**实验一 词法分析程序**

**57119108 吴桐**

**一、实验目的**

通过本实验的编程实践，了解词法分析过程，以词法DFA为指导编写程序，分析输入字符串的正确性，即是否符合规定的词法规则，深度理解词法分析程序设计的原理和构造方法。

**二、实验内容**

用C或C++语言编写一个简单的词法分析程序，扫描C语言小子集的源程序，根据给定的词法规则识别单词并生成Token序列。如果产生词法错误，则给出提示并终止分析。

**三、实验设计思路**

首先选定分析使用的词法规则：

<标识符>🡪<字母>(字母|数字)\*  
<常量>🡪<数字>(<数字>)\*  
<字母>🡪a|b|c|……|x|y|z  
<数字>🡪0|1|2|3|4|5|6|7|8|9  
<加法运算符>🡪+|-  
<乘法运算符>🡪\*|／  
<关系运算符>🡪<|>|!=|>=|<=|==  
<分界符>🡪,|;|(|)|{|}  
<保留字>🡪 auto|break|case|char|class|const|continue|default|do|double|else|enum|extern|float|for|goto|if|int|long|main|register|return|short|signed|sizeof|static|struct|switch|typedef|union|unsigned|void|volatile|while

根据以上词法规则构造DFA（包含错误提示状态），之后根据DFA构造词法分析程序。词法分析程序主体为一个switch语句，每个状态对应一个case，当前状态用整形变量state记录，状态转移使用if语句与state赋值语句实现。

**四、DFA的设计**

根据词法规则构造的DFA如图1所示。

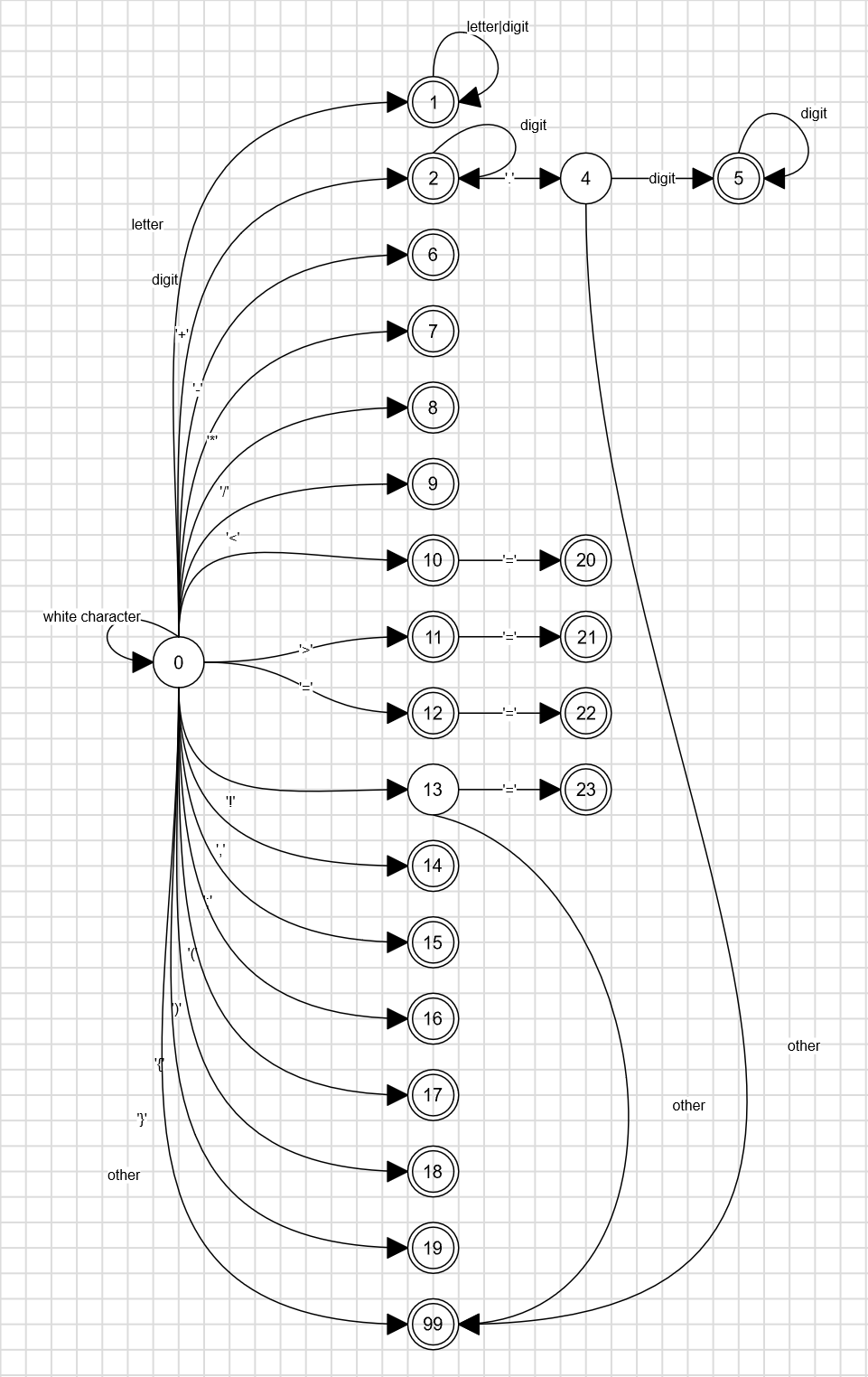


图1 词法分析DFA

这里解释一下初始状态和终止状态的含义：

case 0：初始状态，可用于匹配space、‘\n’、‘\t’、‘\0’等white character。如果输入字符为EOF（文件终止字符）则说明文件分析成功，输出“Successfully Done!”，否则全部认为是非法字符，进入状态99。

case 1：标识符（ID）或关键字（KEY）

case 2：十进制整数（INT）

case 5：十进制小数（FLOAT）

case 6：加法运算符‘+’（ADD）

case 7：减法运算符‘-’（MINUS）

case 8：乘法运算符‘\*’（MULTI）

case 9：除法运算符‘/’（DIVEDE）

case 10：关系运算符小于‘<’（Less Than，LT）

case 11：关系运算符大与‘>’（Greater Than，GT）

case 12：赋值运算符‘=’（Assignment，ASS）

case 14：分隔符逗号‘,’（COMMA）

case 15：分隔符分号‘;’（SEMICOLON）

case 16：分隔符左小括号‘(’（Left Parenthesis，LP）

case 17：分隔符右小括号‘)’（Right Parenthesis，RP）

case 18：分隔符左大括号‘{’（Left Brace，LB）

case 19：分隔符右大括号‘}’（Right Brace，RB）

case 20：关系运算符小于等于‘<=’（Less or Equal to，LE）

case 21：关系运算符大于等于‘>=’（Greater or Equal to，GE）

case 22：关系运算符等于‘==’（Equal to，EQ）

case 23：关系运算符不等于‘!=’（Not Equal to，NEQ）

case 99：错误提示状态（Error）

**五、核心算法描述**

词法分析程序主体为一个switch语句，每个状态对应一个case，当前状态用整形变量state记录，状态转移使用if语句与state赋值语句实现。下面以状态1为例说明算法构造原则。

状态1有一条发出边，并且为终止状态。因此case块中有一个if语句，判断是否符合转移条件，发出边的目的状态仍为状态1，因此转移后state值仍未1。如果读入了不符合转移条件的符号，则认为该字符是下一个词的首字符，我们首先需要将该字符回退到输入流中，以便于之后的词法分析过程，之后判定当前得到的标识符是否为关键字，输出该词的Token，将状态置为0进行下一个词的分析。

case 1:

ch = getchar(); //读入字符

if ((ch >= 'a' && ch <= 'z') || (ch >= '0' && ch <= '9')) { //输入为数字或者小写字母

state = 1;

lexeme[sz++] = ch; //在日志中记录当前字符

}

else { //读入了下一个词的首字符

ungetc(ch, stdin); //将下一个词的首字符回退到输入流中

if (!isKey()) prt("id","ID"); //判断是否为关键字，若不是则认定为一般标识符并输出Token

state = 0; //回到初始状态，进行下一个词的分析

}

break;

其中，isKey函数用来判定当前标识符是否为关键字。程序中内设了34个关键字，将日志中记录的词与关键字库进行比对，如果匹配成功则直接认定该词为关键字，否则认为是一般标识符。

int isKey() {

int k = 0;

lexeme[sz] = '\0'; //限定词的结束

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

if (!strcmp(lexeme, key[i])) { //进行字符串比较，等于0时说明两字符串相等

printf("(%s,%s,", lexeme, "keyword");

for (int k = 0; k < strlen(key[i]); ++k) { //输出大写形式的关键词字符串

printf("%c",toupper(key[i][k]));

}

printf(")\n");

sz = 0;

return 1;

}

}

return 0;

}

其余状态的设计都是大同小异，具体见源代码lexicalAnalyzer.c。

**六、测试用例**

输入用例如图2所示。

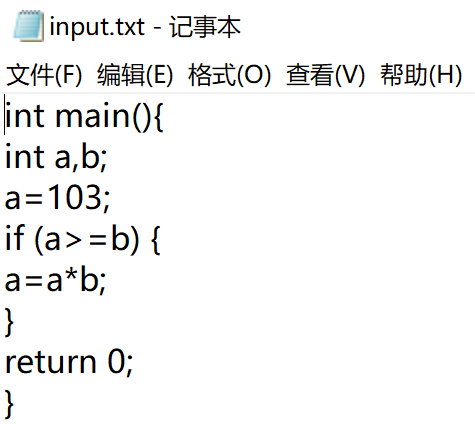


图2 输入用例

相应的输出如图3所示。可见词法分析程序可以正确地将一个简单的C语言源程序转换为相应的Token序列。

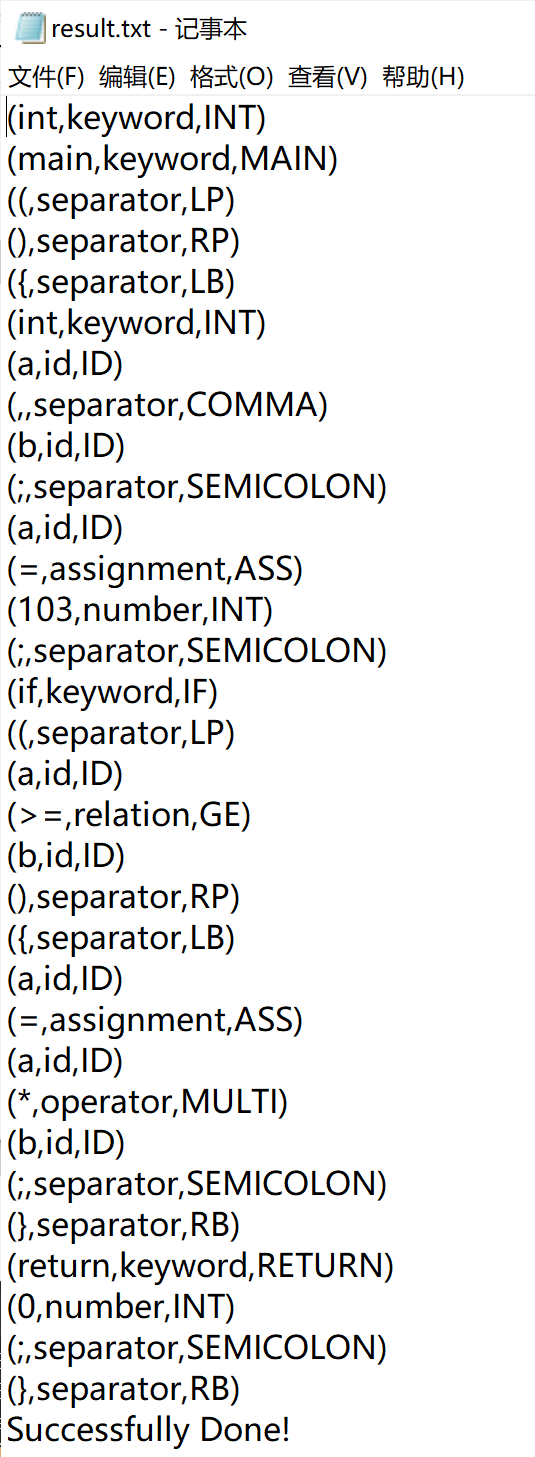


图3 输出用例

**七、出现的问题及解决办法**

在实验过程中，遇到了较多的细节问题，例如：DFA的设计，特殊情况的考虑，字母的大小写转换并输出，内部码的设定，文件读写等。总体来说都不是很复杂的问题，可以通过查找资料自己解决。

在DFA设计过程中，由于状态数较多，前前后后更改了五到六版才确定下来。文件读写问题可以通过freopen语句实现。比较繁琐的就是特殊情况的处理，例如错误判定和分析结束判定。最终我选择了EOF（文件终止字符）作为分析结束的标志，将space、‘\n’、‘\t’、‘\0’设定为程序可以接受的white character，其余未设定字符均视为非法字符，引发分析错误。

**八、实验总结**

通过本次实验，我深入了解了词法分析的过程，加深了对编译器的工作原理的理解，利用编程解决实际问题，也获得了一些编程经验。这次实验还有可以改进的地方，如实现更复杂词法集的分析，或者编写一个完整的Lex，实现根据词法自动地生成词法分析程序。总的来说，本次实验从词法规则选择到DFA设计，再到具体程序实现，让我体验了一个完整的科研学习过程，从中也学习到了新的编程实践知识。