第1章　编译概述

知识点：

翻译、解释、编译

编译的阶段、任务、及典型结构

编译程序的伙伴工具

教学目标与要求

- 认识编译程序在计算机系统中的地位；
- 了解编译程序的构成及模块功能；
- 了解编译程序的伙伴工具及其作用。

wenshli@bupt.edu.cn

本章内容

简介

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

wenshli@bupt.edu.cn

简介

- 什么是编译？
  - 把源程序转换成等价的目标程序的过程
- 编译程序的设计涉及到的知识：
  - 程序设计语言
  - 形式语言与自动机理论
  - 数据结构
  - 算法分析与设计
  - 操作系统
  - 计算机组成及体系结构
  - 软件工程等

wenshli@bupt.edu.cn

1.1 翻译和解释

一、程序设计语言

二、翻译程序

wenshli@bupt.edu.cn

一、程序设计语言

- 低级语言
  - 机器语言
  - 符号语言 汇编语言
- 高级语言
  - 过程性语言—面向用户的语言 如：C、Pascal
  - 专用语言—面向问题的语言 如：SQL
  - 面向对象的语言 如：Java、C++

wenshli@bupt.edu.cn

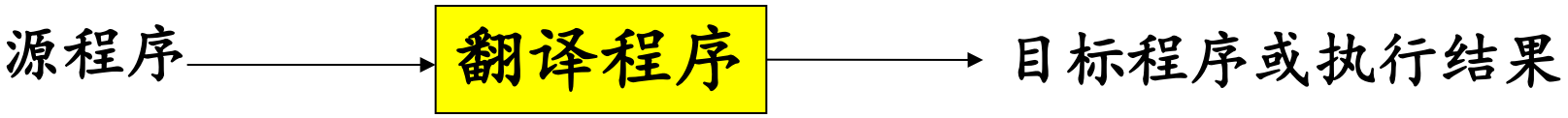
高级语言的优点

- 高级语言独立于机器。所编程序移植性比较好。
- 不必考虑存储空间的分配问题，不需要了解数据从外部形式转换成机器内部形式的细节。
  - 用变量描述存储单元
- 具有丰富的数据结构和控制结构。
  - 数据结构：数组、记录等
  - 控制结构：循环、分支、过程调用等。
- 更接近于自然语言。
  - 可读性好，便于维护。
- 编程效率高。

wenshli@bupt.edu.cn

二、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。



- 翻译程序分为两大类：
  1. 编译程序（即编译器）：把源程序翻译成目标程序
  2. 解释程序（即解释器）：直接执行源程序

wenshli@bupt.edu.cn

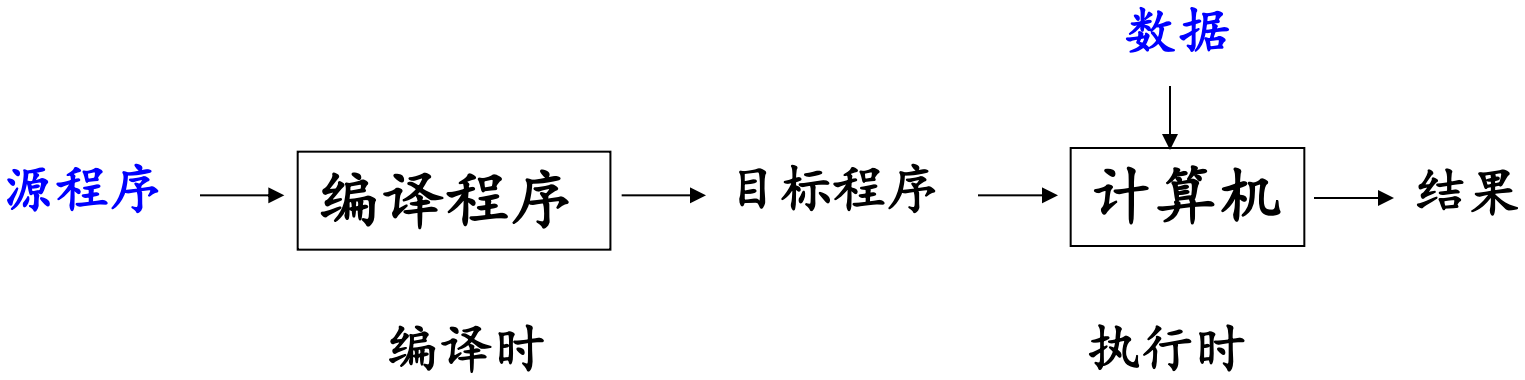
编译程序

- 源程序是用高级语言或汇编语言编写的，目标程序是用汇编或机器语言表示的。
- 两类编译程序：
  - 汇编程序

  - 编译程序

wenshli@bupt.edu.cn

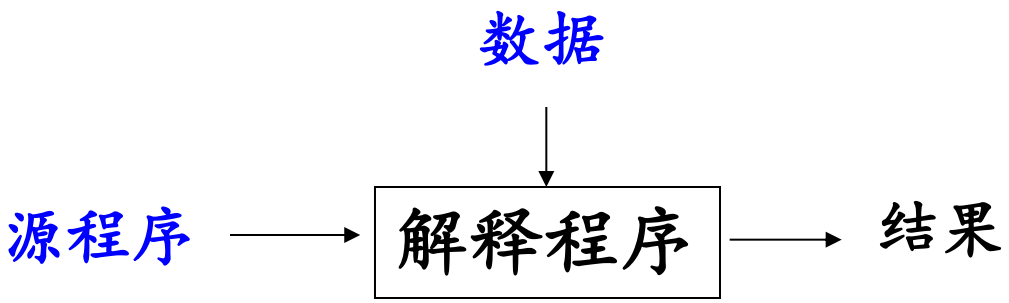
编译和执行阶段



- 编译时间：实现源程序到目标程序的转换所占用的时间。
- 源程序和数据分别在不同时间进行处理
  - 编译阶段
  - 运行阶段

wenshli@bupt.edu.cn

解释程序

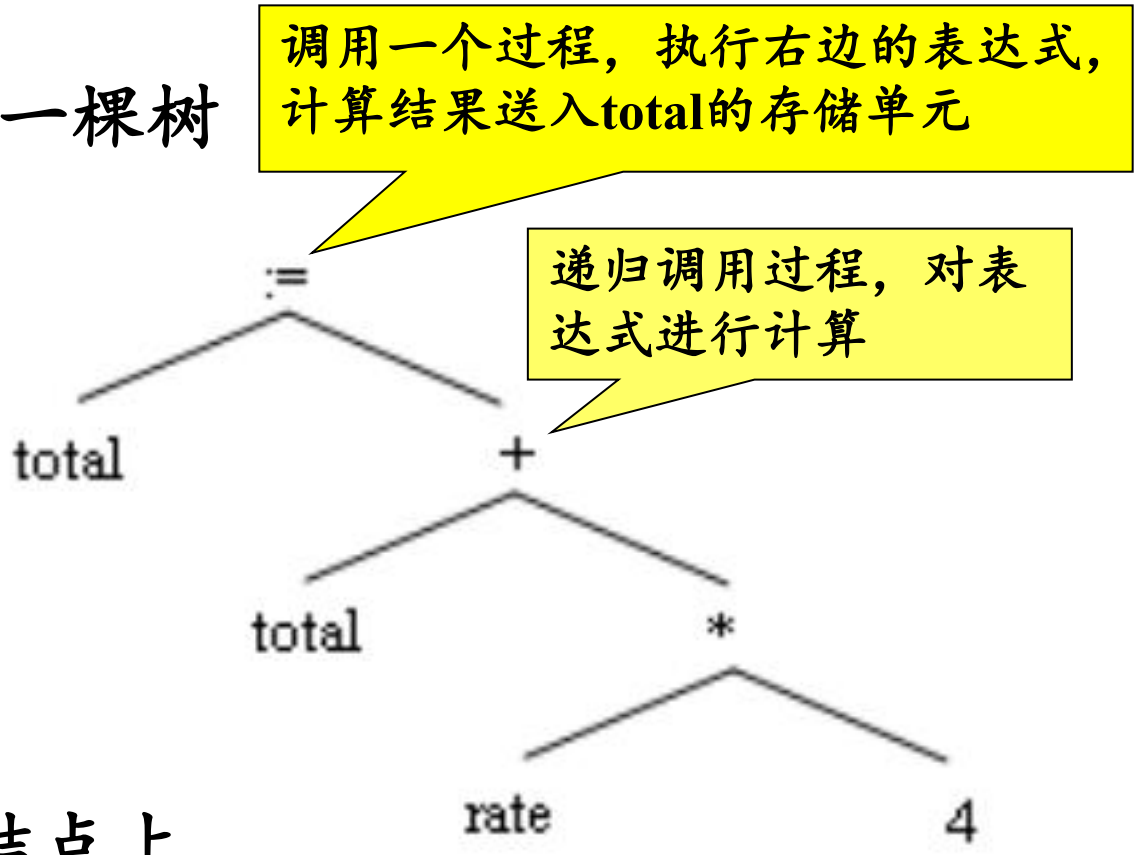


- 同时处理源程序和数据
- 解释执行源程序，不生成目标程序
- 一种有效的方法：
  先将源程序转换为某种中间形式，
  然后对中间形式的程序解释执行。

wenshli@bupt.edu.cn

total:=total+rate*4 的解释过程

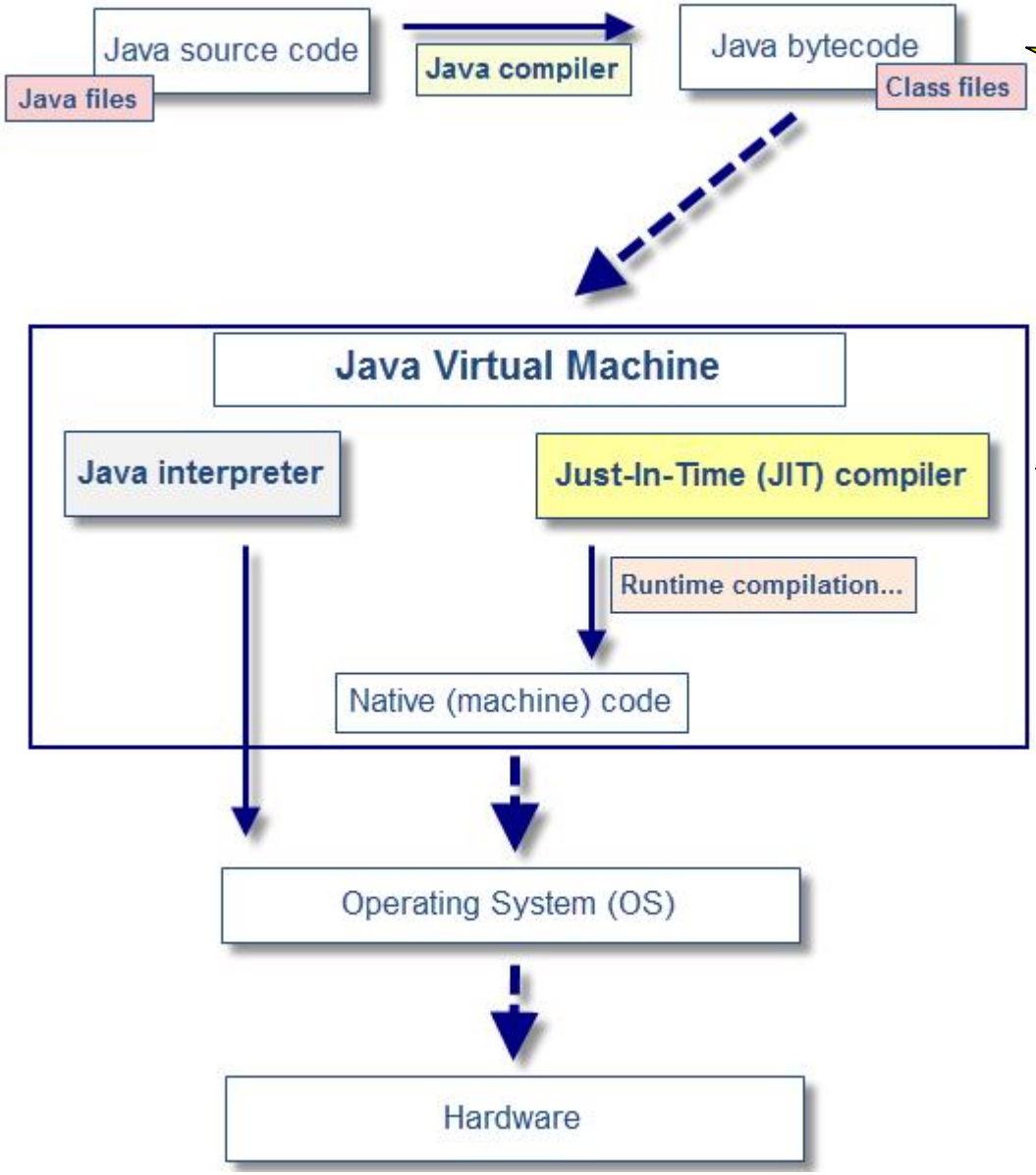
- 先将源程序转换成一棵树



- 遍历该树，执行结点上所规定的动作。

wenshli@bupt.edu.cn

* Java语言处理器：编译+解释



编译：
用javac 把源代码编译成 .class 文件（平台无关的字节码）。

解释：
Java虚拟机对字节码解释执行。

wenshli@bupt.edu.cn

* Java语言处理器（续）

- JIT（Just-in-time）即时编译
  - 在运行中，程序处理输入的前一刻首先把字节码翻译成为机器语言，然后再执行程序。
  - 可以加快java程序的启动速度
  - 代码的执行效率相对差些
- HotSpot VM中的JIT
  - 组合了编译、性能分析以及动态编译。
  - 只编译“热门”代码，即执行最频繁的代码。
  - client版和server版两个编译器。
    - 默认：用client版的。
    - 启动时，指定-server参数，启动server版的编译器。
    - Server版适用于需要长期运行的服务器应用程序，针对最大峰值操作速度进行了优化。

wenshli@bupt.edu.cn

1.2 编译的阶段和任务

一、分析阶段：根据源语言的定义，分析源程序

1.词法分析

2.语法分析

3.语义分析

二、综合阶段：根据分析结果构造目标程序

4.中间代码生成

5.代码优化

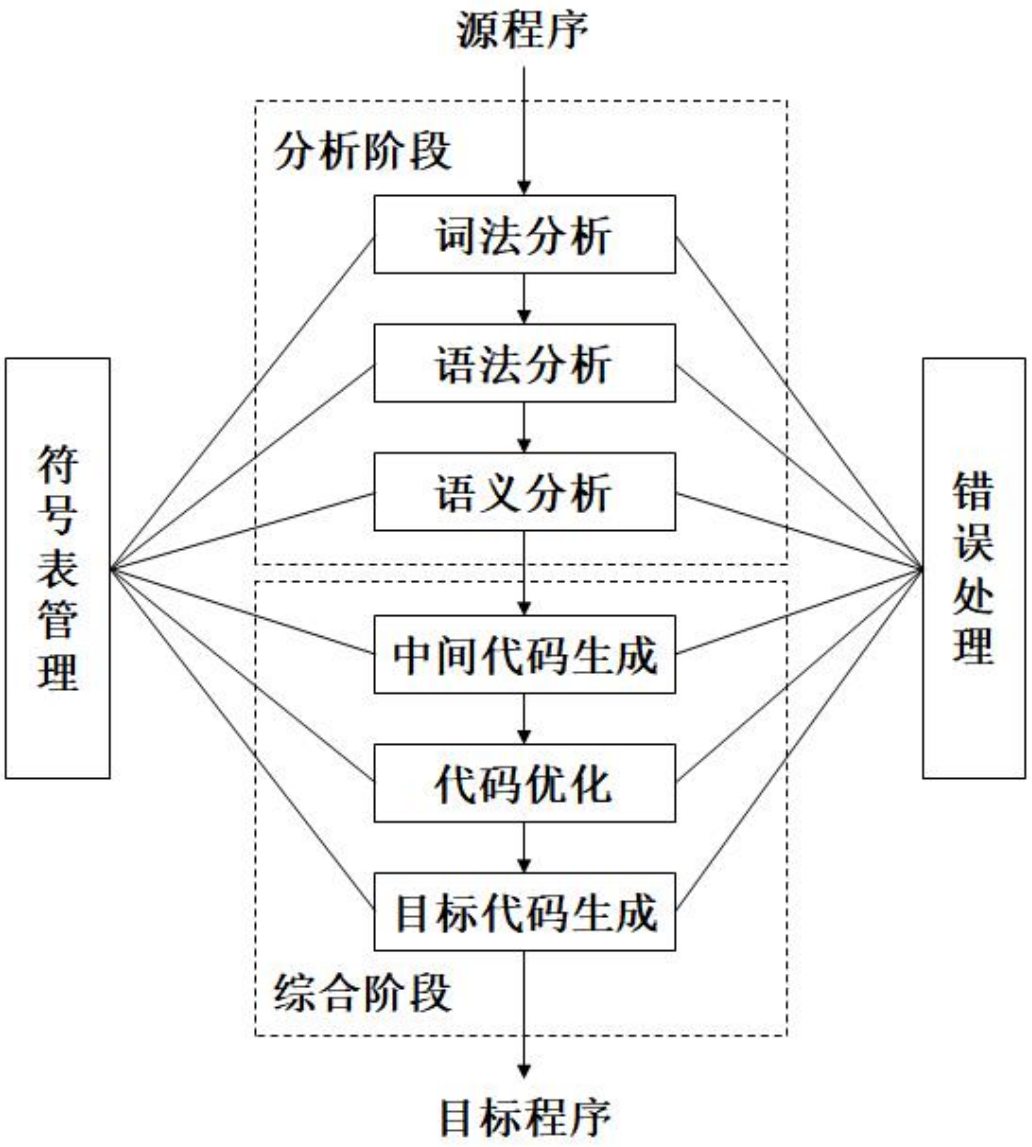
6.目标代码生成

三、符号表的管理

四、错误诊断和处理

wenshli@bupt.edu.cn

编译程序的典型结构



wenshli@bupt.edu.cn

一、分析阶段

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

wenshli@bupt.edu.cn

1. 词法分析

- 线性分析，扫描
- 词法分析程序：
  - 扫描，对构成源程序的字符串进行分解，识别出每个具有独立意义的单词（lexeme），将其转换成记号（token），并组织成记号流。
  - 把需要存放的单词放到符号表中，如变量名，标号，常量名等（视情况需要）。
- 工作依据：构词规则（即词法），也称为模式（pattern）。
  - C语言的标识符的模式是：以字母或下划线开头，由字母、数字或下划线组成的符号串。

wenshli@bupt.edu.cn

对 total:=total+rate*4 的词法分析

(1) 标识符 total

(2) 赋值号 :=

(3) 标识符 total

(4) 加号 +

(5) 标识符 rate

(6) 乘号 *

(7) 整常数 4

wenshli@bupt.edu.cn

空格、注释的处理及其他

非块结构的语言：√
块结构语言，如C/C++、Pascal、Java等，词法分析时无法完成。

- 分隔单词的空格：被跳过
- 源程序中的注释：被跳过
- 识别出来的标识符要放入符号表（视情况）。
- 某些记号还要具有“属性值”
  - 如发现标识符total时，词法分析程序不仅产生一个单词符号的类别标记如id，还把它的单词total填入符号表（如果total在表中不存在的话），则total的记号就包括两部分：单词符号的类别标记id和属性值（即指向符号表中total条目的指针）。
  - 如发现常数3.14时，词法分析程序产生一个类别标记如num，这样，3.14的记号就包括两部分：单词符号的类别标记num 和属性值（3.14）

wenshli@bupt.edu.cn

2. 语法分析

- 层次结构分析
- 工作依据：语法规则
- 程序的层次结构通常由递归的规则表示，如表达式的定义如下：

  (1) 任何一个标识符是一个表达式

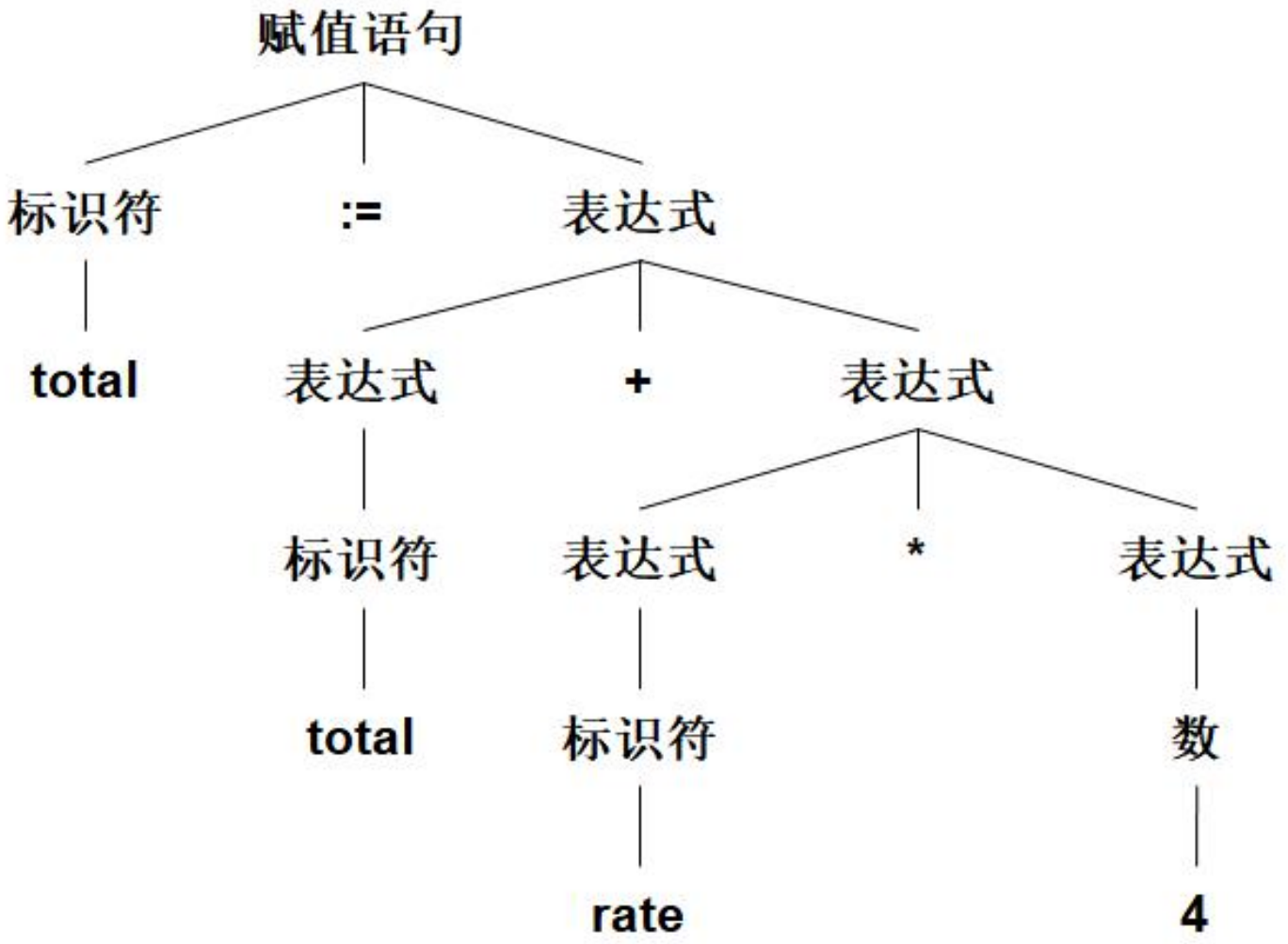
  (2) 任何一个数是一个表达式

  (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。

- 任务：把记号流按语言的语法结构层次地分组，形成语法短语。
- 源程序的语法短语常用分析树表示

wenshli@bupt.edu.cn

total:=total+rate*4 的分析树



wenshli@bupt.edu.cn

语句的递归定义

- 如果id是一个标识符，expr是一个表达式，则
  id:=expr 是一个语句。

- 如果expr是表达式，stmt是语句，则
  while (expr) do stmt 是语句。

  if (expr) then stmt 是语句。

wenshli@bupt.edu.cn

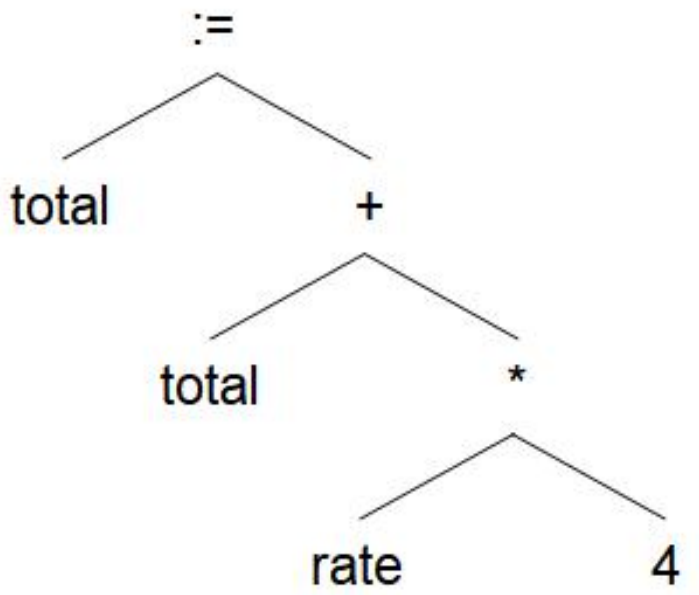
3. 语义分析

- 对语法成份的意义进行检查分析
- 语法成份：语法分析确定的层次结构
- 收集必要信息：类型、作用域等
- 工作依据：语义规则
- 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。

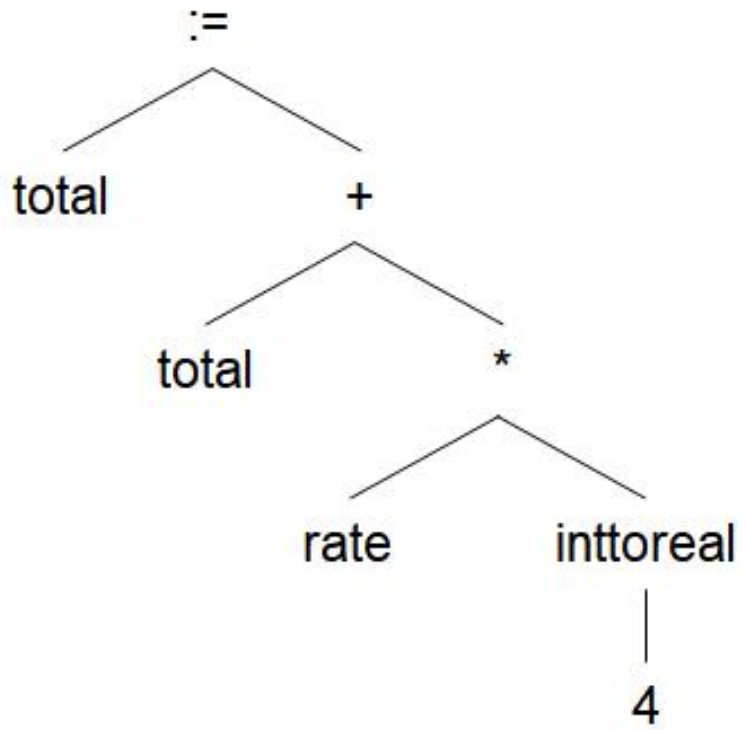
wenshli@bupt.edu.cn

赋值语句 total:=total+rate*4

- 语法分析结果

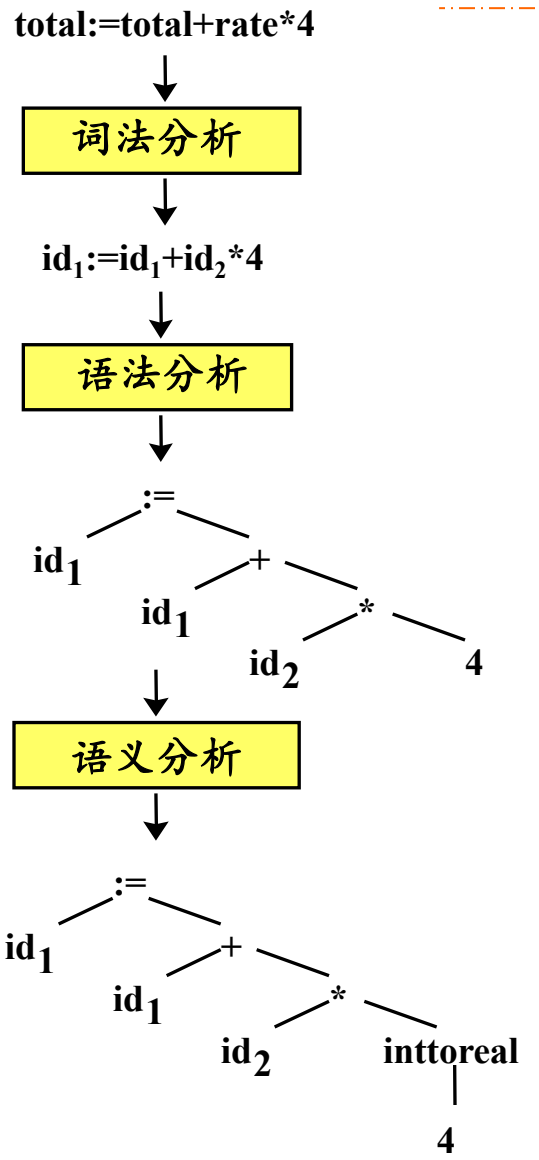


- 插入转换符的语法树



wenshli@bupt.edu.cn

total:=total+rate*4 的各分析步骤及其中间结果



wenshli@bupt.edu.cn

二、综合阶段

- 任务：
  根据源语言到目标语言的映射关系，对分析阶段产生的中间形式进行综合加工，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

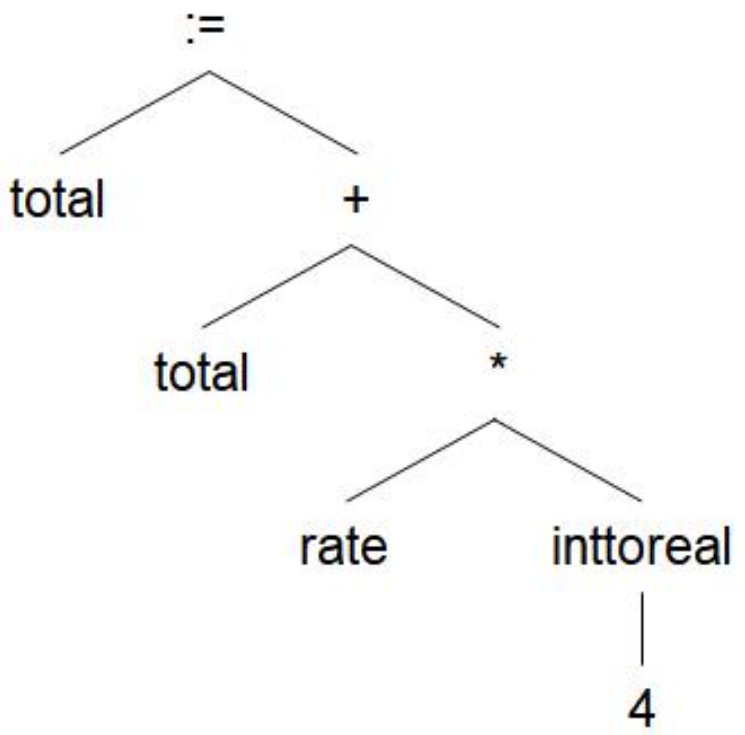
wenshli@bupt.edu.cn

4. 中间代码生成

- 中间代码：一种抽象的机器程序
- 中间代码应具有的两个重要特点：
  - 易于产生
  - 易于翻译成目标代码
- 中间代码有多种形式
- 三地址代码具有的特点：
  - 每条指令除了赋值号之外，最多含有一个运算符。
  - 编译程序必须生成临时变量名，以便保留每条指令的计算结果。
  - 有些“三地址”指令少于三个操作数

wenshli@bupt.edu.cn

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:=$temp_3$

wenshli@bupt.edu.cn

5. 代码优化

- 代码优化：
  对代码进行改进：占用空间少、运行速度快。
- 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:=$temp_2$

- 优化编译程序：能够完成大多数优化的编译程序。

wenshli@bupt.edu.cn

6. 目标代码生成

- 目标代码：
  - 可重定位的机器代码
  - 汇编语言代码
- 涉及到的两个重要问题：
  - 对程序中使用的每个变量要指定存储单元
  - 对变量进行寄存器分配

wenshli@bupt.edu.cn

total:=total+rate*4 的目标代码

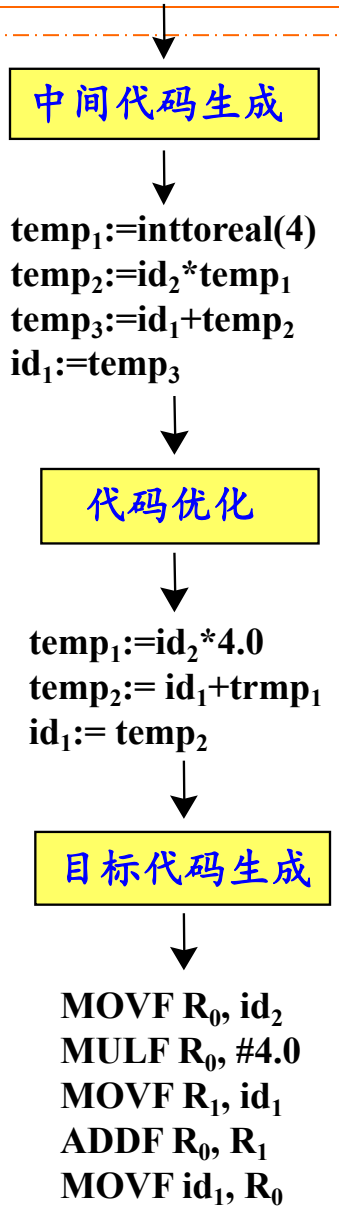
$temp_1 := id_2 * 4.0$
$temp_2 := id_1 + temp_1$
$id_1 := temp_2$

MOVF R_0, id_2
MULF R_0, #4.0
MOVF R_1, id_1
ADDF R_1, R_0
MOVF id_1, R_1

wenshli@bupt.edu.cn

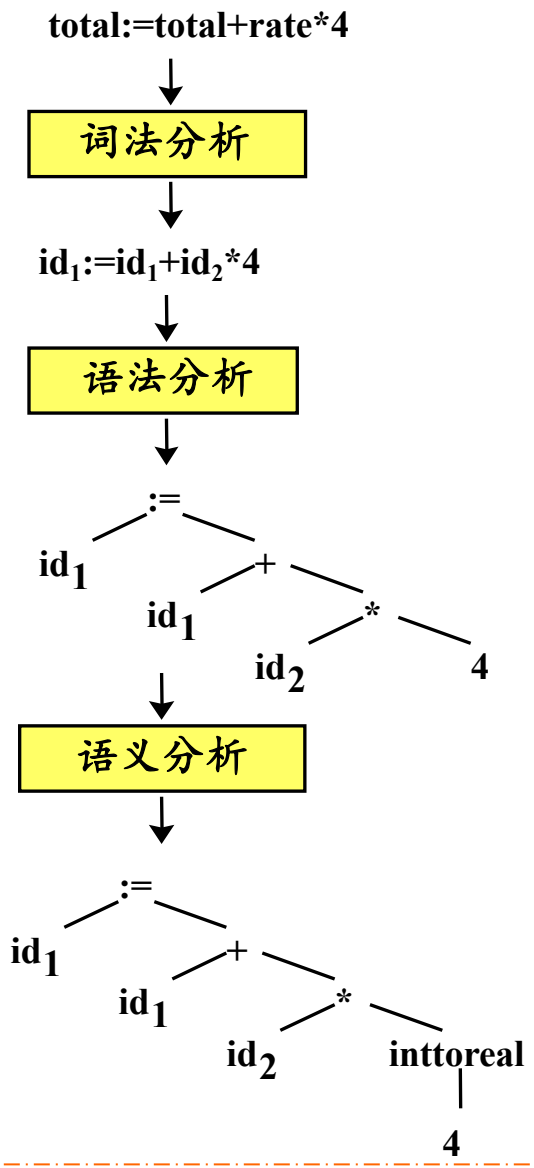
赋值语句total:=total+rate*4

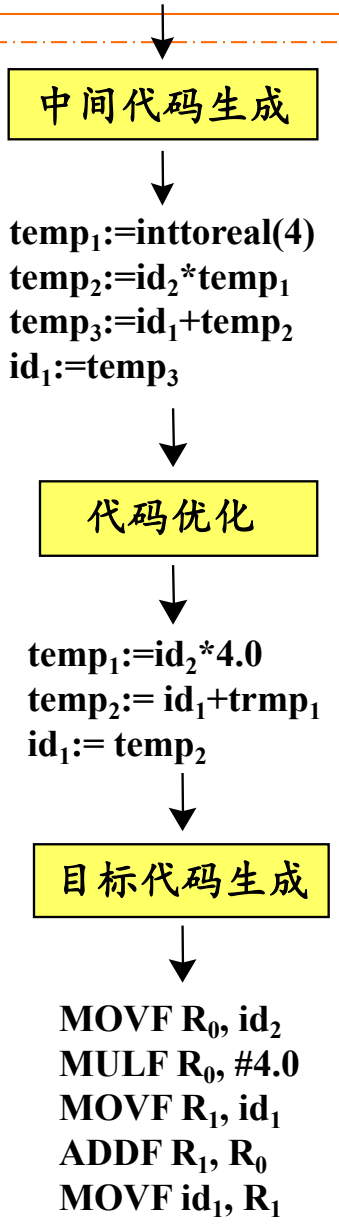
- 各综合步骤及结果：



wenshli@bupt.edu.cn

total:=total+rate*4 的翻译过程





wenshli@bupt.edu.cn

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
- 符号表：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
- 要求：
  - 快速地找到标识符的记录
  - 可存取数据
- 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。

wenshli@bupt.edu.cn

声明语句：float total, rate;

- 词法分析程序：
  - 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
- 语义分析程序：
  - total、rate都表示变量
  - float表示这两个变量的类型
  - 可以把这两种属性填入符号表
  - 引用时，类型检查
- 在语义分析和生成中间代码时，还要在符号表中填入对变量进行存储分配的信息。

wenshli@bupt.edu.cn

四、错误处理

- 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记号流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。
- 处理与恢复
  - 判断位置和性质
  - 适当的恢复

wenshli@bupt.edu.cn

1.3 编译有关的其他概念

一、前端和后端

二、“遍”

wenshli@bupt.edu.cn

一、前端和后端

- 前端：与源语言有关而与目标机器无关的部分
  - 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
- 后端：与目标机器有关的部分
  - 目标代码的生成、与机器有关的代码优化
  - 相应的错误处理和符号表操作
- 划分前端和后端的优点：
  - 便于编译程序的移植
  - 便于编译程序的构造

wenshli@bupt.edu.cn

二、遍

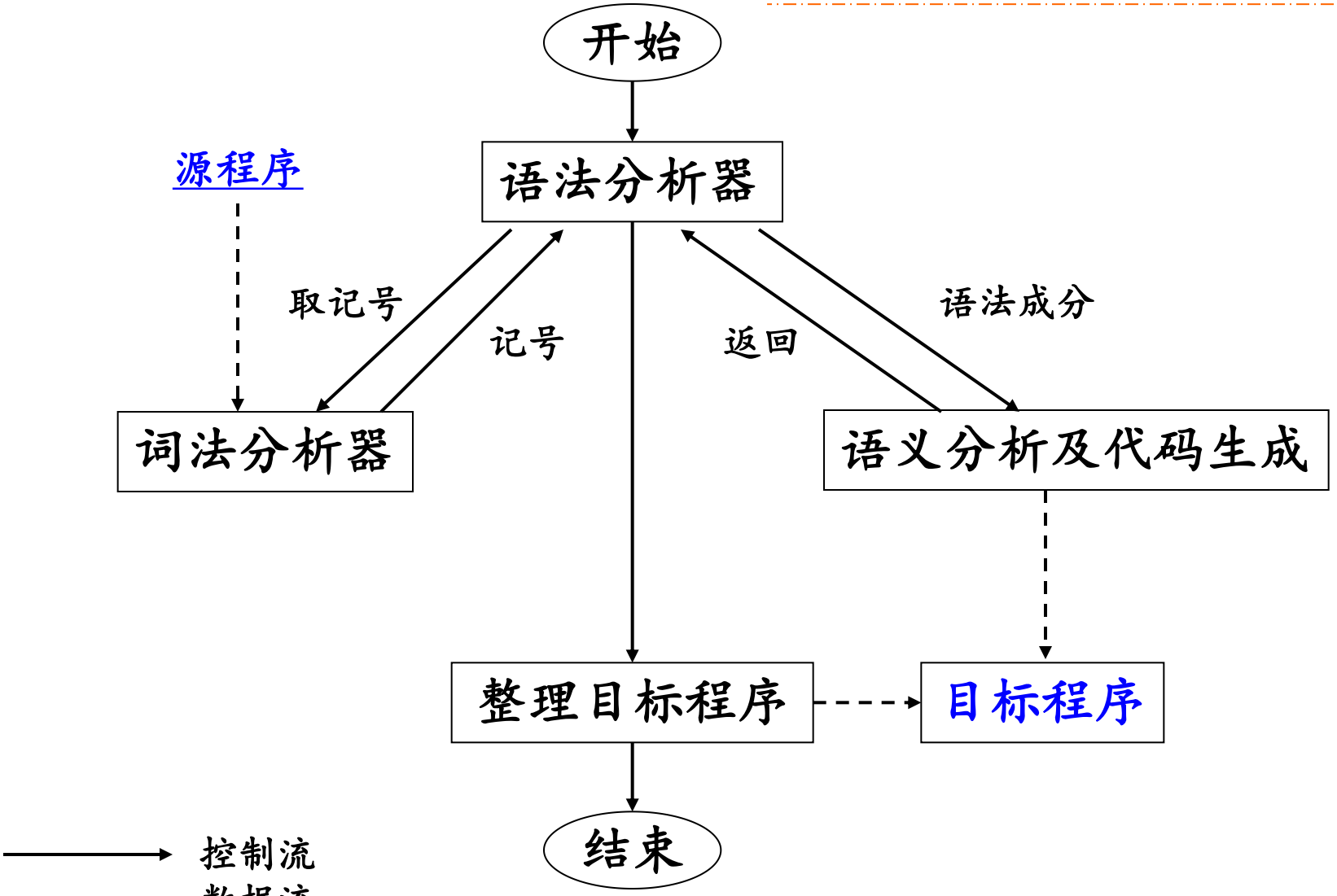
- 一“遍”：
  对源程序或其中间形式从头到尾扫描一遍，并作相关的加工处理，生成新的中间形式或目标程序。
- 编译程序的结构受“遍”的影响
  - 遍数
  - 分遍方式

一遍扫描的编译程序

多遍编译程序

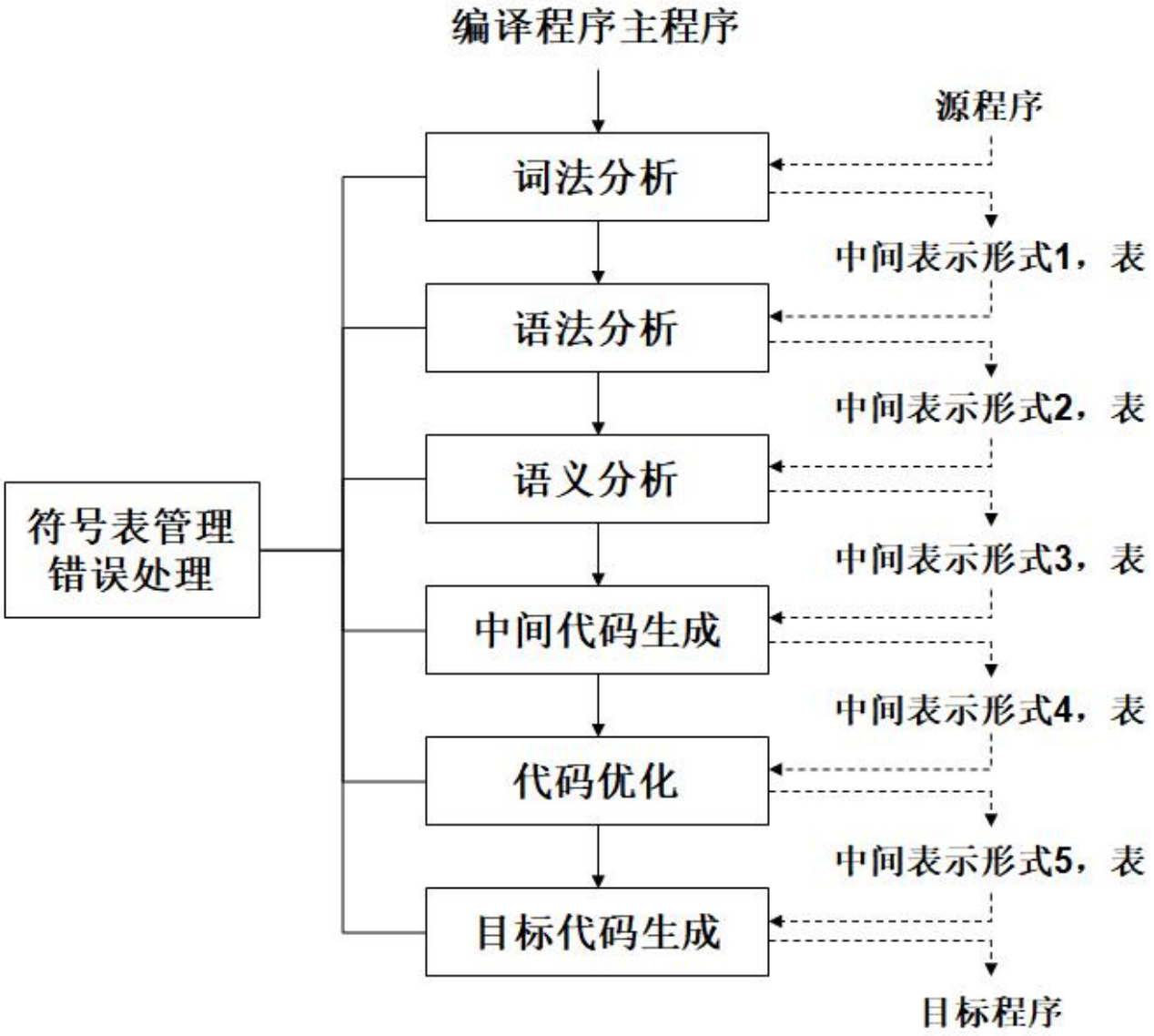
wenshli@bupt.edu.cn

一遍扫描的编译程序



wenshli@bupt.edu.cn

多遍编译程序



wenshli@bupt.edu.cn

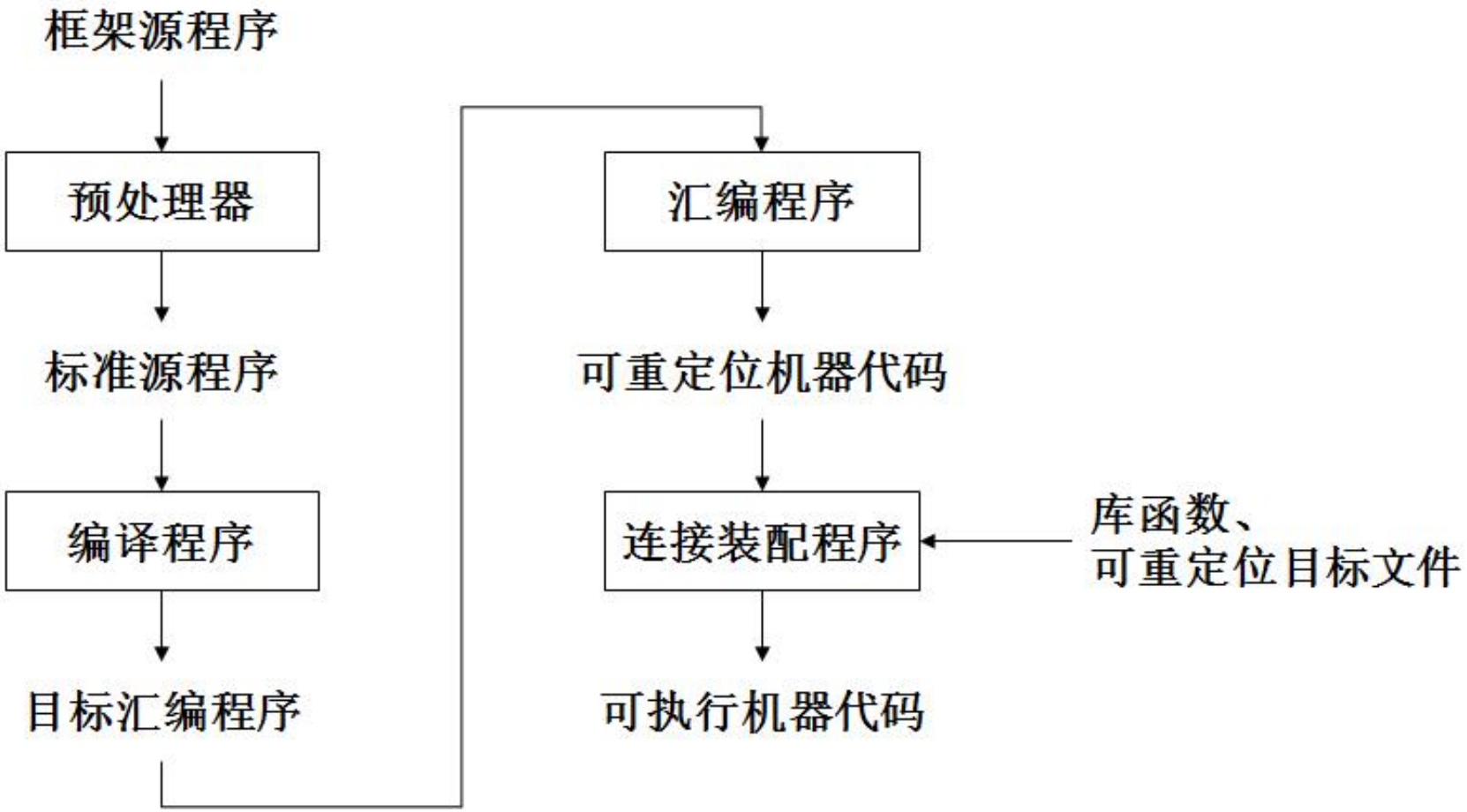
编译程序分遍的优缺点

- 好处：
  - 减少对主存容量的要求；
  - 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；
  - 更充分的优化工作，获得高质量目标程序；
  - 为编译程序的移植创造条件。

- 缺点：
  - 增加了不少重复性的工作。

wenshli@bupt.edu.cn

1.4 编译程序的伙伴工具



wenshli@bupt.edu.cn

编译程序的前后处理器

一、预处理器

二、汇编程序

三、连接装配程序

wenshli@bupt.edu.cn

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

wenshli@bupt.edu.cn

1. 宏处理器

- 预处理器允许用户在源程序中定义宏。
- C语言源程序中的一个宏定义：

  #define prompt(s) fprintf(stderr, s)
- 宏处理器处理两类语句：宏定义和宏调用
- 宏定义
  - 用统一的字符或关键字表示，如define或macro
  - 由宏名字及宏体组成
  - 允许在宏定义中使用形参
- 宏调用
  - 由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。

wenshli@bupt.edu.cn

2. 文件包含

- 预处理器把文件的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。

wenshli@bupt.edu.cn

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：如源语言中没有while/if-then-else语句：
  - 预处理器可以将类似于while或if-then-else语句结构的内部宏提供给用户使用。
  - 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。

wenshli@bupt.edu.cn

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 R_1, a

  ADD R_1, #2

  MOV b, R_1

- 最简单的汇编程序对输入作两遍扫描。

wenshli@bupt.edu.cn

第一遍

- 找出标志存储单元的所有标识符，并将它们写入汇编符号表中。
  - 汇编符号表独立于编译程序的符号表
- 在符号表中记录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：32位机器，一个字由4个字节组成，每个整型变量占一个字，完成第一遍扫描后，得到汇编符号表：

标识符	地址
a	0
b	4

wenshli@bupt.edu.cn

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对应的地址。

- 输出：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。

wenshli@bupt.edu.cn

可重定位机器代码

b:=a+2的汇编代码

MOV R_1, a
ADD R_1, #2
MOV b, R_1

- 假定机器指令的格式为：

  操作符 寄存器 寻址模式 地址

- 假定：

  0001 代表 MOV R, S

  0011 代表 ADD

  0010 代表 MOV D, R

- 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*

wenshli@bupt.edu.cn

绝对机器代码

可重定位机器代码：

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

- 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

wenshli@bupt.edu.cn

三、连接装配程序

- 可重定位目标程序的组成：
  - 正文，目标程序的主要部分，包括指令代码和数据；
  - 外部符号表(也称全局符号表)，记录有本程序段引用的名字和被其他程序段引用的名字；
  - 重定位信息表，记录有重定位所需要的有关信息。
- 连接装配程序
  - 作用：把多个经过编译或汇编的目标模块连接装配成一个完整的可执行程序。
  - 连接编辑程序：扫描外部符号表，寻找所连接的程序段，根据重定位信息表解决外部引用和重定位，最终将整个程序涉及到的目标模块逐个调入内存并连接在一起，组合成一个待装入的程序。
  - 重定位装配程序：把目标模块的相对地址转换成绝对地址。

wenshli@bupt.edu.cn

hello 程序是如何被编译出来的？

- hello.c

```
#include<stdio.h>
int main(int argc,char *argv[])
{
    printf("Hello World!");
    return 0 ;
}
```

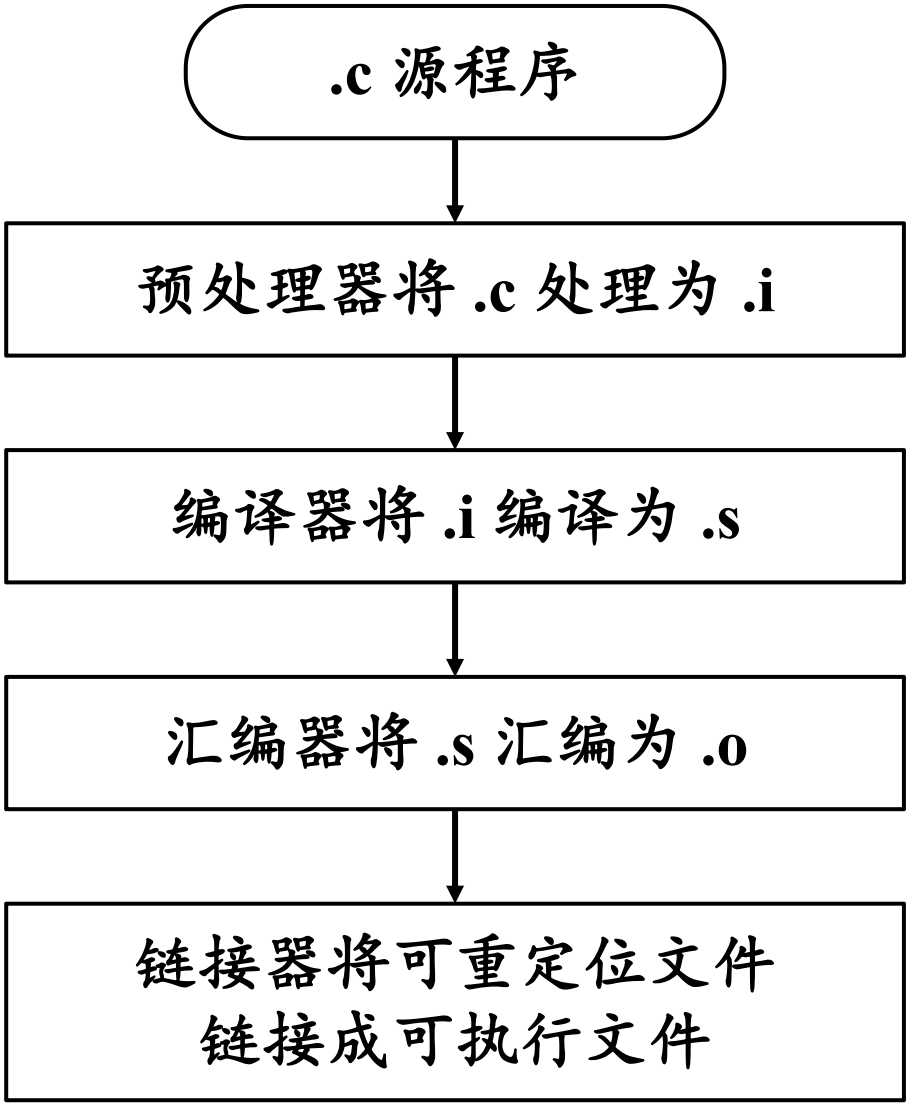
- 编译并运行:

```
gcc  -o  helloWorld  helloWorld.c

./helloWorld

Hello World!
```

wenshli@bupt.edu.cn

hello程序是如何被编译出来的？

- 编译过程：



wenshli@bupt.edu.cn

hello程序是如何被编译出来的？

- 预处理：

```
gcc  -E  -o  helloWorld.i  helloWorld.c
```

  - -E ： 只进行预处理
  - 预处理之后的输出文件：helloWorld.i

- 编译：

```
gcc  -S  -o  helloWorld.s  helloWorld.i
```

  - -S ： 生成汇编
  - 输出文件：helloWorld.s
  - 查看输出文件内容
    cat helloWorld.s

wenshli@bupt.edu.cn

hello程序是如何被编译出来的？

- 汇编：

```
gcc  -o  helloWorld.o   -c helloWorld.s
```

  - 输出机器指令程序
  - 输出文件：helloWorld.o
  - 查看二进制内容：
    od helloWorld.o

- 链接：

```
gcc -o helloWorld  helloWorld.o
```

  - Printf 函数，存在于libc.so或libc.a中
  - 生成的 helloWorld 程序，在linux下，是一种ELF格式的文件
  - 查看 helloWorld程序链接的系统库：
    ldd helloWorld

wenshli@bupt.edu.cn

用g++ 编译 C++ 程序

```
#include<iostream>
Using namesoace std;
int main()
{
    cout<<"Hello World!"<<endl;
    return 0 ;
}
```

- 预处理：g++ -E main.cpp -o main.i
- 编译：g++ -S main.i -o main.s
- 汇编：g++ -c main.s -o main.o
- 链接：g++ main.o

wenshli@bupt.edu.cn

1.5 编译原理的应用

- 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

wenshli@bupt.edu.cn

语法制导的结构化编辑器

- 具有通常的正文编辑和修改功能
- 能象编译程序那样对源程序进行分析，把恰当的层次结构加在程序上。
- 可以保证源程序
  - 无语法错误
  - 有统一的可读性好的程序格式
- 结构化编辑器能够执行一些对编制源程序有用的附加任务，如：
  - 检查用户的输入是否正确
  - 自动提供关键字
  - 从BEGIN或左括号跳到与其相匹配的END或右括号

wenshli@bupt.edu.cn

程序格式化工具

- 读入并分析源程序
- 使程序结构变得清晰可读，如：
  - 用缩排方式表示语句的嵌套层次结构
  - 用一种专门的字型表示注释等

wenshli@bupt.edu.cn

软件测试工具

静态测试 ——静态测试器

读入源程序
在不运行该程序的情况下对其进行分析，以发现程序中潜在的错误或异常。

- 不可能执行到的死代码
- 未定义就引用的变量
- 试图使用一个实型变量作为指针等。

动态测试 ——动态测试器

利用测试用例实际执行程序
记录程序的实际执行路线
将实际运行结果与期望结果进行比较，以发现程序中的错误或异常。

wenshli@bupt.edu.cn

程序理解工具

- 对源程序进行分析
- 确定各模块间的调用关系
- 记录程序数据的静态属性和结构属性
- 画出控制流程图

wenshli@bupt.edu.cn

高级语言的翻译工具

- 将用某种高级语言开发的程序翻译为另一种高级语言表示的程序

wenshli@bupt.edu.cn

编译技术的其他应用

- 高级程序设计语言的实现
  - 编译技术和语言发展互动的一个早期的例子：C语言中的关键字register
- 针对计算机体系结构的优化
  - 并行性
    - 现代微处理器采用的指令级并行性，对程序员透明，硬件自动检测顺序指令流之间的依赖关系，一个硬件调度器可以改变指令的执行顺序。
    - 处理器并行
  - 内存层次结构
    - 高效利用寄存器、高速缓存
- 新计算机体系结构的设计
  - RISC
  - 专用体系结构

wenshli@bupt.edu.cn

本章小结

- 什么是编译
- 翻译程序：
  - 编译程序（编译程序、汇编程序）、解释程序
  - 二者的异同
- 编译系统（编译环境）
- 编译程序的伙伴工具、功能及工作原理
  - 预处理器
  - 汇编程序
  - 连接装配程序
- 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
- 编译程序的设计涉及到的知识

wenshli@bupt.edu.cn

作业

- 1.1
- 1.2
- 1.4
- 1.5
- 1.6
- 1.7

wenshli@bupt.edu.cn