**实 验 报 告**

**课程名称** 编译原理

**实验项目** 词法分析器的实现

**实验仪器**  PC机

**系 别**  计算机

**专 业**  计算机科学与技术

**班级/学号**  计科1606班/2016011440

**学生姓名**  刘逸芸

**实验日期**  2019/5/6

**成 绩**

**指导教师**  范艳芳

**实验一 词法分析器的实现**

**一、实验目的**

设计并实现一个PL/0语言(或其它语言的子集,如C语言的子集)的词法分析程序，加深对词法分析原理的理解。

**二、实验原理：**

词法分析器是执行词法分析的程序，其功能为输入源程序，输出单词符号。单词符号是一个程序语言的基本语法符号。程序语言的单词符号一般可分为下列物种。

（1）**关键字**：是由程序语言定义的具有固定意义的标识符。有时称这些标识符为保留字或基本字。如const, main , if , else , while , int , for, return等；

（2）**标识符**：用来表示各种名字，如变量名，数组名，过程名等；

（3）**常数**：常数的类型有很多，本实验只判断无符号整数；

（4）**运算符**：如+、-、\*、/等

（5）**界符**：又称分隔符，如（、）、；、，等

词法分析器所输出的单词符号，一般为Token字。Token字是如下形式的二元式：

（单词种别，单词符号的属性值）

单词种别通常用整数编码。本实验的编码为：

（1）**关键字**：单词种别码为1，输出样式示例：（1，if）；

（2）**标识符**：单词种别码为2，输出样式示例：（2，x）；

（3）**常数**：单词种别码为3，输出样式示例：（3，10）；

（4）**运算符**：单词种别码为4，输出样式示例：（4，+）；

（5）**分隔符**：单词种别码为5，输出样式示例：（5， ；）；

**三、程序功能与框架**

程序的主要功能就是对一个源程序进行扫描，过滤无关字符后分析出所有的关键字、标识符、无符号整数、运算符、分隔符，然后输出单词符号。

程序的主要框架是：首先读源程序文件，然后从头到尾扫描程序，分析出单词符号，然后关闭文件流，输出结果。

**四、设计说明**

1. 数据结构

本程序定义了两个结构体指针NormalNode和IdentifierNode，分别用来表示token字和标识符。NormalNode有三个结构体成员变量：token字的内容（即属性值）、token字种别码、下一个token字的首地址。IdentifierNode有三个结构体成员变量：标识符的内容、标识符的种别码（即为普通标识符还是保留字）、下一个标识符的首地址。具体代码如下所示：

|  |
| --- |
| //单词指针结构体  typedef struct NormalNode  {  char content[50];//内容  int type;//种别码  struct NormalNode \*next;//下一个token字指针结构体  }NormalNode;  //标识符指针结构体  typedef struct IdentifierNode  {  char content[50];//内容  int type;//种别码  struct IdentifierNode \*next;//下一个标识符指针结构体  }IdentifierNode; |

2. 主要变量

char \*keyword[]：二维字符数组，用于存放保留字单词表

NormalNode \*normalHead：token 字指针，存放所有已扫描的token字

IdentifierNode \*identifierHead：标识符指针，存放所有已扫描的标识符（包括保留字）

FILE \*fp：需要做词法分析的源程序文件

char ch：正在扫描分析的字符

3. 算法思路

首先，对所有变量进行初始化操作，然后读源程序文件，如果读文件失败，则打印提示文字“文件不存在”，然后退出程序。读入源程序文件后，对文件内容进行从头到尾的扫描，开始识别单词符号。

如果扫描为“/”字符，有可能有三种情况：

（1）“//…//”类型的注释

（2）“/\*…\*/”类型的注释

（3）除法运算符

对以上三种情况，只需要判断第一个“/”字符的下一个字符类型，如果也为“/”，那么就是情况（1），直接跳转到下一行；如果为“\*”，那么就是情况（2），直接跳转到注释结束；否则，视为情况（3），该单词的种别码为4。

如果扫描到字母，那么继续扫描下一个字符，循环这个操作直到下一个字符不是字母或者数字，并且将所有扫描到的<字母>{<字母>|<数字> }\*集合存放在一个字符数组中。然后调用marchID方法，将该数组与保留字单词表中的一一匹配，判断该单词是否为保留字，如果为保留字，那么该单词的单词种别码为1，否则为2。

如果扫描到数字，那么继续扫描下一个字符，循环这个操作指导下一个字符不是数字，并将所有扫描到的<数字>{<数字>}\*集合存放在一个字符数组中。该数字集合就是单词的内容，该单词种别码为3。

如果扫描到+、-、\*、=、>、<等符号，则该单词种别码为4。

如果扫描到；、，、”、:、(、) 、}、{等符号，则该单词种别码为5。

扫描完所有字符后，关闭文件流，然后调用displaynormalNode方法，将所有的token字打印出来。

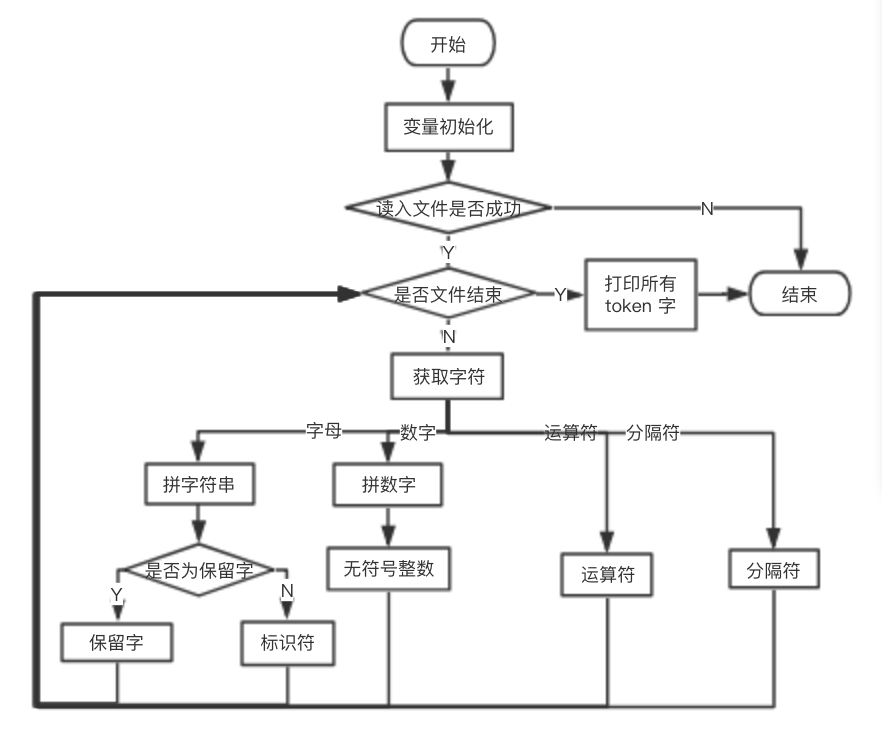


图1 算法流程图

**五、用户操作指南**

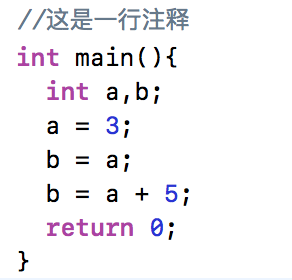
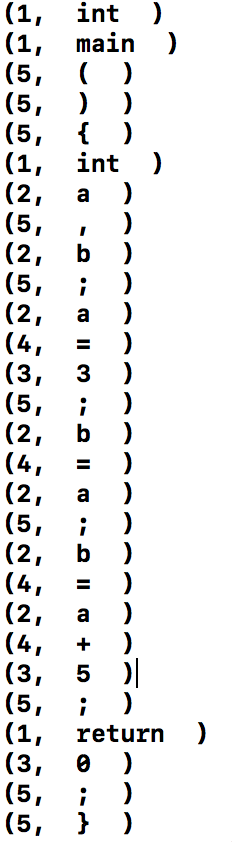
修改词法分析器中的源程序文件地址为需要做词法分析的源程序文件的绝对地址，然后运行词法分析器，就可以在控制台看到词法分析的结果了，每一行为一个token字。

**六、调试报告**

通过本次试验，让我了解到如何设计、编制并调试词法分析程序，加深对词法分析原理的理解；熟悉了构造词法分析程序的手工方式的相关原理。在本次试验中，对于每一类单词，都是一种一码，没有具体区分每一个关键字或运算符或分隔符，而是按大类区分的。而且本实验值讨论了无符号整数，没有考虑浮点数。因此我设想改进程序，除了无符号整数外，还应该判断出双精度浮点数和单精度浮点数，而且对于关键字、运算符、分隔符都应该为一词一码。

**七、测试数据及运行结果**

测试数据： 运行结果：

附录1:（词法分析器源代码）

|  |
| --- |
| //  // main.c  // myLexer  //  // Created by lyy on 2019/5/5.  // Copyright © 2019年 lyy. All rights reserved.  //  #include <stdio.h>  #include <stdlib.h>  #include <string.h>  //存储保留字  char \*keyword[] = {"auto","double","int","struct","break","else","long","switch","case","enum","register",  "main","typedef","char","extern","return","union","const","float","short","unsigned",  "continue","for","signed","void","default","goto","sizeof","volatile","do","while",  "static","if"};  #define KEY 1  #define KEY\_DESC "保留字"  #define IDENTIFIER 2  #define IDENTIFIER\_DESC "标识符"  #define INTEGER 3  #define INTEGER\_DESC "无符号整数"  #define OPERATOR 4  #define OPERATOR\_DESC "运算符"  #define BOUNTWROD 5  #define BOUNTWROD\_DESC "分隔符"  #define HEADER 6  #define HEADER\_DESC "头文件"  #define MAX\_CHAR\_NUMBER 100000  char charSET[MAX\_CHAR\_NUMBER];  //单词指针结构体  typedef struct NormalNode  {  char content[50];  int type;  struct NormalNode \*next;  }NormalNode;  //标识符指针结构体  typedef struct IdentifierNode  {  char content[50];  int type;  struct IdentifierNode \*next;  }IdentifierNode;  NormalNode \*normalHead;//单词指针头  IdentifierNode \*identifierHead;//标识符指针头  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  函数名：  函数返回值：  入口参数：  函数功能：初始化单词指针和标识符指针，内容为空，类型码-1，next空  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  void initNode(){  normalHead = (NormalNode \*)malloc(sizeof(NormalNode));  strcpy(normalHead->content, "");  normalHead->type = -1;  normalHead->next = NULL;    identifierHead = (IdentifierNode \*)malloc(sizeof(IdentifierNode));  strcpy(identifierHead->content, "");  identifierHead->type = -1;  identifierHead->next = NULL;  }  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  函数名：  函数返回值：  入口参数：  函数功能：创建一个新的单词（词法分析的单位）  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  void createNorNode(char \*content,int type){  NormalNode \*p = normalHead;  NormalNode \*tmp = (NormalNode \*)malloc(sizeof(NormalNode));    while ( p->next != NULL) {  p = p->next;  }  strcpy(tmp->content,content);//复制单词内容  tmp->type = type;//单词类型码  tmp->next = NULL;  p->next = tmp;  }  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  函数名：  函数返回值：  入口参数：  函数功能：创建一个新的标识符（如果标识符已存在，那么直接返回）  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  void createIdeNode(char \*content,int type){  IdentifierNode \*p = identifierHead;  IdentifierNode \*tmp = (IdentifierNode \*)malloc(sizeof(IdentifierNode));    int address\_count = 1;    while (p->next != NULL){  p = p -> next;  address\_count++;  if(!strcmp(p->content, content))//标识符已经存在  return;  }  strcpy(tmp->content,content);  tmp->type = type;  tmp->next = NULL;  p->next= tmp;  }  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  函数名：  函数返回值：  入口参数：  函数功能：判断这个单词是一个标识符还是一个保留字  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  int marchIde(char \*word){  for (int i = 0; i < 32; i++) {  if(strcmp(word, keyword[i]) == 0){  return KEY;  }  }  return IDENTIFIER;  }  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  函数名：  函数返回值：  入口参数：  函数功能：打印单词结果，格式为：(content,type)  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  void displaynormalNode(){  NormalNode \*p = normalHead;  while ( (p = p -> next)) {  printf("(%d, ",p->type);  printf("%s )",p->content);  printf("\n");  }  }  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  函数名：  函数返回值：  入口参数：  函数功能：程序入口主函数  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  int main(int argc, const char \* argv[]) {  initNode();  FILE \*fp;  char ch;  int count = 0;  int line = 1;    char tmp[30];  char \*word;    if((fp = fopen("/Users/lyy/Desktop/junior2/Compiling/ComExperiment/Lexer/myLexer/myLexer/options.c", "r")) == NULL){  printf("文件不存在\n");  return 0;  }  while( ch!= EOF){  ch = fgetc(fp);    //注释  if (ch == '/') {  ch = fgetc(fp);  // //...// 类型注释  if(ch == '/') {  while (ch != ' ' && ch != '\n') {  ch = fgetc(fp);  }  }  // /\*...\*/ 类型注释  else if(ch == '\*'){  line++;  ch = fgetc(fp);  while (ch != '\*'){  ch = fgetc(fp);  if(ch == '\n'){  line++;  }  }while (ch != '/'){  ch = fgetc(fp);  }  }  // 运算符"/"  else {  createNorNode("/", OPERATOR);  }  }    //保留字和标识符  if((ch >= 'a' && ch <= 'z') || (ch >= 'A' && ch <= 'Z') || ch == '\_'){  count = 0;  while ((ch >= 'a' && ch <= 'z') || (ch >= 'A' && ch <= 'Z') || ch == '\_' || (ch >= '0' && ch<= '9')) {  tmp[count++] = ch;  ch = fgetc(fp);  }  word = (char \*)malloc(sizeof(char)\*count);  memcpy(word, tmp, count);  word[count] = '\0';  int id = marchIde(word);  if (id == IDENTIFIER ) {  createIdeNode(word,IDENTIFIER);  createNorNode(word,IDENTIFIER);  }else{  createNorNode(word,id);  }  fseek(fp,-1L,SEEK\_CUR);//向后回退一位  }    //无符号整数  else if(ch >= '0' && ch <= '9'){  count = 0;  while (ch >= '0' && ch <= '9') {  tmp[count++] = ch;  ch = fgetc(fp);  }  word = (char \*)malloc(sizeof(char)\*(count+1));  memcpy(word, tmp, count);  word[count] = '\0';  createNorNode(word,INTEGER);  fseek(fp,-1L,SEEK\_CUR);//向后回退一位  }  //换行  else if(ch == ' ' || ch == '\t' || ch == '\r' || ch =='\n' ){  if(ch == '\n'){  line++;  }  }    //运算符  else if(ch == '+'){  createNorNode("+",OPERATOR);  }else if(ch == '-'){  createNorNode("-",OPERATOR);  }else if(ch == '\*'){  createNorNode("\*",OPERATOR);  }else if(ch == '='){  createNorNode("=",OPERATOR);  }else if(ch == '>'){  createNorNode(">",OPERATOR);  }else if(ch == '<'){  createNorNode("<",OPERATOR);  }    //分隔符  else if(ch == ';'){  createNorNode(";",BOUNTWROD);  }else if(ch == ','){  createNorNode(",",BOUNTWROD);  }else if(ch == '"'){  createNorNode("\"",BOUNTWROD);  }else if(ch == ':'){  createNorNode(":",BOUNTWROD);  }else if(ch == '('){  createNorNode("(",BOUNTWROD);  }else if(ch == ')'){  createNorNode(")",BOUNTWROD);  }else if(ch == '{'){  createNorNode("{",BOUNTWROD);  }else if(ch == '}'){  createNorNode("}",BOUNTWROD);  }    //去掉空格和换行符  if (ch != ' '&& ch != '\n') {  charSET[count] = ch;  count++;  }  }  fclose(fp);    displaynormalNode();    return 0;  } |