# 词法分析程序

## 一、实验题目及要求

### 1. 实验题目：

c 语言词法分析程序的设计与实现

### 2. 实验要求：

* 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。
  + 输入文件(测试文件)为当前运行文件夹下的 input.txt，输出文件为当前运行文件夹下的 output.txt 。
  + 能够识别输入文件中的每个单词，并输出该单词的记号: 每个单词属于什么记号、记号的个数、记号的属性需要自行设计并说明。
  + 输出格式可自行定义并说明，例如 <记号，属性>、行数:<记号，属性> 等。
* 可以识别并跳过源程序中的注释。
* 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。
  + 字符总数中是否包含换行、空格、退格等分隔符需要进行说明，统计结果报告在输出文件中。
* 检查源程序中存在的词法错误，并报告错误所在的位置。
  + 错误内容、错误类别等需要进行设计并说明。
* 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，即可检查并报告源程序中存在的所有词法错误。
  + 源程序中出现的错误需要依次报告在输出文件中。
  + 错误输出格式可以自行定义并说明，例如 ERROR(错误id)：错误位置，错误说明等。

## 程序设计说明

问题分析：

1. 文件的读入

采用双重缓冲区读入文件，在缓冲区结尾添加文件结束符EOF作为判断缓冲区结束并加载下一缓冲区的哨兵。

**int** i;

for (i = 0; i < MAXLENGTH - 1; i++) {

**char** ch = fgetc(inputfile);

if (ch == EOF) {

buffer1[i] = EOF;

break;

}

buffer1[i] = ch;

}

buffer1[MAXLENGTH-1] = EOF;

在获取下一个forward指针时，完成对缓冲区的结束判断及加载。

forward++;

if(\*forward == EOF){

if (forward == &buffer1[MAXLENGTH-1]) {

loadbuffer2(inputfile);

forward = &buffer2[0];

return forward;

}

else if (forward == &buffer2[MAXLENGTH-1]) {

loadbuffer1(inputfile);

forward = &buffer1[0];

return forward;

}

else {

endofthefile = true;

return forward;

}

}

else {

return forward;

}

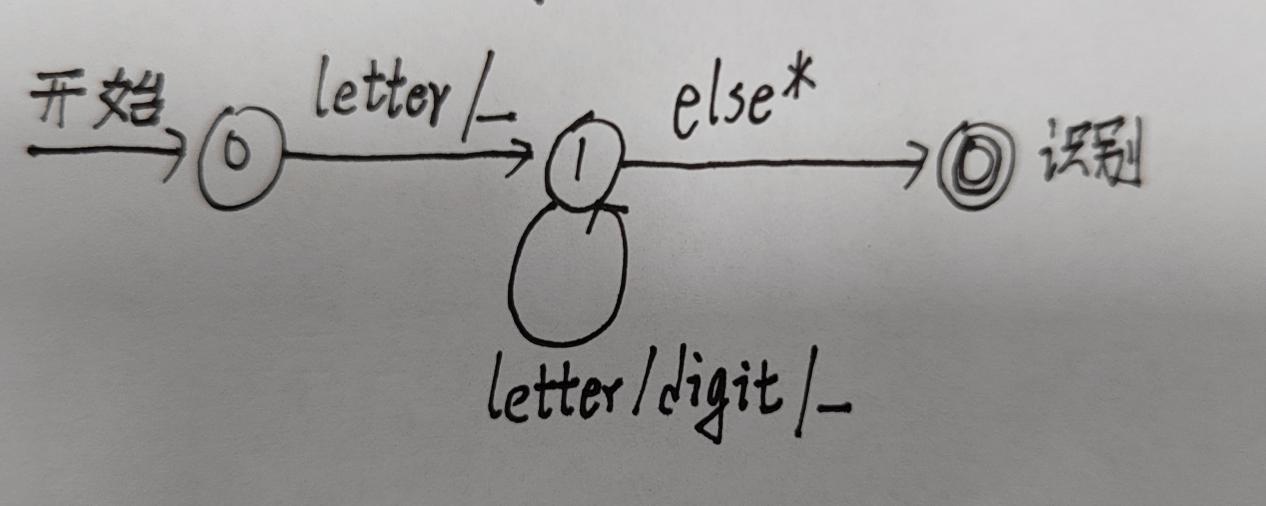
当获取到下一个forward指向EOF且未在缓冲区结尾，则说明文件读取完毕，关闭lex词法分析主程序。

采用双重缓冲区，可以更加高效的读取文件，将大量字符串一次性读入缓冲区；可以进行超前扫描该词素后面的若干个字符；当需要回退forward指针时，可以轻松回退到上一个字符；双重缓冲区使得一次可以超前扫描的字符串长度上限变为两个缓冲区大小。

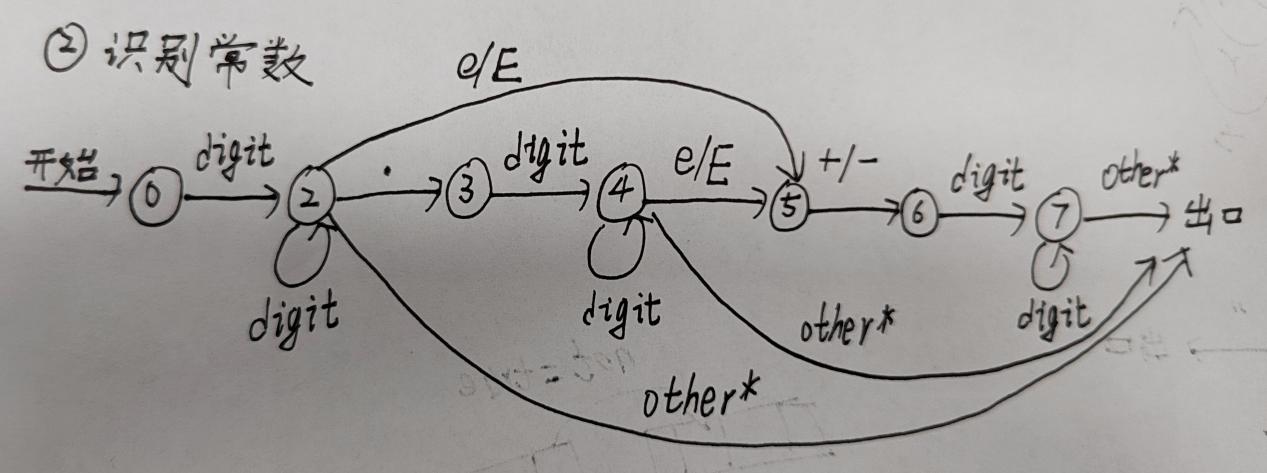
1. 问题核心

问题的核心是构建状态机，状态机的状态分为初始状态和其他状态当识别到有效字符时，状态机状态完成改变。

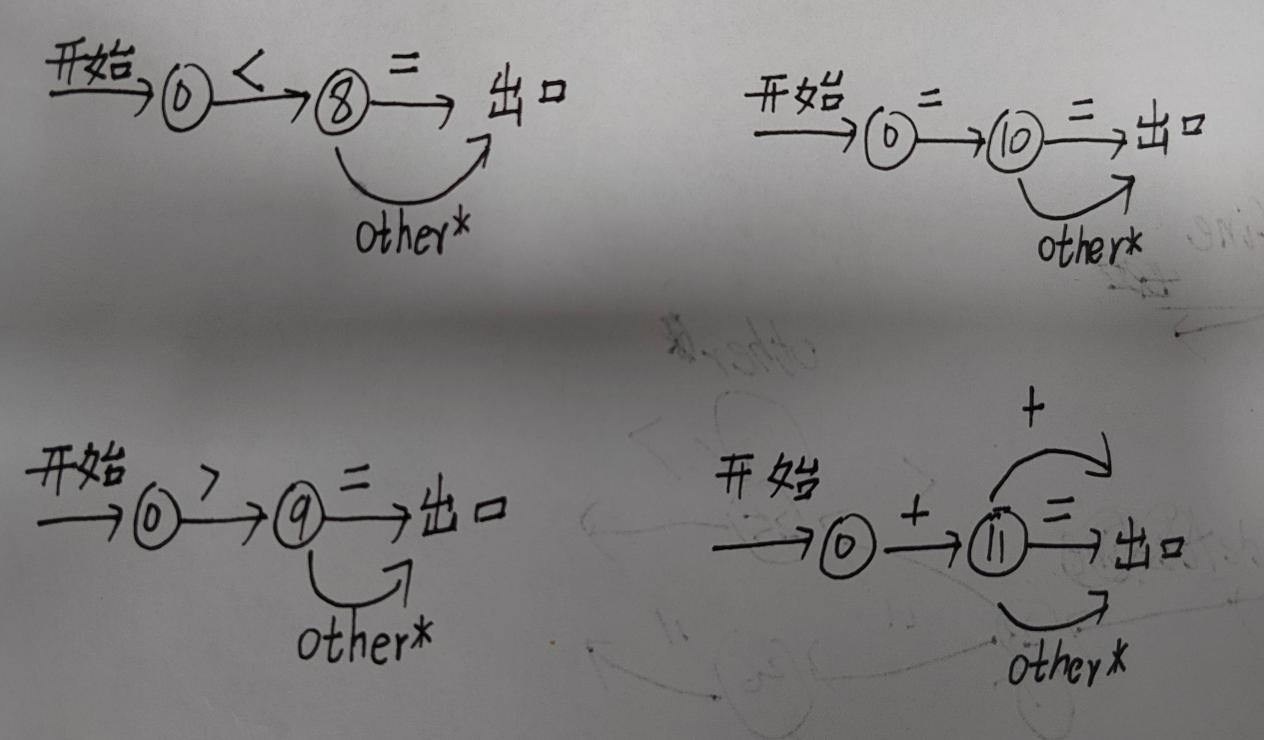
1. 识别标志符的状态机：

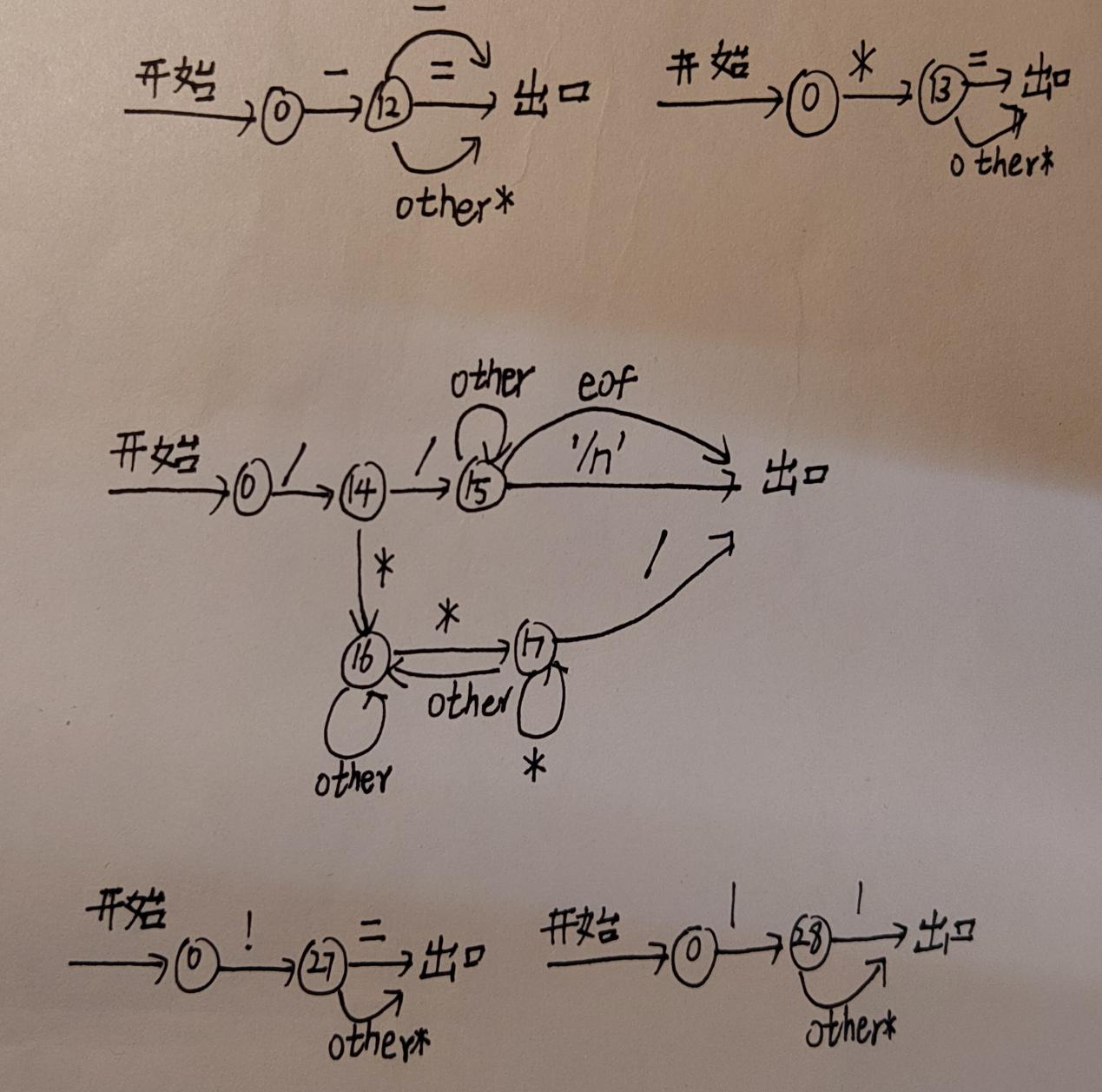


1. 识别常数的状态机：

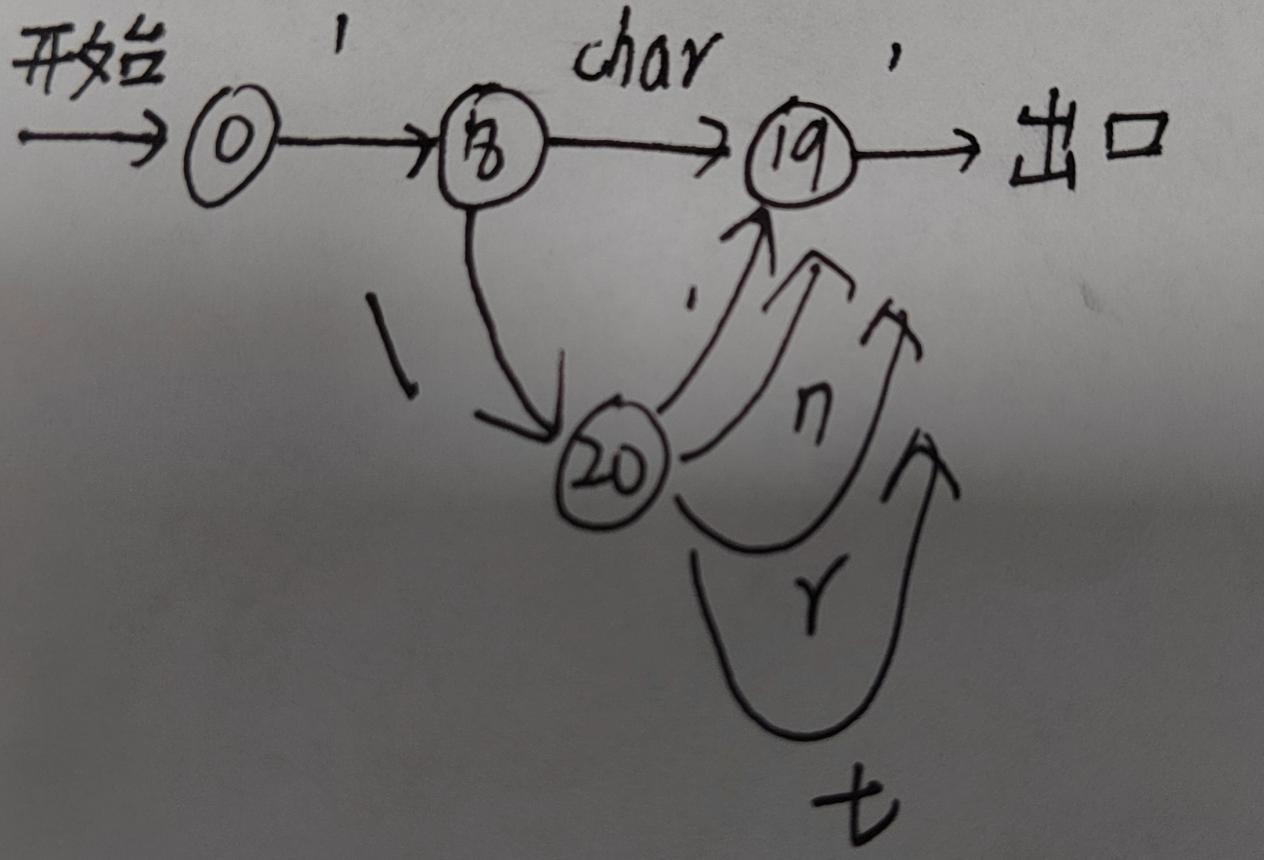


1. 识别运算符及注释的状态机：

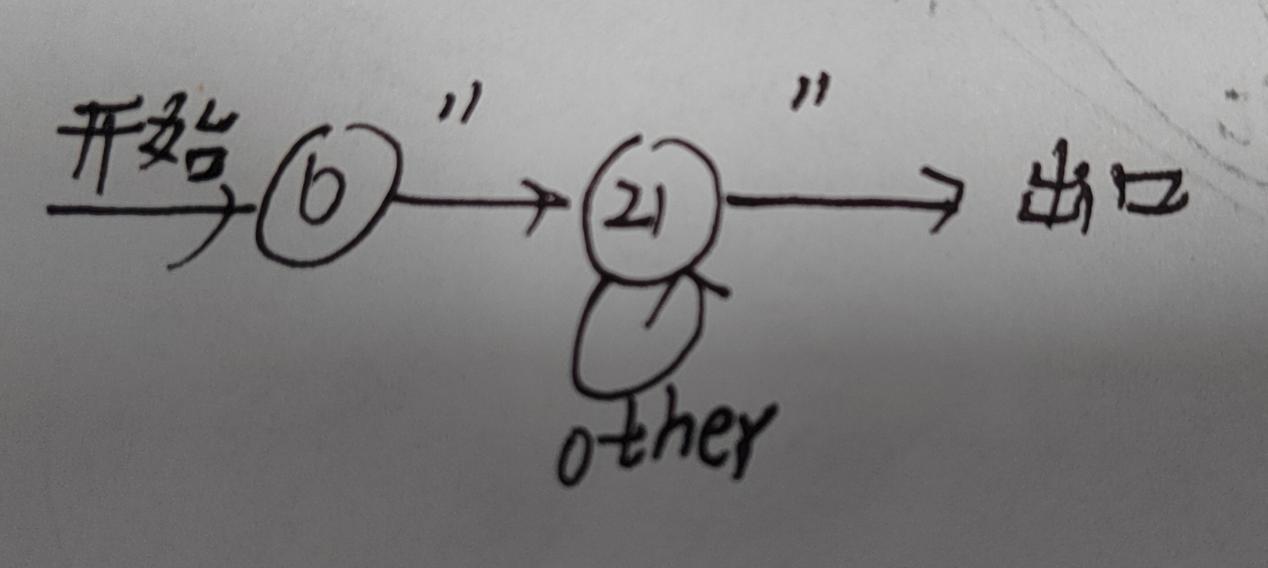




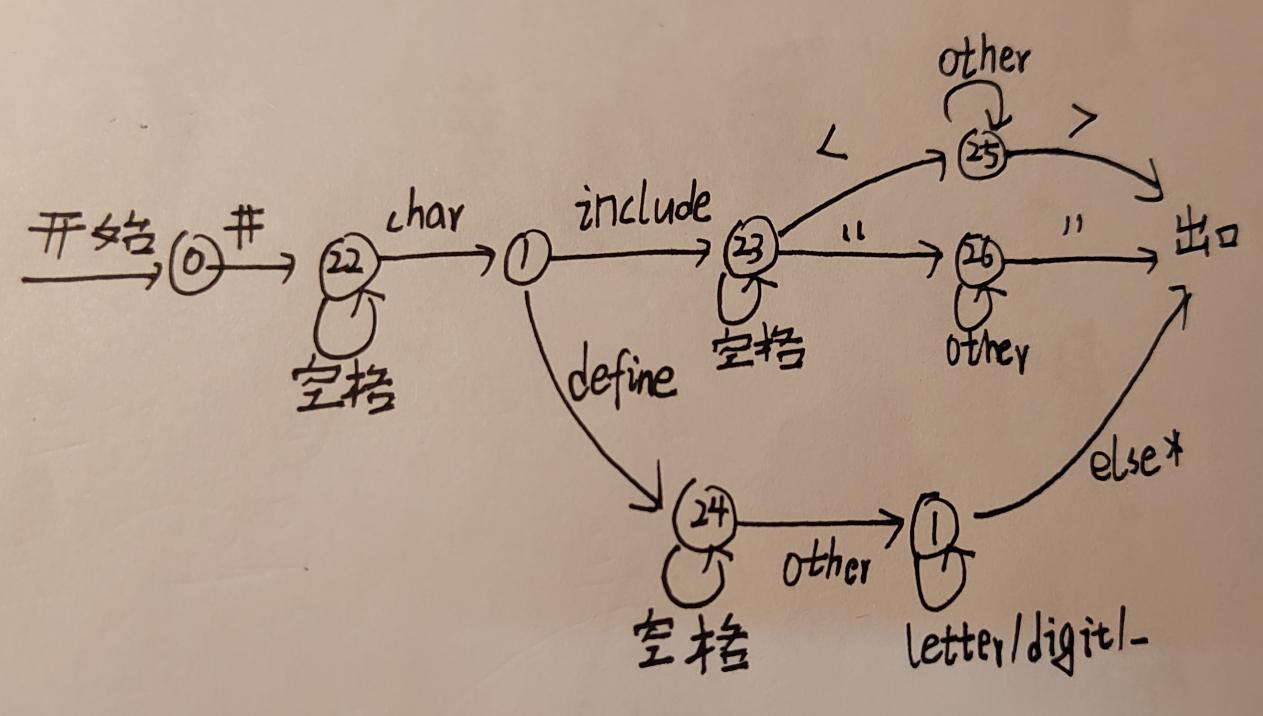
1. 识别单个字符的状态机：



1. 识别字符串的状态机：



1. 识别宏定义的状态机：



### 定义的全局常量

单个缓冲区大小为10字节：

#define MAXLENGTH 10 *// 缓冲区大小*

### 定义的全局变量

当前状态机状态：

**int** state; *// 当前状态指示*

当前状态机读入的字符：

**char** C; *// 存放当前读入的字符*

关键字查询结果：

**int** iskey= -1; *// 表示识别出的单词是用户自定义标志符，否则识别出的单词是关键字，其值为关键字的记号。*

存放正在识别的记号字符串：

**char** token[100] = ""; *// 存放当前正在识别的单词字符串*

指向输入缓冲区中当前记号的开始位置：

**char** \*lexemebegin; *// 指向输入缓冲区中当前单词的开始位置*

向前指针，指向正在识别的字符：

**char** \*forward; *// 向前指针*

采用双缓冲区读取文件中的字符，缓冲区结尾是文件结束符EOF：

**char** buffer1[MAXLENGTH] = "\0";

**char** buffer2[MAXLENGTH] = "\0"; *// 缓冲区大小是相同的两个4096字节大小的数组*

当forward指针回退一个字符到上个缓冲区时，下次切换缓冲区时禁止重复填充缓冲区：

**bool** notreloadbuffer = false;

文件结束标志，作为文件结束判断：

**bool** endofthefile = false;

输入文件详细信息，包含文件行数、各种记号总数、标志符数量、关键字数量、数字数量、运算符数量、字符数量、字符串数量、宏定义文件数量、分隔符数量、错误数量：

**int** linenumber = 1;

**int** totalnumber = 0;

**int** identifiernumber = 0;

**int** keywordnumber = 0;

**int** numbernumber = 0;

**int** operatornumber = 0;

**int** characternumber = 0;

**int** stringnumber = 0;

**int** filenumber = 0;

**int** separatornumber = 0;

**int** errornumber = 0;

### 函数作用及其调用关系

#### main函数

函数声明：

int main();

函数作用：

读入输入文件，写出输出文件，状态机状态初始化，缓冲区初始化，forward,lexmebegin指针初始化，清理符号表内存。

函数调用关系：

调用initbuffer函数，完成缓冲区初始化；

调用lex函数，获取token;

调用putfileinfor函数，打印输入文件信息。

#### lex函数

函数声明：

**char**\* lex(FILE \*inputfile);

函数作用：

词法分析主程序，死循环，根据状态机状态和当前读入的字符决定下一状态，或者识别记号，或者返回错误信息。

函数调用关系：

被main函数循环调用，只要文件没有结束就一直调用lex函数；

调用get\_char函数，从forward指针处取到字符C；

调用get\_nbc函数，取到下一个不为空格的字符；

调用cat函数，将C连接到token后面；

调用gettuple函数，将识别到的token或者错误信息返回给main函数；

调用retract函数，将C回退一个字符；

调用reserve函数，返回关键字查询结果；

调用table\_insert函数，将token插入到符号表中；

调用letter函数，判断字符C是否是字母；

调用digit函数，判断字符C是否是数字。

#### initbuffer函数

函数声明：

**void** initbuffer(FILE \*inputfile);

函数作用：

初始化缓冲区，填充一号缓冲区。

函数调用关系：

被main函数调用，完成缓冲区初始化；

调用loadbuffer1函数，从输入文件中加载一号缓冲区。

#### get\_char函数：

函数声明：

**void** get\_char(FILE \*inputfile);

函数作用：

从forward指针获取下一个字符C。

函数调用关系：

被lex函数调用；

调用changeforward函数，完成获取字符后对forward指针的改变。

#### changeforward函数：

函数声明：

**char** \*changeforward(FILE \*inputfile);

函数作用：

完成forward指针变化以便获取下一字符C，并完成forward指向缓冲区结尾（哨兵EOF）时下一缓冲区的加载，并将forward指针指向下一缓冲区的开始，置全局布尔变量notreloadbuffer为false允许下一次填充缓冲区，完成输入文件读取结束的判断。

函数调用关系：

被get\_char函数调用，根据双缓冲区的设计改变forward指针；

调用loadbuffer2函数，当forward指针指向一号缓冲区哨兵时，加载二号缓冲区；

调用loadbuffer1函数，当forward指针指向二号缓冲区哨兵时，加载一号缓冲区。

#### loadbuffer1函数

函数声明：

**void** loadbuffer1(FILE \*inputfile);

函数作用：

填充一号缓冲区，从输入文件中不断获取字符，并在缓冲区结束位置固定填充EOF作为哨兵。

函数调用关系：

被changeforward函数调用，完成forward指针在缓冲区之间切换时缓冲区的加载。

#### loadbuffer2函数：

函数声明：

**void** loadbuffer2(FILE \*inputfile);

函数作用：

填充一号缓冲区，从输入文件中不断获取字符，并在缓冲区结束位置固定填充EOF作为哨兵。

函数调用关系：

被changeforward函数调用，完成forward指针在缓冲区之间切换时缓冲区的加载。

#### get\_nbc函数

函数声明：

**void** get\_nbc(FILE \*inputfile);

函数作用：

获取下一个不为空格的字符C。

函数调用关系：

被lex函数调用，获取到下一个不为空格的字符；

调用get\_char函数，获取字符C。

#### cat函数

函数声明：

**void** cat();*// 把C中的字符连接到token后面*

函数作用：

把C中的字符连接到token后面。

函数调用关系：

被lex函数调用。

#### gettuple函数

函数声明：

**char**\* gettuple(**char** \*tag, **char** \*attribute);

函数作用：

判断token的类型tag及其属性attribute，向main返回词法分析主程序lex的识别结果，返回结果是一个字符串，用于写入到输出文件中。并根据token的类型对输入文件详细属性进行更改。

函数调用关系：

被lex函数调用。

#### retract函数

函数声明：

**void** retract(); *// 向前指针forward后退一个字符*

函数作用：

使向前指针指向上一个字符。需要注意的是，当从一个缓冲区开始位置回退到上个缓冲区结尾时，使全局布尔变量notreloadbuffer置为true，禁止下一次填充缓冲区。

函数调用关系：

被lex函数调用。

#### reserve函数：

函数声明：

**int** reserve(**char** token**[]**); *// 根据token中的单词查关键字表，若token是关键字则返回该关键字的记号，否则返回“-1”*

函数作用：

根据token中的单词查关键字表，若token是关键字则返回该关键字的记号，否则返回“-1”。

函数调用关系：

被lex函数调用。

#### table\_insert函数：

函数声明：

**char**\* table\_insert(**char** token**[]**); *// 将识别出来的用户自定义标志符插入符号表，返回该单词在符号表中的指针位置*

函数作用：

首先根据token查询符号表，若查询到则直接返回单词在符号表中的指针位置，若未查询到则将识别出来的用户自定义标志符插入符号表，返回该单词在符号表中的指针位置

#### letter函数：

函数声明：

**bool** letter(**char** C); *// 判断C中的字符是否是字母，是则返回true，否则返回false*

函数作用：

判断C中的字符是否是字母，是则返回true，否则返回false

函数调用关系：

被lex函数调用。

#### digit函数：

函数声明：

**bool** digit(**char** C); *// 判断C中的字符是否是数字，是则返回true，否则返回false*

函数作用：

判断C中的字符是否是数字，是则返回true，否则返回false

函数调用关系：

被lex函数调用。

## 测试报告

### 测试用例1

#### 测试输入(input1.txt)：

// 测试词法分析程序的基本功能，即可以识别各种记号并正确输出，可以跳过注释。

# include <stdio.h> // 识别关键字、文件

#include "main.h"

#define MAX 100

/\* 这是一个多行注释

这是第二行

这是第三行

\*/

int main() { // 识别标识符

int a = 110; // 识别整数

float b = 13.4; // 识别小数

char c = 'i'; // 识别字符

char \*d = "helloworld!"; // 识别字符串

if (a != b) { // 识别运算符

return 0;

}

}

#### 测试输出(output1.txt)：

<keyword , #include>

<file , <stdio.h>>

<keyword , #include>

<file , "main.h">

<separator , 换行符>

<keyword , #define>

<identifier , MAX>

<number , 100>

<separator , 换行符>

<separator , 换行符>

<keyword , int>

<identifier , main>

<separator , (>

<separator , )>

<separator , {>

<separator , 制表符>

<keyword , int>

<identifier , a>

<operator , =>

<number , 110>

<separator , ;>

<separator , 制表符>

<keyword , float>

<identifier , b>

<operator , =>

<number , 13.4>

<separator , ;>

<separator , 制表符>

<keyword , char>

<identifier , c>

<operator , =>

<character , 'i'>

<separator , ;>

<separator , 制表符>

<keyword , char>

<operator , \*>

<identifier , d>

<operator , =>

<string , "helloworld!">

<separator , ;>

<separator , 换行符>

<separator , 制表符>

<keyword , if>

<separator , (>

<identifier , a>

<operator , !=>

<identifier , b>

<separator , )>

<separator , {>

<separator , 制表符>

<separator , 制表符>

<keyword , return>

<number , 0>

<separator , ;>

<separator , 换行符>

<separator , 制表符>

<separator , }>

<separator , 换行符>

<separator , }>

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

The number of the line: 18

The number of the identifier: 8

The number of the keyword: 10

The number of the number: 4

The number of the operator: 6

The number of the character: 1

The number of the string: 1

The number of the file: 2

The number of the separator: 27

The number of the error: 0

The total number: 59

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#### 结果说明：

测试用例1用于测试词法分析程序的基本功能，测试词法分析程序可以识别各种记号的类型，记号类型包含identifier标识符、keyword关键字、number数字、operator运算符、character字符、string字符串、file宏引用文件、separator分隔符。可以跳过单行和多行注释。记号输出格式为<记号 , 属性>。字符总数包含上述各类记号，也包含换行符、回车符、制表符等分隔符。可以统计各种标记的数量以及输入文件的行数。

### 测试用例2

#### 测试输入(input2.txt)：

// 检测程序错误检测及恢复能力

// 检测文件格式输入

#include 你好

#include "hello.h"

#include <stdio.h>

// 检测标志符输入

1a

a1

\_b2.

// 检测数字输入

0

1.m

2.

3e

3e-5

3.14E+10

5.20E

// 检测输入字符

char c = '';

char d = '

';

char e = 'o';

char f = '\m';

char g = '\t';

// 检测输入字符串

char \*s = "你好";

char \*s2 = "hello;

#### 测试输出(output2.txt)：

<keyword , #include>

error in line: 3, 宏定义文件名称格式有误

error in line: 3, 识别出了错误字符

error in line: 3, 识别出了错误字符

error in line: 3, 识别出了错误字符

error in line: 3, 识别出了错误字符

error in line: 3, 识别出了错误字符

error in line: 3, 识别出了错误字符

<separator , 换行符>

<keyword , #include>

<file , "hello.h">

<separator , 换行符>

<keyword , #include>

<file , <stdio.h>>

<separator , 换行符>

<separator , 换行符>

<number , 1>

<identifier , a>

<separator , 换行符>

<identifier , a1>

<separator , 换行符>

<identifier , \_b2>

error in line: 10, 识别出了错误字符

<separator , 换行符>

<separator , 换行符>

<number , 0>

<separator , 换行符>

error in line: 14, digits.需要更多digit

<identifier , m>

<separator , 换行符>

error in line: 15, digits.需要更多digit

<separator , 换行符>

error in line: 16, digits e/E || digits.digits. e/E后需要digit | + | -

<separator , 换行符>

<number , 3e-5>

<separator , 换行符>

<number , 3.14E+10>

<separator , 换行符>

error in line: 19, digits e/E || digits.digits. e/E后需要digit | + | -

<separator , 换行符>

<separator , 换行符>

<keyword , char>

<identifier , c>

<operator , =>

error in line: 22, 读取到了错误的字符

<separator , ;>

<separator , 换行符>

<keyword , char>

<identifier , d>

<operator , =>

error in line: 24, 读取到了错误的字符

error in line: 24, 读取到了错误的字符

<separator , 换行符>

<keyword , char>

<identifier , e>

<operator , =>

<character , 'o'>

<separator , ;>

<separator , 换行符>

<keyword , char>

<identifier , f>

<operator , =>

error in line: 26, 读取到了错误的转义字符

error in line: 26, 读取到了错误的字符

<separator , ;>

<separator , 换行符>

<keyword , char>

<identifier , g>

<operator , =>

<character , '\t'>

<separator , ;>

<separator , 换行符>

<separator , 换行符>

<keyword , char>

<operator , \*>

<identifier , s>

<operator , =>

<string , "你好">

<separator , ;>

<separator , 换行符>

<keyword , char>

<operator , \*>

<identifier , s2>

<operator , =>

error in line: 31, 字符串需要匹配双引号

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

The number of the line: 31

The number of the identifier: 11

The number of the keyword: 10

The number of the number: 4

The number of the operator: 9

The number of the character: 2

The number of the string: 1

The number of the file: 2

The number of the separator: 28

The number of the error: 18

The total number: 85

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#### 结果说明：

测试用例2主要用来测试词法分析程序的错误检测及错误恢复能力。输入文件中包含了多个词法错误，比如宏定义格式错误、标识符格式错误、常数格式错误、字符格式错误、字符串格式错误。错误输出格式为：error in line: 行号, 错误信息

可以看到词法分析程序可以将它们检测出来并恢复程序继续执行。

### 测试用例3

#### 测试输入(input3.txt)：

// 程序稳定性检测，将lex.c粘贴在下面

（由于过长，在此省略）

#### 测试输出(output3.txt)：

（省略记号流，只保留文件结果统计）

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

The number of the line: 961

The number of the identifier: 902

The number of the keyword: 425

The number of the number: 185

The number of the operator: 348

The number of the character: 91

The number of the string: 177

The number of the file: 2

The number of the separator: 5970

The number of the error: 0

The total number: 8100

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#### 结果说明：

测试用例3用来测试词法分析程序的稳定性，直接将本次实验源程序lex.c文件内容作为输入，可以看到，即使是相对较大的源文件，词法分析程序仍能出色完成词法分析任务。

## 源程序和可执行文件

### 源程序清单

lex.c

词法分析程序。

lex.h

词法分析程序头文件，定义全局常量、变量、函数。

token.h

定义关键字表和符号表，定义符号表大小。

### 可执行文件

lex

执行lex.c得到，通过改变main函数中的输入文件和输出文件的名字来读取和生成不同的文件，完成不同的词法分析任务。