编译原理

Compilers Principles

Compiling Techniques

Programming Languages Design and Implementation

基本信息

任课教师：李素建　(lisujian@pku.edu.cn)
计算机学院计算语言学研究所
理科1号楼1443N

助教：朱大卫
zhudawei@pku.edu.cn

- 教学安排
- 课程内容

教学安排

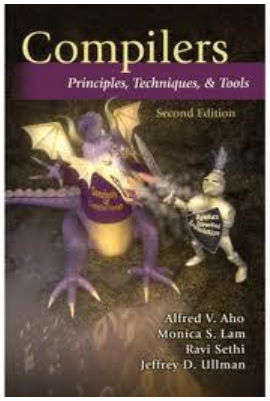
- 上课时间：周二上午3-4节
  周五上午3-4节（双周）
- 上课地点：文史110
- 上机地点：中文系机房
- 学习方式：课堂讲解+课后作业+上机实践
- 考试成绩：试卷成绩+作业成绩+上机成绩

参考教材

- **编译原理，清华大学出版社 张素琴、吕映芝 等编著，2005年**
- 编译程序设计原理 北京大学出版社，杜淑敏等编著，1986年
- 陈火旺 刘春林等 程序设计语言编译原理 国防工业出版社，2000年

参考书

- Alfred V. Aho, Monica S. Lam, Ravi Sethi, Jeffrey D. Ullman, Compilers: Principles, Techniques, &Tools (2nd Edition), Addison-Wesley, 2007.
- 中文翻译版或本科教学版：《编译原理(第2版)》 赵建华等译, 机械工业出版社, 2009.



- 【虎书】Andrew Appel, Modern Compiler Implementation in Java (2nd Edition), Cambridge University Press, 2002
- 《编译原理》孙家骕 编著，北京大学出版社，2008年。

课程内容

- 掌握编译的基本理论
  - 结构化思想, 分治等
- 掌握常用的编译技术
  - 词法分析、正则表达式、有限自动机
  - 上下文无关文法
  - 自底向上和自顶向下的语法分析
  - 基本的编译优化技术
- 了解编译过程及编译系统的构造(实践)

学习本课程的意义

- Alfred V. Aho [龙书作者]:“编写编译器的原理和技术具有十分普遍的意义，以至于在每个计算机科学家的研究生涯中，本书中的原理和技术都会反复用到。”
- “自顶向下的方法”和“自底向上的方法”系统设计方法（思想、方法）
- 应用：文本编辑器、自动排版、模式识别、程序自动验证、程序自动调试
- 为计算机分析和理解自然语言提供参考

教学要求(1/2)

- 掌握编译系统的一般构造原理
- 掌握编译系统的基本实现技术
- 熟悉一些自动构造工具

教学要求(2/2)

- 课堂：按时上课、认真听讲，积极参与
- 作业：一定要认真独立完成
- 考试：鼓励知识的灵活运用，不提倡死记硬背
- 实践：提升动手编程能力

授课内容

第一章　编译程序概述
第二章　PL/0编译程序的实现
第三章　文法和语言
第四章　词法分析
第五章　自顶向下语法分析方法
第六章　自底向上优先分析方法
第七章　LR分析方法
第八章　语法制导翻译和中间代码生成
第九章　符号表
第一〇章　代码优化
第一一章　代码生成

编译程序概述

- 什么是程序设计语言
- 什么是编译程序
- 编译过程和编译程序的结构
- 编译技术和软件工具的介绍

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。

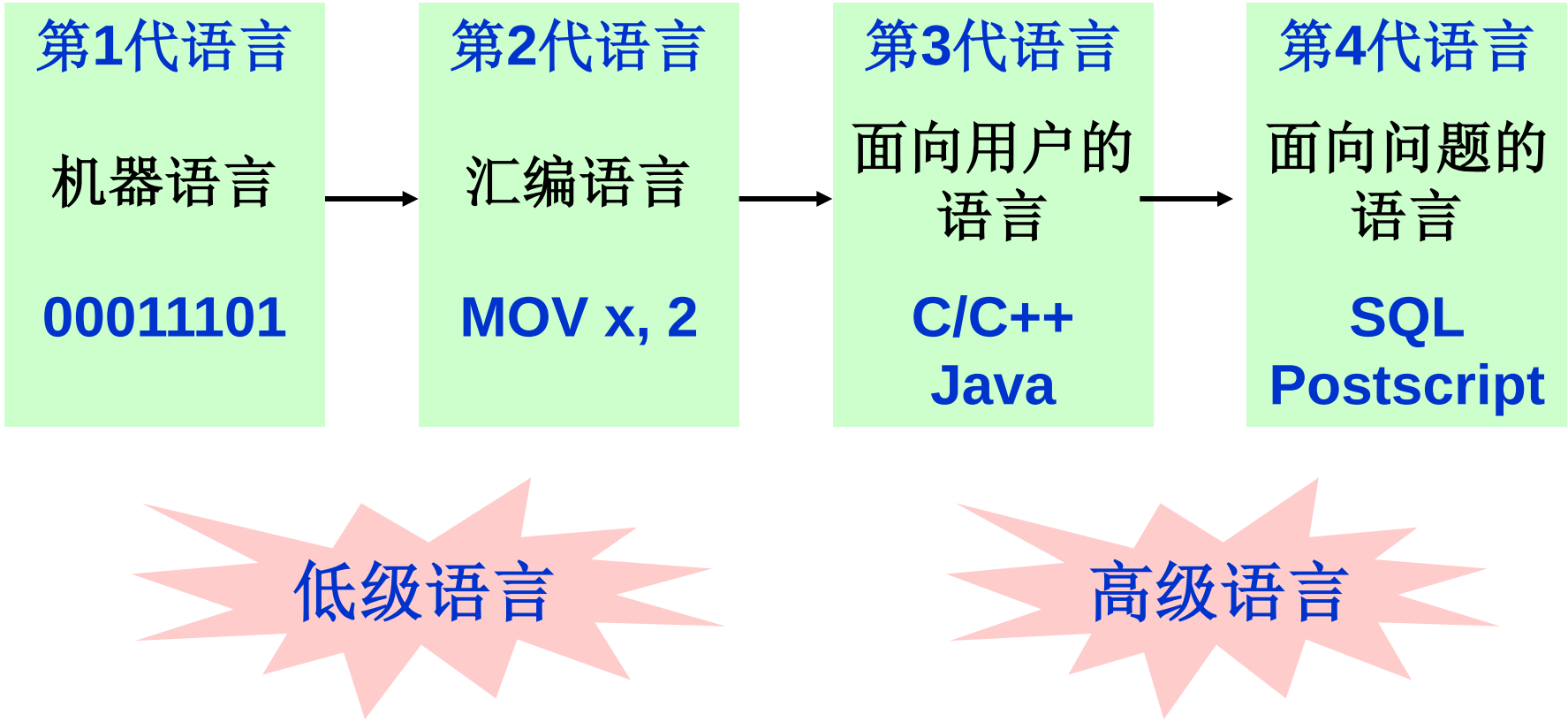
程序设计语言

- 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，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。

程序设计语言的发展


第1代语言
机器语言
00011101
第2代语言
汇编语言
MOV x, 2
第3代语言
面向用户的语言
C/C++
Java
第4代语言
面向问题的语言
SQL
Postscript
低级语言
高级语言

低级语言 vs. 高级语言

- 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）
  - Fortran、Pascal、C/C++、Java 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

用高级语言编写的程序，计算机不能立即执行，必须通过一个“翻译程序”进行加工，转化为与其等价的机器语言程序，机器才能执行。

这种翻译程序，被称为“编译程序”。

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

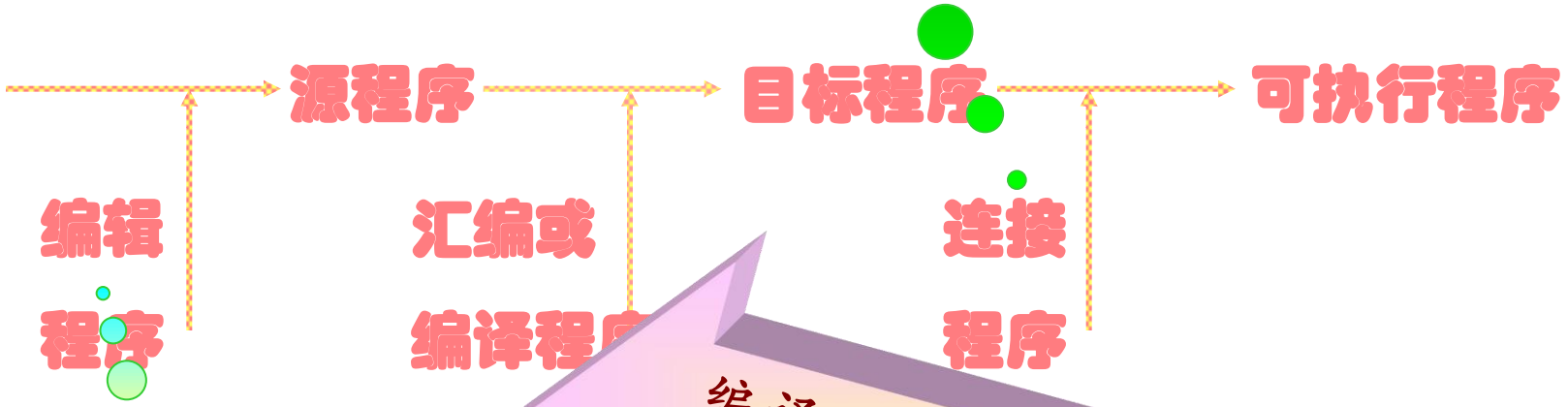
编译程序概述

- 什么是程序设计语言
- 什么是编译程序
- 编译过程和编译程序的结构
- 编译技术和软件工具的介绍

最早的编译器

- 第一个编译器（1952年）
  - A-0 System on UNIVAC I
  - 编写者：Grace Murray Hopper
  - 美国海军上校(Captain)
  - COBOL发明人
  - 创造了术语“debug”
- 第一个完整的编译器 （1957年）
  - John Backus 领导的 FORTRAN编译器 (IBM)
- 第一个高级语言书写的编译器（1962年）
  - Lisp编译器（MIT, Tim Hart and Mike Levin）

把目标程序以及所需的功能库等转换成一个可执行的装入程序。完成此功能的程序叫连接程序。



用于编写高级语言程序

编译方式是将高级语言编写的源程序整个地翻译成机器语言表示的目标程序的方式。完成此功能的程序叫编译程序。

什么是编译程序

- 编译程序的功能：把高级语言程序翻译成等价的低级语言程序。
- 源程序——》编译程序——》目标程序
- 编译程序是现代计算机系统的基本组成部分。
- 从功能上看，一个编译程序就是一个语言翻译程序，它把一种语言(称作源语言)书写的程序翻译成另一种语言(称作目标语言)的等价的程序。

如何构造第一个编译器？

- 在机器W上，如何为语言X构造一个用X编写的编译器？
- 假设机器W上已经存在某种语言Y的编译器
  - 首先用Y语言编写X语言的编译器，
  - 然后用该编译器编译用X语言编写X的编译器
- 假设机器W上不存在任何语言的编译器
  - Bootstrapping

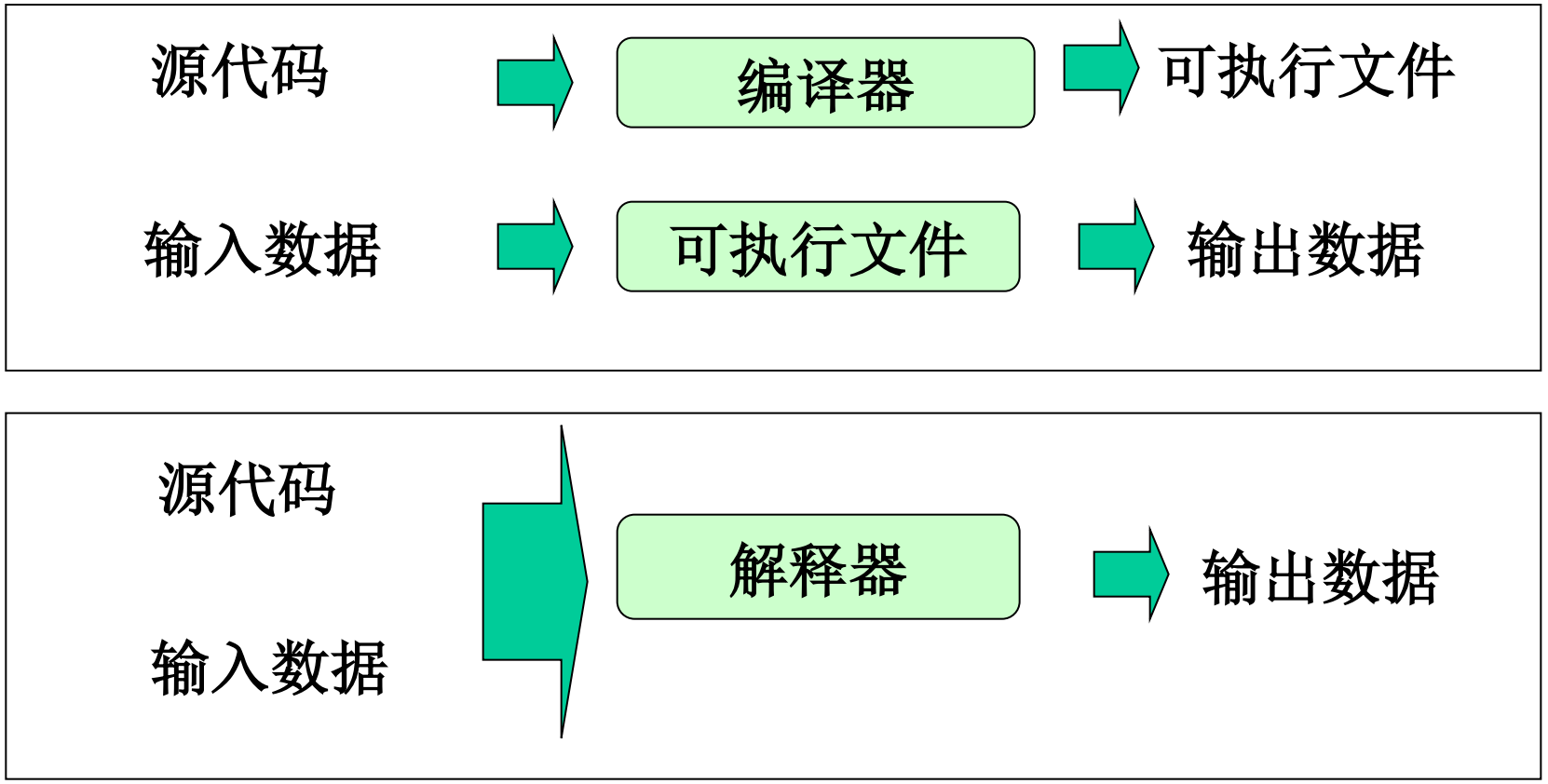
编译器 vs. 解释器 (1/5)

- 编译器（Compiler）：把（人编写的）源程序转化为（机器可读的）可执行程序，线下的过程(offline)

- 解释器（Interpreter）：在转换源程序的同时执行程序, 线上的过程(online)
  - 不生成目标代码，而是直接执行源程序所指定的运算
  - 解释器也需要对源程序进行词法，语法和语义分析，中间代码生成

编译器 vs. 解释器 (2/5)

- 理想状态



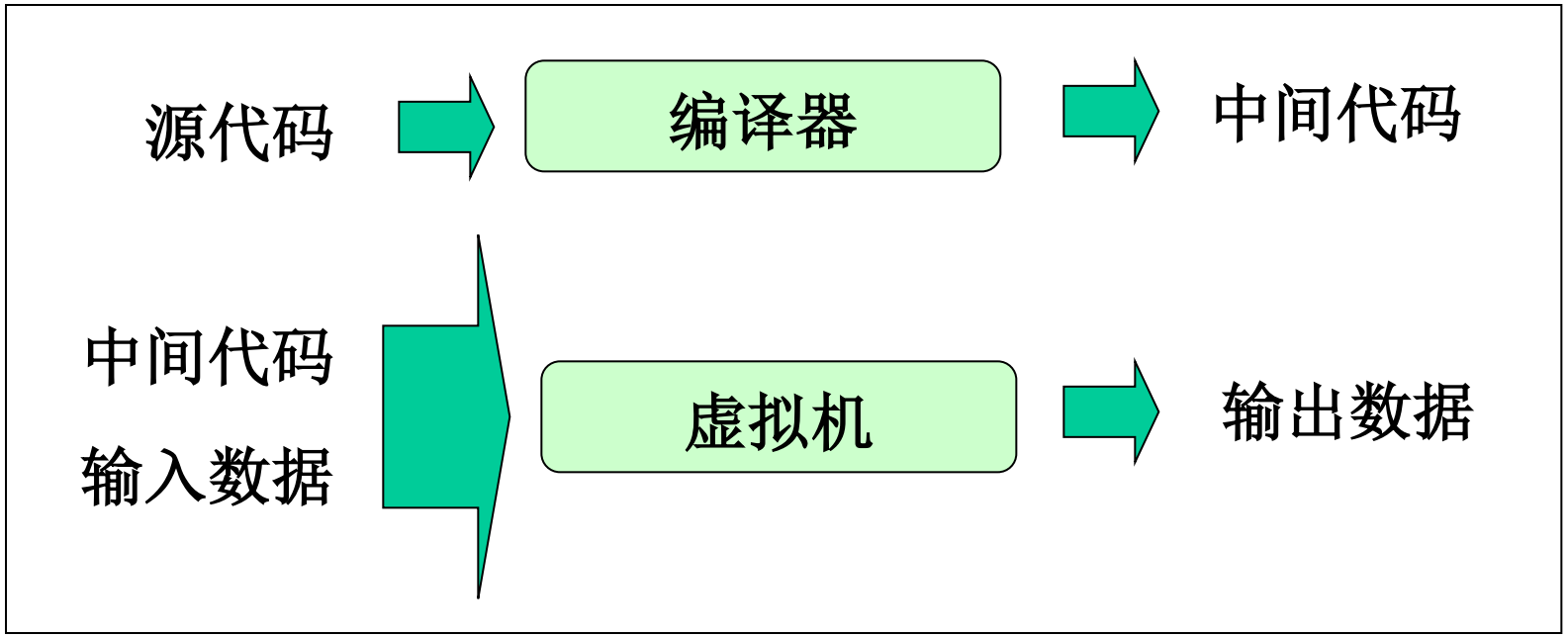
编译器 vs. 解释器 (3/5)

- 大多数语言通常属于二者之一：
  - 编译器：FORTRAN, COBOL, C, C++, Pascal, PL/1
  - 解释器：Lisp, BASIC, APL, Perl, Python, Smalltalk

- 在现实中并不总是这样来区分的
  - e.g., Java, JVM(Just-in-time, JIT Compiler)

编译器 vs. 解释器 (4/5)

- 实际使用中，在二者之间并没有明确的边界
  - 常见的情形是二者混合起来



编译器 vs. 解释器 (5/5)

编译器

- 优点
  - 执行速度快
  - 占空间小
- 缺点
  - 调试困难
  - 目标代码不可移植
  - 额外的编译时间

解释器

- 优点
  - 易于调试
  - 开发快捷
  - 便于移植
- 缺点
  - 执行速度较慢
    - 解释器必须常驻内存
    - JIT Compiler提高了性能
  - 占用较多空间

编译程序概述

- 什么是程序设计语言
- 什么是编译程序
- 编译过程和编译程序的结构
- 编译技术和软件工具的介绍

一、人工翻译

要求：信 准确 达 流畅 雅 优美

eg：翻译下面的字符串

thequickbrownfoxjumpsoveralazydog

1.区分单词（词法分析）

单词（word）：具有独立意义的最短字符串

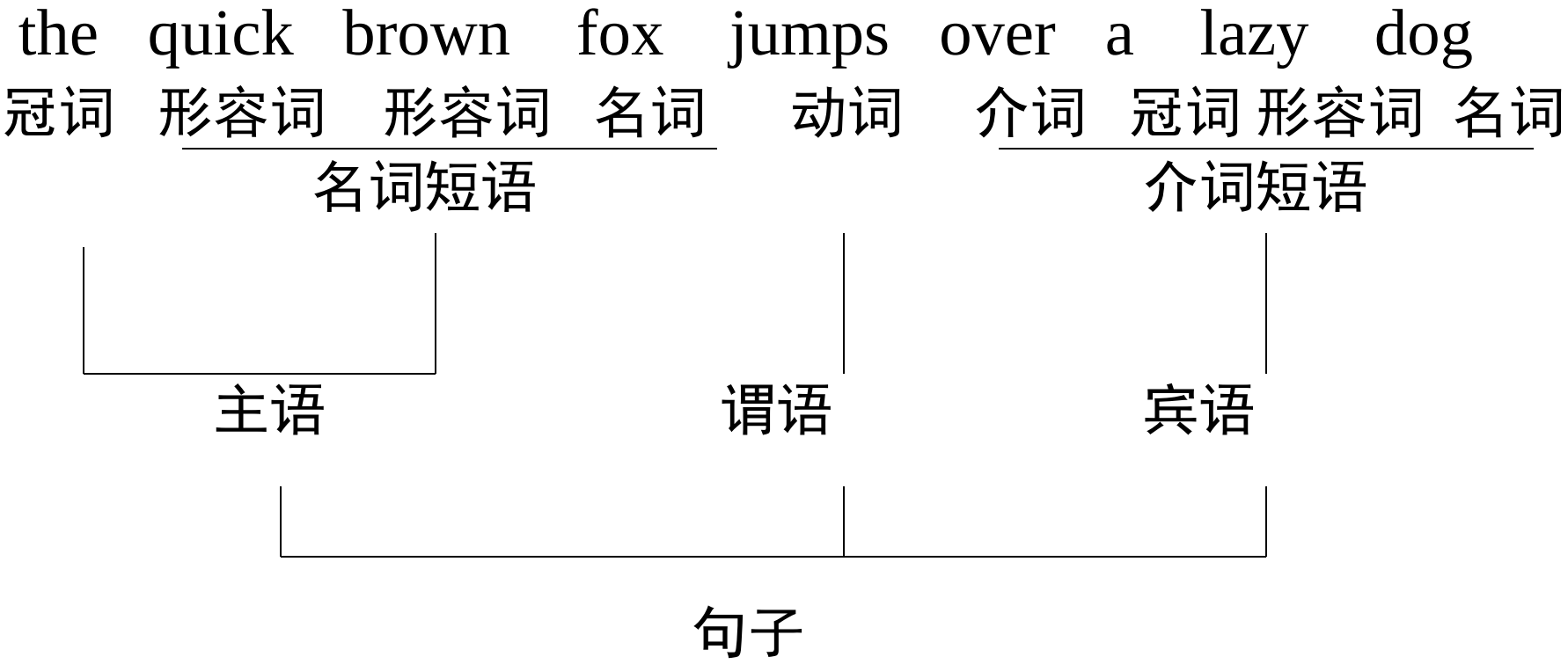
the	quick	brown	fox	jumps	over	a	lazy	dog
冠词	形容词	形容词	名词	动词	介词	冠词	形容词	名词

2.识别句子的语法结构（语法分析）

找出主、谓、宾、定、状、补

a.短语　b.主谓宾状

语法树



- 3.分析句子的含义，检查语义错误（语义分析）

这只伶俐的棕色的桌子跳过了一只懒惰的狗

- 4.根据句子结构和单词含义译出粗稿（中间代码生成）

这只伶俐的棕色的狐狸跳过了一只懒惰的狗

- 5.对粗稿进行润色（代码优化）

通顺流畅 琅琅上口

- 6.得到最后译文（目标代码生成）

那棕狐跃过一只懒狗

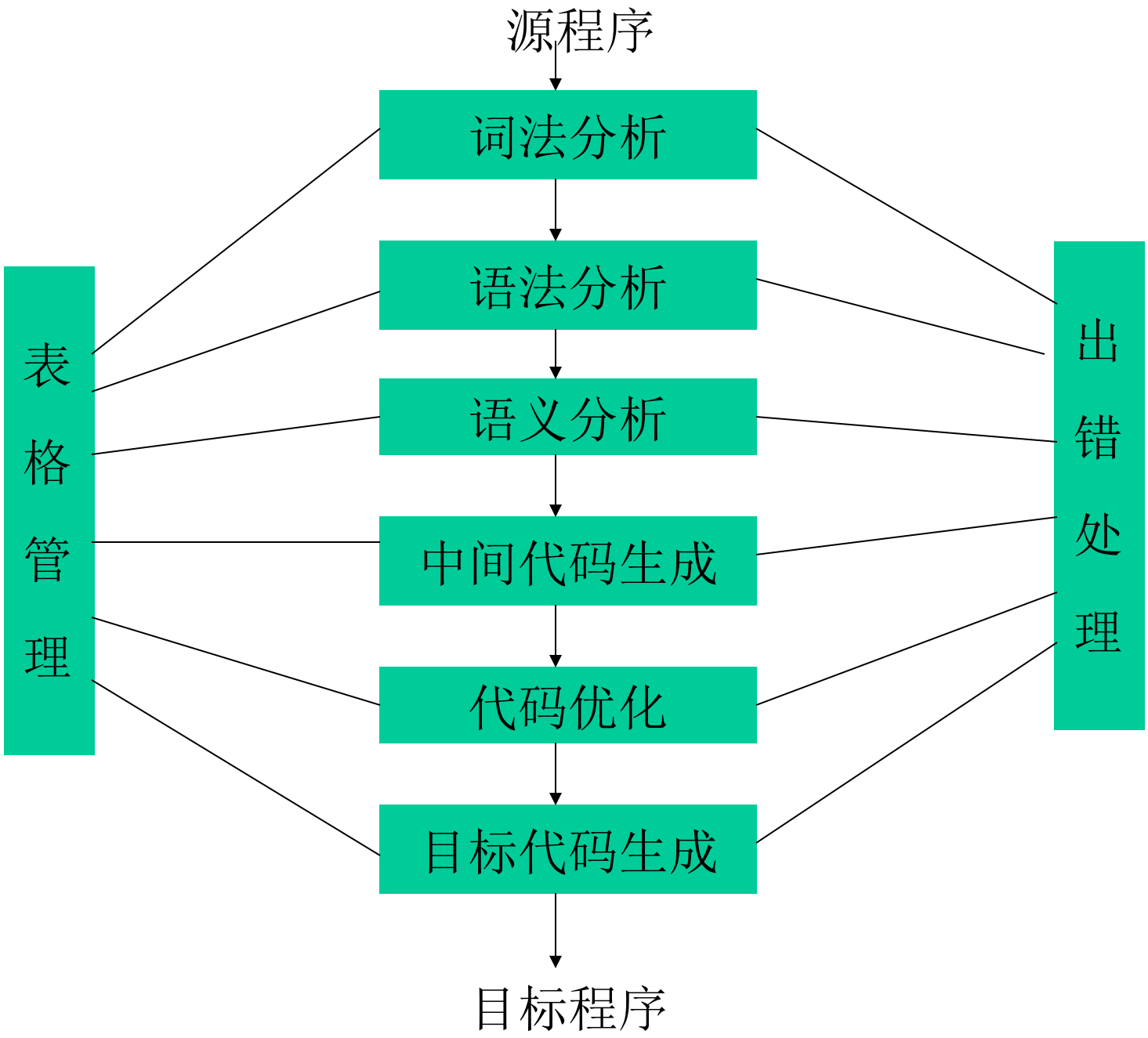
编译的各个阶段与人工翻译的步骤非常类似

编译过程概述

- 编译器内部包括了许多步骤或称为阶段，它们执行不同的逻辑操作。将这些阶段设想为编译器中一个个单独的片断是很有用的，尽管在应用中它们是经常组合在一起的，但它们确实是作为单独的代码操作来编写的。

编译过程概述

- 编译工作的基本过程是：
- 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6个阶段。
- 每个阶段都有表格管理和出错处理部分。


源程序
词法分析
语法分析
语义分析
中间代码生成
代码优化
目标代码生成
目标程序
表格管理
出错处理

编译过程概述

- 机器翻译：让计算机翻译人类语言，例如：将汉语翻译为英文。
- 编译：让计算机翻译程序设计语言（人工语言），例如：将C语言编译为机器语言。

词法分析（自动分词+词性标注）

- 像翻译英文句子一样，先要分析单词，弄清各单词的意义和句中的作用，才能对句子进行翻译。
- 主要任务：从左到右一个字符一个字符地读入源程序，对构成源程序的字符流进行扫描和分解，从而识别出一个个单词。
- 单词包括：保留字、标识符、运算符、分界符等。
- 例：

```
position := initial + rate * 60;
```

词法分析（自动分词+词性标注）

position := initial + rate * 60;

单词类型	单词值
标识符1(id1)	position
算符(赋值)	:=
标识符2(id2)	initial
算符(加)	+
标识符3(id3)	rate
算符(乘)	*
整数	60
界符	;

又如一个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	;

有关术语

- 词法分析(lexical analysis or scanning)
  --The stream of characters making up a source program is read from left to right and grouped into tokens,which are sequences of characters that have a collective meaning.
- 单词---token
- 保留字---reserved word
- 标识符 ---identifier(user-defined name)

语法分析（自动句法分析）

- 语法分析程序与自然语言中句子的语法分析类似。语法分析定义了程序的结构元素及其关系。通常将语法分析的结果表示为分析树或语法树。
- 主要任务：在词法分析的基础上，将单词分解成各类语法短语。
- 一般语法短语可表示成语法树。
- 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 ;

语法分析（自动句法分析）

规则

<赋值语句>::=<标识符>“:=”<表达式>

<表达式>::=<表达式>“+”<表达式>

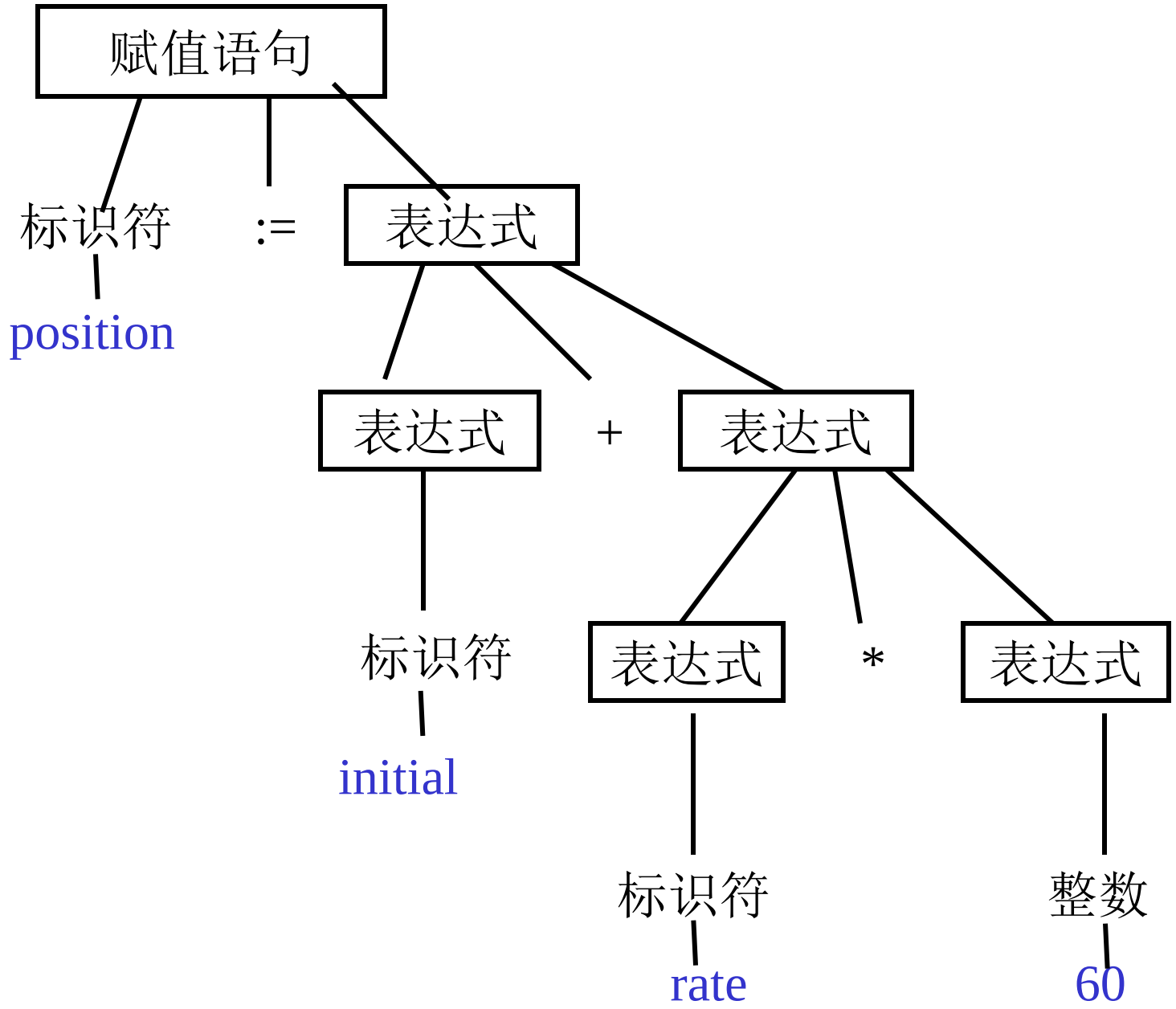
<表达式>::=<表达式>“*”<表达式>

<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
position
表达式
+
表达式
标识符
表达式
*
表达式
initial
标识符
整数
rate
60

术语

- 语法分析(syntax analysis or parsing)

The purpose of syntax analysis is to determine the source program's phrase structure. This process is also called parsing. The source program is parsed to check whether it conforms to the source language's syntax, and to construct a suitable representation of its phrase structure.

- 语法树(推导树)(parse tree or derivation tree)

语义分析

语义审查(静态语义)

- 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。
  - 类型匹配
  - 类型转换

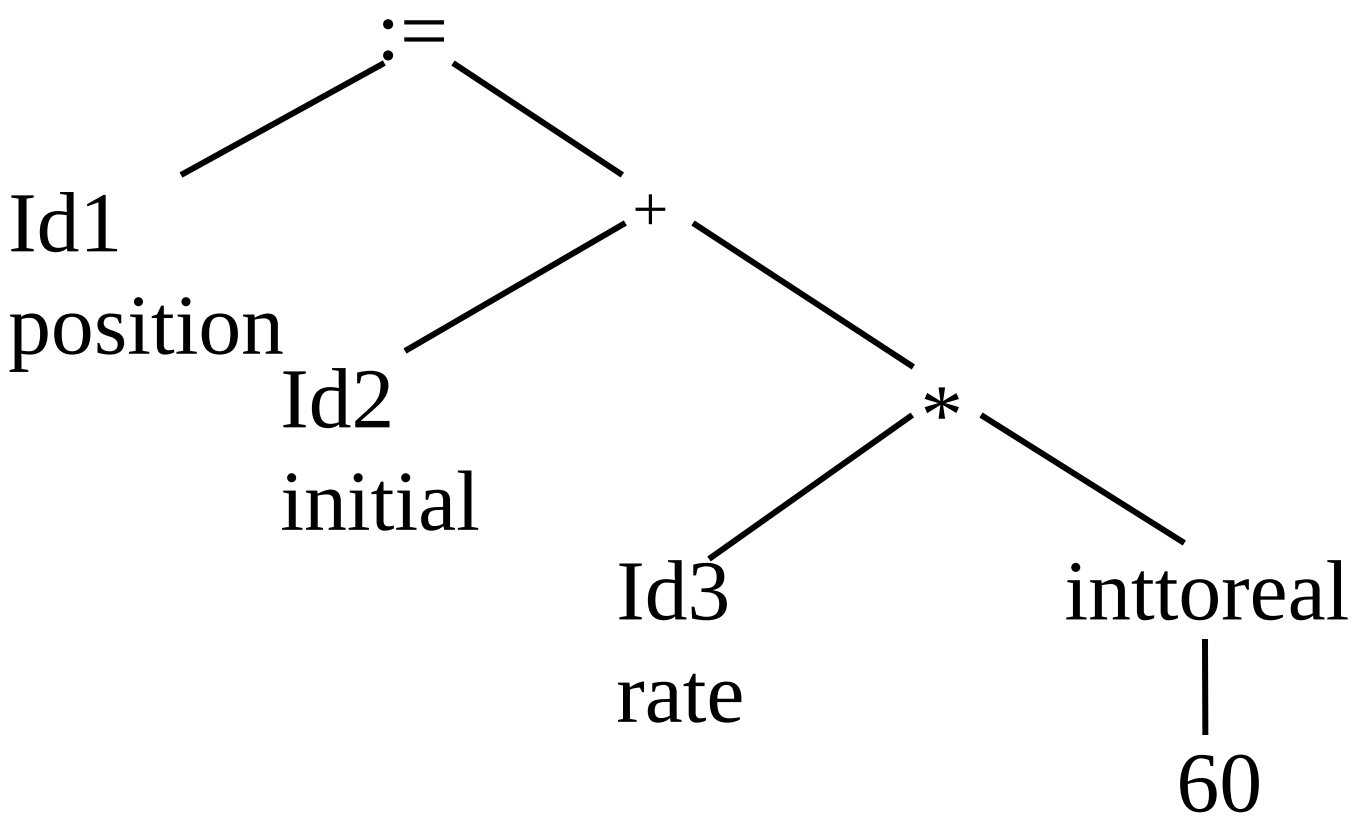
例:

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

又如:

- Program p();
- Var rate:real;
- Var initial :real;
- Var position :real ;
- …
- position := initial + rate * 60

语义分析



术语

- 语义分析(semantic analysis)

  The parsed program is further analyzed to determine whether it conforms to the source language's contextual constraints: scope rules, type rules

e.g. To relate each applied occurrence of an identifier in the source program to the corresponding declaration.

4、中间代码生成：

语法分析和语义分析之后，有时将源程序变成一种内部表示形式，这种内部表示形式叫中间代码，该代码是一种简单的记号系统。三元式、四元式等

四元式的形式为：（算符，运算对象1，运算对象2，结果）

id1　id2　id3

如：sum：＝first＋count×10 生成四元式序列：

（inttoreal 10 － t1 ）

（× id3 t1 t2）

（＋ id2 t2 t3）

（：＝ t3 － id1）

t1，t2，t3是临时变量

运算符　运算对象1　运算对象2　结果

中间代码生成(intermediate code generation)

This is where the intermediate representation of the source program is created. We want this representation to be easy to generate, and easy to translate into the target program. The representation can have a variety of forms, but a common one is called three-address code or 4- tuple code.

代码优化

主要任务：对中间代码进行变换，使目标代码更为高效。（节省时间和空间）

id1:= id2 + id3 * 10

(1)	(inttoreal	10	-	t1	)
(2)	(*	id3	t1	t2	)
(3)	(+	id2	t2	t3	)
(4)	(:=	t3	-	id1	)

变换 ⇒

（1）	(*	id3	10.0	t1	)
(2）	（ +	id2	t1	id1	)

代码优化(code optimization)

- Intermediate code optimization

The optimizer accepts input in the intermediate representation and output a version still in the intermediate representation. In this phase, the compiler attempts to produce the smallest, fastest and most efficient running result by applying various techniques.

- Object code optimization

目标代码生成

主要任务：将中间代码变换成特定机器上的绝对指令代码或可重定位的汇编指令代码。主要与硬件系统结构和指令含义有关。

(* id3 60.0 t1)

(+ id2 t1 id1)

```
MOV   id3,R2
MUL   #60.0,R2
MOV   id2,R1
ADD   R2,R1
MOV   R1,id1
```

- 目标代码三种形式：
  - 绝对指令代码：可直接运行
  - 可重新定位指令代码：需要连接装配
  - 汇编指令代码：需要进行汇编

符号表管理

- 记录源程序中使用的名字
- 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

符号表(symbol table)

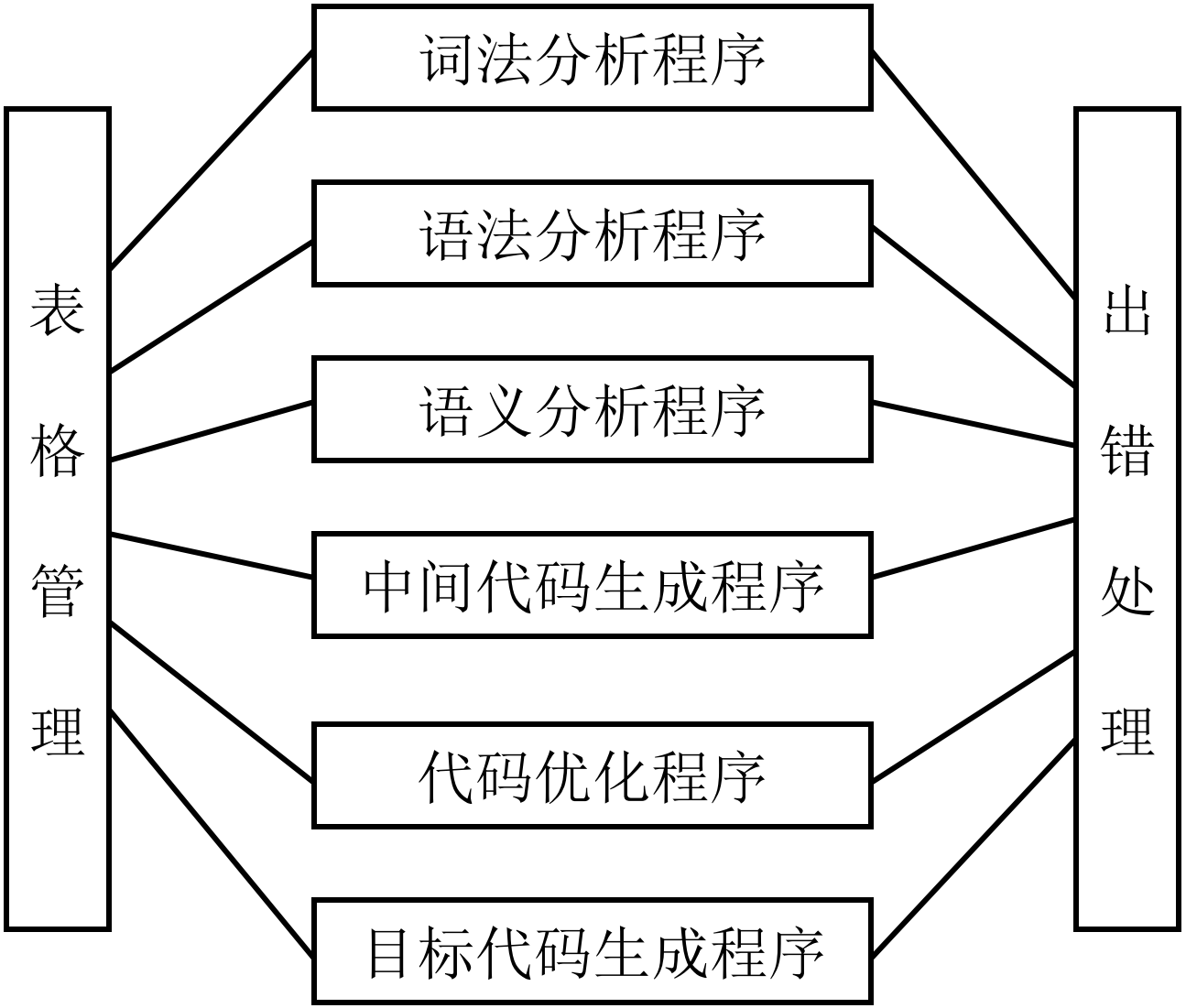
- Symbol table is a data structure which is employed to associate identifiers with their attributes .
- An identifier’s attribute consists of information relevant to contextual analysis, and is obtained from the identifier’s declaration.

错误诊断与处理

- 编译程序在各个阶段应诊断和报告源程序中的错误
  - 词法错误
  - 语法错误
  - 语义错误
- 编译程序应报告出错位置（文件、行、列），并给出相应的纠错提示信息。
- 出错处理能力的大小是衡量编译程序质量好坏的一个重要指标。

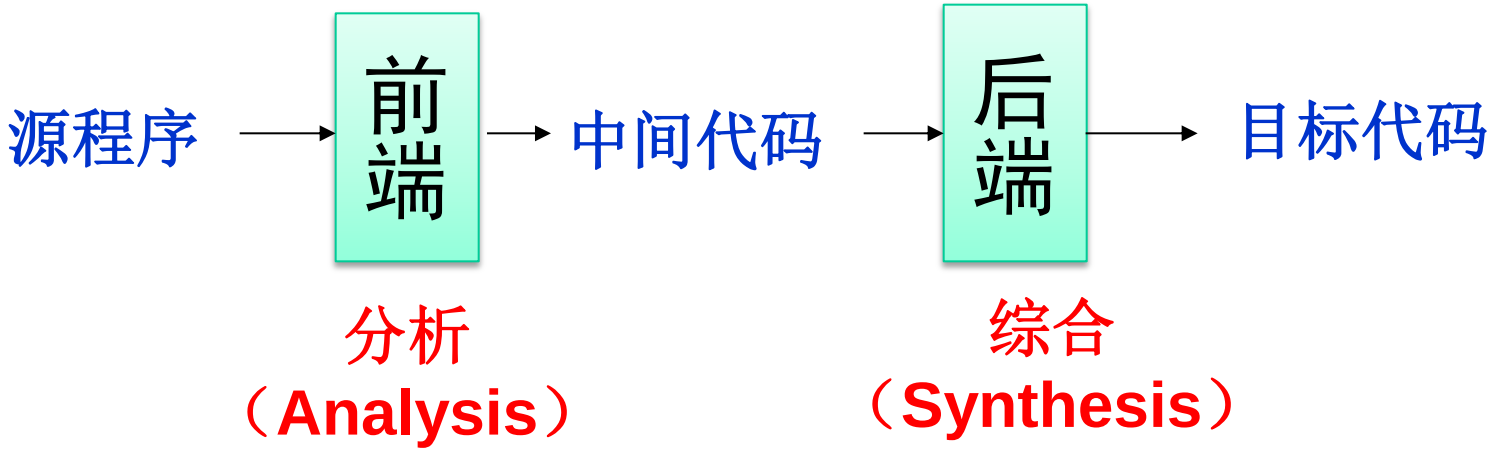
出错处理(error handling)(error reporting and error recovery)

- The compiler should report the location of each error,together with some explanation. The major categories of compile-time error: syntax error, scope error, type error.
- After detecting and reporting an error,the compiler should attempt error recovery,means that the compiler should try to get itself into a state where analysis of the source program can continue as normally as possible.


词法分析程序
语法分析程序
语义分析程序
中间代码生成程序
代码优化程序
目标代码生成程序
表格管理
出错处理

编译程序（器）的组织

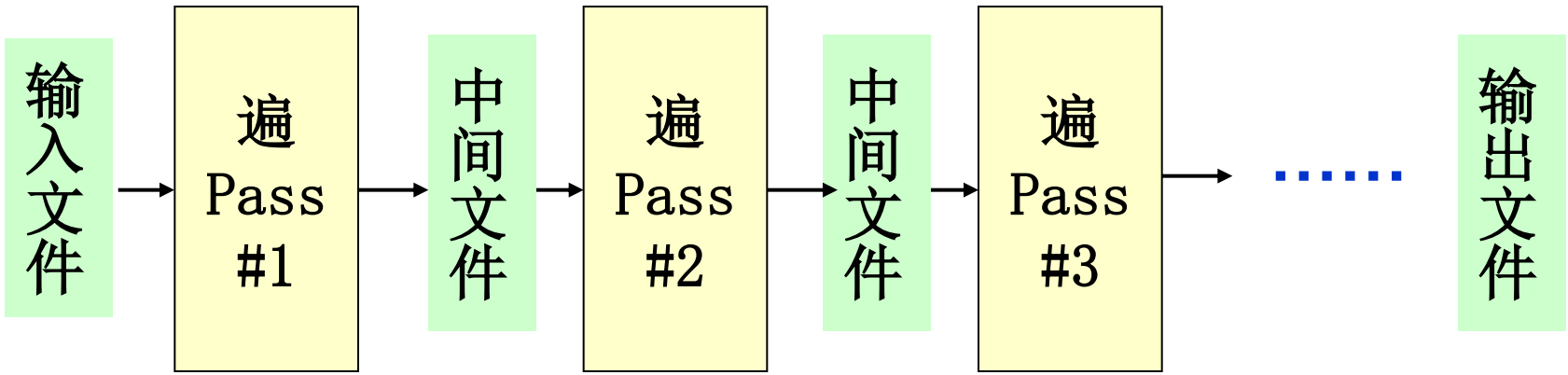
- 前端（frontend）：从源代码到中间代码的翻译，与具体机器无关
- 后端（backend）：把中间代码翻译为具体的机器指令，与源语言无关



划分为前后端，有利于移植编译系统和利用后端为同一目标机配置不同语言的编译系统。

遍（PASS）

- 对源程序或其等价的中间形式（通常以文件形式存在）从头到尾扫视一次，完成预定的处理任务。



编译阶段的组合

主要与源语言有关

编译过程分为

前端：词法分析、语法分析、语义分析、中间代码生成、优化工作。

后端：目标代码生成、出错处理、符号表操作。

主要与目标代码有关

- 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

编译技术和软件工具

用到编译原理与技术的常见软件工具：

1、语言的结构化编辑器：

提供关键字及其匹配的关键字。

可减少语法错误，加快源程序调试。

2、语言程序的调试工具

提供判定程序的算法与功能是否正确。

3、程序格式化工具：使程序呈现清晰的结构

4、语言程序的测试工具：

测试工具

静态分析器 — 对源程序进行语法分析

动态测试器 — 测试用例、对程序进行动态测试。

5、高级语言之间的转换工具：

一种高级语言转化成另一种高级语言。

编译器构造工具

- 目标：辅助用户生成复杂的、容易出错的编译分析程序
- 词法分析工具
  - Lex, Flex
- 语法分析工具
  - YACC, Bison, ANTLR

总结

- 什么是编译程序
- 编译和解释的区别
- 编译过程和编译程序的结构
- 一些相关的软件工具

课后准备

- 准备作业本
- 借到或买到教材
  - **编译原理，清华大学出版社 张素琴、吕映芝 等编著，2005年**
- 作业：编译的发展历程