《编译原理》上机实验（3）

语法制导翻译绘制函数图形

简单复习

```
---------------- 函数f(t)=t的图形
origin is (200, 300);    -- 设置原点的偏移量
rot is pi/6;              -- 设置旋转角度
scale is (2, 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
for T from 0 to 180 step 1 draw (0, -t);  -- 纵坐标的轨迹
for T from 0 to 150 step 1 draw (t, -t);  -- f(t)=t的轨迹
```

词法分析器：

识别输入序列，并为语法分析器提供记号。

语法分析器：

根据记号流识别句子，并为表达式构造语法树。

语义分析器：

根据语言结构，处理函数绘图语言程序的语义。

2.3.3 语法制导翻译绘制图形

2.3.3.1 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：画出每个坐标点

绘图所需的语义处理：

1. 从origin、rot和scale中得到坐标变换所需的信息；
2. for_draw语句根据t的每一个值进行如下处理：
   - 计算被绘制点的横、纵坐标值；
   - 根据坐标变换信息进行坐标变换，得到实际坐标；
   - 根据点的实际坐标画出该点。

语法制导翻译的基本步骤

- 为文法符号设计属性；
- 设计语义规则中所需的辅助函数；
- 为产生式设计语义规则。

例子：比例设置语句的语义设计

文法：ScaleStatment → SCALE IS

L_BRACKET Expression COMMA Expression R_BRACKET

可简写为：S → SCALE IS (E,E)

它的作用是提供横、纵坐标的比例因子。

因此需要：

1. 设计属性：.x和.y，分别保存比例因子；
2. 设计计算表达式值的辅助函数：

   GetExprValue(nptr)，它返回表达式树的值；
3. 设计语义规则：

   S → SCALE IS (E1,E2)　S.x:=GetExprValue(E1.nptr);

   S.y:=GetExprValue(E2.nptr);

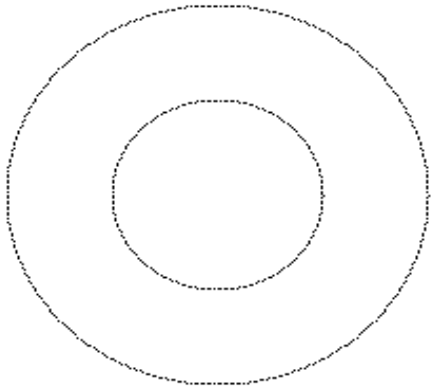
2.3.3.2 语义函数的设计

〈1〉全程变量：

```
double Parameter=0;                          // 为参数T分配的变量
double Origin_x=0.0, Origin_y=0.0;// 用于记录平移距离
double Rot_ang=0.0;                          // 用于记录旋转角度
double Scale_x=1, Scale_y=1;                 // 用于记录比例因子
```

例：以点(350, 220)为圆心绘制两个同心园

```
origin is (350, 220);    // Origin_x=350，Origin_y=220
scale is (50, 50);       // Scale_x=50，Scale_y=50
for t from 0 to 2*pi step pi/100 draw(cos(t), sin(t));
scale is (100, 100);     // Scale_x=100，Scale_y=100
for t from 0 to 2*pi step pi/200 draw(cos(t), sin(t));
```



〈2〉辅助语义函数

a) 计算表达式的值：深度优先后序遍历语法树

```
double GetExprValue(struct ExprNode * root);
```

b) 计算点的坐标值：首先获取坐标值，然后进行坐标变换

```
static void CalcCoord( struct ExprNode * x_nptr,
                       struct ExprNode * y_nptr,
                       double &x_val,
                       double &y_val);
```

c) 绘制一个点(与环境有关)：

```
void DrawPixel(unsigned long x, unsigned long y);
```

d) 循环绘制所有的点：

```
void DrawLoop(  double Start,
                double End,
                double Step,
                struct ExprNode * HorPtr,
                struct ExprNode * VerPtr);
```

〈3〉 辅助语义函数设计举例

a)

```
typedef double (* FuncPtr)(double);
struct ExprNode
{  enum Token_Type OpCode;    // 记号种类
   union
   {
        struct { ExprNode *Left, *Right;
               } CaseOperator;      // 二元运算
        struct { ExprNode * Child;
                   FuncPtr MathFuncPtr;
               } CaseFunc;    // 函数调用
        double CaseConst;     // 常数，绑定右值
        double * CaseParmPtr; // 参数T，绑定左值
   }Content;
};
```

b) 计算点的坐标值：

```
static void CalcCoord (   struct ExprNode * x_nptr,
                          struct ExprNode * y_nptr,
                          double &x_val,
                          double &y_val )
{   double local_x, local_y, temp;
    local_x=GetExprValue(x_nptr);        // 计算点的原始坐标
    local_y=GetExprValue(y_nptr);
    local_x *= Scale_x;                  // 比例变换
    local_y *= Scale_y;
    temp=local_x*cos(Rot_angle)+local_y*sin(Rot_angle);
    local_y=local_y*cos(Rot_angle)-local_x*sin(Rot_angle);
    local_x = temp;                      // 旋转变换
    local_x += Origin_x;                 // 平移变换
    local_y += Origin_y;
    x_val = local_x;                     // 返回变换后点的坐标
    y_val = local_y;
}
```

d) 点轨迹的循环绘制

```
void DrawLoop(    double Start,
                  double End,
                  double Step,
                  struct ExprNode * x_ptr,
                  struct ExprNode * y_ptr)
{ extern double Parameter;    // 参数T的存储空间
  double x, y;
  for(Parameter=Start; Parameter<=End; Parameter+=Step)
  { CalcCoord(x_ptr, y_ptr, x, y);    // 计算实际坐标
    DrawPixel((unsigned long)x, (unsigned long)y);
                                      // 根据坐标绘制点
  }
}
```

2.3.3.3 递归子程序中语义规则的嵌入

语义规则可以嵌入在递归子程序的任何位置。根据语法制导翻译的基本思想，如果希望从某部分语言结构中获取语义，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

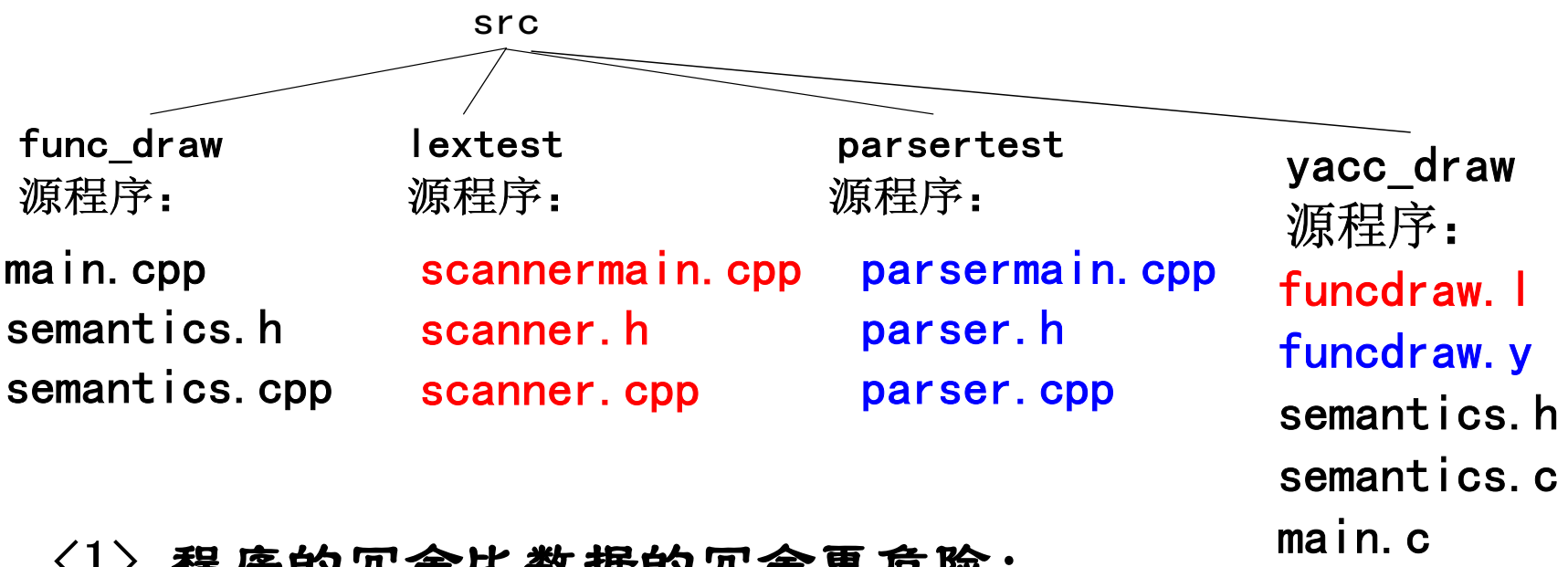
a) OriginStatement

```
static void OriginStatement (void)
{     double x, y;
      struct ExprNode *tmp;

      MatchToken (ORIGIN);
      MatchToken (IS);
      MatchToken (L_BRACKET);
      tmp = Expression();
      x = GetExprValue(tmp);// 获取横坐标的平移值
      MatchToken (COMMA);
      tmp = Expression();
      y = GetExprValue(tmp);// 获取纵坐标的平移值
      MatchToken (R_BRACKET);
      SetOrigin(x, y);        // 置坐标平移全程量
}
```

b) ForStatement

```
static void ForStatement (void)
{ double Start, End, Step;
  struct ExprNode *start_ptr,*end_ptr,*step_ptr,*x_ptr,*y_ptr

  MatchToken (FOR);          MatchToken(T);
  MatchToken (FROM);         start_ptr=Expression();
  Start = GetExprValue(start_ptr);
  MatchToken (TO);           end_ptr=Expression();
  End = GetExprValue(end_ptr);
  MatchToken (STEP);         step_ptr=Expression();
  Step = GetExprValue(step_ptr);
  MatchToken (DRAW);
  MatchToken (L_BRACKET);
  x_ptr = Expression(); MatchToken(COMMA); y_ptr=Expression()
  MatchToken (R_BRACKET);
  DrawLoop (Start, End, Step, x_ptr, y_ptr);
}
```

2.3.3.4 解释器的源程序组织（看实际环境）



〈1〉程序的冗余比数据的冗余更危险；

〈2〉资源的物理位置与逻辑位置无关；

〈3〉合理组织资源，利于团队合作与回归测试；

〈4〉辅助信息同等重要。

End

2.3.3.5 解释器主程序的生成

2.3.3.6 测试例程与测试结果（略）

绘图主程序与词法分析和语法分析器主程序的区别：需要在窗口环境中运行，因此需要建立选项为Win32 Application的主程序。

2.5 上机题的改进建议

2.5.1 函数绘图语言的扩充

〈1〉修改显示屏的直角坐标系，使与习惯上的坐标系一致；

〈2〉扩充语句类型，使得用户可以规定图形颜色；

〈3〉扩展循环绘图语句，使得for_draw语句可以嵌套；

〈4〉增加文本框，使得用户可以在图形中添加文字说明；

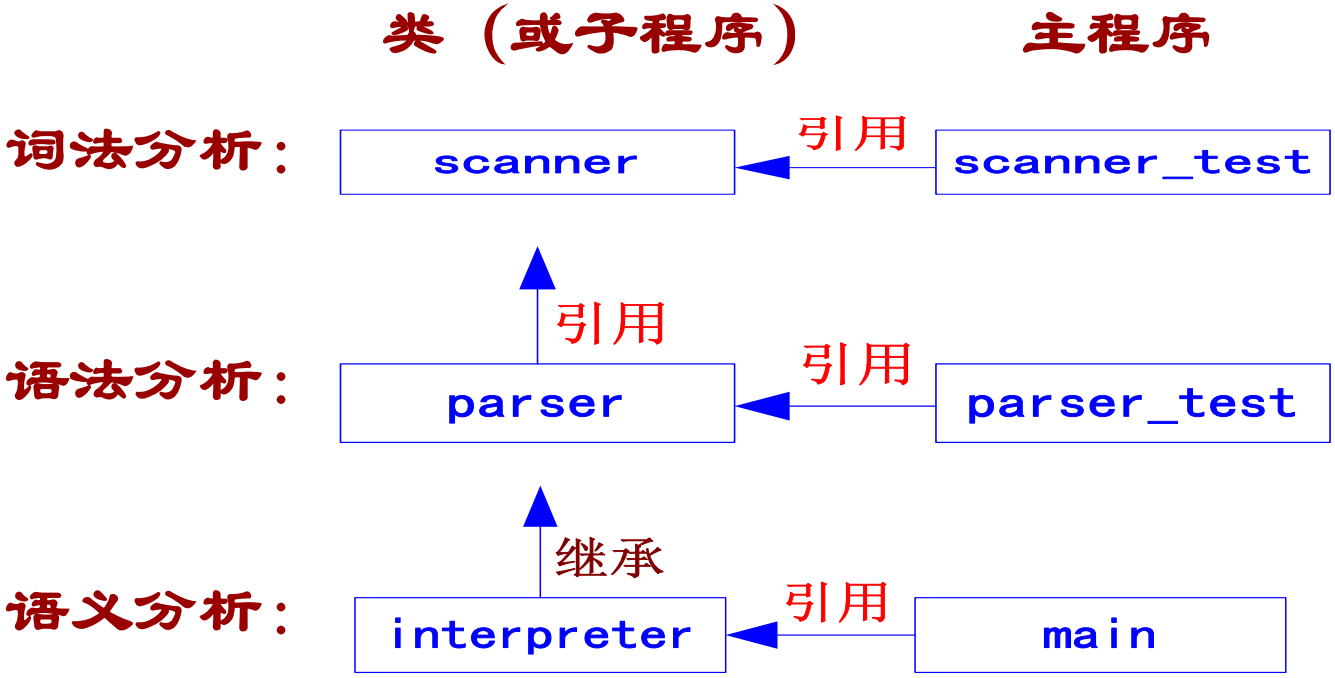
〈5〉增加清图功能，使得图形可以具有简单的动画效果。

2.5.2 语法制导翻译中继承的不同实现方法

语法制导翻译的本质反映在面向对象的程序设计方法上就是一种继承关系。

习惯上：服务提供者（server）－类（子程序）

服务请求者（client）－主程序



用类实现继承

```
tree_node_ptr start_ptr, end_ptr, step_ptr,x_ptr, y_ptr;

//---------------- 基类中的for_statement
void parser_class::for_statement ()
{     match_token (FOR);     match_token(T);
      match_token (FROM);    start_ptr = expression();
      match_token (TO);      end_ptr   = expression();
      match_token (STEP);    step_ptr  = expression();
      match_token (DRAW);
      match_token (L_BRACKET);
      x_ptr = expression();  match_token(COMMA);
      y_ptr = expression();  match_token(R_BRACKET);
}
```

```
//----------------- 派生类中的for_statement
void interpreter_class::for_statement ()
{ double start_val, end_val, step_val;
  parser_class::for_statement();          //语法分析
  start_val = GetExprValue(start_ptr); //语义分析
  end_val   = GetExprValue(end_ptr);
  step_val  = GetExprValue(step_ptr);
  DrawLoop (start_val, end_val, step_val, x_ptr, y_ptr);
}
```

用过程实现继承

```
tree_node_ptr start_ptr,end_ptr,step_ptr,x_ptr, y_ptr;

void for_statement_syntax() // for_statement的语法分析
{ match_token (FOR);          match_token(T);
  match_token (FROM);         start_ptr = expression();
  match_token (TO);           end_ptr   = expression();
  match_token (STEP);         step_ptr  = expression();
  match_token (DRAW);
  match_token (L_BRACKET);
  x_ptr = expression();  match_token(COMMA);
  y_ptr = expression();  match_token (R_BRACKET);
}
```

```
void for_statement_semantics()   // for_stat的语义分析
{ double start_val, end_val, step_val;
  for_statement_syntax();  // 语法分析获取表达式的语法树
  start_val = GetExprValue(start_ptr);
                                // 计算各表达式的值并绘图
  end_val   = GetExprValue(end_ptr);
  step_val  = GetExprValue(step_ptr);
  DrawLoop (start_val, end_val, step_val, x_ptr, y_ptr);
}
```

弱点：程序员负责命名管理

用编译开关实现继承

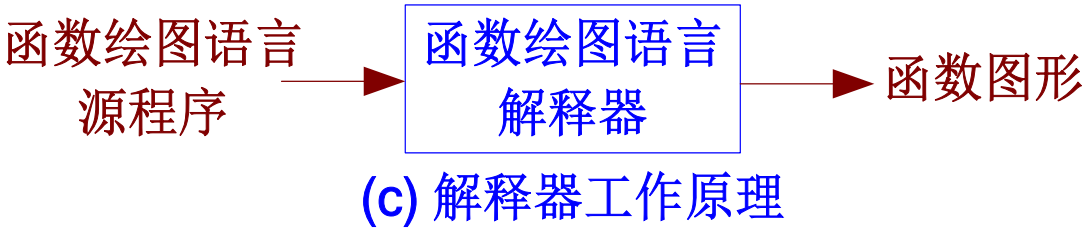
```
static void ForStatement (void)
{  tree_node_ptr start_ptr, end_ptr, step_ptr, x_ptr, y_ptr;
   #ifndef PARSER_DEBUG
        double start_val, end_val, step_val;  //起点、终点、步长
   #endif
   MatchToken (FOR);          MatchToken(T);
   MatchToken (FROM);         start_ptr=Expression();//起点语法树
   MatchToken (TO);           end_ptr=Expression();  //终点语法树
   MatchToken (STEP);         step_ptr=Expression(); //步长语法树
   MatchToken (DRAW);
   MatchToken (L_BRACKET);
   x_ptr = Expression();      MatchToken (COMMA);
   y_ptr = Expression();      MatchToken (R_BRACKET);
   #ifndef PARSER_DEBUG
        start_val = GetExprValue(start_ptr);  //起点值
        end_val   = GetExprValue(end_ptr);    //终点值
        step_val  = GetExprValue(step_ptr);   //步长值
        DrawLoop(start_val,end_val,step_vak,x_ptr, y_ptr);//绘图
   #endif
}
```

体会：

1. 程序设计方法与程序设计语言支持方法无必然联系；
2. C++对C的发展不仅是提供了对面向对象方法的支持，还扩展了常量定义、类属机制、异常处理，等等；
3. 灵活利用C++提供的机制可提高程序的可读性与可维护性；
4. 低层次的语言支持机制，可以给程序员以更大的灵活性，并产生更高效的目标代码；
5. 程序设计方法是通过多看、多做、多想、多比较“悟”出来的，而不是靠听老师灌输“学”来的。

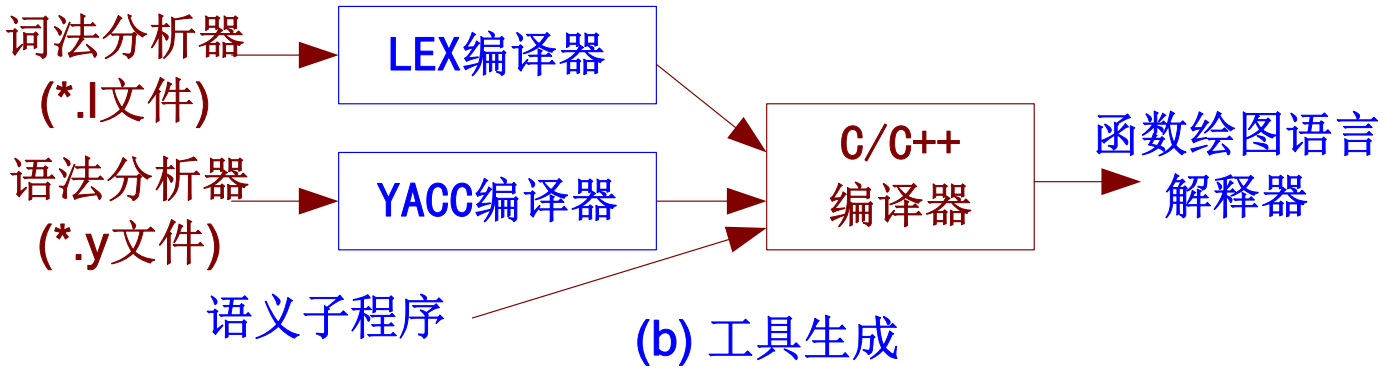
2.5.2 Lex/Yacc编写解释器简介

简单回顾：



(c) 解释器工作原理

(a) 手工编写



(b) 工具生成

关键：如何编写Lex和Yacc源程序

常用的工具：Flex/Bison，Jlex/CUP（自下而上分析）

本课程提供：XDCFLEX/XDYACC

参考资料：

- 上网查找
- Thomas Niemann，"A COMPACT GUIDE TO LEX & YACC"
- 杨作梅 张东旭等译，“Lex与Yacc”

自上而下分析：ANTLR

Lex源程序(*.l)

特点：提供扩展的正规式，擅长描述记号

源程序（三段式）：

[定义（C声明，正规式）]

%%

规则（正规式）

[%%

用户子程序（C源程序）]

实例：（看源程序及程序的运行）

Yacc源程序(*.y)

特点：

- 擅长描述文法，与BNF几乎相同的产生式表示；
- 支持二义文法，提供语义栈（伪变量）；
- 与Lex近似的格式；

源程序（三段式）：

[定义]

%%

规则（产生式）

[%%

用户子程序（C源程序）]

其中，定义包括：C声明，记号、语义栈、优先级与结合性、文法开始符号等的声明。

实例：（看源程序及程序的运行）

发挥你的聪明才智!

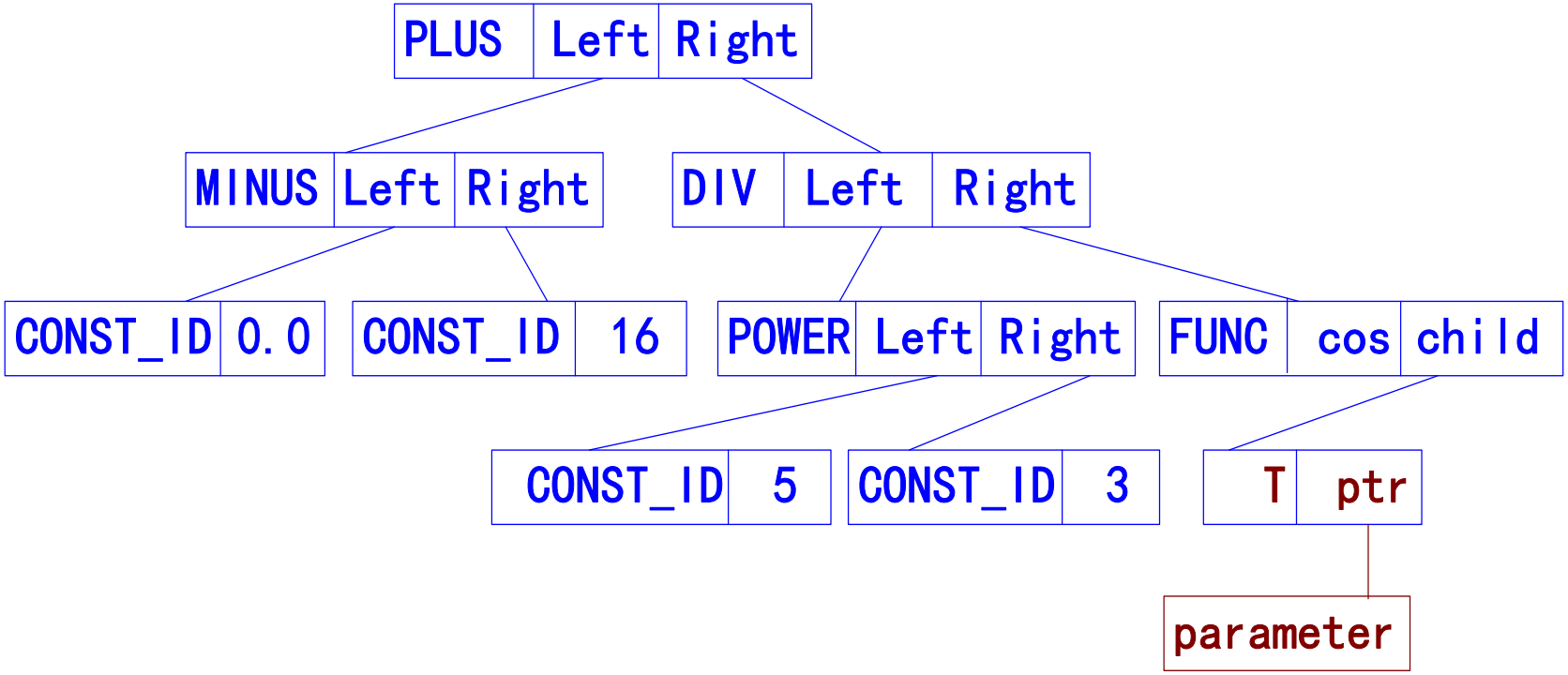
结　束

参考文献

1. Waterloo, Ontario, Canada.Johnson, Stephen C. [1975]. "Yacc: Yet Another Compiler Compiler". Computing Science Technical Report No. 32, Bell Laboratories, Murray hill, New Jersey. A PDF version is available at ePaperPress.
2. Lesk, M. E. and E. Schmidt [1975]. "Lex - A Lexical Analyzer Generator". Computing Science Technical Report No. 39, Bell Laboratories, Murray Hill, New Jersey. A PDF version is available at ePaperPress.
3. Axel T. Schreiner,[1985]."Introduction to Compiler Construction with UNIX". Prentice-Hall,Inc. Englewood Cliffs, NJ 07632
4. Gardner, Jim, Chris Retterath and Eric Gisin [1988]. "MKS Lex & Yacc". Mortice Kern Systems Inc.,
5. Levine, John R., Tony Mason and Doug Brown [1992]. "Lex & Yacc". O'Reilly & Associates, Inc.Sebastopol, California.

表达式值的计算（深度优先后序遍历）

表达式-16+5**3/cos(T)的语法树



返回