信息与软件工程学院

编译技术

主讲教师：陈安龙

自我介绍

- 姓　　名：陈安龙
- 主讲课程：软件类相关课程
- 电　　话：13688434459
- E-Mail: chenanlong@foxmail.com
- QQ课程群：2024-编译技术，454217875

课程的性质与任务

- 编译技术课程通常属于计算机科学或软件工程专业的高级课程，旨在教授学生如何设计、实现和优化编译器。
- 该课程涵盖了编程语言的词法分析、语法分析、语义分析、中间代码生成、代码优化和目标代码生成等内容。
- 在学习编译技术课程时，可以深入了解编程语言的内部工作原理，掌握编译器的设计和实现方法，以及优化编译器性能的技术。

课程的特点和目标

- 该课程的性质是理论性和实践性相结合的，既需要理解编译原理的基本概念和算法，又需要通过实际编写编译器或相关项目来加深对知识的理解和掌握。
- 通过学习编译技术或编译原理，可以提高对编程语言和计算机系统的理解，培养编写高效、可靠编译器的能力，为日后从事软件开发、系统设计等工作打下坚实的基础。

与编译技术相关的工作

- 编译工程师：负责设计、开发和优化编译器，需要具备扎实的编译原理知识和算法能力，能够设计高效的编译器和优化算法。
- 编程语言设计师：负责设计新的编程语言或对现有编程语言升级。需要了解编程语言的语法、语义和设计原则，能够设计出易于使用和高效的编程语言。
- 性能优化工程师：负责优化编译器生成的目标代码，包括代码优化、内存管理优化等。需要深入了解硬件架构和编译器优化技术，提高系统的性能和效率。
- 编译工具开发工程师：负责开发与编译技术相关的工具和软件，如调试器、性能分析工具等。需要具备良好的编程和设计能力，开发出高质量的工具软件。
- 编译技术研究员：负责研究新的编译技术和算法，探索未来编译技术的发展方向。需要具备扎实的理论基础和创新能力，能够推动编译技术领域的发展。

学习《编译技术》的现实意义

- **需求持续增长：**随着软件行业的快速发展，对高效、可靠的编译器需求不断增加。掌握编译技术可以提高就业的竞争力，满足企业对编译技术专业人才的需求。
- **技术创新驱动：**编译技术是软件开发的核心技术之一，对编译器的设计和优化能够推动软件业的技术创新。学习编译技术可以培养学生的创新意识和解决问题的能力。
- **提升软件质量：**学习编译技术可以帮助学生设计高效的编译器和优化算法，提升软件质量，满足企业对高质量软件的需求。
- **适应快速变化的技术环境：**学习编译技术可以让学生更好地适应技术变化，掌握核心技术，提高自身的适应能力和竞争力。
- **开拓职业发展空间：**掌握编译技术可以开拓更广阔的职业发展空间，如编译工程师、编程语言设计师、性能优化工程师等。这些职业领域在软件企业中需求持续增长，为学生提供更多的职业选择机会。

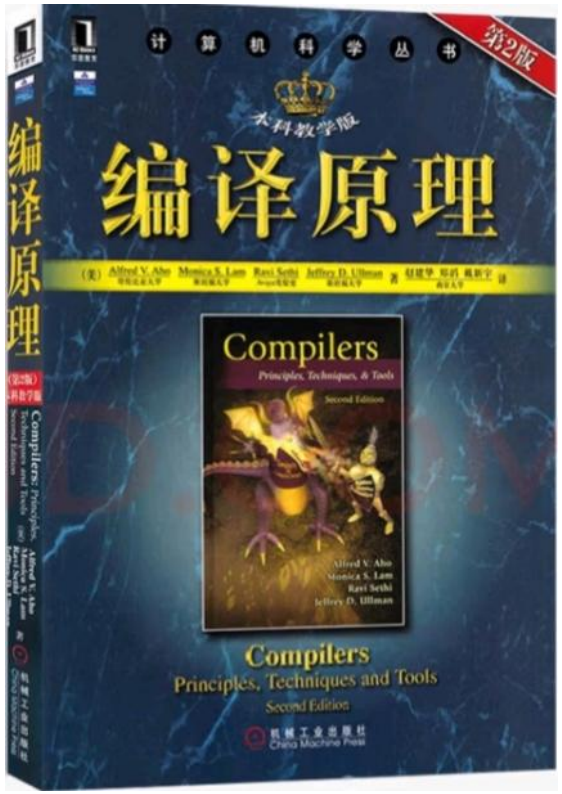
与编译器开发相关的工具

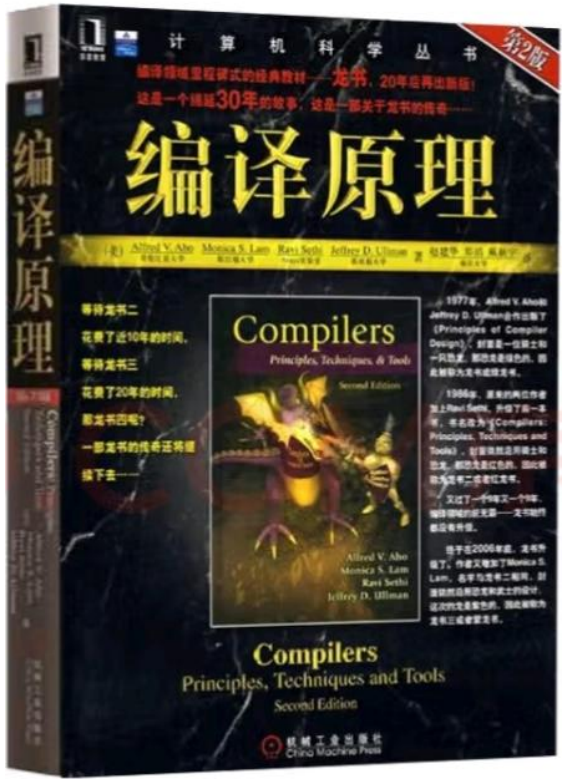
- **Flex 和 Bison：** Flex 用于生成词法分析器，Bison用于生成语法分析器。它们可以帮助开发人员快速构建编译器的前端部分。
- **LLVM：** LLVM 是一个开源的编译器基础软件，提供了丰富的编译器组件和工具，如中间表示(IR)、代码优化器、目标代码生成器等。开发人员可以使用 LLVM 来构建高性能的编译器。
- **GCC：** GCC 是一个流行的开源编译器套件，支持多种编程语言和平台。开发人员可以使用 GCC 来开发和测试编译器，学习编译器的设计和实现。
- **Clang：** Clang 是基于 LLVM 的 C/C++ 编译器前端，提供了高性能和可扩展的编译器前端实现。开发人员可以使用 Clang 来构建自定义的编译器前端。
- **IDE（集成开发环境）：** 开发编译器时，可以选择一些功能强大的集成开发环境，如Visual Studio、Eclipse、IntelliJ IDEA 等，来编写、调试和测试编译器代码。

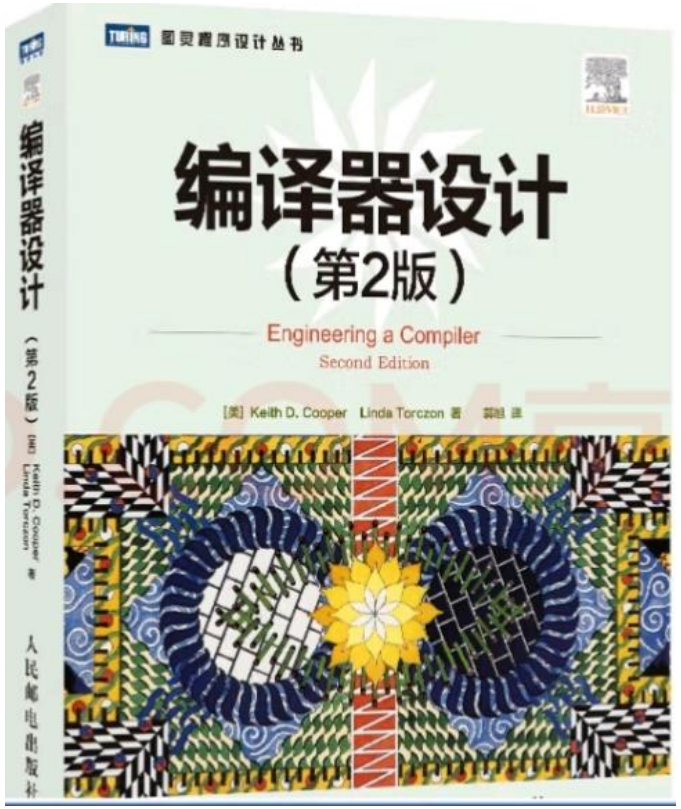
网络学习资料

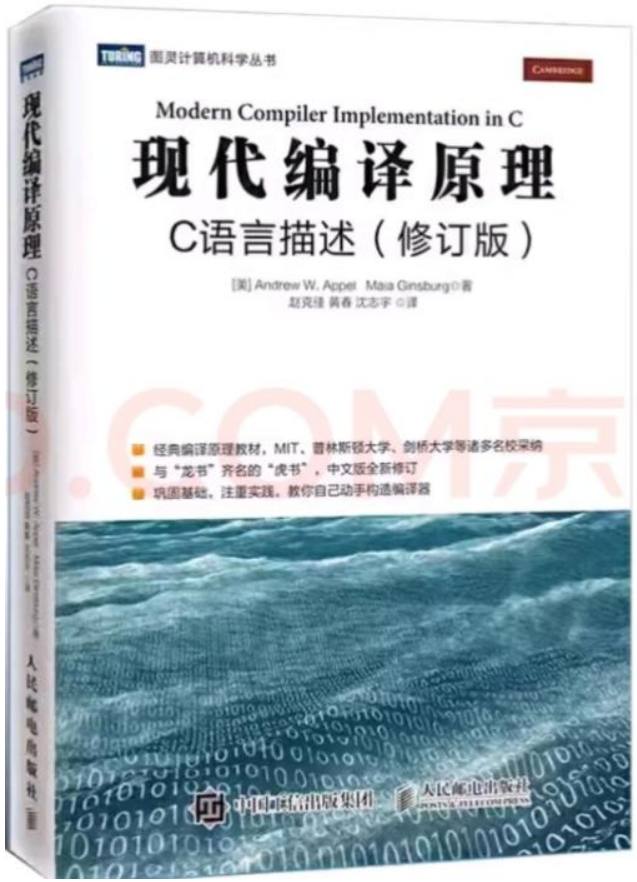
- **https://llvm.org/docs/LangRef.html**
- **https://llvm-tutorial-cn.readthedocs.io/en/latest/index.html**
- **https://kaleidoscope-llvm-tutorial-zh-cn.readthedocs.io/zh-cn/latest/**
- **https://llvm.org/**
- **https://releases.llvm.org/**
- **https://llvm.org/docs/tutorial/MyFirstLanguageFrontend/index.html**

课程教材及参考书

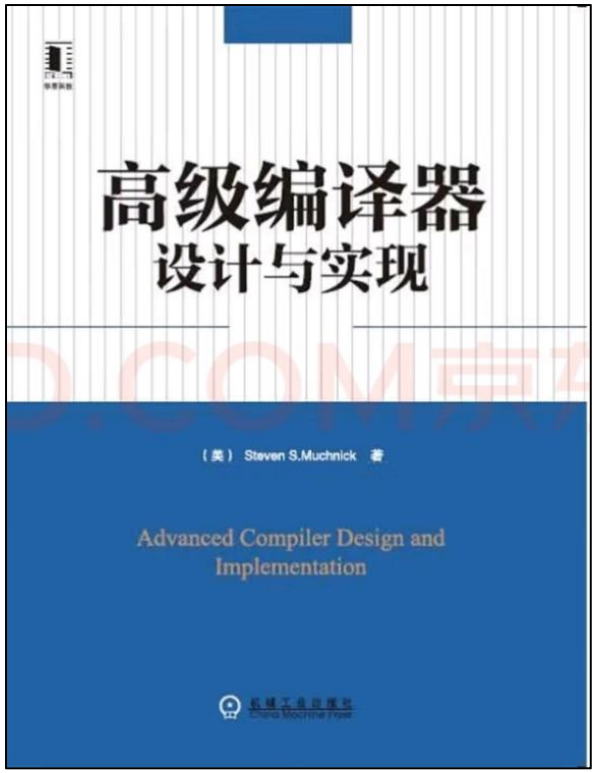

计 算 机 科 学 丛 书
第2版
本科教学版
编译原理
Compilers
Principles, Techniques, & Tools
Second Edition
Alfred V. Aho
Monica S. Lam
Ravi Sethi
Jeffrey D. Ullman
Compilers
Principles, Techniques and Tools
Second Edition
机械工业出版社
China Machine Press

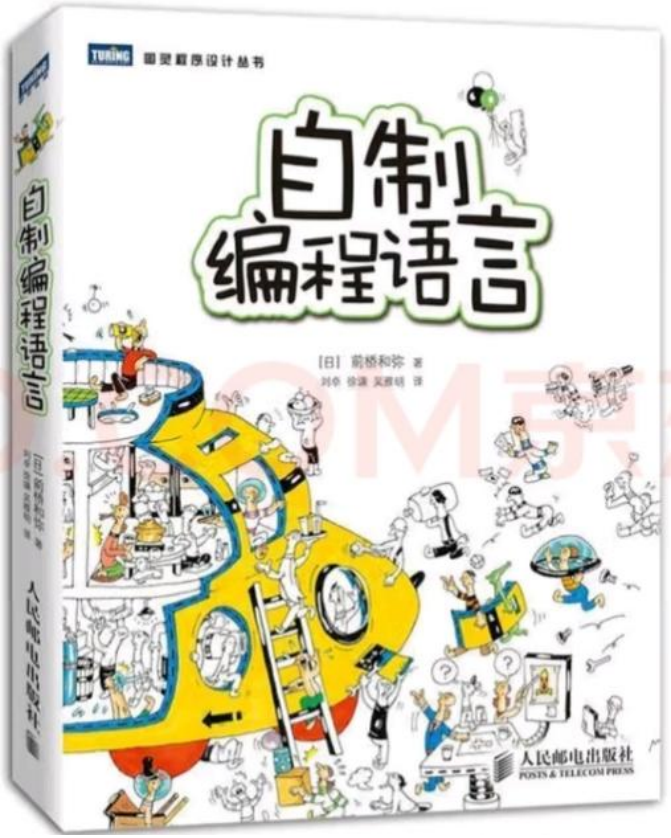

计 算 机 科 学 丛 书
第2版
编译原理
龙书，20年后再出新版！
这是一个持续30年的故事，这是一部关于龙书的传奇——
等待龙书二
花费了近10年的时间，
等待龙书三
花费了20年的时间，
那龙书四号？
一部龙书的传奇还将继续下去——
Compilers
Principles, Techniques, & Tools
Second Edition
Alfred V. Aho
Monica S. Lam
Ravi Sethi
Jeffrey D. Ullman
Compilers
Principles, Techniques and Tools
Second Edition
机械工业出版社
China Machine Press

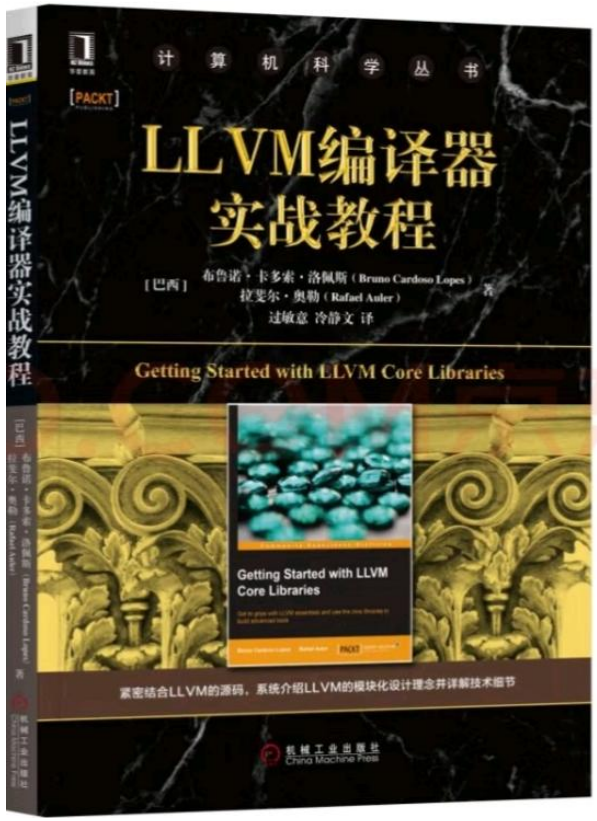

TURING 图灵程序设计丛书
编译器设计
（第2版）
Engineering a Compiler
Second Edition
[美] Keith D. Cooper Linda Torczon 著
人民邮电出版社

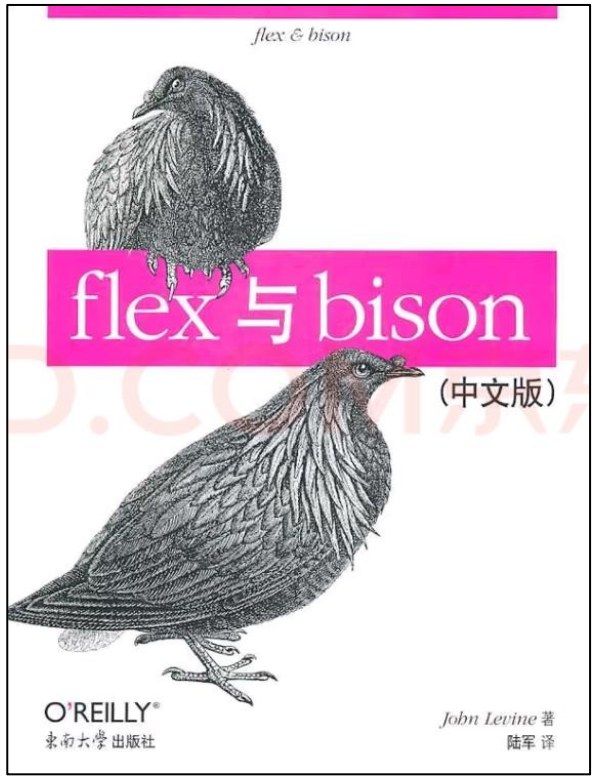

TURING 图灵计算机科学丛书
现代编译原理
C语言描述（修订版）
Modern Compiler Implementation in C
现代编译原理
C语言描述（修订版）
[美] Andrew W. Appel Maia Ginsburg 著
经典编译原理教材，MIT、普林斯顿大学、剑桥大学等诸多名校采纳
与“龙书”齐名的“虎书”，中文版全新修订
巩固基础，注重实践，教你自己动手构造编译器
中国邮电出版集团
人民邮电出版社

课程教材及参考书


高级编译器
设计与实现
[美] Steven S.Muchnick 著
Advanced Compiler Design and Implementation


TURING
图灵程序设计丛书
自制编程语言
[日] 前桥和弥 著
人民邮电出版社
POSTS & TELECOM PRESS


计 算 机 科 学 丛 书
[PACKT]
LLVM编译器实战教程
[巴西] 布鲁诺·卡多索·洛佩斯 (Bruno Cardoso Lopes)
拉斐尔·奥勒 (Rafael Auler) 著
过敏意 冷静文 译
Getting Started with LLVM Core Libraries
Getting Started with LLVM Core Libraries
紧密结合LLVM的源码，系统介绍LLVM的模块化设计理念并详解技术细节
机械工业出版社
China Machine Press


flex & bison
flex与bison
(中文版)
O'REILLY
东南大学出版社
John Levine 著
陆军 译

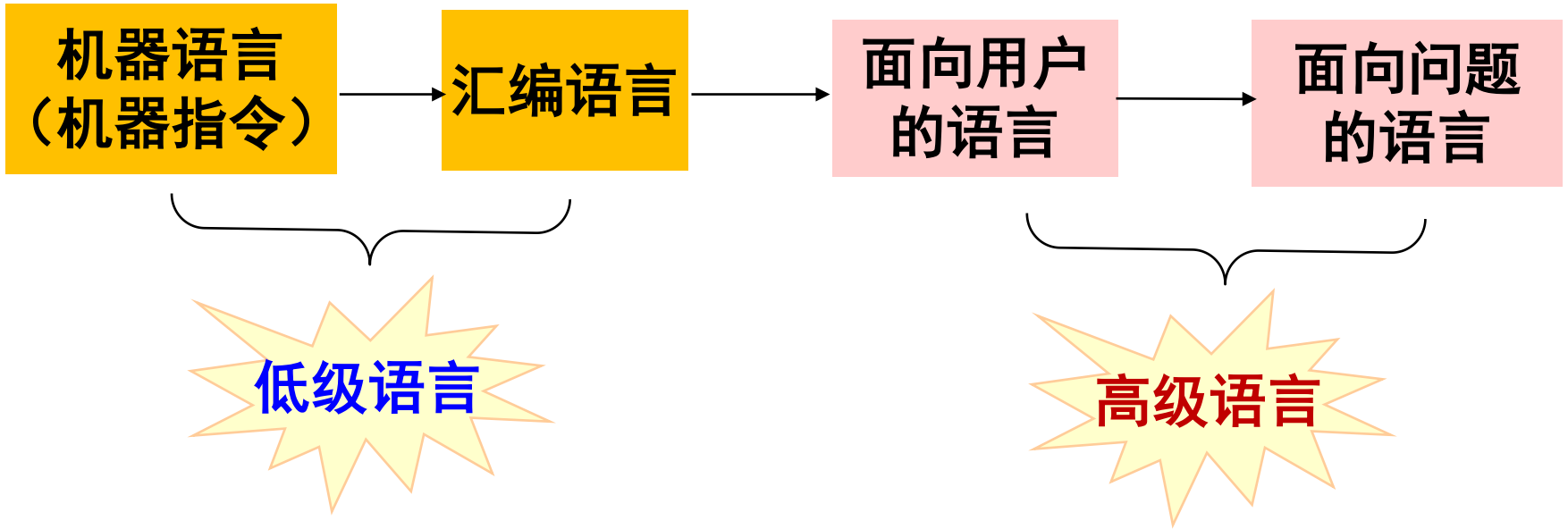
该课程的考核与评价

- **考核方式：**
  - ✓ 课程考核评价：平时成绩和期末考试
  - ✓ 综合评价成绩 = （平时成绩：课堂+作业 + 实验）50% + 期末考试50%
  - ✓ 如果期末笔试成绩低于规定分数时，平时成绩打折处理后，按照成绩构成比例计算最终成绩。具体打折规则等候课程组讨论后通知大家，请等候关注！

第1章 引论

- 计算机语言的简介
- 编译程序所处的层次
- 编译器的特点及功能
- 编译器的内部结构
- 编译器的构造工具

1. 计算机语言的发展



机器语言(Machine Language)

0、1代码与助记符：更接近于计算机硬件指令系统的工作

1011 1000 0010 1011 0001 0101 (B82B15)
1000 1110 1101 1000 (8ED8)
1010 0001 0000 0000 0000 0000 (A10000)
1000 1011 0001 1110 0000 0010 0000 0000 (8B1E0200)
1011 1001 0000 0000 0000 0000 (B90000)
0000 0011 1100 1000 (03C8)
0000 0011 1100 1011 (03CB)
1000 1011 0000 1110 0000 0100 0000 0000 (8B0E0400)
1011 1000 0000 0000 0100 1100 (B8004C)
1100 1101 0010 0001 (CD21)

汇编语言(Assemble Languag)

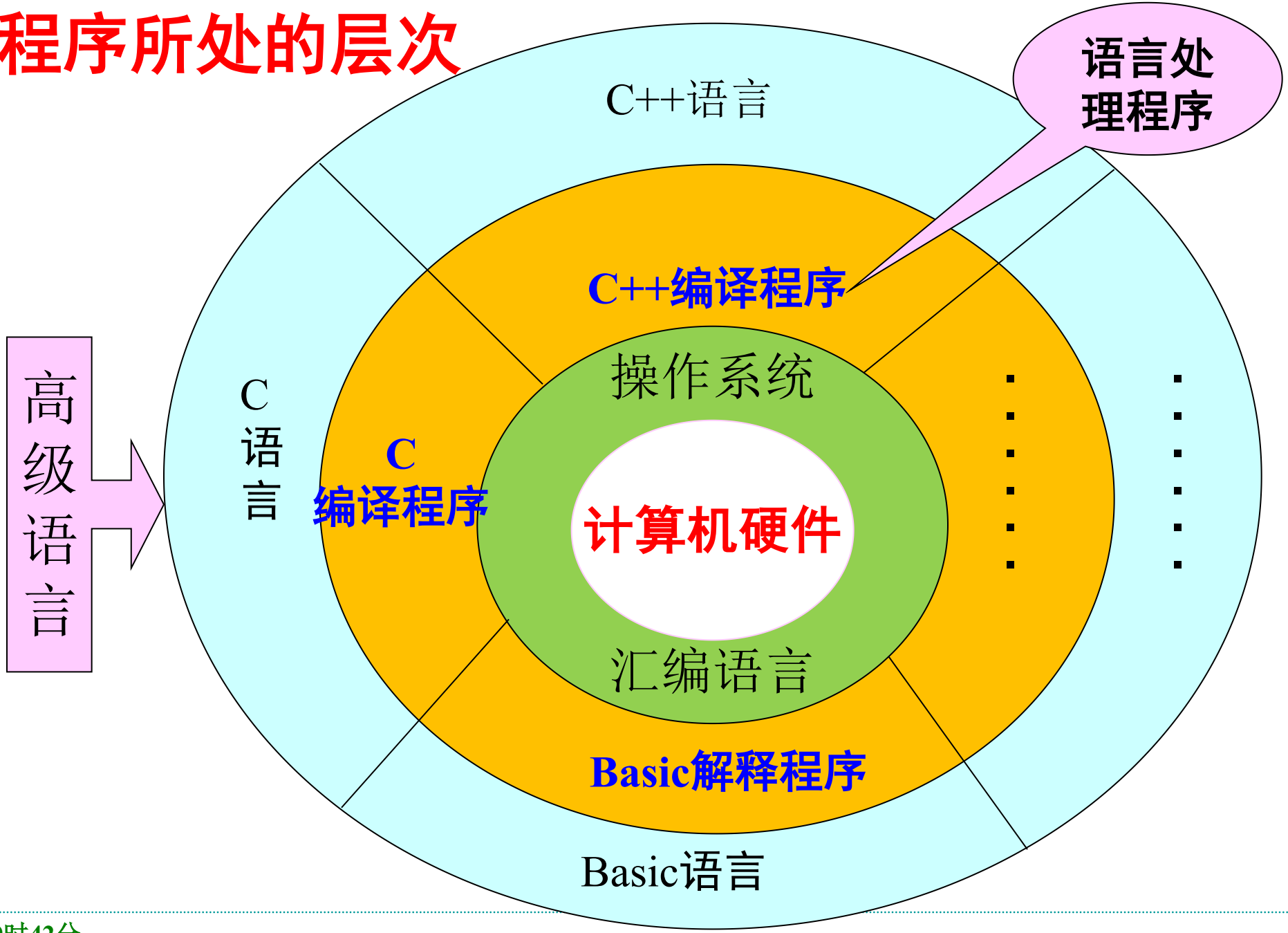
```
assume cs:code, ds:data
data segment
        dw 1234h,5678h
data ends
code segment
start:  mov ax, data
        mov ds, ax
        mov ax, ds:[0]
        mov bx, ds:[2]
        mov cx, 0
        add cx, ax
        add cx, bx
        mov cx, ds:[4]
        mov ax, 4c00h
        int 21h
end start
code ends
```

■ 高级语言(High Level Language)

- 其表示方法更接近于待解问题的表示方法
- 定义数据、描述运算、控制流程、传输数据
- 如：C、FORTRAN、PASCAL、C++、JAVA、SQL语言

```
int main
{
     int a,b,c;
    a=1234h;
    b=5678h;
    c=a+b;
    return 0;
}
```

2、编译程序所处的层次


语言处理程序
C++语言
C++编译程序
操作系统
C语言
C编译程序
计算机硬件
汇编语言
Basic解释程序
Basic语言
高级语言

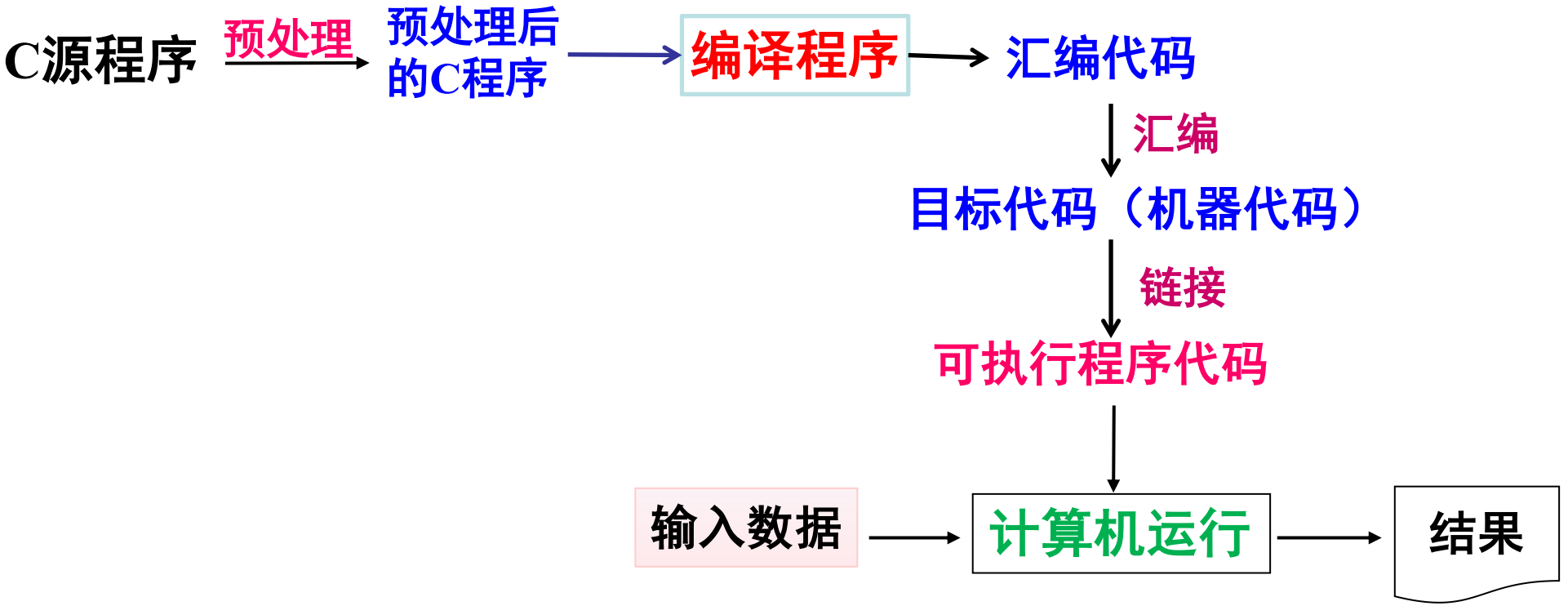
什么叫翻译程序

- **翻译程序：** 能够将用某种编程语言编写的程序转换成另一种编程语言的程序，而且后者与前者的保持**逻辑上是等价的**。

例如：

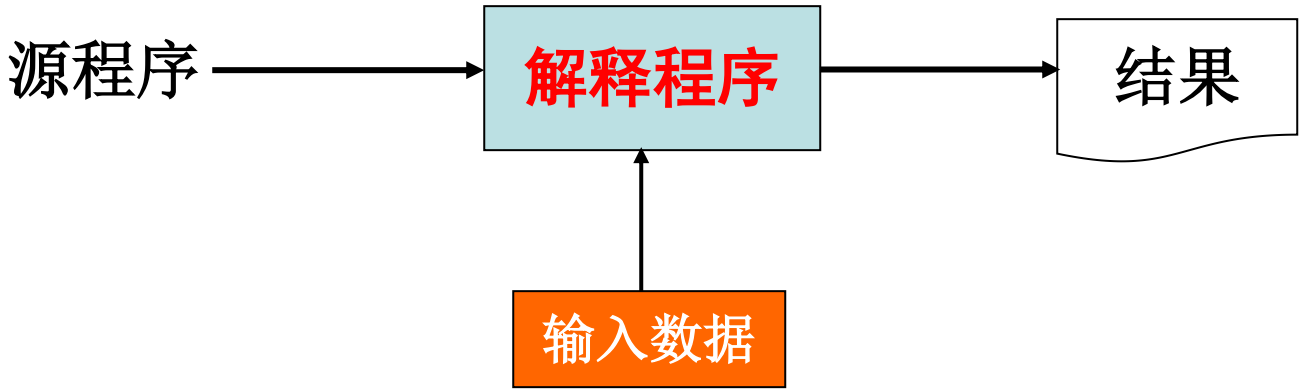
什么是编译程序

- **编译程序(Compiler)**：将高级程序设计语言程序翻译成**逻辑上等价**的**低级语言(汇编语言,机器语言)**程序的翻译程序。

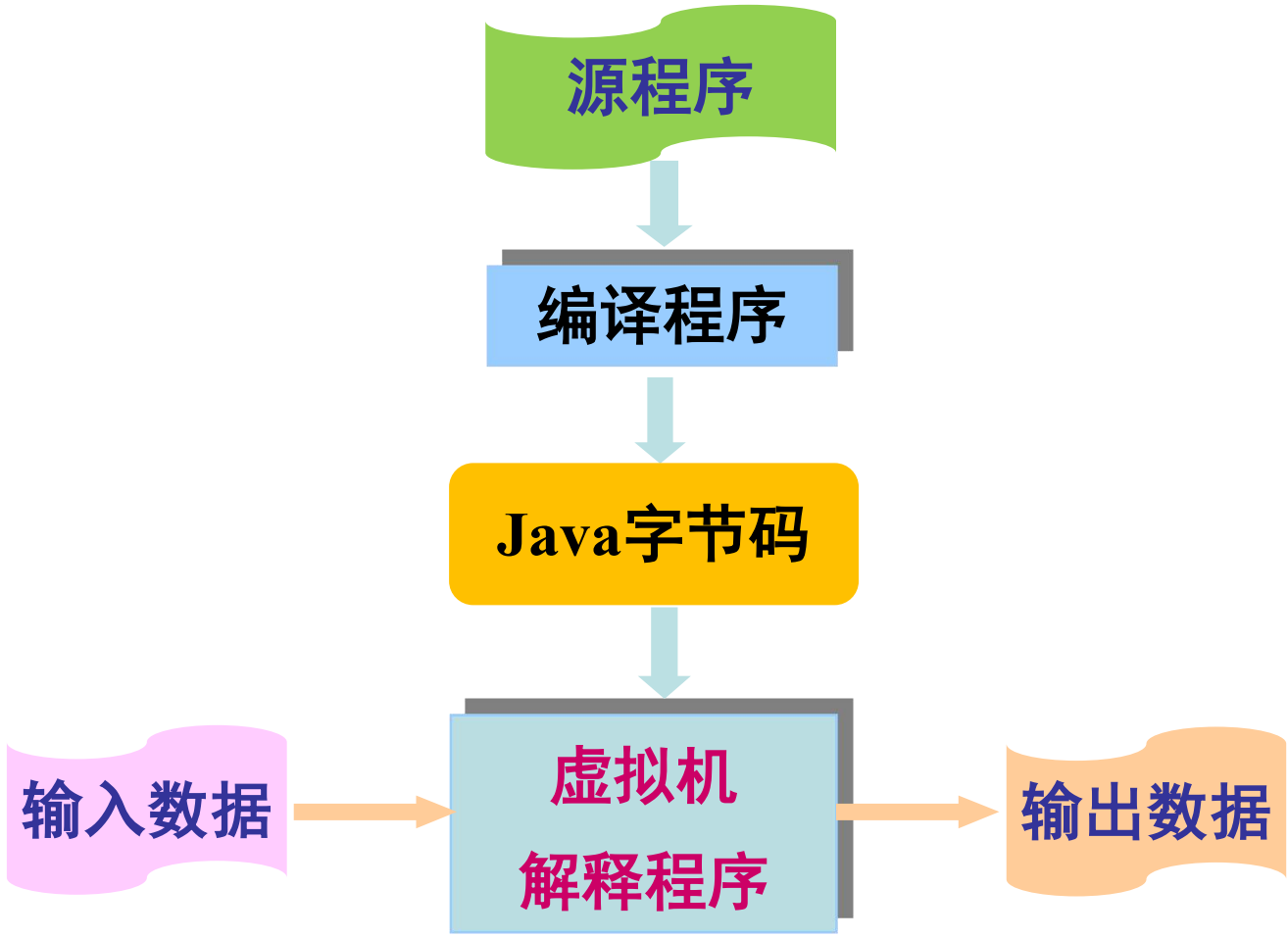


什么是解释程序

- **解释程序(Interpreter)**：将高级程序设计语言写的源程序作为输入，**边解释边执行源程序本身**，而**不产生目标程序**的翻译程序。



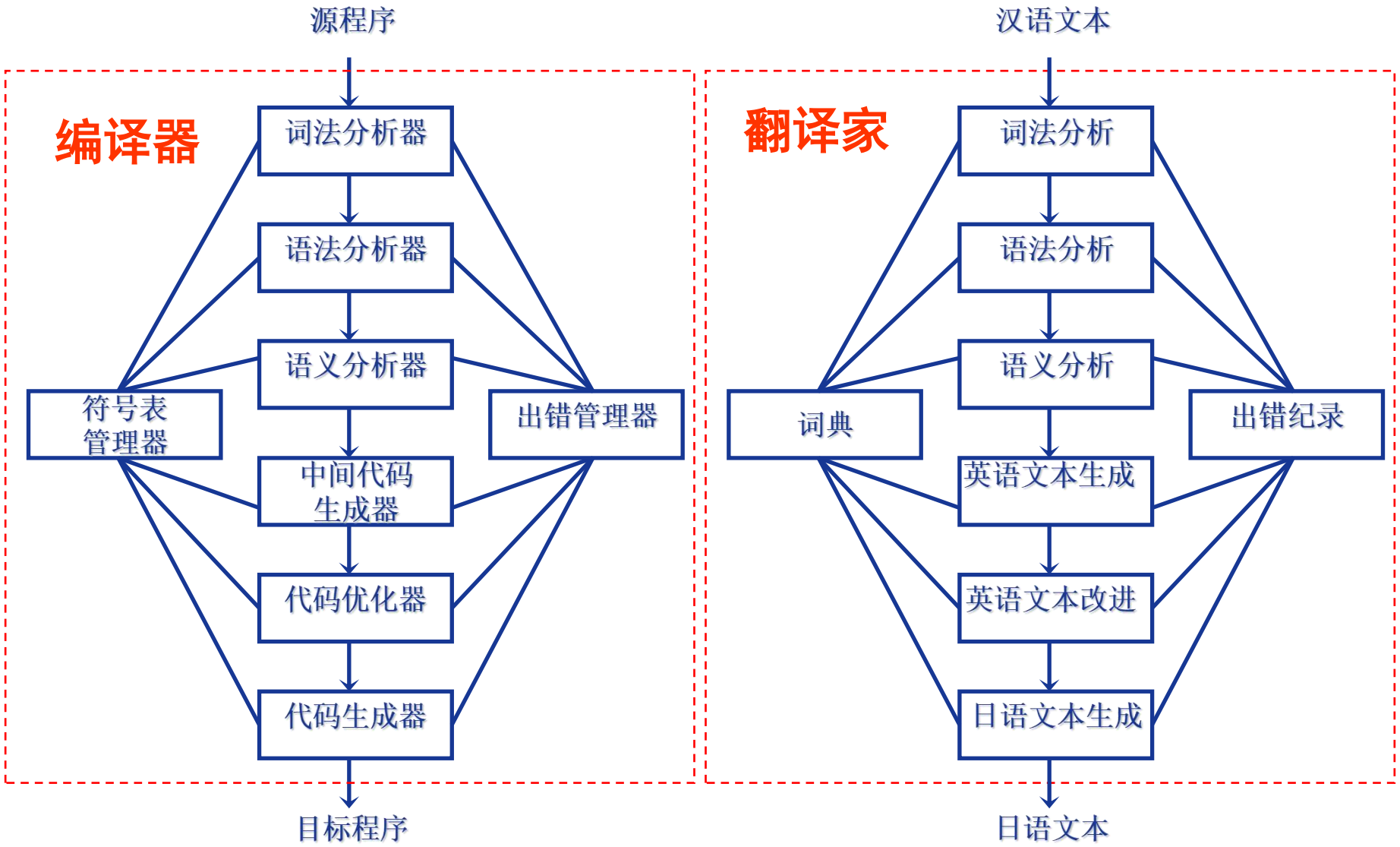
Java语言的编译-解释混合编译器



编译器功能

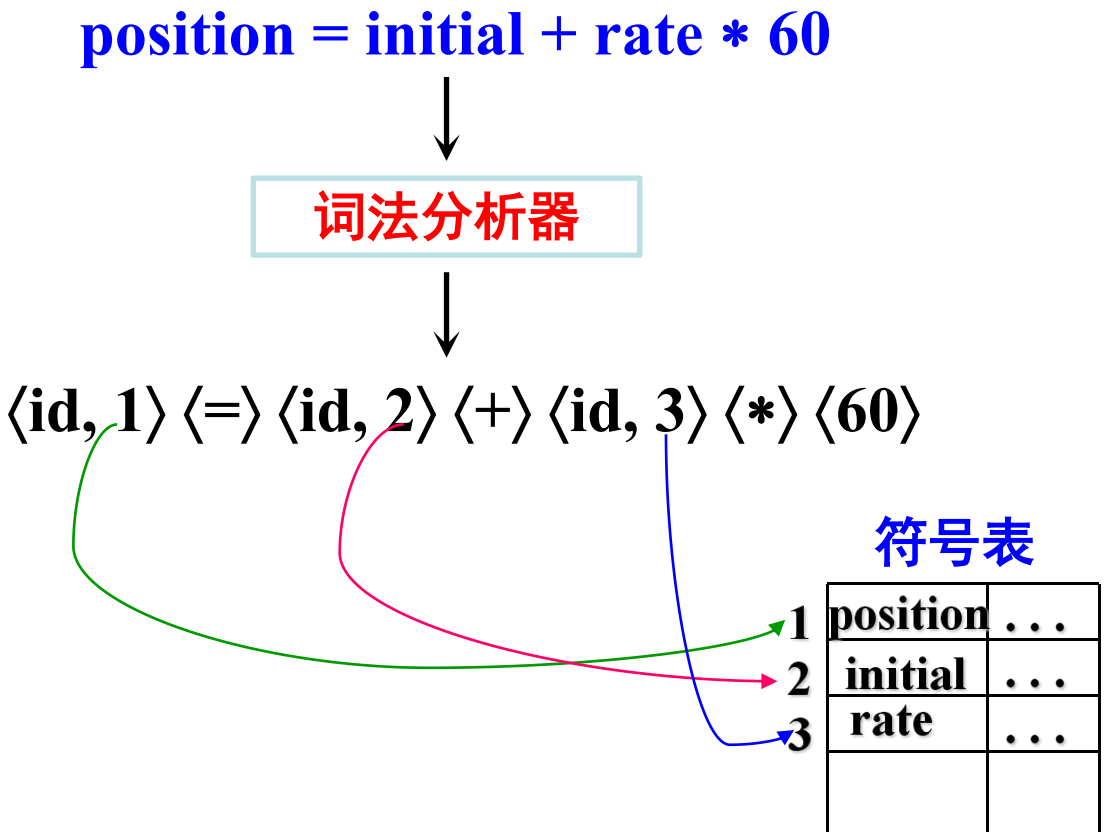
- 它读入用某种语言编写的源程序，并翻译成一个与之等价的目标语言编写的源程序。完成从源语言到目标语言的转换
  - ✓ 源语言：通常是高级语言（C,Java,…）
  - ✓ 目标语言：汇编语言，或者其他形式的低级语言（如Java字节码）
- 编译器实现技术已经发展成熟，并且划分为多个功能相对明确的模块

4、编译器的基本结构


源程序
编译器
词法分析器
语法分析器
语义分析器
符号表
管理器
出错管理器
中间代码
生成器
代码优化器
代码生成器
目标程序
汉语文本
翻译家
词法分析
语法分析
语义分析
词典
出错纪录
英语文本生成
英语文本改进
日语文本生成
日语文本

词法分析

- 词法分析将源代码分解成词法单元(token)，并识别每个词法单元的类型。



请思考下列问题：

1. 如何告诉词法分析器，什么样程序片段是正确的词法单元？
2. 如何设计词法分析器，使它能够正确识别用该语言编写的程序中的词法单元？

符号表(Symbol Table)

- 符号表是编译器的重要数据结构，用于存储程序中出现的标识符（如变量名、函数名等）及其相关信息。符号表在编译过程中起着记录和管理符号信息的作用，以便编译器在后续的各个阶段使用。
- 符号表由多个符号表条目组成，每个符号表条目对应一个标识符，包含该标识符的信息。符号表通常包括标识符名称、数据类型、作用域、存储位置等字段。
- 为了快速查找符号表中的符号信息，符号表通常使用哈希表来实现。哈希表能够根据标识符名称快速定位到对应的符号表条目，提高符号查找的效率。
- 通常编程语言中的关键字单独建立符号表。

符号表的示例

假设有以下简单的代码段：

```
int main() {
    int a = 5;
    float b = 3.14;
    return 0;
}
```

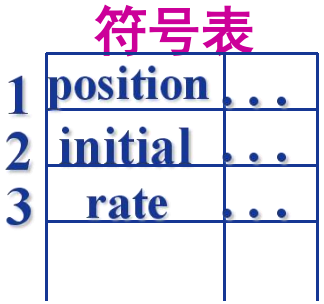
标识符名称	数据类型	作用域	其他信息	……
main	int()	global	函数	
a	int	main	变量	
b	float	main	变量	

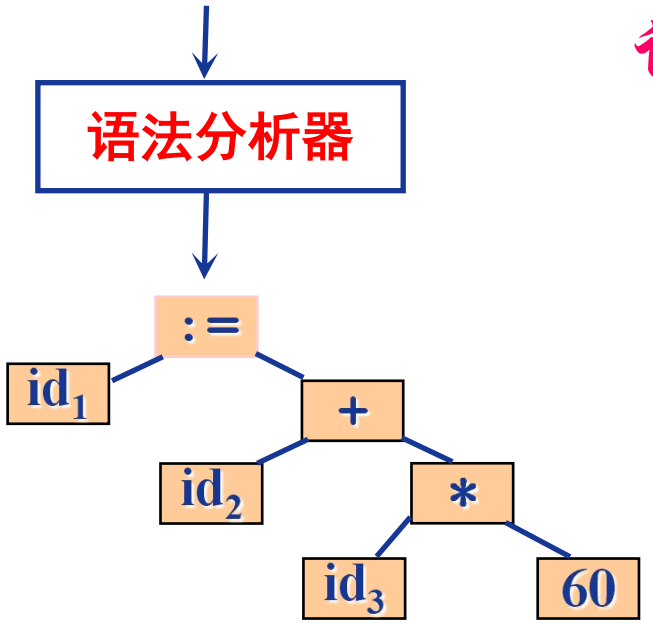
示例展示了一个编译器符号表的基本结构，包括标识符名称、数据类型、作用域等信息。实际的符号表会更加复杂，包括更多的字段和更多的标识符信息，用于支持编译器在编译过程中的各种分析和优化操作。

语法分析

- 语法分析是编译过程中的重要步骤，其主要功能是根据源代码的词法单元(token)流构建语法树（Parse Tree）或抽象语法树（Abstract Syntax Tree，AST）。

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



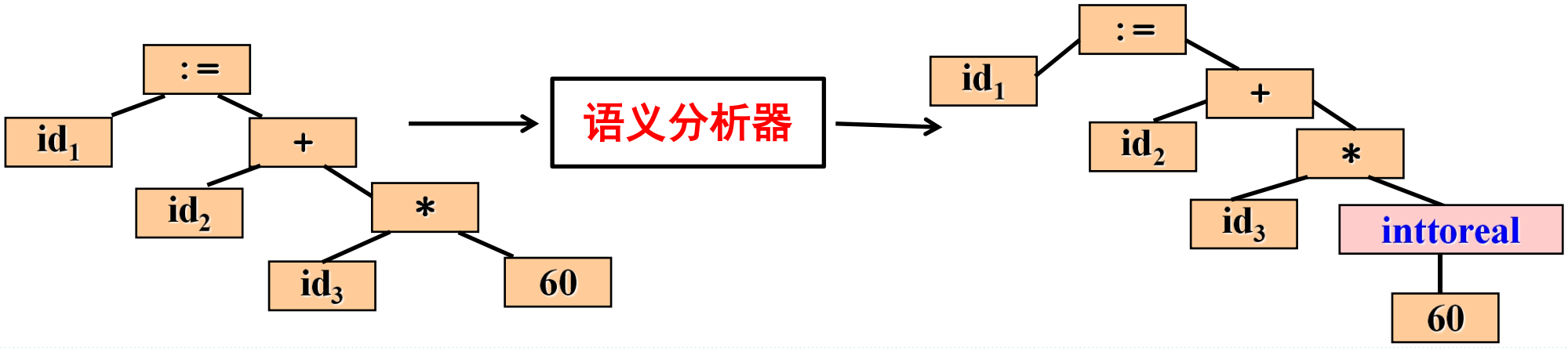


请思考下列问题：

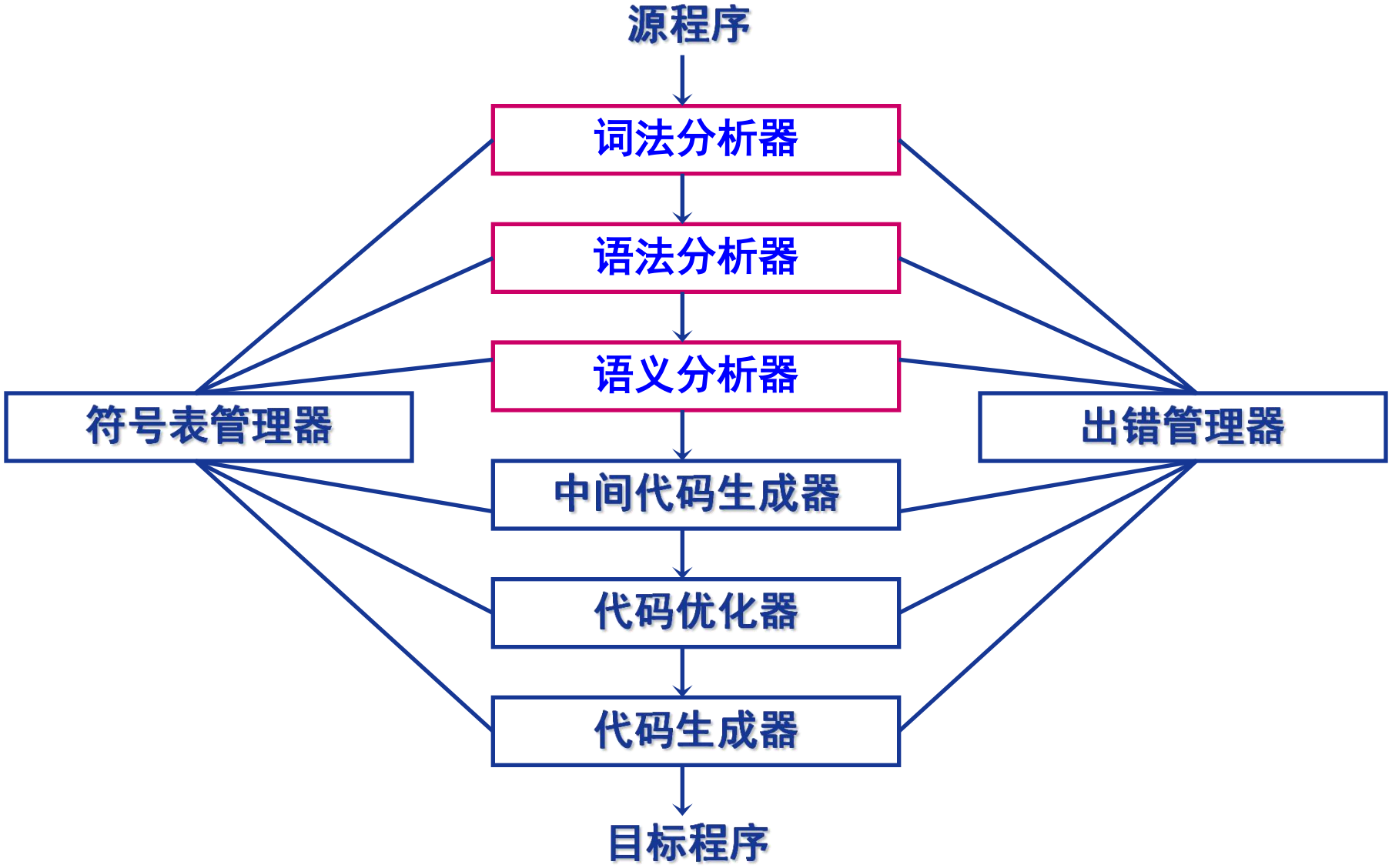
1. 如何告诉语法分析器，该语言能够表达哪些正确语法结构？
2. 如何设计语法分析器（算法），使它能够正确识别用该语言编写的程序中的语法结构？

语义分析

- 语义分析的主要任务是对源代码的语义进行分析和检查。语义分析器会根据语法树或抽象语法树进行语义检查，确保程序的语义是正确的。包括变量的作用域、类型检查、函数调用等。
- 类型检查：确保变量的使用和赋值符合类型规定。如果源代码中存在类型不匹配的情况，语义分析器会报告类型错误。
- 作用域分析：确保变量在正确的作用域内被引用和赋值。如果源代码中存在作用域问题，如变量未声明或重复声明等，语义分析器会进行检查并报告错误。
- 语义错误检测：语义分析器会检测源代码中的语义错误，如未定义的变量、不兼容的类型、不合法的操作等，并生成相应的错误消息。



前三个阶段完成对源程序的分析

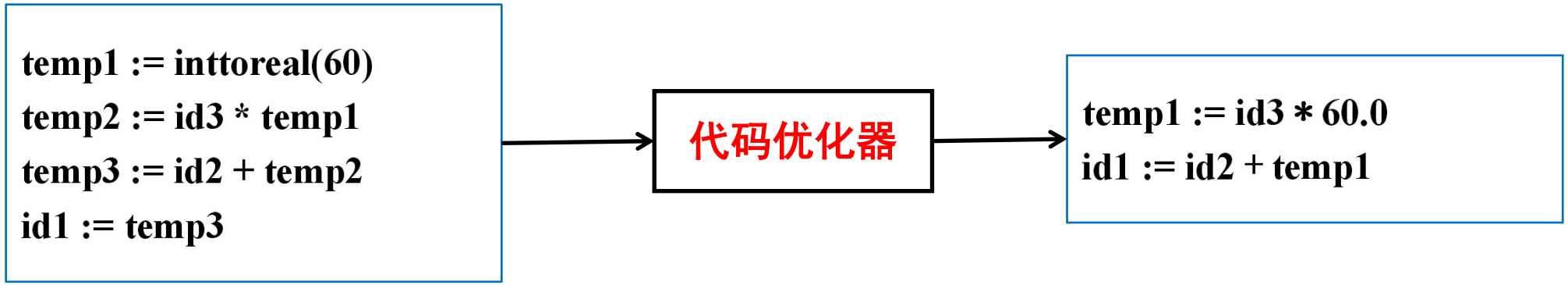


中间代码生成

- 中间代码生成的目标是将源代码转换为一种中间表示形式，通常是一种抽象的、与具体硬件无关的表示形式（如三地址码、四地址码）。中间代码是一种介于源代码和目标代码之间的表示形式，便于后续的优化和代码生成。

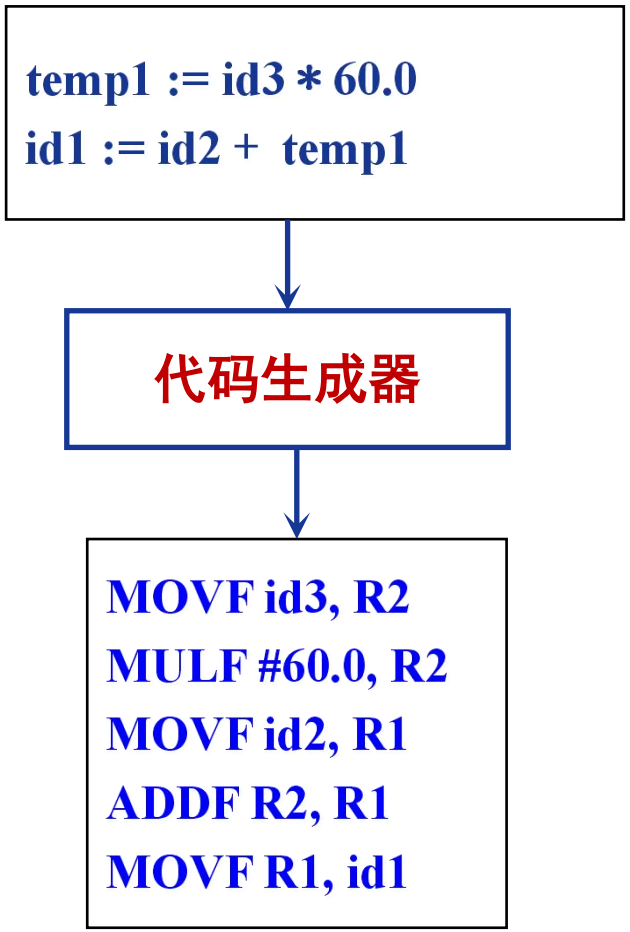
代码优化

- 代码优化的目标是通过改进程序的性能、减少资源消耗或者改善代码质量来提高程序的效率。代码优化可以在中间代码生成阶段或目标代码生成阶段进行。
- 代码优化涉及多种技术，包括但不限于死代码消除、循环优化、数据流分析、指令调度等。旨在改进程序的执行效率、减少资源消耗和提高代码质量。

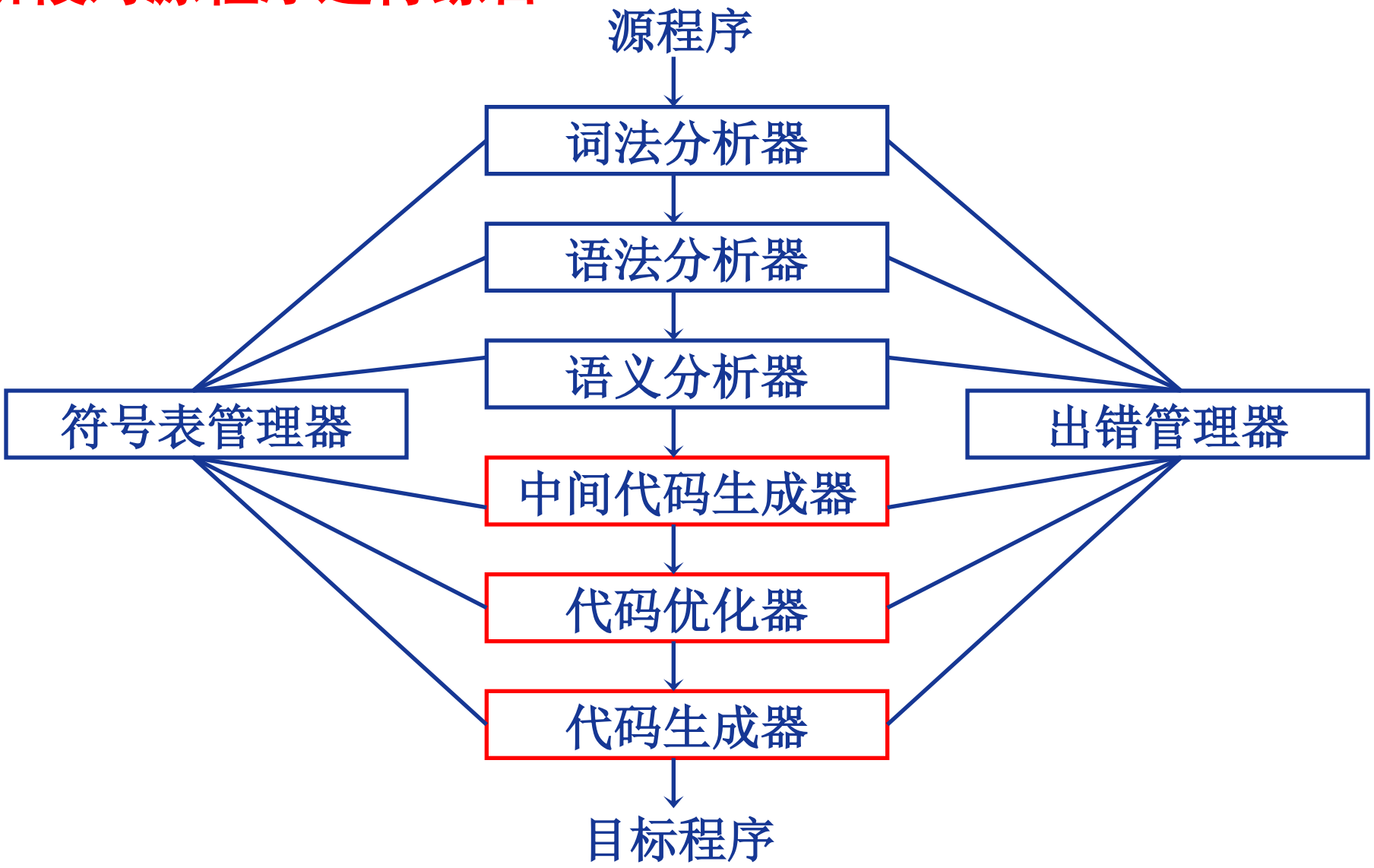


目标代码生成

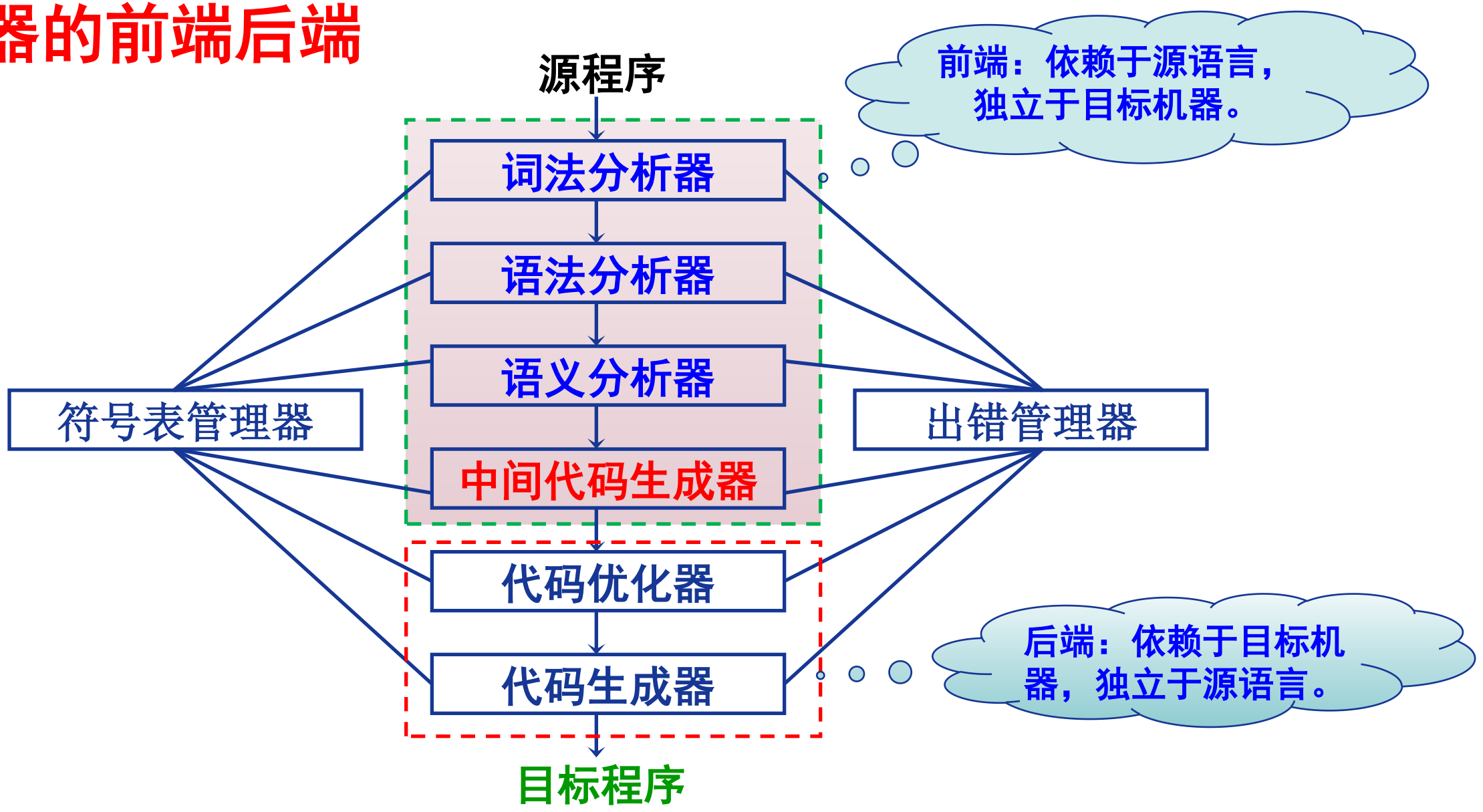
- 输入中间代码，映射为目标语言代码，如果为机器代码，必须为程序变量选择寄存器或内存地址



后三个阶段对源程序进行综合



编译器的前端后端

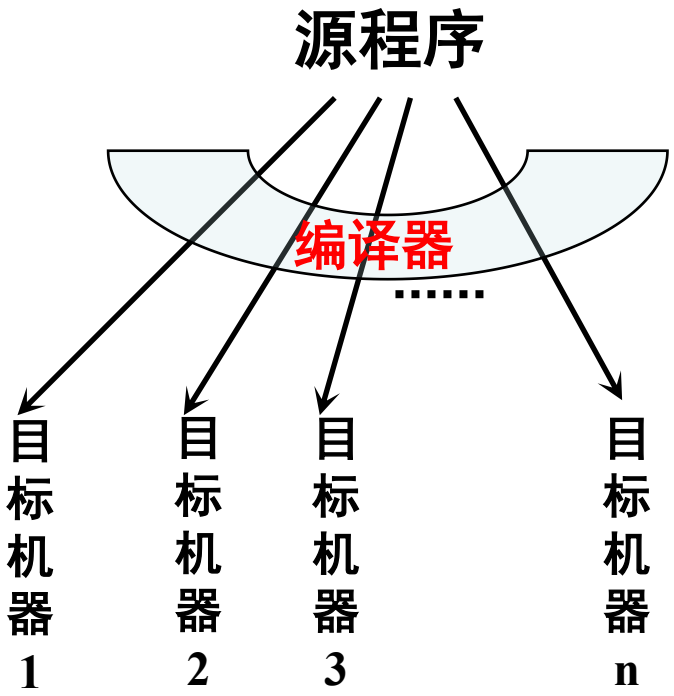

源程序
前端：依赖于源语言，
独立于目标机器。
词法分析器
语法分析器
语义分析器
中间代码生成器
符号表管理器
出错管理器
代码优化器
代码生成器
后端：依赖于目标机
器，独立于源语言。
目标程序

编译器划分为前端和后端的优点

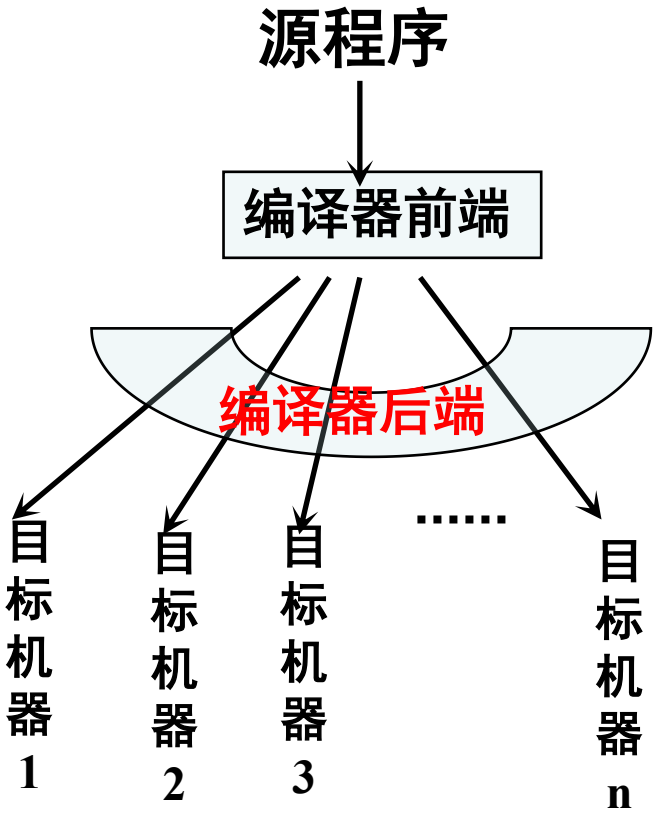
- 把编译过程分成前端和后端两部分
- 前端：只依赖于源程序，独立于目标机器 （生成中间代码）
- 后端：依赖于目标机器，与源程序无关，只与中间语言有关（从中间代码生成目标代码）
- 好处：提高开发编译器的效率
  - ① 取一个编译器的前端，重写它的后端以产生同一源语言在另一机器上的编译器
  - ② 不同的前端使用同一个后端，从而得到一个机器上的几个编译器（采用同一中间语言）

编译器划分为前端和后端的优点

不区分前端和后端的编译器



区分前端和后端的编译器



本章作业

1. 简要解释编译器的作用以及编译过程的各个阶段。
2. 解释编译器前端和后端的区别，并列举各自的主要任务。
3. 什么是语法分析？列举一些常见的语法分析方法。
4. 解释词法分析的作用，并说明词法分析器的输入和输出。
5. 什么是中间代码？列举几种常见的中间表示形式。