# 编译原理与技术

中国科学技术大学
计算机科学与技术学院
张 昱、陈意云
**0551-63603804, 63607043**
**yuzhang, yiyun@ustc.edu.cn**

# 课程简介

课程内容

- 介绍编译器构造的一般原理和基本实现方法
- 包括的理论知识：形式语言和自动机理论、语法制导的定义等

- 课程特点
  - 强调对编译原理和技术的宏观理解，不把注意力引导到理论和一些枝节算法上
  - 不偏向于任何源语言或目标机器
  - 鼓励用所学的知识去分析和解决实际问题

# 课程简介

学习意义

- 理解编程语言的设计和实现，了解和编程语言有关的理论

  对宏观把握编程语言来说，起到奠基的作用

- 从软件工程看，编译器是一个很好的实例，所介绍的概念和技术能应用到一般的软件设计之中
- 编译技术的应用和编译技术的发展

  高级语言设计、计算机系统结构的优化（并行、内存分层）、新型计算机系统结构的设计、程序翻译、提高软件开发效率的工具、高可信软件

# 课程简介

教材和参考书

- 张昱、陈意云，*编译原理与技术*，高等教育出版社，**2010**
- 张昱、陈意云，*编译原理实验教程*，高等教育出版社， **2009**
- **A. V. Aho, M. S. Lam, R. Sethi, and J. D. Ullman, *Compilers: Principles, Techniques, and Tools* , 2nd edition, Addison-Wesley, 2007**
- 陈意云、张昱，*编译原理习题精选与解析*，高等教育出版社，**2005**
- 教学资源网页：**http://staff.ustc.edu.cn/~yiyun**
  **http://staff.ustc.edu.cn/~yuzhang/compiler**

# 课 程 简 介

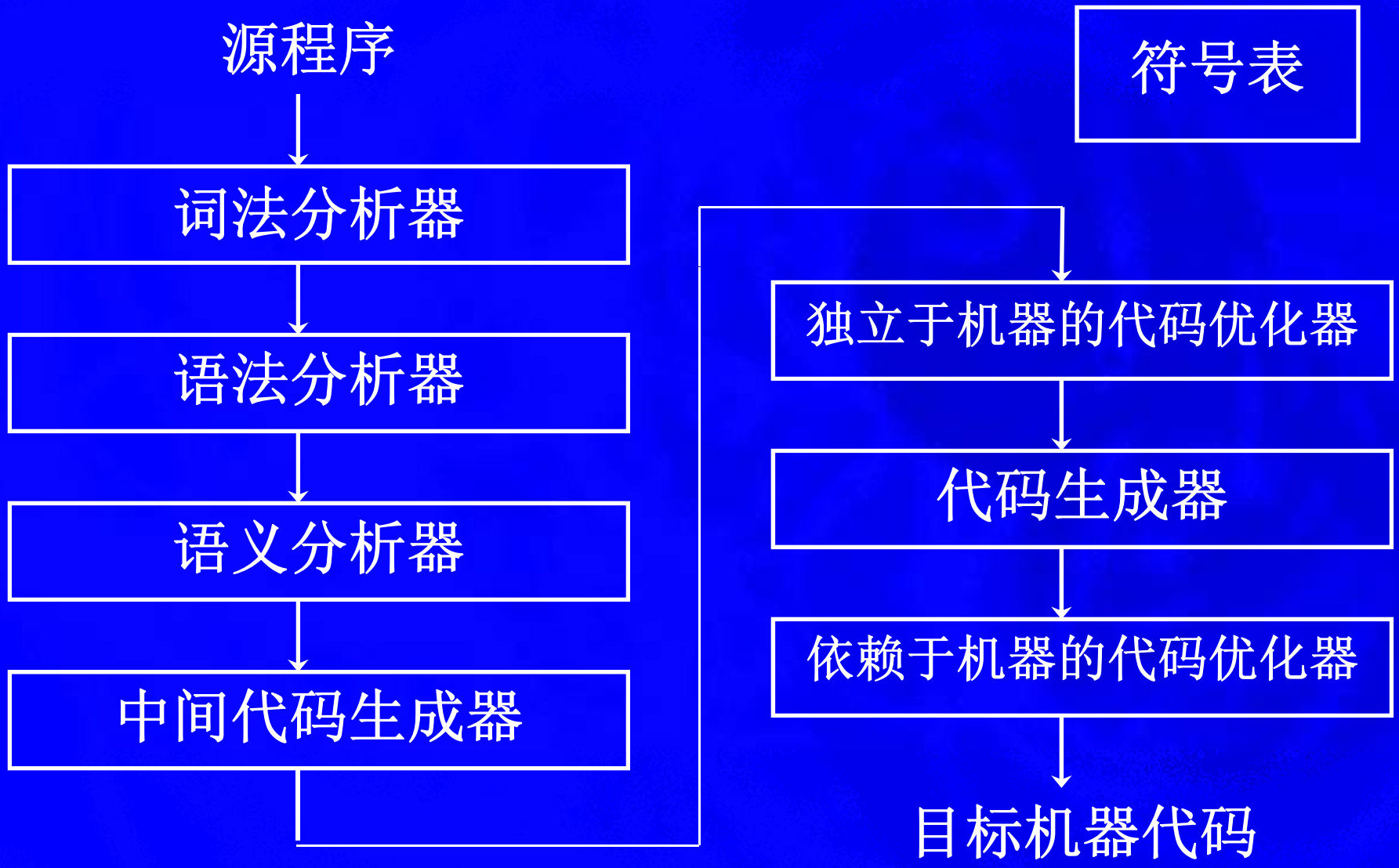
课程要求

- 质量上的目标：师生共同努力，达国内最好水平
- 讲课进展较快，平时不复习并加深理解，后面将听不懂
- 作业：少而精，每周交一次作业
- 课程设计：自己动手，丰产丰收
- 考试：开卷，灵活运用知识
- 学期总评 = 考试成绩占**60%**，作业占**10%**，课程设计**30%**
- 上课、设计、考试时间大体安排

# 第1章　引　论

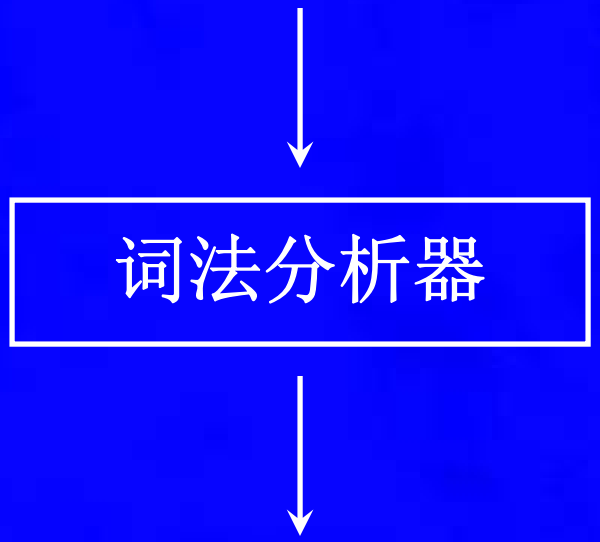
- 名词解释
  - 翻译器(translator)、编译器(compiler)
  - 解释器(interpreter)
- 编译器从逻辑上可以分成若干个阶段
- 每个阶段把源程序从一种表示变换成另一种表示
- 本章通过描述编译器的各个阶段来介绍编译这个课题

# 1.1 编译器概述


源程序
词法分析器
语法分析器
语义分析器
中间代码生成器
符号表
独立于机器的代码优化器
代码生成器
依赖于机器的代码优化器
目标机器代码

# 1. 1 编译器概述

**position = initial + rate * 60**




**⟨id, 1⟩⟨=⟩⟨id, 2⟩⟨+⟩⟨id, 3⟩⟨*⟩⟨60⟩**

符 号 表

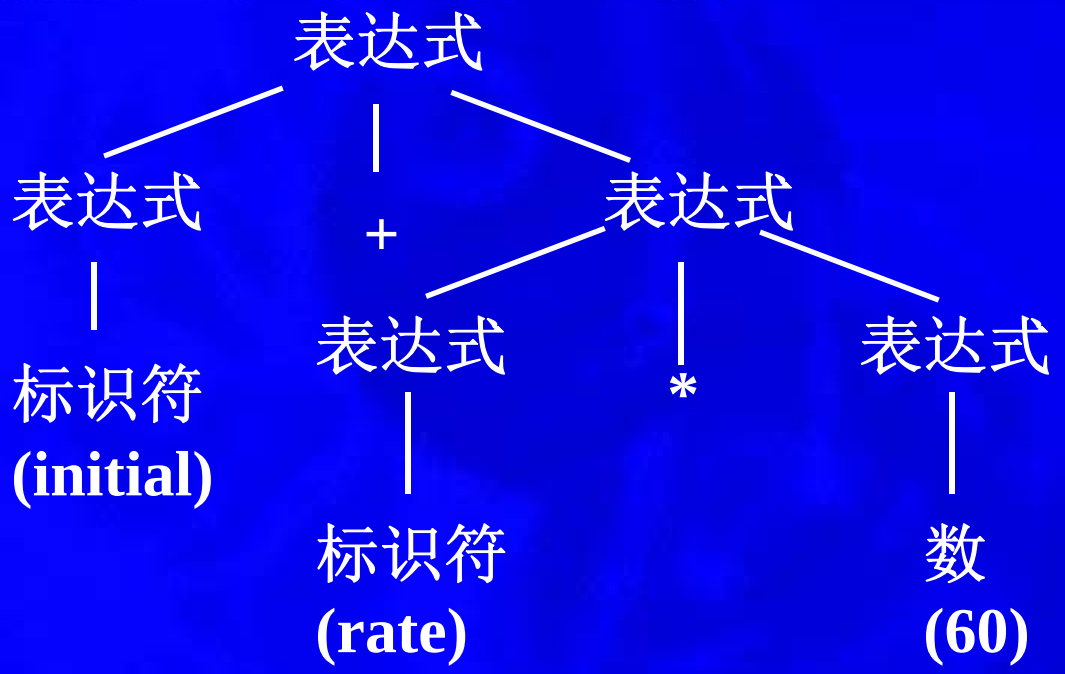
| | | |
|---|---|---|
| 1 | position | · · · |
| 2 | initial | · · · |
| 3 | rate | · · · |
| | | |

# 1.1 编译器概述

表达式的语法特征

- 任何一个标识符都是表达式
- 任何一个数都是表达式
- 如果$e_1$和$e_2$都是表达式，那么
  - $e_1 + e_2$
  - $e_1 * e_2$
  - $(e_1)$

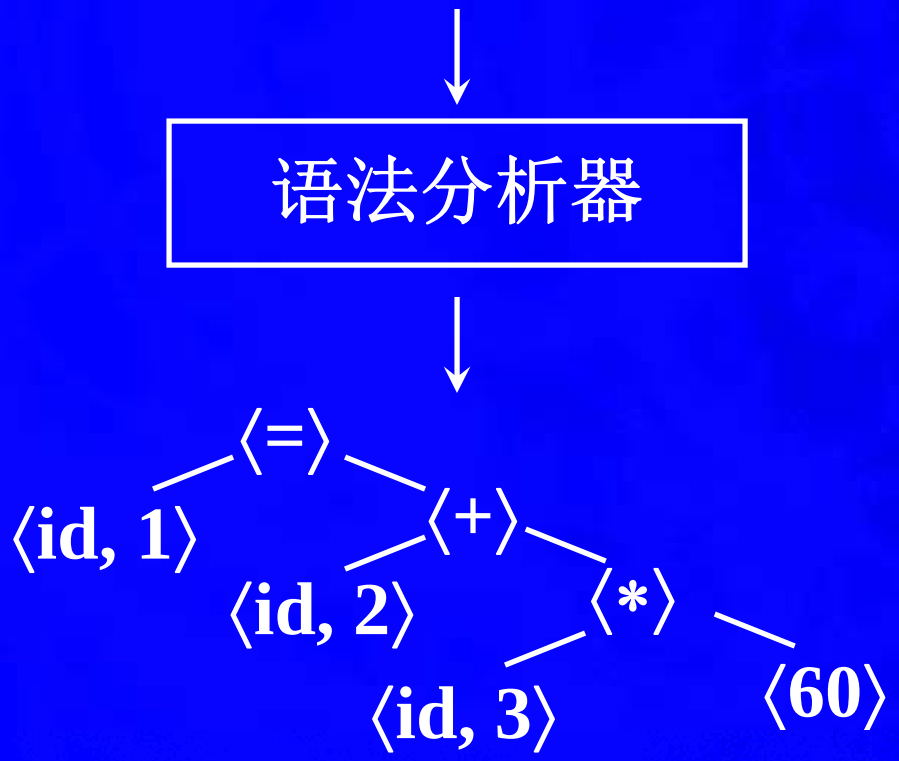
也都是表达式




**initial + rate * 60**的分析树

# 1. 1 编译器概述

⟨id, 1⟩ ⟨=⟩ ⟨id, 2⟩ ⟨+⟩ ⟨id, 3⟩ ⟨*⟩ ⟨60⟩

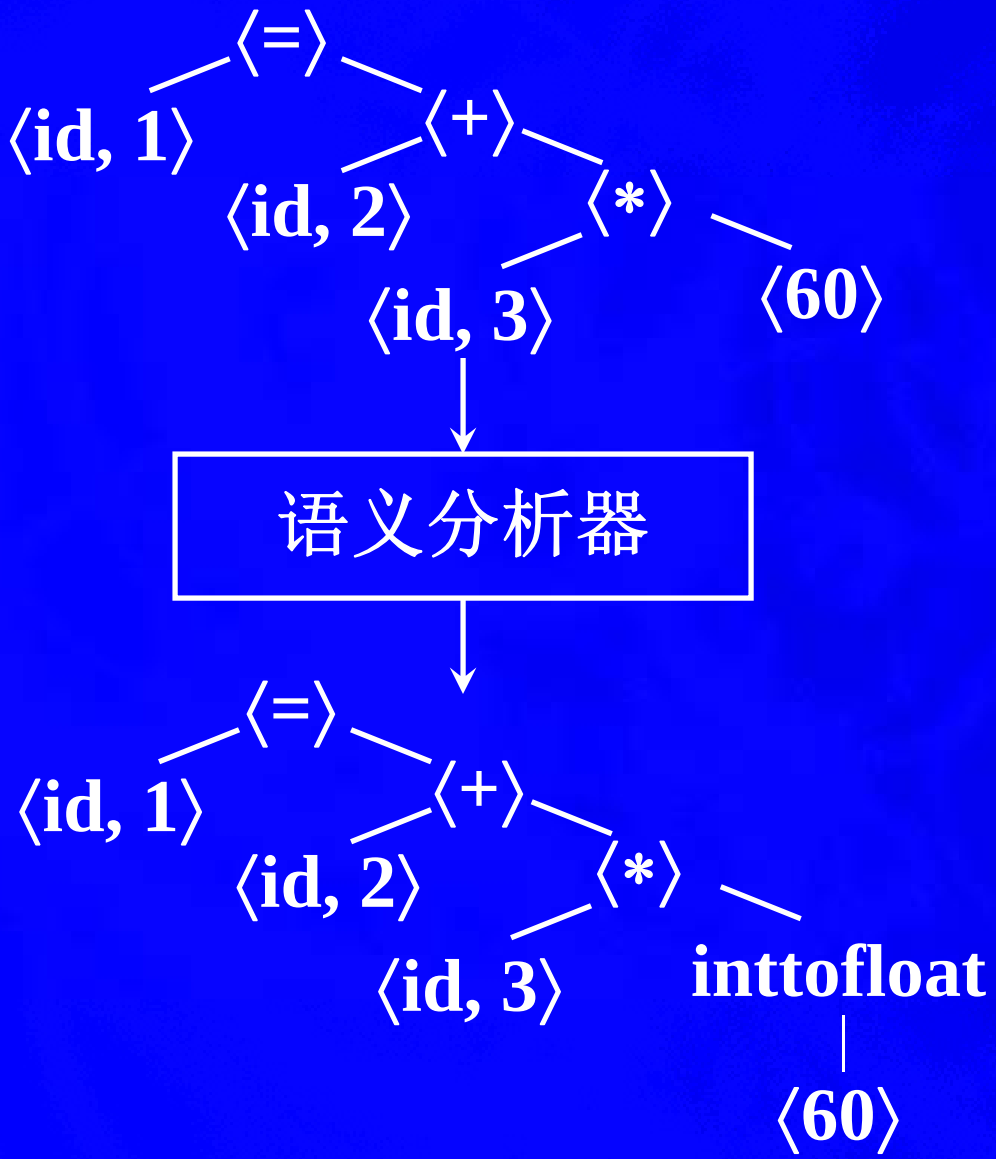



← 语法树

符 号 表

| | | |
|---|---|---|
| 1 | position | · · · |
| 2 | initial | · · · |
| 3 | rate | · · · |
| | | |

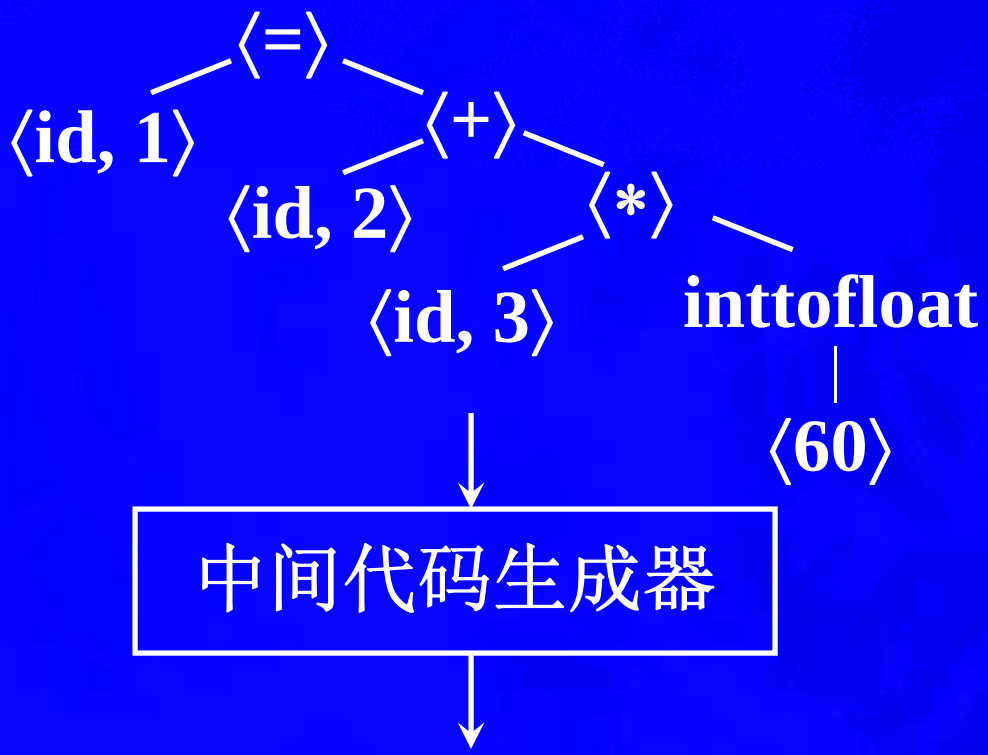
# 1. 1 编译器概述




符 号 表

| | | |
|---|---|---|
| 1 | position | · · · |
| 2 | initial | · · · |
| 3 | rate | · · · |
| | | |

# 1. 1 编译器概述




符 号 表

| | | |
|---|---|---|
| 1 | position | · · · |
| 2 | initial | · · · |
| 3 | rate | · · · |
| | | |

t1 = inttofloat(60)

t2 = id3 * t1

t3 = id2 + t2

id1 = t3

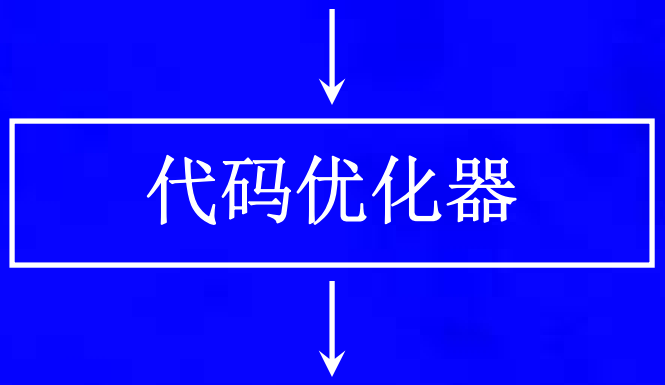
# 1. 1 编译器概述

t1 = inttofloat(60)

t2 = id3 * t1

t3 = id2 + t2

id1 = t3




t1 = id3 * 60.0

id1 = id2 + t1

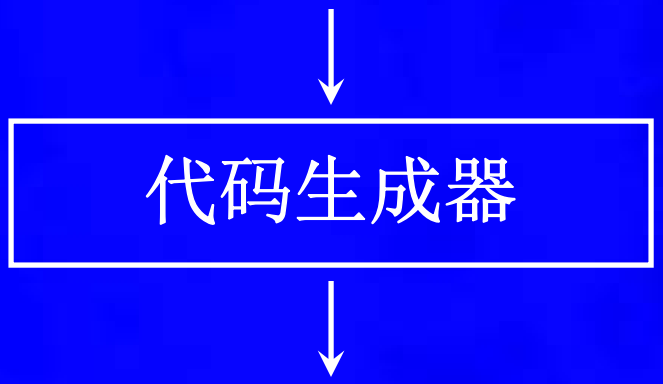
符 号 表

| | | |
|---|---|---|
| 1 | position | · · · |
| 2 | initial | · · · |
| 3 | rate | · · · |
| | | |

# 1. 1 编译器概述

**t1 = id3 * 60.0**

**id1 = id2 + t1**




**MOVF id3, R2**

**MULF #60.0, R2**

**MOVF id2, R1**

**ADDF R2, R1**

**MOVF R1, id1**

符 号 表

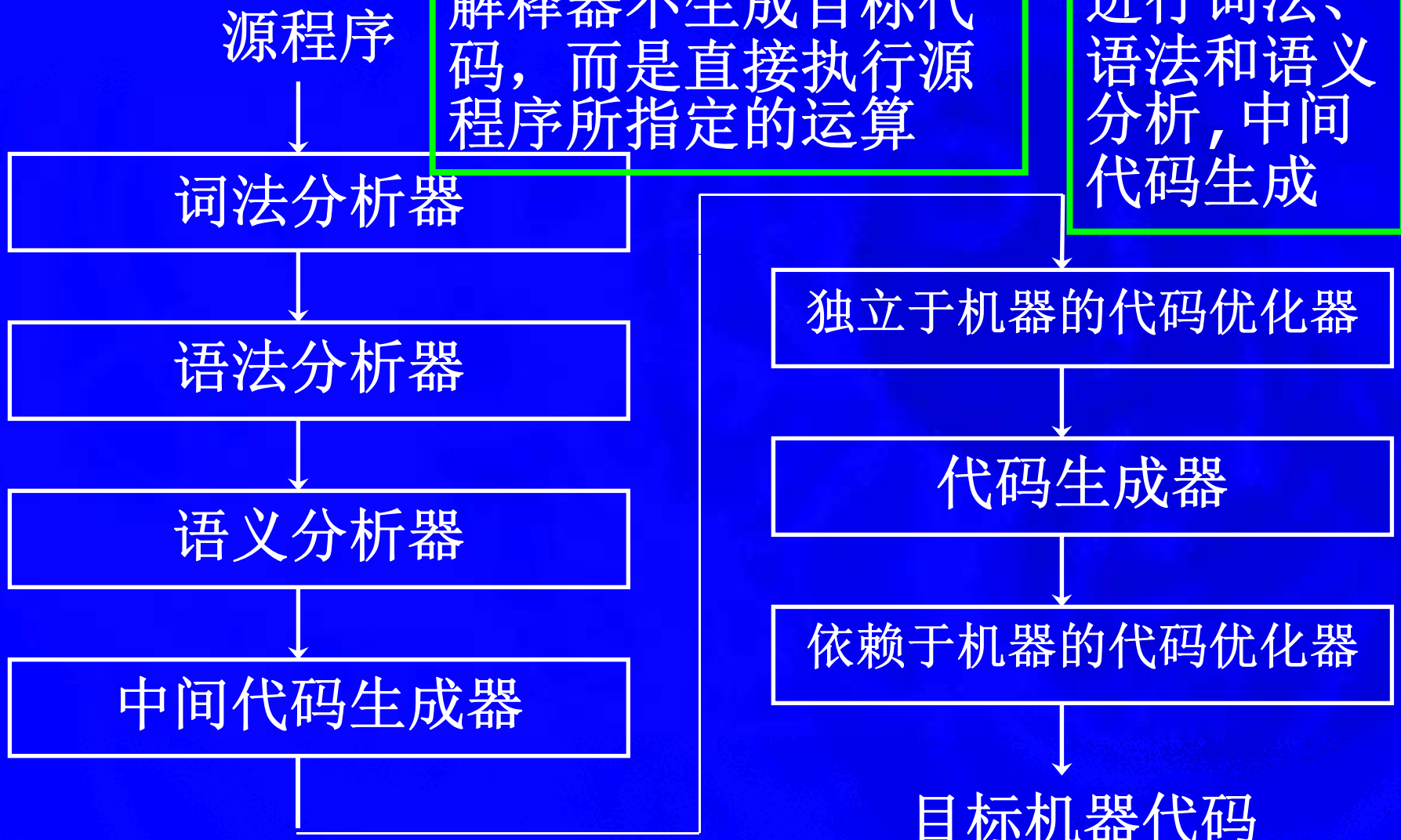
| | | |
|---|---|---|
| 1 | position | · · · |
| 2 | initial | · · · |
| 3 | rate | · · · |
| | | |

# 1.1 编译器概述

解释器和编译器的区别

解释器不生成目标代码，而是直接执行源程序所指定的运算

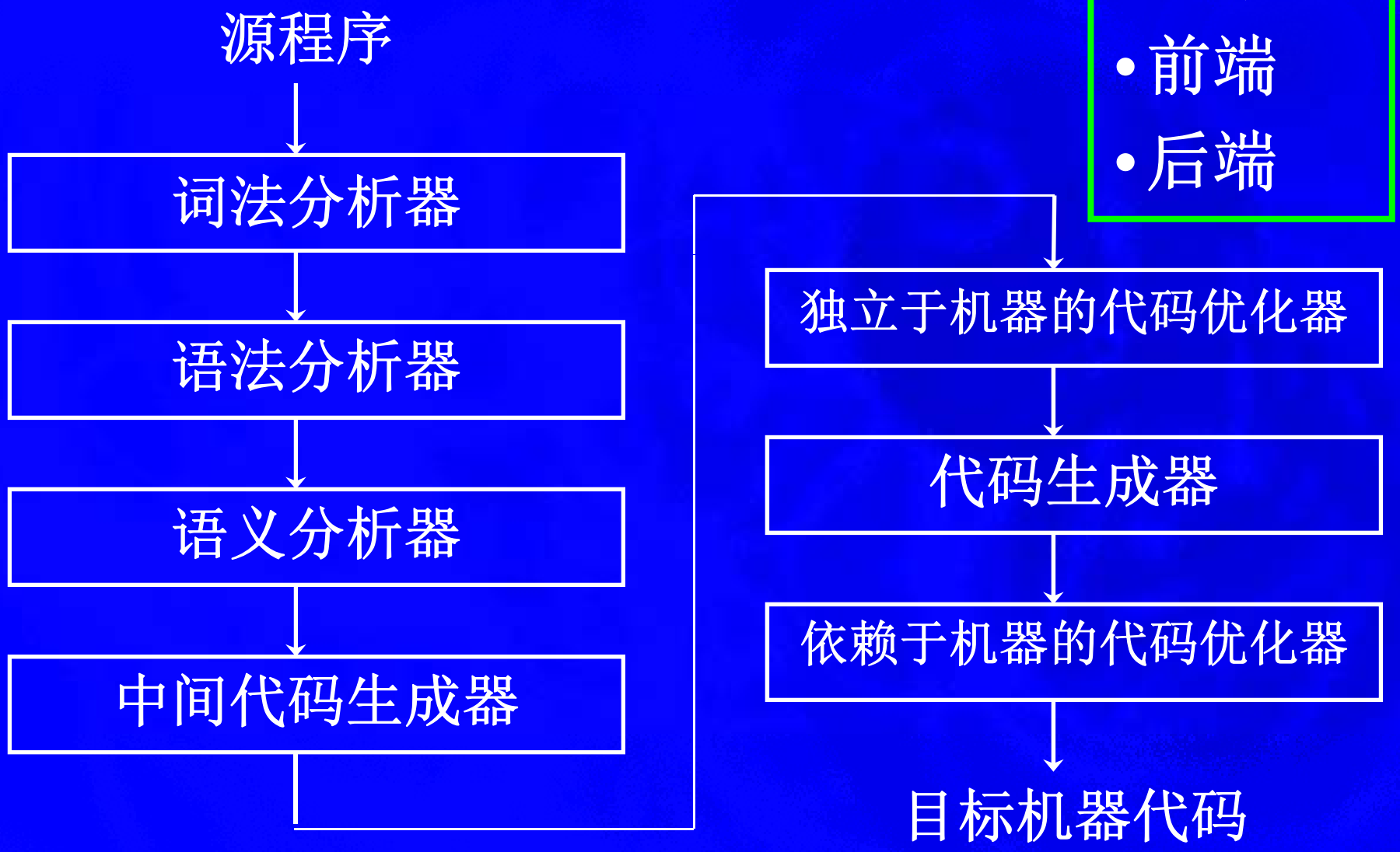
解释器也需要对源程序进行词法、语法和语义分析，中间代码生成



# 1.1 编译器概述

- **BASIC年代的解释器**
  - 功能：它将高级语言的源程序翻译成一种中间语言程序，然后对中间语言程序进行解释执行
  - 在那个年代，编译和解释两个功能是合在一个程序中，该程序被称为解释器
- **Java年代的解释器**
  - 解释器的上述两个功能分在两个程序中
  - 前一个叫做编译器，它把源程序翻译成一种叫做字节码的中间语言程序
  - 后一个叫做解释器，它对字节码程序进行解释执行

# 1.1 编译器概述



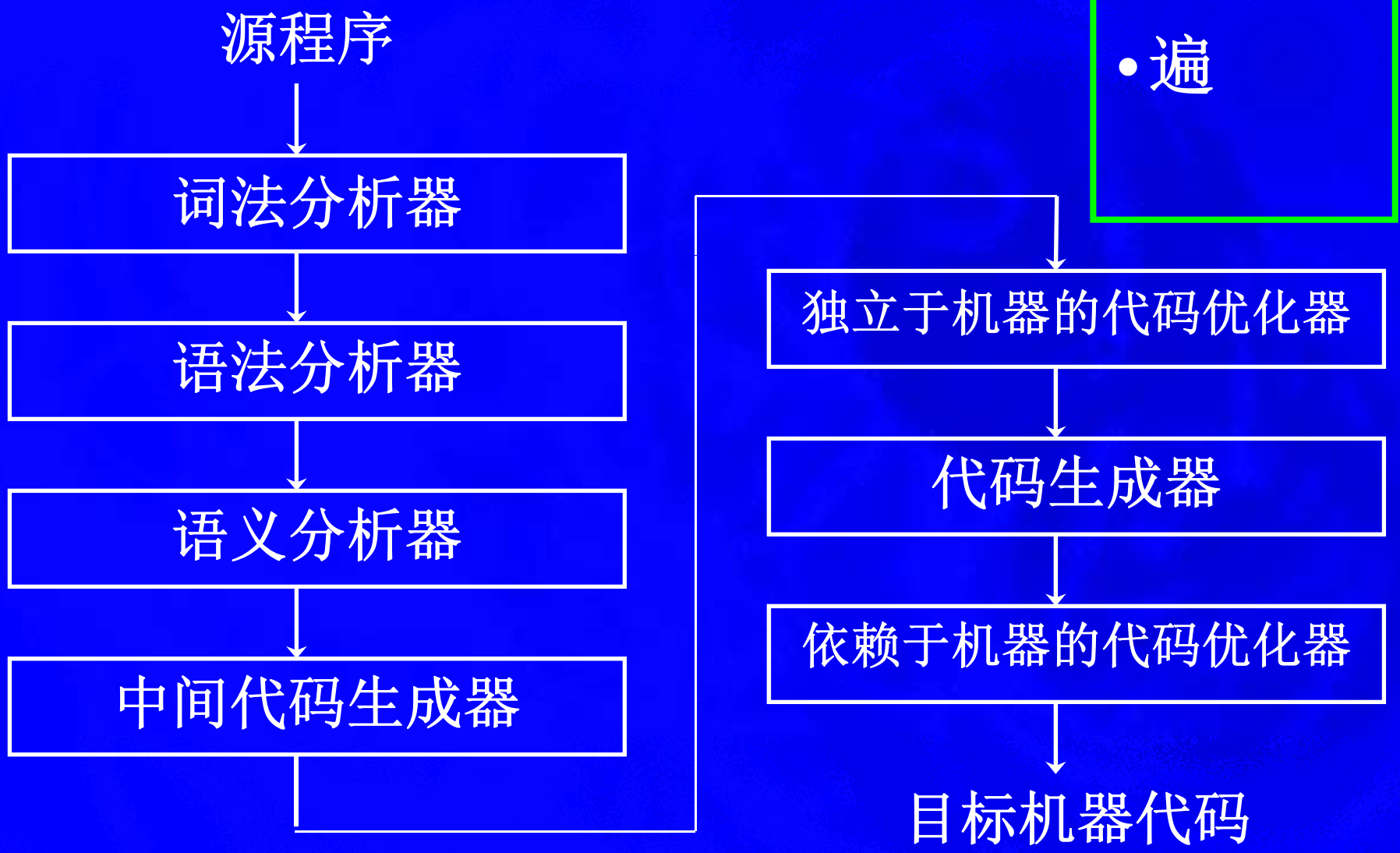

阶段分组

- 前端
- 后端

# 1.1 编译器概述

阶段分组

- 遍



# 1.2 编译器技术的应用

- 高级语言的实现
  - 高级编程语言易于编程，但程序运行较慢
  - 采用低级语言编程，可实施更有效的控制方式，得到更有效的代码，但难编写、易出错、难维护
  - 流行的编程语言大多数都是朝着提高抽象级别的方向演变
  - 每一轮编程语言新特征的出现都促进了编译器优化的新研究

# 1.2 编译器技术的应用

- 高级语言的实现

  每一轮编程语言新特征的出现都促进了编译器优化的新研究

  - 支持用户定义的聚合数据类型和高级控制流，如数组和记录、循环和过程调用：**C**、**Fortran**
  - 面向对象的主要概念是数据抽象和性质继承，使得程序更加模块化并易于维护：**Smalltalk**、**C++**、**C#**、**Java**
  - 类型安全的语言：**Java**没有指针，也不允许指针算术。它用无用单元收集机制来自动回收那些不再使用的对象占据的内存
  - **Java**设计能支持代码移植和代码移动

# 1.2 编译器技术的应用

- 针对计算机体系结构的优化
  - 计算机体系结构的迅速演化对编译器技术不断提出新的需要
  - 并行化
    - 编译器重新整理指令，使得指令级并行更有效
    - 编译器从传统的串行程序自动生成并行代码，使之运行于多处理器上
  - 内存分层
    - 编译器的优化历来集中在优化处理器的执行上，但是现在更强调要使内存分层更有效

# 1.2 编译器技术的应用

- 新计算机体系结构的设计
  - 现在计算机系统的性能不仅仅取决于它的原始速度，还取决于编译器是否能生成充分利用其特征的代码
  - 在现代计算机体系结构的研究中，在处理器的设计阶段就开发编译器，并将编译生成的代码在模拟器上运行，以评价拟采用体系结构的特征
  - 编译器技术影响计算机体系结构设计的一个著名例子是精简指令集计算机（**RISC**）的发明

# 1.2 编译器技术的应用

- 程序翻译
  - 二进制翻译

    编译器技术可用于把一种指令集的二进制代码翻译成另一种指令集的代码，以运行原先为别的指令集编译的代码
  - 数据库查询解释器

    数据库查询由一些谓词组成，这些谓词由包含关系运算的布尔表达式组成，可以被解释执行,也可以被编译成搜索数据库的命令

# 1.2 编译器技术的应用

- 提高软件开发效率的工具

  源于编译器优化的许多程序分析技术一直在改进软件开发的效率

  – 类型检查

    类型检查是一种捕捉程序中的不一致性的成熟而有效的技术

  – 边界检查

    数据流分析技术可用来定位缓冲区溢出

  – 内存管理

    自动的内存管理删除内存泄漏等内存管理错误