电子科技大学信息与软件工程学院

**实 验 报 告**

学 号 2017221302006

姓 名 周玉川

（实验） 课程名称 编译技术

理论教师 邓伏虎

实验教师 邓伏虎

**电 子 科 技 大 学**

**实 验 报 告**

**学生姓名：周玉川 学号：2017221302006 指导教师：邓伏虎**

**组长姓名：周玉川 组长学号：2017221302006 组内排序：第一**

**组员姓名：无 组员学号：无 组内排序：无**

**实验地点：信软楼305 实验时间：2018.11.02**

1. **实验名称：编译技术实验一 词法分析器的自动生成**
2. **实验学时：课堂4学时 课后4学时**
3. **实验目的：**
4. 学习词法分析概念。
5. 了解单词符号的分类，以及如何从文件中读取字符串并识别。
6. 学习C语言的保留字，运算符以及届符的构成。
7. 输入一个C语言源程序，输出单词符号。
8. **实验原理：**
9. 词法分析的功能是，从左到右逐个扫描源程序的字符串，按照词法规则，识别出单词符号作为输出，对识别过程中发现的错误，输出相关的错误信息。
10. 单词符号分为五类，分别是标识符，基本字，常数，运算符，届符。
11. 基本字，运算符，届符是固定的，容易识别，由数字组成的字符串为常数，剩下的就是字符串为标识符。
12. **实验内容：**
13. 用代码写一个词法分析器，输入一个源程序，输出单词符号
14. 把源程序中的字符串进行分类，输出到屏幕以及对应文件中
15. **实验器材（设备、元器件）：**

Pc一台

1. **实验步骤：**

步骤一：写词法分析器程序

步骤二：识别源程序，把结果按照格式打印到屏幕以及对应文件中

1. **实验结果与分析（含重要数据结果分析或核心代码流程分析）**

## 目录

[目录 3](#_Toc531200289)

[8.1程序代码 4](#_Toc531200290)

[8.1.1头文件 4](#_Toc531200291)

[8.1.2源程序 7](#_Toc531200292)

[8.2代码流程分析 19](#_Toc531200293)

[8.3输出结果 21](#_Toc531200294)

[8.3.1屏幕控制台上显示结果（截取一部分） 21](#_Toc531200295)

[8.3.2 Lex.txt文件中内容 24](#_Toc531200296)

[8.3.3 Symtable.txt文件中内容 28](#_Toc531200297)

[8.3.4 Error.txt文件中内容 29](#_Toc531200298)

## 8.1程序代码

### 8.1.1头文件

|  |
| --- |
| 头文件digit.h  /\*  \* digit.h  \*  \* Created on: 2018年11月27日  \* Author: ZhouYuchuan  \*/  /\*\*\*\*\*助记符的定义\*\*\*\*\*/  //标识符  #define T\_SYMBOL 1  //常数  #define T\_CONSTANT 2  //保留字 （基本字）  #define T\_INT 3  #define T\_IF 4  #define T\_ELSE 5  #define T\_WHILE 6  #define T\_FOR 7  #define T\_RETURN 8  #define T\_FOPRN 9  #define T\_FCLOSE 10  #define T\_PRINTF 11  #define T\_SCANF 12  #define T\_FLOAT 13  #define T\_DOUBLE 14  #define T\_INCLUDE 15  #define T\_CIN 16  #define T\_COUT 17  #define T\_NEW 18  #define T\_DELETE 19  #define T\_MAIN 20  //运算符  #define T\_ADD 80 // +  #define T\_SUB 81 // -  #define T\_MUL 82 // \*  #define T\_DIV 83 // /除  #define T\_L 84 // <  #define T\_LE 85 // <=  #define T\_G 86 // >  #define T\_GE 87 // >=  #define T\_NE 88 // !=  #define T\_EQ 89 // ==  #define T\_N 90 // !  #define T\_AND 91 // &&  #define T\_OR 92 // ||  #define T\_MOD 93 // %  //届符  #define T\_WELL 60 //井号#  #define T\_ASSIGN 61 //=号  #define T\_LPAR 62 //(左括号  #define T\_RPAR 63 //)右括号  #define T\_COM 64 //,逗号  #define T\_SEM 65 //;分号  #define T\_EOLN 66 //\n下一行  #define T\_EOF 67 //文本结束结束  #define T\_LMPAR 68 //[左中括号  #define T\_RMPAR 69 //]右中括号  #define T\_LBPAR 70 //{左大括号  #define T\_RBPAR 71 //}右大括号  #define T\_COLON 72 //：冒号  #define T\_POINT 73 //.点  #define T\_SINGQU 74 //'单引号  #define T\_DOUBQU 75 //"双引号  #define MAX\_TAB\_SIZE 50 //表的最大容量  #define MAX\_BIT 128 //最大位数  #define MAX\_ID\_LENGTH 128 //常量的最长长度  // 符号表存储类型定义  typedef struct symTableItem{  char name[MAX\_ID\_LENGTH];  int type;  int ln;  } symTableItem;  // 常量表存储类型定义  typedef struct numTableItem {  int \*binary; //二进制（数组）  int bit; //二进制位数  } numTableItem;  // 保留字表  extern char reserve\_table[MAX\_TAB\_SIZE][10];  extern int reserve\_size;  // 符号表，不规则数组  extern symTableItem symbol\_table[MAX\_TAB\_SIZE];  extern int symbol\_size;  // 常量表  extern numTableItem constant\_table[MAX\_TAB\_SIZE]; //二进制存储  extern int constant\_size;  extern int line; //记录行号  // 词法分析所用到的函数声明  char getnbc(FILE \*sfp);  void concat(char character, char token[], int \*n);  int digit(char character);  int retract(char character, char \*back);  int reserve(char token[]);  int symbol(char token[]);  int constant(char token[], int n);  void error1(FILE \*fp4, char character, int line);  int analyze(FILE \*fp1, FILE \*fp2, FILE \*fp3, FILE \*fp4, int \*back\_flag, char \*back); |

### 8.1.2源程序

|  |
| --- |
| 源程序digit.c  /\*  \* digit.c  \*  \* Created on: 20181127  \* Author: ZhouYuchuan  \*/  #include <stdio.h>  #include <string.h>  #include <stdlib.h>  #include "digit.h"  numTableItem constant\_table[MAX\_TAB\_SIZE]; //常数表  int constant\_size = 0; //常数表个数  char reserve\_table[MAX\_TAB\_SIZE][10]; //保留字表  int reserve\_size = 0; //保留字个数  int line = 1; //初始化为第一行  int num = 1; // 标识符字编号  int change\_line = 1;  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* 函数名称： getnbc  \* 功能描述： 从文件中读入字符，检查变量是否为空白字符，若是，则读入下一个字符，直到读到  \* 非空白字符为止  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  char getnbc(FILE \*fp1) //fp1源文件名  {  char ch = fgetc(fp1);  while (ch == ' ' || ch == '\t') ch = fgetc(fp1); //略过空格和制表符  return ch; //返回符号  }  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* 函数名称： concat  \* 功能描述： 将character连接到token数组的末尾，然后再插入'\0'，并修改token的长度  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  void concat(char character, char token[], int \*n)  {  token[\*n] = character; //将character添加到token中  (\*n)++; //token个数加一  token[\*n] = '\0'; //字符结束符  }  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* 函数名称： symbol  \* 功能描述： 返回token类型，是否为标识符或者关键字  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  int symbol(char token[])  {  int i;  for (i = 0 ; i < reserve\_size ; i++){ //在保留字表中逐个比较，是否为保留字  if(strcmp(reserve\_table[i],token)==0)  return i+3; //返回保留字类型，保留字类型与数组底数有加三关系  }  return T\_SYMBOL; //若不是保留字，返回标识符类型  }  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* 函数名称： digit  \* 功能描述： 判断是否是数字，若是则返回1，否则返回0  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  int digit(char character)  {  return (character >= '0' && character <= '9');  }  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* 函数名称： retract  \* 功能描述： 回退字符，若是非空白字符，将字符保存在back中；若是空白字符，不回退  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  int retract(char character, char \*back)  {  if (character != ' ') {  \*back = character;  return 1;  }  return 0;  }  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* 函数名称： constant  \* 功能描述： 对token中的字符串查常数表，若查到，返回该常数的序号，否则将新的常数插入  表中，再返回序号  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  int constant(char token[], int n)  {  int i, j, b;  int dec, rem, tmp;  int stack[MAX\_BIT];//栈  int top = 0;  // 转换成十进制数  for (i = 0, dec = 0; i < n; i++) {  dec = dec \* 10 + (token[i] - '0');  }  // 转换成二进制数，倒序放在栈中，高位放在栈底  rem = dec;  if (dec == 0) stack[top++] = 0; //“0”单独处理  while (rem > 0) {  stack[top++] = rem % 2;  rem = rem / 2;  }  tmp = top; //暂存栈顶指针，代表二进制数的位数  for (i = 0; i < constant\_size; i++) {  b = constant\_table[i].bit; //取二进制数的位数  if (top != b) continue; //位数不相等，直接跳过该数  for (j = 0; j < b; j++) { //逐位比较  if (stack[--top] != constant\_table[i].binary[j]) break;  }  if (j >= b) return i; //找到相同的数，直接返回该数在表中的位置  top = tmp; //还原top  }  // 如果没有找到相同的数，就将该数录入表中  i = constant\_size++;  constant\_table[i].binary = (int \*)malloc((top + 1)\* sizeof(int));//动态分配内存  constant\_table[i].bit = top; //bit存储该二进制数的位数  for (j = 0; top > 0; j++) {  constant\_table[i].binary[j] = stack[--top]; //顺序存储进数组中  }  return i;  }  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* 函数名称： error1  \* 功能描述： 打印出错的信息  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  void error1(FILE \*fp4, char character, int line)  {  printf("\nerror: line %.2d: 含有非法字符:\t%c\n\n", line, character); //打印屏幕上  fprintf(fp4, "error: line %.2d: 含有非法字符:\t%c \n", line, character);//打印error文件中  }  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* 函数名称： analyze  \* 功能描述： 主词法分析函数，每调用一次获取一个单词，将其打印到屏幕并输出到文本  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  int analyze(FILE \*fp1, FILE \*fp2, FILE \*fp3, FILE \*fp4, int \*back\_flag, char \*back)  { //fp1为输入源文件，fp2为输出文件，fp3为符号表文件，fp4为错误输出文件  char character;  char token[24]={'\0'};  int n = 0; //n记录token中的字符串长度  int type = 0; //单词类型  int val = 0; //单词属性，只有标识符和常数不是0  if (\*back\_flag) { //上一次有回退，就使用上一次的字符，否则读取文件中下一个字符  character = \*back;  \*back\_flag = 0;  }  else character = getnbc(fp1);      if (digit(character)) { //第一个数是常数  while (digit(character)) {//下一个数字也是常数  concat(character, token, &n);//添加到token数组中  character = fgetc(fp1);  }  val = constant(token, n);//在常数表中位置  type = T\_CONSTANT;//类型是常数  \*back\_flag = retract(character, back);//是否有回退  }  else if(isalpha(character)||character == '\_')//是字母下，划线组成 ,可能是保留字，标识符  {  while(isalpha(character)||digit(character)||character == '\_')//是字母，下划线，数字组成 ,可能是保留字，标识符  {  concat(character, token, &n); //添加到token数组中  character = fgetc(fp1);  }  type = symbol(token); //返回字符的类型  \*back\_flag = retract(character, back); //判断回退  }  else {  switch (character) {  case '+': type = T\_ADD; concat(character, token, &n); break;//为加号  case '-': type = T\_SUB; concat(character, token, &n); break;//为减号  case '\*': type = T\_MUL; concat(character, token, &n); break;//为乘号  case '%': type = T\_MOD; concat(character, token, &n); break;//为求模号  case '/': //进行判断，为除号，还是注释开始符号  character = fgetc(fp1); //下一个符号  if ( character == '/' ){ //为注释符号 "//"  while( (character = fgetc(fp1)) != '\n');//该行符号忽略  line++; //行数加一  return 0; //结束  }  else if (character == '\*'){//为注释符号 "/\*"  char ch; //协助判断注释结束符 "\*/"  character = fgetc(fp1); //读下一个字符  if ( character == '\n')  line++; //行数加一  while (1){  ch = fgetc(fp2); //读源文件  if ( ch == '\n')  line++; //行数加一  if ( ( character ) == '\*' && ( ch == '/' )) //遇到结束符  break;  else{  character = ch ;  }  }  return 0;  }  else  {  \*back\_flag = retract(character, back);  type = T\_DIV; concat('/', token, &n);//为除号  }  break;    case '<': type = T\_L; concat(character, token, &n);//先让它为小于符号  character = fgetc(fp1);  if (character == '='){  type = T\_LE; concat(character, token, &n);}  else{  \*back\_flag = retract(character, back); //否则回退一个字符  }  break;  case '>': type = T\_G; concat(character, token, &n); // 先让它为大于符号  character = fgetc(fp1);  if (character == '='){  type = T\_GE; concat(character, token, &n);}//若后面为等号 即为大于等于号  else{  \*back\_flag = retract(character, back); //否则回退一个字符  }  break;  case '=': type = T\_ASSIGN;concat(character, token, &n);//先记为赋值符号  character = fgetc(fp1);  if (character == '='){ //若下一个字符为等号，记为等号  type = T\_EQ; concat(character, token, &n);}  else{  \*back\_flag = retract(character, back); //否则回退一个字符  }  break;  case '!': type = T\_N;concat(character, token, &n); //先记为非  character = fgetc(fp1);  if (character == '='){ //下一个字符为等号，则记为不等于  type = T\_NE; concat(character, token, &n);}  else{  \*back\_flag = retract(character, back); //否则会退一个字符  }  break;  case '&': concat(character, token, &n);  character = fgetc(fp1);  if (character == '&'){ //两位都是&，记为逻辑与  type = T\_AND;concat(character, token, &n);  }  else { //否则记为错误符号  error1(fp4,'&',line);  \*back\_flag = retract(character, back);//回退一个字符  }  break;  case '|': concat(character, token, &n);  character = fgetc(fp1);  if (character == '|'){ //连个|符号记为逻辑或  type = T\_AND;concat(character, token, &n);  }  else {  error1(fp4,\*token,line); //否则记为非法字符  \*back\_flag = retract(character, back); //回退一个字符  }  break;  case '\\': concat(character, token, &n);  character = fgetc(fp1);  if ( character == 'n'){ //若两个符号为'/n'  type = T\_EOLN;concat(character, token, &n);  change\_line = 0;  }  else {  error1(fp4,\*token,line); //否则记为非法字符  \*back\_flag = retract(character, back); //回退一个字符  }  break;  case '(': type = T\_LPAR; concat(character, token, &n); break;  case ')': type = T\_RPAR; concat(character, token, &n); break;  case '[': type = T\_LMPAR; concat(character, token, &n); break;  case ']': type = T\_RMPAR; concat(character, token, &n); break;  case '{': type = T\_LBPAR; concat(character, token, &n); break;  case '}': type = T\_RBPAR; concat(character, token, &n); break;  case ',': type = T\_COM; concat(character, token, &n); break;  case ':': type = T\_COLON; concat(character, token, &n); break;  case ';': type = T\_SEM; concat(character, token, &n); break;  case '.': type = T\_POINT; concat(character, token, &n); break;  case'\'': type = T\_SINGQU;concat(character, token, &n); break;  case'\"': type = T\_DOUBQU;concat(character, token, &n); break;  case EOF: type = T\_EOF; concat('#', token, &n); break;  case'\n': type = T\_EOLN; change\_line = 1 ;  concat(character, token, &n); break;  case '#': type = T\_WELL; concat(character, token, &n); break;  default: error1(fp4, character, line);  }  }  // 在屏幕上打印并输出到文本  if ( type >= 3 && type < reserve\_size+3){  printf("line %.2d: 保留字 %s\n", line, token);  fprintf(fp2, "line %.2d: 保留字 %s\n", line, token);  }  else if ( type >= T\_ADD && type <= T\_MOD){  printf("line %.2d: 运算符 %s\n", line, token);  fprintf(fp2, "line %.2d: 运算符 %s\n", line, token);  }  else {  switch ( type ){  case T\_SYMBOL:  printf("line %.2d: 标识符 %s\n", line, token);  fprintf(fp2,"line %.2d 标识符 %s\n", line,token);  fprintf(fp3, "%d\t%d\t%s\n", num++,line, token);  break;  case T\_CONSTANT:  printf("line %.2d: CONSTANT %s\n", line, token);  fprintf(fp2, "line %.2d: CONSTANT %s\n", line,token);  break;  case T\_LPAR:  printf("line %.2d: 左括号 %s\n", line, token);  fprintf(fp2, "line %.2d: 左括号 %s\n", line, token);  break;  case T\_RPAR:  printf("line %.2d: 右括号 %s\n", line, token);  fprintf(fp2, "line %.2d: 右括号 %s\n", line, token);  break;  case T\_LMPAR:  printf("line %.2d: 左中括号 %s\n", line, token);  fprintf(fp2, "line %.2d: 左中括号 %s\n", line, token);  break;  case T\_RMPAR:  printf("line %.2d: 右中括号 %s\n", line, token);  fprintf(fp2, "line %.2d: 右中括号 %s\n", line, token);  break;  case T\_LBPAR:  printf("line %.2d: 左大括号 %s\n", line, token);  fprintf(fp2, "line %.2d: 左大括号 %s\n", line, token);  break;  case T\_RBPAR:  printf("line %.2d: 右大括号 %s\n", line, token);  fprintf(fp2, "line %.2d: 右大括号 %s\n", line, token);  break;  case T\_ASSIGN:  printf("line %.2d: 等号 %s\n", line, token);  fprintf(fp2, "line %.2d: 等号 %s\n", line, token);  break;  case T\_COM:  printf("line %.2d: 逗号 %s\n", line, token);  fprintf(fp2, "line %.2d: 逗号 %s\n", line, token);  break;  case T\_COLON:  printf("line %.2d: 冒号 %s\n", line, token);  fprintf(fp2, "line %.2d: 冒号 %s\n", line, token);  break;  case T\_SEM:  printf("line %.2d: 分号 %s\n", line, token);  fprintf(fp2, "line %.2d: 分号 %s\n", line, token);  break;  case T\_POINT:  printf("line %.2d: 点 %s\n", line, token);  fprintf(fp2, "line %.2d: 点 %s\n", line, token);  break;  case T\_SINGQU:  printf("line %.2d: 单引号 %s\n", line, token);  fprintf(fp2, "line %.2d: 单引号 %s\n", line, token);  break;  case T\_DOUBQU:  printf("line %.2d: 双引号 %s\n", line, token);  fprintf(fp2, "line %.2d: 双引号 %s\n", line, token);  break;  case T\_EOF:  printf("line %.2d: 结尾符 %s\n", line, token);  fprintf(fp2, "line %.2d: 结尾符 %s\n", line, token);  break;  case T\_WELL:  printf("line %.2d: 井号 %s\n", line, token);  fprintf(fp2, "line %.2d: 井号 %s\n", line, token);  break;  case T\_EOLN:  printf("line %.2d: 行尾符 %s\n", line, token);  fprintf(fp2, "line %.2d: 行尾符 %s\n", line, token);  if (change\_line)//若遇到换行符就换行，行数加一，若'\n'就不换行  line++;  break;  default: break;  }  }  return type; //返回类型  }  int main(void)  {  char \*soure = "leapyear.txt";  char \*output = "lex.txt";  char \*error = "error.txt";  char \*symtable = "symtable.txt";  char \*reserve = "reserve.txt";    FILE \*fp\_src = fopen(soure, "r");  if (fp\_src == NULL) {  printf("打开源文件失败！请确定源文件的位置！\n");  return -1;  }  //从保留字文件中读取保留字  FILE \*fp\_res = fopen(reserve, "r");  if (fp\_res == NULL) {  printf("打开保留字文件失败！请确定源文件的位置！\n");  return -1;  }  else{  char ch[20];  fscanf (fp\_res, "%s", ch);  while ( strchr(ch, '#') == NULL)  fscanf (fp\_res, "%s", ch);    while (fscanf (fp\_res, "%s", ch) != EOF)  {  strcpy(reserve\_table[reserve\_size], ch);  reserve\_size++;  }  }  FILE \*fp\_output = fopen(output, "w");  FILE \*fp\_err = fopen(error, "w");  FILE \*fp\_sym = fopen(symtable,"w");  int back\_flag = 0; //回退的标记，1代表上一次有回退(空格不算)，0代表没有  char back; //存放上一次的回退值  printf("周玉川-2017221302006\n\n");  fprintf(fp\_output, "周玉川-2017221302006\n\n");  fprintf(fp\_sym, "符号表\n编号\t行号\t名字\n");    while (analyze(fp\_src, fp\_output, fp\_sym, fp\_err, &back\_flag, &back) != T\_EOF);//返回类型为文件结束，结束循环  fclose(fp\_src);  fclose(fp\_output);  fclose(fp\_sym);  fclose(fp\_err);  fclose(fp\_res);  return 0;  } |

## 8.2代码流程分析

1. 从文件中逐个读入字符串，若上一次有字符回退，则不从文件中读入，利用回退的字符
2. 若是字符是数字，继续读入字符直到不是数字，它们组成的字符串为数字
3. 若是字母或下划线，继续读入字符直到读入的不是数字或者字母或者下划线，它们组成的字符串存在token[]中，遍历保留字数组，判断token中数组是否为保留字，否则为标识符
4. 否则若是单个运算符以及届符，由于数量多利用switch进行判断较为方便
5. 若判断多个字符的运算符，或者多个字符届符，可采用以下方法
   1. 拿不等于!=举例，遇到非！字符后，先让字符类型记为非，然后读取下一个字符，若下个字符为=，把=添加到token中，字符串类型改为不等于，否则回退一个字符

关键代码如下

|  |
| --- |
| case '!': type = T\_N;concat(character, token, &n);  //先记为非  character = fgetc(fp1);  if (character == '='){  //下一个字符为等号，则记为不等于  type = T\_NE; concat(character, token, &n);}  else{  \*back\_flag = retract(character, back);  //否则会退一个字符  }  break; |

* 1. 届符以注释开始/\*和//举例，若遇到除号/，先把字符串类型记为除，读取下一个字符，若下一个字符为\*号，则注释开始，开始从源文件中读字符串并忽略，直到遇到\*/结束，然后结束此次操作，中间注意若遇到换行符让行数加一。若下一个字符还为/，则为单行注释，开始从源文件中读字符串并忽略，直到遇到换行符，记得换行时让行数加一，结束此次操作。若是其他字符，不做其他操作，回退一个字符，进行下一步，核心代码如下

|  |
| --- |
| case '/': //进行判断，为除号，还是注释开始符号  character = fgetc(fp1); //下一个符号  if ( character == '/' ){ //为注释符号 "//"  while( (character = fgetc(fp1)) != '\n');//该行符号忽略  line++; //行数加一  return 0; //结束  }  else if (character == '\*'){//为注释符号 "/\*"  char ch; //协助判断注释结束符 "\*/"  character = fgetc(fp1); //读下一个字符  if ( character == '\n')  line++; //行数加一  while (1){  ch = fgetc(fp2); //读源文件  if ( ch == '\n')  line++; //行数加一  if ( ( character ) == '\*' && ( ch == '/' )) //遇到结束符  break;  else{  character = ch ;  }  }  return 0;  }  else  {  \*back\_flag = retract(character, back);  type = T\_DIV; concat('/', token, &n);//为除号  }  break; |

6）重复上述操作直到到达文件结尾

## 8.3输出结果

### 8.3.1屏幕控制台上显示结果（截取一部分）

|  |
| --- |
| 开头处    中间部分    结尾处 |

### 8.3.2 Lex.txt文件中内容

|  |
| --- |
| 周玉川-2017221302006  line 01: 井号 #  line 01: 保留字 include  line 01: 双引号 "  line 01 标识符 stdio  line 01: 点 .  line 01 标识符 h  line 01: 双引号 "  line 01: 行尾符  line 02: 保留字 int  line 02 标识符 fun  line 02: 左括号 (  line 02: 保留字 int  line 02 标识符 year  line 02: 右括号 )  line 02: 左大括号 {  line 02: 行尾符  line 03: 保留字 if  line 03: 左括号 (  line 03: 左括号 (  line 03 标识符 year  line 03: 运算符 %  line 03: CONSTANT 4  line 03: 运算符 ==  line 03: CONSTANT 0  line 03: 运算符 &&  line 03 标识符 year  line 03: 运算符 %  line 03: CONSTANT 100  line 03: 运算符 !=  line 03: CONSTANT 0  line 03: 右括号 )  line 03: 运算符 ||  line 03 标识符 year  line 03: 运算符 %  line 03: CONSTANT 400  line 03: 运算符 ==  line 03: CONSTANT 0  line 03: 右括号 )  line 03: 保留字 return  line 03: CONSTANT 1  line 03: 分号 ;  line 03: 行尾符  line 04: 保留字 return  line 04: CONSTANT 0  line 04: 分号 ;  line 04: 行尾符  line 05: 右大括号 }  line 05: 行尾符  line 06: 行尾符  line 07: 保留字 int  line 07: 保留字 main  line 07: 左括号 (  line 07: 右括号 )  line 07: 行尾符  line 08: 左大括号 {  line 08: 行尾符  line 09: 保留字 int  line 09 标识符 i  line 09: 逗号 ,  line 09 标识符 sum  line 09: 等号 =  line 09: CONSTANT 0  line 09: 分号 ;  line 09: 行尾符  line 10: 保留字 for  line 10: 左括号 (  line 10 标识符 i  line 10: 等号 =  line 10: CONSTANT 2001  line 10: 分号 ;  line 10 标识符 i  line 10: 运算符 <=  line 10: CONSTANT 2999  line 10: 分号 ;  line 10 标识符 i  line 10: 运算符 +  line 10: 运算符 +  line 10: 右括号 )  line 10: 行尾符  line 11: 保留字 if  line 11: 左括号 (  line 11 标识符 fun  line 11: 左括号 (  line 11 标识符 i  line 11: 右括号 )  line 11: 右括号 )  line 11: 左大括号 {  line 11: 行尾符  line 12: 保留字 printf  line 12: 左括号 (  line 12: 双引号 "  line 12: 运算符 %  line 12 标识符 d  line 12 标识符 leap  line 12: 等号 =  line 12: CONSTANT 1  line 12: 行尾符 \n  line 12: 双引号 "  line 12: 逗号 ,  line 12 标识符 i  line 12: 右括号 )  line 12: 分号 ;  line 12: 行尾符  line 13 标识符 sum  line 13: 运算符 +  line 13: 运算符 +  line 13: 分号 ;  line 13: 行尾符  line 14: 右大括号 }  line 14: 行尾符  line 15: 保留字 else  line 15: 保留字 printf  line 15: 左括号 (  line 15: 双引号 "  line 15: 运算符 %  line 15 标识符 d  line 15 标识符 leap  line 15: 等号 =  line 15: CONSTANT 0  line 15: 行尾符 \n  line 15: 双引号 "  line 15: 逗号 ,  line 15 标识符 i  line 15: 右括号 )  line 15: 分号 ;  line 15: 行尾符  line 16: 保留字 printf  line 16: 左括号 (  line 16: 双引号 "  line 16 标识符 There  line 16 标识符 are  line 16: 运算符 %  line 16 标识符 d  line 16 标识符 leapyears  line 16: 运算符 !  line 16: 运算符 !  line 16: 行尾符 \n  line 16: 双引号 "  line 16: 逗号 ,  line 16 标识符 sum  line 16: 右括号 )  line 16: 分号 ;  line 16: 行尾符  line 17: 右大括号 }  line 17: 结尾符 # |

### 8.3.3 Symtable.txt文件中内容

|  |
| --- |
| 符号表  编号 行号 名字  1 1 stdio  2 1 h  3 2 fun  4 2 year  5 3 year  6 3 year  7 3 year  8 9 i  9 9 sum  10 10 i  11 10 i  12 10 i  13 11 fun  14 11 i  15 12 d  16 12 leap  17 12 i  18 13 sum  19 15 d  20 15 leap  21 15 i  22 16 There  23 16 are  24 16 d  25 16 leapyears  26 16 sum |

### 8.3.4 Error.txt文件中内容

|  |
| --- |
|  |

1. **总结及心得体会：**
2. 完成了老师的基本要求后，扩展了程序，可以识别更多的运算符，关键字，届符。
3. 解决了多个字符无法识别的问题，升级版可以识别标识符，多个运算符组成的字符串，以及注释。
4. 解决了单个字符可能属于多个单词类型的问题，比如符号除/可能是除号也可能是注释符号//或/\*的一部分。
5. C语言的功底很重要。
6. 学习书本后，再写代码会容易一些。
7. 带着兴趣会事半功倍。
8. **对本实验过程及方法、手段的改进建议：**
9. 希望实验能多一些，能更有趣一点，最好有选做题让基础好一点的同学挑战自我。
10. 希望老师和同学之间能多交流，答疑课多一点点。

**报告评分：**

**指导教师签字：**