《编译原理 》上机实验（1）

词法分析器

一、上机实验的目的

通过做上机题加深对编译器构造原理和方法的理解，巩固所学知识。

<1> 会用正规式和产生式设计简单语言的语法；

<2> 会用递归下降子程序编写编译器或解释器；

<3> 会写上机报告。

二、上机题目－简单函数绘图语言的解释器

2.1 题目简述

<1> 实现简单函数绘图的语句

- 循环绘图 (FOR-DRAW)
- 比例设置 (SCALE)
- 角度旋转 (ROT)
- 坐标平移 (ORIGIN)
- 注释 (-- 或 //)

<2> 屏幕（窗口）的坐标系

- 左上角为原点
- x方向从左向右增长
- y方向从上到下增长(与一般的坐标系方向相反)

〈3〉 函数绘图源程序举例

```
---------------- 函数f(t)=t的图形
origin is (100, 300);    -- 设置原点的偏移量
rot is 0;                -- 设置旋转角度(不旋转)
scale is (1, 1);         -- 设置横坐标和纵坐标的比例
for T from 0 to 200 step 1 draw (t, 0);
                         -- 横坐标的轨迹 (纵坐标为0)
for T from 0 to 150 step 1 draw (0, -t);
                         -- 纵坐标的轨迹 (横坐标为0)
for T from 0 to 120 step 1 draw (t, -t);
                         -- 函数f(t)=t的轨迹
```

默认值：

```
origin is (0, 0)
rot is 0;
scale is (1, 1)
```

2.2 语句的语法和语义（syntax & semantics）

语句满足下述规定(原则)：

〈1〉各类语句可以按任意次序书写，且语句以分号结尾。源程序中的语句以它们出现的先后顺序处理。

〈2〉ORIGIN、ROT和SCALE 语句只影响其后的绘图语句，且遵循最后出现的语句有效的原则。例如，若有下述ROT语句序列：

```
ROT IS 0.7 ;
ROT IS 1.57 ;
```

则随后的绘图语句将按1.57而不是0.7弧度旋转。

〈3〉无论ORIGIN、ROT和SCALE语句的出现顺序如何，图形的变换顺序总是：比例变换→旋转变换→平移变换

〈4〉语言对大小写不敏感，例如for、For、FOR等，均被认为是同一个保留字。

〈5〉语句中表达式的值均为双精度类型，旋转角度单位为弧度且为逆时针旋转，平移单位为点。

2.2.1 循环绘图（FOR-DRAW ）语句

语法：

FOR T FROM 起点 TO 终点 STEP 步长 DRAW(横坐标, 纵坐标);

语义：

令T从起点到终点、每次改变一个步长，绘制出由(横坐标，纵坐标)所规定的点的轨迹。

举例：

FOR T FROM 0 TO 2*PI STEP PI/50 DRAW (cos(T), sin(T));

说明：

该语句的作用是令T从0到2*PI、步长 PI/50，绘制出各个点的坐标(cos(T)，sin(T))，即一个单位园。

注意： 由于绘图系统的默认值是

```
ORIGIN IS (0,0);
ROT IS 0;
SCALE IS (1, 1);
```

所以实际绘制出的图形是在屏幕左上角的一个点。

2.2.2 比例设置(SCALE)语句

语法：SCALE IS (横坐标比例因子，纵坐标比例因子)；

语义：设置横坐标和纵坐标的比例，并分别按照比例因子进行缩放。

举例：SCALE IS (100, 100)；

说明：将横坐标和纵坐标的比例设置为1:1，且放大100倍。

若：SCALE IS (100, 100/3)；

则：横坐标和纵坐标的比例为3:1。

2.2.3 坐标平移(ORIGIN)语句

语法：ORIGIN IS (横坐标，纵坐标)；

语义：将坐标系的原点平移到横坐标和纵坐标规定的点处。

举例：ORIGIN IS (360, 240)；

说明：将原点从(0, 0)平移到(360, 240)处。

2.2.4 角度旋转(ROT)语句

语法： ROT IS 角度;

语义： 逆时针旋转角度所规定的弧度值。具体计算公式：

旋转后X=旋转前X*COS(角度)+旋转前Y*SIN(角度)

旋转后Y=旋转前Y*COS(角度)-旋转前X*SIN(角度)

公式的推导可参阅辅助教材。

举例： ROT IS PI/2;

说明： 逆时针旋转PI/2，即逆时针旋转90度。

2.2.5 注释语句

注释的作用： 便于理解；

屏蔽暂时不需要的语句。

语法： // This is a comment line

或 -- 此行是注释

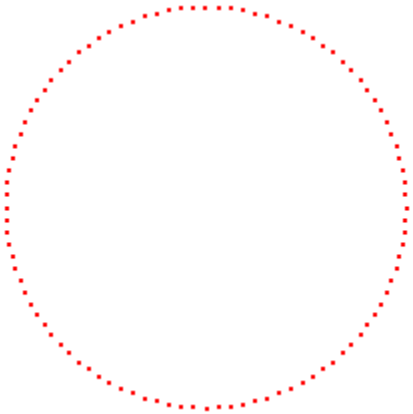
语义： // 或 -- 之后，直到行尾，均是注释

语句功能的测试

```
ORIGIN IS (360, 120);     // (1) 原点移至(360, 120)
SCALE IS (100, 100);      // (2) 图形放大100
SCALE IS (100, 100/3);    // (3) 纵坐标缩小为三分之一
ROT IS PI/2;              // (4) 逆时针旋转90度
```

-- 绘制园的轨迹

```
FOR T FROM 0 TO 2*PI STEP PI/50 DRAW (cos(T), sin(T));
```

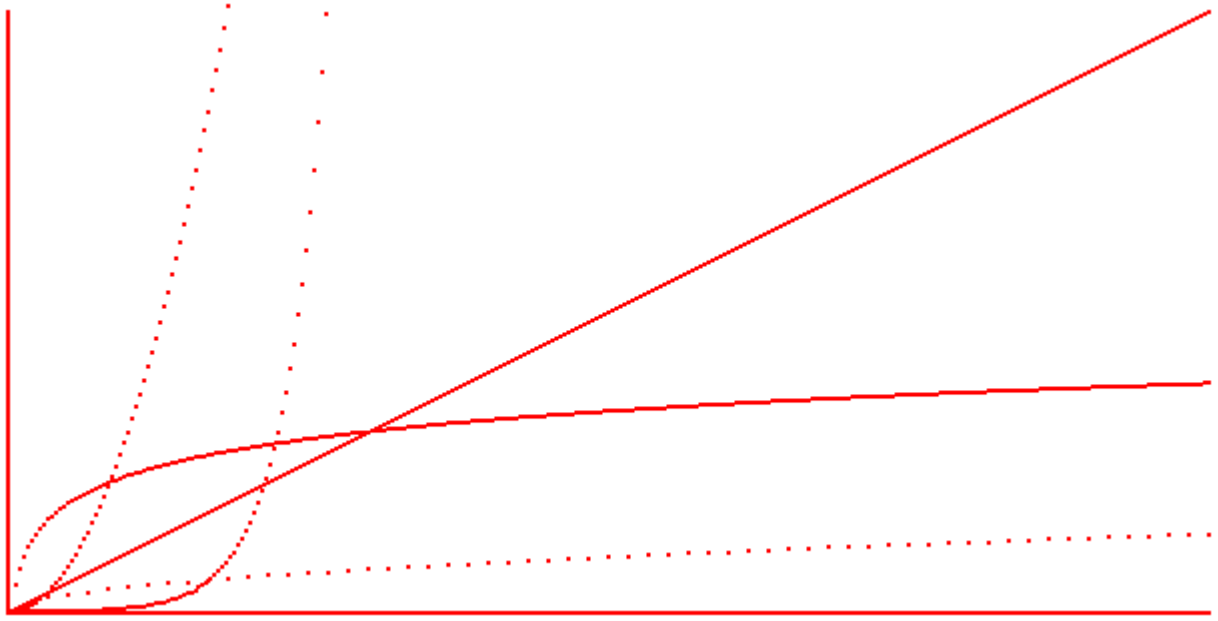


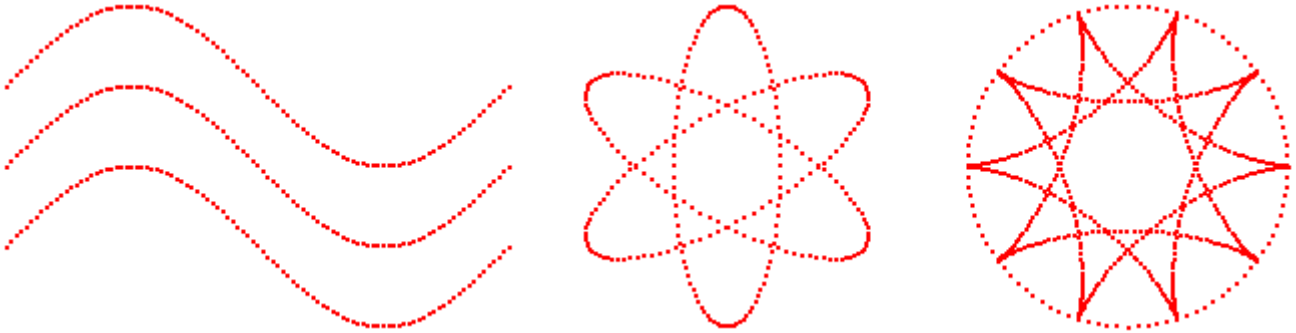
仅(1)和(2)

加入(3)

加入(4)

其他函数图形：





看实例

2.3 记号的语法和语义

记号的种类： 常数、参数、函数、保留字、运算符、分隔符

〈1〉 **常数**

常数字面量和标识符形式的**常量名**均称为常数。字面量的形式为普通的数值，如果没有小数部分，可以省略小数点。例如2、2.、2.0都是合法的常数。 标识符PI、E也是常数，它们分别代表圆周率和自然对数的底。常数不能有符号位，如-1和+2不是常数而是（一元运算的）表达式。

〈2〉 **参数**

本作图语言中唯一的、已经被定义好的**变量名T被称为参数**，它也是一个表达式。由于作图语言中只有这唯一的变量，因此作图语言中无需变量或参数的声明和定义语句。

〈3〉 函数（调用）

为简单起见，当前函数调用仅支持Sin、Cos、Tan、Sqrt以及Exp和Ln。有兴趣的同学可以再加入其他函数。

〈4〉 保留字：语句中具有固定含义的标识符，包括：

ORIGIN, SCALE, ROT, IS, TO,
STEP, DRAW, FOR, FROM

〈5〉 运算符

PLUS, MINUS, MUL, DIV, POWER
即： + - * / **

〈6〉 分隔符

SEMICO, L_BRACKET, R_BRACKET, COMMA
即： ; ( ) ,

三、 题目与要求

题目：为函数绘图语言编写一个解释器

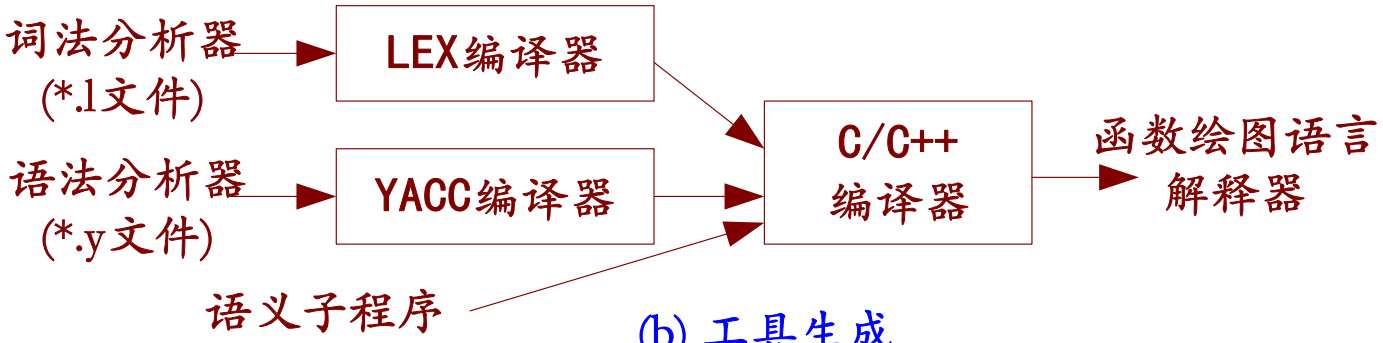
解释器接受用绘图语言编写的源程序，经语法和语义分析之后，将源程序所规定的图形显示在显示屏（或窗口）中。

目的：通过自己动手编写解释器，掌握语言翻译特别是语言识别的基本方法。

3.1 解释器的实现方法

1. 用某种程序设计语言（如C/C++、Pascal、Java等）和递归下降子程序方法编写完整的解释器，由于环境限制，本书统一采用C/C++程序设计语言；
2. 利用编译器编写工具LEX/YACC提供的方式规定绘图语言的词法和语法，用C/C++语言编写解释器的语义。

(a) 手工编写



(b) 工具生成

(c) 解释器工作原理

两种方法的语义部分基本相同，主要区别在于词法和语法分析器的构造是手工完成还是借助于工具完成。

3.3 任务划分与上机报告

任务划分：（三个阶段）

词法分析器、语法分析器、语义分析器

机时比例（大概）： 1:1:1

要求： 验收经过测试的程序

提交上机报告。其中上机报告可以包括以下内容：

〈1〉 任务与目的

〈2〉 软件设计

a. 软件的总体结构与模块划分

b. 关键算法与重要数据结构

〈3〉 测试例程设计与测试结果分析

〈4〉 总结、体会、改进建议等

工作方法建议： 每个阶段均进行设计与测试，并且写出报告；采用增量式设计；工作全部完成后将三个阶段的工作进行总结即可。

四、递归子程序方法的参考解决方案

4.1 词法分析器的构造

步骤：正规式－NFA－DFA－最小DFA－编写程序－测试

4.1.1 记号的设计

〈1〉词法分析器的三个任务：

1. 滤掉源程序中的无用成分；
2. 输出记号供语法分析器使用；
3. 识别非法输入，并将其标记为“出错记号”

。

〈2〉记号的组成：记号的类别和属性。

〈3〉记号的数据结构：

```
struct Token                    // 记号的数据结构
{   Token_Type  type;           // 类别
    char * lexeme;              // 属性，原始输入的字符串
    double value;               // 属性，若记号是常数则是常数的值
    double (* FuncPtr)(double);
                                // 属性，若记号是函数则是函数指针
};
```

例2.2

〈4〉 函数绘图语言中记号的分类与表示

```
enum Token_Type                                  // 记号的类别，共22个
{     ORIGIN, SCALE, ROT, IS,                    // 保留字（一字一码）
      TO, STEP, DRAW,FOR, FROM,                  // 保留字
      T,                                         // 参数
      SEMICO, L_BRACKET, R_BRACKET, COMMA,       // 分隔符
      PLUS, MINUS, MUL, DIV, POWER,              // 运算符
      FUNC,                                      // 函数（调用）
      CONST_ID,                                  // 常数
      NONTOKEN,                                  // 空记号（源程序结束）
      ERRTOKEN                                   // 出错记号（非法输入）
};
```

下一页 关系

4.1.2 模式的正规式表示

```
letter = [a-zA-Z]
digit  = [0-9]

COMMENT       = "//"|"--"
WHITE_SPACE   = (" "|\t|\n)+
SEMICO        = ";"
L_BRACKET     = "("
R_BRACKET     = ")"
COMMA         = ","
PLUS          = "+"
MINUS         = "-"
MUL           = "*"
DIV           = "/"
POWER         = "**"
CONST_ID      = digit+("." digit*)?
ID            = letter+
```

（去除注释与白空有11个正规式）

由于是手工构造词法分析器，而正规式个数越少越便于程序的编写，因此设计上采用相同模式的记号共用一个正规式的方法。

常数的字面量部分设计为CONST_ID，而常量名则合并到ID中。

这就带来一个问题，函数绘图语言中的保留字、常量名、参数名、以及函数名均被描述为ID，当识别出ID时，如何再细分它们？

4.1.3 区分记号的符号表

预先定义且内容不变的符号表更多被习惯地称为字典。

```
static Token TokenTab[] =
{       {CONST_ID,      "PI",           3.1415926,      NULL},
        {CONST_ID,      "E",            2.71828,        NULL},
        {T,             "T",            0.0,            NULL},
        {FUNC,          "SIN",          0.0,            sin},
        {FUNC,          "COS",          0.0,            cos},
        {FUNC,          "TAN",          0.0,            tan},
        {FUNC,          "LN",           0.0,            log},
        {FUNC,          "EXP",          0.0,            exp},
        {FUNC,          "SQRT",         0.0,            sqrt},
        {ORIGIN,        "ORIGIN",       0.0,            NULL},
        {SCALE,         "SCALE",        0.0,            NULL},
        {ROT,           "ROT",          0.0,            NULL},
        {IS,            "IS",           0.0,            NULL},
        {FOR,           "FOR",          0.0,            NULL},
        {FROM,          "FROM",         0.0,            NULL},
        {TO,            "TO",           0.0,            NULL},
        {STEP,          "STEP",         0.0,            NULL},
        {DRAW,          "DRAW",         0.0,            NULL}
};
```

关系

三者之间的关系

1. 记号类别（enum Token_Type）：作用是规定所有记号的类别；
2. 正规式：作用是描述所有类别的记号；
3. 字典：作用是区分同一模式下的不同记号；

当输入被匹配为ID时，首先去查字典，从字典中得到此ID所代表的记号，然后返回记号信息给语法分析器。

内部识别的模式与外部返回的记号之间不是一一对应的，可以根据实际情况设计，原则是能够正确识别和返回所有记号，且便于程序实现。

ROT IS PI/6

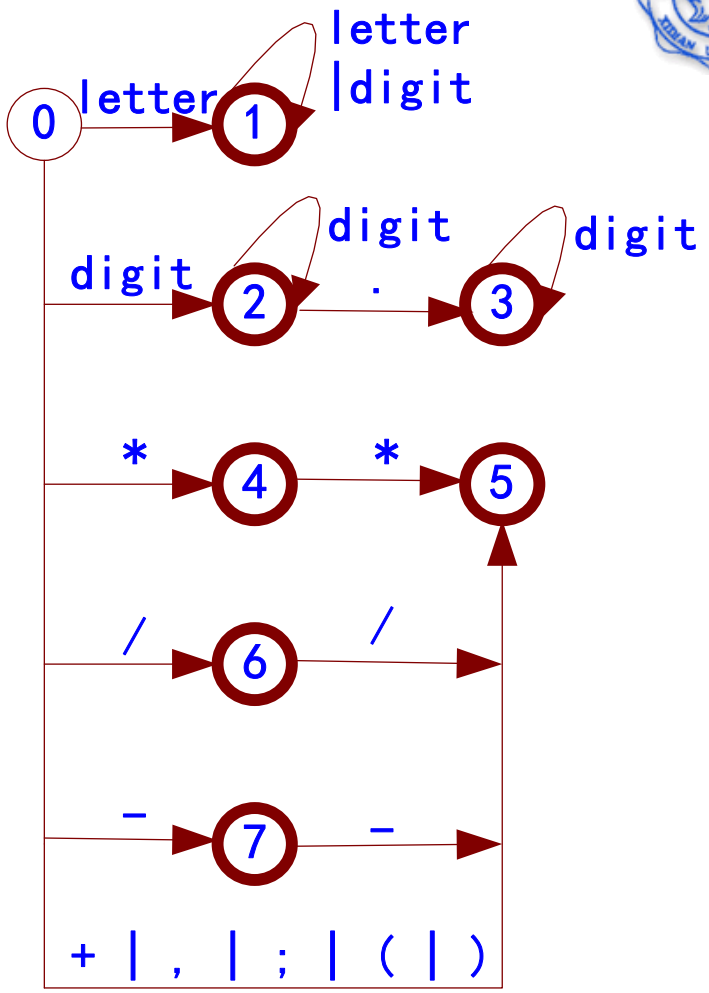
例2.2 语句 ROT IS PI/6 的记号流

<类别	原始输入	值	函数地址>
<ROT	"ROT"	0.0	NULL>
<IS	"IS"	0.0	NULL>
<CONST_ID	"PI"	3.141593	NULL>
<DIV	"/"	0.0	NULL>
<CONST_ID	"6"	6.0	NULL>

存放记号的数据结构

4.1.4 正规式的DFA

```
letter = [a-zA-Z]
digit  = [0-9]
ID           = letter+
CONST_ID     = digit+("." digit*)?
POWER        = "**"
COMMENT      = "//"|"--"
SEMICO       = ";"
L_BRACKET    = "("
R_BRACKET    = ")"
COMMA        = ","
PLUS         = "+"
MINUS        = "-"
MUL          = "*"
DIV          = "/ “
WHITE_SPACE = (" "|\t|\n)+
```



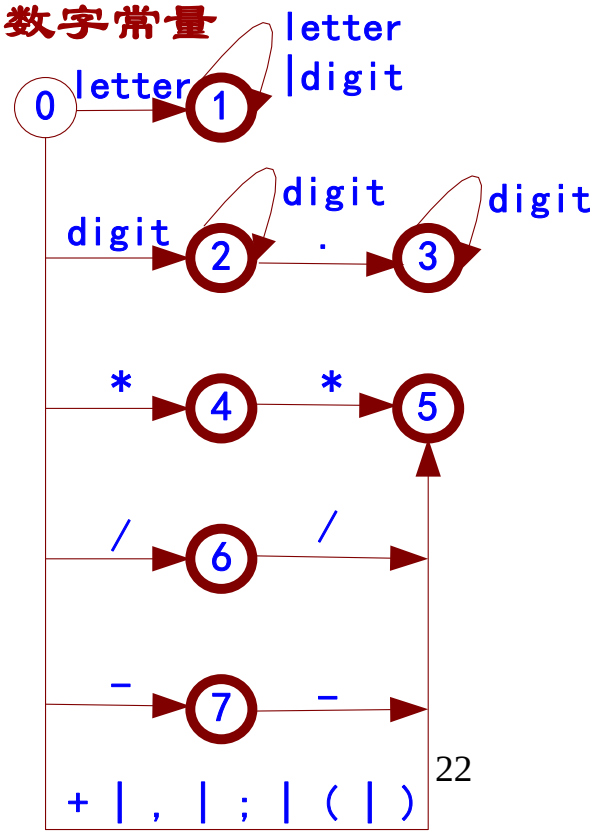
注意：WHITE_SPACE（白空）没有在DFA中。

如何处理白空？

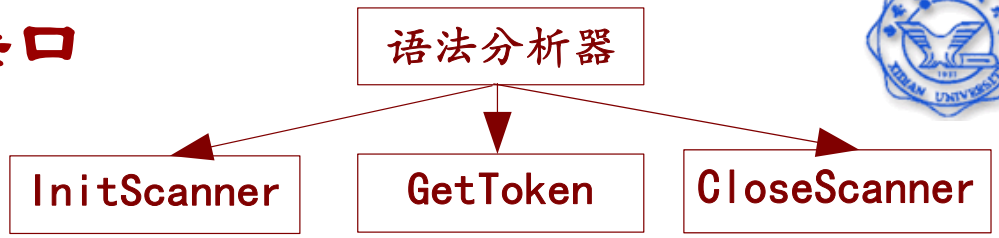
4.1.5 词法分析器的程序框架

```
struct Token token={ERRTOKEN, " ", 0.0, NULL}; // 用于返回记号
token.lexeme = TokenBuffer;  // 记号的字符指针指向字符缓冲区
char = GetChar();            // 从源文件中读取一个字符
……                           // 空格、TAB、回车等字符的过滤
AddInTokenString (char);     // 将读入的字符放进缓冲区TokenBuffer中
if  (isalpha(char))  { …… }      //  识别ID
  else if  (isdigit(char))  { …… }  // 识别数字常量
  else
  {   switch(char)
    {  case   ';' : token.type = SEMICO;
                    return token;
       ……
    }
  }
```

(其中的辅助函数和变量看教材)



4.1.6 与语法分析器的接口



〈1〉词法分析器的测试

```
#include "scanner.h"
void main(int argc, char *argv[])
{  Token token;
   if (argc<2) { printf("please input Source File !\n" ); return;}
   if (!InitScanner(argv[1]))                    // 初始化词法分析器
      { printf("Open Source File Error ! \n"); return; }
   printf("记号类别    字符串       常数值       函数指针\n");
   printf("_____________________________________________\n");
   while(1)
   { token = GetToken();          // 通过词法分析器获得一个记号
     if(token.type != NONTOKEN) // 打印记号的内容
          printf("%4d,%12s,%12f,%12x\n",
          token.type, token.lexeme, token.value, token.FuncPtr);
     else break;                  // 源程序结束，退出循环
   };
   printf("_____________________________________________\n");
   CloseScanner();                // 关闭词法分析器
}
```

〈2〉 测试实例与测试结果

词法分析器的测试比较简单，可以分为三个部分：

1. 全部合法的输入
2. 各种组合的非法输入
3. 由记号组成的句子

例如源程序：

```
FOR T FROM 0 TO 2*PI STEP PI/50 DRAW (cos(T),sin(T));
```

测试结果：(看运行结果)

课程及第一次上机要点

1. 分析研究函数绘图语言，深刻理解每条语句的语法和语义；深刻理解记号的语法和语义；
2. 分析上机题目与要求，思考词法分析器的设计要点；
3. 参考课堂上的记号设计与DFA的实现方案，设计并实现词法分析器，并自己设计测试例程，进行详细测试；
4. 写出词法分析器部分的上机报告。

结束（第一次上机课）