第一章 引论

内容线索

- 什么叫编译程序
- 编译过程概述
- 编译程序的结构
- 编译程序的生成
- 总结

程序语言技术的发展

表示在IBM PC 上使用的Intel 8x86处理器将数字2移至地址0 0 0 0（16进制）的指令

表示机器实际操作的数字代码

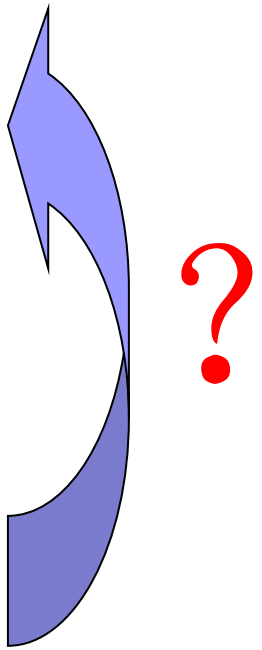
机器语言　C7 06 0000 0002

以符号形式给出指令和存储地址的

汇编语言　MOV X, 2

类似于数学定义或自然语言的简洁形式编写程序的操作

高级语言　X=2



程序的执行方式

- 高级语言程序通常采用**解释方式**和**编译方式**两种方式执行

解释方式

逐个语句地分析和执行

如Basic，Prolog

优点：易于查错

缺点：效率低，运行速度慢

编译方式

对整个程序进行分析，翻译成等价机器语言程序后执行

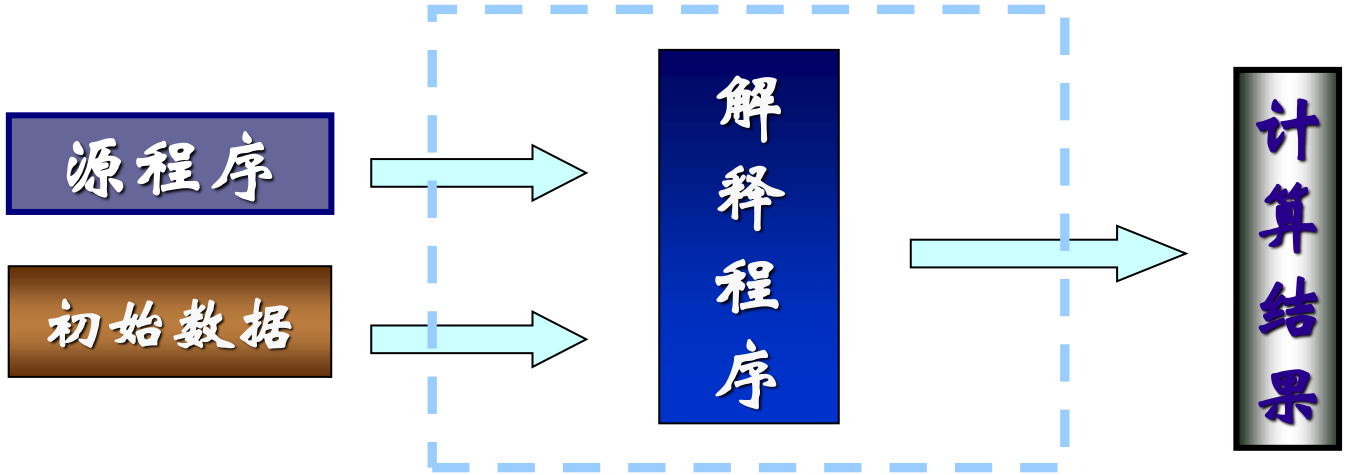
如Pascal，Fortran，C

优点：只需分析和翻译一次，

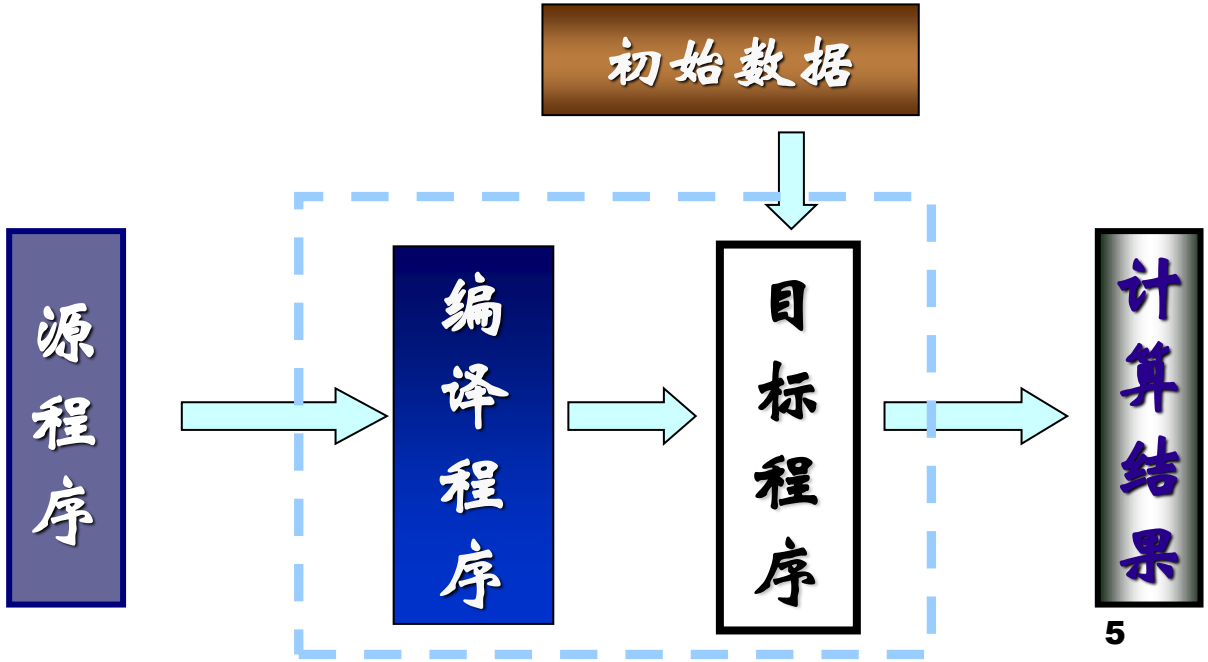
缺点：在运行中发现的错误必须查找整个程序确定

解释程序和编译程序

解释程序：边解释边执行源语言程序，不产生目标语言程序。



编译程序：能够把某种语言的程序转换成另一种语言的程序，而后者与前者在逻辑上是等价的。



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

- 简单的说，编译就是全文翻译，全部翻译完才执行。
  解释就相当于同声翻译，边翻译边执行。

编译程序的分类

- **依据编译程序的不同用途和侧重，可分类为:**
  - **诊断型编译程序(Diagnostic Compiler)**

    帮助程序员开发和调试，发现程序中的错误
  - **优化型编译程序(Optimizing Compiler)**

    侧重于提高目标代码的执行效率
  - **交叉型编译程序(Cross Compiler)**

    宿主机和目标机，一般为相同的机器，如果不同，则为交叉编译
  - **可变目标型编译程序(Retargetable Compiler)**

    只要改变与目标机器有关的部分就可以生成目标代码

编译技术的发展

- 在1954年至1957年期间，IBM的John Backus带领的一个研究小组对FORTRAN语言及其编译器的开发
  - 将算术公式翻译成机器代码。
- Noam Chomsky 开始了自然语言结构的研究。他的发现最终使得编译器结构异常简单
- 70年代，有穷自动机和形式语言的研究，促进了编译器的发展
  - Steve Johnson为Unix系统编写了Yacc （yet another compiler-compiler）。
  - Mike Lesk为Unix系统开发的 Lex
- 至今已形成一套比较成熟、系统的理论与方法及开发环境，但并行编译、嵌入式应用的交叉编译等仍在研究和发展中。

内容线索

- ✓ 什么叫编译程序
- ■ **编译过程概述**
- ■ 编译程序的结构
- ■ 编译程序的生成
- ■ 总结

编译过程概述

- 掌握编译过程的五个基本阶段，是我们学习编译原理课程的基本内容，把编译的五个基本阶段与英译中的五个步骤相比较，有利于对编译过程的理解：

英译	编译
1. 识别出句子中的一个个单词	1. 词法分析
2. 分析句子的语法结构	2. 语法分析
3. 初步翻译句子的含义	3. 语义分析中间代码生成
4. 译文修饰	4. 优化
5. 写出最后译文	5. 目标代码生成

词法分析

- 词法分析程序又称扫描程序。
  - 任务：读源程序的字符流、识别单词（也称单词符号，或简称符号），如标识符、整数、界限符等，并转换成内部形式。
  - 输入：源程序中的字符流
  - 输出：等长的内部形式，即属性字。
- 在词法分析阶段工作所依循的是语言的词法规则。
- 描述词法规则的有效工具是正规式和有限自动机。
- 方法：状态图；DFA；NFA

词法分析示例

例：一个C源程序片段：

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

词法分析后返回(如右图)：

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

语法分析

- 语法分析程序又称识别程序。
  - 任务：读入由词法分析程序识别出的符号，根据给定语法规则，识别出各个语法单位（如：短语、子句、语句、程序段、程序），并生成另一种内部表示。
  - 输入：由词法分析程序识别出并转换的符号
  - 输出：另一种内部表示，如语法分析树或其它中间表示。
- 语法规则通常用上下文无关文法描述。
- 方法：递归子程序法、LR分析法、算符优先分析法。

例：id1:=id2+id3*10

- 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>

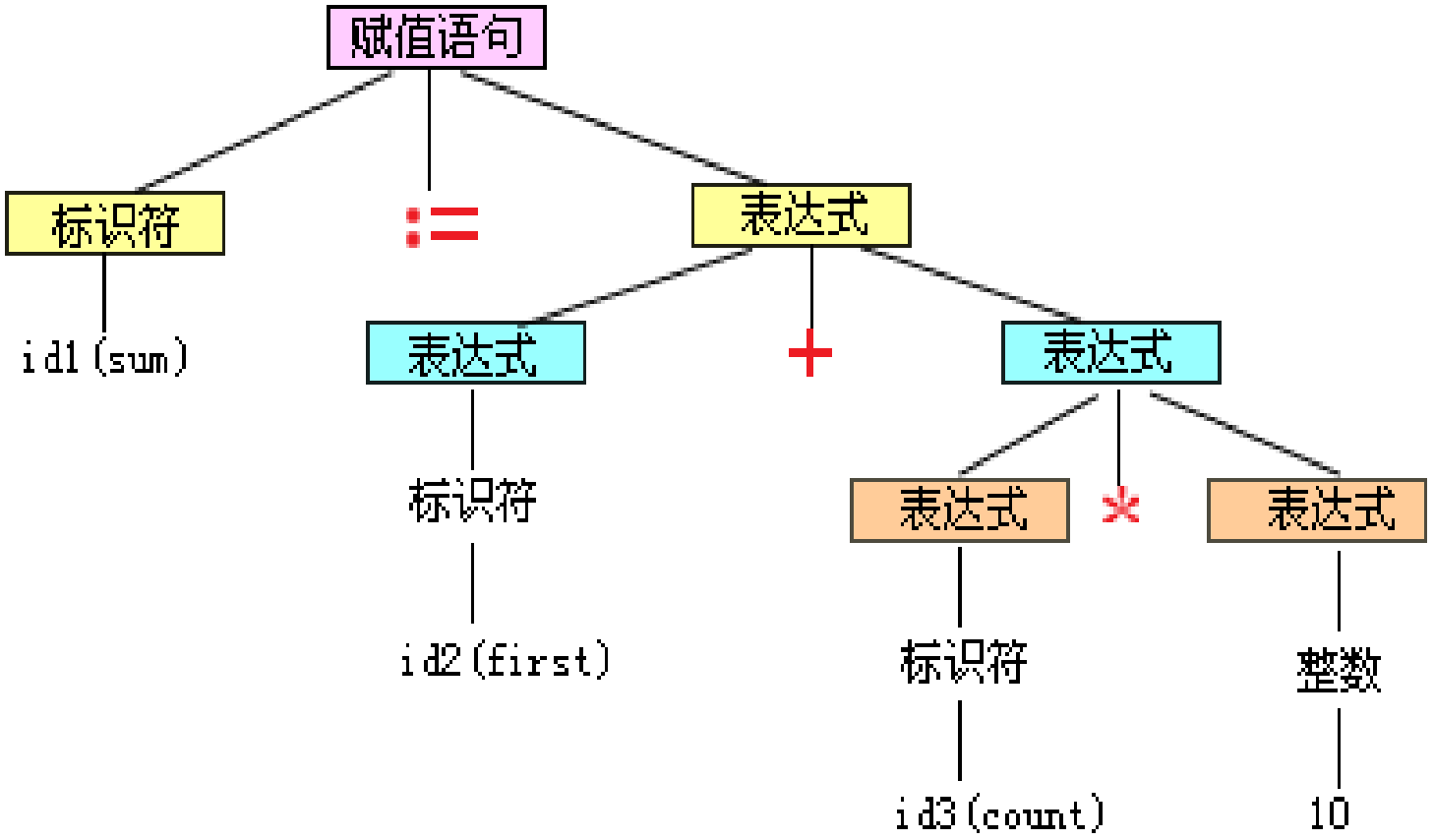
  <表达式>::=“(”<表达式>“)”

  <表达式>::=<标识符>

  <表达式>::=<整数>

  <表达式>::=<实数>

例：$id_1:=id_2+id_3*10$ 的语法树



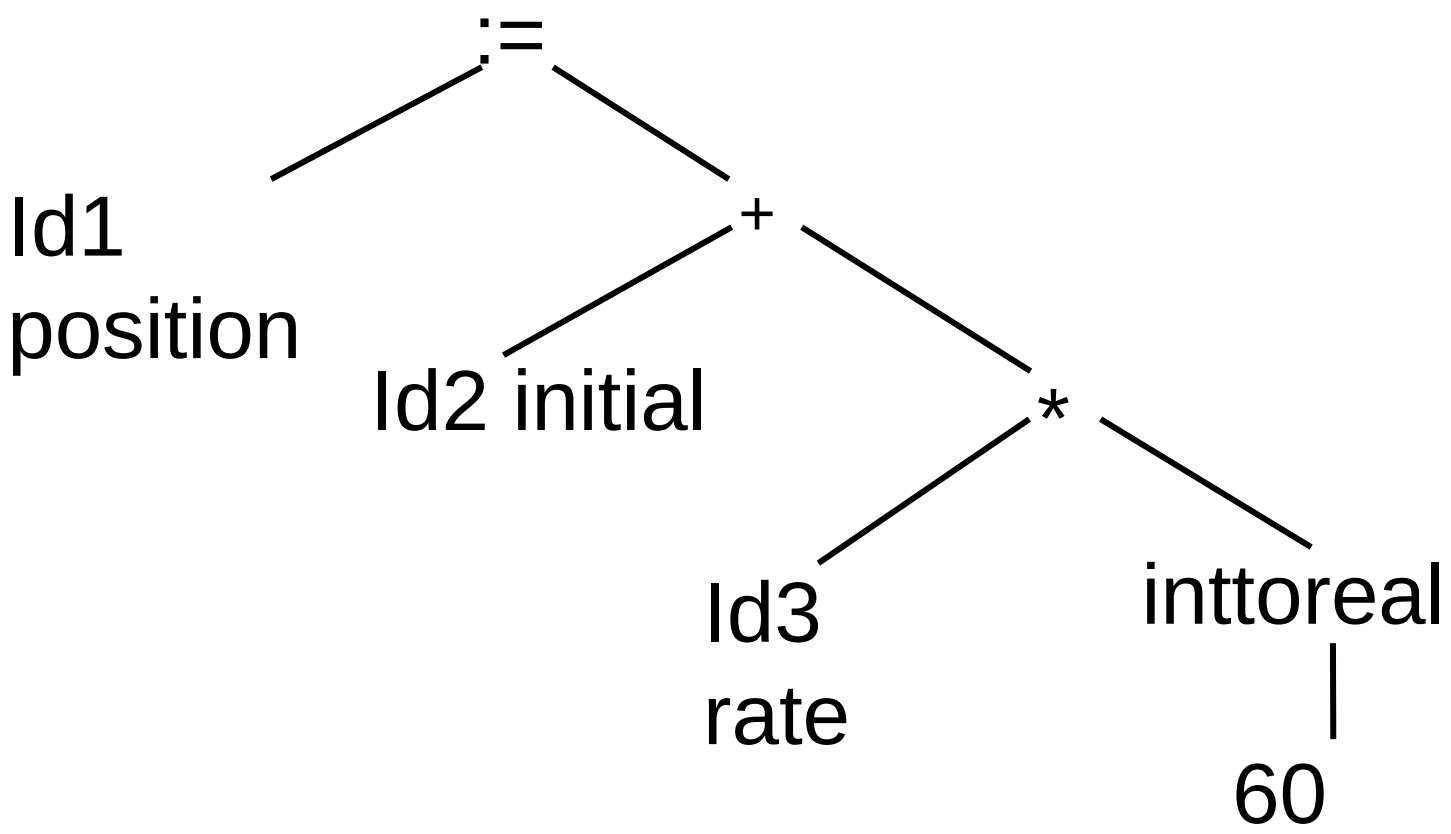
语义分析

- 对语法分析树或其他内部中间表示进行**静态语义检查**，如果正确则进行中间代码的翻译。
  - 按照语法树的层次关系和先后次序，逐个语句地进行语义处理。
  - 主要任务： 进行类型审查，审查每个算符是否符合语言规范，不符合时应报告错误。
    - 变量是否定义
    - 类型是否正确

语义分析示例

例:
```
Program p();
Var rate:real;
 procedure initial;
  …
position := initial    +    rate * 60
/* error */   /* error */    /* warning */;
```

插入语义处理结点的语法树



中间代码

- 中间代码是一种独立于具体硬件的记号系统，或者与现代计算机的指令形式有某种程度的接近，或者能比较容易地变换成机器指令。
  - 任务：将各类语法单位，如“表达式”、“语句”、“程序”等翻译为中间代码序列。
  - 输入：句子
  - 输出：中间代码序列
- 中间代码的形式：常见的有四元式、三元式和逆波兰式等
- 方法：语义子程序；DAG图，语法制导翻译

- 四元式的形式为：

  （算符，运算对象1，运算对象2，结果）；

id1　　id2　　id3

- 对于源程序sum:= first + count * 10可以生成如下所示的四元式：

(1) (inttoreal，10 ，- ，t1)

(2) (* ，id3 ，t1 ， t2)

(3) (+ ，id2 ，t2 ， t3)

(4) (:= ， t3 ，— ，id1)

t1，t2，t3是临时变量

- 其中：id1、id2、id3分别表示sum、first、count的机器内部表示，t1、t2、t3是临时生成的名字，表示中间运算结果。

例2：C语言的源程序a = b*c + b*d 的三地址序列（赋值语句形式的四元式）：

(1) t1 := b*c

(2) t2 := b*d

(3) t3 := t1 + t2

(4) a := t3

例3:源程序:

```
if (a<=b)
   a=a-c;
c=b*c;
```

翻译成 ——>

四元式:

100 (j<=,a,b,102)

101 (j, _, _, 104)

102 (-, a, c, t1)

103 (=, t1, _, a)

104 (*, b, c, t2)

105 (=, t2, _, c)

优化

- 优化的任务在于对前段产生的中间代码进行加工，把它变换成功能相同，但功效更高的优化了的中间表示代码，以期在最后阶段产生更为高效（省时间和空间）的代码
- 优化所依循的原则是程序的等价变换规则
- 其方法有：公共子表达式的提取、循环优化、删除无用代码等。

举例：

id1:= id2 + id3 * 60

(1)	**(inttoreal**	**60**	**-**	**t1**	**)**
(2)	**(***	**id3**	**t1**	**t2**	**)**
(3)	**(+**	**id2**	**t2**	**t3**	**)**
(4)	**(:=**	**t3**	**-**	**id1**	**)**

变换成 ⇒

（1）	**(***	**id3**	**60.0**	**t1**	**)**
(2）	**（ +**	**id2**	**t1**	**id1**	**)**

目标代码生成

- 这一阶段的任务：把中间代码（或经优化处理后）变换成特定机器上的低级语言代码。它有赖于硬件系统结构和机器指令含义。
- 目标代码的三种形式
  - 汇编指令代码：需要进行汇编
  - 绝对指令代码：可直接运行
  - 可重新定位指令代码：需要连接

```
(*,    id3    60.0    t1)
(+,    id2    t1      id1)
```

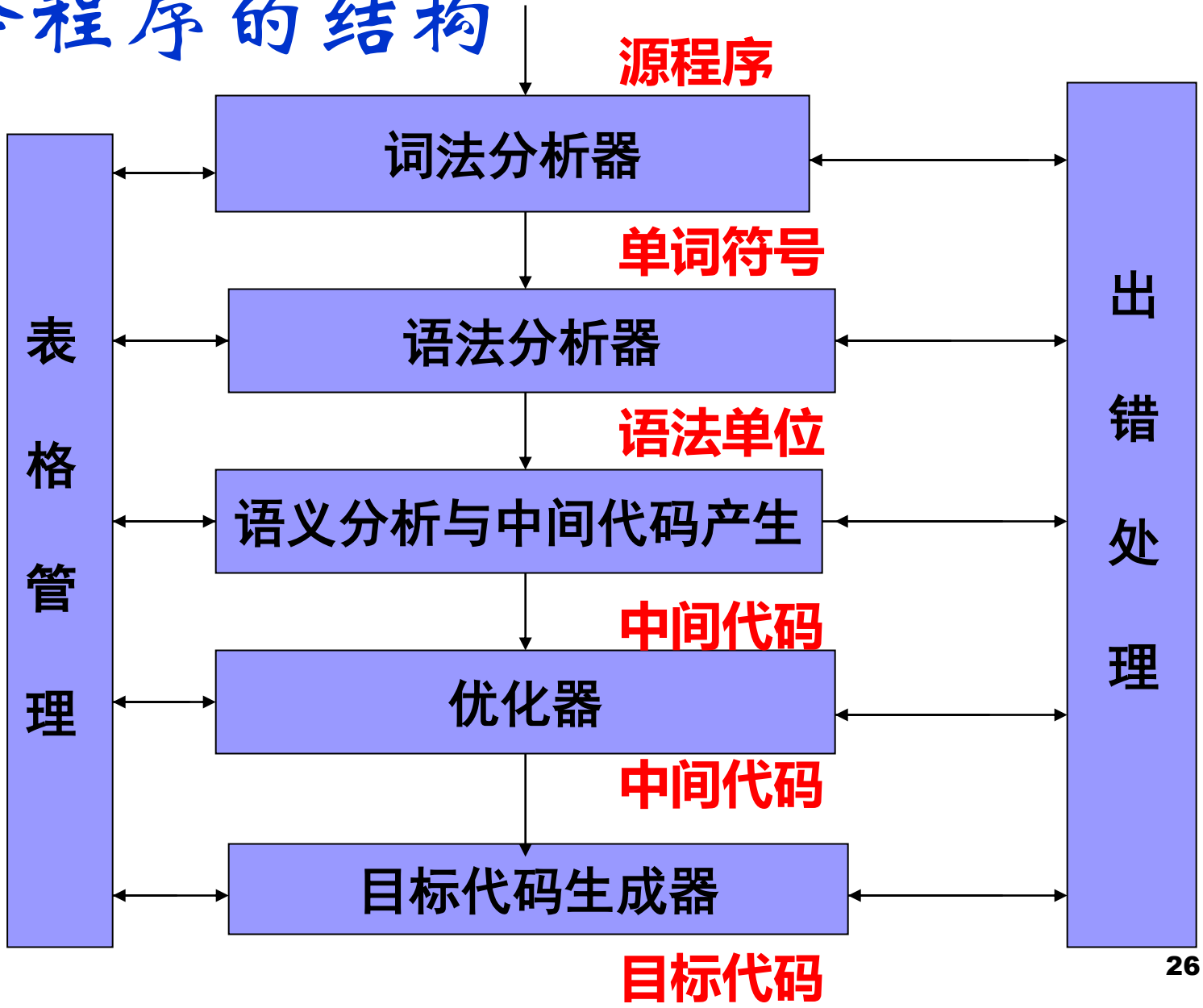


```
movf   id3   R2
mulf   #60.0   R2
movf   id2   R1
addf   R2   R1
movf   R1   id1
```

内容线索

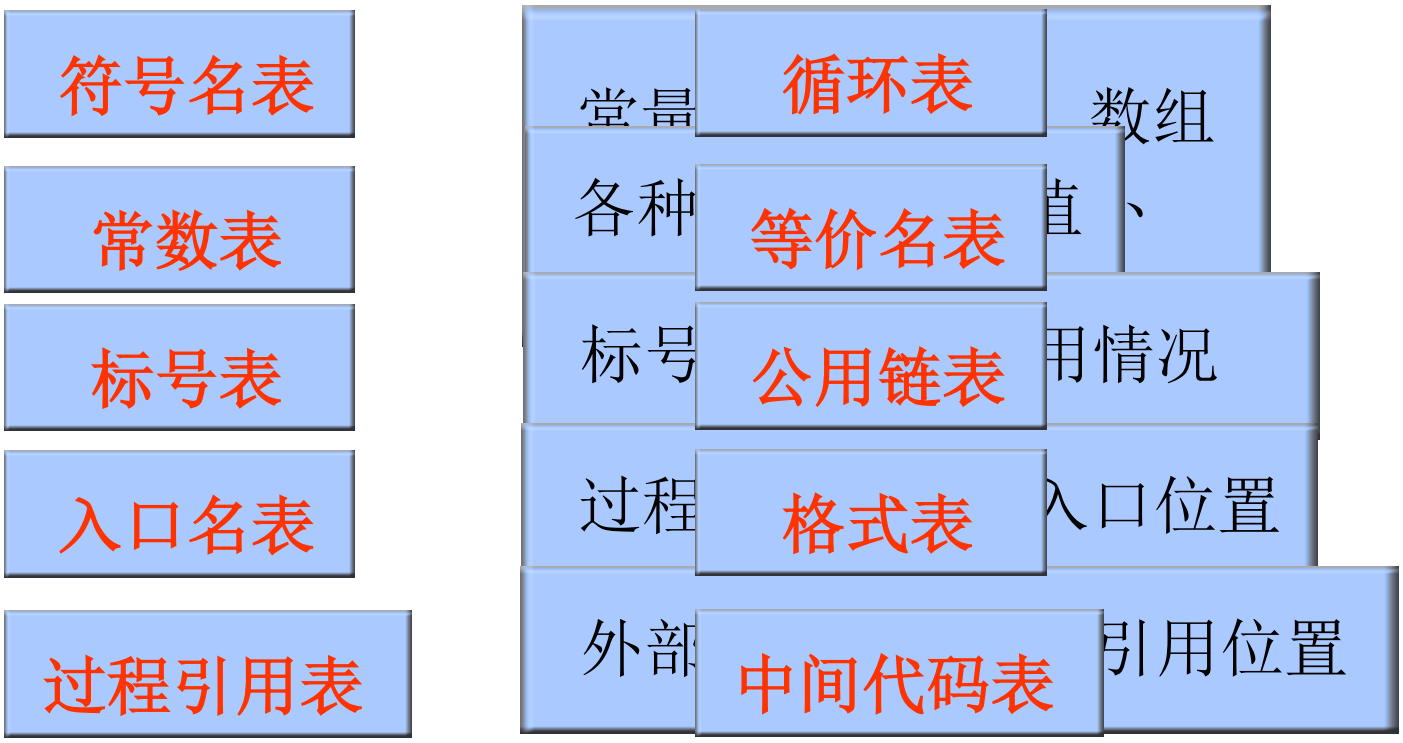
- ✓ 什么叫编译程序
- ✓ 编译过程概述
- ■ **编译程序的结构**
- ■ 编译程序的生成
- ■ 总结

编译程序的结构


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产生
中间代码
优化器
中间代码
目标代码生成器
目标代码
表格管理
出错处理

表格与表格管理

- 编译程序涉及的表格有



符号表

- 在编译程序使用的表格中最重要的是符号表。
  - 记录源程序中使用的名字(标识符)
  - 收集每个名字的各种属性信息
    - 类型、作用域、分配存储信息

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

出错处理

- 程序中的错误可分为语法错误和语义错误两类。
- 语法错误可在词法分析和语法分析阶段查出来。
- 例如，下列错误都属于语法错误：
  - （1）缺少分隔符号：DIMENSION A(10) B(10)
  - （2）保留字拼写错误：DEMENSON A(10)
  - （3）括号不配对：

    WRITE（6,10)（A（1，J），J＝1，10），I＝1，10）
  - 此外还有多余分隔符号，无循环终结语句等等。

语义错误

- 语义错误有些可在编译时查出来，有些则需在运行时才能查出来。
- 典型的语义错误：
  - 标识符没有说明就使用；
  - 标号有引用而无定义；
  - 形式参数和实在参数结合时在类型、个数、位置等方面不一致等等。
- 这些错误可在编译时查出来，而另一些错误如下标越界、运算溢出、调用某些标准函数时自变量的值不符合要求等，则需要到程序运行时才能查出来。

一个好的编译程序应该：

- **全**　　最大限度发现错误
- **准**　　准确指出错误的性质和发生地点
- **局部化**　将错误的影响限制在尽可能小的范围内

若能**自动校正错误**则更好，但其**代价非常高**

编译阶段的组合

- 编译程序可以从逻辑上分成几个阶段，对于各个阶段的划分仅仅是指其逻辑结构，而在具体实现时，经常是将几个阶段组合在一起。例如，可以将各部分组合成前端和后端。

主要与源语言有关

编译过程

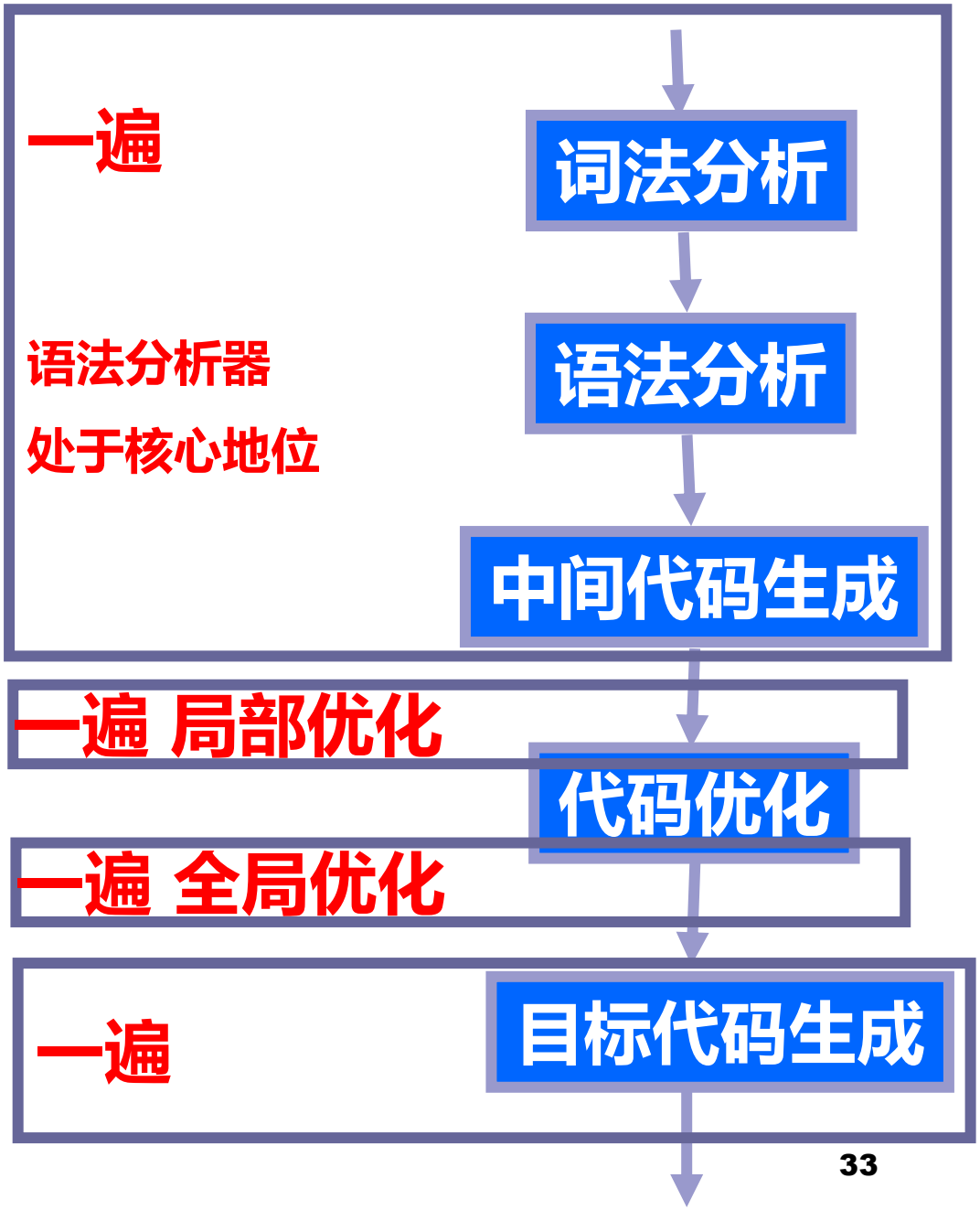
前端：词法分析、语法分析、语义分析、中间代码生成、目标代码无关的优化工作。

后端：目标代码生成、目标代码相关的优化工作。

主要与目标代码有关

遍（Pass）

- 对源程序或源程序的中间结果从头到尾扫描一次，并做相关处理，生成新的中间结果或目标程序的过程。
- “遍”是处理数据的一个完整周期，每遍工作从外存上获得前一遍的中间结果（源程序），完成它所含的有关工作之后，再把结果记录于外存。



遍的次数和效果

- 一个编译程序可由一遍、两遍或多遍完成。每一遍可完成不同的阶段或多个阶段的工作。

从时间和空间角度看

- 多遍编译 — 少占内存，多耗时间
- 一遍编译 — 多占内存，少耗时间

注意区分：“遍”和“阶段”
物理上&逻辑上

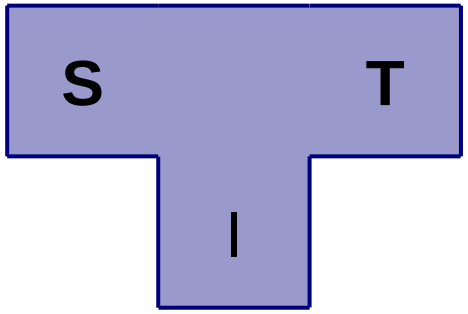
内容线索

- ✓ 什么叫编译程序
- ✓ 编译过程概述
- ✓ 编译程序的结构
- ■ **编译程序的生成**
- ■ 总结

编译程序实现语言

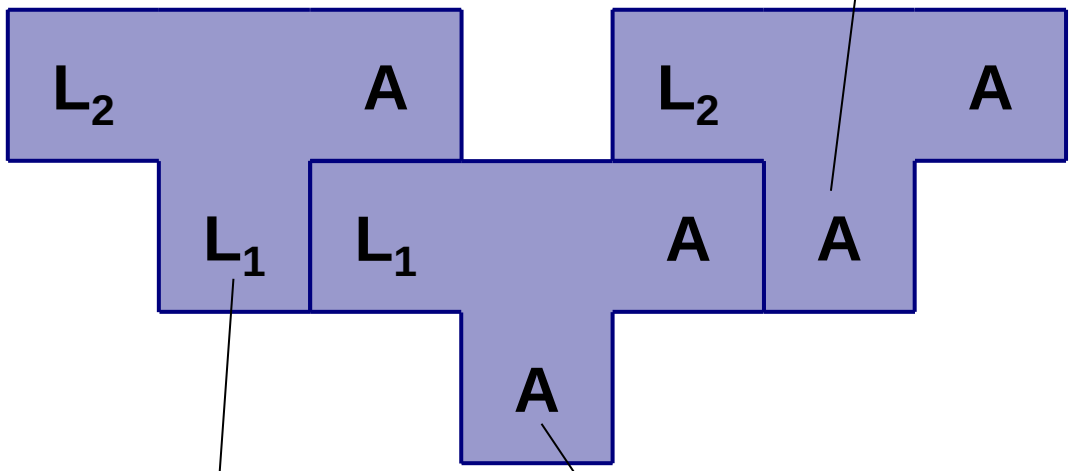
- 机器语言
- 汇编语言
  - 充分发挥各种不同硬件系统的效率
  - 满足各种不同的具体要求
- 高级语言
  - 大大节省程序设计的时间
  - 构造出来的编译程序易于阅读、维护和移植

T形图



- 其中：
  - S:源语言(程序)，Source language(program)
  - T:目标语言(程序), target/objectlanguage(program)
  - I:实现语言, implementation language

示例



A机器代码实现的L_2编译程序

用L_1语言编写另一种高级语言L_2的编译程序

A机器上已有一个用A机器码实现的某高级语言L1的编译程序

编译程序的生成技术

- 自编译
- 交叉编译
- 自展
- 移植

自编译

- 用某种高级语言书写自己的编译程序称为自编译。
- 例如,假定A机器上已有一个PASCAL语言编译程序, 则可用PASCAL语言编写一个功能更强的PASCAL语言编译程序，然后借助于原有的编译程序对新编写的PASCAL编译程序进行编译, 从而得到一个能在A机器上运行的功能更强的PASCAL编译程序。

交叉编译

- 用x机器上的编译程序产生可在 y 机器上运行的目标代码称为交叉编译。
- 例如

  1、在Windows PC上，利用ADS（ARM 开发环境）,使用armcc编译器，则可编译出针对ARM CPU的可执行代码。

  2、在Linux PC上，利用arm-linux-gcc编译器, 可编译出针对Linux ARM平台的可执行代码。
- 上述两种方法假定已有一个编译程序, 若没有, 则可采用自展或移植法。

自展

- 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出其编译程序T0；
- 到L1，L0⊂L1，并用L0编写L1的编译程序T1(自编译)；
- 然后再把语言L1扩充为L2，L1⊂ L2，并用L1编写L2的编译程序T2；

……

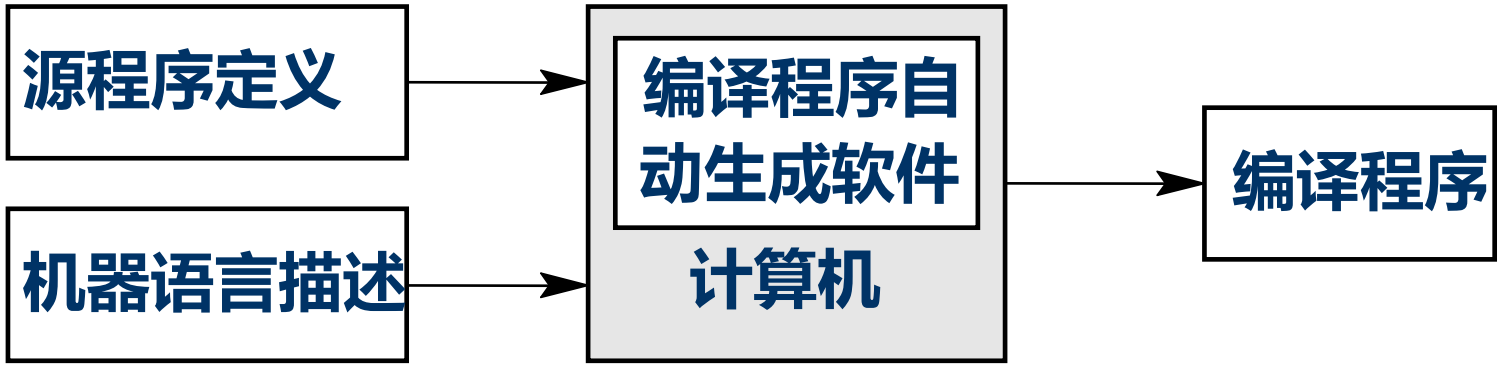
这样不断扩展，直到完成所要求的编译程序为止。

移植

- 移植是指A机器上的某种高级语言的编译程序稍加改动后能在B机器上运行。
- 一个程序若能较易地从A机移到B机上运行，则称该程序可移植。
- 例如：可移植C编译器（Portable C Compiler），一直到1994年4.4BSD发表时，它都是BSDUNIX系统上的默认C语言编译器，一直到被gcc取代为止。gcc基于yacc之上，只有少部分代码是与机器相关的，具备可移植性。

自动编译

- 自动生成编译程序的软件工具，只要把源程序的定义及机器语言的描述输入到该软件中，就能自动生成该语言的编译程序。



- 目前的编译程序自动生成系统，如LEX和YACC等。

编译技术的发展

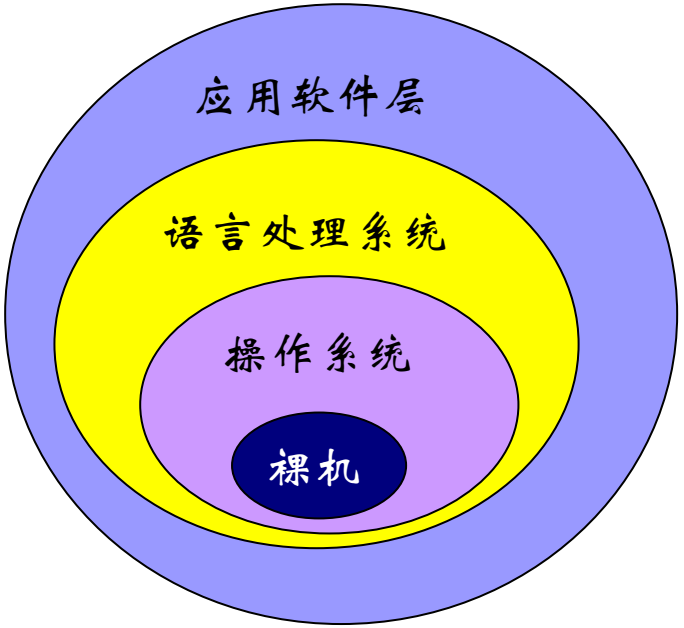
- 基本的编译器设计近20多年来没有多大的改变，现在成为计算机科学课程中的中心一环。
- 与复杂的程序设计语言的发展结合在一起。面向对象技术兴起和应用对传统的编译技术提出新的要求。
- 编译器已成为基于窗口的交互开发环境(IDE)的一部分，近年来对此进行了大量研究.
- 由多处理机的发展以及对并行处理的要求，研究方向是并行编译。
- 嵌入式系统、手机操作系统的发展：研究方向是交叉编译技术。

内容线索

- ✓ 什么叫编译程序
- ✓ 编译过程概述
- ✓ 编译程序的结构
- ✓ 编译程序的生成
- ■ 总结

编译程序在计算机系统中的地位

- 编译系统是一种系统软件
  - 软件：计算机系统中的程序及其文档。
  - 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。和具体的应用领域无关，如编译系统和操作系统等。
    - 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，如编译系统。



计算机系统

思考题

- ✓ **面向对象的高级语言的编译过程是怎样的?**