编译原理

第一章 引 论

颜波

复旦大学计算机科学技术学院
byan@fudan.edu.cn
http://homepage.fudan.edu.cn/boyan

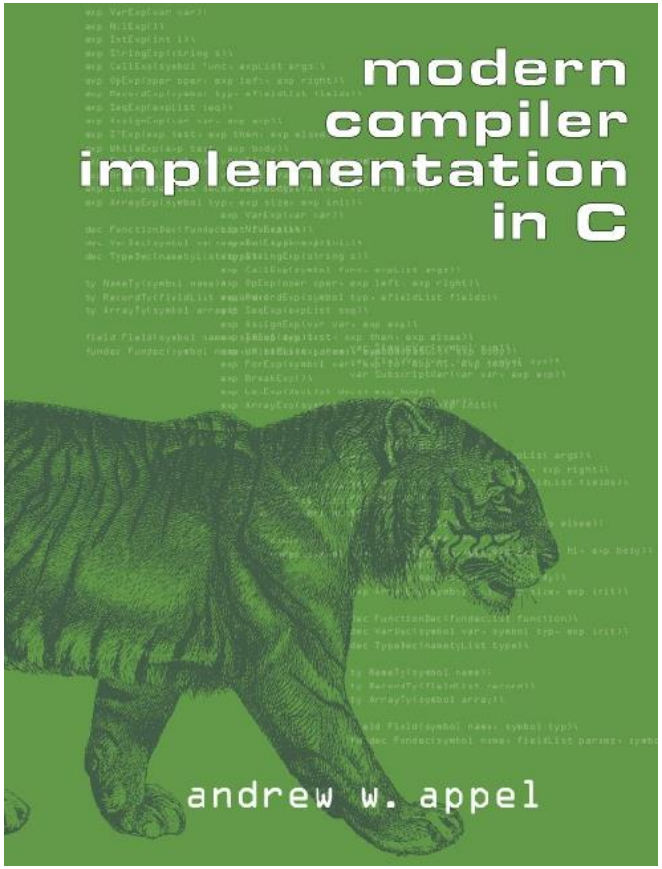
教材

陈火旺等,

“程序设计语言编译原理（第3版）”

国防工业出版社

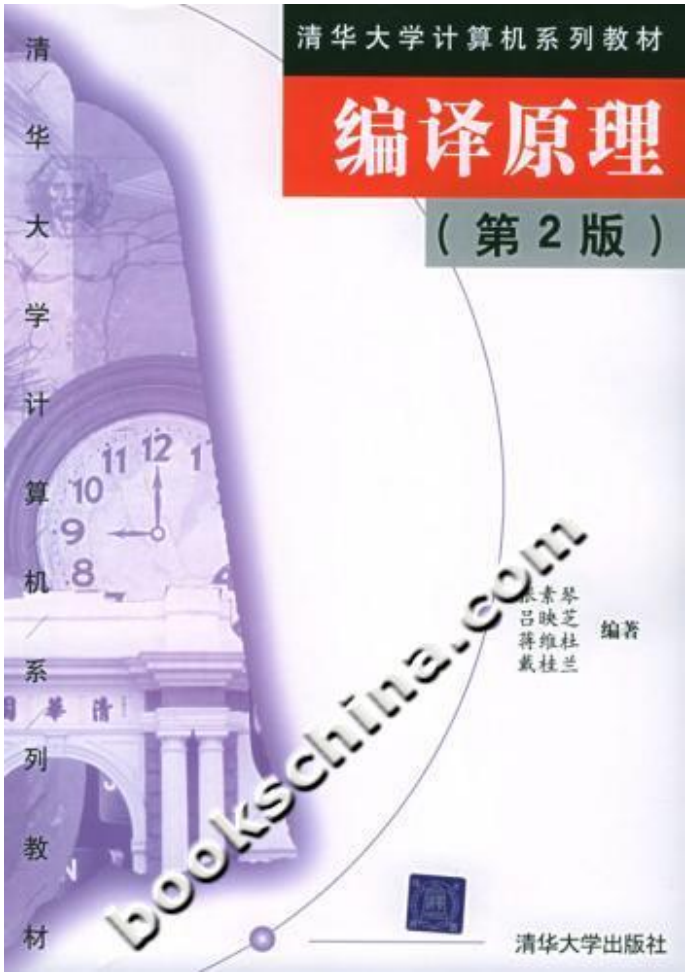
参考教材1



Andrew W. Appel,

“Modern Compiler Implementation in C”

参考教材2

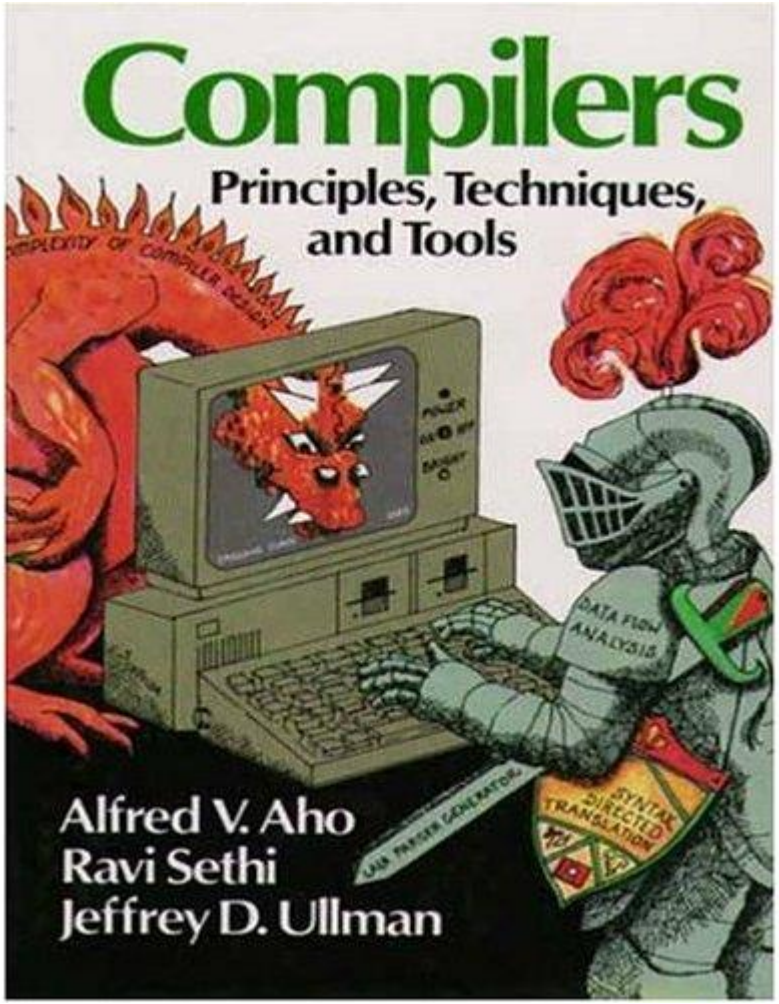


张素琴等，

“编译原理 （第二版）”

清华大学出版社

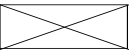
参考教材3



“Compilers Principles, Techniques, and Tools”

课程信息

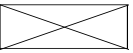
- 上课：周一，6-8　　Z 2307B
- 助教：林青（Email: 18210240028@fudan.edu.cn）
  王峻逸（Email: 18110240005@fudan.edu.cn）
- 上机实验：周五 3-4　计算机学院机房



考核方式

- 平时成绩40%
  - 作业、课堂练习等10%,
  - 课程设计30%

- 考试成绩60% (闭卷考试)

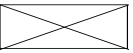


概 要

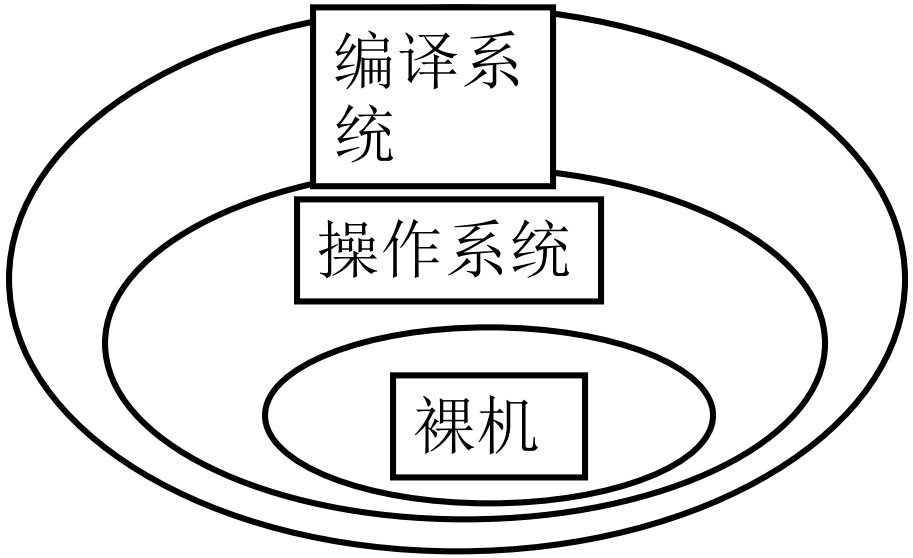
- 本课程介绍程序设计语言编译程序构造的基本原理和基本实现技术.

- 编译理论与方法
  - 计算机科学与技术中理论和实践相结合的最好典范
  - ACM 图灵奖，授予在计算机技术领域作出突出贡献的科学家
    - 程序设计语言、编译理论与方法约占1/3

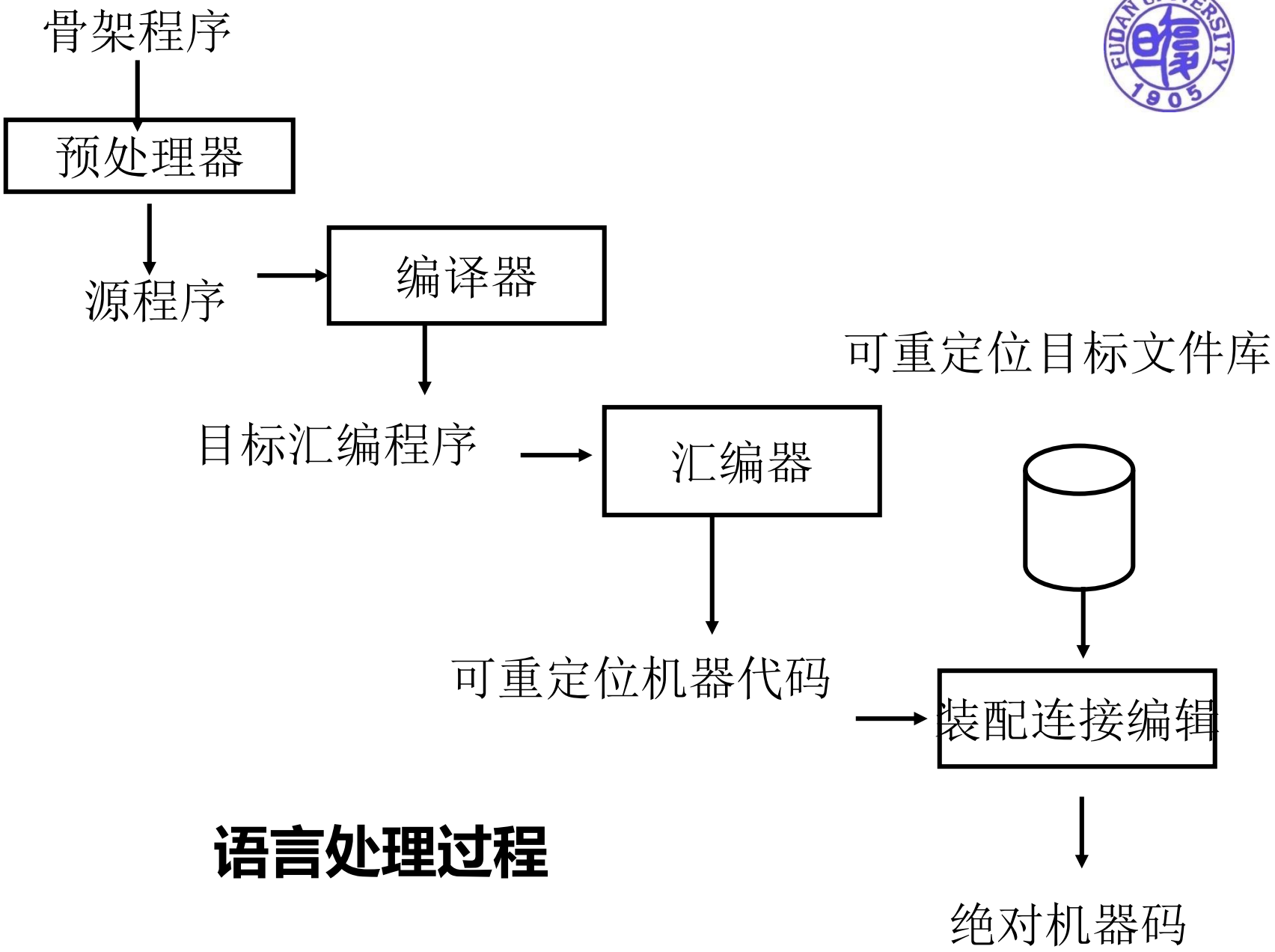
- 意义:
  - 学习编译程序构造原理，技术;
  - 更好地理解高级语言, 加深对程序内部执行过程的理解;
  - 利于编写高效的程序



软件分类



- 软件：计算机系统中的程序。
- 软件语言：用于书写软件的语言。它主要包括需求定义语言，功能性语言，设计性语言，程序设计语言以及文档语言。
- 语言处理系统：把软件语言书写的各种程序处理成可在计算机上执行的程序。
- 系统软件：居于计算机系统中最靠近硬件的一层，其他软件一般都通过系统软件发挥作用。他和具体的应用领域无关，如操作系统。


骨架程序
预处理器
源程序
编译器
目标汇编程序
汇编器
可重定位目标文件库
可重定位机器代码
装配连接编辑
绝对机器码

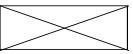
语言处理过程

程序设计语言

- 程序设计语言分成两大类，
  - 低级语言，包括机器语言和汇编语言，主要是面向机器的。
  - 高级语言,它是面向应用的，分成很多种，如FORTRAN、Pascal、C、Ada、Java等。

- 机器语言本身是由0和1组成的，符合计算机的硬件特性，因此能够直接执行。但用机器语言编写程序很不方便且容易出错，因此就用助记符代替机器语言，产生了汇编语言。

- 汇编语言比机器语言在可读性方面有了进步，但是其依赖具体机器的特性无法改变，给程序设计语言增添了难度。



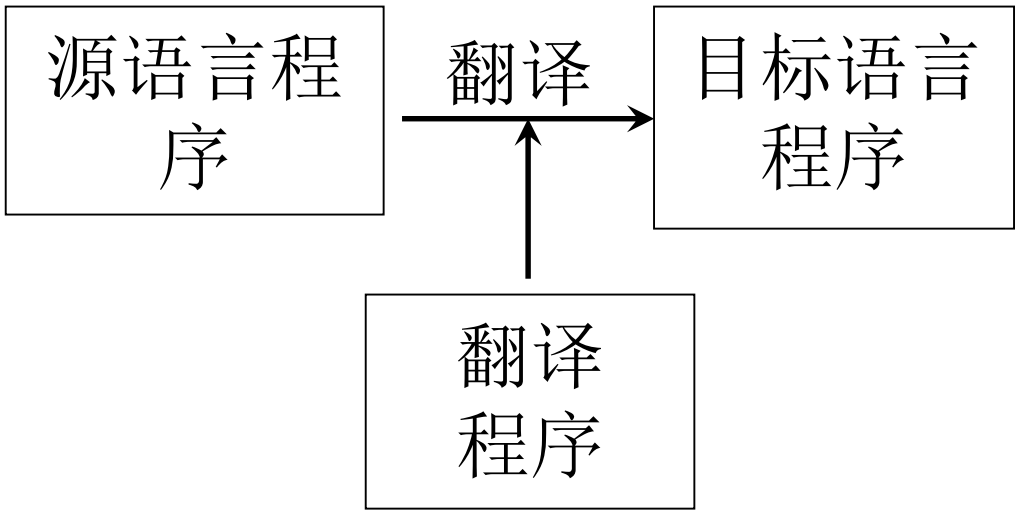
程序设计语言

- 通用程序设计语言与汇编语言（包括机器指令）
  - Pascal语句：x := a+b;
  - 汇编指令：

十六进制代码	汇编指令
A10002	MOV AX, [A]
8B1E0202	MOV BX, [B]
01D8	ADD AX, BX
A30402	MOV [X], AX

什么是编译程序

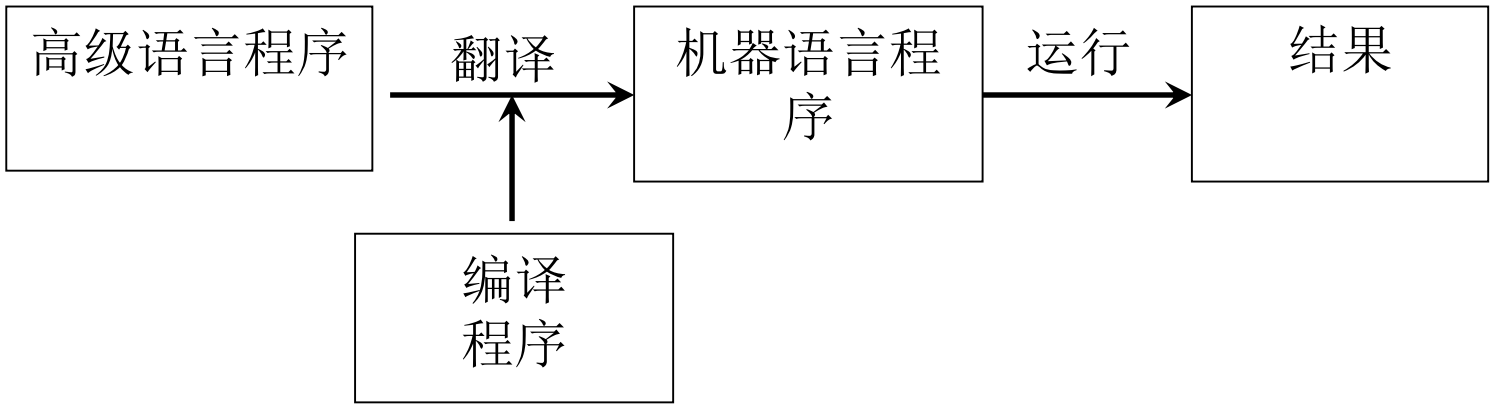
- 高级语言不能直接在机器上运行，它不是面向机器，而是面向应用的，因此，要想让高级语言运行必须有编译程序。
- 翻译程序：把某一种语言程序(称为源语言程序)等价地转换成另一种语言程序(称为目标语言程序)的程序。



什么是编译程序

- 编译程序(compiler)

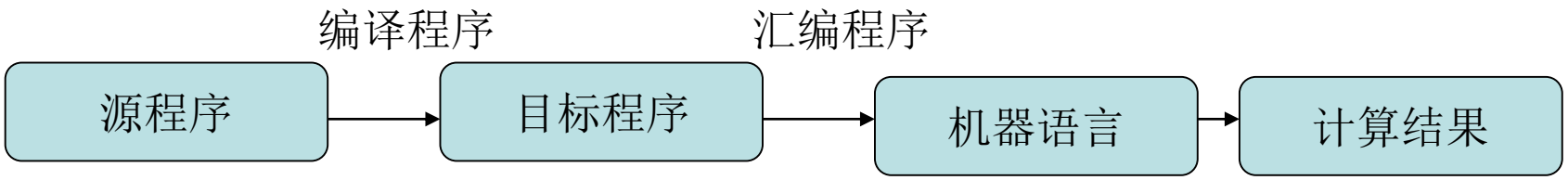
  把某一种高级语言程序等价地转换成另一种低级语言程序(如汇编语言或机器语言程序)的程序。



- 分类：
  - -诊断编译程序：用于程序开发和调试；
  - -优化编译程序：着重于提高目标代码效率；
  - -交叉编译程序：可产生不同于宿主机的机器代码；
  - -可变目标编译程序：不需重写其中与机器无关的部分就能改变目标机；

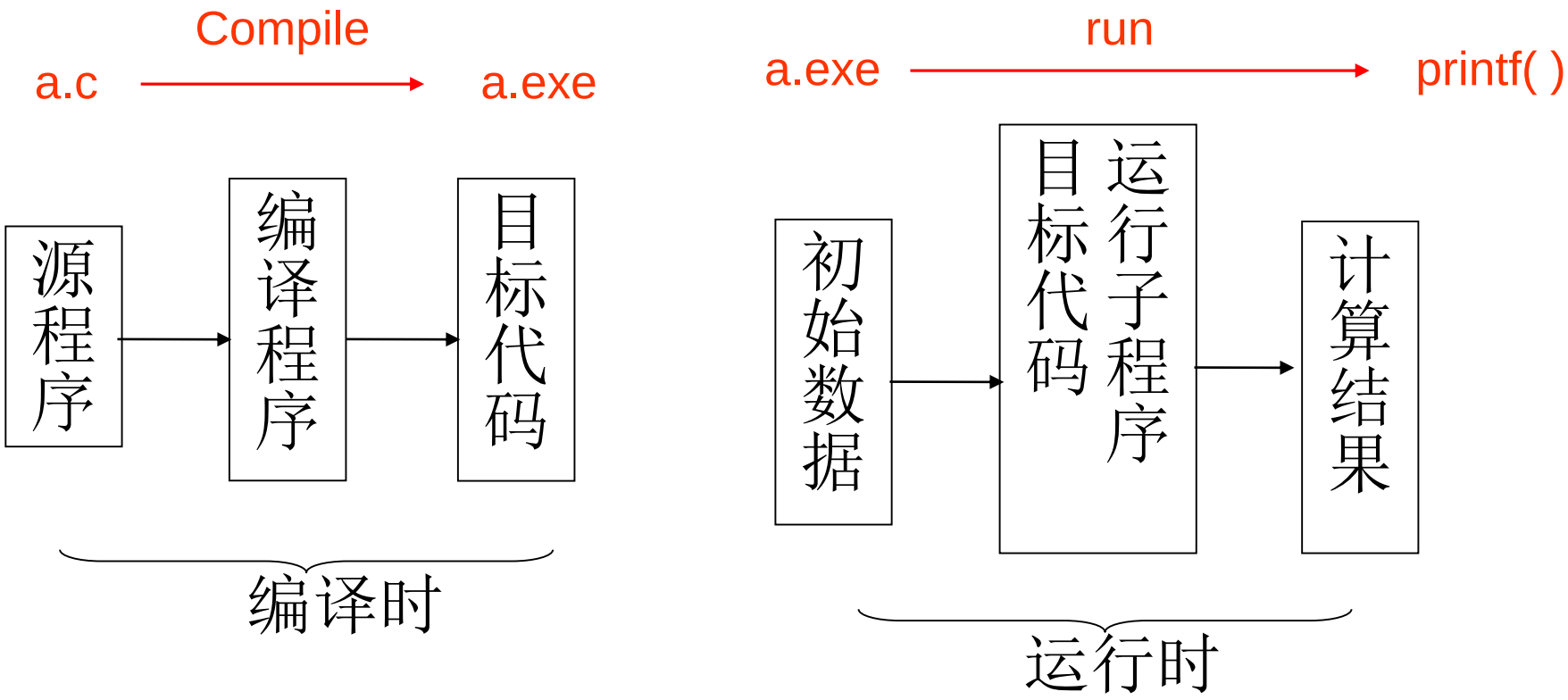
什么是编译程序

- 高级语言程序的执行通常分为两个阶段，
  - 编译阶段，编译阶段将源程序变换成目标程序。
  - 运行阶段，运行阶段则由所生成的目标程序连同运行系统（数据空间分配子程序、标准函数程序等）接受程序的初始数据作为输入，运行后输出计算结果。
    - 如果目标程序是汇编语言的形式，则需要在编译阶段和运行阶段之间加一个汇编阶段。



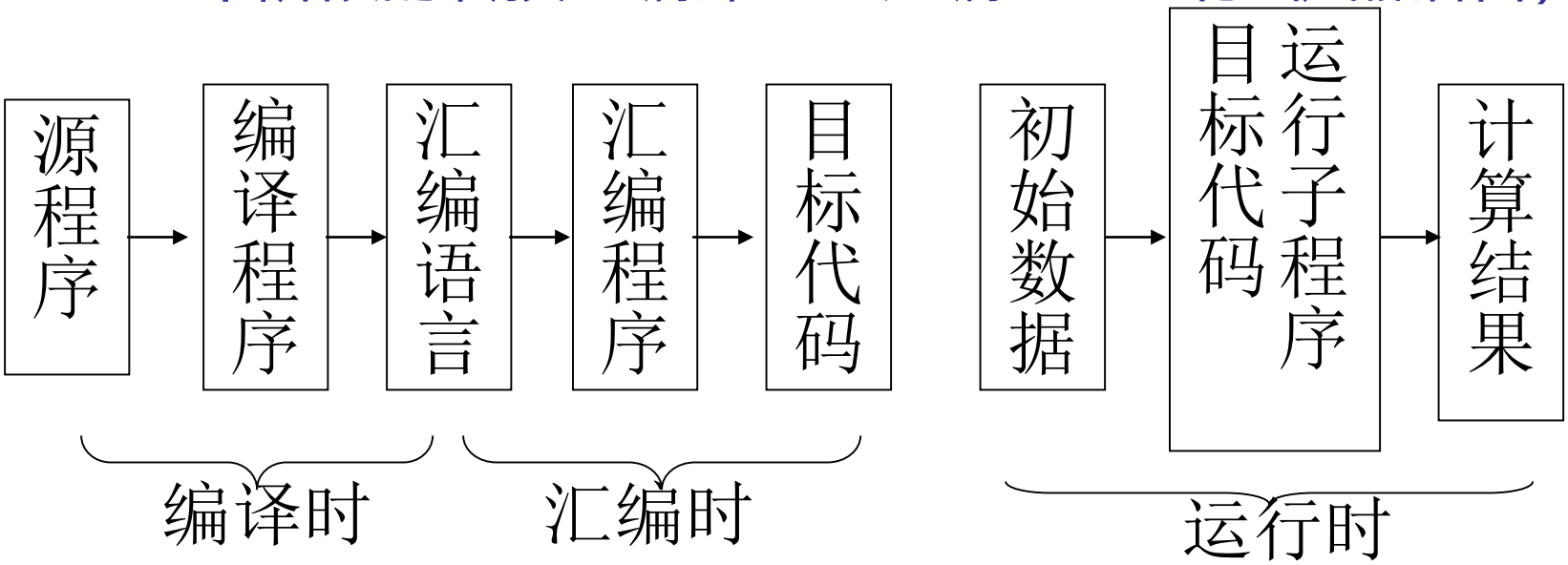
程序设计语言的转换与编译

- 编译的转换过程
  - 两阶段转换：编译——运行(机器语言)



程序设计语言的转换与编译

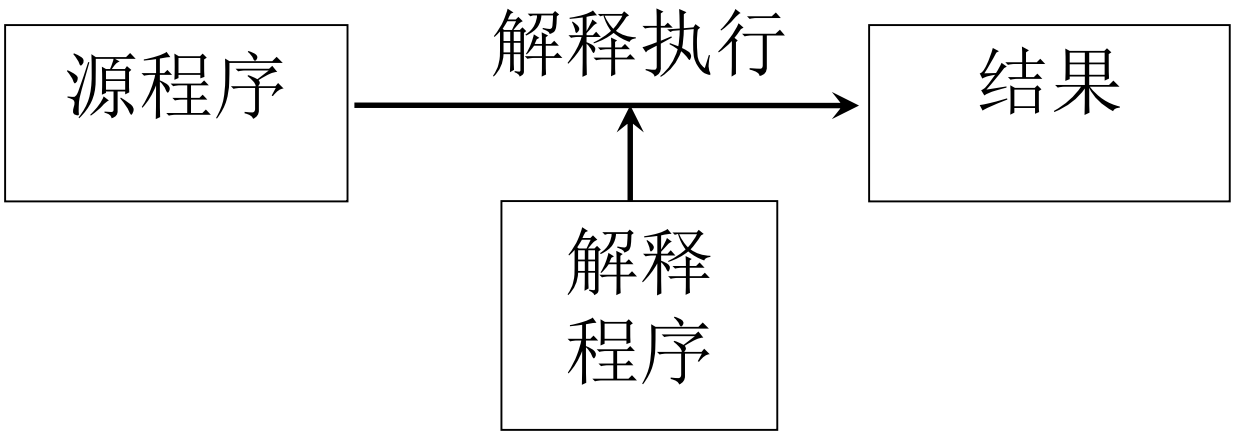
- 编译的转换过程
  - 三个阶段的转换：编译——汇编——运行(机器语言)



- 目标代码(机器语言)
  - .exe文件：直接运行。（绝对机器代码 ）
  - .obj文件：需要link。为防止C内核过大，使用“#include”。（可重定位机器代码 ）

解释程序

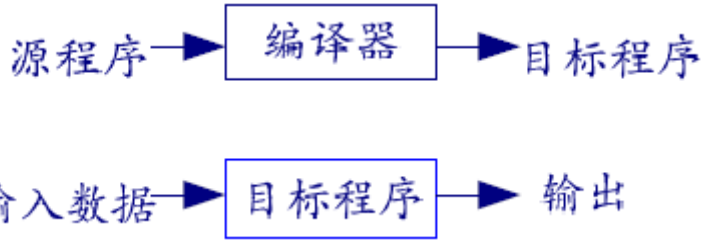
- 解释程序：把源语言写的源程序作为输入，但不产生目标程序，而是边解释边执行源程序本身。



解释程序与编译程序

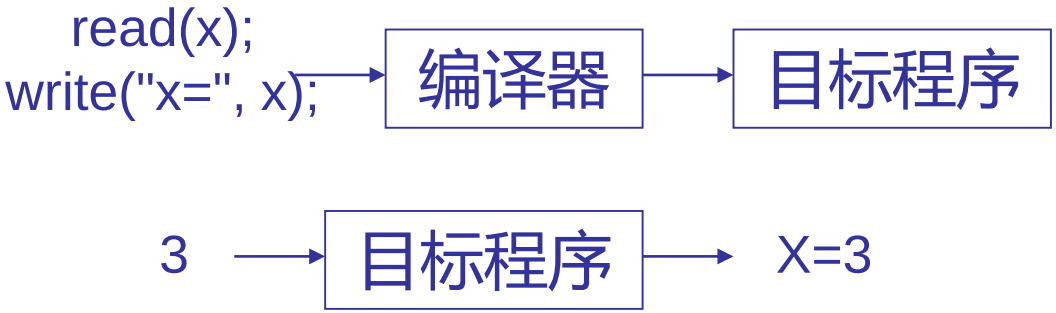
- 语言翻译的两种基本形态：

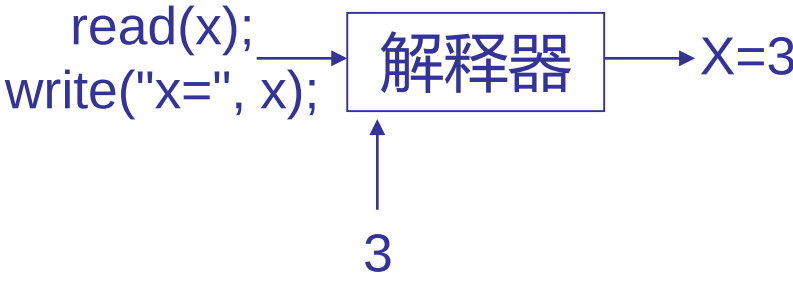
1.先翻译后执行



2. 边翻译边执行

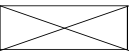
例: 假设有源程序： read(x); write("x=", x);





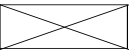
解释程序与编译程序

- 解释程序与编译程序的主要区别是：
  - 编译程序将源程序翻译成目标程序后再执行目标程序，
  - 而解释程序是逐条读出源程序中的语句并执行，即在解释程序的执行过程中并不产生目标程序。
- 特点：
  - 编译器：工作效率高，即时间快、空间省；交互性与动态特性差、可移植性差。大多数PL采用此种方法翻译；
  - 解释器：工作效率低，即时间慢、空间费；交互性与动态特性好、可移植性好。早期的Basic和现在的Java等。
- 基本功能：二者相同；
- 所采用的技术：从翻译的角度来讲，两种方式所涉及的原理、方法、技术相似。



编译过程

- 编译程序的工作一般分为五个阶段：
  - 词法分析
  - 语法分析
  - 语义分析和中间代码产生
  - 优化
  - 目标代码产生



1. 词法分析

- 任务：输入源程序，对构成源程序的字符串进行扫描和分解，识别出一个个单词符号。

- 依循的原则：构词规则。
  - 在高级语言中，所谓单词，就是指逻辑上紧密相连的一组字符，这些字符具有集体含义。单词是语言中最小的语义单位，如语言中的关键字、标识符、运算符和界限符。词法分析的依据是词的构造。
  - 单词的构造规则在高级语言中有明确的规定，比如哪些为保留字、变量如何定义、常量如何构造、分界符有哪些等。

- 描述工具：正规式和有限自动机

- FOR I := 1 TO 100 DO
  保留字 标识符 赋值号 整常数 保留字 整常数 保留字

1. 词法分析

例如，用C 语言编写的程序段如下：

```
main( )
{
    float x=2,y=3,s;
        s=x+y*5;
}
```

识别出的单词序列如表所示

类型名	单词	类型名	单词
保留字	main	左括号	(
右括号	)	花括号	{
保留字	float	标识符	x
等号	=	常量	2
逗号	,	标识符	y
等号	=	常量	3
逗号	,	标识符	s
分号	;	标识符	s
等号	=	标识符	x
运算符	+	标识符	y
运算符	*	常量	5
分号	;	花括号	}

2. 语法分析

- 任务:
  - 组词成句——在词法分析的基础上，根据语言的语法规则或文法，把单词符号组成各类的语法单位，如：短语、语句、过程、程序。
  - 通过语法分解，确定整个输入串是否构成语法上正确的句子、程序等。
- 依循的原则：语法规则
- 描述工具：上下文无关文法
- Z := X + 0.618 * Y
  - 算术表达式
  - 赋值语句

2. 语法分析

```
int fun()                    ----函数定义语句
{
    int i,sum;               ----变量定义语句
     sum=0;                   ----赋值语句
   for(i=0;i<=100;i++) ---- for语句
      sum+=i;                 ----赋值语句
   return sum;               ----返回语句
}
```

2. 语法分析

- 语法分析方法：推导（derive)和归约(reduce)
  - 推导：从文法的开始符号开始，按照语法规则，每次选择某规则右部的一个候选式取代左部，直至识别了语句或者找到错误为止。其过程可用语法树描述.
  - 归约：按照语法规则，每次选择某规则左部取代右部的一个候选式
  - 注：语法 = 词法规则 + 语法规则

2. 语法分析

- 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).

position := initial + rate * 60 ;

规则

<赋值语句>::=<标识符>“:=”<表达式>

<表达式>::=<表达式>“+”<表达式>

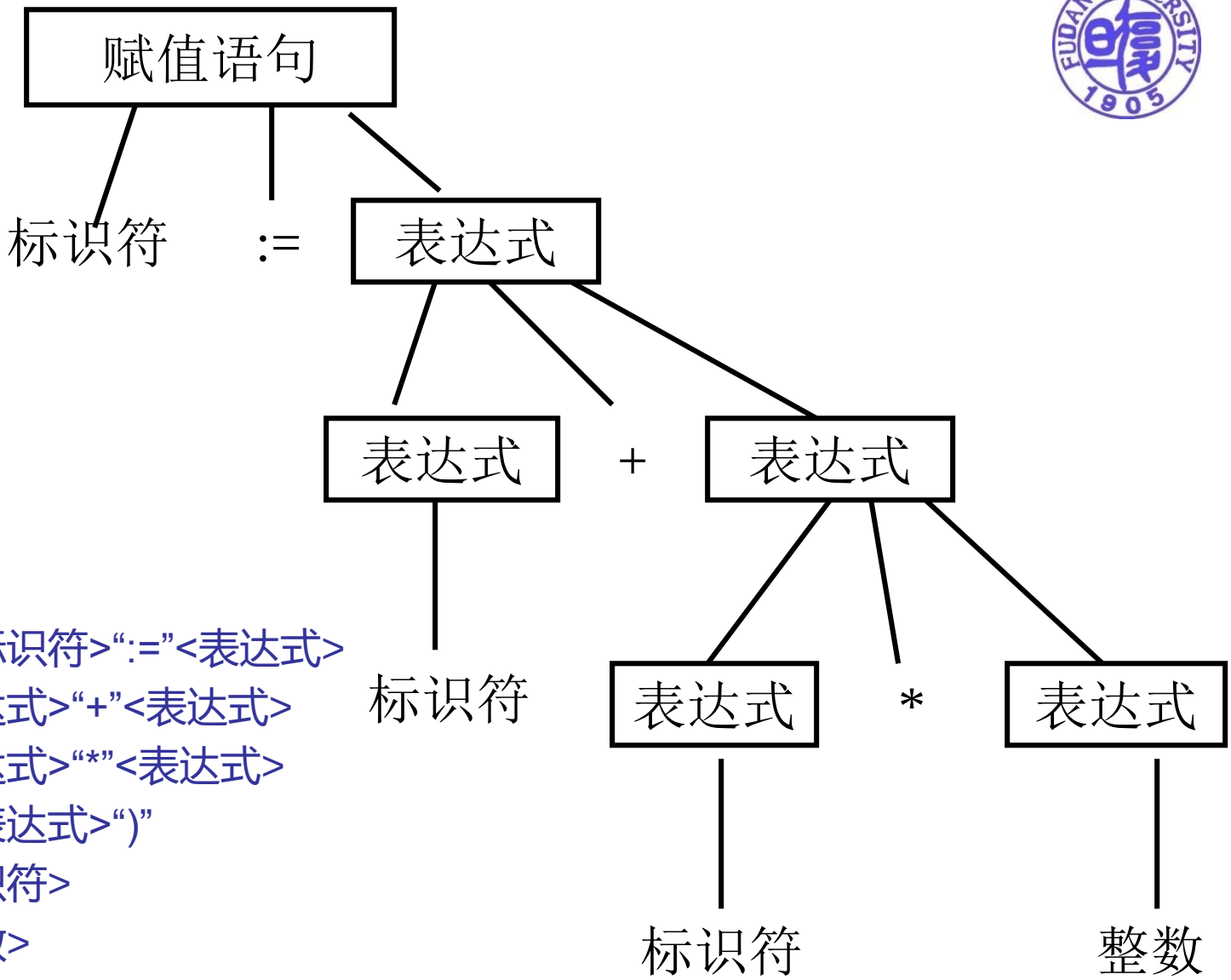
<表达式>::=<表达式>“*”<表达式>

<表达式>::=“(”<表达式>“)”

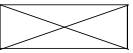
<表达式>::=<标识符>

<表达式>::=<整数>

<表达式>::=<实数>



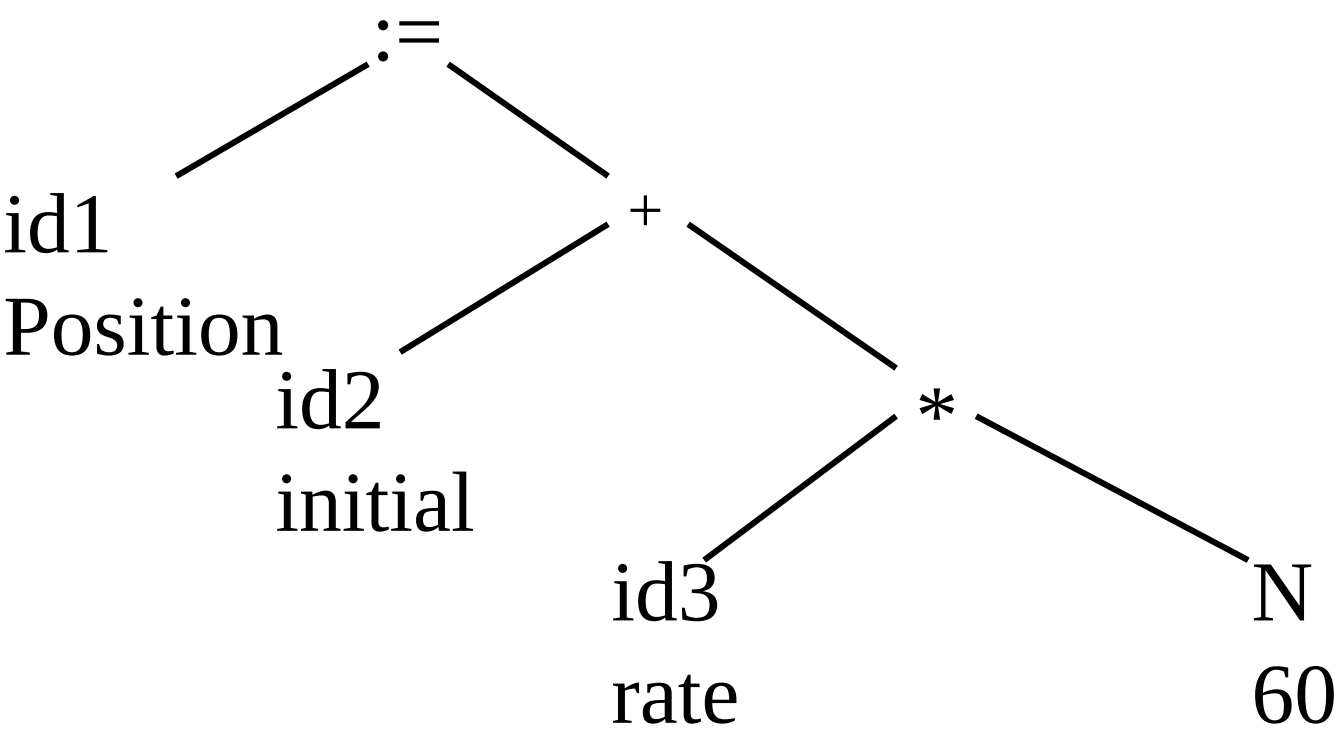
<赋值语句>::=<标识符>“:=”<表达式>
<表达式>::=<表达式>“+”<表达式>
<表达式>::=<表达式>“*”<表达式>
<表达式>::=“(”<表达式>“)”
<表达式>::=<标识符>
<表达式>::=<整数>
<表达式>::=<实数>



2. 语法分析

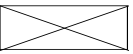
- id1:=id2+id3*N

position := initial + rate * 60 ;



3. 语义分析和中间代码产生

- 任务:对各类不同语法范畴按语言的语义进行初步翻译。
  - 一是对每种语法范畴进行语义检查，如变量是否定义、类型是否正确等;
  - 二是在语义检查正确的情况下，进行中间代码的翻译。
    - 中间代码是介于高级语言语句和低级语言语句之间的一种独立于具体硬件的记号系统，它即有一定程度的抽象，又和低级语言十分接近，因此，转换成目标代码很容易。
- 依循的原则：语义规则
- 中间代码:三元式，四元式，树形结构等



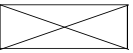
3. 语义分析和中间代码产生

- 将 x=a + b * 50 翻译成四元式为:

序号	算符	左操作数	右操作数	结果
(1)	将整常数50转换为实常数			T_1
(2)	*	b	T_1	T_2
(3)	+	a	T_2	T_3
(4)	=	T_3		x

4. 优化

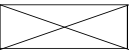
- 任务：对于前阶段产生的中间代码进行加工变换，以期在最后阶段产生更高效的目标代码。
- 依循的原则：程序的等价变换规则
- 优化的主要手段包括：
  - 公共子表达式的提取，如：x=(a+b)*c+(a+b)*d。
  - 删除无用赋值；
  - 合并已知量；
  - 循环优化：改造成循环内的操作都是必须的。



4. 优化

- 例:

```
FOR K:=1 TO 100 DO
  BEGIN
    X:=I+1;
    M := I + 10 * K;
    N := J + 10 * K;
    Print M, N;
  END
```



中间代码（一）

- 例:

```
FOR K:=1 TO 100 DO
  BEGIN
    X:=I+1;
    M := I + 10 * K;
    N := J + 10 * K;
    Print M, N;
  END
```

序号	OPR	OPN1	OPN2	RESULT	注释
(1)	:=	1		K	K:=1
(2)	j<	100	K	(10)	if (100<K) goto (10)
(3)	+	I	1	X	X:=I+1
(4)	*	10	K	T1	T1:=10*K
(5)	+	I	T1	M	M:=I+T1
(6)	*	10	K	T2	T2:=10*K
(7)	+	J	T2	N	N:=J+T2
(8)	+	K	1	K	K:=K+1
(9)	j			(2)	goto (2)
(10)					

400次加法
200次乘法

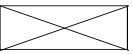
转换后的等价代码（二）

序号	OPR	OPN1	OPN2	RESULT	注释
(1)	+	I	1	X	X:=I+1
(2)	:=	I		M	M:=I
(3)	:=	J		N	N:=J
(4)	:=	1		K	K:=1
(5)	j<	100	K	(10)	if (100<K) goto (10)
(6)	+	M	10	M	M:=M+10
(7)	+	N	10	N	N:=N+10
(8)	+	K	1	K	K:=K+1
(9)	j			(5)	goto (5)
(10)					

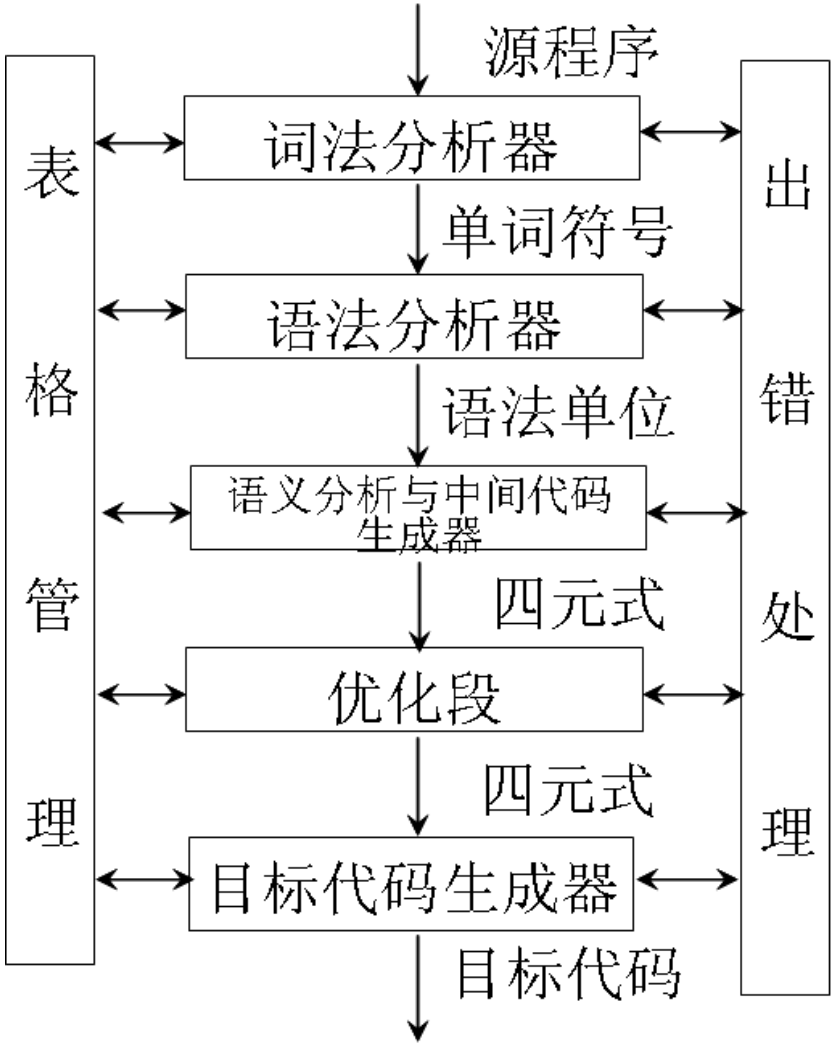
301次加法

5. 目标代码产生

- 任务：把前一阶段的中间代码变换成特定机器上的机器语言或汇编语言程序，实现机器的最终翻译。
- 该阶段的工作因为目标语言的关系而十分依赖机器的硬件系统，
  - 如何充分利用机器现存的寄存器，
  - 合理的选择指令，生成尽可能短的目标代码，
  - 这与机器的硬件有关。
- 目标代码三种形式：
  - 绝对指令代码：可直接运行
  - 汇编指令代码：需要进行汇编
  - 可重新定位指令代码：需要连接装配。先将各目标模块连接起来，确定变量、常数在主存中的位置，装入主存后才能成为可以运行的绝对指令代码。



编译程序总框


源程序
词法分析器
单词符号
语法分析器
语法单位
语义分析与中间代码生成器
四元式
优化段
四元式
目标代码生成器
目标代码
表格管理
出错处理

2. 表格和表格管理

- 表格作用：
  - 用于记录源程序的各种信息以及编译过程中的各种状况，以便后续阶段使用。
- 与编译前三阶段有关的表格有：
  - 符号表、常数表、标号表、分程序入口表、中间代码表等。
  - 注：在编译过程中，随着源程序的不断被改造，编译的各阶段常常需要不同的表格，编译过程的绝大多数时间是花在查表、造表和更新表格的事务上。在大多数的编译程序中，表格专门由表格管理程序来处理。

- 格式：

名字	信息

例：PASCAL程序段：

```
PROCEDURE INCWAP(M，N:INTEGER);
LABEL  START;
VAR
  K:INTEGER;
BEGIN
START:
     K:=M+1;
      M:=N+4;
      N:=K;
END.
```

表 0.1 符号名表 SNT

NAME	INFORMATION
M	形式参数，整型
N	形式参数，整型
K	整型，变量

- 符号表：用来登记源程序中的常量名、变量名、数组名、过程名等的性质、定义和引用状况。

例：PASCAL程序段：

```
PROCEDURE INCWAP(M，N:INTEGER);
LABEL  START;
VAR
  K:INTEGER;
BEGIN
START:
    K:=M+1;
    M:=N+4;
    N:=K;
END.
```

表 0.2 常数表 CT

	值(VALUE)
(1)	1
(2)	4

- 常数表：
  - 登记各类常量(直接量)值
  - 每一种类型的常数对应一个表。（数值型，逻辑型，字符型。。。。。）

例：PASCAL程序段：

```
PROCEDURE INCWAP(M，N:INTEGER);
LABEL  START;
VAR
  K:INTEGER;
BEGIN
START:
     K:=M+1;
      M:=N+4;
      N:=K;
END.
```

表 0.5　四元式表 QT

	OPR	OPN1	OPN2	RESULT
(1)	link			
(2)	par	INCWAP	1	M
(3)	par	INCWAP	2	N
(4)	+	M	1	K
(5)	+	N	4	M
(6)	:=	K		N
(7)	return			

- 中间代码表：记录四元式序列的表

例：PASCAL程序段：

```
PROCEDURE INCWAP(M，N:INTEGER);
LABEL  START;
VAR
  K:INTEGER;
BEGIN
START:
     K:=M+1;
     M:=N+4;
     N:=K;
END.
```

表 0.3 入口名表 ENT

	NAME	INFORMATION
(1)	INCWAP	二目子程序，入口四元式:1

- 入口名表：登记过程的层号，分程序符号表的入口(指分程序结构的语言)等

例：PASCAL程序段：

```
PROCEDURE INCWAP(M，N:INTEGER);
LABEL  START;
VAR
  K:INTEGER;
BEGIN
START:
     K:=M+1;
      M:=N+4;
      N:=K;
END.
```

表 0.4　标号表 LT

	NAME	INFORMATION
(1)	START	四元式:(4)

- 标号表：登记标号的定义与引用。
  代表执行的哪一个四元式。

3. 出错处理

- 任务：

  如果源程序有错误，编译程序应设法发现错误，并报告给用户。

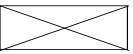
- 完成：由专门的出错处理程序来完成
- 错误类型：
  - 语法错误:在词法分析和语法分析阶段检测出来。
  - 语义错误：一般在语义分析阶段检测。比如：除0等等，常量可以检测出，变量不行。
- 不能处理逻辑错误：
  - 比如：变量的值在运行时发生改变，引起的逻辑错误。

```
for(i=1; i<=k; i++){
.....
}
```

（可能陷入死循环）

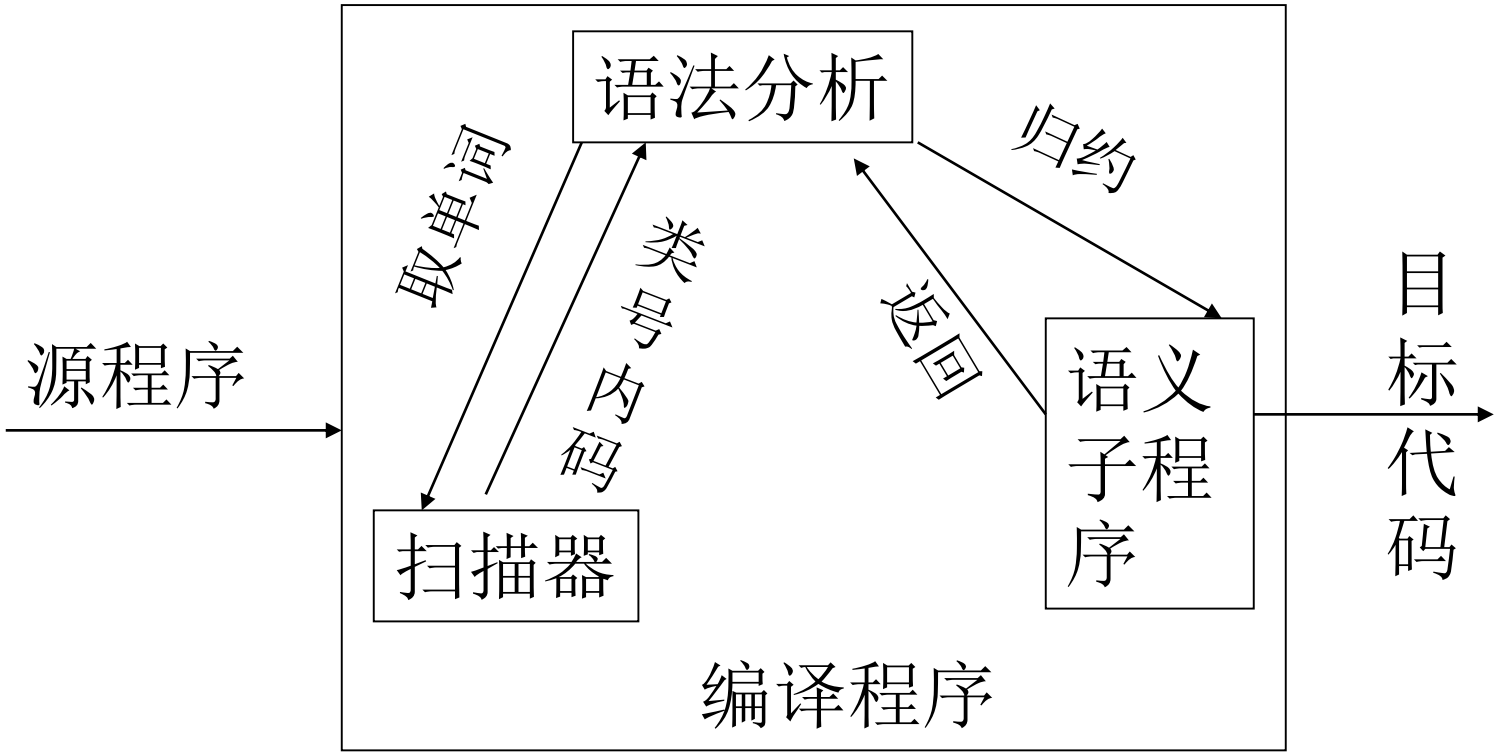
4. 遍(pass)

- 遍：指对源程序或源程序的中间结果从头到尾扫描一次，并做有关的加工处理，生成新的中间结果或目标代码的过程。
  - 注：遍与阶段的含义毫无关系。
- 阶段与遍是不同的概念。一遍可以由若干段组成，一个阶段也可以分若干遍来完成。
- 多遍扫描的好处
  - 使编译的逻辑结构清晰（各遍有明确的功能），提高目标代码质量；节省内存空间（减少表格）。
- 多遍扫描的缺点
  - 编译时间较长。
  - 注：在内存许可情况下，还是遍数尽可能少些为好。



4. 遍(pass)

- 一遍扫描（以语法分析为中心）

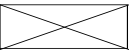


5. 编译前端与后端

- 编译前端：与源语言有关，如词法分析，语法分析，语义分析与中间代码产生，与机器无关的优化
- 编译后端：与目标机有关，包括与目标机有关的优化，目标代码产生
  - 优点：程序逻辑结构清晰；优化更充分，有利于移植。
  - 不足：编译程序运行的效率低

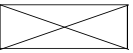
编译程序与程序设计环境

- 程序设计环境
  - 编辑程序
  - 编译程序
  - 连接程序
  - 调试工具
- 集成化的程序设计环境



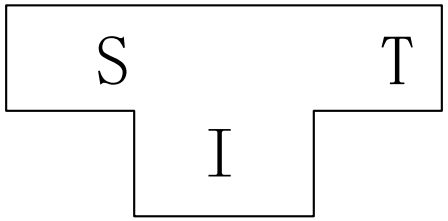
编译程序生成

- 以汇编语言和机器语言为工具
  - 优点： 可以针对具体的机器，充分发挥计算机的系统功能。生成的程序效率高。
  - 缺点： 程序难读、难写、易出错、难维护、生产的效率低。



编译程序生成

- 高级语言书写



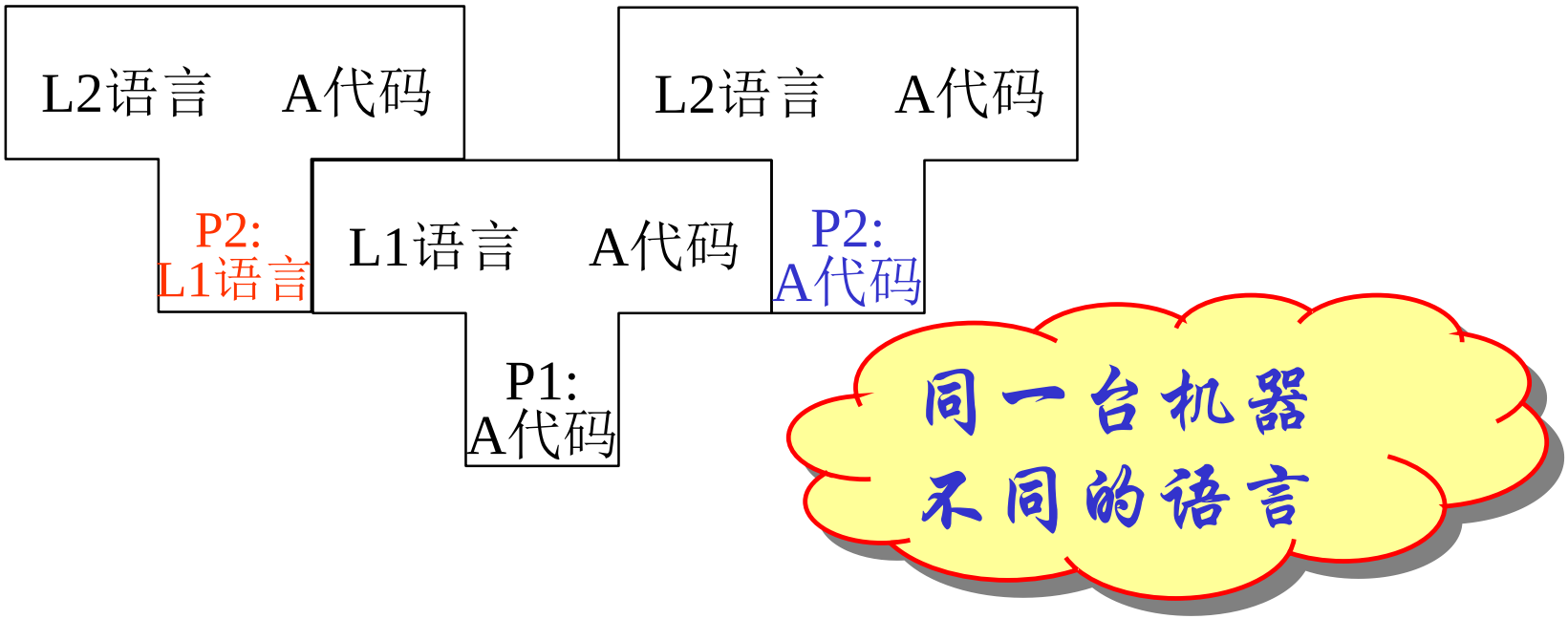
S 源程序　　T 目标程序　　I 实现语言

- 优点：　程序易读、易理解、容易维护、生产的效率高。
- 缺点：　难以充分发挥计算机的系统功能，生成的程序效率低。

编译程序生成

•高级语言书写

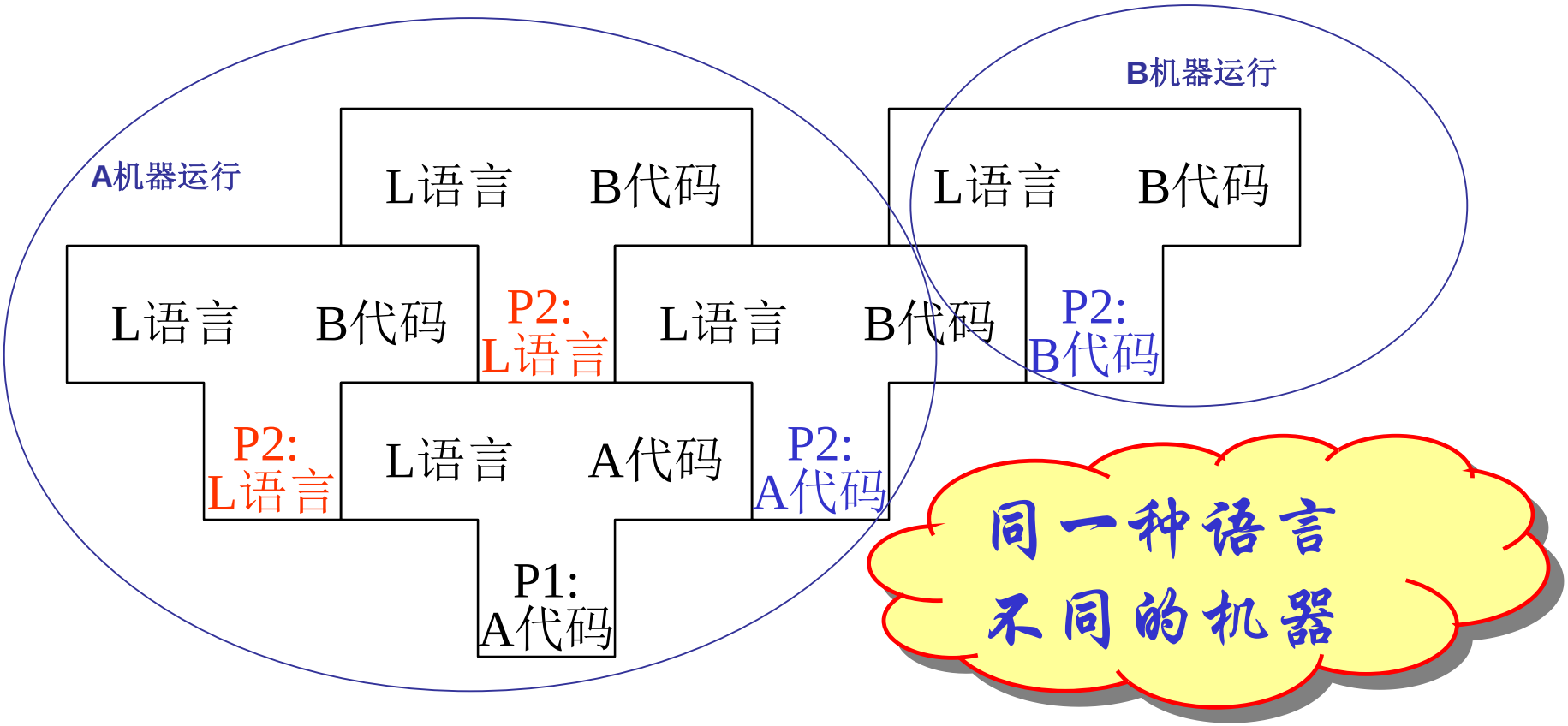
利用已有编译程序的某种语言L1来实现另一语言L2的编译程序。



编译程序生成

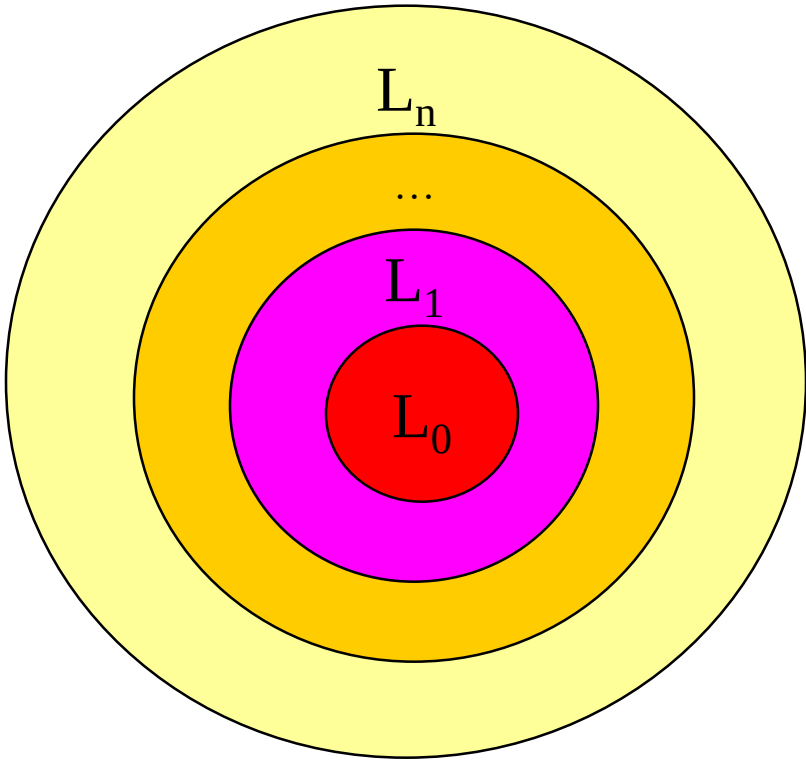
- 移植方法

把一种机器A上的编译程序移植到另一种机器B上。



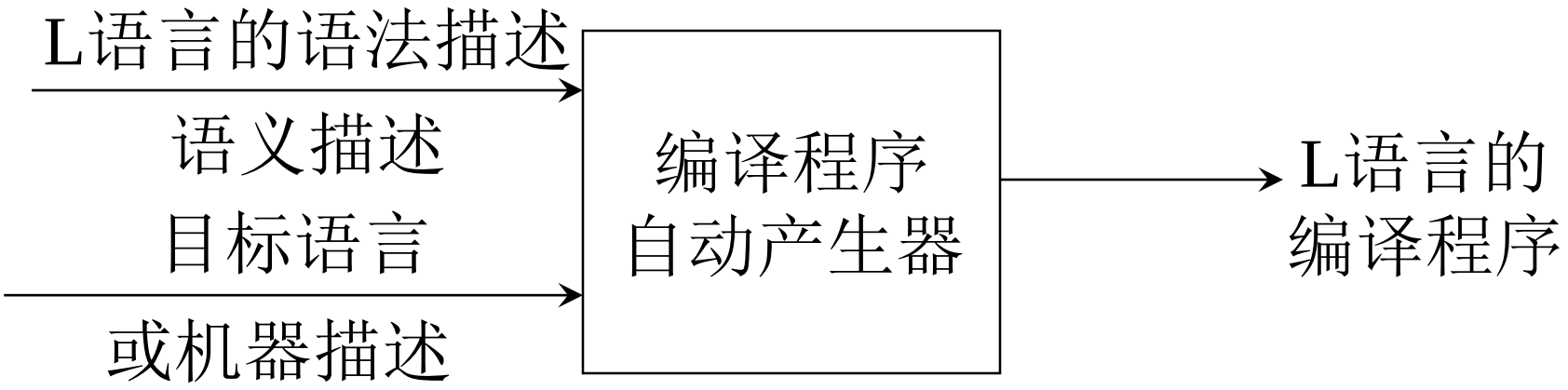
五. 编译程序生成

- 自展技术
  - 首先确定一个非常简单的核心语言L0，然后用机器语言或汇编语言书写它的编译程序T0；
  - 再把语言L0扩充到L1,此时有L0属于L1，并用L0编写出L1的编译程序T1，
  - 然后再把语言L1扩充为L2，此时，L1属于L2，并用L1编写L2的编译程序T2,……这样不断的扩展下去，直到完成所要求的编译程序为止。



五. 编译程序生成

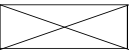
- 编译程序自动产生，这些工具称为：
  - 编译程序-编译程序
  - 编译程序书写系统
  - 编译程序产生器



LEX 词法分析程序产生器
YACC 语法分析程序产生器

形式语言和编译实现技术

- 形式语言是一种不考虑含义的符号语言，形式语言的理论研究的是组成这种符号语言的符号串集合、它们的表示法、结构和特性。
- 按Chomsky文法分类法，文法可以分成四大类，即短语结构文法、上下文有关文法、上下文无关文法和正则文法。
- 通常的程序设计语言，其符号的定义和正则文法相关联，语法定义和上下文无关文法相关联，而语义一般需要上下文有关文法来定义。

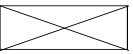


形式语言和编译实现技术

- 形式语言理论采用数学那样的符号形式表示，数学那样的严格推理。采用形式语言方式讨论程序设计语言及其编译程序的构造，可以使我们不仅了解一些编译实现技术如何应用，而且还可以理解如此做的原因。

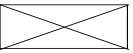
- 对编译程序的设计中所采用的技术，不但要知其然，而且还要知道所以然。从理论高度上去掌握编译技术。可以用下列公式来表示一种关系，即：

  编译原理=形式语言理论+编译技术



课程的主要内容

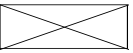
- 引论
- 基础知识：文法
- 词法分析
  - 理论模型——正规文法与有限自动机
  - 实现——词法分析程序
- 语法分析
  - 理论模型：自上而下分析和自下而上分析
  - 实现——YACC
- 中间代码生成
- 运行时数据区的管理
- 中间代码优化
- 目标代码生成



关于学习编译原理

- 构造编译程序的前提：
  - 掌握源语言，包括其结构（语法）和含义（语义）；
  - 掌握目标语言，若是机器语言，必须清楚硬件的系统结构和操作系统的功能；
  - 掌握编译方法（本课程内容）。

- 课程特点：
  - 理解性
  - 实践性



本章小结

- 掌握编译程序和高级程序设计语言之间的关系。
- 掌握分为哪几个阶段，各个阶段的主要功能和采用的主要方法：
  - 词法分析
  - 语法分析
  - 中间代码产生
  - 优化
  - 目标代码产生