

编译原理

**词法分析程序设计原理与实现**

**实验报告**

学 院：

专 业：

学生姓名：

学 号：

北京交通大学

2023年10月

1. **程序功能描述**

打开存放C语言程序的txt文件，进行词法分析，输出标识符表和二元式序列组成的文件，其中第一个元素为符号本身，第二个元素为种类码。

1. **主要数据结构**

|  |  |
| --- | --- |
| 变量及类型 | 用途 |
| static char Reserve\_Word\_list [19][15] = {"void","int","float", "double", "if", "else","for","do","while","break", "case","char","continue","default", "return","struct","switch", "typedef","unsigned",}; | 设置二维数组Reserve\_Word\_list，保存经过补充的19个保留字。 |
| static char Operation\_Word\_list [36][5] = {"+", "-", "\*", "/", "<", "<=", ">", ">=", "=", "==","!=", ";", "(", ")", "^", ",", "\"", "\'", "#", "&","&&", "|", "||", "%", "~", "<<", ">>", "[", "]", "{","}", "\\", ".", "\?", ":", "!"}; | 设置界符运算符表Operation\_Word\_list |
| char Resource\_Code[10000]; | 将读取到的源程序储存 |
| char token[15] = { 0 }; | 将读取到的标识符储存 |

1. **程序结构描述**
2. 词法分析程序打开源文件，读取文件内容，直至遇上’$’文件结束符，然后读取结束。
3. 对读取的文件进行预处理，逐个扫描字符，去除注释中的内容，以及换行符、回车符、制表符等。
4. 开始扫描程序，如果是空格，继续扫描下一个字符，直到不是空格，然后判断如果是字母主要判断是标识符或者保留字；如果是数字，则进行数字的判断。如果不属于任何类型则是非法符号并报告错误。每次识别一个单词后，单词都会存在token[ ]中。然后确定这个单词的类别码，对于不同的单词类别做出不同的反应，如对于标识符则将其插入标识符表中。对于保留字则输出该保留字的类别码和助记符，然后进行下一个单词的识别，结束条件是遇到$符号，如果无法被程序识别会输出错误。
5. 最终输出结果为二元式序列组成的文件，第一个元素为种类码，第二个元素为符号本身。

提前定义好数组保存相关的内容，如图3.1所示。

|  |  |
| --- | --- |
| 变量及类型 | 用途 |
| static char Reserve\_Word\_list [19][15] = {"void","int","float", "double", "if", "else","for","do","while","break", "case","char","continue","default", "return","struct","switch", "typedef","unsigned",}; | 设置二维数组Reserve\_Word\_list，保存经过补充的19个保留字。 |
| static char Operation\_Word\_list [36][5] = {"+", "-", "\*", "/", "<", "<=", ">", ">=", "=", "==","!=", ";", "(", ")", "^", ",", "\"", "\'", "#", "&","&&", "|", "||", "%", "~", "<<", ">>", "[", "]", "{","}", "\\", ".", "\?", ":", "!"}; | 设置界符运算符表Operation\_Word\_list |
| char Resource\_Code[10000]; | 将读取到的源程序储存 |
| char token[15] = { 0 }; | 将读取到的标识符储存 |

图3.1

状态转换图如图3.2所示。

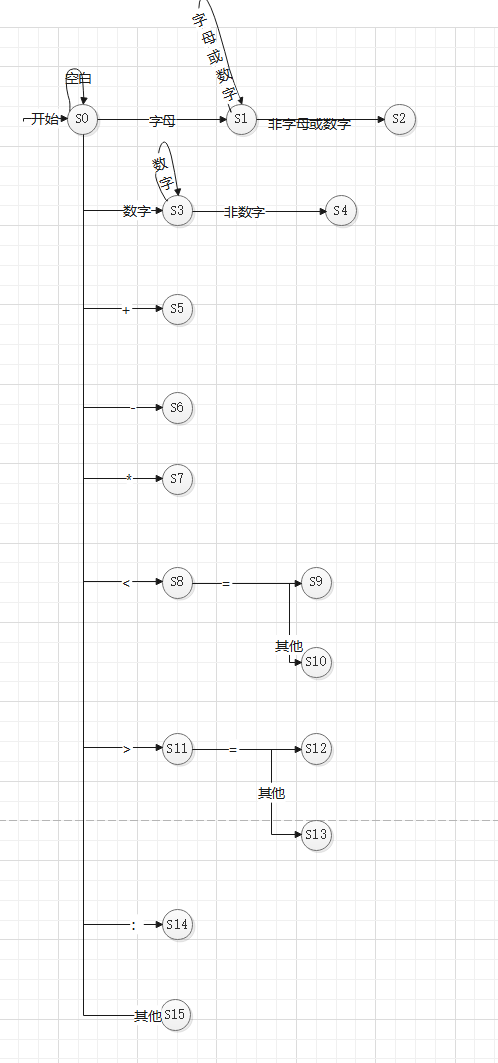


图3.2 状态转换图

程序流程图如图3.3所示：

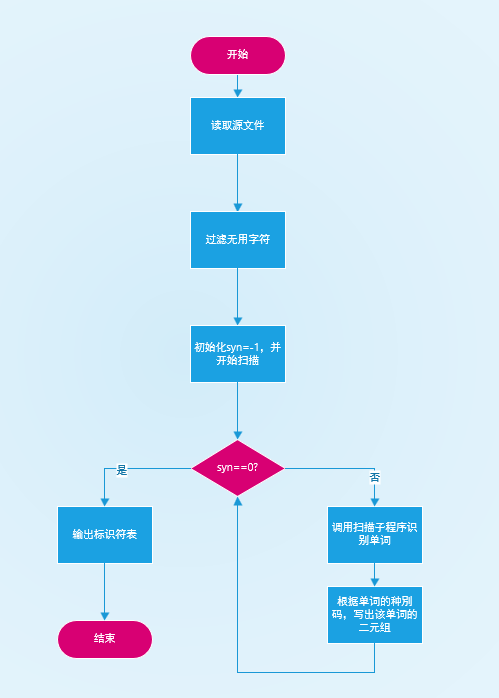


图3.4 程序流程图

1. **函数定义及函数间的调用关系**

|  |  |
| --- | --- |
| 函数名称 | 函数功能描述 |
| static char Reserve\_Word\_list [19][15] = {"void","int","float", "double", "if", "else","for","do","while","break", "case","char","continue","default", "return","struct","switch", "typedef","unsigned",}; | 设置二维数组Reserve\_Word\_list，保存经过补充的19个保留字。 |
| static char Operation\_Word\_list [36][5] = {"+", "-", "\*", "/", "<", "<=", ">", ">=", "=", "==","!=", ";", "(", ")", "^", ",", "\"", "\'", "#", "&","&&", "|", "||", "%", "~", "<<", ">>", "[", "]", "{","}", "\\", ".", "\?", ":", "!"}; | 设置界符运算符表Operation\_Word\_list |
| static char Flag\_table [1000][50] = { "" }； | 定义二维数组Flag\_table为标识符表。 |
| Int searchReserveWord(char Reserve\_Word\_list[][15], char s[]) | 用来查找保留字 |
| bool isLetter(char letter)  用来判断是否是字母 | 用来判断是否是字母 |
| void pretreatment(char r[], int Code\_Pointer) | 编译预处理 |
| void Scanner(int &syn, char Resource\_Code [], char token[], int & Code\_Pointer) | 逐个扫描字符，并判断种类 |
| bool IsDigit(char digit) | 用来判断是否是数字 |

函数间的调用关系如图3.5所示：

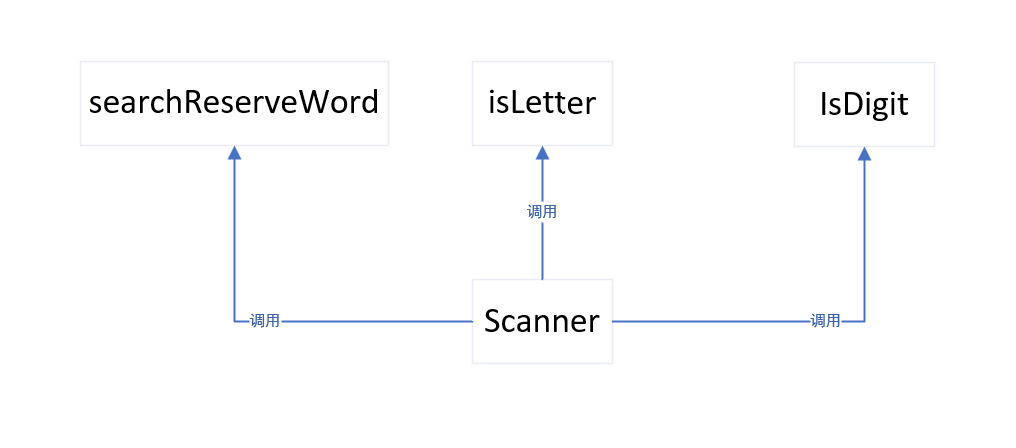


图3.5 函数间调用关系

1. **程序测试**

测试用例1：test1.cpp

测试程序1如下，此程序中没有错误：

#include <stdio.h>

#include <string.h>

#include <pthread.h>

#include <unistd.h>

int current = 0; // producer运行加1，consumer运行减1

int buf[10];

int in = 0, out = 0;

int items = 0, spaces = 10;

int flag=1; // 标记线程结束运行

pthread\_mutex\_t mutex = PTHREAD\_MUTEX\_INITIALIZER;

pthread\_cond\_t notfull = PTHREAD\_COND\_INITIALIZER; // 缓冲区不满

pthread\_cond\_t notempty = PTHREAD\_COND\_INITIALIZER; // 缓冲区不空

void \*producer( void \*arg ) {

while( flag ) {

pthread\_mutex\_lock( &mutex ); // 为保证条件变量不会因为多线程混乱，所以先加锁

while( !spaces ) { // 避免“惊群”效应，避免因其他线程实现得到事件而导致该线程“假醒”

pthread\_cond\_wait( &notfull, &mutex );

}

buf[in] = ++current;

in = ( in + 1 ) % 10;

items++;

spaces--;

printf( "producer %zu , current = %d\n", pthread\_self(), current );

for( int i = 0; i < 10; i++ ) {

printf( "%-4d", buf[i] );

}

printf( "\n\n" );

pthread\_cond\_signal( &notempty );

pthread\_mutex\_unlock( &mutex );

}

pthread\_exit( NULL );

}

void \*consumer( void \*arg ) {

while( flag ) {

pthread\_mutex\_lock( &mutex );

while( !items ) {

pthread\_cond\_wait( &notempty, &mutex );

}

buf[out] = -1;

out = ( out + 1 ) % 10;

current--;

items--;

spaces++;

printf( "consumer %zu , current = %d\n", pthread\_self(), current );

for( int i = 0; i < 10; i++ ) {

printf( "%-4d", buf[i] );

}

printf( "\n\n" );

pthread\_cond\_signal( &notfull );

pthread\_mutex\_unlock( &mutex );

}

pthread\_exit( NULL );

}

int main() {

memset( buf, -1, sizeof(buf) );

flag = 1;

pthread\_t pro[10], con[10];

int i = 0;

for( int i = 0; i < 10; i++ ) {

pthread\_create( &pro[i], NULL, producer, NULL );

pthread\_create( &con[i], NULL, consumer, NULL );

}

sleep(1); // 让线程运行一秒

flag = 0;

for( int i = 0; i < 10; i++ ) {

pthread\_join( pro[i], NULL );

pthread\_join( con[i], NULL );

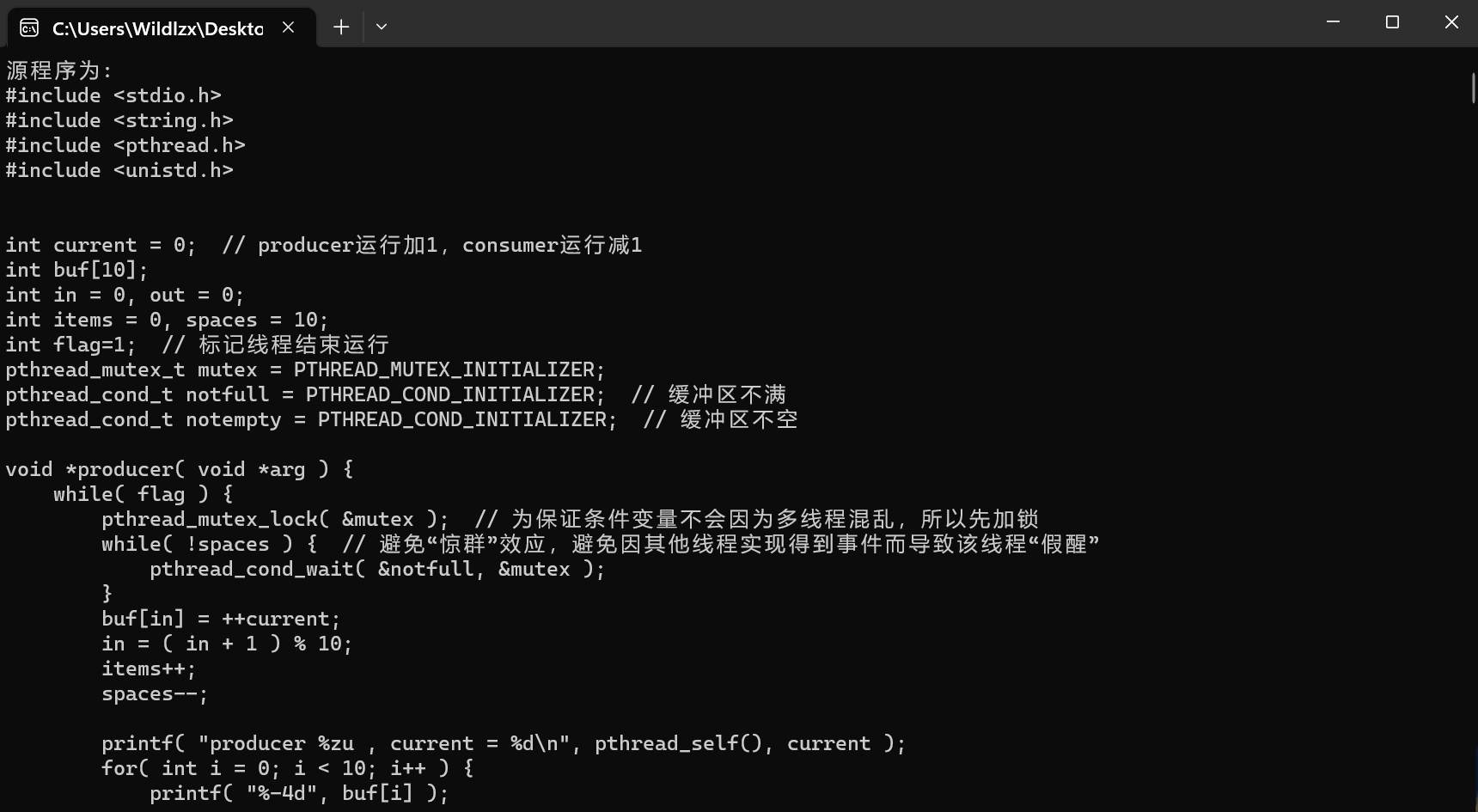
}

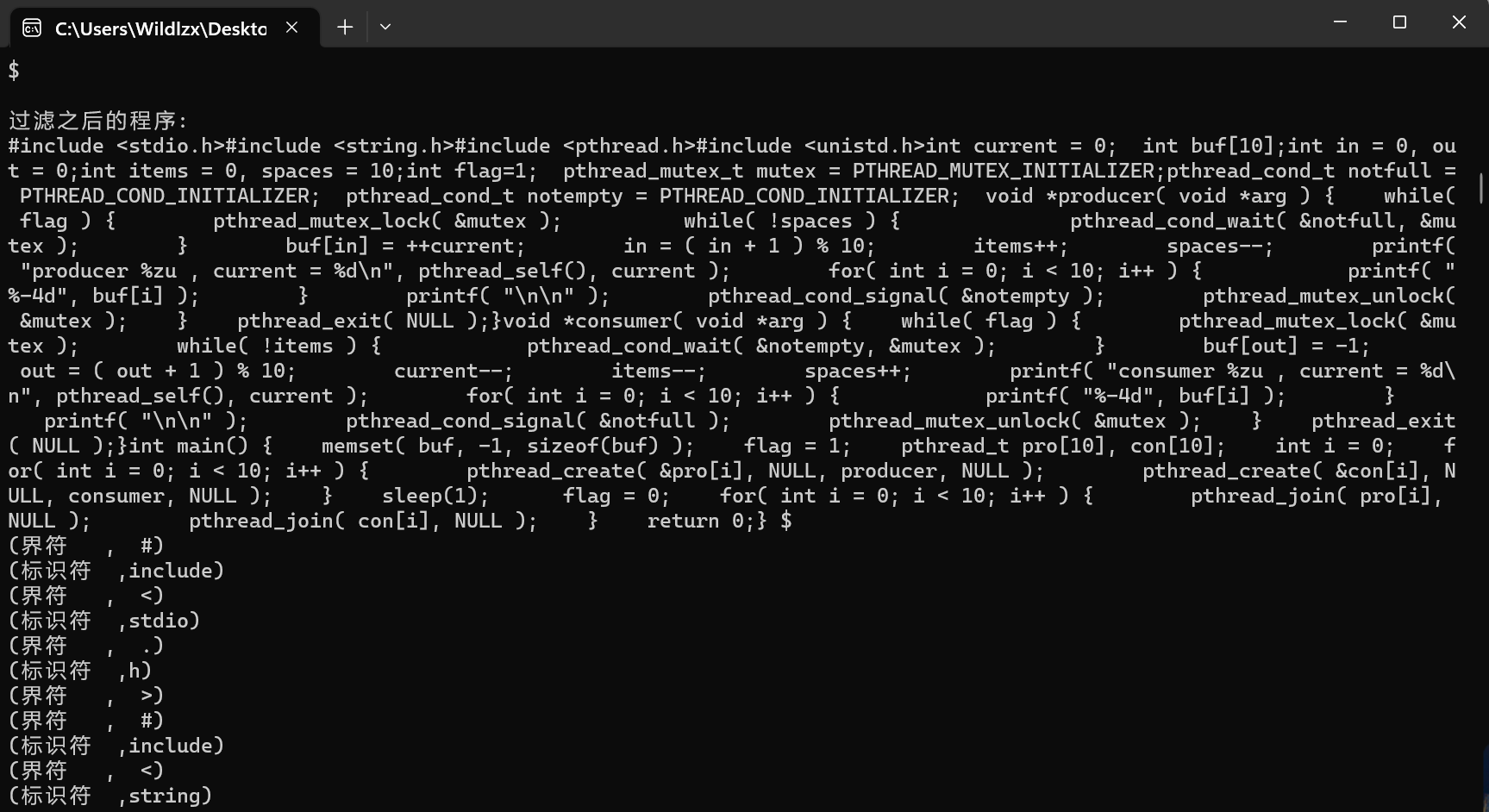
return 0;

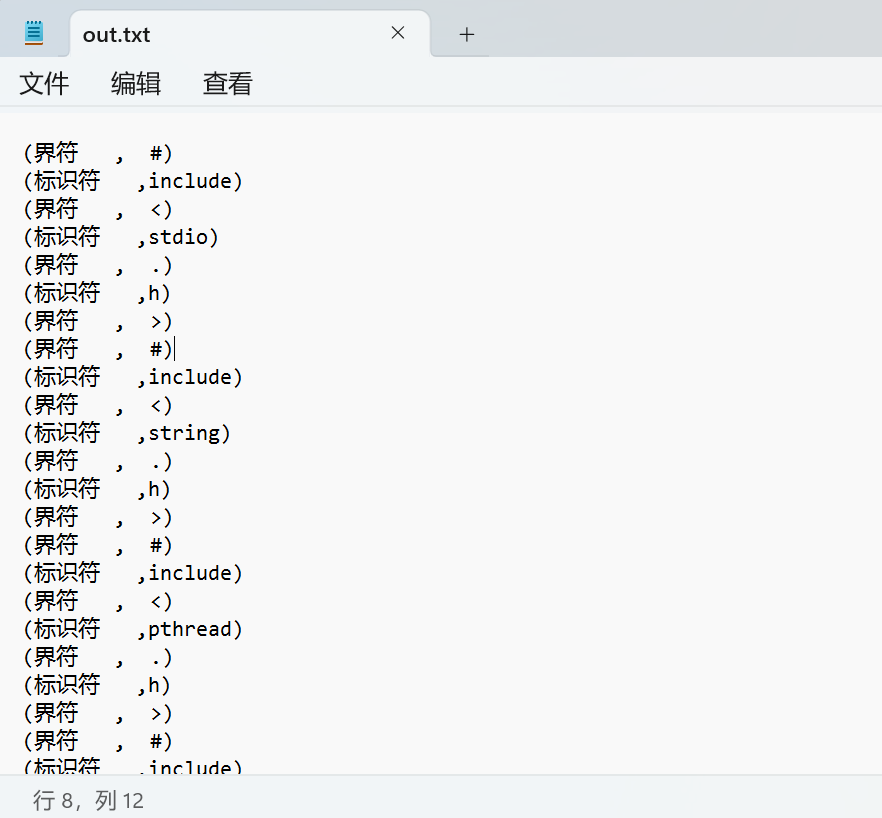
}

图4.1 测试用例1

程序执行结果：（输出至out.txt）







测试用例2：test2.cpp

有错误的字符@：

#include <stdio.h>

#include <string.h>

int main(){

for( int i = 0; i < 10; i++ ) {

printf("123");

}

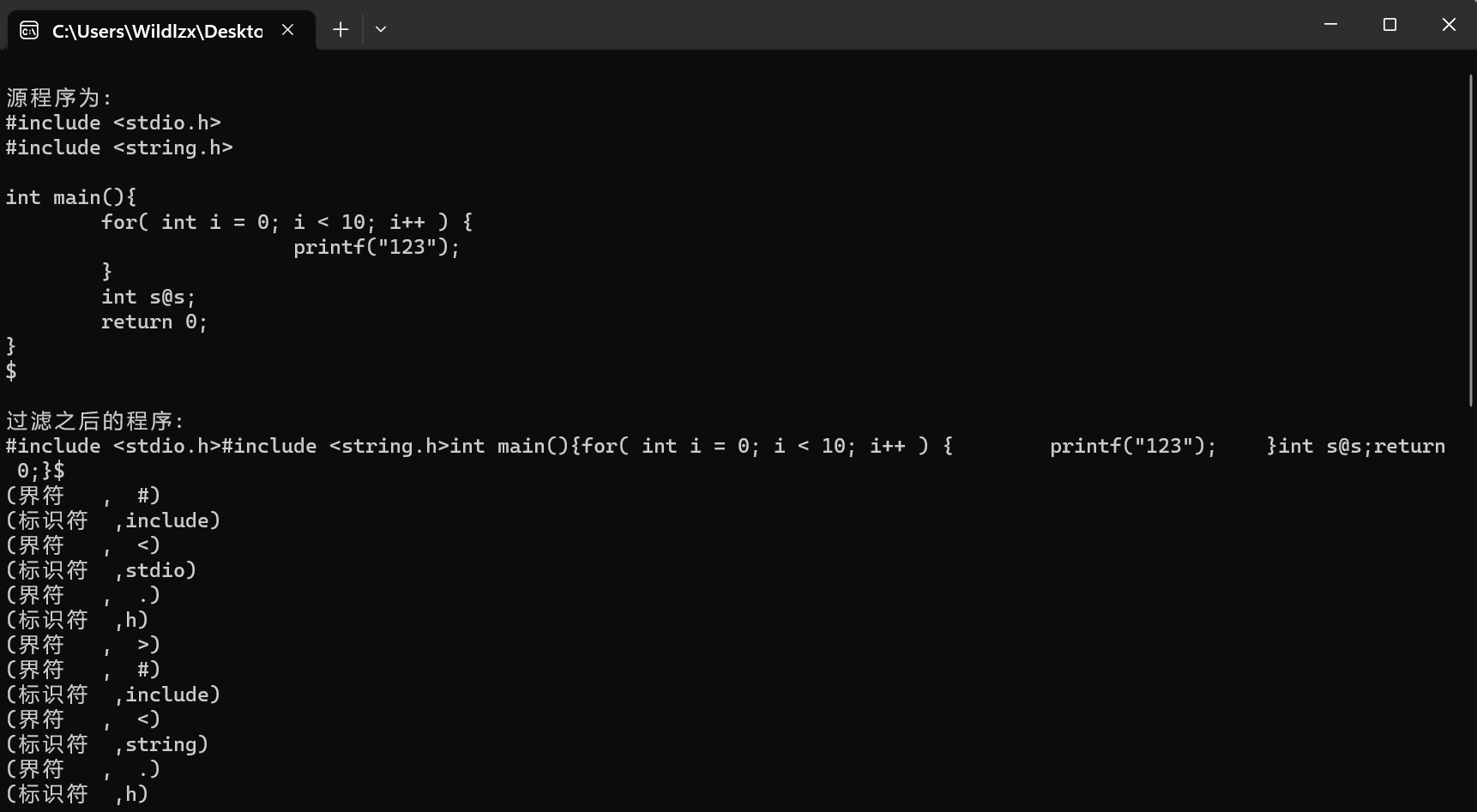
int s@s;

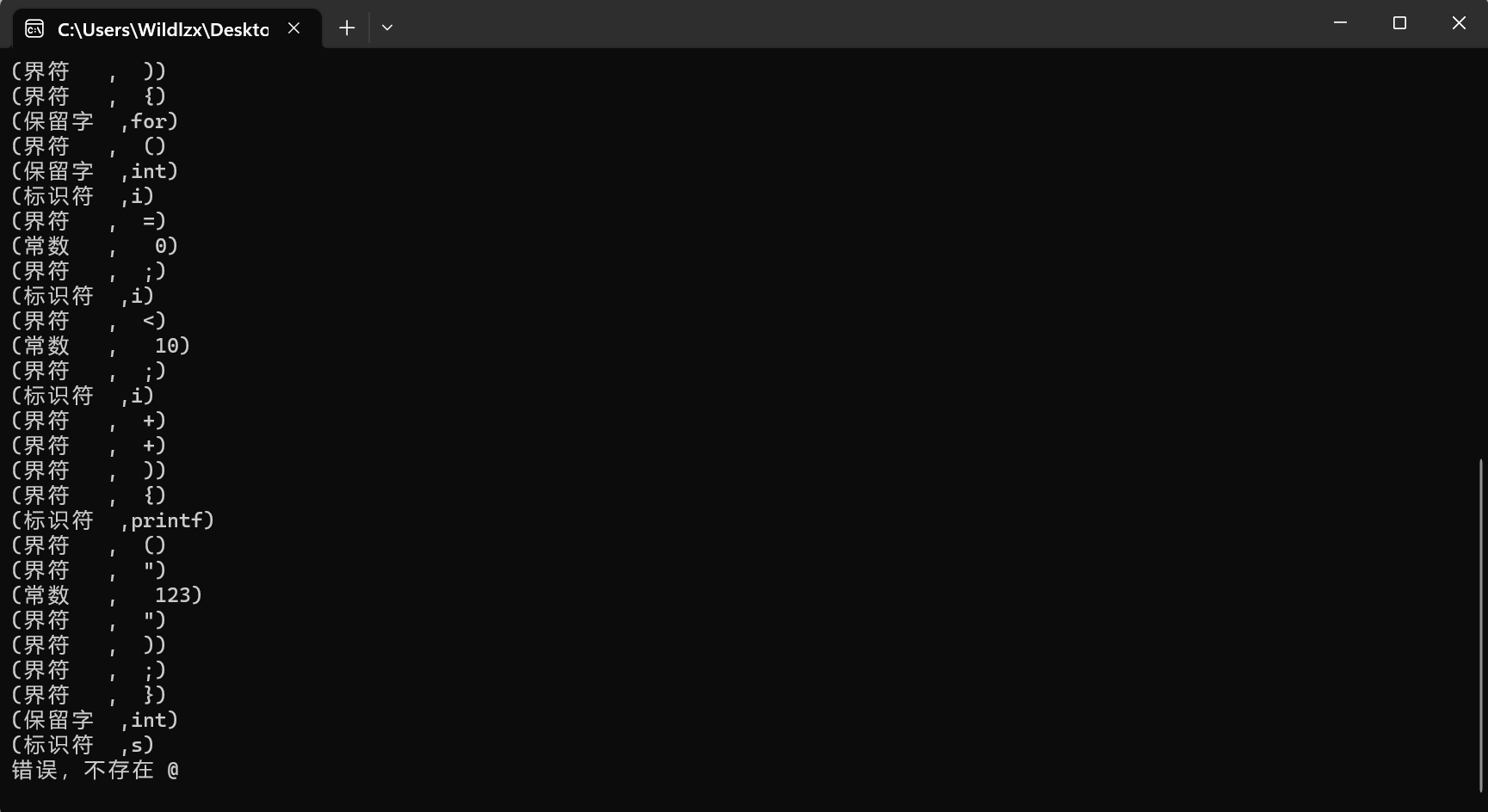
return 0;

}

图4.2 测试用例2

程序执行结果：





可见程序可以识别不应出现的字符，发出错误。

图4.3 测试结果

**五．实验结果分析**

由测试结果可见，在本次实验中我编写的程序能够成功实现词法分析功能。并且，由于我扩展了保留字和符号，以及增加了数据结构初始定义的大小，增强了程序的鲁棒性，对于不同类的输入程序都能很好的完成词法分析的功能。最后，由于实现的功能函数化，便于扩展，为之后做一个简单的编译器提供了词法分析的接口。

**六．研究性教学实验感受**

刚开始写实验的时候有些无从下手，后面通过看书和查阅资料逐步了解需要完成任务和实现的方法，然后开始逐步实现应有的功能函数。通过这次实验让我对词法分析的理解更深了一步，这也是做实验和学习这门课程的目的所在。一步步完善实验的过程也是在增强自己的知识。

**七. 附件（源代码列表）**

#pragma warning(disable:4996)

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <iostream>

#include <conio.h>

using namespace std;

//词法分析程序

//首先定义类别码

/\*

第一类：标识符 letter(letter | digit)\* 无穷集

第二类：常数 (digit)+ 无穷集

第三类：保留字(32)

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

第四类：界符 ‘/\*’、‘//’、 () { } [ ] " " '

第五类：运算符 <、<=、>、>=、=、+、-、\*、/、^、

对所有可数符号进行编码：

<$,0>

<auto,1>

...

<while,32>

<+，33>

<-,34>

<\*,35>

</,36>

<<,37>

<<=,38>

<>,39>

<>=,40>

<=,41>

<==,42>

<!=,43>

<;,44>

<(,45>

<),46>

<^,47>

<,,48>

<",49>

<',50>

<#,51>

<&,52>

<&&,53>

<|,54>

<||,55>

<%,56>

<~,57>

<<<,58>左移

<>>,59>右移

<[,60>

<],61>

<{,62>

<},63>

<\,64>

<.,65>

<?,66>

<:,67>

<!,68>

"[","]","{","}"

<常数99 ,数值>

<标识符100 ，标识符指针>

\*/

//全局变量，保留字表

static char Reserve\_Word\_list[19][15] = {

"void","int","float", "double", "if", "else",

"for","do","while" "break", "case", "char",

"continue","default", "return", "struct",

"switch", "typedef","unsigned",

};

//界符运算符表

static char Operation\_Word\_list[36][10] = {

"+", "-", "\*", "/", "<", "<=", ">", ">=", "=", "==",

"!=", ";", "(", ")", "^", ",", "\"", "\'", "#", "&",

"&&", "|", "||", "%", "~", "<<", ">>", "[", "]", "{",

"}", "\\", ".", "\?", ":", "!"

};

static char Flag\_table[1000][50] = { "" };//标识符表

/\*\*\*\*\*\*\*\*查找保留字\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int searchReserveWord(char Reserve\_Word\_list[][15], char s[])

{

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

{

if (strcmp(Reserve\_Word\_list[i], s) == 0)

{//若成功查找，则返回类别码

return i + 1;//返回类别码

}

}

return -1;//否则返回-1，代表查找不成功，即为标识符

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*判断是否为字母\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

bool isLetter(char letter)

{

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

{

return true;

}

else

{

return false;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*判断是否为数字\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

bool IsDigit(char digit)

{

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

{

return true;

}

else

{

return false;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*编译预处理，取出无用的字符和注释\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void pretreatment(char r[], int Code\_Pointer)

{

char tempString[10000];

int count = 0;

for (int i = 0; i <= Code\_Pointer; i++)

{

if (r[i] == '/'&&r[i + 1] == '/')

{//若为单行注释“//”,则去除注释后面的东西，直至遇到回车换行

while (r[i] != '\n')

{

i++;//向后扫描

}

}

if (r[i] == '/'&&r[i + 1] == '\*')

{//若为多行注释“/\* 。。。\*/”则去除该内容

i += 2;

while (r[i] != '\*' || r[i + 1] != '/')

{

i++;//继续扫描

if (r[i] == '$')

{

printf("注释出错，没有找到 \*/，程序结束！！！\n");

exit(0);

}

}

i += 2;//跨过“\*/”

}

if (r[i] != '\n'&&r[i] != '\t'&&r[i] != '\v'&&r[i] != '\r')

{//若出现无用字符，则过滤；否则加载

tempString[count++] = r[i];

}

}

tempString[count] = '\0';

strcpy(r, tempString);//过滤之后的源程序

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*扫描分析，状态转换\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Scanner(int &syn, char Resource\_Code[], char token[], int &Code\_Pointer)

{//根据DFA的状态转换图设计

int i, count = 0;//count用来做token[]的指示器，收集有用字符

char ch;//作为判断使用

ch = Resource\_Code[Code\_Pointer];

while (ch == ' ')

{//过滤空格

Code\_Pointer++;

ch = Resource\_Code[Code\_Pointer];

}

for (i = 0; i < 15; i++)

{//初始化

token[i] = '\0';

}

if (isLetter(Resource\_Code[Code\_Pointer]))

{//开头为字母

token[count++] = Resource\_Code[Code\_Pointer];//收集

Code\_Pointer++;//下移

while (isLetter(Resource\_Code[Code\_Pointer]) || IsDigit(Resource\_Code[Code\_Pointer]))

{//后跟字母或数字

token[count++] = Resource\_Code[Code\_Pointer];//收集

Code\_Pointer++;//下移

}//多读了一个字符既是下次将要开始的指针位置

token[count] = '\0';

syn = searchReserveWord(Reserve\_Word\_list, token);//查表找到类别码

if (syn == -1)

{//若不是保留字则是标识符

syn = 100;//标识符类别码

}

return;

}

else if (IsDigit(Resource\_Code[Code\_Pointer]))

{//首字符为数字

while (IsDigit(Resource\_Code[Code\_Pointer]))

{//后跟数字

token[count++] = Resource\_Code[Code\_Pointer];//收集

Code\_Pointer++;

}//多读了一个字符既是下次将要开始的指针位置

token[count] = '\0';

syn = 99;//常数类别码

}

else if (ch == '+' || ch == '-' || ch == '\*' || ch == '/' || ch == ';' || ch == '(' || ch == ')' || ch == '^'

|| ch == ',' || ch == '\"' || ch == '\'' || ch == '~' || ch == '#' || ch == '%' || ch == '['

|| ch == ']' || ch == '{' || ch == '}' || ch == '\\' || ch == '.' || ch == '\?' || ch == ':')

{//若为运算符或者界符，查表得到结果

token[0] = Resource\_Code[Code\_Pointer];

token[1] = '\0';//形成单字符串

for (i = 0; i < 36; i++)

{//查运算符界符表

if (strcmp(token, Operation\_Word\_list[i]) == 0)

{

syn = 33 + i;

break;//查到即推出

}

}

Code\_Pointer++;//指针下移，为下一扫描做准备

return;

}

else if (Resource\_Code[Code\_Pointer] == '<')

{//<,<=,<<

Code\_Pointer++;

if (Resource\_Code[Code\_Pointer] == '=')

{

syn = 38;

}

else if (Resource\_Code[Code\_Pointer] == '<')

{//左移

Code\_Pointer--;

syn = 58;

}

else

{

Code\_Pointer--;

syn = 37;

}

Code\_Pointer++;//指针下移

return;

}

else if (Resource\_Code[Code\_Pointer] == '>')

{//>,>=,>>

Code\_Pointer++;

if (Resource\_Code[Code\_Pointer] == '=')

{

syn = 40;

}

else if (Resource\_Code[Code\_Pointer] == '>')

{

syn = 59;

}

else

{

Code\_Pointer--;

syn = 39;

}

Code\_Pointer++;

return;

}

else if (Resource\_Code[Code\_Pointer] == '=')

{//=.==

Code\_Pointer++;

if (Resource\_Code[Code\_Pointer] == '=')

{

syn = 42;

}

else

{

Code\_Pointer--;

syn = 41;

}

Code\_Pointer++;

return;

}

else if (Resource\_Code[Code\_Pointer] == '!')

{//!,!=

Code\_Pointer++;

if (Resource\_Code[Code\_Pointer] == '=')

{

syn = 43;

}

else

{

syn = 68;

Code\_Pointer--;

}

Code\_Pointer++;

return;

}

else if (Resource\_Code[Code\_Pointer] == '&')

{//&,&&

Code\_Pointer++;

if (Resource\_Code[Code\_Pointer] == '&')

{

syn = 53;

}

else

{

Code\_Pointer--;

syn = 52;

}

Code\_Pointer++;

return;

}

else if (Resource\_Code[Code\_Pointer] == '|')

{//|,||

Code\_Pointer++;

if (Resource\_Code[Code\_Pointer] == '|')

{

syn = 55;

}

else

{

Code\_Pointer--;

syn = 54;

}

Code\_Pointer++;

return;

}

else if (Resource\_Code[Code\_Pointer] == '$')

{//结束符

syn = 0;//类别码为0

}

else

{//不能被以上词法分析识别，则出错。

printf("错误，不存在 %c \n", ch);

getchar();

exit(0);

}

}

int main()

{

//打开一个文件，读取其中的源程序

char Resource\_Code[10000];

char token[15] = { 0 };

int syn = -1, i;//初始化

int Code\_Pointer = 0;//源程序指针

FILE \*fp, \*fp1;

if ((fp = fopen("read.txt", "r")) == NULL)

{//打开源程序

cout << "文件无法打开！";

getchar();

exit(0);

}

Resource\_Code[Code\_Pointer] = fgetc(fp);

while (Resource\_Code[Code\_Pointer] != '$')

{//将源程序读入Resource\_Code[]数组

Code\_Pointer++;

Resource\_Code[Code\_Pointer] = fgetc(fp);

}

Resource\_Code[++Code\_Pointer] = '\0';

fclose(fp);

cout << endl << "源程序为:" << endl;

cout << Resource\_Code << endl;

//对源程序进行过滤

pretreatment(Resource\_Code, Code\_Pointer);

cout << endl << "过滤之后的程序:" << endl;

cout << Resource\_Code << endl;

Code\_Pointer = 0;//从头开始读

if ((fp1 = fopen("out.txt", "w+")) == NULL)

{//打开源程序

cout << "文件无法打开！";

exit(0);

}

while (syn != 0)

{

//启动扫描

Scanner(syn, Resource\_Code, token, Code\_Pointer);

if (syn == 100)

{//标识符

for (i = 0; i < 1000; i++)

{//插入标识符表中

if (strcmp(Flag\_table[i], token) == 0)

{//已在表中

break;

}

if (strcmp(Flag\_table[i], "") == 0)

{//查找空间

strcpy(Flag\_table[i], token);

break;

}

}

printf("(标识符 ,%s)\n", token);

fprintf(fp1, "(标识符 ,%s)\n", token);

}

else if (syn >= 1 && syn <= 32)

{//保留字

printf("(保留字 ,%s)\n", Reserve\_Word\_list[syn - 1]);

fprintf(fp1, "(保留字 ,%s)\n", Reserve\_Word\_list[syn - 1]);

}

else if (syn == 99)

{//const 常数

printf("(常数 , %s)\n", token);

fprintf(fp1, "(常数 , %s)\n", token);

}

else if (syn >= 33 && syn <= 68)

{

printf("(界符 , %s)\n", Operation\_Word\_list[syn - 33]);

fprintf(fp1, "(界符 , %s)\n", Operation\_Word\_list[syn - 33]);

}

}

fclose(fp1);

getchar();

return 0;

}