第1章　编译概述

知识点：翻译、编译、解释的概念
编译的阶段、任务、及典型结构
编译程序的伙伴工具

编译概述

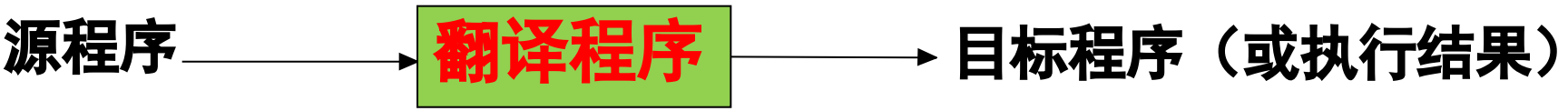
1.1 翻译和解释
1.2 编译的阶段和任务
1.3 编译有关的其他概念
1.4 编译程序的伙伴工具
1.5 编译原理的应用
小结

1.1 翻译和解释

翻译程序、编译程序、汇编程序、解释程序

翻译程序

- **翻译程序**扫描所输入的源程序，并将其转换为目标程序，或将源程序直接翻译成结果。

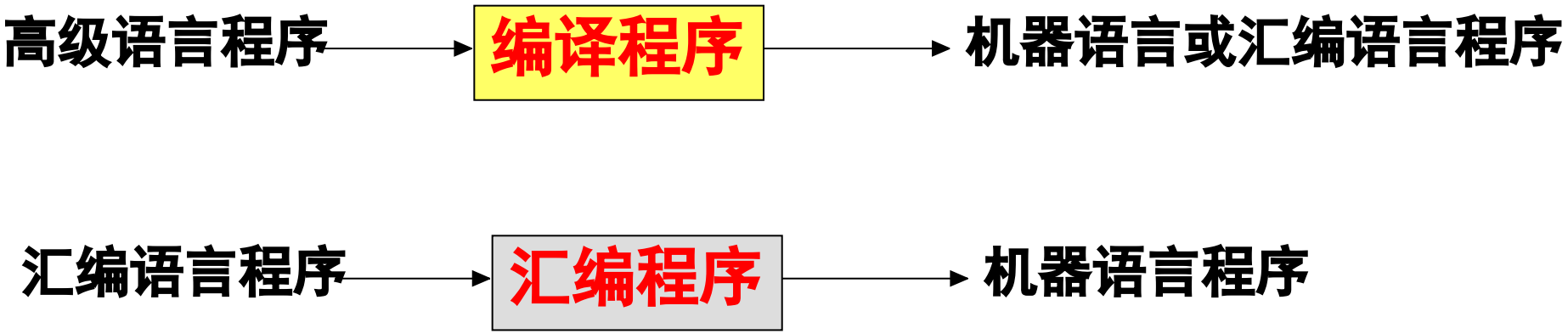


翻译程序的输入是一种泛指的语言编写的源程序，但不一定是高级语言。

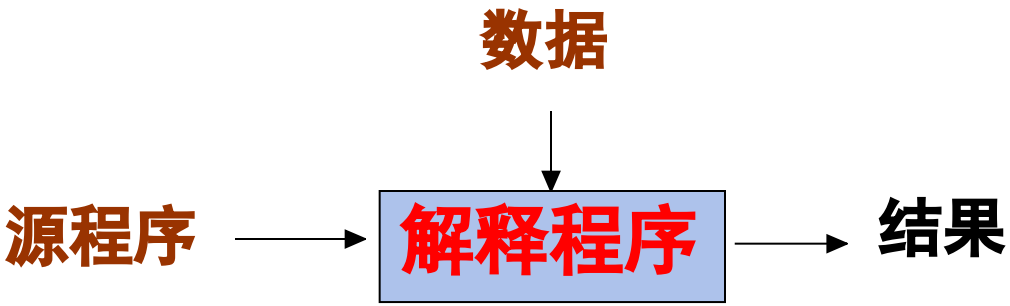
编译程序　　汇编程序

编译程序是翻译程序的一个子集，加上了限制。

- **源程序是用高级语言编写的，目标程序是用目标语言表示的。**



解释程序



- ✓ 解释程序**解释执行**源程序，**不生成目标程序。**
- ✓ 同时处理源程序和数据。

编译程序 &解释程序

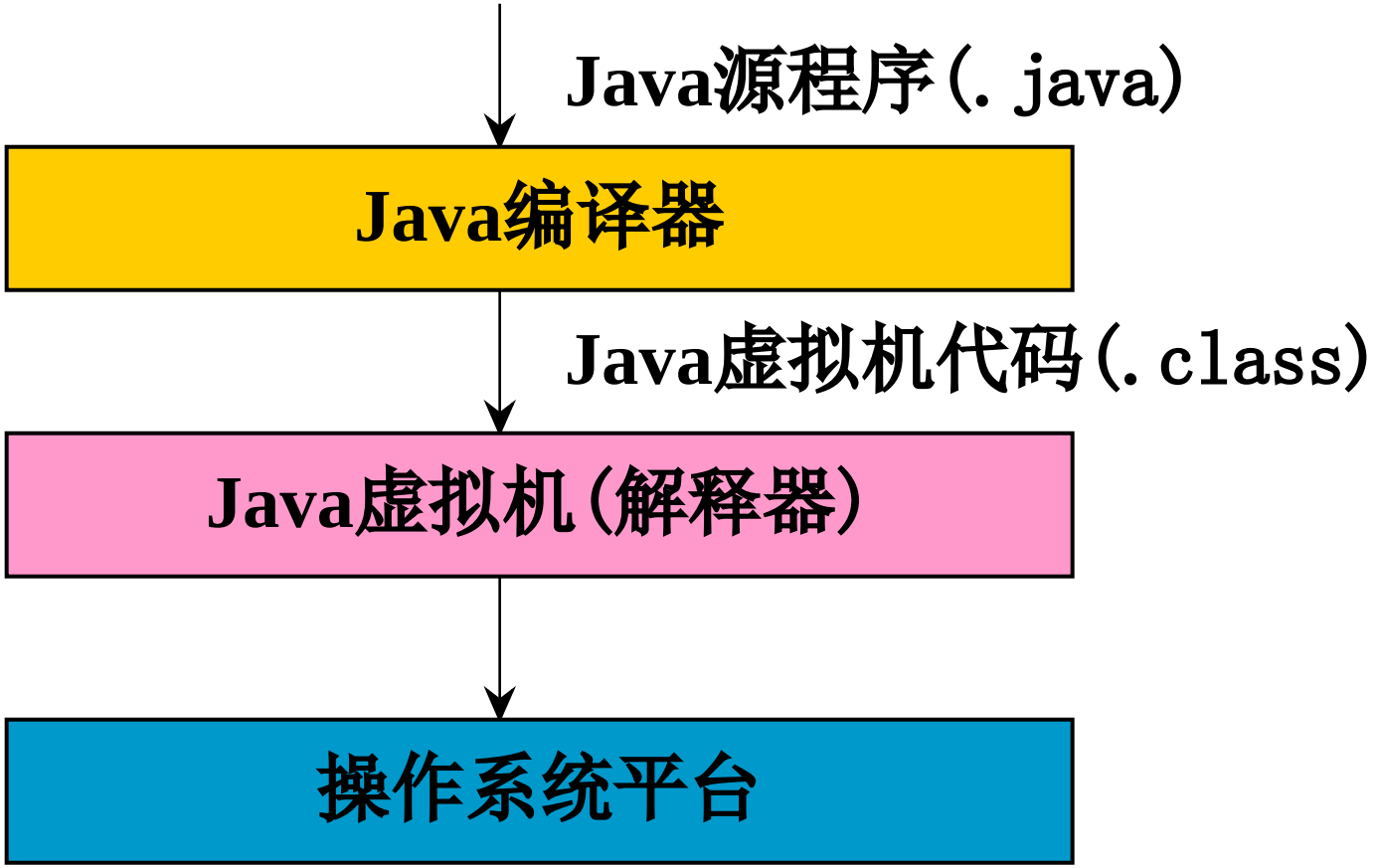
- 编译程序（也称编译器）：将源程序翻译成目标程序的翻译程序。笔译
- 解释程序（也称解释器）：直接执行源程序的翻译程序。口译

解释程序和编译程序的区别

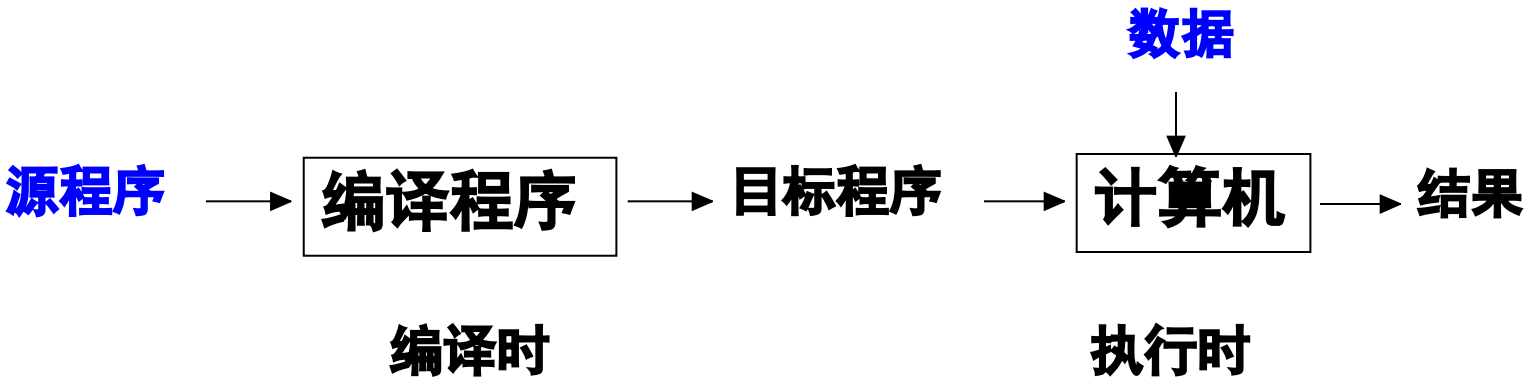
	功能	工作结果	实现技术上
编译程序	源程序的一个转换系统	源程序的目标代码	把中间代码转换成目标程序
解释程序	源程序的一个执行系统	源程序的执行结果	执行中间代码

混合方式

* 另一种有效的方法：先将源程序转换为某种中间形式，然后对中间形式的程序解释执行。**例如：** JAVA语言



编译阶段 & 执行阶段



- 源程序和数据是在不同时间（即分别在编译阶段和运行阶段）进行处理的。
- 编译时间：实现源程序到目标程序的转换所占用的时间。

编译过程

将一篇英文论文翻译为中文的过程：

（1）首先识别出句子中的一个个英文单词；

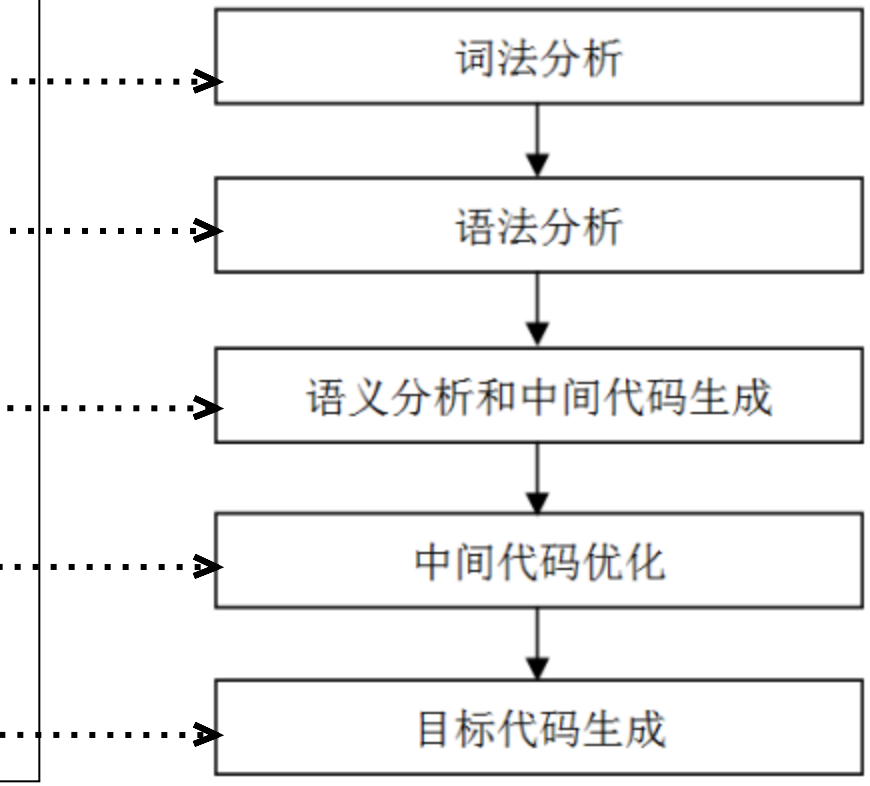
（2）分析句子的语法结构；

（3）根据句子的含义进行初步翻译；

（4）对译文进行修饰；

（5）写出最后的中文译文。

编译过程：

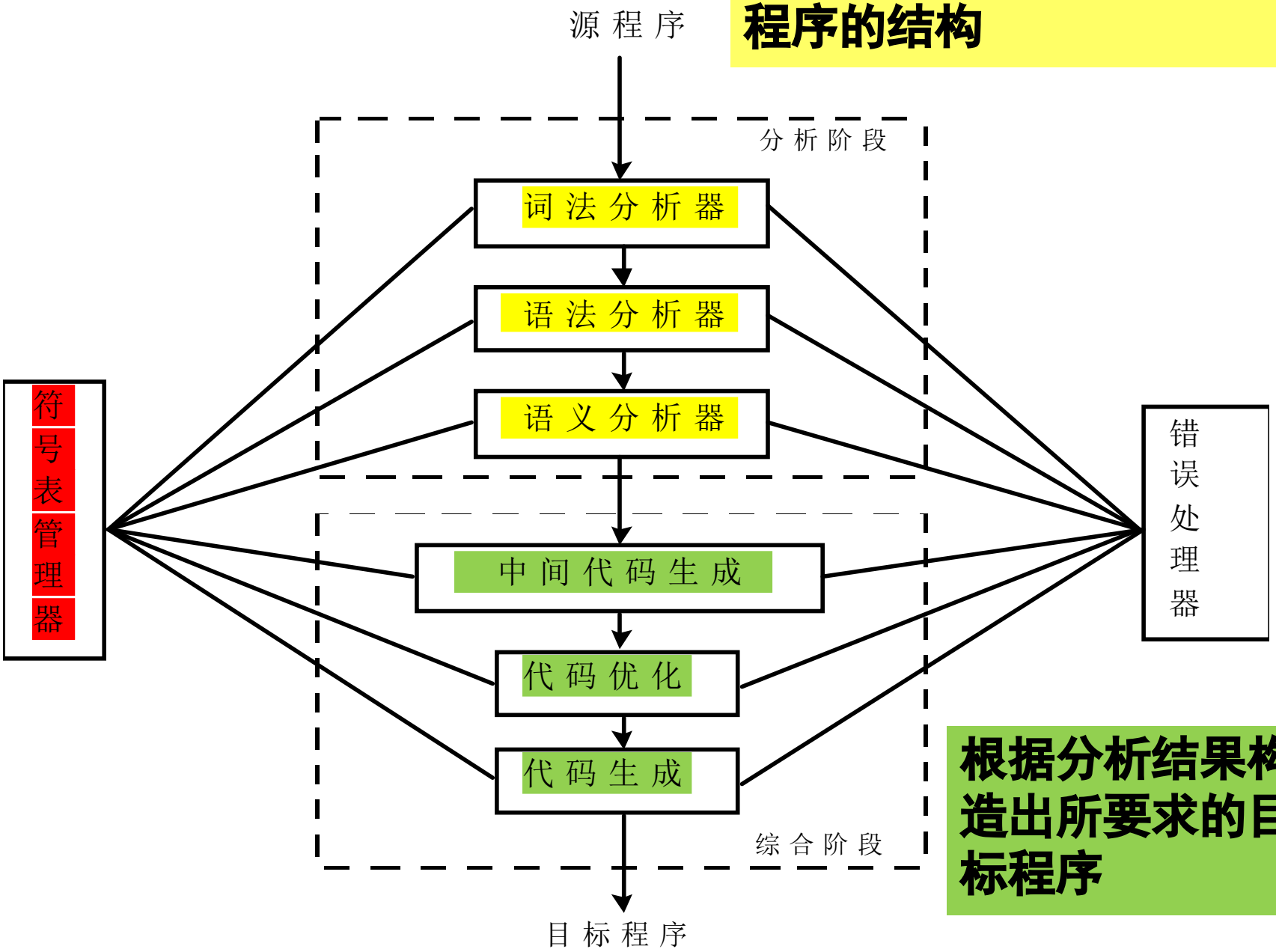


1.2 编译的阶段和任务

一、分析阶段
二、综合阶段
三、符号表的管理
四、错误诊断和处理

编译程序的典型结构

根据源语言的定义，分析源程序的结构



根据分析结果构造出所要求的目标程序

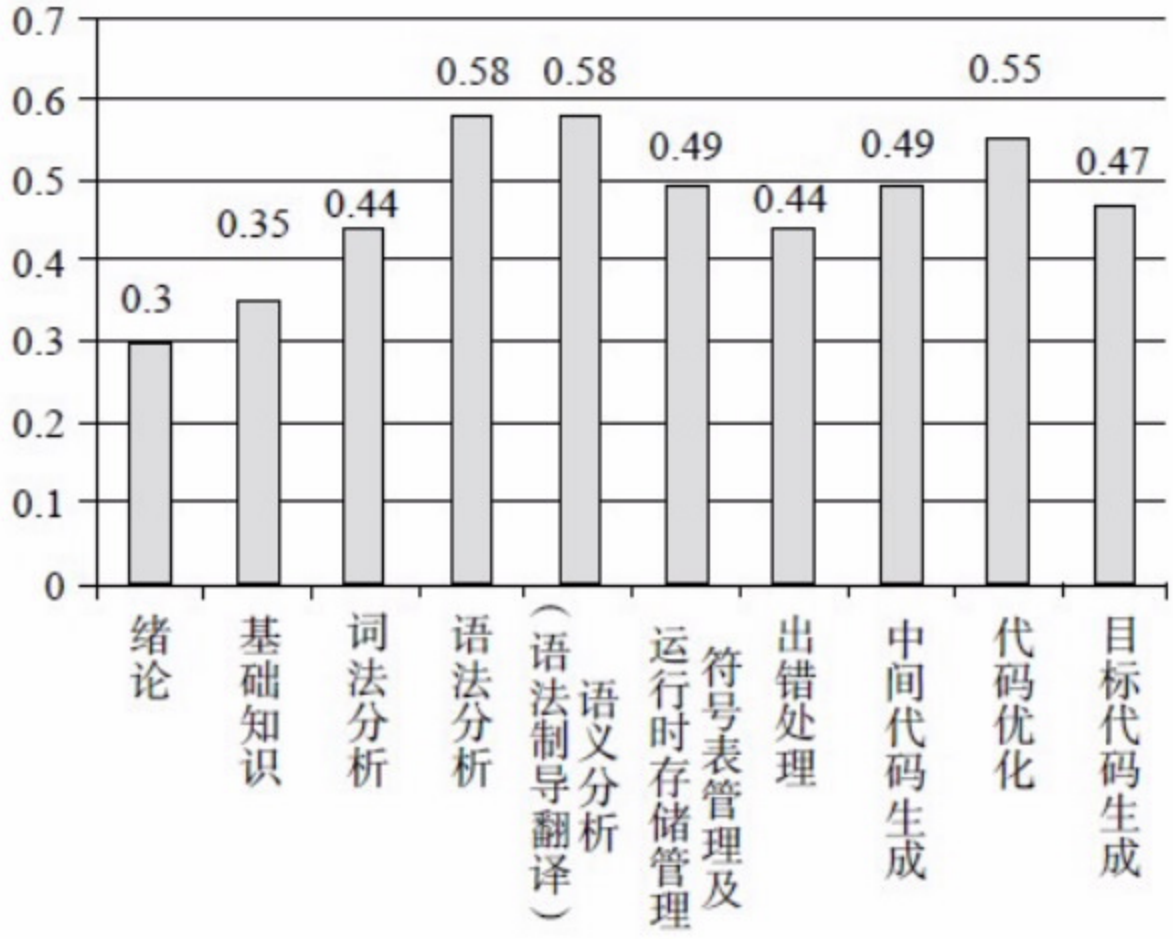


图 1　各章节难度系数示意图

北京航空航天大学软件学院部分已修过该课程的学生进行问卷调查

一、分析阶段

- 对源程序结构进行静态分析。
- 根据源语言的定义，对源程序进行结构分析和语义分析，从而把源程序正文转换为某种内部表示。
- 任务划分：
  1. 词法分析
  2. 语法分析
  3. 语义分析

1. 词法分析

- 输入：源程序
- 输出：单词
- 由词法分析器完成

- ✓ 输入源程序；
- ✓ 断词：扫描，分解字符串；
- ✓ 查字典：识别出具有独立意义的字符串作为记号（token）。
  - 形成记号的字符串叫做该记号的单词(lexeme)。

- 主要理论基础： 自动机理论
- 工作依据：构词规则
  - 源语言的构词规则(即词法规则)，也称为模式(pattern)。

例如：标识符的模式是：以字母开头的字母数字序列。

词法分析器

从左到右扫描组成源程序的字符串，并将其转换成单词串；同时要：检查词法错误，进行标识符登记，即符号表管理。

- 所做的转换：

输出的形式：多种形式

例1：序对**(词性，本身)**

词性通常分为5类：定义符、标识符、运算符、界符、常数

例2：序对(种别码，属性值)　属性值：**token**的机内表示

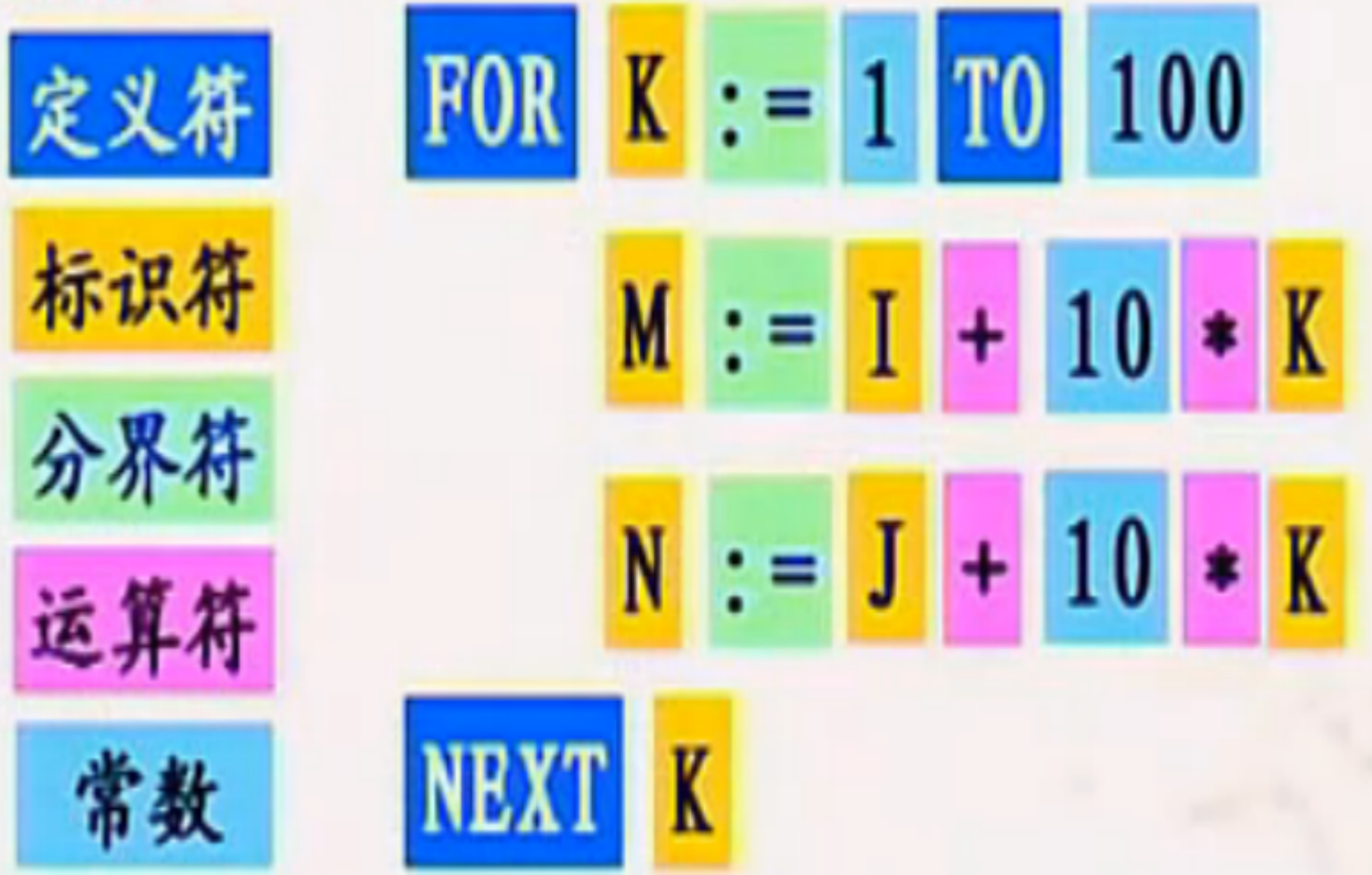
sum=(10+20)*(num+square);

(标识符，sum)
(界符，=)
(界符，（)
(常数，10)
(运算符，+)
(常数，20)
(界符，）)
(运算符，*)
(界符，（)
(标识符，num)
(运算符，+)
(标识符，square)
(界符，）)
(界符，;)

结果：
(标识符，sum)
(赋值号，=)
(左括号，（)
(整常数，10)
(加号，+)
(整常数，20)
(右括号，）)
(乘号，*)
(左括号，（)
(标识符，num)
(加号，+)
(标识符，square)
(右括号，）)
(分号，;)

词法分析

示例



空格、注释的处理及其他

- 分隔记号的空格：被删去
- 源程序中的注释：被跳过
- 识别出来的标识符要放入符号表。
- 对某些记号还要增加一个“属性值”

例如：total:=total+rate*4

– 如发现标识符total时，词法分析器不仅产生一个记号，如id，还把它的单词total填入符号表（如果total在表中不存在的话），记号id的属性值就是指向符号表中R条目的指针。

	名字	属性
1	total	...
2	rate	

对 total:=total+rate*4 的词法分析

(1) 标识符　total （内部名字为id_1）
(2) 赋值号　:=
(3) 标识符　total （内部名字为id_1）
(4) 加号　+
(5) 标识符　rate （内部名字为id_2）
(6) 乘号　*
(7) 整常数　4

$id_1 := id_1 + id_2 * 4$

(标识符，id_1)
(界符，：=)
(标识符，id_1)
(运算符，+)
(标识符，id_2)
(运算符，*)
(常数，4)

2. 语法分析

- 输入：单词符号串
- 输出：语法单位
- 层次结构的分析

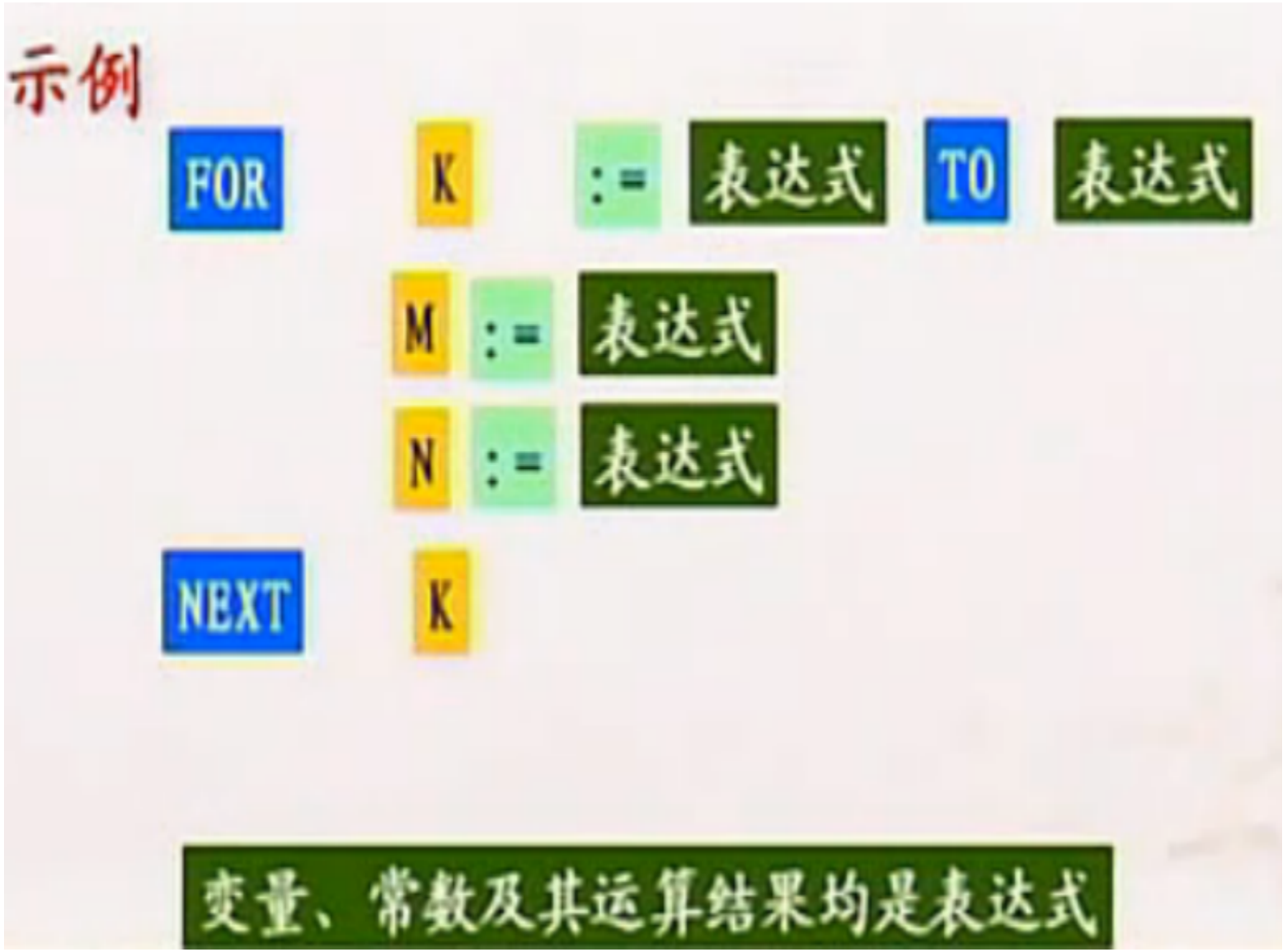
  把单词串按语言的语法结构层次地分组，以形成语法单位（短语、子句、句子、程序段、程序）。

  分析树：表示源程序的语法单位。
- 工作依据：源语言的语法规则。
- 主要理论基础：上下文无关文法

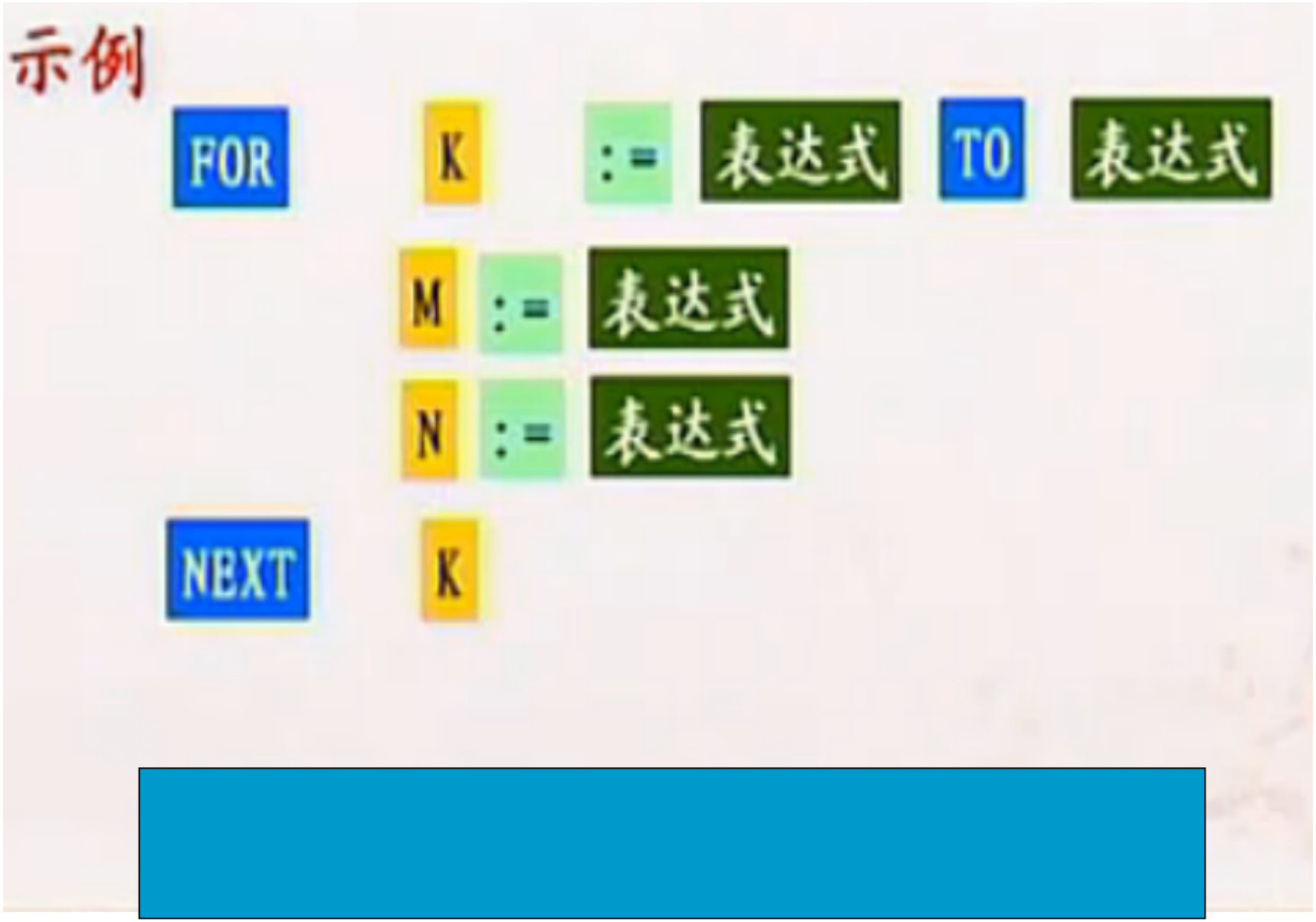
示例

```
FOR K := 1 TO 100
    M := I + 10 * K
    N := J + 10 * K
NEXT K
```

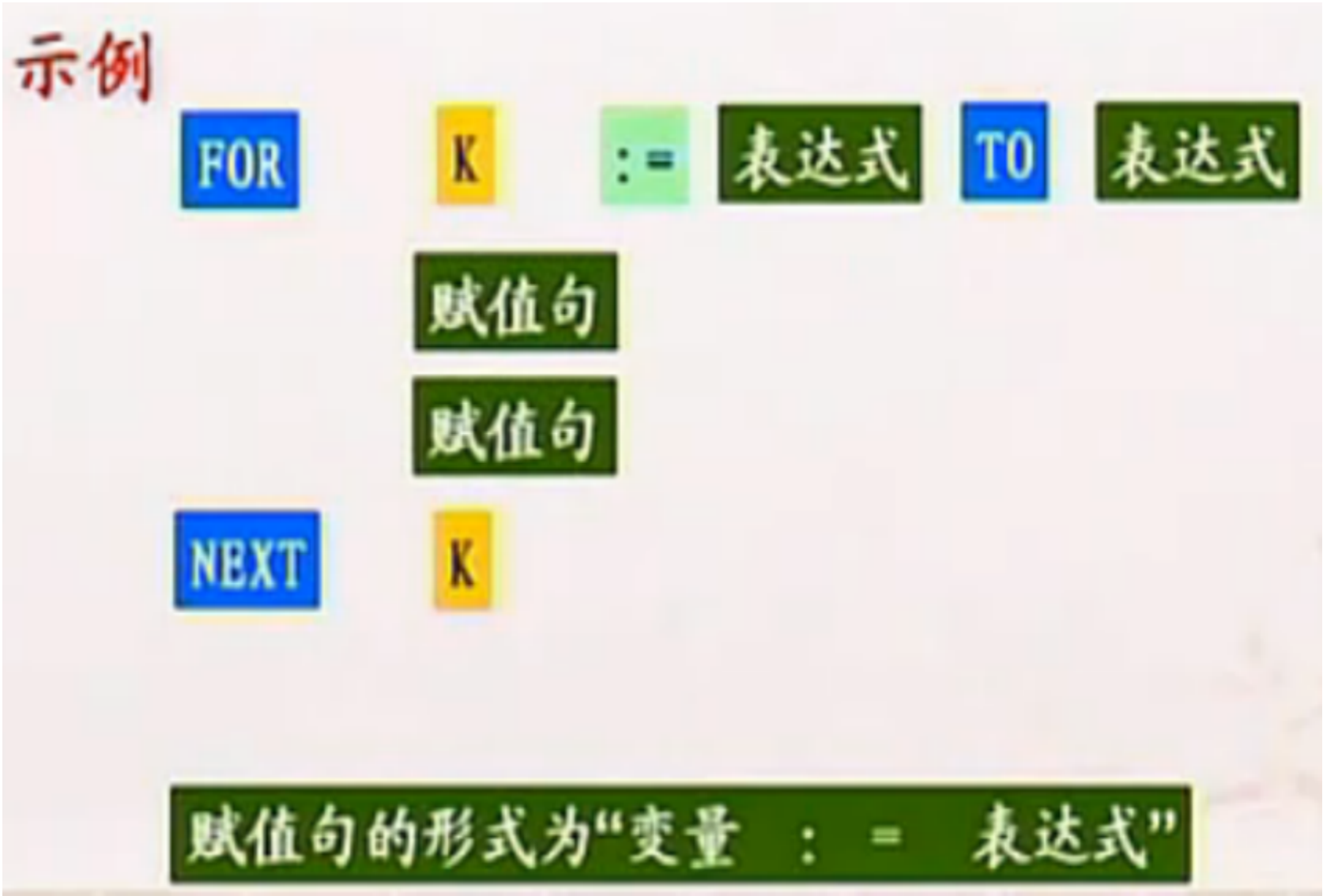
示例



示例

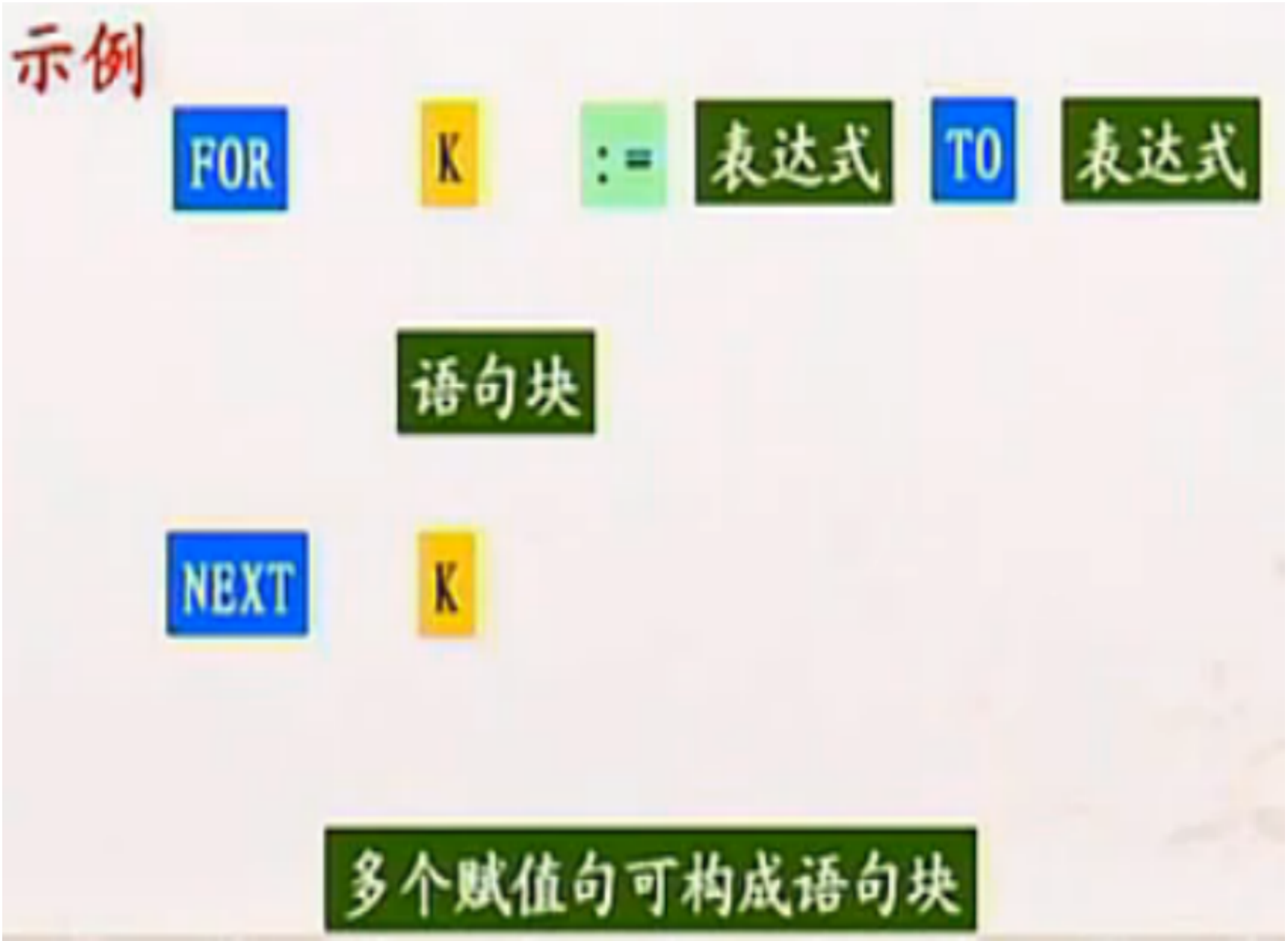


示例



示例

示例



示例

FOR K := 初值 TO 终值

语句块

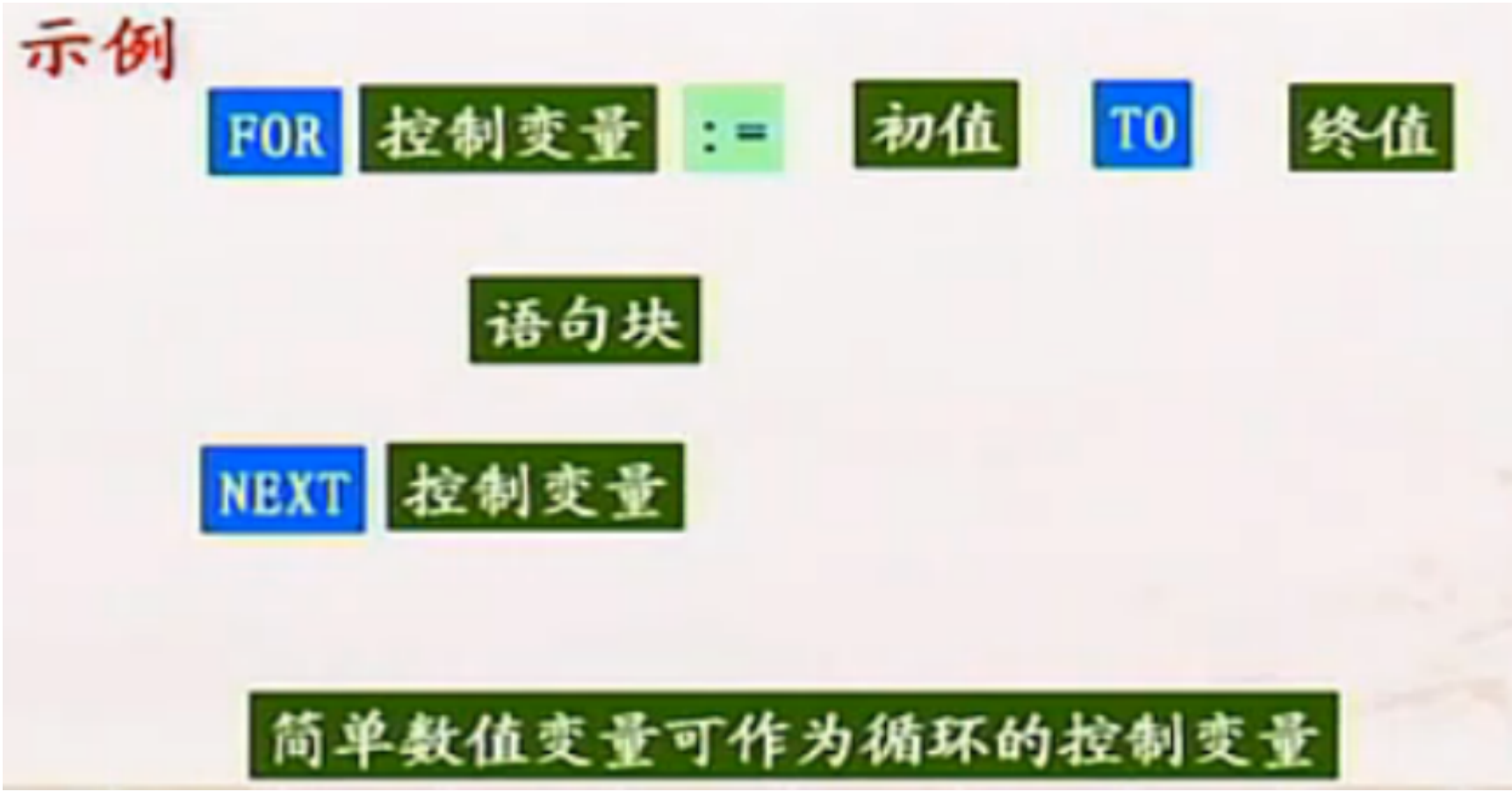
NEXT K

示例

FOR K := 初值 TO 终值

语句块

NEXT K



上述结果符合FOR循环语句的语法定义，故语法分析成功完成。

* 程序的层次结构通常由递归的规则表示，

【例如】表达式的定义规则如下：

(1) 任何一个标识符是一个表达式

(2) 任何一个数是一个表达式

(3)如果$expr_1$和$expr_2$是表达式，$expr_1+expr_2$、$expr_1*expr_2$、$(expr_1)$也都是表达式。

【例如】使用如下规则递归地定义语句：

(1) 如果id是一个标识符，expr是一个表达式，则

id:= expr是一个语句。

(2) 如果expr是一个表达式，stmt是语句，则

while(expr) do stmt和if(expr) then stmt

都是语句。

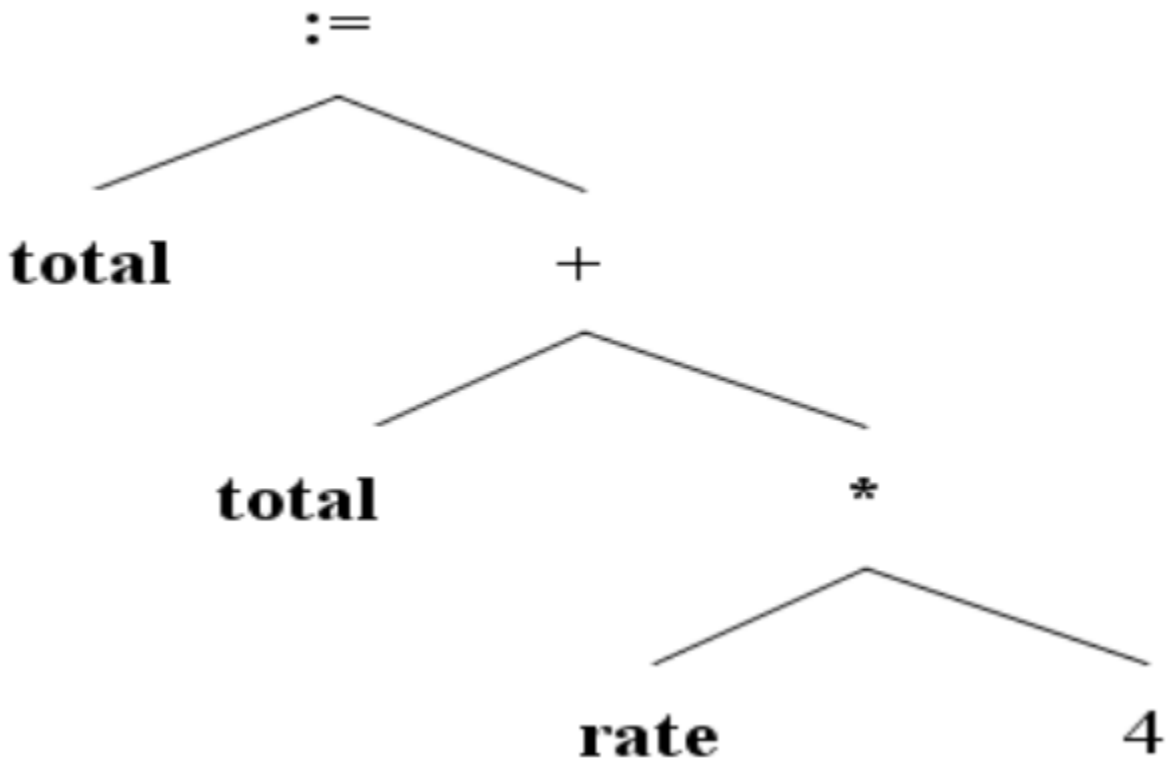
根据语句和表达式的定义规则，构造total:=total+rate*4 的分析树

total是表达式；
rate 是表达式；
4是表达式
rate*4是表达式；
total+rate*4是表达式
total:=total+rate*4是一个语句

total:=total+rate*4 的分析树

与该分析树对应的语法树如下：

语法结构的内部表示形式



语法树是分析树的浓缩表示，其中，算符作为内部结点，运算对象作为子结点。

3. 语义分析

- 对语句的意义进行检查分析
- 收集类型等必要信息
- 工作依据：源语言的语义规则（程序语义正确性的规定）
- 理论基础：属性文法
- 一个重要任务：类型检查
  - 根据规则检查每个运算符及其运算对象是否符合要求；
  - 数组的下标是否合法；
  - 过程调用时，形参与实参个数、类型是否匹配等。


循环语句
FOR K := 1 TO 100
M := I + 10 * K
N := J + 10 * K
NEXT K
出口语句
循环语句和出口语句彼此相连地被定义
含义：这里有个转移语句。

循环块

```
FOR K := 1 TO 100
    M := I + 10 * K
    N := J + 10 * K
NEXT K
```

包括循环语句开始到有同一控制变量的第一个出口语句的那些语句的自然序列称为一循环块

示例

```
FOR K := 1 TO 100
    M := I + 10 * K
    N := J + 10 * K
NEXT K
```

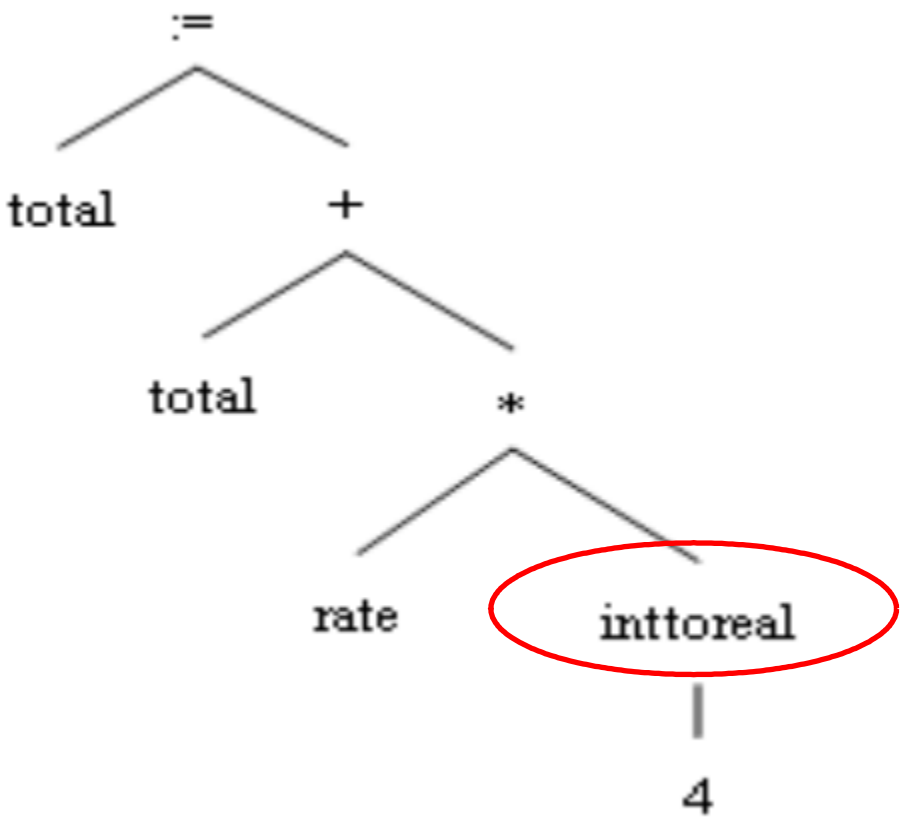
块嵌套不可交叉，嵌套块控制变量不可同名

不正确　　正确嵌套

赋值语句 total:=total+rate*4

插入转换符的语法树

float total, rate;



增加一个语义处理结点：将整型变为实型的一目运算符

二、综合阶段

- 任务：根据所制定的源语言到目标语言的对应关系，对分析阶段所产生的**中间形式**进行**综合加工**，从而得到**与源程序等价的目标程序**。
- 任务划分：
  - (4) 中间代码生成
  - (5) 代码优化
  - (6) 目标代码生成

4. 中间代码生成

- 中间代码：一种抽象的机器程序（与机器无关）
- 中间代码应具有两个重要的特点：
  - 易于产生
  - 易于翻译成目标代码
- 中间代码有多种形式：

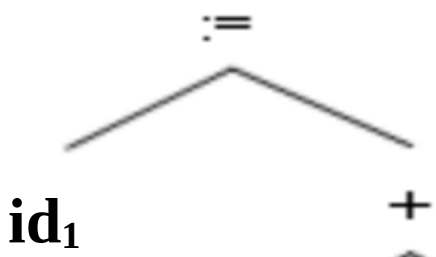
total:=total+rate*4 的三地址代码

$temp_1$:=inttoreal(4)

$temp_2$:=id_2*$temp_1$

$temp_3$:=id_1+$temp_2$

id_1:=$temp_3$



根据语义分析所产生
为每个运算符生成一

三地址代码(形如 x:=y op z) 的特点:

- ✓ 每条语句除了赋值号之外，最多还有一个运算符。
- ✓ 当生成这些语句时，编译器必须确定要进行运算的顺序。
- ✓ 编译程序必须生成临时变量名，以便保留每条语句的计算结果。
- ✓ 语句中出现的地址可以少于三个。

生成四元式

(1) (:=, 1, , K)

(2) (j<, 100, K, (9))

(3) (*, 10, K, T_1)

(4) (+, I, T_1, M)

(5) (*, 10, K, T_2)

(6) (+, J, T_2, N)

(7) (+, K, 1, K)

(8) (j, , , (2))

(9) ( )

5. 代码优化

- 对中间代码进行改进(加工变换)，使之占用的空间少、运行速度快。
- 工作依据：程序等价变换规则
- 理论基础：数据流方程

代码优化首先是在中间代码上进行的。

$temp_1$:=inttoreal(4)

$temp_2$:=id_2*$temp_1$

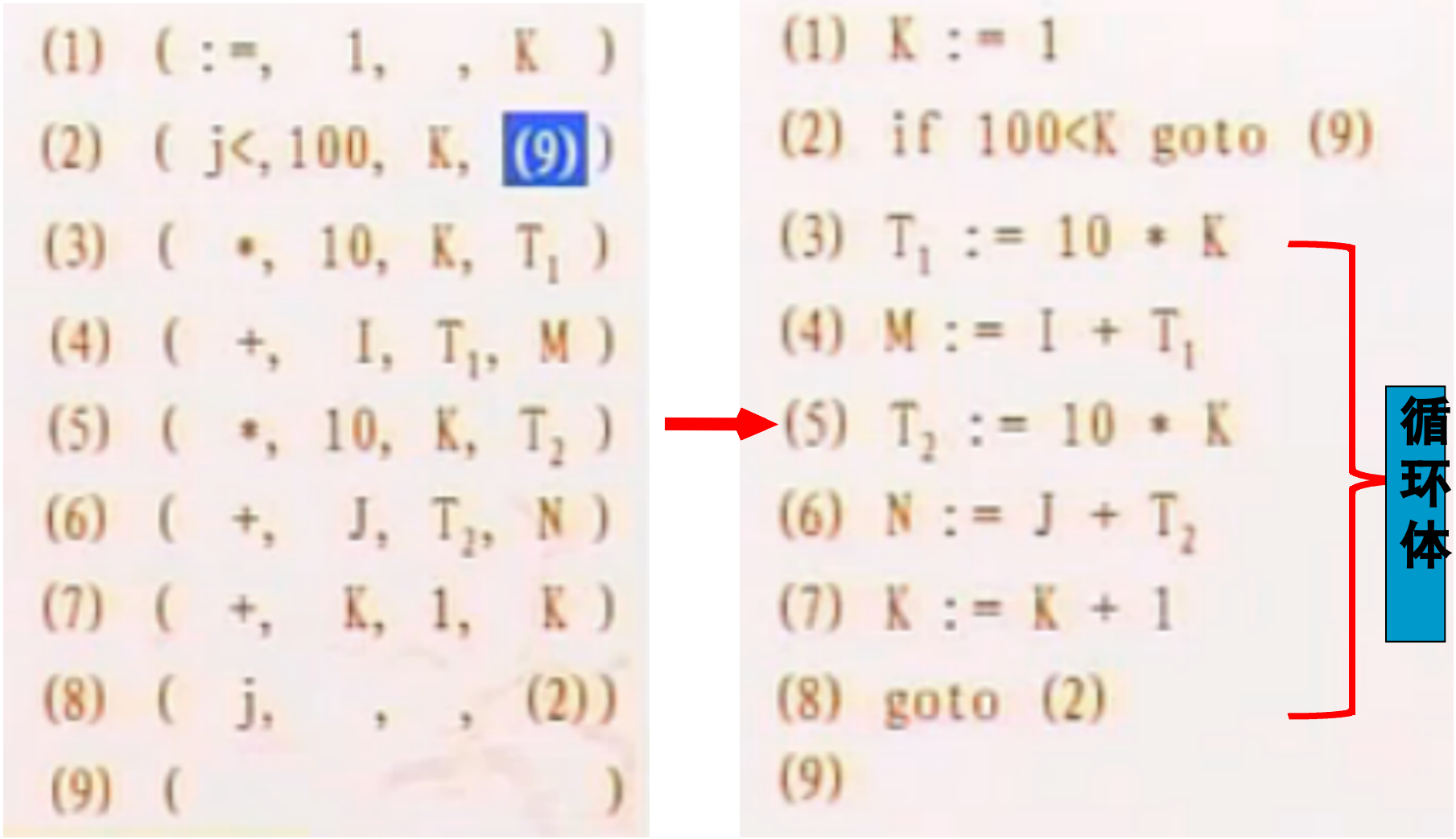
$temp_3$:=id_1+$temp_2$

id_1:=$temp_3$

$temp_1$:=id_2*4.0

$temp_2$:=id_1+$temp_1$

id_1:=$temp_2$

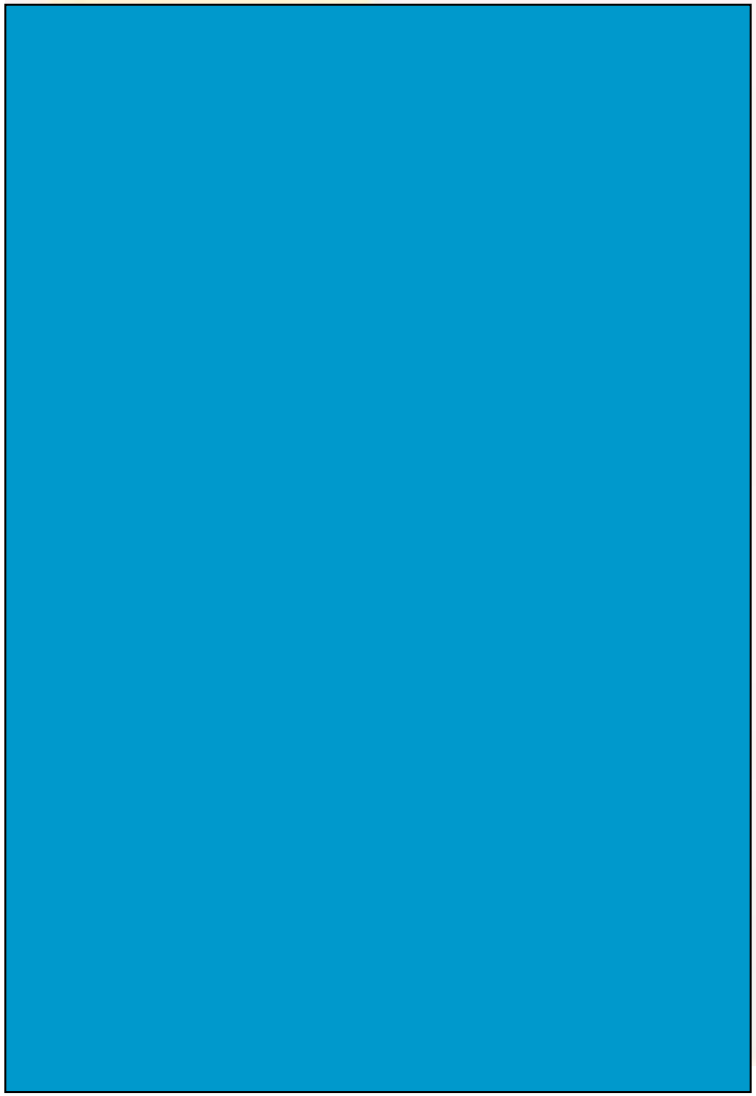


将四元式转换成另一种形式的中间代码

循环体中进行了 ？次加法，？次乘法。

代码优化

优化的结果：循环体中进行了300次加法，没有乘法了。



```
(1) K := 1
(2) if 100<K goto (9)
(3) T1 := 10 * K
(4) M := I + T1
(5) T2 := 10 * K
(6) N := J + T2
(7) K := K + 1
(8) goto (2)
(9)
```

6. 目标代码生成

- 将中间代码变换成特定机器上的目标代码。
- 工作依据：特定机器的硬件系统结构和机器指令的含义。
- **生成的目标代码一般是可重定位的机器代码或汇编语言代码。**
- **涉及到的重要问题：**
  - **对程序中使用的每个变量要指定存储单元（即，分配具体的存储地址）**
  - **对变量进行寄存器分配（即，使用实际机器的寄存器）**
  - **指令的选择：例如，加法指令（内存加？还是寄存器加？）**

total:=total+rate*4 的目标代码

目标机器的指令形式：

OP S, D

$temp_1 := id_2 * 4.0$
$temp_2 := id_1 + temp_1$
$id_1 := temp_2$

```
MOVF id2, R2
MULF #4.0, R2
MOVF id1, R1
ADDF R2, R1
MOVF R1, id1
```

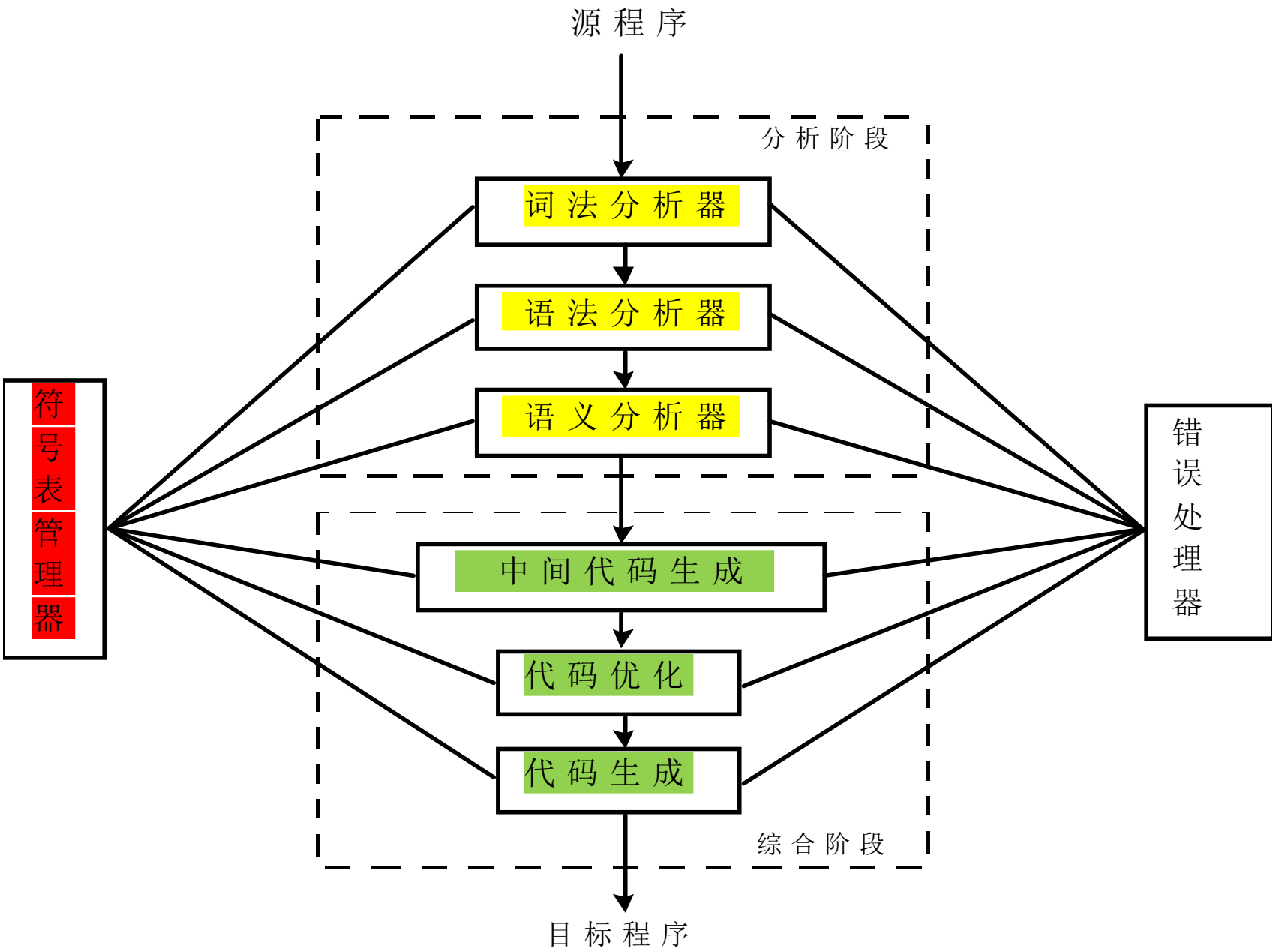
目标机器的指令形式：

OP DEST, SRC

```
MOVF R0, id2
MULF R0, #4.0
MOVF R1, id1
ADDF R0, R1
MOVF id1, R0
```

以二地址机器指令为例

```
      LD  R1,  I          (1)  M := I
      LD  R2,  J          (2)  N := J
      LD  R0,  1          (3)  K := 1
L1:   CMP 100, R0         (4)  if 100<K goto (9)
      J<  L2
      ADD R1,  10         (5)  M := M + 10
      ADD R2,  10         (6)  N := N + 10
      ADD R0,  1          (7)  K := K + 1
      J   L1
L2:   ST  R1,  M          (8)  goto (4)
      ST  R2,  N          (9)
```


源 程 序
分 析 阶 段
词 法 分 析 器
语 法 分 析 器
语 义 分 析 器
符 号 表 管 理 器
错 误 处 理 器
中 间 代 码 生 成
代 码 优 化
代 码 生 成
综 合 阶 段
目 标 程 序

三、符号表管理

- 编译程序的一项重要工作：
  - 记录源程序中使用的标识符
  - 收集每个标识符的各种属性信息
- 建表的技术支持是数据结构
- 符号表是由若干记录组成的数据结构
  - 每个标识符在表中有一条记录
  - 记录的域是标识符的属性
- 对符号表结构的要求：
  - 应允许快速地找到标识符的记录
  - 可在其中存取数据
- 标识符的各种属性是在编译的各个不同阶段填入符号表的。

声明语句：float total, rate;

- 词法分析器：
  - float是关键字
  - total、rate是标识符
  - 在符号表中创建这两个标识符的记录
- 语法分析器：
  - total、rate都表示变量
  - float表示这两个变量的类型
  - 可以把这两种属性及存储分配信息填入符号表。
- 在语义分析和生成中间代码时，还要在符号表中填入对这个float型变量进行存储分配的信息。

四、错误处理

- 一个好的编译器：
  - 全：最大限度地发现错误，一次可以发现尽可能多的错误。
  - 准：准确定位错误。（判断位置和性质）
  - 局部化：将错误的影响限制到尽可能小的范围。不要将错误蔓延。
  - 能自动修改错误，代价很高。适当的恢复

- 在编译的每个阶段都可能检测到源程序中存在的错误
  - 词法分析程序可以检测出非法字符错误。
  - 语法分析程序能够发现不符合语法规则的错误。
  - 语义分析程序试图检测出具有正确的语法结构，但对所涉及的操作无意义的结构。
  - 代码生成程序可能发现目标程序区超出了允许范围的错误。
  - 由于计算机容量的限制，编译程序的处理能力受到限制而引起的错误。

Error classification

按照发现时翻译器所处的阶段进行分类：

- Lexical: character-level error, such as illegal character (hard to distinguish from syntax).
- Syntax: error in structure (e.g., missing semicolon or keyword).
- Static semantic: non-syntax error prior to execution (e.g., undefined vars, type errors).
- Dynamic semantic: non-syntax error during execution (e.g., division by 0).
- Logic: programmer error

Sample Errors (Java):

```
public int gcd ( int v# )// lexical syntax
{ int z = 5              // syntax - missing ;
  y = v;                 // static semantic - y undefined
  while ( y >= 0 )// dynamic semantic - division
  { int t = y;              by 0
    y = z % y;
    z = t;
  }
  return y;              // logic - should return z
}
```

1.3 编译有关的其他概念

一、前端和后端

与源语言有关而与目标机器无关的部分组成

- 词法分析、语法分析、符号表的建立、语义分析和中间代码生成
- 与机器无关的代码优化工作
- 相应的错误处理工作和符号表操作

与目标机器有关的部分组成

- 与机器有关的代码优化、目标代码生成
- 相应的错误处理和符号表操作

- **把编译程序划分成前端和后端的优点：**
  - **便于移植、便于编译程序的构造。**

例如：

在不同的机器构造同一源语言的编译程序。

将某一编译程序的前端加上相应不同的后端。

在同一机器构造几个语言的编译程序。

将不同语言编译的前端生成同一种中间语言，再使用一个共同的后端。

二、遍

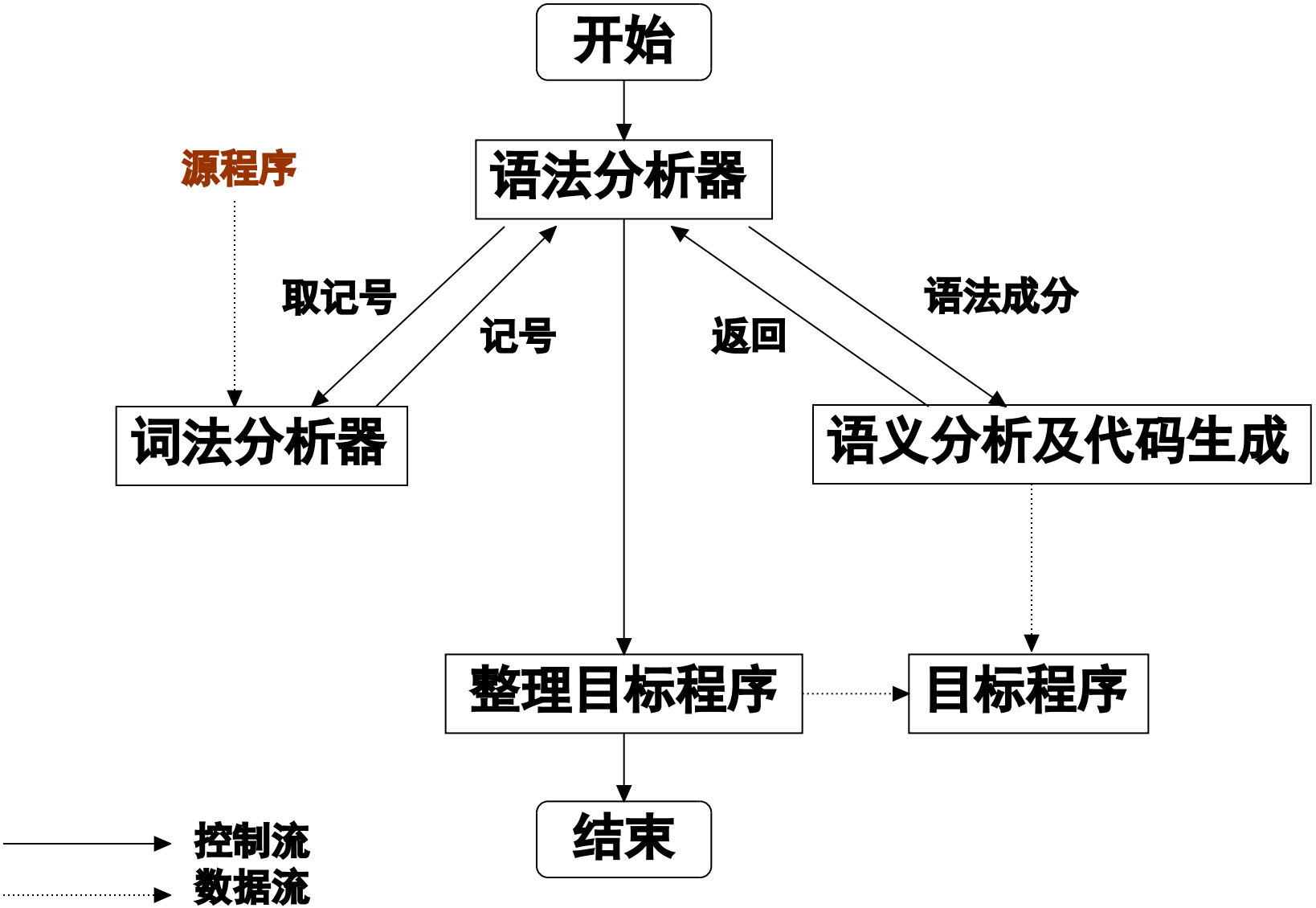
- 一“遍”是指对源程序或其中间形式从头到尾扫描一趟，并作相关的加工处理，生成新的中间表示形式或目标程序。
- 根据系统资源的状况、运行目标的要求等，可以将一个编译程序设计成多遍（Pass）扫描的形式，在每一遍扫描中，完成不同的任务。
- 遍可以和阶段相对应，也可以和阶段无关

* 编译程序的结构受“遍”的影响
  - 扫描遍数
  - 分遍方式

✓一遍扫描的编译程序

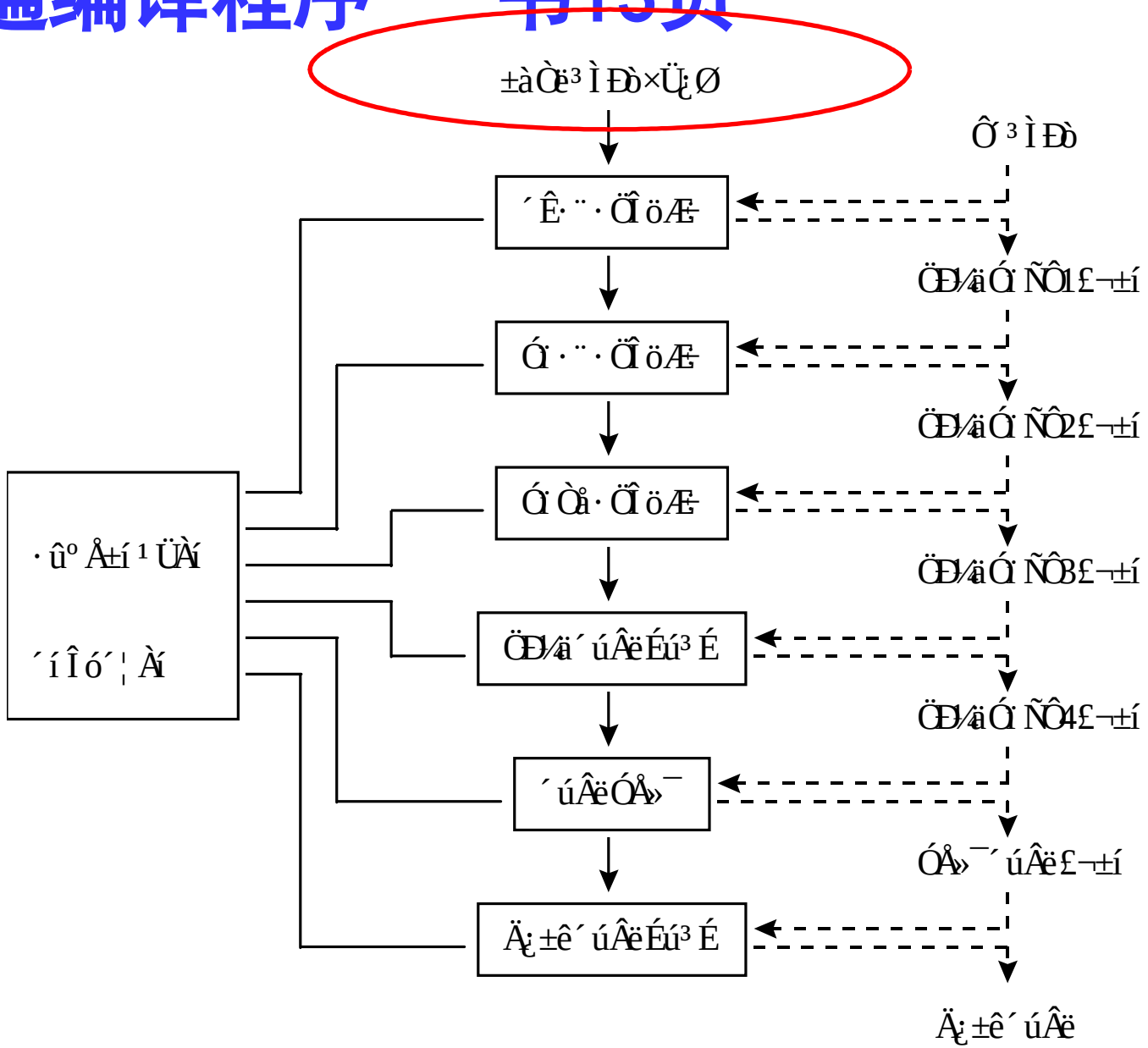
✓多遍编译程序

一遍扫描的编译程序　书12页



61

多遍编译程序　　书13页

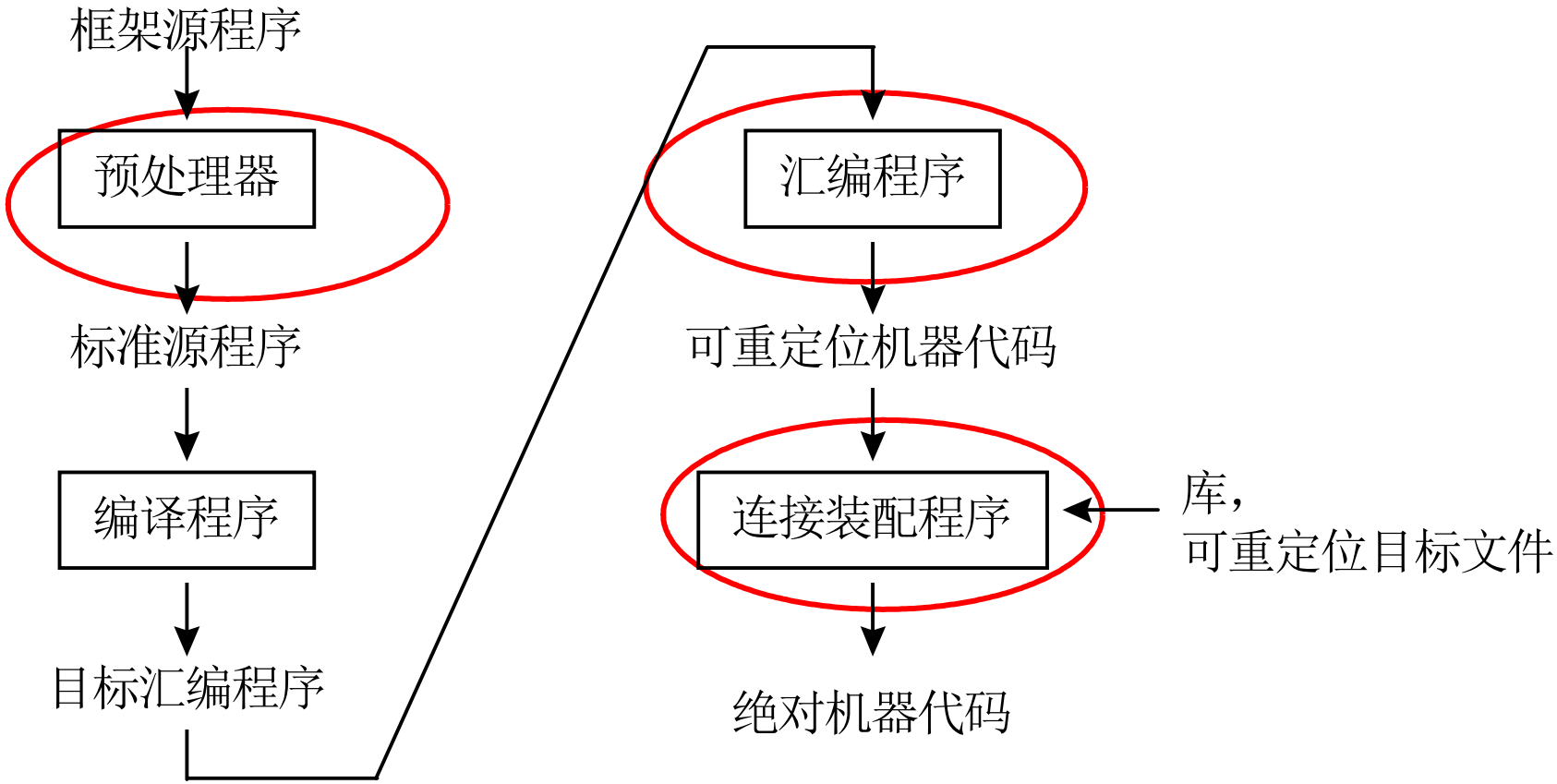

±àÒë³ÌÐò×Ü¿Ø
Ô³ÌÐò
´Ê·¨·ÖÎöÆ÷
ÖÐ¼ä´úÂëÉú³É
Ó﷨·ÖÎöÆ÷
ÖÐ¼ä´úÂëÉú³É
Óïò·ÖÎöÆ÷
ÖÐ¼ä´úÂëÉú³É
·ûºÅ±í¹ÜÀí
´íÎó´¦Àí
´úÂëÓÅ»¯
ÖÐ¼ä´úÂëÉú³É
Ä¿±ê´úÂëÉú³É
ÓÅ»¯´úÂë¡¬±í
Ä¿±ê´úÂë

编译程序分遍的优缺点

- 分遍的主要好处：
  - 可以减少对主存容量的要求
  - 可使各遍编译程序功能独立、单纯，相互联系简单，编译程序结构清晰。
  - 能够实现更充分的优化工作，获得高质量目标程序。
  - 通过分遍将编译程序的前端和后端分开，可以为编译程序的移植创造条件。

- 分遍的缺点：
  - 增加了不少重复性的工作。（读符号、送符号）

1.4 编译程序的伙伴工具



一个语言处理系统

一、预处理器

- 预处理器的主要功能：
  1. 宏处理
  2. 文件包含
  3. 语言扩充

1. 宏处理

- 处理两类语句，即宏定义和宏调用。
  - **宏定义**通常用统一的字符或关键字表示，如define或macro，宏定义由宏名字及宏体组成，通常宏处理器允许在宏定义中使用形参。

C语言源程序中的一个宏定义：

```
#define  prompt(s)  fprintf(stderr, s)
```

  - **宏调用**由调用宏的命令名（宏名）和所提供的实参组成。宏处理器用实参代替宏体中的形参，再用变换后的宏体替换宏调用本身。

2. 文件包含

- 将文件包含声明扩展为程序正文。

- C语言程序中的“头文件”包含声明行：

```
#include <stdio.h>
```

预处理器处理到该语句时，就用文件stdio.h的内容替换此语句。

3. 语言扩充

- 用更先进的控制流和数据结构来增强原来的语言。

- 例如：
  - 将类似于while或if-then-else语句结构的**内部宏**提供给用户使用，而这些结构在原来的程序设计语言中是没有的。
  - 当程序中使用了这样的结构时，由**预处理器**通过**宏调用**实现语言功能的扩充。

二、汇编程序

- 汇编语言用助记符表示操作码，用标识符表示存储地址。
- 赋值语句b:=a+2相应的汇编语言程序为：

```
MOV  a,  R1
ADD  #2,  R1
MOV  R1,  b
```

或者

```
MOV R1, a
ADD R1, #2
MOV b, R1
```

- 最简单的汇编程序对输入作两遍扫描。

第一遍

- 找出标志存储单元的所有标识符，并将它们存储到**汇编符号表**中。
  - **汇编符号表**独立于**编译程序的符号表**
- 在符号表中指定该标识符所对应的存储单元地址，此地址是在首次遇到该标识符时确定的。

- 假定一个字包括4个字节，每个变量占一个字，完成第一遍扫描后，得到**汇编符号表**：

标识符	地址
a	0
b	4

第二遍

- 把每个用助记符表示的操作码翻译为二进制表示的机器代码。
- 把用标识符表示的存储地址翻译为汇编符号表中该标识符所对应的地址。
- 输出通常是可重定位的机器代码。
  - 起始地址为0，各条指令及其所访问的地址都是相对于0的逻辑地址。
  - 当装入内存时，可以指定任意的地址L作为开始单元。
- 输出中要对那些需要重定位的指令做出标记
  - 标记供装入程序识别，以便计算相应的物理地址。

举例：可重定位机器代码

b:=a+2的汇编代码

MOV a, R_1

ADD #2, R_1

MOV R_1, b

- 假定：

  0001　代表　MOV S,　R

  0011　代表　ADD

  0010　代表　MOV R,　D

- 假定机器指令的格式为：

  操作符　寄存器　寻址模式　地址

- 第二遍输出的可重定位机器代码：

  0001　01　00　00000000*

  0011　01　10　00000010

  0010　01　00　00000100*

绝对机器代码

- 假如装入内存的起始地址为L=00001111
- 则a和b的地址分别是15和19
- 则装入后的机器代码为：

```
0001   01   00   00001111
0011   01   10   00000010
0010   01   00   00010011
```

可重定位机器代码：

```
0001  01  00  00000000*
0011  01  10  00000010
0010  01  00  00000100*
```

b:=a+2的汇编代码

```
MOV a, R1
ADD #2, R1
MOV R1, b
```

三、连接装配程序

- **装入**
  - 读入可重定位的机器代码
  - 修改重定位地址
  - 把修改后的指令和数据放在内存的适当地方或形成可执行文件

- **连接**
  - 把几个可重定位的机器代码文件连接成一个可执行的程序

1.5 编译原理的应用（自学）

- 语法制导的结构化编辑器
- 程序格式化工具
- 软件测试工具
- 程序理解工具
- 高级语言的翻译工具

75

小　　结

汇编语言程序　　汇编程序

目标程序=目标代码　　机器码的目标程序

目标语言与目标语言程序

源语言与源程序

- 什么是编译
- 翻译程序：
  - 编译程序、汇编程序、解释程序
  - 异同

小　　结（续）

- 编译程序的各组成部分及其功能
  - 词法分析
  - 语法分析
  - 语义分析
  - 中间代码生成
  - 代码优化
  - 目标代码生成
- 前端和后端
- 遍

本章结束

补充知识；

* 目标代码三种形式：
  - 绝对指令代码：

    具有绝对地址的机器指令，可直接运行
  - 可重新定位指令代码：

    模块结构的机器指令，需要连接装配
  - 汇编指令代码：

    汇编语言形式的目标程序，需要进行汇编

返回

- 中间代码：sum=(10+20)*(num+square)；

后缀表示(逆波兰Anti-Polish Notation)

sum 10 20 + num square +*=

前缀表示(波兰Polish Notation)

= sum *+10 20+num square

四元式表示

(三地址码)

(+，10，20，T_1)
(+，num，square，T_2)
(*，T_1，T_2，T_3)
(=，T_3，，sum)

三元式表示

(+，10，20)
(+，num，square)
(*，(1)，(2))
(=，sum，(3))

返回

- 管理各种符号表(常数、标号、变量、过程、结构。。。)，查、填（登记、查找）源程序中出现的符号和编译程序生成的符号，为编译的各个阶段提供信息。
  - ✓**辅助语法检查、语义检查**
  - ✓**完成静态绑定、管理编译过程**

- **Hash**表、链表等各种表的查、填技术。

致谢

PPT参考资料：

- 编译原理与技术 李文生 清华大学出版社
- 编译原理 国防科技大学 计算机系王挺
- 编译原理 哈工大 计算机科学与技术系姜守旭
- 编译原理 西安交通大学 冯博琴