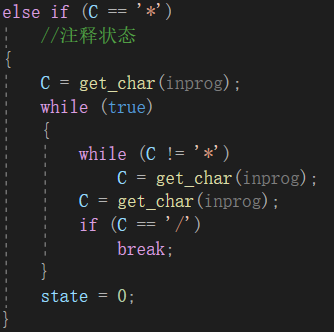
1. **识别并跳过源程序中的注释**

**在识别到一个/符号后，再识别一个字符，进而识别注释：**

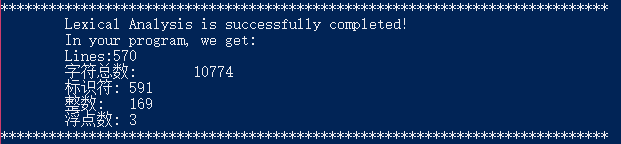
**以//开始的整行注释：**

****

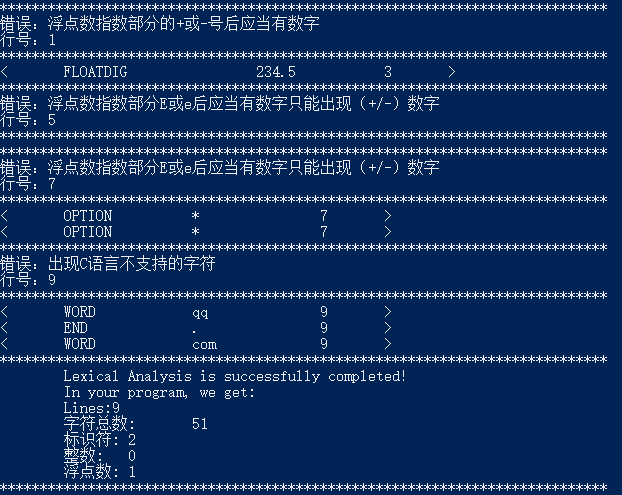
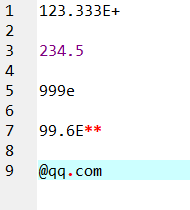
**在/\*和\*/之间的注释：**

****

1. **统计行数、各类单词的个数，字符总数，并输出结果**

****

1. **检测词法错误**

****

1. **有一定的错误恢复能力，仅用一遍分析出所有词法错误**
2. **程序结构**

**由main函数调用Getarg函数，获取被分析文件的名字，打开文件，然后调用analysis函数，开始语法分析。**

**analysis函数可以调用get\_char函数读取一个字符，当缓存区已处理完，会由getinput函数从文件中读取。analysis函数中主要是一个while语句，里面是switch语句，即由状态转换图构造的有限状态自动机。自动机主要会调用的是add函数，添加一个记号到序列中；error函数，进行错误处理。当词法分析完成后，会打印汇总信息，然后退出while循环，analysis函数结束，词法分析结束。**

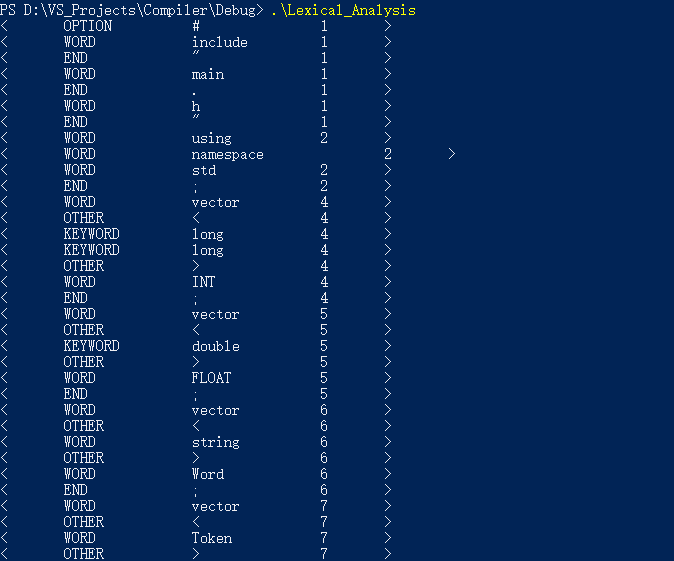
1. **使用说明**

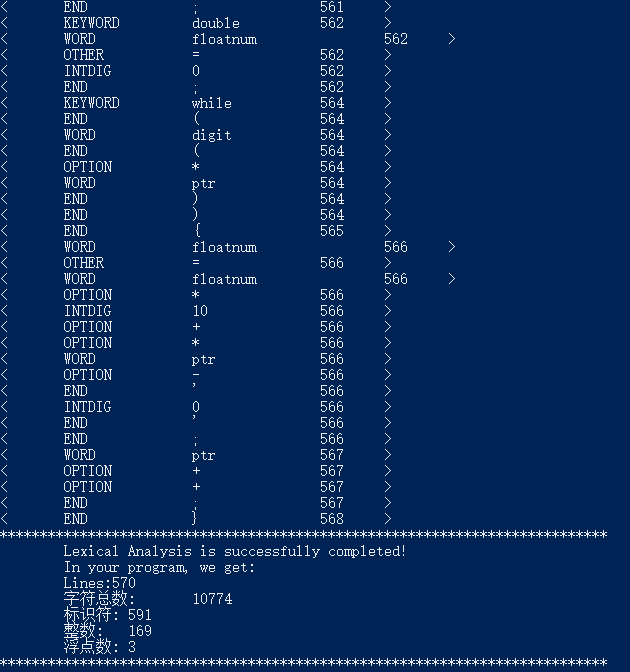
用户可以在命令行中运行可执行文件，可以指定进行词法分析的程序，也可以使用缺省值（可执行文件同目录下的test.c文件）。

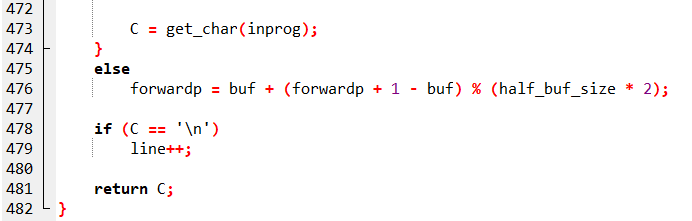
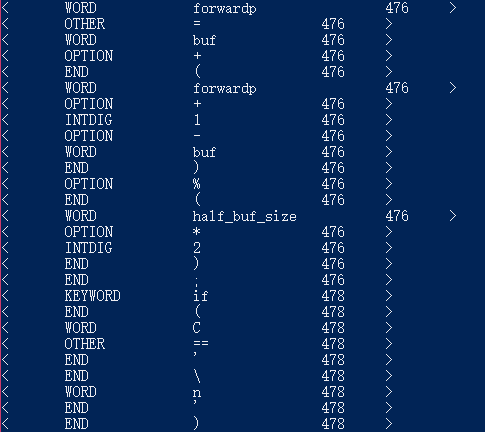




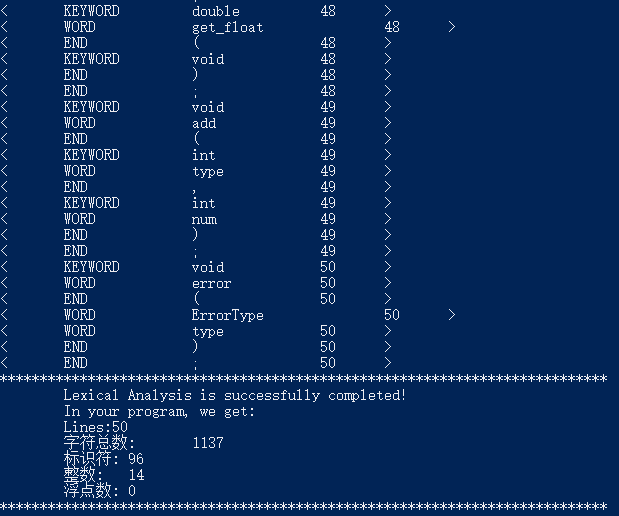
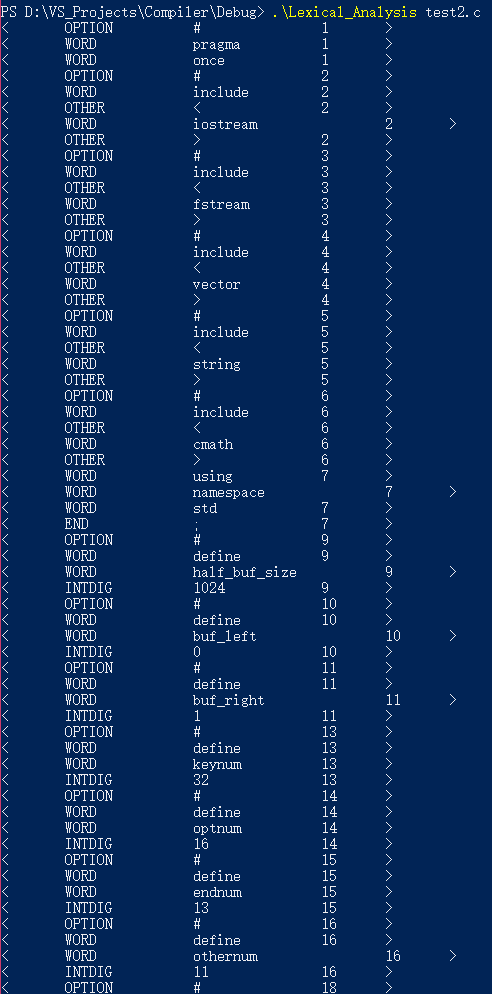
1. **运行截图**
2. **缺省文件**

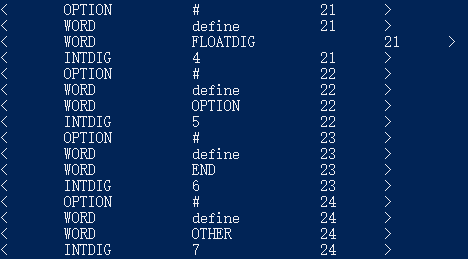


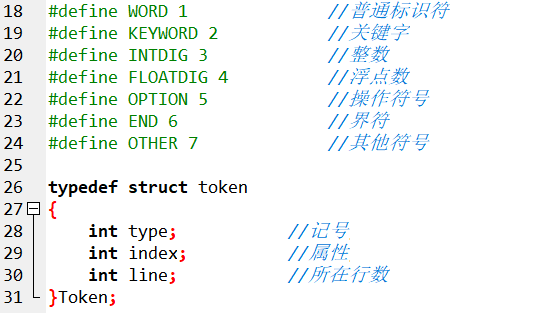




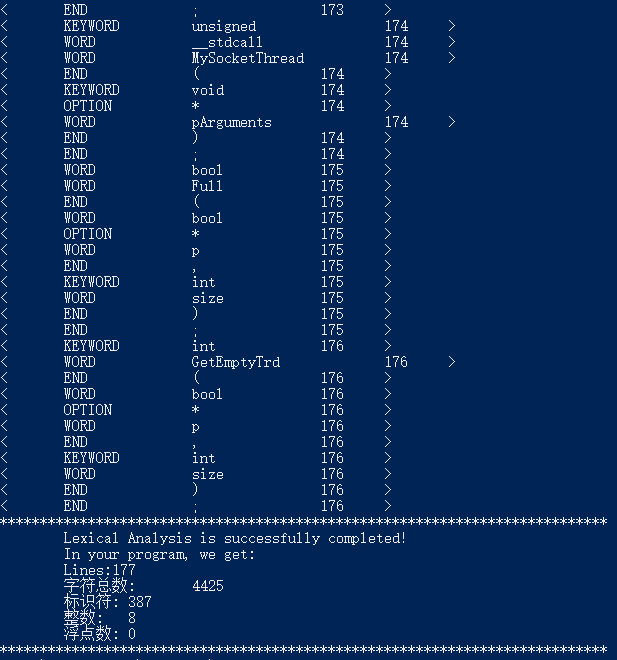
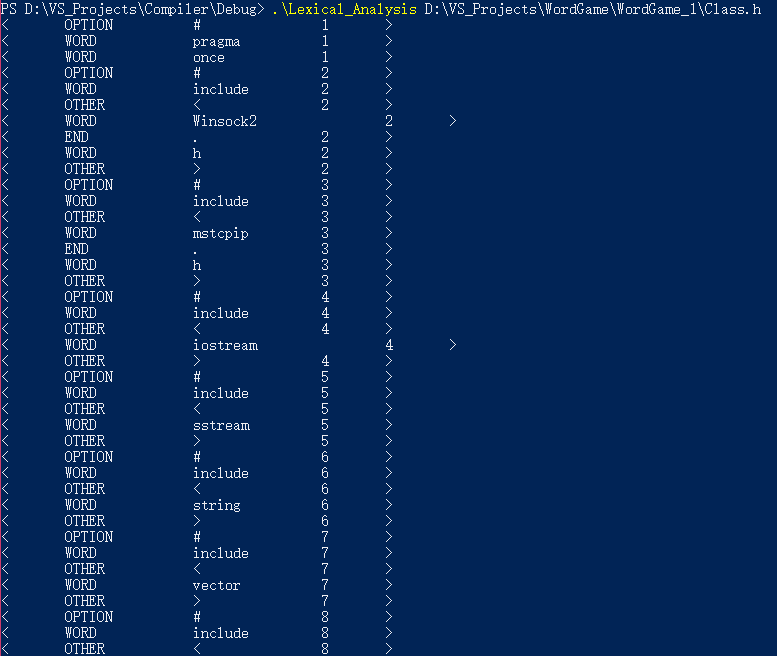
1. **使用参数指明的文件**

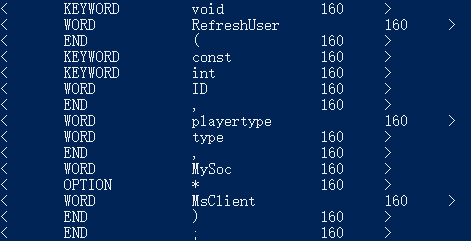


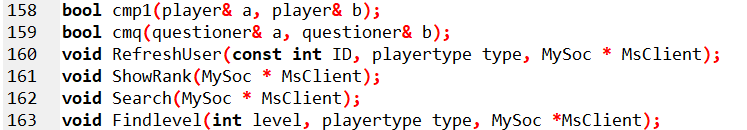




1. **使用参数指明的其他目录下的文件**







1. **源代码**
2. **main.h**

#pragma once

#include<iostream>

#include<fstream>

#include<vector>

#include<string>

#include<cmath>

using namespace std;

#define half\_buf\_size 1024

#define buf\_left 0

#define buf\_right 1

#define keynum 32

#define optnum 16

#define endnum 13

#define othernum 12

#define WORD 1 //普通标识符

#define KEYWORD 2 //关键字

#define INTDIG 3 //整数

#define FLOATDIG 4 //浮点数

#define OPTION 5 //操作符号

#define END 6 //界符

#define OTHER 7 //其他符号

typedef struct token

{

int type; //记号

int index; //属性

int line; //所在行数

}Token;

enum Status{OK, ERR, TER};

enum ErrorType{Wrong\_dot, Wrong\_neg, Wrong\_E, Invalid\_character, Other\_wrong};

void getinput(ifstream& inprog, int pos);

Status analysis(ifstream& inprog);

char get\_char(ifstream& inprog);

void get\_npc(ifstream& inprog, char& C);

bool letter(char& C);

bool digit(char& C);

int reserve(char\* token);

void cat(char& C, int& cur);

void retract(void);

long long get\_int(char\* buf);

int get\_pnum\_length(long long temp);

double get\_float(void);

void add(int type, int num);

void error(ErrorType type);

1. **main.cpp**

#include"main.h"

using namespace std;

char defaultfile[] = "D://VS\_Projects//Compiler//Debug//test.c";

char\* filename = defaultfile;

char buf[half\_buf\_size \* 2];

char\* lexemebegin;

char\* forwardp;

char token[half\_buf\_size];

//获取被编译文件文件名

bool Getarg(int argc, char\* argv[])

{

if (argc == 2)

{

filename = argv[1];

return true;

}

else if (argc > 2)

return false;

}

int main(int argc, char\* argv[])

{

//获取被编译文件文件名

if (Getarg(argc, argv))

{

//打开文件

ifstream inprog(filename, ifstream::in | ifstream::binary);

//运行词法分析程序

Status result = analysis(inprog);

}

else

cout << "输入的初始化参数错误！" << endl;

return 0;

}

1. **fileIO.cpp**

#include"main.h"

using namespace std;

extern char buf[half\_buf\_size \* 2];

//从文件中读取缓存区的一半长度的字符

void getinput(ifstream &inprog, int pos)

{

/\*inprog.seekg(0, fstream::end);

cout << inprog.tellg() << EOF << endl;\*/

char temp[half\_buf\_size + 1] = { '\0' };

inprog.read(/\*buf + pos \* half\_buf\_size\*/temp, half\_buf\_size - 1);

int num = strlen(temp);

temp[num] = -1;

//cout << inprog.tellg();// EOF = -1

strcpy\_s(buf + pos \* half\_buf\_size, sizeof(temp), temp);

buf[half\_buf\_size - 1 + pos \* half\_buf\_size] = -1;

}

1. **analysis.cpp**

#include"main.h"

using namespace std;

vector<long long>INT;

vector<double>FLOAT;

vector<string>Word;

vector<Token>TOKEN;

int line = 1;

extern char buf[half\_buf\_size \* 2];

extern char\* lexemebegin;

extern char\* forwardp;

extern char token[half\_buf\_size];

const char\* key[keynum] =

{

"auto", "break", "case", "char", "const", "continue", "default", "do",

"double", "else", "enum", "extern", "float", **"for", "goto", "if",**

"int", "long", "register", "return", "short", "signed", "sizeof", "static",

"struct", "switch", "typedef", "union", "unsigned", "void", "volatile", "while"

};

const char\* opt[optnum] =

{

"+", "+=", "-", "-=", "\*", "\*=", "/", "/=", "%", "%=", "^", "^=",

"#", "&", "?", "|"

};

const char End[endnum] =

{

'(', ')', '{', '}', '[', ']', ':', ';', ',', '.', '\"', '\'', '\\'

};

const char\* other[othernum] =

{

"!", "&&", "||", "<", "<=", ">", ">=", "=", "==", "!=", "\_", "~"

};

Status analysis(ifstream& inprog)

{

//初始化缓存区

memset(buf, 0, sizeof(buf));

buf[half\_buf\_size - 1] = buf[half\_buf\_size \* 2 - 1] = -1;

memset(token, 0, sizeof(token));

lexemebegin = buf;

forwardp = buf;

int cur = 0;

getinput(inprog, buf\_left);

int state = 0;

bool ter = false;

char C;

while (!ter)

{

switch (state)

{

case 0://初始状态

memset(token, 0, sizeof(token));

cur = 0;

//读取一个字符

C = get\_char(inprog);

get\_npc(inprog, C);

if (C == EOF) { ter = true; break; }

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

//进入标识符状态

state = 1;

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

//进入数字状态

state = 2;

else

switch (C)

{

case '#':state = 0; add(OPTION, 12); break;

case '?':state = 0; add(OPTION, 14); break;

case '(':state = 0; add(END, 0); break;

case ')':state = 0; add(END, 1); break;

case '{':state = 0; add(END, 2); break;

case '}':state = 0; add(END, 3); break;

case '[':state = 0; add(END, 4); break;

case ']':state = 0; add(END, 5); break;

case ':':state = 0; add(END, 6); break;

case ';':state = 0; add(END, 7); break;

case ',':state = 0; add(END, 8); break;

case '.':state = 0; add(END, 9); break;

case '\"':state = 0; add(END, 10); break;

case '\'':state = 0; add(END, 11); break;

case '\\':state = 0; add(END, 12); break;

case '\_':state = 0; add(OTHER, 10); break;

case '~':state = 0; add(OTHER, 11); break;

case '+':state = 8; break;

case '-':state = 9; break;

case '\*':state = 10; break;

case '/':state = 11; break;

case '%':state = 12; break;

case '^':state = 13; break;

case '&':state = 14; break;

case '<':state = 15; break;

case '>':state = 16; break;

case '!':state = 17; break;

case '=':state = 18; break;

case '|':state = 19; break;

default:state = 20; break;

}

break;

case 1://标识符状态

cat(C, cur);

C = get\_char(inprog);

//判断是否是合格的字符，属于超前验证

if (letter(C) || digit(C) || C == '\_')

state = 1;

else

//已达到标识符结尾

{

retract();

state = 0;

//判断是否是关键字

int iskey = reserve(token);

if (iskey != -1)

//识别结果：关键字

add(KEYWORD, iskey);

else

//识别结果：普通标识符

add(WORD, 0);

}

break;

case 2://数字状态

cat(C, cur);

C = get\_char(inprog);

if (digit(C))

//还在整数状态

state = 2;

else if (C == '.')

//进入小数点状态

state = 3;

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

//进入指数状态

state = 5;

else

//在整数状态结束

{

retract();

state = 0;

//识别结果：整数

add(INTDIG, 0);

}

break;

case 3://小数点状态

cat(C, cur);

C = get\_char(inprog);

if (digit(C))

state = 4;

else

{

//在小数点状态出现了错误字符，错误处理

error(Wrong\_dot);

state = 0;

}

break;

case 4://小数状态

cat(C, cur);

C = get\_char(inprog);

if (digit(C))

state = 4;

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

state = 5;

else

{

retract();

state = 0;

//识别结果：浮点数

add(FLOATDIG, 0);

}

break;

case 5://指数状态

cat(C, cur);

C = get\_char(inprog);

if (digit(C))

state = 7;

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

state = 6;

else

{

retract();

//在指数状态出现错误字符，错误处理

error(Wrong\_E);

state = 0;

}

break;

case 6://指数的符号后的第一个数字

cat(C, cur);

C = get\_char(inprog);

if (digit(C))

state = 7;

else

{

retract();

//指数部分符号后没有数字，错误处理

error(Wrong\_neg);

state = 0;

}

break;

case 7://指数的数字部分

cat(C, cur);

C = get\_char(inprog);

if (digit(C))

state = 7;

else

{

retract();

state = 0;

//识别结果，带有指数部分的浮点数

add(FLOATDIG, 0);

}

break;

case 8:// +状态

C = get\_char(inprog);

if (C == '=')

{

state = 0;

//识别结果：+=

add(OPTION, 1);

}

else

{

retract();

state = 0;

//识别结果：+

add(OPTION, 0);

}

break;

case 9:// -状态

C = get\_char(inprog);

if (C == '=')

{

state = 0;

//识别结果：-=

add(OPTION, 3);

}

else

{

retract();

state = 0;

//识别结果：-

add(OPTION, 2);

}

break;

case 10:// \*状态

C = get\_char(inprog);

if (C == '=')

{

state = 0;

//识别结果：\*=

add(OPTION, 5);

}

else

{

retract();

state = 0;

//识别结果：\*

add(OPTION, 4);

}

break;

case 11:// /状态

C = get\_char(inprog);

if (C == '=')

{

state = 0;

//识别结果：/=

add(OPTION, 7);

}

else if (C == '/')

//注释状态

{

while (C != '\n')

C = get\_char(inprog);

state = 0;

}

else if (C == '\*')

//注释状态

{

C = get\_char(inprog);

while (true)

{

while (C != '\*')

C = get\_char(inprog);

C = get\_char(inprog);

if (C == '/')

break;

}

state = 0;

}

else

{

retract();

state = 0;

//识别结果：/

add(OPTION, 6);

}

break;

case 12:// %状态

C = get\_char(inprog);

if (C == '=')

{

state = 0;

//识别结果：%=

add(OPTION, 9);

}

else

{

retract();

state = 0;

//识别结果：%

add(OPTION, 8);

}

break;

case 13:// ^状态

C = get\_char(inprog);

if (C == '=')

{

state = 0;

//识别结果：^=

add(OPTION, 11);

}

else

{

retract();

state = 0;

//识别结果：^

add(OPTION, 10);

}

break;

case 14:// &状态

C = get\_char(inprog);

if (C == '&')

{

state = 0;

//识别结果：&&

add(OTHER, 1);

}

else

{

retract();

state = 0;

//识别结果：&

add(OPTION, 13);

}

break;

case 15:// <状态

C = get\_char(inprog);

if (C == '=')

{

state = 0;

//识别结果：<=

add(OTHER, 4);

}

else

{

retract();

state = 0;

//识别结果：<

add(OTHER, 3);

}

break;

case 16:// >状态

C = get\_char(inprog);

if (C == '=')

{

state = 0;

//识别结果：>=

add(OTHER, 6);

}

else

{

retract();

state = 0;

//识别结果：>

add(OTHER, 5);

}

break;

case 17:// !状态

C = get\_char(inprog);

if (C == '=')

{

state = 0;

//识别结果：!=

add(OTHER, 9);

}

else

{

retract();

state = 0;

//识别结果：!

add(OTHER, 0);

}

break;

case 18:// =状态

C = get\_char(inprog);

if (C == '=')

{

state = 0;

//识别结果：==

add(OTHER, 8);

}

else

{

retract();

state = 0;

//识别结果：=

add(OTHER, 7);

}

break;

case 19:// |状态

C = get\_char(inprog);

if (C == '|')

{

state = 0;

//识别结果：||

add(OTHER, 2);

}

else

{

retract();

state = 0;

//识别结果：|

add(OPTION, 15);

}

break;

case 20:// 错误处理：识别到非法字符

error(Invalid\_character);

state = 0;

break;

default:

if (C == EOF) { ter = true; break; }

error(Other\_wrong);

state = 0;

break;

}

if (C == EOF)

ter = true;

if (ter)

//编译程序完成，完成收尾工作

{

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl

<< "\tLexical Analysis is successfully completed!" << endl

<< "\tIn your program, we get:" << endl

<< "\tLines:" << line - 1 << endl

<< "\tIndentifiers:\t" << Word.size() << endl

<< "\tIntegers:\t" << INT.size() << endl

<< "\tFloats:\t" << FLOAT.size() << endl

<< "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

break;

}

}

return OK;

}

//从缓存区读取一个字符，前向指针向前移动一个单位

char get\_char(ifstream& inprog)

{

char C = \*forwardp;

if (C == EOF)

//判断是否到达文件尾或到达双缓存区的两个边界

{

forwardp = buf + (forwardp + 1 - buf) % (half\_buf\_size \* 2);

if (forwardp == buf + half\_buf\_size)

getinput(inprog, buf\_right);

else if (forwardp == buf)

getinput(inprog, buf\_left);

else

return C;

C = get\_char(inprog);

}

else

forwardp = buf + (forwardp + 1 - buf) % (half\_buf\_size \* 2);

if (C == '\n')

line++;

return C;

}

//判断读入的字符是否是空字符

void get\_npc(ifstream& inprog, char& C)

{

if (C == ' ' || C == '\t' || C == '\n' || C == '\r' || C == '\0')

{

C = get\_char(inprog);

get\_npc(inprog, C);

}

}

//判断字符是否为字母

bool letter(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 digit(char& C)

{

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

return true;

else

return false;

}

//将字符C加入到记号缓存区中

void cat(char& C, int& cur)

{

token[cur] = C;

cur++;

}

//超前识别截止，前向指针后退，经过分析，前向指针后退时，不会碰到EOF，不会越界

void retract(void)

{

forwardp--;

}

//判断标识符是否为保留关键字

int reserve(char\* token)

{

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

if (strcmp(token, key[i]) == 0)

return i;

return -1;

}

//将整数从字符串转换为数字

long long get\_int(char\* buf)

{

char\* ptr = buf;

long long integer = 0LL;

while (digit(\*ptr))

{

integer = integer \* 10 + \*ptr - '0';

ptr++;

}

return integer;

}

int get\_pnum\_length(long long temp)

{

int length = 1;

while (temp > 10)

{

length++;

temp /= 10;

}

return length;

}

double get\_float(void)

{

char\* ptr = token;

double floatnum = 0;

while (digit(\*ptr))

{

floatnum = floatnum \* 10 + \*ptr - '0';

ptr++;

}

if (\*ptr == '.')

{

ptr++;

long long temp = get\_int(ptr);

int length = get\_pnum\_length(temp);

int power = pow(10, length);

floatnum += (double)temp / power;

}

while (digit(\*ptr)) ptr++;

if (\*ptr == 'E' || \*ptr == 'e')

{

ptr++;

int flag = 0;

if (\*ptr == '-')

{

flag = 1;

ptr++;

}

else if (\*ptr == '+')

ptr++;

long long num = get\_int(ptr);

if (flag)

num = -num;

double power = pow(10, num);

floatnum \*= power;

}

return floatnum;

}

//将识别出的记号添加到记号序列中

void add(int type, int num)

{

string word;

long long integer;

double floatnum;

int index;

Token temp;

switch (type)

{

case WORD:

//普通标识符

word = token;

index = Word.size();

Word.push\_back(word);

temp.type = WORD;

temp.index = index;

temp.line = line;

TOKEN.push\_back(temp);

cout << "<\tWORD\t\t" << word << "\t\t" << line << "\t>" << endl;

break;

case KEYWORD:

//关键字

temp.type = KEYWORD;

temp.index = num;

temp.line = line;

TOKEN.push\_back(temp);

cout << "<\tKEYWORD\t\t" << key[num] << "\t\t" << line << "\t>" << endl;

break;

case INTDIG:

//整数

integer = get\_int(token);

index = INT.size();

INT.push\_back(integer);

temp.type = INTDIG;

temp.index = index;

temp.line = line;

TOKEN.push\_back(temp);

cout << "<\tINTDIG\t\t" << integer << "\t\t" << line << "\t>" << endl;

break;

case FLOATDIG:

//浮点数

floatnum = get\_float();

index = FLOAT.size();

FLOAT.push\_back(floatnum);

temp.type = FLOATDIG;

temp.index = index;

temp.line = line;

TOKEN.push\_back(temp);

cout << "<\tFLOATDIG\t\t" << floatnum << "\t\t" << line << "\t>" << endl;

break;

case OPTION:

//操作符号

temp.type = OPTION;

temp.index = num;

temp.line = line;

TOKEN.push\_back(temp);

cout << "<\tOPTION\t\t" << opt[num] << "\t\t" << line << "\t>" << endl;

break;

case END:

//界符

temp.type = END;

temp.index = num;

temp.line = line;

TOKEN.push\_back(temp);

cout << "<\tEND\t\t" << End[num] << "\t\t" << line << "\t>" << endl;

break;

case OTHER:

//其他符号

temp.type = OTHER;

temp.index = num;

temp.line = line;

TOKEN.push\_back(temp);

cout << "<\tOTHER\t\t" << other[num] << "\t\t" << line << "\t>" << endl;

break;

default:

error(Other\_wrong);

break;

}

}

void error(ErrorType type)

{

switch (type)

{

case Wrong\_dot:

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl

<< "错误：浮点数的小数点后应当有数字" << endl

<< "行号：" << line << endl

<< "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

break;

case Wrong\_neg:

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl

<< "错误：浮点数指数部分的+或-号后应当有数字" << endl

<< "行号：" << line << endl

<< "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

break;

case Wrong\_E:

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl

<< "错误：浮点数指数部分E或e后只能出现（+/-）数字" << endl

<< "行号：" << line << endl

<< "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

break;

case Invalid\_character:

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl

<< "错误：出现C语言不支持的字符" << endl

<< "行号：" << line << endl

<< "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

break;

default:

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl

<< "错误：编译器运行时错误" << endl

<< "行号：" << line << endl

<< "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

break;

}

}