### 1、PL/0语言的单词结构

#### 关键字和标识符的识别：

这块的思路是构造一个关键字表，将读入的有可能是关键字的缓冲与该关键词表中的单词逐一比较，如果相同，则缓冲区内存放的单词为关键字，否则为标识符。

核心代码如下

if (isLetter(ch)) {  
 // 如果读取到的字符是字母  
 // 有可能为IDENTIFIER  
 // 有可能为KEYWORD  
 // 思路是整个缓冲区，往缓冲区里写单词  
 // 再对比和关键词表中的关键词是否一致  
 // 如果不一致，铁标识符  
 // 标识符只能是字母，最大长度为10  
 char buffer[100]; // 假设标识符或关键字的最大长度为 100  
 int i = 0;  
 buffer[i++] = ch;  
  
 // BUG  
 // while ((ch = fgetc(fp)) != EOF && isLetter(ch = fgetc(fp))) {  
 // 这句BUG因为题意理解有误，标识符第一个字母后可以出现数字  
 while ((ch = fgetc(fp)) != EOF && (isLetter(ch) || isDigit(ch))) {  
 buffer[i++] = ch;  
 }  
 ungetc(ch,fp); // 将读取到的非字母字符放回去  
 buffer[i] = '\0';  
 //puts(buffer);  
 //token.category = -1;  
 //return token;  
 // 检查是否是关键字  
 // 假设有一个关键字列表  
 char \*keywords[] = {"begin", "end", "if", "then", "while", "do", "const", "var", "call" , "procedure", "odd"}; // 关键字表  
 int numKeywords = sizeof(keywords) / sizeof(keywords[0]);  
 for (int j = 0; j < numKeywords; j++) {  
 if (strcmp(buffer, keywords[j]) == 0) {  
 token.category = stringToCategory(categoryStrings[16+j]);  
 return token;  
 }  
 }  
  
 // 不是关键字则为标识符  
 // 不过应该先查找原标识符表中是否存在该word  
 // 不存在再id++  
 // 这块出了BUG  
 token.category = stringToCategory("IDENTIFIER");  
 bool isHadIden = false;  
 int numIdentifier = sizeof(identifierStrings) / sizeof(identifierStrings[0]);  
 int j = 0;  
 for (; j < numIdentifier; j++) {  
 if (strcmp(buffer, identifierStrings[j]) == 0) {  
 isHadIden = true;  
 break;  
 }  
 }  
 if (isHadIden) {  
 token.attribute = j;  
 } else {  
 //identifierStrings[identifierId] = buffer; // BUG!!!  
 strcpy(identifierStrings[identifierId],buffer);  
 token.attribute = identifierId++;  
 }  
}

#### 常数的识别

主要利用数字必定为数字字符开头，如果在开头读到数字字符，则直接把该数字字符放回流中，利用fscanf扫描流中的完整数字，从而达到对常数的提取。

核心代码如下

if (isDigit(ch)) {  
 token.category = stringToCategory(categoryStrings[CONSTANT]); // toekn类型为常数  
 ungetc(ch, fp); // 将读取到的数字字符放回流中  
 fscanf(fp, "%d", &token.attribute); // 将数字值放入属性值  
 }

#### 算符和界符的识别

利用getchar到的ch字符对比识别即可，比较简单。

核心代码如下

// 根据读取到的字符判断 Token 类型  
 switch (ch) {  
 case EOF:  
 token.category = -1;  
 break;  
 case '+':  
 token.category = stringToCategory(categoryStrings[2]);  
 break;  
 case '-':  
 token.category = stringToCategory(categoryStrings[3]);  
 break;  
 case '\*':  
 token.category = stringToCategory(categoryStrings[4]);  
 break;   
 case '/':  
 token.category = stringToCategory(categoryStrings[5]);  
 break;  
 case '=':  
 token.category = stringToCategory(categoryStrings[6]);  
 break;  
 case '#':  
 token.category = stringToCategory(categoryStrings[7]);  
 break;  
 case '<':  
 token.category = stringToCategory(categoryStrings[8]);  
 break;  
 case '>':  
 token.category = stringToCategory(categoryStrings[9]);  
 break;  
 case ':':  
 if (ch = fgetc(fp) == '=')  
 token.category = stringToCategory(categoryStrings[10]);  
 break;  
 case '(':  
 token.category = stringToCategory(categoryStrings[11]);  
 break;  
 case ')':  
 token.category = stringToCategory(categoryStrings[12]);  
 break;  
 case ',':  
 token.category = stringToCategory(categoryStrings[13]);  
 break;  
 case '.':  
 token.category = stringToCategory(categoryStrings[14]);  
 break;  
 case ';':  
 token.category = stringToCategory(categoryStrings[15]);  
 break;  
 }

### 2、单词的种别划分

#### 要求

标识符：作为一种

常数：作为一种

算符和界符：每种作为一个单独种别

关键字：每个作为一个单独种别

#### 代码实现

// 定义单词种别的字符串表示数组  
const char \*categoryStrings[] = {  
 "IDENTIFIER", // 标识符  
 "CONSTANT", // 常数  
 "OPERATOR\_ADD", // 加法运算符 '+'  
 "OPERATOR\_SUB", // 减法运算符 '-'  
 "OPERATOR\_MUL", // 乘法运算符 '\*'  
 "OPERATOR\_DIV", // 除法运算符 '/'  
 "OPERATOR\_EQ", // 等号 '='  
 "OPERATOR\_NE", // 不等号 '#'  
 "OPERATOR\_LT", // 小于 '<'  
 "OPERATOR\_GT", // 大于 '>'  
 "OPERATOR\_ASSIGN", // 赋值运算符 ':='  
 "DELIMITER\_LPAREN", // 左括号 '('  
 "DELIMITER\_RPAREN", // 右括号 ')'  
 "DELIMITER\_COMMA", // 逗号 ','  
 "DELIMITER\_PERIOD", // 句号 '.'  
 "DELIMITER\_SEMICOLON", // 分号 ';'  
 "KEYWORD\_BEGIN", // begin  
 "KEYWORD\_END", // end  
 "KEYWORD\_IF", // if  
 "KEYWORD\_THEN", // then  
 "KEYWORD\_WHILE", // while  
 "KEYWORD\_DO", // do  
 "KEYWORD\_CONST", // const  
 "KEYWORD\_VAR", // var  
 "KEYWORD\_CALL", // call  
 "KEYWORD\_PROCEDURE", // procedure  
 "KEYWORD\_ODD" // odd  
};  
  
// 定义单词种别  
typedef enum {  
 IDENTIFIER, // 标识符  
 CONSTANT, // 常数  
 OPERATOR\_ADD, // 加法运算符 '+'  
 OPERATOR\_SUB, // 减法运算符 '-'  
 OPERATOR\_MUL, // 乘法运算符 '\*'  
 OPERATOR\_DIV, // 除法运算符 '/'  
 OPERATOR\_EQ, // 等号 '='  
 OPERATOR\_NE, // 不等号 '#'  
 OPERATOR\_LT, // 小于 '<'  
 OPERATOR\_GT, // 大于 '>'  
 OPERATOR\_ASSIGN, // 赋值运算符 ':='  
 DELIMITER\_LPAREN, // 左括号 '('  
 DELIMITER\_RPAREN, // 右括号 ')'  
 DELIMITER\_COMMA, // 逗号 ','  
 DELIMITER\_PERIOD, // 句号 '.'  
 DELIMITER\_SEMICOLON, // 分号 ';'  
 KEYWORD\_BEGIN, // begin  
 KEYWORD\_END, // end  
 KEYWORD\_IF, // if  
 KEYWORD\_THEN, // then  
 KEYWORD\_WHILE, // while  
 KEYWORD\_DO, // do  
 KEYWORD\_CONST, // const  
 KEYWORD\_VAR, // var  
 KEYWORD\_CALL, // call  
 KEYWORD\_PROCEDURE,// procedure  
 KEYWORD\_ODD // odd  
} Category;  
  
// 定义单词符号结构体  
typedef struct {  
 Category category; // 单词的种别  
 int attribute; // 属性值，可以根据需要选择合适的数据类型  
} Token;

### 3、PL/0的语言的词法分析器将要完成以下工作：

#### （1） 跳过分隔符（如空格，回车，制表符）；

如果遇到分隔符，继续getchar，知道遇到有意义的字符。

核心代码如下

// 跳过空白字符  
do {  
 ch = fgetc(fp);  
} while(ch == ' ' || ch == 10 || ch == 9 || ch == 13);

#### （2） 识别诸如begin，end，if，while等关键字；

前面关键字和标识符的识别部分已经给出思路和核心代码

#### （3） 识别非关键字的一般标识符。

前面关键字和标识符的识别部分已经给出思路和核心代码

#### （4） 识别常数数字序列。

前面常数识别部分已经给出了思路和核心代码

#### （5） 识别前面列出的单字符操作符和:=双字符特殊符号。

前面算符和界符识别部分已经给出了思路和核心代码

### 4、词法分析器的实现方式：

建议把词法分析器设计成一个***\*独立子程序\****，运行一次产生一个单词符号。

【提示】实验二要在实验一基础上编写，把词法分析程序写出独立子程序，语法分析程序才可以反复调用以获取单词符号（终结符）。

综合上面的思路和核心代码，词法分析器核心识别如下

// 词法分析，得到单词符号  
// getToken 函数：从文件中读取下一个 Token  
Token getToken(FILE \*fp) {  
 Token token;  
  
 // 跳过空白字符  
 do {  
 ch = fgetc(fp);  
 } while(ch == ' ' || ch == 10 || ch == 9 || ch == 13);  
  
 // 根据读取到的字符判断 Token 类型  
 switch (ch) {  
 case EOF:  
 token.category = -1;  
 break;  
 case '+':  
 token.category = stringToCategory(categoryStrings[2]);  
 break;  
 case '-':  
 token.category = stringToCategory(categoryStrings[3]);  
 break;  
 case '\*':  
 token.category = stringToCategory(categoryStrings[4]);  
 break;   
 case '/':  
 token.category = stringToCategory(categoryStrings[5]);  
 break;  
 case '=':  
 token.category = stringToCategory(categoryStrings[6]);  
 break;  
 case '#':  
 token.category = stringToCategory(categoryStrings[7]);  
 break;  
 case '<':  
 token.category = stringToCategory(categoryStrings[8]);  
 break;  
 case '>':  
 token.category = stringToCategory(categoryStrings[9]);  
 break;  
 case ':':  
 if (ch = fgetc(fp) == '=')  
 token.category = stringToCategory(categoryStrings[10]);  
 break;  
 case '(':  
 token.category = stringToCategory(categoryStrings[11]);  
 break;  
 case ')':  
 token.category = stringToCategory(categoryStrings[12]);  
 break;  
 case ',':  
 token.category = stringToCategory(categoryStrings[13]);  
 break;  
 case '.':  
 token.category = stringToCategory(categoryStrings[14]);  
 break;  
 case ';':  
 token.category = stringToCategory(categoryStrings[15]);  
 break;  
 default:  
 // 如果读取到的是数字，则继续读取直到非数字字符，并将其转换为整数  
 if (isDigit(ch)) {  
 token.category = stringToCategory(categoryStrings[CONSTANT]); // toekn类型为常数  
 ungetc(ch, fp); // 将读取到的数字字符放回流中  
 fscanf(fp, "%d", &token.attribute); // 将数字值放入属性值  
 } else if (isLetter(ch)) {  
 // 如果读取到的字符是字母  
 // 有可能为IDENTIFIER  
 // 有可能为KEYWORD  
 // 思路是整个缓冲区，往缓冲区里写单词  
 // 再对比和关键词表中的关键词是否一致  
 // 如果不一致，铁标识符  
 // 标识符只能是字母，最大长度为10  
 char buffer[100]; // 假设标识符或关键字的最大长度为 100  
 int i = 0;  
 buffer[i++] = ch;  
  
 // BUG  
 // while ((ch = fgetc(fp)) != EOF && isLetter(ch = fgetc(fp))) {  
 while ((ch = fgetc(fp)) != EOF && (isLetter(ch) || isDigit(ch))) {  
 buffer[i++] = ch;  
 }  
 ungetc(ch,fp); // 将读取到的非字母字符放回去  
 buffer[i] = '\0';  
 //puts(buffer);  
 //token.category = -1;  
 //return token;  
 // 检查是否是关键字  
 // 假设有一个关键字列表  
 char \*keywords[] = {"begin", "end", "if", "then", "while", "do", "const", "var", "call" , "procedure", "odd"}; // 关键字表  
 int numKeywords = sizeof(keywords) / sizeof(keywords[0]);  
 for (int j = 0; j < numKeywords; j++) {  
 if (strcmp(buffer, keywords[j]) == 0) {  
 token.category = stringToCategory(categoryStrings[16+j]);  
 return token;  
 }  
 }  
  
 // 不是关键字则为标识符  
 // 不过应该先查找原标识符表中是否存在该word  
 // 不存在再id++  
 // 这块出了BUG  
 token.category = stringToCategory("IDENTIFIER");  
 bool isHadIden = false;  
 int numIdentifier = sizeof(identifierStrings) / sizeof(identifierStrings[0]);  
 int j = 0;  
 for (; j < numIdentifier; j++) {  
 if (strcmp(buffer, identifierStrings[j]) == 0) {  
 isHadIden = true;  
 break;  
 }  
 }  
 if (isHadIden) {  
 token.attribute = j;  
 } else {  
 //identifierStrings[identifierId] = buffer; // BUG!!!  
 strcpy(identifierStrings[identifierId],buffer);  
 token.attribute = identifierId++;  
 }  
 }  
 }  
  
 return token;  
}

### 5、词法分析器的输出形式（建议）

（种别，属性值）

其中：种别在“2、单词的种别”中进行了定义；  
属性值：若单词种别只代表唯一单词，属性值为空；  
 若单词种别是标识符，为该单词在标识符表中的位置；  
 若单词种别是常数，属性值为对应常数值。

【提示】此处建议的是实验报告或实验检查时的输出格式，词法分析器作为子程序应返回的是一个***\*单词符号\****，是一个预先定义的结构，最终程序的输出应该是在外部对单词符号内容的逐个打印输出，而不是在词法分析器内部直接进行打印输出。

main函数中调用分析和打印的代码如下

while ((token = getToken(preFile)).category != -1) {  
 //打印验证  
 printToken(token);  
 }

打印函数

// 定义打印单词符号的函数  
void printToken(Token token) {  
 if (token.category == 0 || token.category == 1) {  
 printf("(%s, %d)\n", categoryStrings[token.category], token.attribute);  
 } else {  
 printf("(%s, NULL)\n", categoryStrings[token.category]);   
 }  
}

6、标识符表可以简单定义为内容不重复的字符串数组。

标识符表定义为全局变量的指针数组

int identifierId = 0; // 标识符表中的位置  
char \*identifierStrings[1000]; // 标识符表

其赋值条件为如果遇到新的标识符，将其加入到标识符表

token.category = stringToCategory("IDENTIFIER");  
 bool isHadIden = false;  
 int numIdentifier = sizeof(identifierStrings) / sizeof(identifierStrings[0]);  
 int j = 0;  
 for (; j < numIdentifier; j++) {  
 if (strcmp(buffer, identifierStrings[j]) == 0) {  
 isHadIden = true;  
 break;  
 }  
 }  
 if (isHadIden) {  
 token.attribute = j;  
 } else {  
 //identifierStrings[identifierId] = buffer; // BUG!!!  
 strcpy(identifierStrings[identifierId],buffer);  
 token.attribute = identifierId++;  
 }

7、程序的输入：可以读取标准输入，或者打开指定的源程序文件。

本程序从文件输入， 通过命令行参数参数输入，运行后提示要分析的文件名都做了，只要得到文件名就可。核心代码如下

int main(int argc, char \*argv[]) {  
 //得到文件名  
 FILE \*file;  
 FILE \*preFile;  
 char fileName[MAXFILENAME];  
 char preFileName[MAXFILENAME+2];  
 if(argc > 1) {  
 strcpy(fileName, argv[1]);  
 } else {  
 printf("请输入要分析的文件名:\n");  
 scanf("%99s",fileName);  
 }  
 //puts(fileName);  
 //printf("%s",fileName);  
}

【选做内容】 使用LEX工具实现词法分析

1、基于pl0-lex项目，验证使用flex工具自动生成PL/0语言的词法分析器。

2、研究pl0.l.l文件内容，理解LEX源文件的编写规则和实际意义。