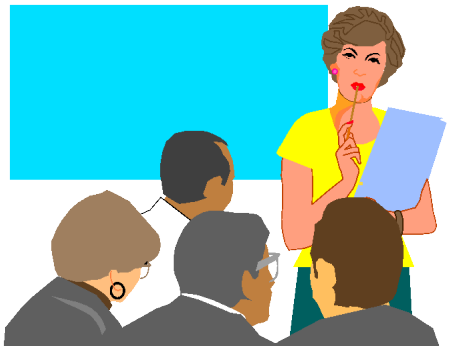
编译技术



课程简介

学时:40+8

分为两部分:

- 理论基础(40学时）:课堂教学（30%平时+70%闭卷考试）
- 实践教学(8学时）:实验教学（4个实验，实验代码+实验报告）

课程要求:

掌握编译技术的基本理论、编译系统的构造（结构和机理）并进行编译程序相关部分的设计，加深对计算机高级语言的理解，在此基础上能够灵活运用所学技术解决实际问题。

程序如何被翻译为机器指令

1. 变量的本质是什么？

2. 顺序、选择、循环语句对应哪些中间或目标代码？

3. 函数调用时局部变量在内存中如何分配？

4. 如何深入理解对象与变量的区别？

5. 函数重载编译器都做了什么？

6. 什么是多态？编译器做了哪些工作来支持虚函数？虚函数运行时是什么情况？

7. 任意给出一个面向过程或面向对象的程序，如何深入分析执行过程？

……

为什么要学习编译原理—理论价值

- 学习这门课程有利于加深对程序设计语言的理解，可以比较迅速掌握新的语言工具，也是读懂相关计算机著作的基础
- 编译课程蕴含着计算机学科中解决问题的思路、抽象问题和解决问题的方法

为什么要学习编译原理

—实践价值（软件开发）

- 编译程序是计算机系统中的系统软件，包含许多软件技术，对于进行软件设计很有价值
- 从软件工程角度，首先进行需求分析，进行阶段划分，逐一完成每个阶段的功能

参考书

（1）姜淑娟，刘兵等.编译原理及实现，清华大学出版社，2016.（教材）

（2）陈火旺,刘春林等.程序设计语言编译原理（第3版）.北京：国防工业出版社，2014.

（3）陈英,陈朔鹰等.编译原理.北京：清华大学出版社,2012.

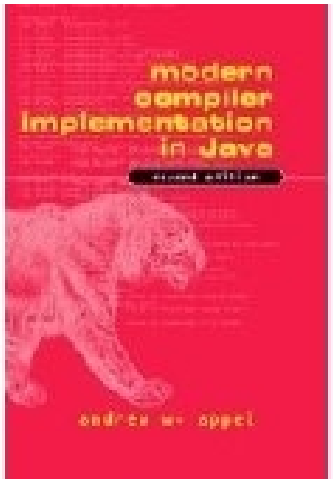
（4 ） Kenneth C.Louden ，赵建华等译.编译原理及实践（第2版）.机械工业出版社，2009.

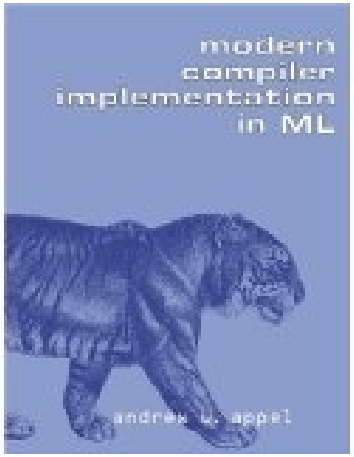
国外参考书（编译圣经）

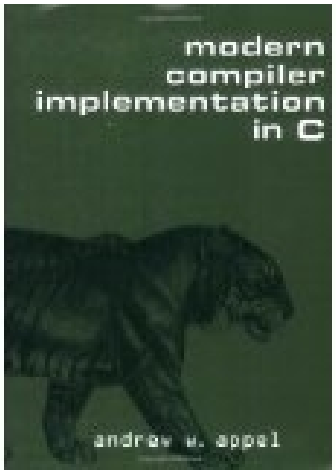
- *Compilers: Principles, Techniques and Tools* ***(Dragon book)***

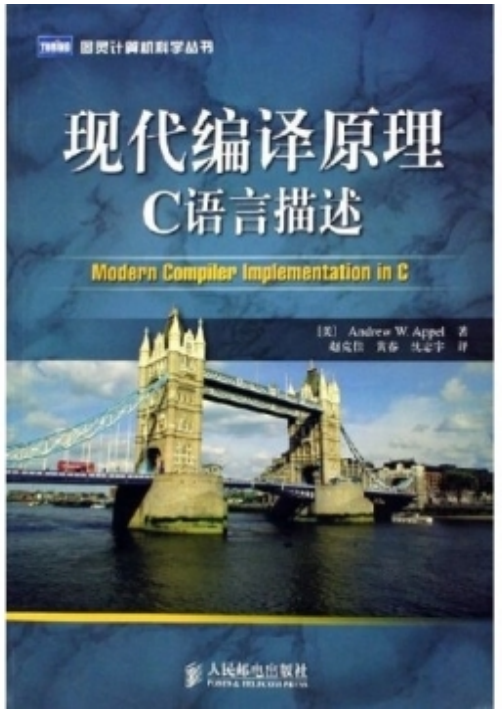
国外参考书（编译圣经）

- *Modern Compiler Implementation in Java,C,ML* ***(Tiger book)***



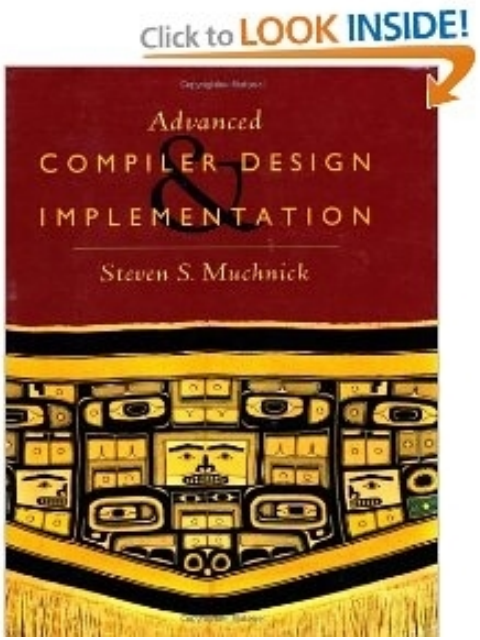


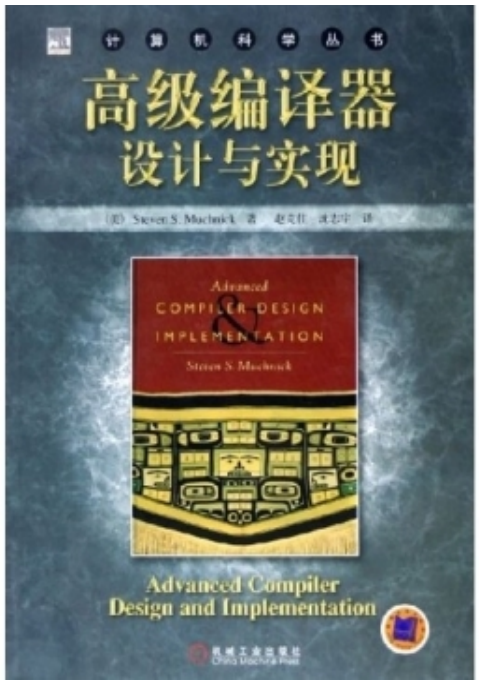




国外参考书（编译圣经）

- *Advanced Compiler Design and Implementation* ***(Whale book，鲸书)***





第一章 编译概述

- 1.1 什么是编译程序
- 1.2 编译程序的逻辑结构
- 1.3 编译阶段的组合
- 1.4 编译程序实现的途径
- 1.5 语言开发环境中的伙伴程序
- 1.6 编译程序的构造工具

1.1 什么是编译程序

程序设计语言：用来编写计算机程序的语言。

- 程序设计语言
  - 高级语言
    - 过程式语言　Fortran,Pascal,C
    - 函数式语言　Lisp
    - 逻辑式语言　Prolog
    - 对象式语言　C++
  - 低级语言：面向机器的语言
    - 汇编语言
    - 机器语言

程序设计语言

- 机器语言：直接用计算机能够识别的二进制代码指令来编写程序的语言。由二进制的指令代码组成。
- **1 + 3 表示为 10000001 00000001 00000011**
- 是最底层的计算机语言，不需要翻译就可以直接被计算机硬件识别。对应不同的计算机硬件有不同的机器语言。
- 特点：执行速度快，但编写程序的难度大，修改、调试不方便，直观性差，不易移植。

程序设计语言

- 汇编语言：又称为符号语言。与机器语言一一对应，采用能帮助记忆的英文缩写符号（指令助记符）来代替机器语言指令中的操作码，用地址符号来代替地址码。用指令助记符及地址符号书写的指令称为汇编指令，用汇编指令编写的程序称为汇编语言源程序。
- **将X、Y中的内容相加 表示为 ADD X Y**
- 机器不能直接识别汇编语言程序，必须把它翻译为机器语言程序才能执行。
- 特点：比机器语言直观，容易理解和记忆，比高级语言的执行效率高，但通用性和移植性较差。

程序设计语言

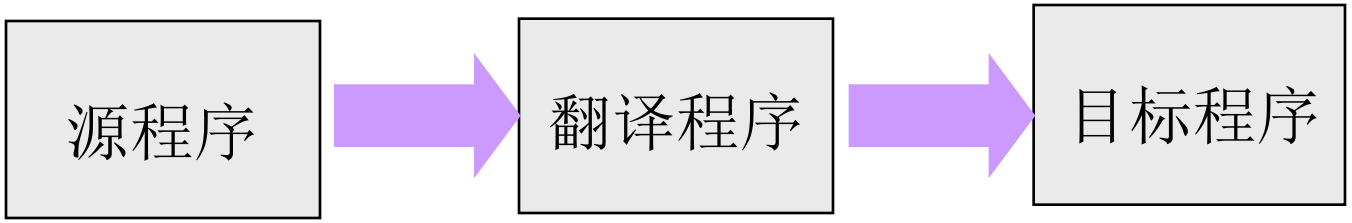
- 高级语言：与具体的计算机硬件无关，是面向问题的程序设计语言，其表达方式接近于自然语言和数学语言，易于人们接受和掌握。
- 采用类似于数学公式的书写方式：$x = 1 + 3$
- **特点：独立于具体的计算机硬件，程序的编制和调试方便，通用性和可移植性好。在计算机执行之前，需要通过编译程序翻译成目标语言程序，或需要通过解释程序边解释，边执行。时间与空间效率比较低。**
- 目前比较流行的高级语言有：Visual C, Visual Basic, Java，FoxPro，Pascal,Lisp, Cobol等。

比较	机器语言	汇编语言	高级语言
硬件识别	是唯一可以识别的语言	不可识别	不可识别
是否可直接执行	可直接执行	不可，需汇编、连接	不可，需编译/解释、连接
特点	✓面向机器 ✓占用内存少 ✓执行速度快 ✓使用不方便	➢面向机器 ➢占用内存少 ➢执行速度快 ➢较为直观 ➢与机器语言一一对应	❖面向问题/对象 ❖占用内存大 ❖执行速度相对慢 ❖标准化程度高 ❖便于程序交换，使用方便
定位	低级语言，极少使用	低级语言，很少使用	高级语言，种类多，常用

翻译程序

把一种语言书写的程序（源程序）翻译成另一种语言的**等价**的程序（目标程序）。是汇编程序、编译程序以及各种变换程序的总称。

源程序、翻译程序、目标程序 三者关系：



即源程序是翻译程序的输入，目标程序是翻译程序的输出

•汇编程序

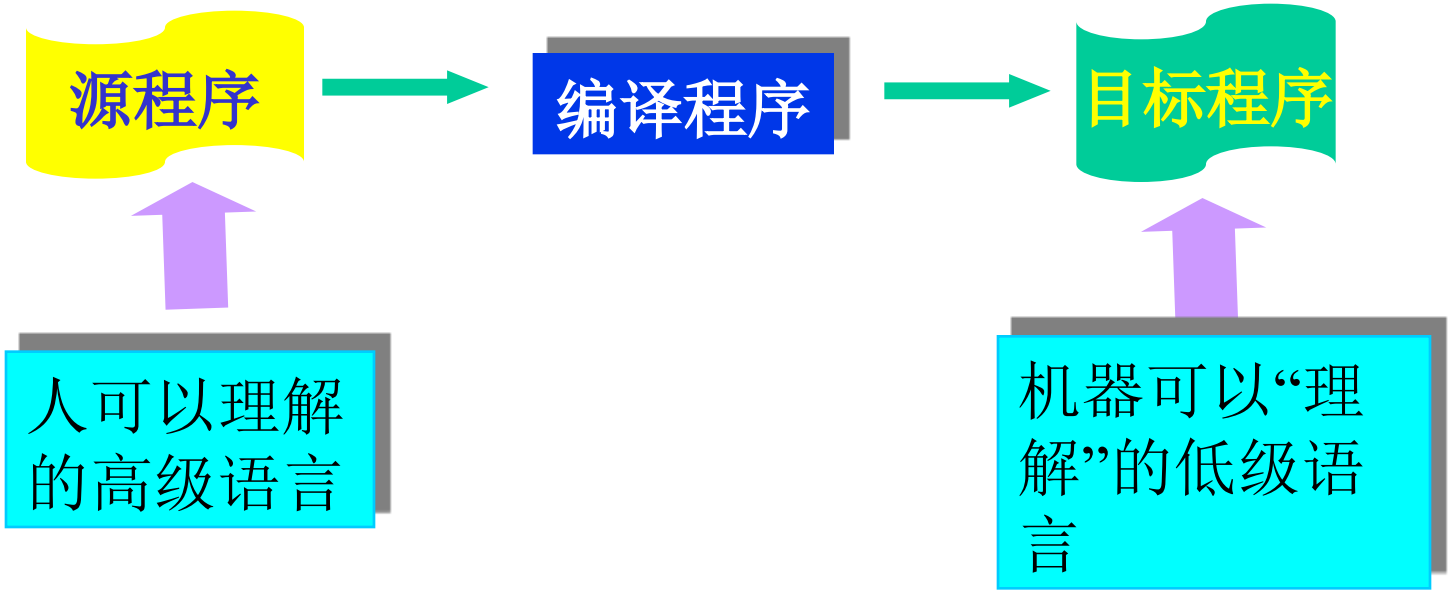
把汇编语言程序翻译成机器语言程序，这时的翻译程序称之为汇编程序，这种翻译过程称为“汇编”（Assemble）

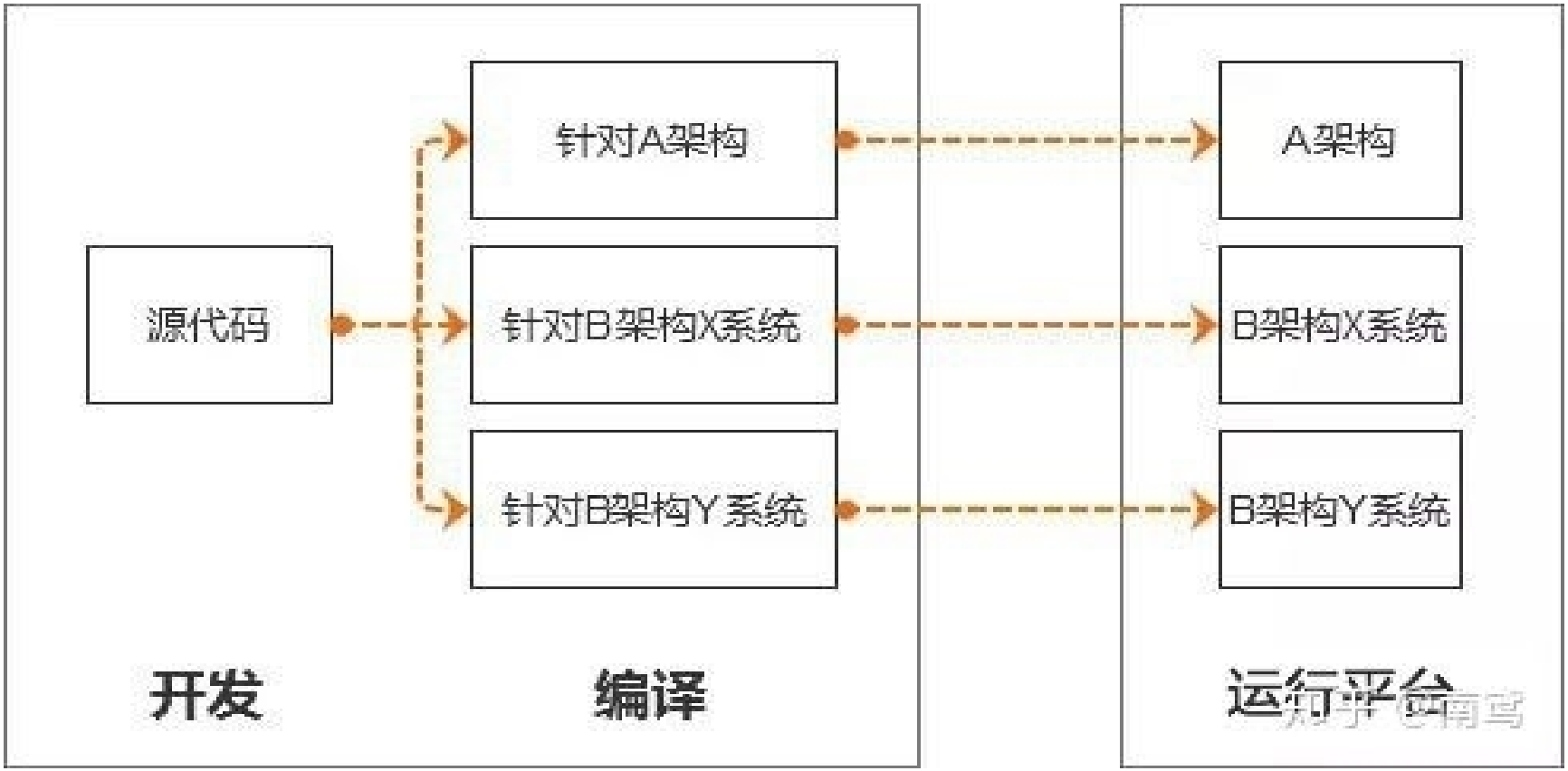
•编译程序

把高级语言程序翻译成低级语言程序，这时的翻译程序称之为编译程序，这种翻译过程称“编译”（Compile)

汇编程序与编译程序都是翻译程序，主要区别是加工对象的不同。由于汇编语言格式简单，常与机器语言之间有一一对应的关系。汇编程序所要做的翻译工作比编译程序简单的多。

- 编译程序是翻译程序的一种，它将一个用面向人的源语言书写的程序（高级语言程序）翻译成一个等价的面向硬件的目标程序（低级语言程序）

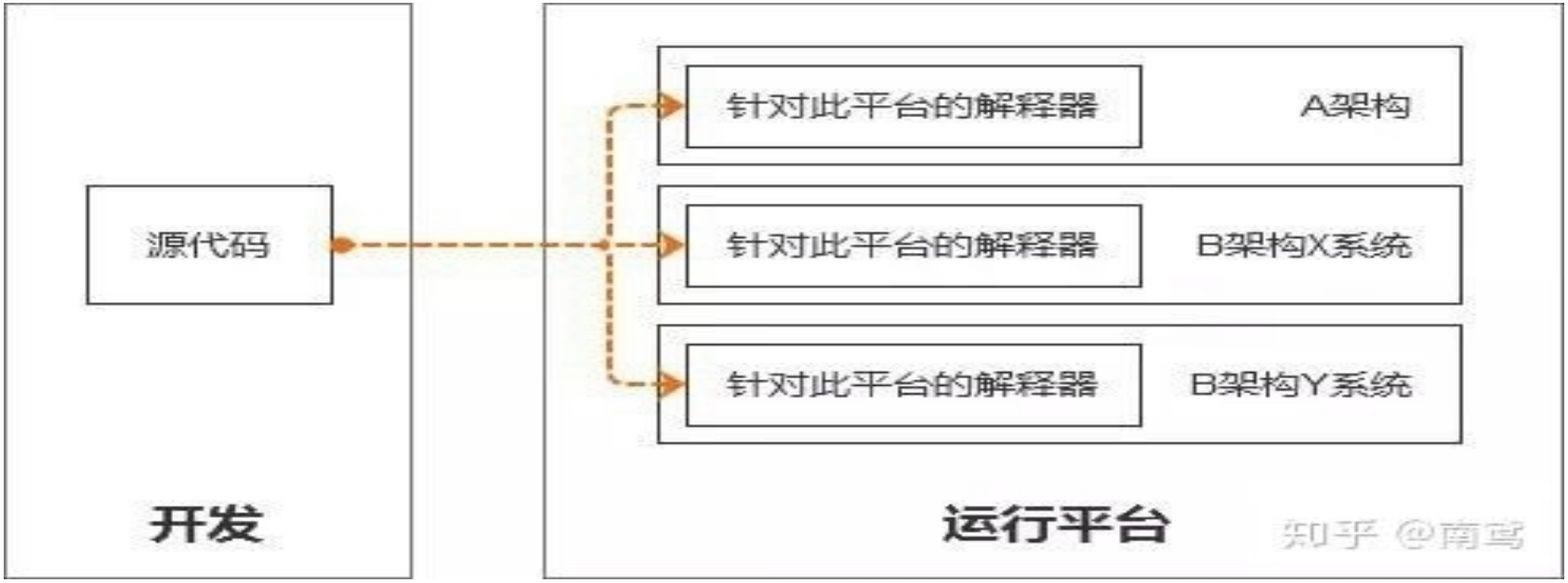


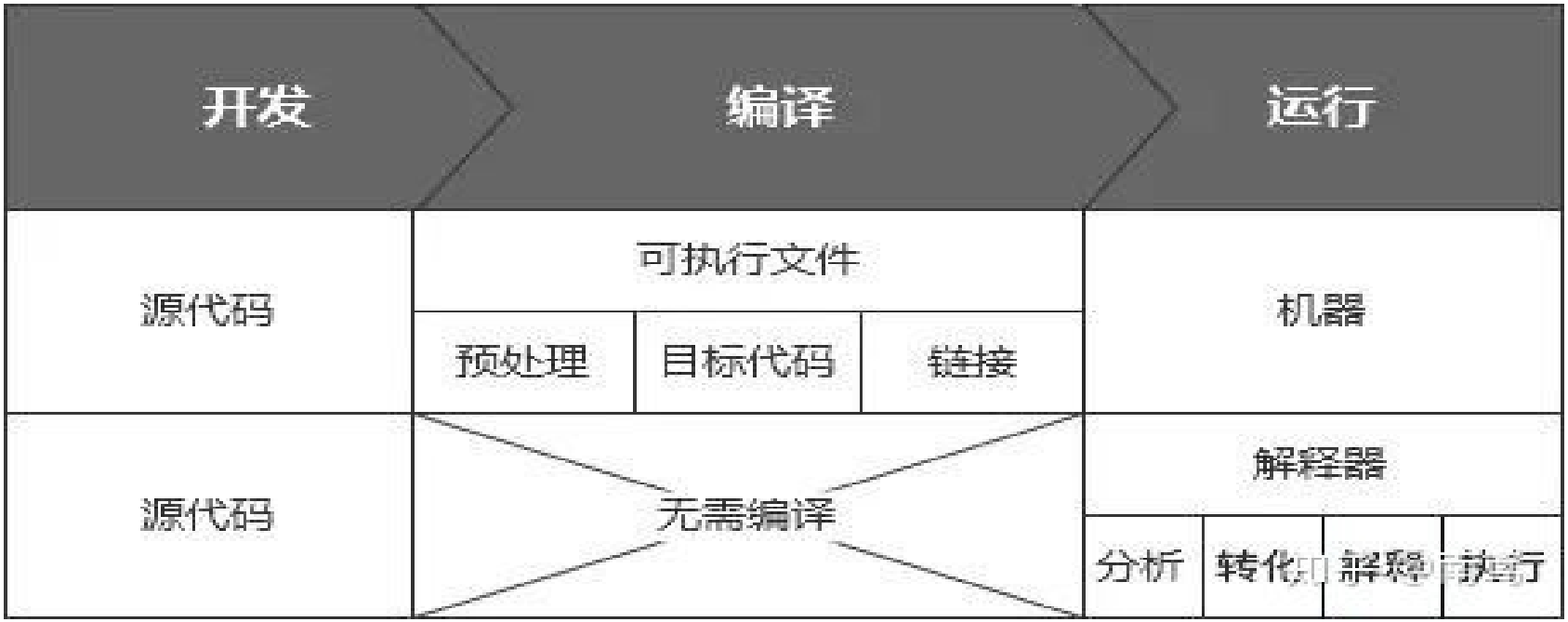


- **解释程序**

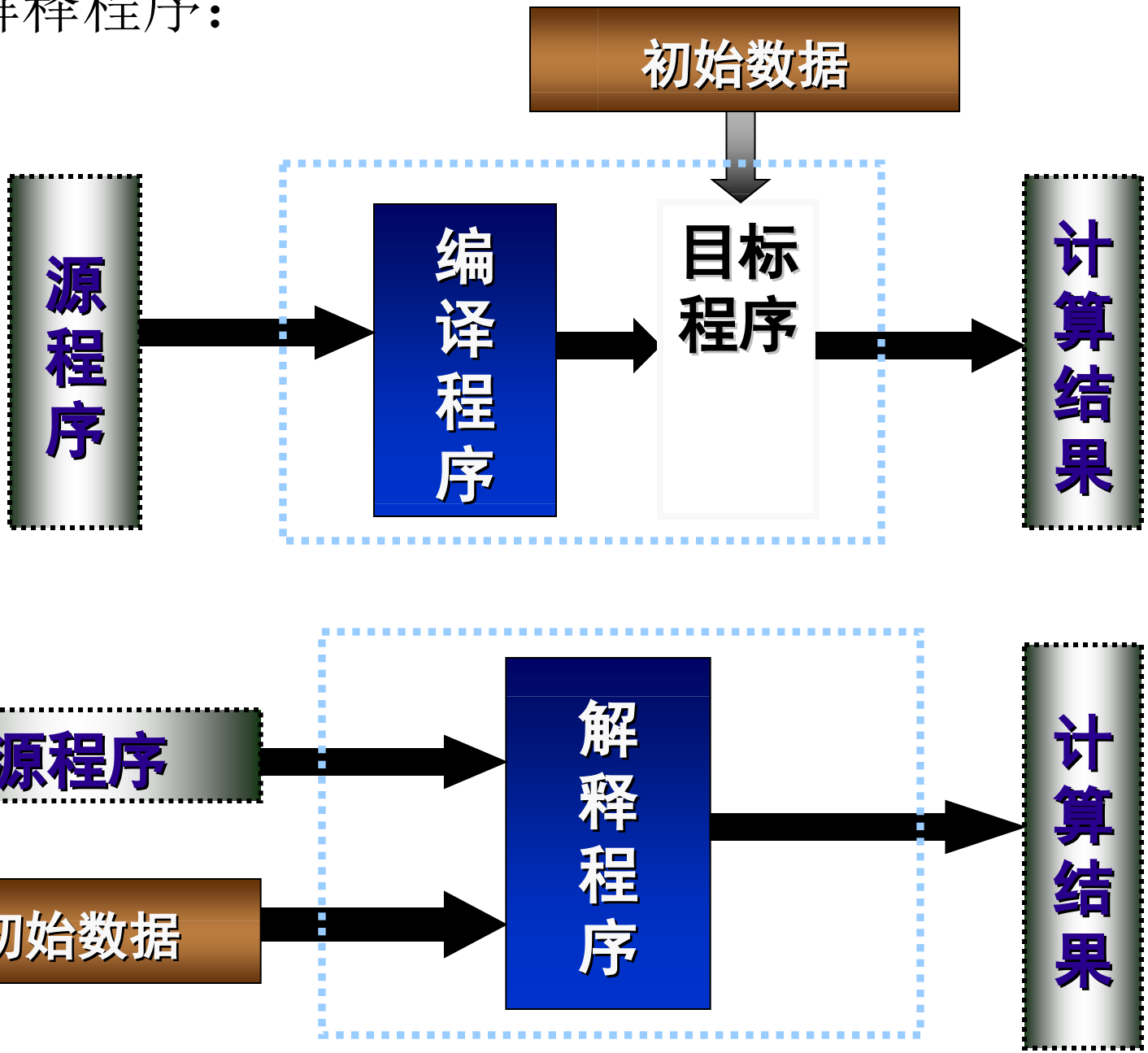
  对源程序进行解释执行的程序

  以源程序作为输入，但不产生目标程序，对源程序边解释边执行


针对此平台的解释器
A架构
源代码
针对此平台的解释器
B架构X系统
针对此平台的解释器
B架构Y系统
开发
运行平台
知乎 @南鸢


开发
编译
运行
源代码
可执行文件
预处理
目标代码
链接
机器
源代码
无需编译
解释器
分析
转化
解释
执行

编译和解释程序：



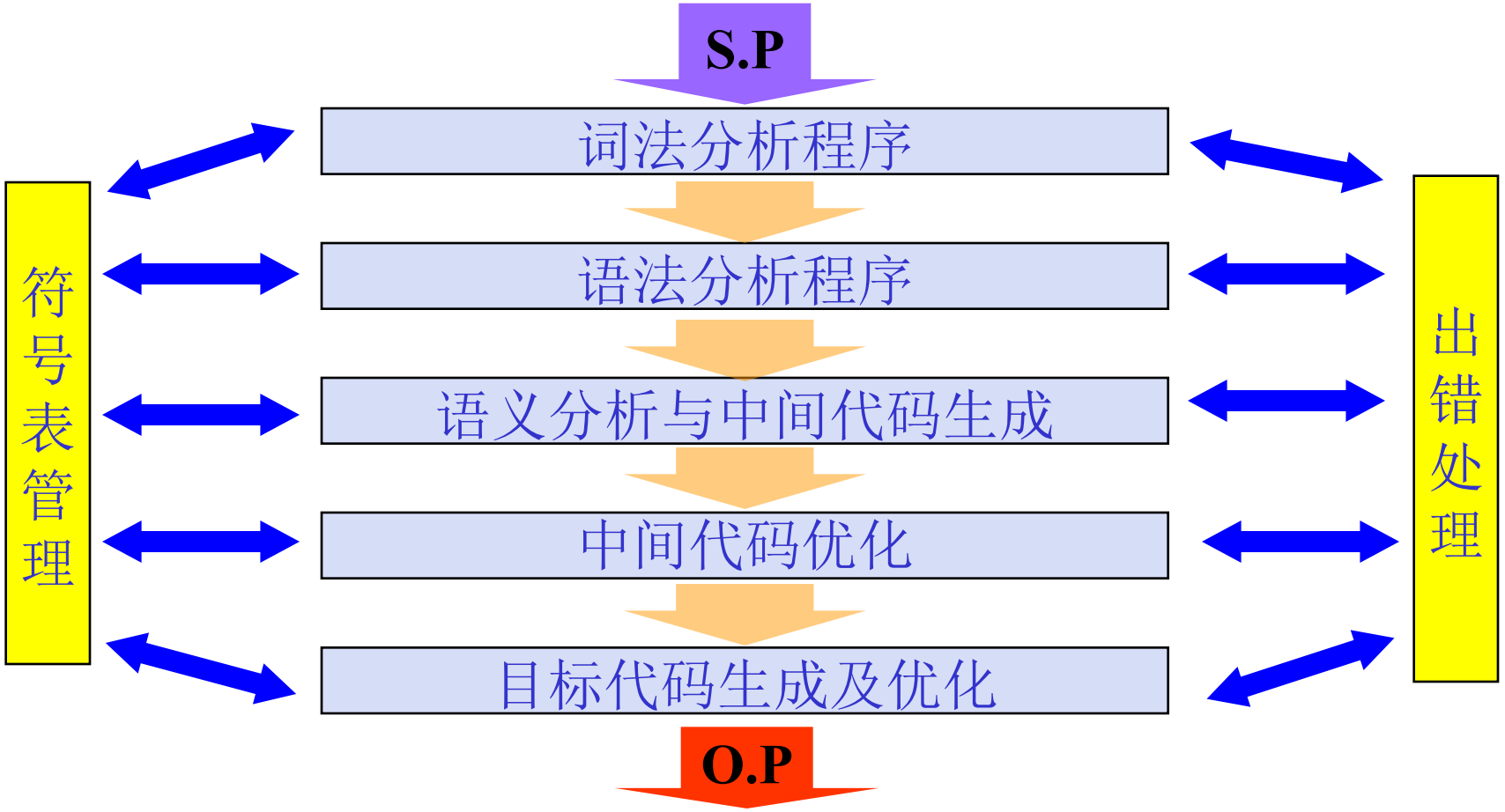
解释程序和编译程序的区别

	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

解释程序和编译程序的**根本区别**：
是否生成目标代码

1.2 编译程序的逻辑结构

典型的编译过程具有5个基本阶段



源程序看作什么形式便于进行处理?

1 词法分析

任务：分析和识别单词。

源程序是由字符序列构成的，词法分析从左到右一个字符一个字符地扫描源程序, 根据语言的词法规则识别出一个个单词符号（语义上关联的符号串），并以某种编码形式输出。

单词主要包括：保留字、标识符、运算符、分界符等。

例：对于如下的字符串,词法分析程序将分析和识别出9个单词：

X1	=	(	2.0	+	0.8	)	*	C1
1	2	3	4	5	6	7	8	9

例：一个C源程序片断：

```
int  a;
a = a + 2;
```

词法分析后可能返回：

单词类型	单词值
保留字	int
标识符(变量名)	a
界符	;
标识符(变量名)	a
算符(赋值)	=
标识符(变量名)	a
算符(加)	+
整数	2
界符	;

2 语法分析

任务：在词法分析的基础上，根据语法规则分析单词序列对应的语法成分，如表达式、各种说明、各种语句、函数等，并进行语法正确性检查。

例如，对于前面提到的例子X1 = (2.0+ 0.8) * C1

语法分析将**<变量>**、**<赋值操作符>**、**<表达式>**识别出来，进而将**<赋值语句>**识别出来，在识别过程中进行语法检查，若有错误，则应输出出错信息。

语法分析规则举例

规则

<赋值语句>::=<标识符>“=”<表达式>

<表达式>::=<表达式>“+”<表达式>

<表达式>::=<表达式>“*”<表达式>

<表达式>::=“(”<表达式>“)”

<表达式>::=<标识符>

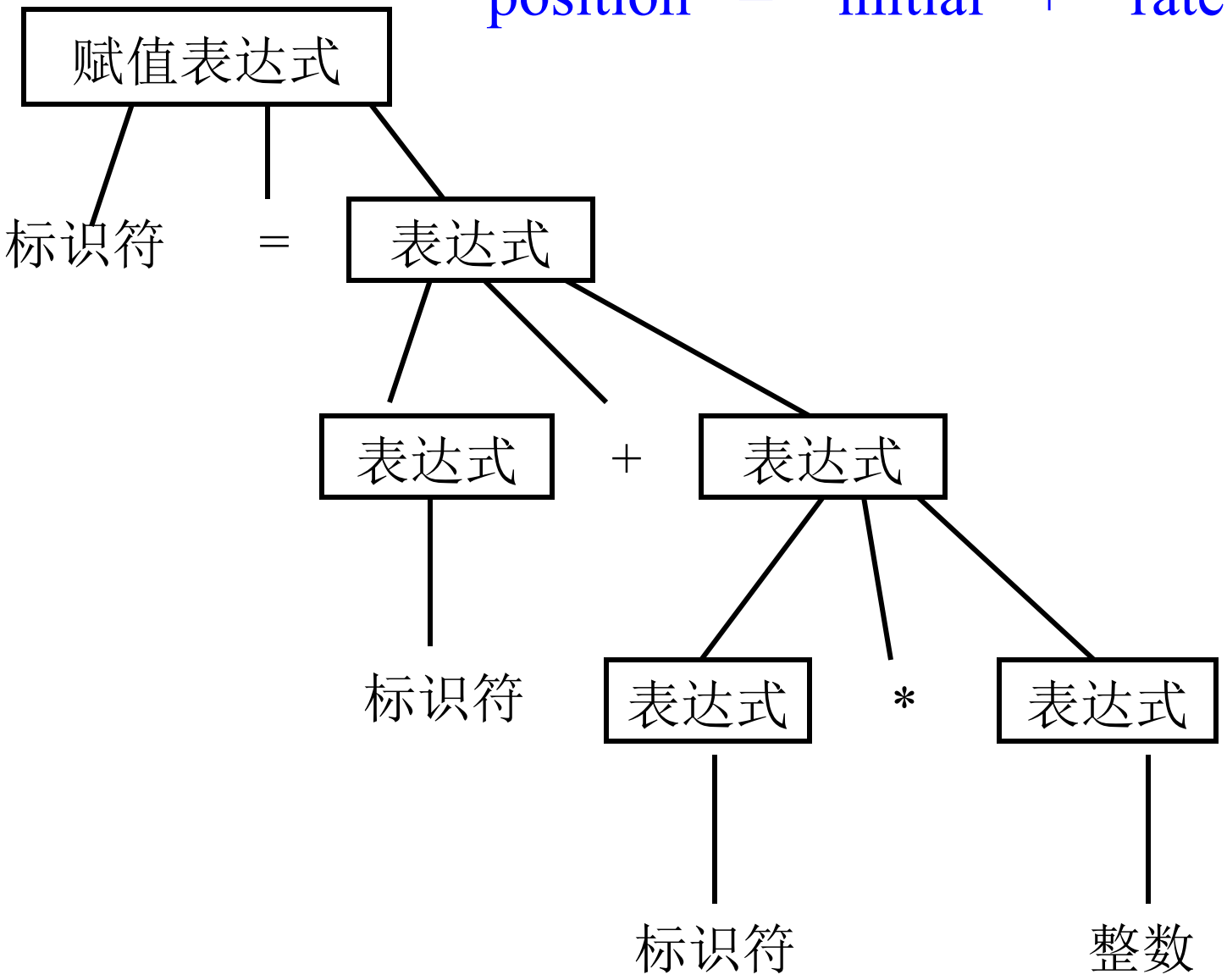
<表达式>::=<整数>

<表达式>::=<实数>

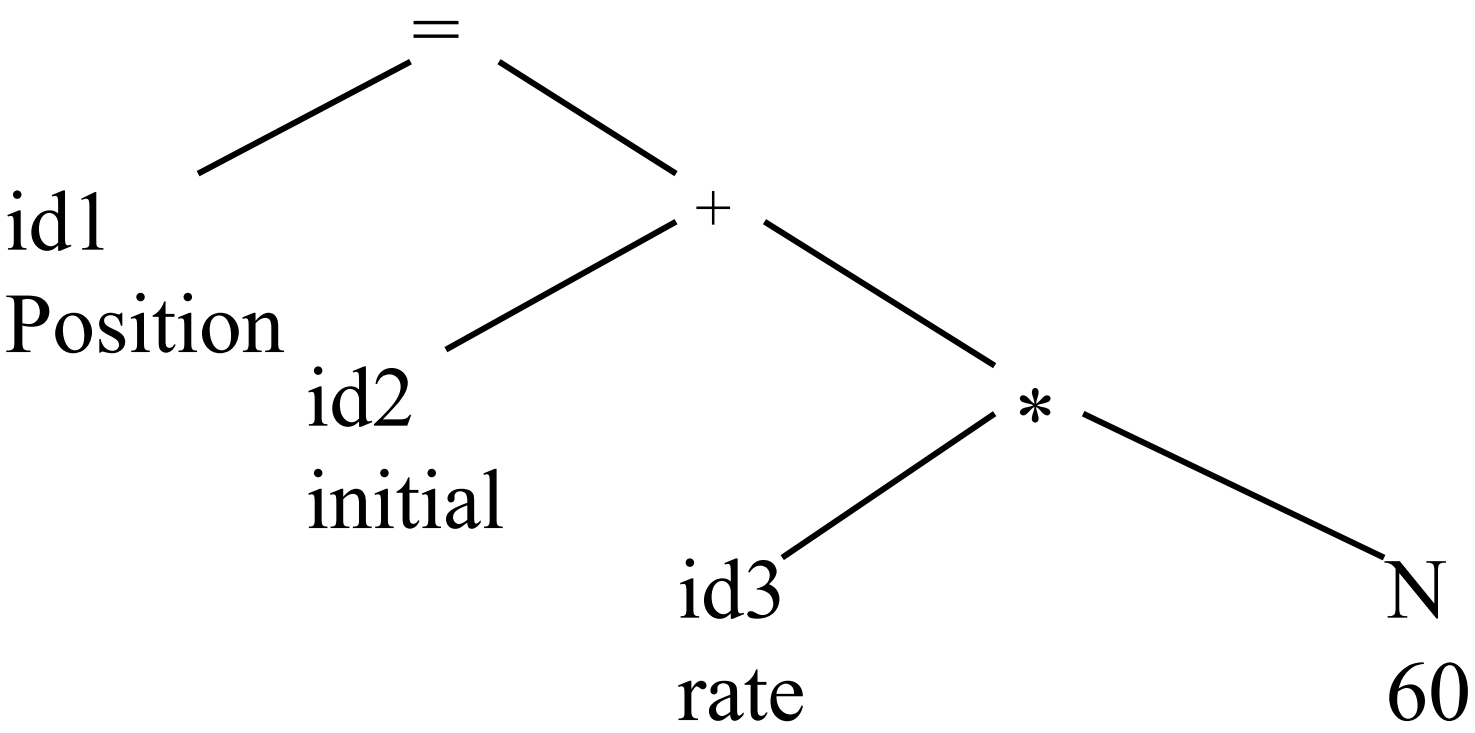
语法分析语法树举例

- 功能:层次分析. **依据**源程序的**语法规则**把源程序的单词序列组成语法短语(表示成语法树).
- 树的表现形式：
  - 语法分析树
  - 抽象语法树

position = initial + rate * 60



position = initial + rate * 60



3 语义分析和中间代码生成

语义分析：静态语义处理

审查源程序有无语义错误，为代码生成阶段收集类型信息。

中间代码生成：动态语义处理

- 中间代码：一种介于源语言和目标语言之间的内部表示形式。
- 生成中间代码的目的：
  〈1〉便于做优化处理；
  〈2〉便于编译程序的移植（中间代码不依赖于目标计算机）。
- 中间代码的形式：编译程序设计者可以自己设计，常用的有四元式、三元式等。

★ 四元式（三地址指令）

对于前面提到的例子 X1 = (2.0+ 0.8) * C1

	运算符	左运算对象	右运算对象	结果	
(	+	2.0	0.8	T1	)
(	*	T1	C1	T2	)
(	=	X1	T2		)

其中T1和T2为编译程序引入的临时变量

四元式的语义为：
$$2.0 + 0.8 \rightarrow T1$$
$$T1 * C1 \rightarrow T2$$
$$T2 \rightarrow X1$$

这样所生成的四元式与原来的赋值语句在语言的形式上不同，但语义上等价。

4 中间代码优化

优化：上面的四元式中第一个四元式是计算常量表达式值，该值在编译时就可以算出并存放在临时变量中，不必生成目标指令来计算，这样四元式可优化为：

编译时：　　2.0 + 0.8 → T1

运算符	左运算对象	右运算对象	结果
(*	2.8	C1	T1)
(=	X1	T1	)

5 目标代码生成及优化

由中间代码很容易生成目标代码（地址指令序列）。这部分工作与机器关系密切 ，所以要根据机器进行，也可以进行优化处理。

注意：在翻译成目标程序的过程中，要切记保持语义的等价性。

在上列五个阶段中都要做两件事：

（1）建表和查表；（2）出错处理；

所以编译程序中都要包括表格管理和出错处理两部分

★ 表格管理---符号表

在整个编译过程中始终都要贯穿着建表（填表）和查表的工作。即要及时的把源程序中的信息和编译过程中所产生的信息登记在表格中，而在随后的编译过程中同时又要不断的查找这些表格中的信息。

★ 出错处理

规模较大的源程序难免有多种错误，编译程序必须要有出错处理的功能。即能诊查出错误，并能报告用户错误性质和位置，以便用户修改源程序。出错处理能力的优劣是衡量编译程序质量好坏的一个重要指标。

源程序示例

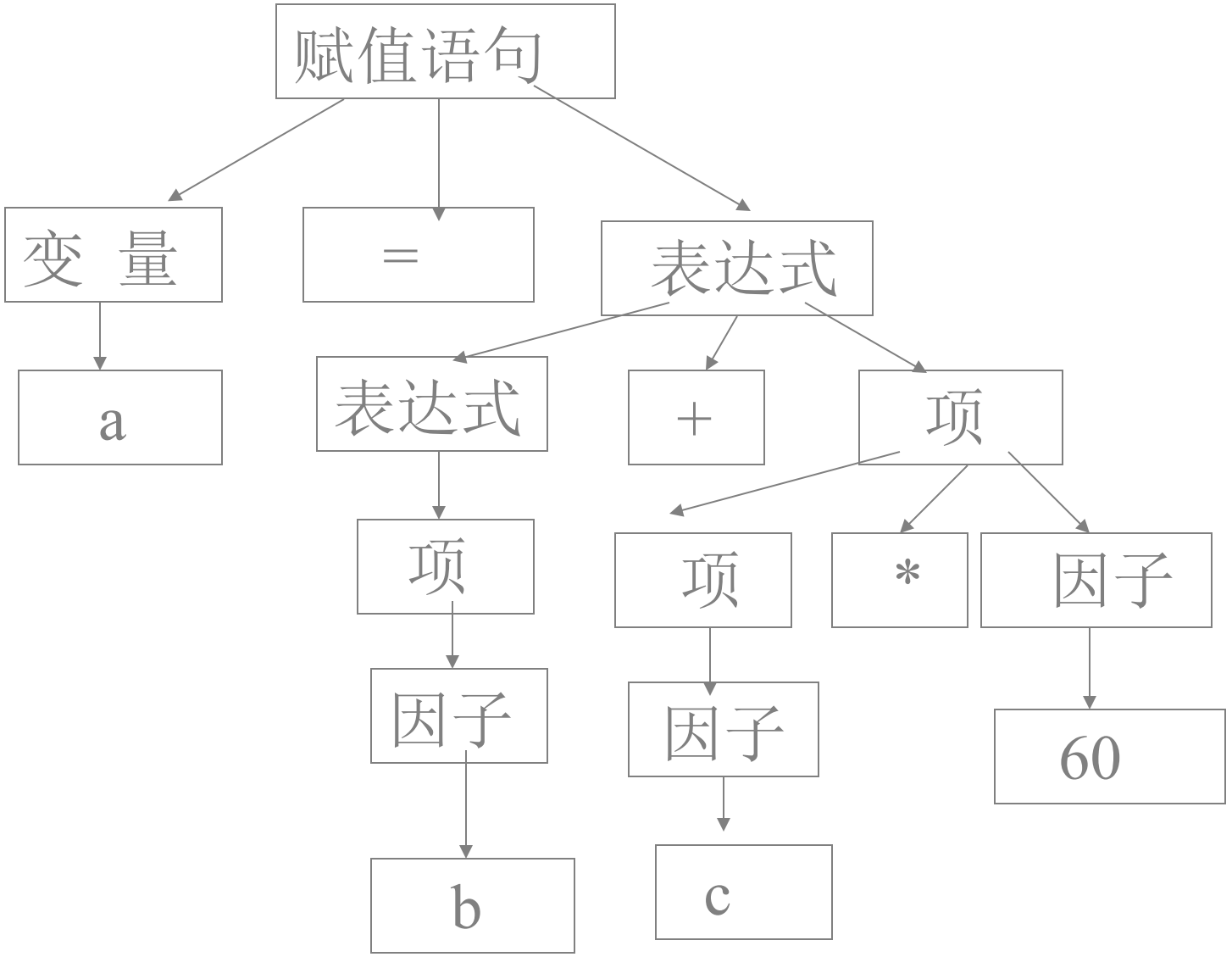
obj
linker

```
#include<stdio.h> //iostream.h
void main()
{
    float a,b,c;
    scanf("%f,%f,%f",&a,&b,&c);//cin>>
    a=b+c*60;
    printf("a=%f",a);//cout<<
}
```

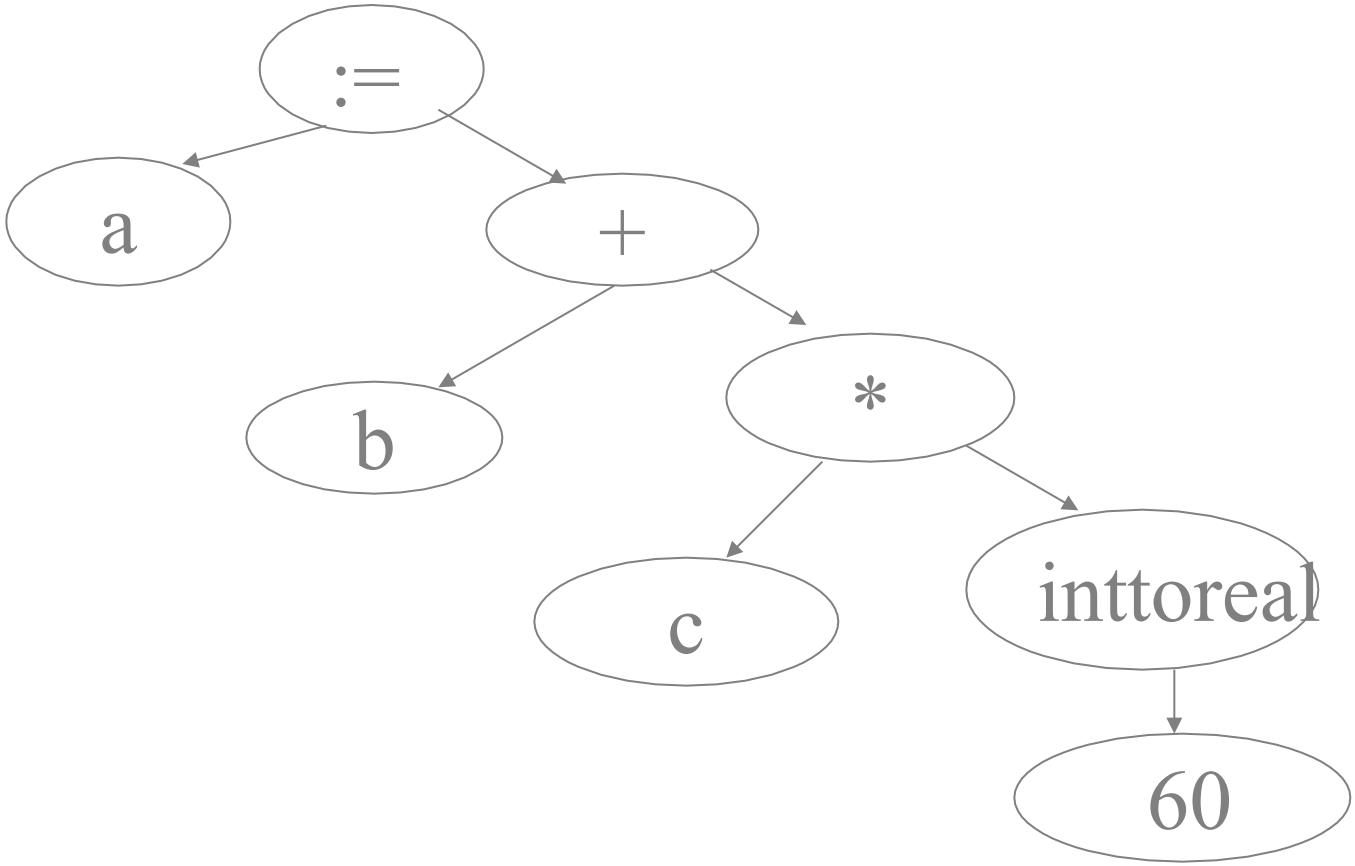
经词法分析源程序被加工成单词流

- <保留字，void> <保留字，main> <分隔符, (>
  <分隔符,) >
- <分隔符, { >
- <保留字,float> <标识符,a> <分隔符, , >
  <标识符,b> <分隔符, , > <标识符,c>
  <分隔符, ; >
- ……
- < 算符 , = > <标识符,b> < 算符, + >
- <标识符,c> <算符,* > <常数 ,60>……
- <分隔符, ; >
- …….
- <分隔符, } >

赋值语句经语法分析生成分析树

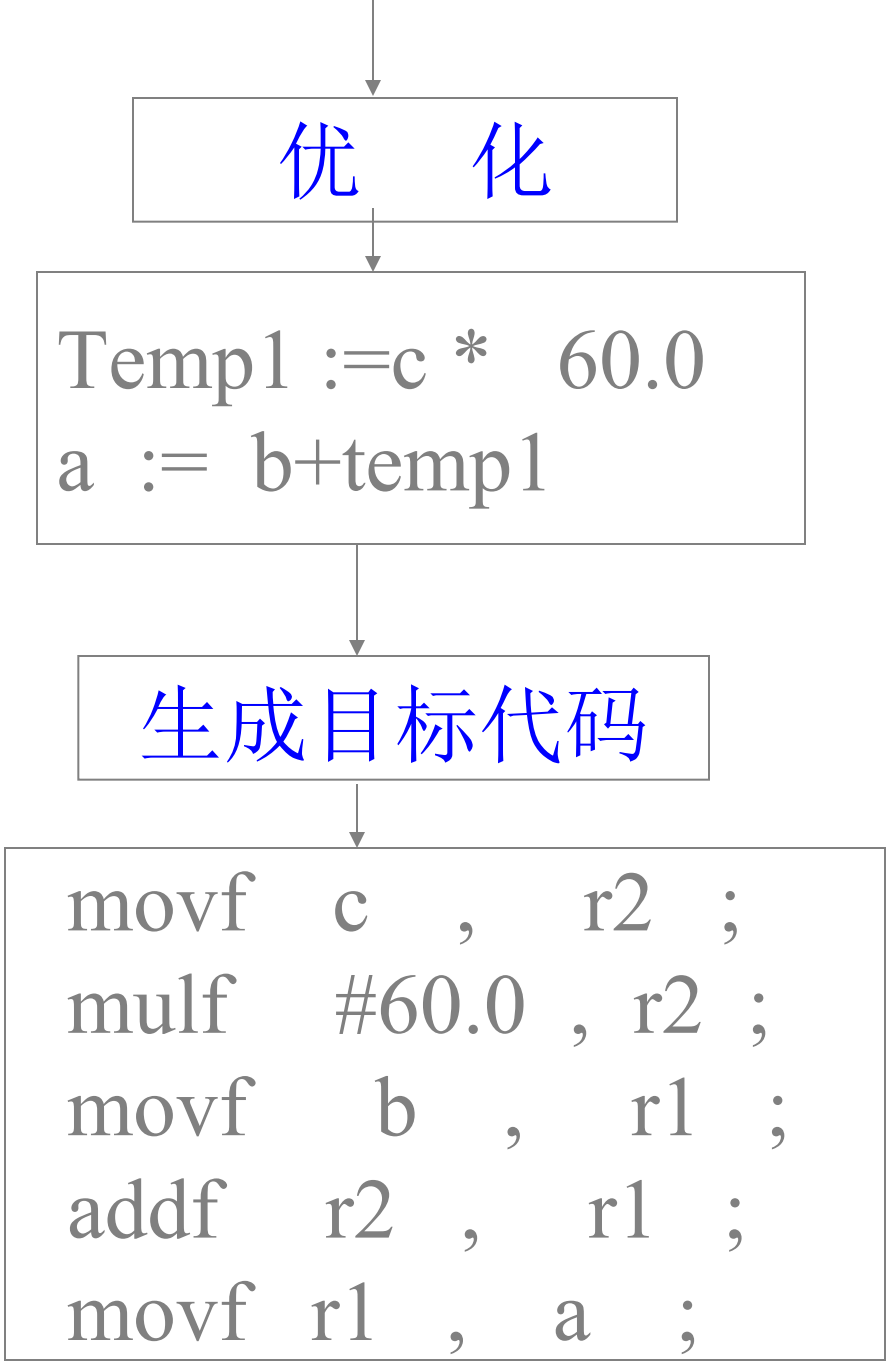


赋值语句经语义分析生成语法树



生成中间代码

- temp1:=inttoreal(60);
- temp2:=c * temp1;
- temp3:=b +temp2;
- a :=temp3;


优　化
Temp1 :=c *　60.0
a　:=　b+temp1
生成目标代码
movf　c　,　r2　;
mulf　#60.0　,　r2　;
movf　b　,　r1　;
addf　r2　,　r1　;
movf　r1　,　a　;

符号表

名　字	种　类	类　型	层　次	偏移量
m	过　程		0	
a	变　量	real	1	d
b	变　量	real	1	d+4
c	变　量	real	1	d+8

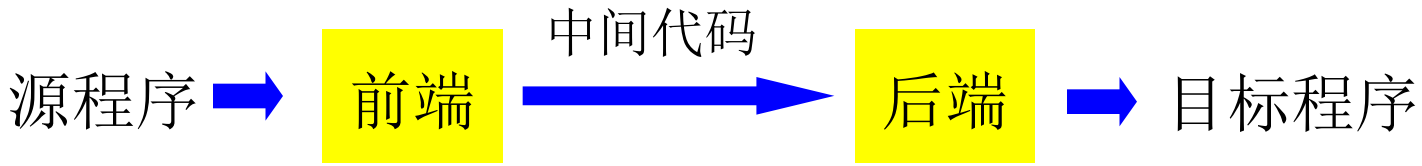
1.3 编译阶段的组合

前端和后端

根据编译程序各部分功能，将编译过程分成前端和后端。

前端：通常将与源程序有关的编译部分称为前端。
词法分析、语法分析、语义分析、中间代码生成（优化）
--------分析部分
特点：与源语言本身有关

后端：与目标机有关的部分称为后端。
目标代码生成、目标代码的优化
--------综合部分
特点：与目标机有关

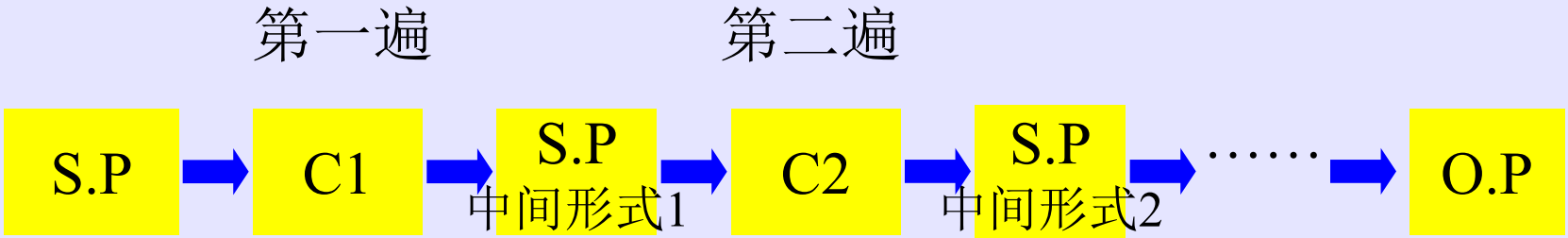


这一结构对于编译器的**可移植性**十分重要。某一编译程序的前端加上相应不同的后端则可以为不同的机器构成同一个源语言的编译程序。

如果不同编译程序的前端生成同一种中间代码，再使用一个共同的后端，则可为同一机器生成几个语言的编译程序。

遍（PASS）

遍：对源程序（或者源程序中间形式）从头到尾扫描一次，进行相应的加工处理 ，生成新的源程序中间形式或目标程序，通常称之为一遍。

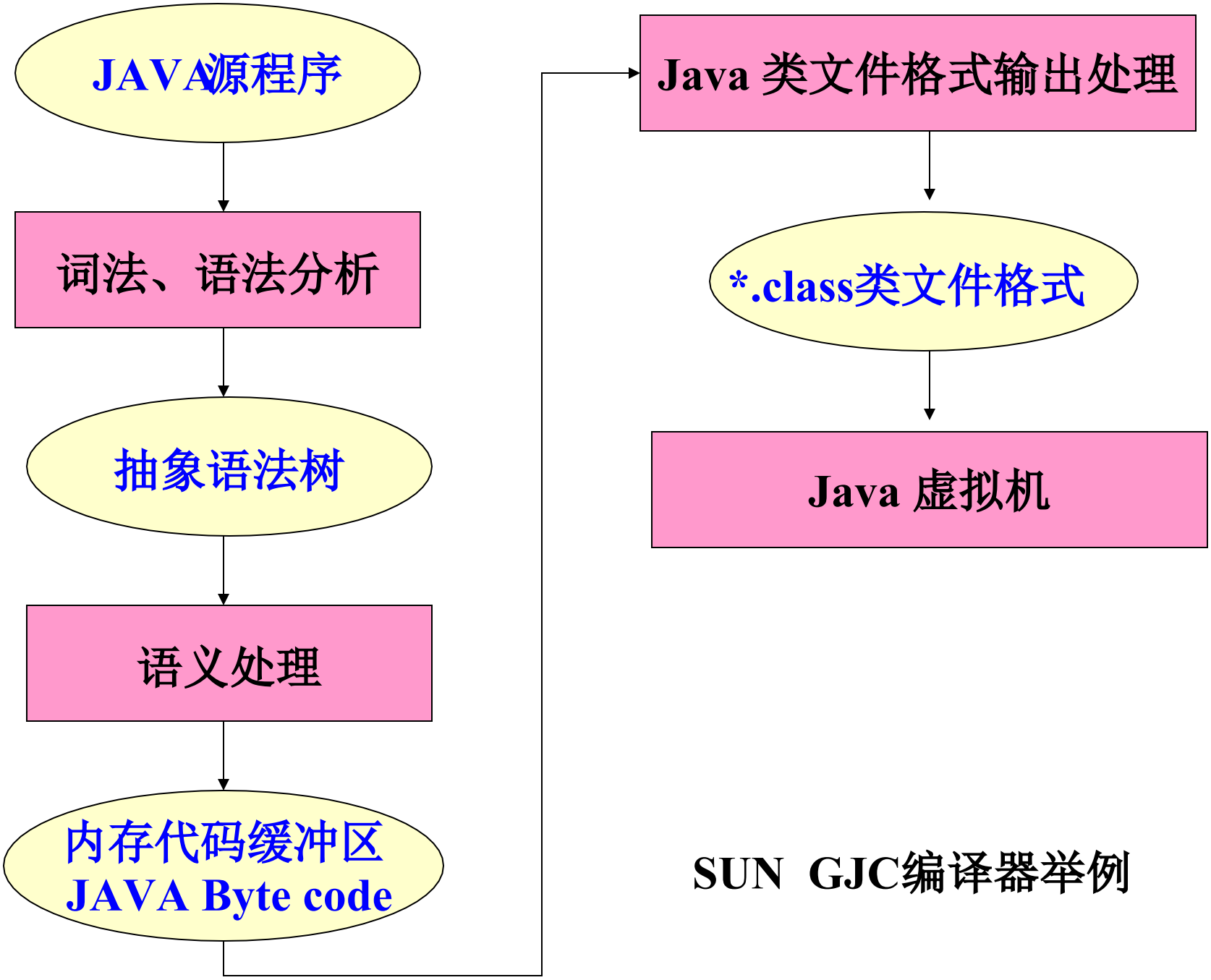


上一遍的结果是下一遍的输入，最后一遍生成目标程序。

★ **要注意遍与基本阶段的区别**

五个基本阶段：是将源程序翻译为目标程序在**逻辑上**要完成的工作。

遍：是指完成上述5个基本阶段的工作，要经过多次扫描处理。每一遍扫描可以完成一个或多个阶段的工作。

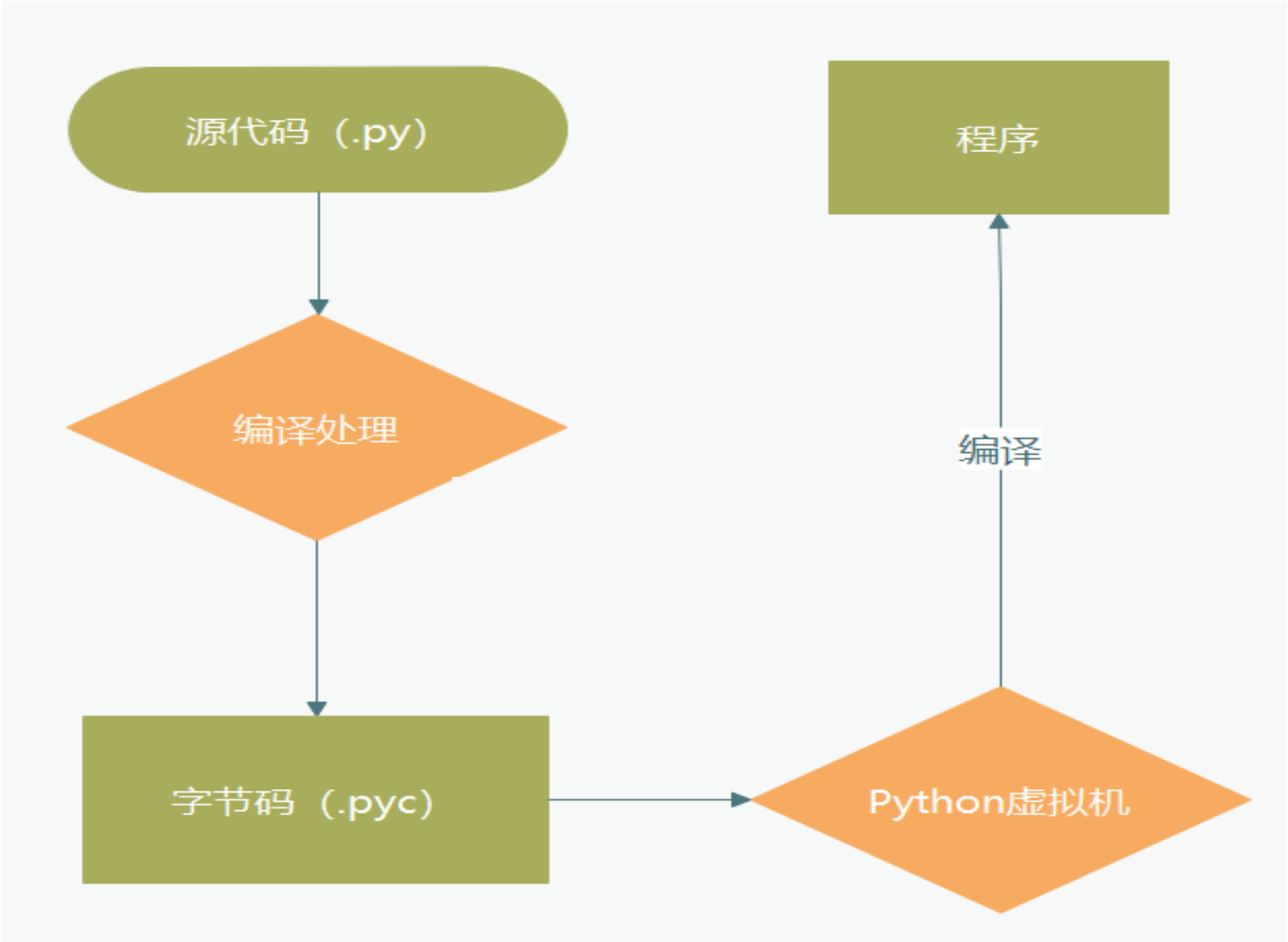


SUN GJC编译器举例

Python解释器

https://www.shiyanlou.com/courses/554

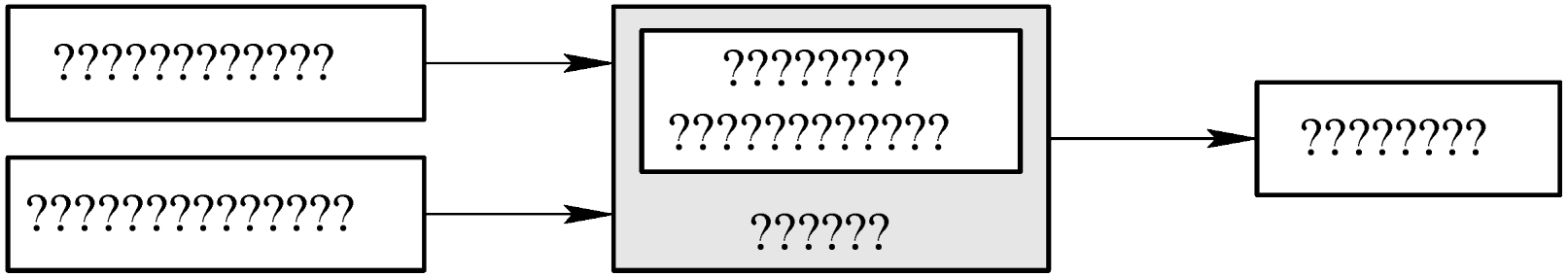
实验楼：Python实现Python解释器



1.4 编译程序实现的途径

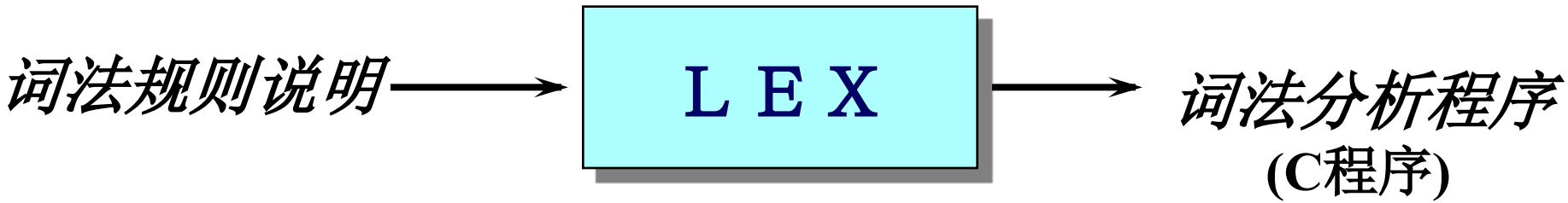
- 手工编写
  – 耗时，“效率”问题。
- 自动生成
  各个阶段基本上有相应的自动生成工具，如lex、yacc。
- 自展（bootstrapping)

编译程序的自动化技术



- **词法分析**：Lex由Mike Lesk在1975年左右为UNIX开发的，现在流行的是Free Software Foundation创建的Gnu complier package包中的Flex
- **语法分析**：Yacc (yet another complier-complier)由Steve Johnson在1975年左右为UNIX开发的，现在流行的是bison++

词法分析器的自动生成程序



输入：

词法（正规表达式）

识别动作（C程序段）

输出：

yylex( ) 函数

语法分析器的自动生成程序

输入：

语法规则（产生式）

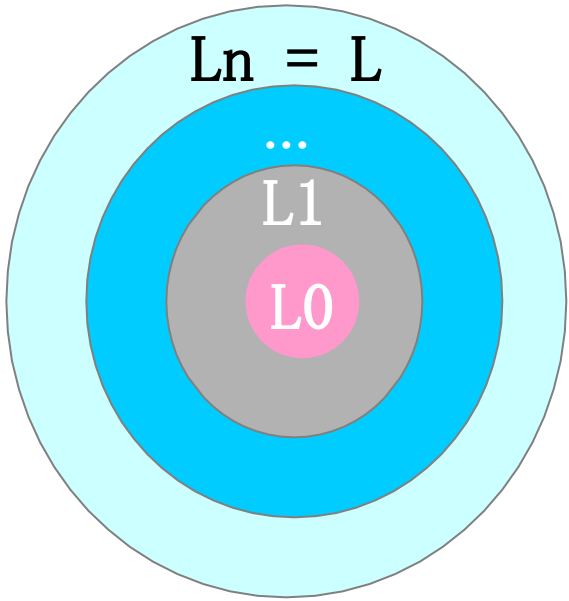
语义动作（C程序段）

输出：

yyparse( ) 函数

自展技术

- 先对语言的核心部分构造一个小小的编译程序（可用低级语言实现），再以它为工具构造一个能够编译更多语言成分的较大编译程序。如此扩展下去，就像滚雪球一样，越滚越大，最后形成人们所期望的整个编译程序。这种通过一系列的自展途径而形成编译程序的过程叫做自编译过程(编译程序的自展技术)。



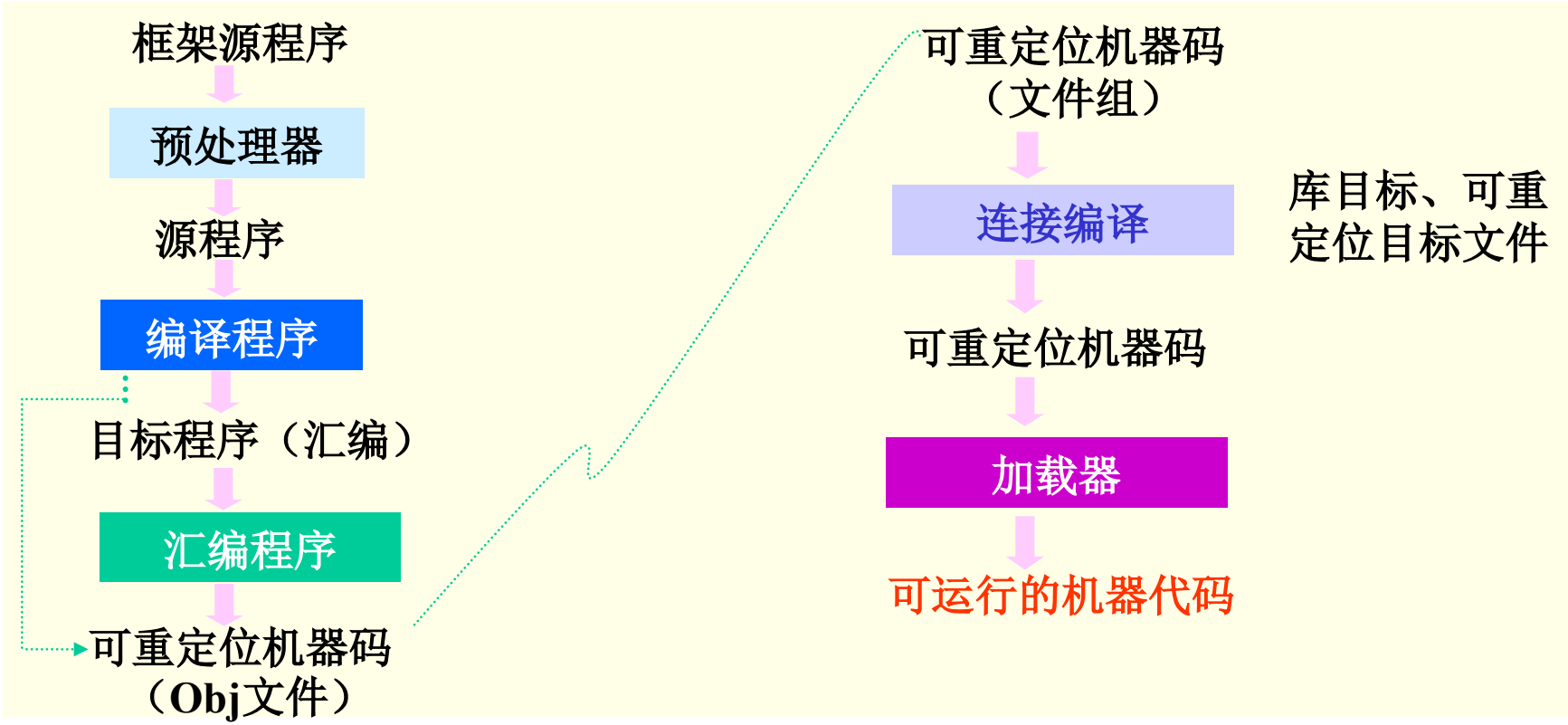
编译程序的移植技术

- 编译程序可以通过移植得到，即可以将一个机器（宿主机）上的一个具有自编译性的高级语言编译程序搬迁到另一个机器（目标机）上。而可移植性则是对这种搬迁过程中难易过程的一种度量。如果工作量不大，则称该程序是可以移植的；若移植一个程序的开销远远低于最初研制程序的开销，那么这种程序就是高度可移植的。

1.5、语言开发环境中的伙伴程序

- 编辑器（editor）
- 预处理器
- 编译器
- 连接程序
- 装配程序
- 调试程序

源程序：多文件、宏定义和宏调用，包含文件
目标程序：一般为汇编程序或可重定位的机器代码



1.6 编译程序的构造工具

- **词法分析**：Lex由Mike Lesk在1975年左右为UNIX开发的，现在流行的是Free Software Foundation创建的Gnu complier package包中的Flex
- **语法分析**：Yacc (yet another complier-complier)由Steve Johnson在1975年左右为UNIX开发的，现在流行的是bison++
- **基于属性文法的系统**：GAG(Generator based on Attributed Grammars), HLP(Helsinki Language Processor)
- **基于语义文法的系统**：CGSG(Compiler Generator for Semantics Grammars)

编译原理的最新发展

- 开发复杂的算法程序，用于优化和简化程序
- **IDE（Interactive development environment）**
- 并行编译技术：适合并行机和多处理机系统
- 硬件描述语言及其编译技术

编技术面对的挑战

新的问题

- New Computer Architectures: VLIW (Very Long Instruction Word), instead of CISC (Complex Instruction Set Computer) and RISC (Reduced Instruction Set Computer), Multi-core processor (superscalar execution, pipelining, and multithreading);
- Embedded environment – Small ROM & RAM, 特殊的指令集；
- Program Security.

面对的挑战

- 指令调度以支持指令的并行执行， 降低CPU能耗 （移动设备） ；
- 代码的空间和内存的优化， 以支持微处理器对ROM和RAM的限制；
- 程序的静态与动态分析以检测程序的安全漏洞；
- 程序附注信息的认证以解决可执行文件在网络上传输的安全问题(Proof-Carrying Code)；
- 认证编译器(Certified Compiler): 保证相关性质经编译后还能保持。

小　结

- 什么是编译程序, 同解释程序的区别
- 编译程序的逻辑结构, 分为那几个阶段
- 编译阶段的组合, 怎么组合？组合的依据
- 编译程序实现的途径,如何实现,如何移植

注意

- https://mooc1-1.chaoxing.com/course/219083741.html
- 矿大在线课程章节题目需要按时完成