编译原理 实验报告

评 语	成 绩	
教 师：邓岳 年 月 日		

学院班级：1513011

学生学号：15130110047

学生姓名：郑 健

实验日期：2017.12

一. 实现目的

通过构造函数绘图语言的解释器，加深对编译器构造原理和方法的理解，巩固课堂所学内容。

(1)运用正规式设计简单语言的词法

(2)运用产生式设计简单语言的语法

(3)运用递归下降子程序编写语言的解释器

具体来说，主要完成以下功能：

(1)用词法分析器识别源程序中的记号

(2)用语法分析器识别记号流中的语句

(3)解释器：能够绘制相应图形，并设置图形颜色

二. 实现环境

IntelliJ IDEA

三. 实现内容

这一部分主要分三部分介绍：词法分析器、语法分析器、语义分析器

(一)词法分析器的构造过程：

(1)词法分析器的主要功能：

① 识别源程序文件中的记号，并供语法分析器使用

② 处理与平台相关的输入

③ 过滤源程序中无用的部分

④ 识别文件中的非法的输入记号，并报告错误

(2)总的实现步骤：正规式 → NFA → DFA → 最小DFA → Java语言程序 → 调试

(3)分析：对于该函数绘制语言来说，其记号种类共有6种：

① 常数：一类是字面量形式的普通数值；另一类是类似 PI、E 这种标识符形式的常量名

② 参数：在该绘图语言中只有一个参数了，已经被定义好的；
因此在绘图语言中无需变量和参数的定义和声明语句

③ 函数：该绘图语言包含六种函数调用：sin、cos、tan、sqrt、Exp 和 Ln

④ 保留字：具有固定含义的标识符，包括 ORIGIN、SCALE、ROT、IS、FOR、TO、STEP、DRAW、FROM，以及（设置图形颜色所需的保留字）COLOR、RED、BLUE、GREEN、PINK...

⑤ 运算符：+、-、*、/、**。结合性：** 不结合，+-*/ 为左结合
优先级：(高→低) **、-(负)、*/、二元+-

1

⑥分隔符： ; ( )

根据所有记号的种类，确定此函数绘图语言中记号类型的表示。

```
Public enum TokenType
{   ORIGIN, SCALE, ROT, IS, TO, STEP, DRAW, FOR, FROM,  // 保留字
    COLOR, RED, BLUE, GREEN, PINK,      // 颜色    T.  // 参数
    SEMICO, L_BRACKET, R_BRACKET, COMMA,  // 分隔符
    PLUS, MINUS, MUL, DIV, POWER,     // 运算符
    FUNC,  // 函数  CONST_ID,  // 常数    NONTOKEN;  // 空记号
    ERRTOKEN,  // 出错记号（非法输入）  COMMENT,  // 注释   NEXT_LINE,  // 换行符
}
```

同时，确定记号的数据结构

```
Pubic class Token
{   private TokenType type;     // 类别
    private String lexme;       // 词义，原始输入的字符串
    private double value;       // 若记号为常数，则其值为常数值
    private Function function;  // 若记号为函数，要与 Function 1)
    public Token (TokenType type, String lexme, double value, Function function)
    {.....}     // 构造函数
    // 各成员变量对应的 setter 和 getter 方法
    .....
}
```

在确定记号的类别和数据结构之后，首先确定模式的正则式表示

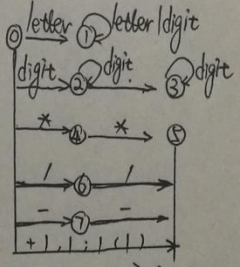
letter = [a-z A-z] digit = [0-9] COMMENT = "//" | "--"

WHITE-SPACE = {" " | \t | \n | \r}+ SEMICO = ";" L_BRACKET = "(" R_BRACKET = ")"

COMMA = "," PLUS = "+" MINUS = "-" MUL = "*" DIV = "/" POWER = "**"

CONST_ID = digit+ ("." digit*) ID = letter+ (letter | digit)*

由正则式构造 DFA：



在同一模式下，仍然有不同种类的记号，为了区分不同种类的记号，需构造符号表。

准备工作完成后，为了完成词法分析器的构造，主要由下列函数完成相应的功能，分别是 LexicalAnalysis(String file); getToken(); isWhiteSpace(int c); close();
相应的功能分别是初始化词法分析器，获取记号函数，判断是否为空格，关闭词法分析器。实现思路：初始化词法分析器利用 FileInputStream(), BufferedInputStream(), PushbackInputStream()等函数打开源程序，为getToken()函数读取源程序中的记号作准备。而getToken()函数从源程序文件中获取一个记号之后，按照定的顺序分别与符号表中预定义的符号进行比较。如果匹配到记号，则表示获取的记号是合法的，否则提示一个ERROTOKEN的提示信息。当读取完源程序中的语句后，调用词法分析器函数，关闭源程序。测试程序：反复利用getToken()函数从源程序文件中读取记号，如果匹配成功，就打印其对应的符号表中的条目，否则打印错误信息，直到遇到文件结束标志为止。

(二) 语法分析器的构造过程

(1) 构造语法分析器的目的，主要为了分析语言的结构信息。

① 为结构正确的输入构造语法树 ② 检查输入序列中的错误。

(2) 要完成的主要工作：

① 为递归下降分析设计相应函数给图语言的文法

② 设计语法树的结点，用于存储表达式的语法树

③ 设计递归下降子程序，分析句子并构造表达式的语法树

④ 设计测试程序和用例，验证词法表达式的正确性

(3) 分析：

设计函数给图语言的文法：

Program → ε | Program Statement SEMICO

Statement → OriginStatement | ScaleStatement | RotStatment | ForStatment | ColorStatment

OriginStatement → ORIGIN IS L_BRACKET Expression COMMA Expression R_BRACKET

ScaleStatment → SCALE IS L_BRACKET Expression COMMA Expression R_BRACKET

RotStatment → ROT IS Expression ColorStatement → COLOR IS Expression

ForStatement → FOR T From Expression To Expression STEP Expression DRAW
L_BRACKET Expression COMMA Expression R_BRACKET

Expression → Expression PLUS Expression | Expression MINUS Expression |
Expression MUL Expression | Expression DIV Expression | PLUS Expression |
MINUS Expression | Expression POWER Expression | CONST_ID | T | RED | GREEN | BLUE |
FUNC L_BRACKET Expression R_BRACKET | L_BRACKET Expression R_BRACKET

3

可见，上述的文法具有特点：二义、左递归、左因子。

接下来即是要消除这些特点，最终完整的EBNF文法：

Program → { Statement SEMICO }　　ColorStatement → COLOR IS Expression

Statement → OriginStatement | ScaleStatement | RotStatement | ForStatement | ColorStatement

OriginStatement → ORIGIN IS L_BRACKET Expression COMMA Expression R_BRACKET

ScaleStatement → SCALE IS L_BRACKET Expression COMMA Expression R_BRACKET

RotStatement → ROT IS Expression.　　Expression → Term { (PLUS | MINUS) Term }

Term → Factor { (MUL | DIV) Factor }.　　Factor → (PLUS | MINUS) Factor | Component

Component → Atom [POWER Component].　　Atom → CONST_ID | T | FUNC L_BRACKET Expression R_BRACKET | L_BRACKET Expression R_BRACKET | RED | BLUE | GREEN | PINK

构造表达式的语法树：

结点可分为以下三类：

① 叶子结点：常数、参数T等

② 两个孩子的内部结点：二元运算如MUL、PLUS等

③ 一个孩子的内部结点：函数调用

由结点类型可以确定结点的数据结构

```
public class ExprNode
{
    private Token token;
    private ExprNode left;
    private ExprNode right;

    public ExprNode() { token = null;  left = null;  right = null; }
    public ExprNode(Token token, ExprNode left, ExprNode right) { this.token = token;
                    this.left = left;  this.right = right; }
    // 各成员变量的相应 setter 和 getter 方法
    ... ...
}
```

在确定文法和结点的数据结构之后，可以编写生成表达式树的函数。

构造语法分析器的递归下降子程序：

```
private ExprNode atom();  private ExprNode component();  private ExprNode factor();
private ExprNode term();  private ExprNode expression();
public void program();  private void statement();  protected void originStatement();
protected void scaleStatement();  protected void rotStatement();  protected void forStatement();
protected void colorStatement();
```

4

语法分析器程序的设计思路：利用词法分析器从源程序文件中每次读取一个记号，如果是合法记号，则利用递归下降子程序分析一句完整的程序的结构是否是符合语法的，如果是合法的，利用之前构造的生成语法树的函数，将源文件中的语句以语法树的形式打印出来。如果语句结构不合法，则打印一条错误的提示信息。

测试程序，需测试的内容包括：① 重要的语言结构，各类语句，各种算术表达式 ② 典型的语法错误：语句错误，算术表达式错误

主程序中通过调用 SyntaxAnalysis.program() 函数完成语法分析，其中使用 printSyntaxTree() 函数，将语句对应的语法树打印出来。

(三) 语义分析器的构造

(1) 语义分析器的功能：根据语言结构，处理函数绘图语言程序的语义

构造的过程主要采用语法制导翻译，具体步骤：

① 为语法符号设计属性

② 设计语义规则所需的辅助函数 Draw()

③ 产生设计语义规则

绘图语言的语义：① 表达式值的计算：深度优先后序遍历语法树

② 图形的绘制：循环绘点

绘图所需的语义处理：① 从 origin、rot、scale 和 color 中得到坐标变换所需的

② for...draw 语句根据 t 的每一个值进行如下处理：计算点的横、纵坐标值，

根据坐标变换信息进行坐标变换，得到实际坐标，进行绘点

(2) 分析：

这种函数绘图语言大概只有七种形式的语句。

对于每种语句，我们需要：① 设计属性 ② 设计计算表达式值的辅助函数

③ 设计语义规则

设计语义函数：① 全程变量 private double xScale=1, yScale=1;

private double originX=0, originY=0;

private double step, start, end, rot=0; private ExprNode t;

② 辅助语义函数：a) 计算表达式的值 getExprValue();

b) 计算点的坐标值 setStepX(ExprNode stepX); .....

private int calcX();

private int calcY();

c) 绘制点 public void paint(Graphics g);

其中含循环绘制函数

5

递归子程序中语义规则的嵌入：语义规则可以嵌入递归子程序的任何位置。根据语法制导翻译的基本思想，如果希望从一个语法结构中获取语义，则相应的语义规则可以紧跟在该结构的语法分析之后。a) OriginStatement b) ForStatement 按照规定的语法判断语句中记号出现的顺序是否正确，如该次匹配 ORIGIN、IS、L_BRACKET 记号，说明该语句的语义是正确的。

四、心得体会

(一) 引入"设置图形颜色"功能

功能要求：源程序中写入如"color is red"语句，则图形绘制以"红色"的点描制。

实现过程：1) 于 public enum TokenType

```
{ ……
    COLOR, RED, BLUE, PINK, GREEN, ...
  ……
} 加入与颜色相关的保留字
```

2) 于 public class LexicalAnalysis { } 中，加入

```
public static Token tokenString[] = {
  …… ……
new Token (TokenType.COLOR, "COLOR", 0.0, null),
new Token (TokenType.GREEN, "GREEN", 0.0, null),
new Token (TokenType.RED, "RED", 0.0, null),
new Token (TokenType.BLUE, "BLUE", 0.0, null), ……
  …… ……
}
```

3) 在 public class SyntaxAnalysis { } 中，递归下降子程序中加入 colorStatement(); 以及在 private ExprNode atom() { } 中加入颜色相关判断

4) 在 public class Interpreter extends SyntaxAnalysis { } 中，加入 protected void ~~Stat~~ colorStatement() 方法重写

5) 在 public class Draw extends Panel { } 中，加入 public void setColor(ExprNode color); 以及绘图时相应语句。

5

(二) 不足

代码可维护性不强，当要给解释器添加新的功能时，需要大量改动代码，使解释器具有一定的局限性。当前解释器的功能使用了 LL(1) 文法，可以简单将绘图语句执行下去，比较简单。要扩展的方面比较有限，只能在显示等方面进行优化。