# 编译原理与技术

## --概论

刘爽

天津大学智算学部

# 关于这门课

- 32学时，2学分，学科核心课程
- 地点：55-B316，
- 时间：5-12周，周一5-6节、周三3-4节，
- 先修课：高级程序语言，汇编语言程序设计，
  数据结构和算法，计算机体系结构
- 教材/参考书：
  - 《编译原理(第二版)》本科教学版 (Compilers: Principles, Techniques, and Tools (2nd Edition))，Alfred V.Aho等主编，赵建华等译，机械工业出版社，2009年
  - 《程序设计语言编译原理（第3版）》，陈火旺等编著，国防工业出版社，2014年

课程作业：50%
平时签到：50%（包括随机签到，平时作业）

请假：提前与老师说明请假原因，提交请假条。
答疑：每次课程结束后

# 课程平台

- 智慧树 课程号：384757
  - 课件、平时作业、大作业、签到

编译原理与技术–2019级软件工程

课程号 K384757

- 微信群
  - 及时课程信息通知

21221编译原理与技术

# 要求

- 按时上课（会有随机签到）
- 上课时间对教师提出的问题积极思考及讨论
- 课后认真阅读参考资料，独立完成作业，按时提交作业
- 禁止抄袭

# 第1章：概论

- 1.1 编译与解释
- 1.2 编译程序概述
- 1.3 编译程序的结构
- 1.4 编译程序的设计与实现

# 一个简单的自然语言翻译例子

“ I wish you success. ”

**1**）<u>词法分析</u>

| I | wish | you | success |
|---|---|---|---|
| （代） | （动） | （代） | （名） |

**2**）语法分析:

| I | wish | you | success |
|---|---|---|---|
| （主语） | （谓语） | （间宾） | （直宾） |

**3**）语义分析:

我希望你成功。

**4**）优化:

祝你成功。

# 计算机语言

- 低级语言（Low level Language)
  - 字位码、机器语言、汇编语言
  - 特点：与特定的机器有关；
    - 👍功效高；
    - 👎但使用复杂、繁琐、费时、易出错

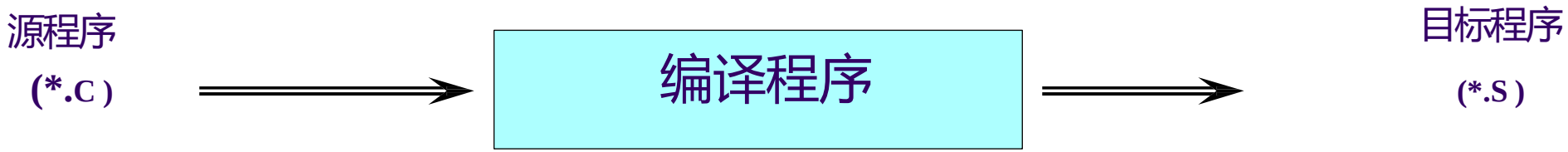
- 高级语言 （High Level Language）
  - C, Java, Python, Ocaml 语言等
  - 特点：不依赖具体机器；
    - 👍移植性好、对用户要求低、易使用、易维护等；
    - 👎无法立即执行，需通过“编译程序”转化为与其等价的机器语言

# 翻译程序


- 翻译程序：汇编程序或者编译程序
  - 汇编（Assemble) 程序 :源语言为汇编语言，目标语言为机器语言。
  - 编译（Compile）程序: 源语言是高级语言，目标程序为某种中间语言或者汇编语言。

# 编译程序

- 将高级程序设计语言（源语言程序）翻译成逻辑上等价的低级语言(目标语言程序，例如汇编语言,机器语言)的翻译程序

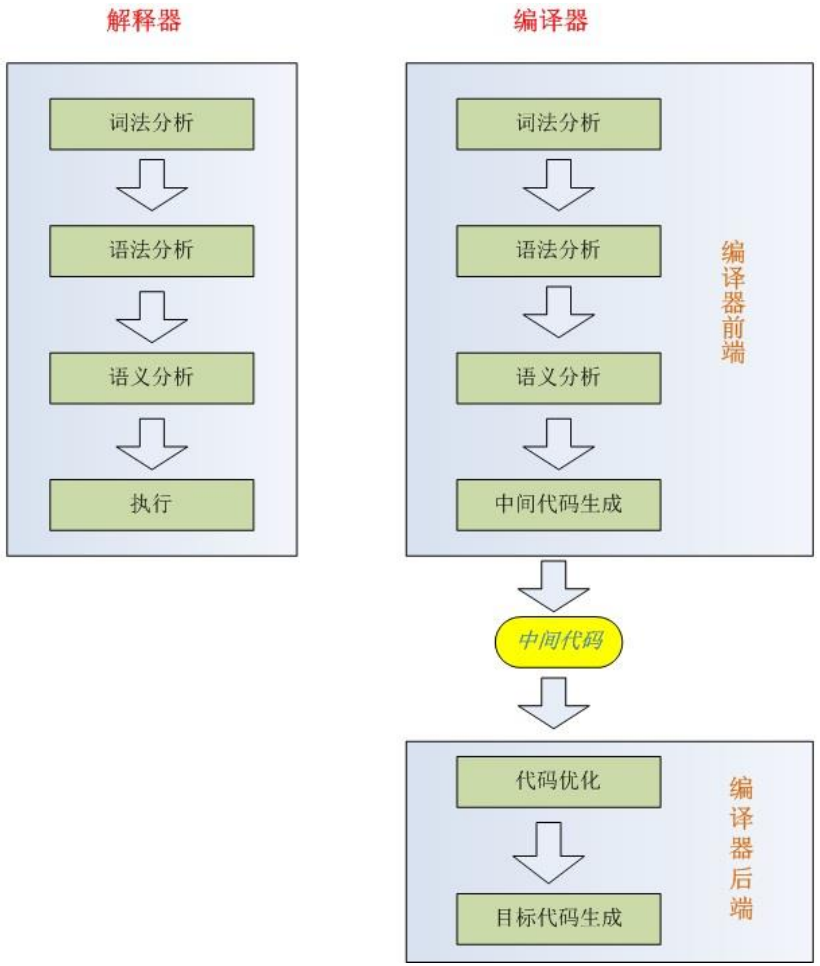


# 编译与解释

- 编译程序：把**高级程序设计语言**翻译成**等价**的**低级语言**的程序。
  - **高级语言源程序→编译程序→目标程序**， e.g., C/C++, Java
- 解释程序：输入源程序，但不产生目标程序，而是按照语言的定义，边解释边执行源程序本身。e.g., scripting languages, Python
- 混合编码：是一种折衷形式。即对运行较慢的部分采用编译，其它部分采取解释执行。
  - 例如JVM以及浏览器中的JIT技术
  - 许多用于编译程序的构造技术同样也适用于解释程序。

# 编译器和解释器

- 编译器(compiler)和解释器(interpreter)的比较
  - 相同点（执行相同的任务）：
    - 检查输入程序并确定这个程序是否为有效程序
    - 建立一个内部模型来刻画输入程序的结构和含义
    - 决定在执行期间值的存放位置
  - 不同点（执行的行为不同）：
    - 编译器以一个可执行程序的描述作为输入，以另一个等价的可执行程序的描述作为输出。
    - 解释器以一个可执行程序的描述作为输入，以执行这一可执行程序描述的结果作为输出。

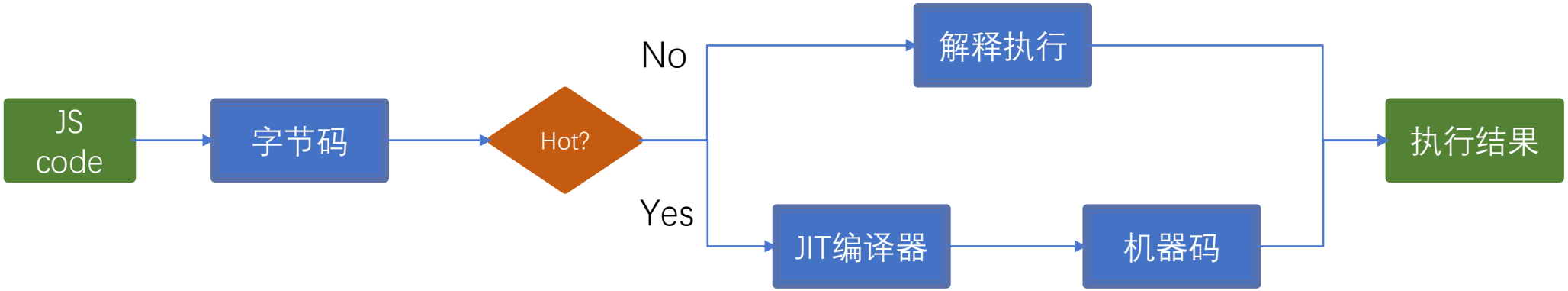


# 我们为什么需要编译程序

- 编写、调试、维护、理解用汇编语言(assemble language)程序是很困难的
- 自从第一个编译器出现之后，软件产品的数量有了巨大的增加
- 好的编译器是计算机科学的缩影
  - 包含大量的技术：贪婪算法（寄存器分配）、动态规划（指令筛选）、有穷自动机和下推自动机（扫描和语法分析）、不动点算法（数据流分析）
  - 处理复杂的问题：动态分配、同步、命名、局部化、存储器分层管理
  - 提供完整的解决方案：有机的结合算法、软件体系结构和软件工程的各种理论，对棘手问题给出综合性的解答方案。

# 思考

- 网页（html/js 格式）的翻译程序是什么？
- （接上题）这个翻译程序是编译器还是解释器？



# 第1章：概论

- 1.1 编译与解释
- **1.2 编译程序概述**
- 1.3 编译程序的结构
- 1.4 编译程序的设计与实现

# 1.2 编译程序概述

- 源程序----------词法分析，语法分析，语义分析和中间代码生成，优化，目标代码生成----------->目标程序

1. **词法分析：输入源程序，对其扫描并分析，识别 各单词符号（关键字、标识符、常数、算符、界符等）**

如：Python语句

- **for　　i　　in　　range　　(　　1　　,　　100　　)**

关键字　整变量　关键字　关键字　界符　整常量　界符　整常量　界符

- 词法分析的依据是词法规则。

# 1.2 编译程序概述

**2. 语法分析：根据语法规则,把单词串分解成各类语法范畴（短语、子句、表达式、程序段、程序）。**

- 语法分析所依循的是语言的语法规则。 **sum=current + accumulative**

**3. 语义分析和中间代码生成**

- 根据各语法成分的**语义规则**写出其中间代码。
- 常用的中间代码有：三元式、四元式, 间接三元式、树等。

例如： 四元式表示为：

| + | i | M | M |
|---|---|---|---|
| 操作码 | 第一操作数 | 第二操作数 | 结果 |

# 1.2 编译程序概述

4. **优化: 对中间代码进行加工变换， 以期在最后 阶段 产生出更为高效的 (省时间和空间 ） 的目标代码。**

•常做的优化有：公共子表达式的外提、循环优化、算符归约等。

例如：

```
for k := 1 to 100 do
begin
  m := I + 10 * k ;
  n := j + 10 * k ;
end;
```

四元式代码

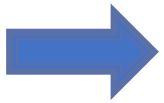
| 序号 | 操作码 | 第一 | 第二 | 结果 | |
|---|---|---|---|---|---|
| 1 | : = | 1 | | k | |
| 2 | j< | 100 | k | (9) | \\ for k:=1 to 100 |
| 3 | * | 10 | k | t1 | |
| 4 | + | I | t1 | m | \\ m:=I+10*k |
| 5 | * | 10 | k | t2 | |
| 6 | + | j | t2 | n | \\ n:=j+10*k |
| 7 | + | 1 | k | k | |
| 8 | j | | | (2) | |

- 3和5, 200次乘法（优化）

# 1.2 编译程序概述

- 优化后的四元式: 优化所依循的原则是程序的**等价变换原则**。

| 序号 | 操作码 | 第一 | 第二 | 结果 | |
|---|---|---|---|---|---|
| 1 | : = | 1 | | k | |
| 2 | j< | 100 | k | (9) | \\ for k:=1 to 100 |
| 3 | * | 10 | k | t1 | |
| 4 | + | I | t1 | m | \\ m:=I+10*k |
| 5 | * | 10 | k | t2 | |
| 6 | + | j | t2 | n | \\ n:=j+10*k |
| 7 | + | 1 | k | k | |
| 8 | j | | | (2) | |



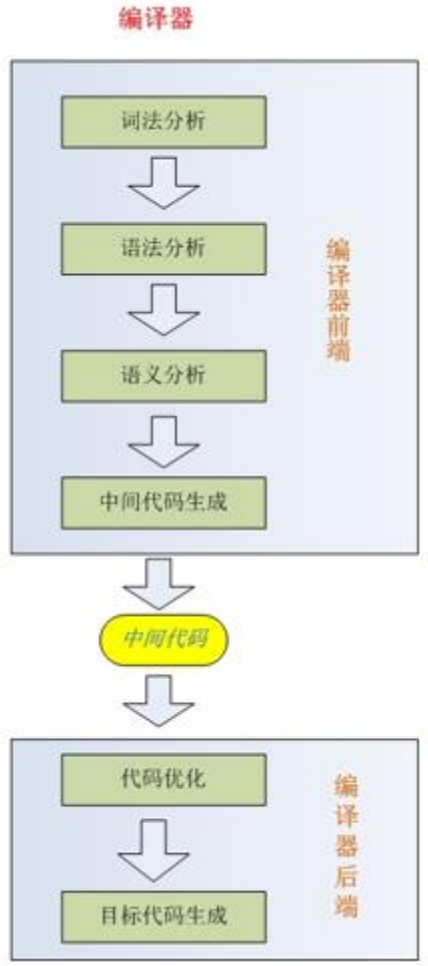
| 序号 | 操作码 | 第一 | 第二 | 结果 |
|---|---|---|---|---|
| 1 | : = | I | | m |
| 2 | : = | j | | n \\ m,n初值 |
| 3 | : = | 1 | | k |
| 4 | j < | 100 | k | (9) \\ for k:=1 to 100 |
| 5 | + | m | 10 | m |
| 6 | + | n | 10 | n \\ m,n循环+10 |
| 7 | + | 1 | k | k |
| 8 | j | | | (4) |

# 1.2 编译程序概述

**5** 目标代码的生成：

- 在中间代码（或经过优化）的基础上，生成某具体的硬件代码，可以是绝对机器代码，或汇编代码。

- 总之，编译要先**分析**，以确定源程序的功能；然后**综合**，以生成该功能的目标程序。



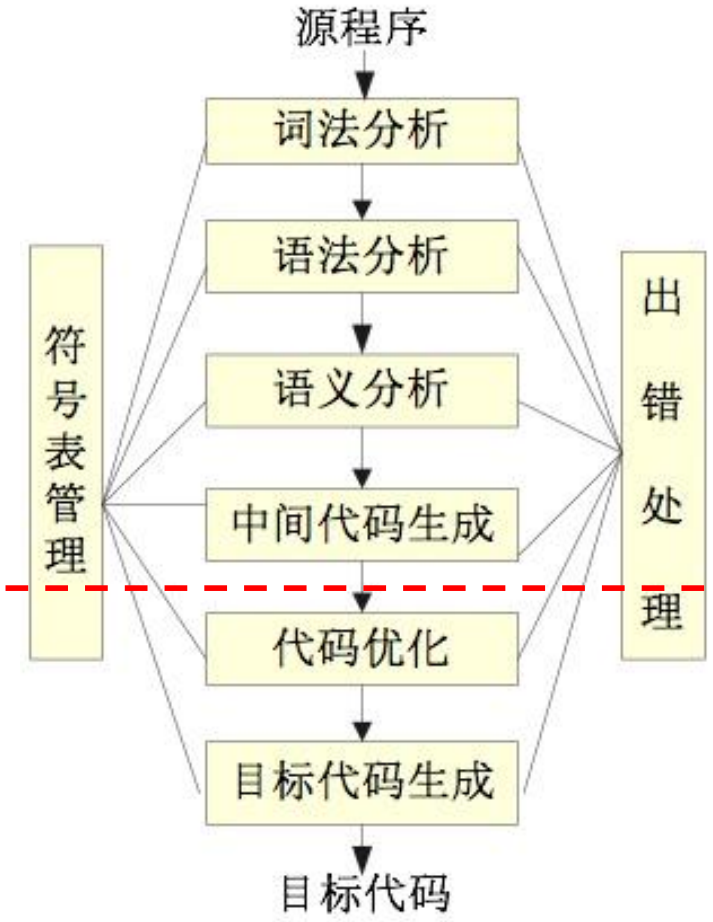
# 思考

- 编译程序的5个阶段分别是什么？

# 第1章：概论

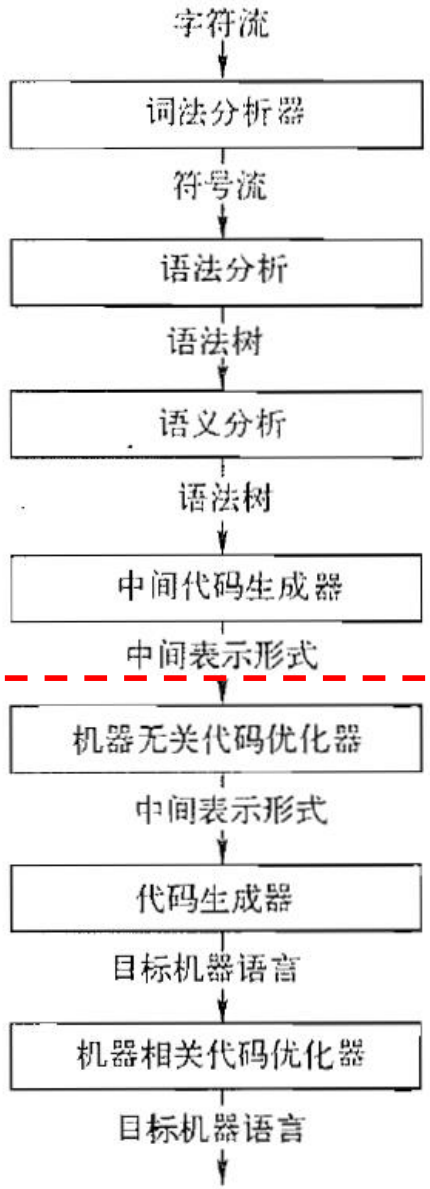
- 1.1 编译与解释
- **1.2** 编译程序概述
- 1.3 编译程序的结构
- 1.4 编译程序的设计与实现

# 编译程序的结构




**编译的前端**
**(Front End)**
**分析部分**
**与源语言有关**

**编译的后端**
**(Back End)**
**综合部分**
**与目标语言有关**



# 例子（1/3）

- 例1： 对赋值语句的分析与综合过程。

position ：= initial+rate* 60（其中 var position, initial, rate: real;)

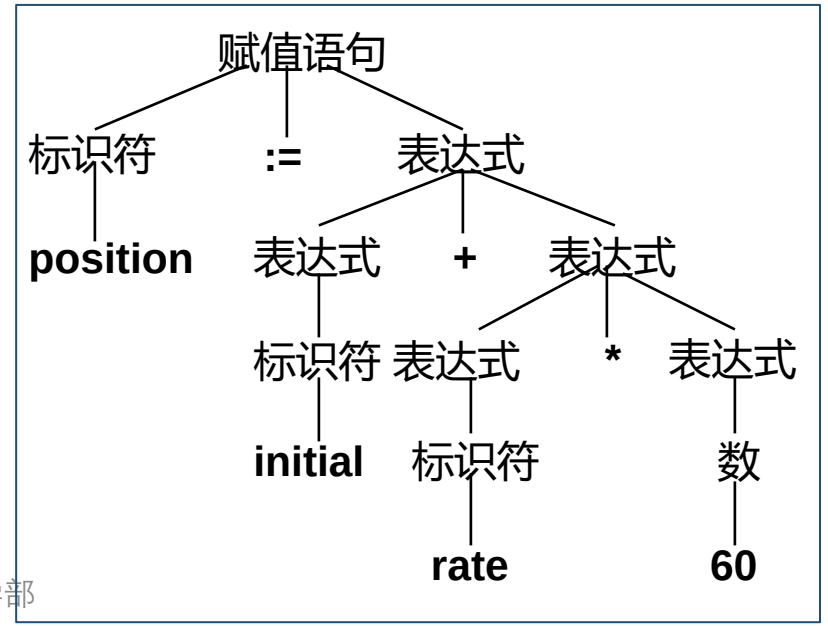
(1) 词法分析

- 跳过空格,注释语句

| | |
|---|---|
| 标识符 | position initial rate |
| 赋值号 | : = |
| 加号 | + |
| 乘号 | * |
| 整形（数） | 60 |

(2) 语法分析

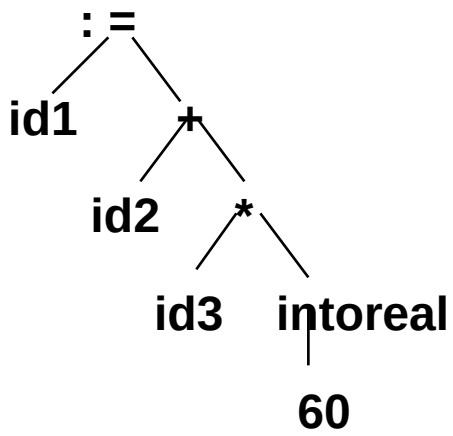
- 结构层次分析—语法分析树，对于表达式的判定。



# 例子（2/3）

position ：= initial + rate * 60（其中 var position, initial, rate: real;)

- （3）语义分析:
  - 类型检查：标识符的类型匹配。
  - 类型转换：整--实（intoreal) itoa,atoi　字符--整 (chartoint) ctoi,itoc



# 例子（3/3）

符号表

| position | --- |
|---|---|
| initial | --- |
| rate | --- |
| | |
| | |

- 4) 中间代码生成：
  - 三地址代码（类似汇编语言）

```
intoreal   60          t1
 *         id3   t1    t2
 +         id2   t2    t3
 :=        t3          id1
```

- 5) 代码优化：（时、空）

```
 *         id3   60.0   t1
 +         id2   t1     id1
```

6)目标代码生成

—— 中间代码 (**R1**、**R2** 寄存器)

```
MOVF   id3 ,  R2
MULF   #60.0 , R2
MOVF   id2 ,  R1
ADDF   R2  ,  R1
MOVF   R1 ,   id1
```

*说明：第一操作数（源操作数） 第二操作数（目标操作数）

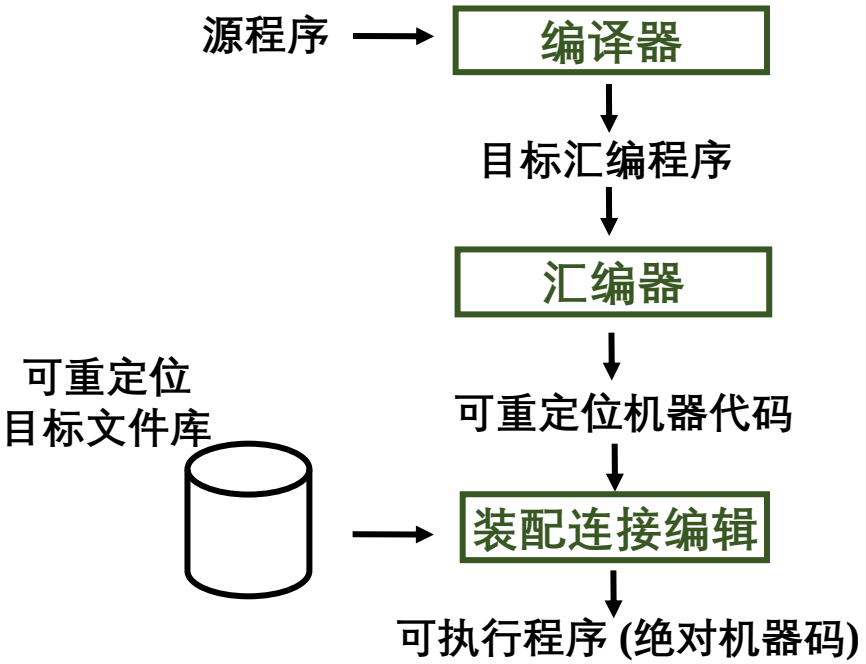
# 遍（PASS）

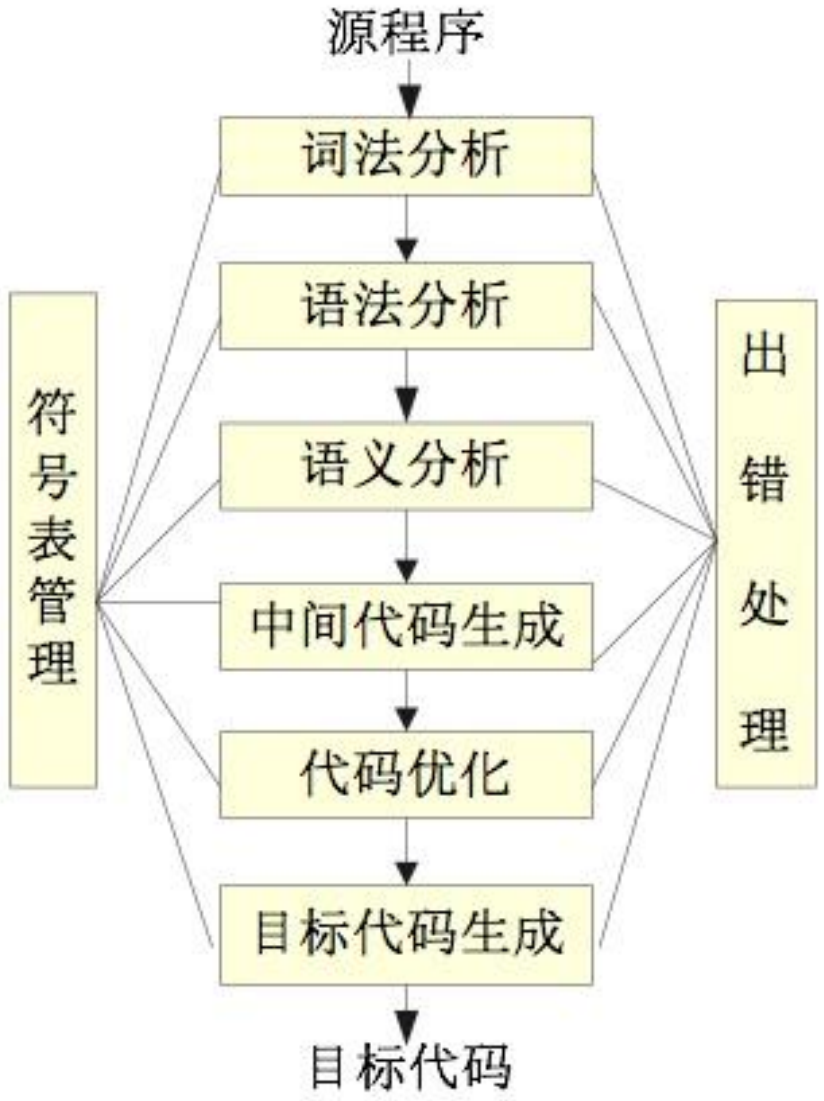
- 遍：对源程序（包括源程序的中间表示形式）**从头到尾扫描一次**并作有关的加工处理，**生成新的源程序中间形式或目标程序**，通常称之为一遍。上一遍的结果是下一遍的输入，最后一遍生成目标程序。

- 遍与基本阶段的区别：
  - 五个基本阶段是将源程序翻译成目标程序在逻辑上要完成的工作
  - 遍是指完成上述五个基本阶段的工作要经过几次扫描处理

# 编译过程

- 源代码到可执行程序
- 本课程关注编译器的原理



# 练习

- 分别列举属于编译器前端和编译器后端的阶段

- 编译器的每个阶段都需要独立的一遍来完成，这种说法是否正确?

# 第1章：概论

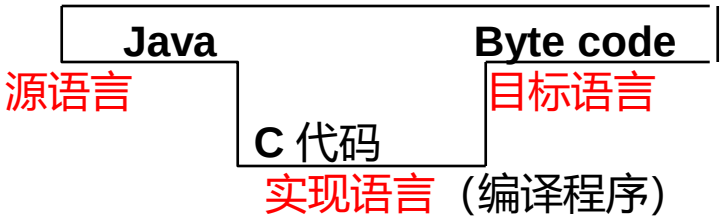
- 1.1 编译与解释
- **1.2** 编译程序概述
- 1.3 编译程序的结构
- 1.4 编译程序的设计与实现

# 编译器的基本原则

- 编译器是工程对象，是具有独特目标的大型软件系统，两个设计原则必须遵守
  - 不违背原义
    - 编译器必须保持被编译程序的含义不变
    - 这一原则是编译器设计者与编译器用户之间的契约的核心
  - 实用性原则
    - 编译器必须用某种明确的方式改进输入程序
    - 例如代码优化等对输入程序的改进

# 1.4编译程序的设计与实现

- 用“ T”型图表示编译程序。
- 读作：用实现（编译）语言编写的，
  生成目标语言程序的源语言编译程序。
- 例如：用C语言编写的，
  生成Java Bytecode的Java编译程序。

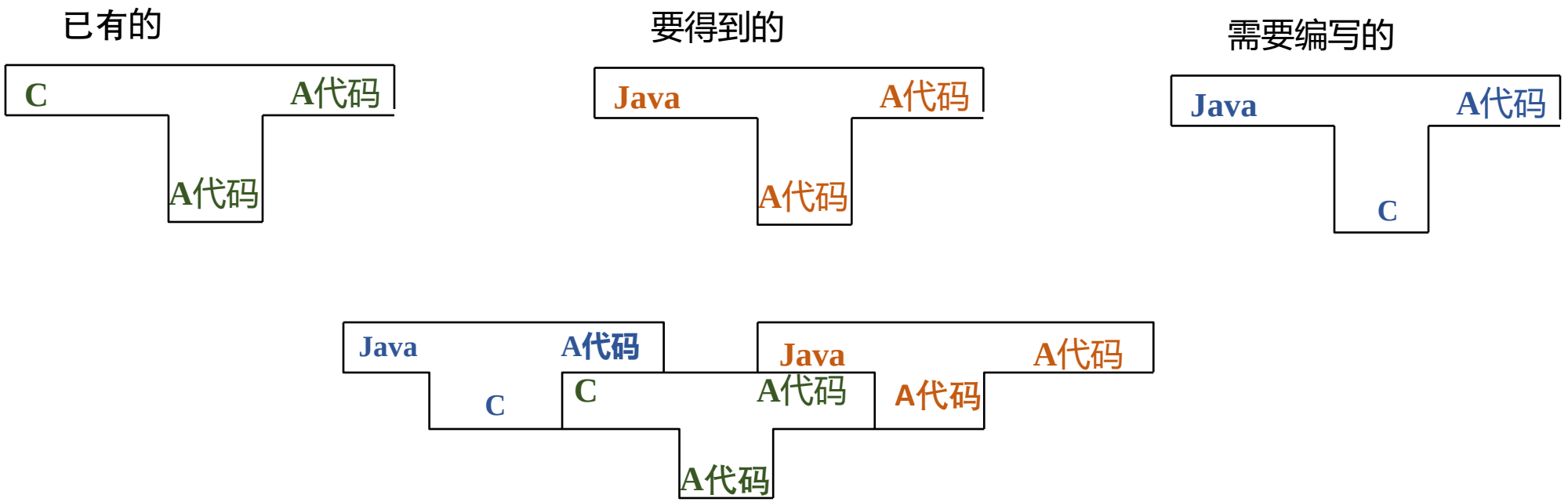



- 一般不采用低级语言作为实现语言。

# 1.4编译程序的设计与实现

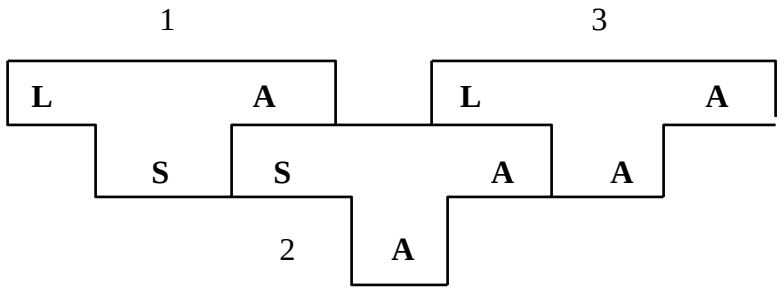
**T型图的结合规则：**

**（1）中间T图的两臂上的语言分别与左右两个T图脚上的语言相同。（2）对于左右两个T图而言，其两个左端的语言必须相同，两个右端的语言亦必须相同。**



# 1.4编译程序的设计与实现（cont）

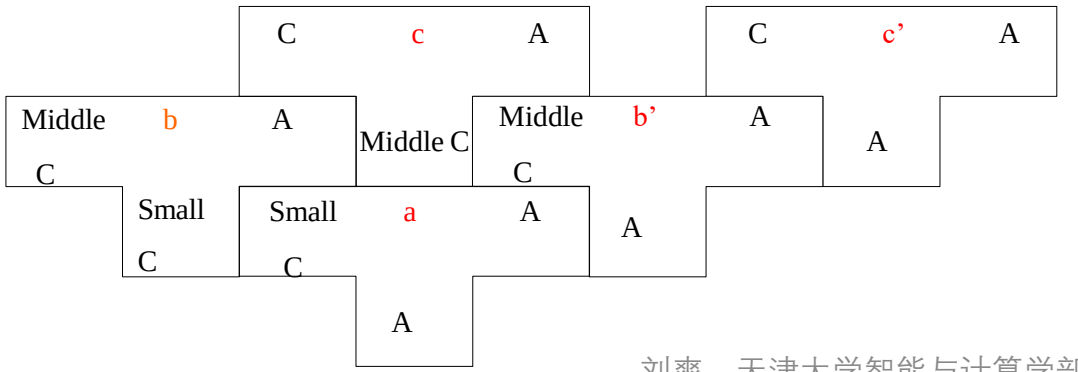
- 用第2个编译程序去编译第1个，得到第3个编译程序




- 如何获得A机器上第一个可书写编译程序的高级语言？
  - 自展
  - 移植

# 自展 (自编译)

- 在A机器上用机器语言或汇编语言编写高级语言的子集，例如，Small C的编译程序(a)
- 经Small C 编写的Middle C的编译程序（b）
- 经Small C 的编译程序编译后，得Middle C的编译程序（b’）
- 由Middle C 编写全C的编译程序（c）
- 经Middle C 的编译程序编译后，得C的编译程序（c’）

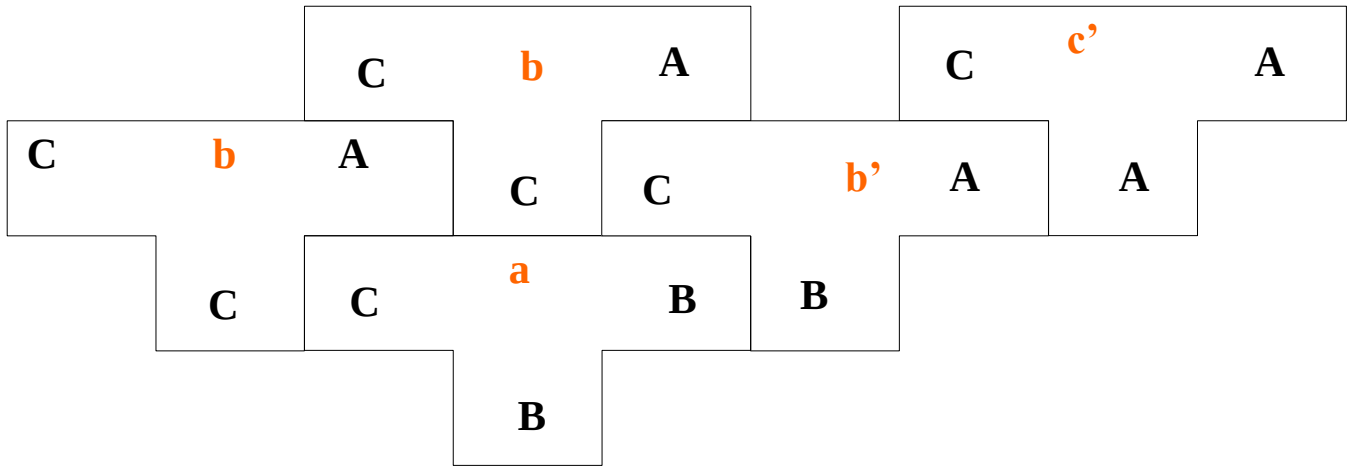


# 自展(cont)

- 自展方式就像滚雪球那样扩大语言的编译程序。
- 利用自展的方式使我们用机器语言或汇编语言编写高级语言的编译程序的工作量减少到最低限度。
- 注意：实际实现要做多次的自编译才能得到可靠的编译。

# 移植

- 在B机器上已有一个可运行的C编译程序（a）
- 只要我们编写一个用C语言书写的，产生A机器代码的C编译程序(b),按如下T型图的运算去做，就得到A机器上所要的C编译程序。
- (b),(a) ⟶ (b’)；移植；(b), (b’) ⟶ (c’);



# 移植(cont)

- 优点：用高级语言书写的编译程序，生产的周期短，可靠性高，易 修改、扩充与维护，并且易于移植。
- 缺点：代码量长，但随着存储能力的提高、运行速度越来越快，程序正确性是主要矛盾。

- UNIX操作系统的实用程序 LEX，YACC 自动地生成词法分析器和语法分析器.

# 练习

- 假设已经有用汇编语言书写的C语言的编译器（目标语言为汇编语言），现在想要得到用汇编语言书写的Python语言的编译器（目标语言为汇编语言），请用移植的方式完成该编译器的设计和实现。（画出T型图的表示方式）

# 练习

- 假设已经有用汇编语言书写的C语言的编译器（目标语言为汇编语言），现在想要得到用Java Bytecode书写的Java语言的编译器（目标语言为Java Bytecode），请用移植的方式完成该编译器的设计和实现。（画出T型图的表示方式）

# 《编译程序》的设计目标：

（1）生成尽量小的目标程序。
（2）目标程序运行速度尽量快。
（3）编译程序尽可能小。
（4）编译所花的时间尽可能少。
（5）有较强的查错和改正错误的能力。
（6）可靠性好。

# 小结

- 1.1 编译与解释
  - 任务相同，行为和输出不同
- 1.2 编译程序概述
  - 五个主要阶段，可以分多遍实现
- 1.3 编译程序的结构
  - 分析与综合
- 1.4 编译程序的设计与实现
  - 自展和移植

阅读材料：
《程序设计语言编译原理（第3版）》第一章