

# 专题1\_词法分析程序设计原理与实现技术

|  |  |  |
| --- | --- | --- |
| 学 号： | 16281002 | |
| 姓 名： | 杜永坤 | |
| 专 业： | 计算机科学与技术 | |
| 学 院： | 计算机与信息技术学院 | |
|  |  |  |
|  |  |  |
| 提交日期： | 2019年04月21日 | |

### 实验要求：

**[**实验项目**]**

以下为正则文法所描述的C语言子集单词符号的示例，请补充单词符号：++，--，>>, <<, += , -= ,\*=, /= ，&&（逻辑与），||（逻辑或），！（逻辑非）等等，给出补充后描述C语言子集单词符号的正则文法，设计并实现其词法分析程序。

<标识符>→字母︱<标识符>字母︱<标识符>数字

<无符号整数>→数字︱<无符号整数>数字

<单字符分界符> →+ ︱－︱\* ︱；︱,︱(︱) ︱{︱}

<双字符分界符>→<大于>=︱<小于>=︱<小于>>︱<感叹号>=︱<等于>=︱<斜竖>\*

<小于>→< <等于>→=<大于>→><斜竖> →/

<感叹号>→!

该语言的保留字：void、int、float、double、if、else、for、do、while 等等（也可补充）。

**[**设计说明**]**

（1）可将该语言设计成大小写不敏感，也可设计成大小写敏感，用户定义的标识符最长不超过32个字符；

（2）字母为a-z A-Z，数字为0-9；

（3）可以对上述文法进行扩充和改造；

（4）“/\*……\*/”和“//”(一行内)为程序的注释部分。

[设计要求]

（1）给出各单词符号的类别编码；

（2）词法分析程序应能发现输入串中的错误；

（3）词法分析作为单独一遍编写，词法分析结果为二元式序列组成的中间文件；

（4）设计两个测试用例（尽可能完备），并给出测试结果。

[任务分析]

重点解决正则文法到状态转换图的转化问题，词法分析程序的实现。

### 实验过程：

##### 1、文法扩充修改

（1）、其中我将每个单字符作为一种类别进行分析；

（2）、加入了保留字 main；

（3）、扩充双字符，并将每一个种类的双字符作为一种类别；

（4）、加入左右注释和单行注释。

扩充修改后的文法：

<标识符>→字母︱ 字母<标识符>︱ <标识符>数字

<无符号整数>→数字︱数字 <无符号整数>

<单字符分界符> →; ︱, ︱(︱) ︱{︱}

//每种运算符都有自己的意义，就跟保留字一样，应当单独分类

//单字符

<加号>→+

<减号>→-

<位与>→&

<位或>→|

<大于>→>

<小于>→<

<星>→\*

<斜杠>→/

<感叹号>→!

<赋值>→=

//双字符

<大于等于>→><赋值>

<右移>→><大于>

<小于等于>→<<赋值>

<左移>→<<小于>

<不等于>→<<大于>︱!<赋值>

<等于>→=<赋值>

<左注释>→/<星>

<右注释>→\*<斜杠>

<单行注释>→/<斜杠>

<加等于>→+<赋值>

<减等于>→-<赋值>

<乘等于>→\*<赋值>

<除等于>→/<赋值>

<与等于>→&<赋值>

<或等于>→|<赋值>

<加加>→+<加号>

<减减>→-<减号>

<与>→&<位与>

<或>→|<位或>

该语言的保留字 ：void int float double if else for do while 等等（也可补充，本程序补充 main 为保留字）。对于每一个的保留字均是一种类别

##### 2、词法分析程序设计说明

设计要求：

（1）给出各单词符号的类别编码；

（2）词法分析程序应能发现输入串中的错误；

（3）词法分析作为单独一遍编写，词法分析结果为二元式序列组成的中间文件；

（4）设计两个测试用例（尽可能完备），并给出测试结果。

设计说明：

（1）该语言大小写不敏感；

（2）字母为a-z A-Z，数字为0-9；

（3）可以对上述文法进行扩充和改造；

（4）“/\*……\*/”为程序的注释部分;

（5）“/\*”作为左注释，“\*/”作为右注释，分别作为一种类别，所以左注释和右注释必须成对出现，否则错误；左右注释中字符忽略不进行识别；

（6）“//”作为单行注释，“//”后的字符忽略不进行识别。

##### 3、程序功能描述

（1）、能够识别一个.c文件中的标识符、无符号整数；

（2）、能够识别一个.c文件中的一部分关键字：void、int、float、double、if、else、for、do、while、main（等等）；

（3）、能够识别一个.c文件中的算术运算符和逻辑运算符等字符符号；

（4）、词法分析结果为二元式组成的中间文件。

##### 4、主要的数据结构描述

词法分析程序LexicalAnalysisProcess ()的主体是一个不断扫描输入串的扫描器和配合该扫描器进行分析的符号表，因此在本程序中，设计了一个描述符号表的类SymbolTable和一个描述扫描器动作的类LexicalScanner。

符号表*symbolTableList*采用List<String>列表保存，在程序中写入，而各类单词符号的类别编码正好可以用其在列表中的序号表示。

**private** **static** List<String> *symbolTableList* = **new** ArrayList<String>();

列表的内容为：

*symbolTableList*.add("标识符");

*symbolTableList*.add("无符号整数");

*symbolTableList*.add("单字符分界符");

*symbolTableList*.add("void");

*symbolTableList*.add("int");

*symbolTableList*.add("float");

*symbolTableList*.add("double");

*symbolTableList*.add("if");

*symbolTableList*.add("else");

*symbolTableList*.add("for");

*symbolTableList*.add("do");

*symbolTableList*.add("while");

*symbolTableList*.add("main");

*symbolTableList*.add("加号");

*symbolTableList*.add("减号");

*symbolTableList*.add("位与");

*symbolTableList*.add("位或");

*symbolTableList*.add("大于");

*symbolTableList*.add("小于");

*symbolTableList*.add("星");

*symbolTableList*.add("斜竖");

*symbolTableList*.add("感叹号");

*symbolTableList*.add("赋值");

*symbolTableList*.add("大于等于");

*symbolTableList*.add("右移");

*symbolTableList*.add("小于等于");

*symbolTableList*.add("左移");

*symbolTableList*.add("不等于");

*symbolTableList*.add("等于");

*symbolTableList*.add("左注释");

*symbolTableList*.add("右注释");

*symbolTableList*.add("单行注释");

*symbolTableList*.add("加等于");

*symbolTableList*.add("减等于");

*symbolTableList*.add("乘等于");

*symbolTableList*.add("除等于");

*symbolTableList*.add("与等于");

*symbolTableList*.add("或等于");

*symbolTableList*.add("加加");

*symbolTableList*.add("减减");

*symbolTableList*.add("与");

*symbolTableList*.add("或");

扫描器LexicalScanner需要读取.c文件中的字符串，用FileReader类打开文件，而由于在词法分析的过程中需要回退字符，因此，用PushbackReader类封装FileReader类作为扫描读取文件字符的主体类。

**private** PushbackReader pr;

##### 5、程序结构描述

LexicalScanner类中用于扫描文件并进行词法分析的主体函数是LexicalAnalysisProcess ()，该函数内部是一个while循环，不断读取文件中的一个字符，一旦扫描到文件尾，词法分析程序自然退出。

开始识别字符符号，由于想要补充单词符号的识别，因此重新构建了文法如下：

<标识符>→字母︱ 字母<标识符>︱ <标识符>数字

<无符号整数>→数字︱数字 <无符号整数>

<单字符分界符> →; ︱, ︱(︱) ︱{︱}

//每种运算符都有自己的意义，就跟保留字一样，应当单独分类

//单字符

<加号>→+

<减号>→-

<位与>→&

<位或>→|

<大于>→>

<小于>→<

<星>→\*

<斜杠>→/

<感叹号>→!

<赋值>→=

//组合字符

<大于等于>→><赋值>

<右移>→><大于>

<小于等于>→<<赋值>

<左移>→<<小于>

<不等于>→<<大于>︱!<赋值>

<等于>→=<赋值>

<左注释>→/<星>

<右注释>→\*<斜杠>

<单行注释>→/<斜杠>

<加等于>→+<赋值>

<减等于>→-<赋值>

<乘等于>→\*<赋值>

<除等于>→/<赋值>

<与等于>→&<赋值>

<或等于>→|<赋值>

<加加>→+<加号>

<减减>→-<减号>

<与>→&<位与>

<或>→|<位或>

该语言的保留字 ：void int float double if else for do while main

之所以如此设计，是由于以上每种运算符或者符号，保留字都有自己的意义，因此必须将其作为不同类别的单词符号保存在符号表中。

其实在编写该程序的时候发现，该文法其实并不适合用状态装换图的方式，状态转换图应当用于识别一类特别复杂的字符串，而本文法中都是简单的单词符号或者运算符，只有识别标识符或者无符号整数时，状态转换图才有比较高的效率，但是为了体现状态转换图的思想，我在本程序中，把对保留字的识别、对标识符的识别和对无符号整数的识别集中在一起用状态转换图的方式去识别，而其他运算符的识别由于过于简单就单独识别。状态装换图构建如下：

###### 状态转换图：



注：上面的状态装换图不完整，任何一步的识别出错就转入中间状态36，此处由于连线太复杂而略去；双字符和单字符没有加入状态转换图。

**if**(taken == **null**)

**continue**;

//查表，输出

categoryCode = lookupSymbolTableList();

outMidFile(); //输出中间文件

变量taken经过主体程序LexicalAnalysisProcess的识别后每次识别出一个的一个字符，每次识别出一个字符串，调用lookupSymbolTableList方法查询符号表得到它的类别编码，然后调用outMidFile ()方法向文件中写二元式。

如果最后查表时，没有找到该类字符的类别编码，说明该类字符是一个不符合规则的字符，在分析时要打印输出错误。

##### 6、程序测试

本次实验有两个测试文件，一个测试分析成功，一个测试分析失败：

**test1.c文件：**

/\*\*

左右注释

\* \*/

void main(){

int num1 = 123;

int num2 = 123;

float num3 = 1234;

double num4 = 12345;

for(int i = 0;i < 100;++i){

do{

if(num1==num2||num3!=num4){

--num2;

return 0;

}else{

return num1&=num2;

}

}while(num1&&num2)

}

//单行注释

int d = num1<<5;

int c = num2>>6;

int a = num1+num2;

int b = num1-num2;

int c = num1\*num2;

int d = num1/num2;

a+=b;

a-=b;

a\*=b;

a/=b;

a|=b;

a&=b;

}

**词法分析结果中间文件结果：**

[30,/\*]

[31,\*/]

[4,void]

[13,main]

[3,(]

[3,)]

[3,{]

[5,int]

[1,num1]

[23,=]

[2,123]

[3,;]

[5,int]

[1,num2]

[23,=]

[2,123]

[3,;]

[1,float]

[1,num3]

[23,=]

[2,1234]

[3,;]

[1,double]

[1,num4]

[23,=]

[2,12345]

[3,;]

[10,for]

[3,(]

[5,int]

[1,i]

[23,=]

[2,0]

[3,;]

[1,i]

[19,<]

[2,100]

[3,;]

[39,++]

[1,i]

[3,)]

[3,{]

[11,do]

[3,{]

[8,if]

[3,(]

[1,num1]

[29,==]

[1,num2]

[42,||]

[1,num3]

[28,!=]

[1,num4]

[3,)]

[3,{]

[40,--]

[1,num2]

[3,;]

[1,return]

[2,0]

[3,;]

[3,}]

[9,else]

[3,{]

[1,return]

[1,num1]

[37,&=]

[1,num2]

[3,;]

[3,}]

[3,}]

[12,while]

[3,(]

[1,num1]

[41,&&]

[1,num2]

[3,)]

[3,}]

[32,//]

[5,int]

[1,d]

[23,=]

[1,num1]

[27,<<]

[2,5]

[3,;]

[5,int]

[1,c]

[23,=]

[1,num2]

[25,>>]

[2,6]

[3,;]

[5,int]

[1,a]

[23,=]

[1,num1]

[14,+]

[1,num2]

[3,;]

[5,int]

[1,b]

[23,=]

[1,num1]

[15,-]

[1,num2]

[3,;]

[5,int]

[1,c]

[23,=]

[1,num1]

[20,\*]

[1,num2]

[3,;]

[5,int]

[1,d]

[23,=]

[1,num1]

[21,/]

[1,num2]

[3,;]

[1,a]

[33,+=]

[1,b]

[3,;]

[1,a]

[34,-=]

[1,b]

[3,;]

[1,a]

[35,\*=]

[1,b]

[3,;]

[1,a]

[36,/=]

[1,b]

[3,;]

[1,a]

[38,|=]

[1,b]

[3,;]

[1,a]

[37,&=]

[1,b]

[3,;]

[3,}]

**test2.c文件**：

int a = 345;

int c = a%3;

int 2num = 34;

**词法分析结果：**

[5,int]

[1,a]

[23,=]

[2,345]

[3,;]

[5,int]

[1,c]

[23,=]

[1,a]

Error:未定义的标识符：%

[2,3]

[3,;]

[5,int]

Error:未定义的标识符：2num

[23,=]

[2,34]

[3,;]

词法分析失败，二元式文件生成失败...