**课程编号：C0801003030**

**编译方法实验报告**



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **姓名** | **岑哲栋** | **学号** | | **20165018** |
| **班级** | **软件1607** | **指导教师** | | **刘洪娟** |
| **实验名称** | **扫描器设计及中间代码生成器设计** | | | |
| **开设学期** | **2018-2019学年秋季学期** | | | |
| **开设时间** | **第15周和第17周** | | | |
| **报告日期** | **2018年12月30日** | | | |
| **评定成绩** |  | | **评定人** | **刘洪娟** |
| **评定日期** | **2018年12月31日** |

**东北大学软件学院**

1. 实验目的

**实验 1：**

熟悉并实现一个扫描器（词法分析程序）。

**实验 2：**

熟悉算术表达式的语法分析与中间代码生成原理，实现算数表达式的中间代码生成器。

1. 实验内容

**实验 1：**

首先了解了扫描器的工作原理，之后仔细阅读题目中的要求，原本应当根据题目中的要求编写出相应的程序，但实验指导书中已经有现成的参考代码。现成代码为C语言代码，因此使用Codeblocks为开发环境。在充分了解该代码运作机制的情况下直接使用该代码进行测试，发现了一些存在的问题，并针对这些问题修改了部分的参考代码直至测试结果与预期结果无误。最后撰写实验报告。

**实验 2：**

首先学习了中间代码生成器的工作原理，之后仔细也读题目中的要求并编写出相应的程序。由于需要用到栈这种高级数据结构，如果使用C语言需要自己编写，比较费劲，而Java中有现成的Stack类可以直接使用，因此选择Java作为开发语言，并使用Eclipse作为开发环境。编写程序完毕后设计测试用例，对输入的数据进行测试，测试无误后撰写实验报告。

1. 实验原理及基本步骤

**实验 1：**

扫描器的工作方式如图所示

扫描器

单词 Token

扫描器具体有两个任务，一是识别单词，即从用户的源程序中把单词分离出来，二是翻译单词，即把单词转换成机内表示，便于后续处理。识别单词的任务本质上是由一个有限自动机来完成，这个有限自动机需要通过状态转换图或状态转换矩阵来确定。识别完成一个token之后，它的最终状态即代表了这个token的类型，在本实验中用一个(code,value)对的方式来表示token,特别要注意的是关键字和界符，它们需要专门存储起来用以比对token是否为关键字或界符。

**实验2：**

中间代码是高级程序语言中，各种语法成分的语义结构表示；它介于源语言和目标语言之间。设置它的目的如下：

(1) 便于进行与机器无关的代码优化；

(2) 使编译程序改变目标机更容易；

(3) 使编译程序的结构在逻辑上更为简单明确，以中间语言为界面，编译前端和后端的接口更清晰。

常用的中间代码有逆波兰式、四元式、语义树、三元式等，本实验中需要产生的结果是四元式。本实验中的终结符号为i, ω0, ω1,(,)，可以将i看作任意字母，ω0看作+，ω1看作-。经过推导可以看出题目中的文法最终产生的结果就是一个带有括号的加减运算表达式，这给测试阶段提供了宝贵的依据。

1. 数据结构设计

**实验 1：**

根据状态转换图设计如下的状态转换矩阵：

int aut[11][8]=

{

0, 0, 0, 0, 0, 0, 0, 0,

0, 2, 0, 0, 0, 8, 9,15,

0, 2, 3, 5,11, 0, 0,11,

0, 4, 0, 0, 0, 0, 0, 0,

0, 4, 0, 5,11, 0, 0,11,

0, 7, 0, 0, 6, 0, 0, 0,

0, 7, 0, 0, 0, 0, 0, 0,

0, 7, 0, 0,11, 0, 0,11,

0, 8, 0, 0, 0, 8, 0,12,

0, 0, 0, 0, 0, 0,10,14,

0, 0, 0, 0, 0, 0, 0,13

}

原本只有10种状态和7种类型的符号，为何这里要设计出11行来表示状态，8行来表示符号类型呢？因为C语言中数组下标从0开始，如果只设计10\*7的矩阵的话，每次根据行和列取得矩阵中的元素时，行和列都要-1，不符合一般人的理解方式，且增大了无谓的计算量。直接设计11\*8的矩阵，使得第0行和第0列都用0填充，这样根据行和列取得矩阵元素时就不需要-1，符合一般人的理解规律。

设计一个30\*12的字符二维数组来存储关键字和界符：

char keywords[30][12]={"program","begin","end","var","while","do","repeat",

"until","for","to","if","then","else",";", ":", "(", ")", ",",

":=", "+", "-", "\*", "/", ">", ">=", "==", "<", "<="} 尽管关键字和界符的总数只有28个，但为了方便以后扩展关键字或界符，将容量设为30。

char ID[50][12]; 用来存储源程序中读取出来的标识符。

double C[20]; 用来存储源程序中读取出来的常数。

结构体

struct token

{

int code;

int value;

}; 来表示token结构。

struct token tok[100]; 来存储识别出的token序列。

char strTOKEN[15]; 用来存储当前正在识别的token。

char w[50]; 来作为源程序的缓冲区，从该数组中取单词。

结构体：

struct map

{

char str[50];

int col;

}; 用来表示当前字符到状态转换矩阵标记的映射。

struct map col1[4]={{"0123456789",1},{".",2},{"Ee",3},{"+-",4}}; struct map col2[2]={{"abcdefghijklmnopqrstuvwxyz",5},{"0123456789",1}}; struct map col3[1]={{";:(),+-\*/=><",6}}; 分别表示数字、关键字或标识符、界符符号的映射。

**实验 2：**

char w; //表示当前正在处理的字符

Stack<String> SYN,SEM; //分别代表算符栈和语义栈

ArrayList<String> QT; //用来存储四元式序列

String alphabet="abcdefghijklmnopqrstuvwxyz"; //用来存储字母表

1. 关键代码分析（带注释）及运行结果

**关键代码：**

**实验 1：**

*判断单词类别：*

while (w[i]==' ') //滤空格

i++;

//判定单词类别

if (w[i]>='a' && w[i]<='z')

{

ptr=col2; num\_map=2; //关键字或标识符

}

else if (w[i]>='0' && w[i]<='9')

{

ptr=col1; num\_map=4; //数字

}

else if (strchr(col3[0].str,w[i])==NULL)

{

printf("非法字符%c\n",w[i]); //非法字符

i++; //跳过

continue;

}

else

{

ptr=col3; num\_map=1; //界符

}

*进行状态变换：*

int find(int s,char ch)

{

int i,col=7;

struct map \*p;

p=ptr; //ptr代表映射到当前字符对应的map数组指针，在判断单词类别时确定

for (i=0;i<num\_map;i++) //num\_map代表对应的map数组中包含的元素个数，和ptr同时确定

if (strchr((p+i)->str,ch)!=NULL&&ch!='\0')

{

col=(p+i)->col; //确定新状态在状态转换矩阵中的列号

break;

}

return aut[s][col];

}

**实验 2：**

*入口程序：*

*E*(); //开始字符为E，因此先进入E的方法

**if**(*w*=='#') { //如果最后一个字符为#

**for**(String qt:*QT*) {

System.***out***.println(qt); //打印出四元式序列

}

}

**else** //如果最后一个字符不是#

**throw** **new** Exception("没有正确结束");

*生成四元式的方法：*

**private** **static** **void** QUAT() {

String newTemp = "T"+*count*; //生成一个每次递增的临时变量名Tn

String s2 = *SEM*.pop(); //弹出符号栈的字符

String s1 = *SEM*.pop(); //弹出符号栈的字符

*QT*.add("(" + *SYN*.pop() + "," + s1 + "," + s2 + "," + newTemp + ")"); //弹出语义栈的字符，同时生成格式化的四元式

*SEM*.push(newTemp);

*count*++; //Tn递增

}

每个非终结字符中的方法基本已经在题目中给出了，具体代码只是将它们转换成了代码的形式，因此不再列出。

**运行结果：**

**实验 1：**

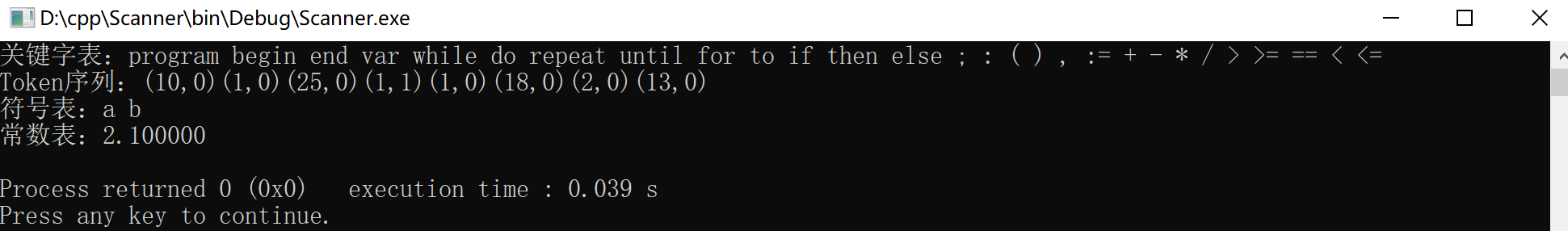
实验指导书中的示例代码在源程序代码只有一行时会出错，而且token数量会莫名其妙多出来一个，同时由于它用int型数组来保存常数值，因此无法正确显示小数。经修改后的程序解决了这些问题。

设计源程序如下：

if a==b

a:=2.1;

运行结果如图：

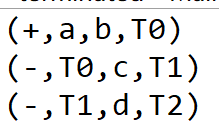


可以看到token序列完美地满足了题目中的要求：标识符的编码（code）为1，值（value）为其在符号表中的位置；常数的编码（code）为2，值（value）为其在常数表中的位置；关键字和界符的编码（code）为其在关键字表中的位置，值（value）为0。

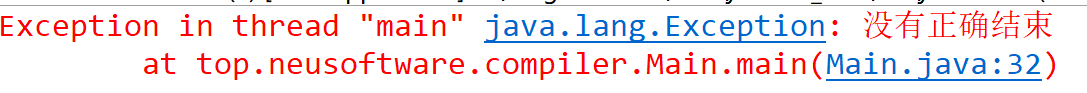
**实验 2：**

实验2会有三种错误情况，分别是“没有正确结束”，即最后一个字符不是#；“缺失右括号)”，即在前面已经读取到左括号的情况下后面没有右括号相对应；“非法字符”，即读出来的符号不属于该文法定义的符号，在本程序中我们设为除+-以外的非字母符号。针对以上几种错误情况设计四种测试用例如下：

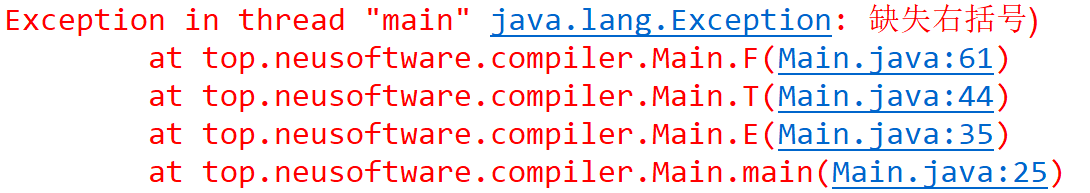
1. 正确的情况 设计读入的字符串为(a+b)-c-d#，运行结果成功输出了四元式序列如图：



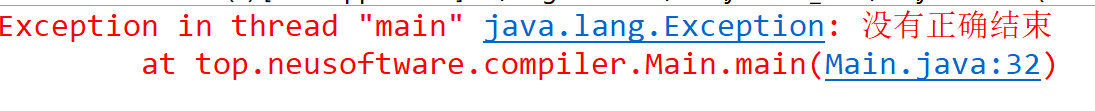
2. 没有正确结束 设计读入的字符串为(a+b)-c-d，运行结果如图



3. 缺失右括号 设计读入的字符为(a+b-c-d#，运行结果如图



4. 非法字符 设计读入的字符为(a+b)=c#，运行结果如图



1. 总结与分析

**实验 1：**

通过本实验更加深入地了解了扫描器的工作流程和内部实现方法。同时发现在别人的代码基础上进行修改完善也并不像想象中的那么容易。尽管本次实验提供了参考代码，不过参考代码中暗含着一些小问题，如果需要修正这些问题，需要对参考代码进行彻底的理解，其工作量和难度不亚于从头编写一个程序。同时自从学习了Java语言之后，再做的实验课以及自己日常编写的小程序都不再使用C语言编写了，而本次实验让我对于C语言又重新复习了一遍，并认识到C语言在处理像编译器这种过程化程序时候的快捷高效。而参考代码中一些创新的方法如设计矩阵时多出冗余的行和列来消除行号和列号的换算也让我眼前一亮。

**实验 2：**

原本想和实验1一样使用C语言进行编写，因为C语言正适合这种抽象的过程化程序。但是本实验需要用到栈这种高级数据结构，C语言里需要自己实现，而Java则有现成的类可以使用，因此选择了Java作为编写的语言。但是实际上只是借用了Java这个壳子，本质的思想依然是C语言的思想。具体代码实现中所有的变量都为静态变量，所有的方法都为静态方法，最终效果和C语言基本上是一样的。本次实验使我对于中间代码生成的过程有了更深的认识，对于今后的学习有很大的帮助。

**成绩评价表格**：

|  |  |
| --- | --- |
| 考核标准 | 得分 |
| （1）正确理解和掌握实验所涉及的基本概念和原理；（10%）； |  |
| （2）按实验要求合理设计数据结构和程序结构（20%）； |  |
| （3）实验中要设计各种测试用例，考虑词法、语法和语义正确和不正确等各种情况（20%）； |  |
| （4）认真记录实验结果，对实验结果的分析准确（20%）； |  |
| （5）实验过程中，具有严谨、认真的学习态度，不做与实验内容无关的其他学习工作（10%）； |  |
| （6）实验2应具有一定的创新性（10%）； |  |
| （7）实验报告内容充实、格式符合规范（10%）。 |  |