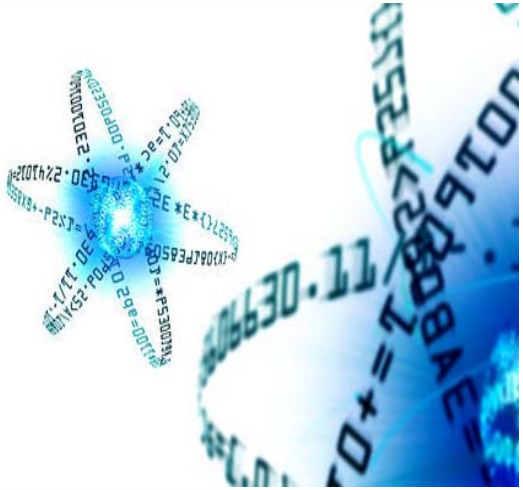
编译原理

Compilers – Principles, Techniques and Tools



计算机科学与技术学院

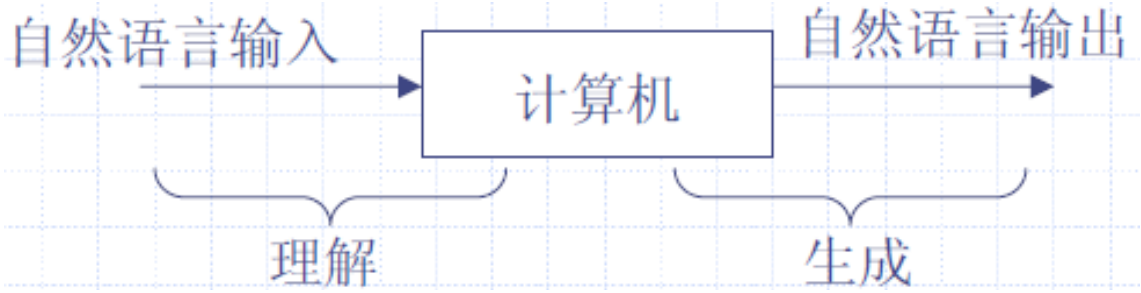
编译原理

- 主讲：周国栋、段湘煜

http://nlp.suda.edu.cn/

编译原理

- 主讲：周国栋、段湘煜



http://nlp.suda.edu.cn/

编译原理

- 讲授时间：星期四 14：00~17：00 文思209
- 实验时间：星期一 08：00 ~09：50 单周
- 课本
  - Compilers – Principles, Techniques and Tools
    - Second Edition
    - Alfred V. Aho, Monica S. Lam, Ravi Sethi and Jeffrey D. Ullman.
      - 中文版（赵建华、郑滔、戴新宇译），机械工业出版社，2009.1， ISBN 978-7-111-25121-7.

编译原理

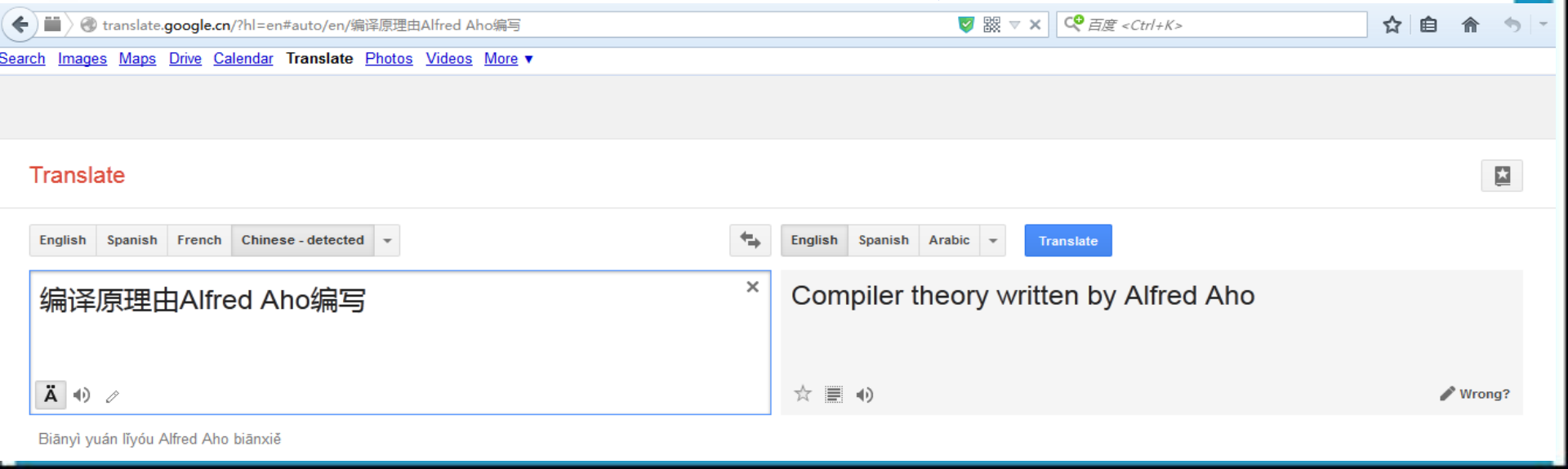
- 讲授时间：星期四 14：00~17：00 文思209
- 实验时间：星期一 08：00 ~09：50 单周
- 课本
  - Compilers – Principles, Techniques and Tools
    - Second Edition
    - Alfred V. Aho, Monica S. Lam, Ravi Sethi and Jeffrey D. Ullman.
      - 中文版（赵建华、郑滔、戴新宇译），机械工业出版社，2009.1， ISBN 978-7-111-25121-7.

编译原理

- 评分
  - 实验考察：20%
  - 期末考试：80%
    - 重点：第三章至第五章 50-60%
    - 其他：20-30%

第一章：编译简介

- 引论
  - 翻译器：
    - 能够完成从一种语言到另一种语言变换的软件



第一章：编译简介

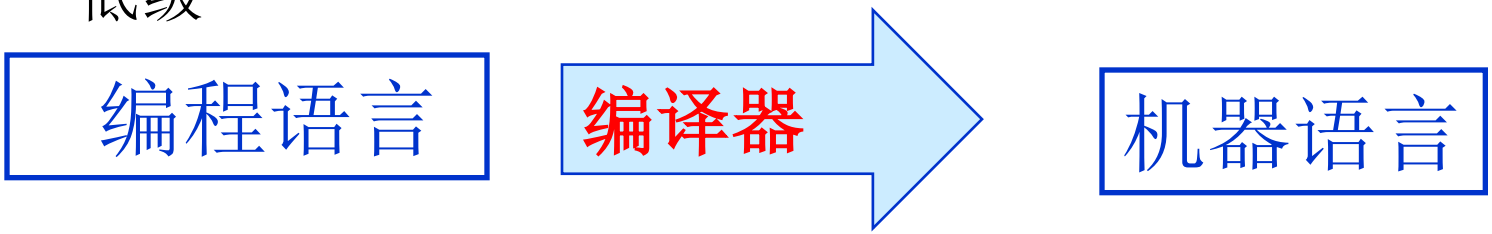
- 引论
  - 翻译器：
    - 能够完成从一种语言到另一种语言变换的软件

第一章：编译简介

- 引论
  - **编译器：**
    - 编译器是一种翻译器，它的特点是目标语言比源语言低级

第一章：编译简介

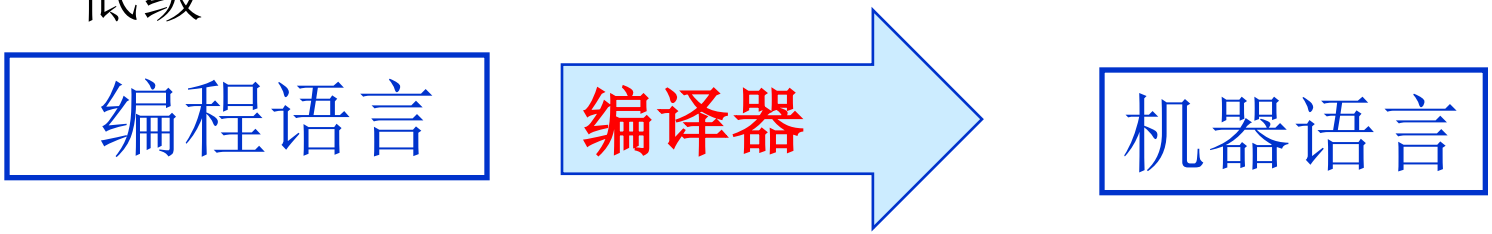
- 引论
  - **编译器：**
    - 编译器是一种翻译器，它的特点是目标语言比源语言低级



  - 编程语言
    - 传统程序设计语言：Pascal、C++、Java
    - 专用语言：Lisp、Prolog、LaTEX

第一章：编译简介

- 引论
  - **编译器：**
    - 编译器是一种翻译器，它的特点是目标语言比源语言低级



- 机器语言例子：

操作：寄存器BX的内容送到AX中

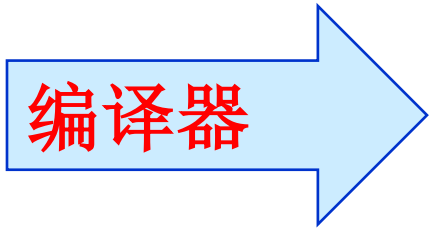
1000100111011000　　机器指令

mov ax,bx　　汇编指令

第一章：编译简介

- 引论
  - **编译器：**
    - 编译器是一种翻译器，它的特点是目标语言比源语言低级

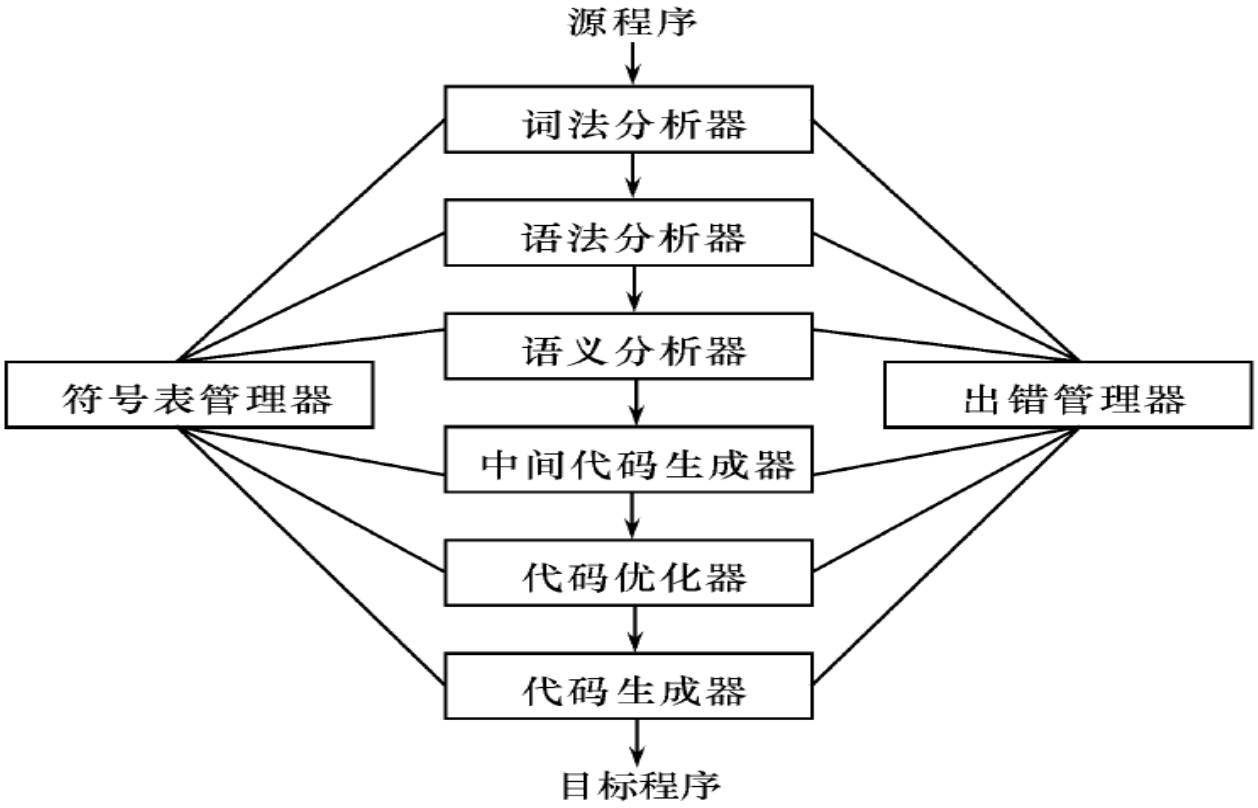
position := initial + rate * 60




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

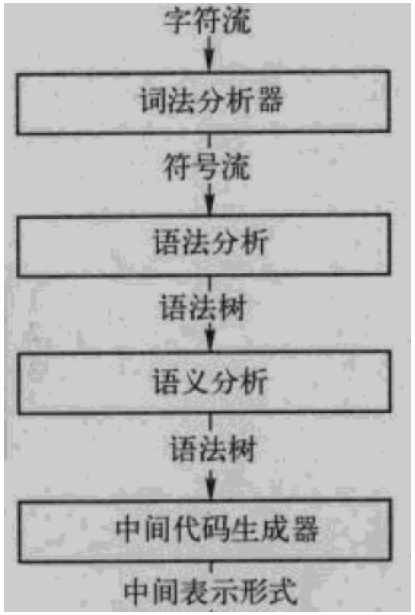
第一章：编译简介

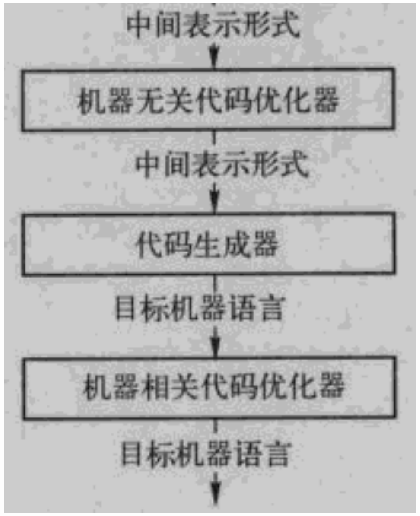
- 引论
  - 编译器的工作可以分成若干阶段，每个阶段把源程序从一种表示变换成另一种表示。



第一章：编译简介

- 引论
  - 编译器的工作可以分成若干阶段，每个阶段把源程序从一种表示变换成另一种表示。



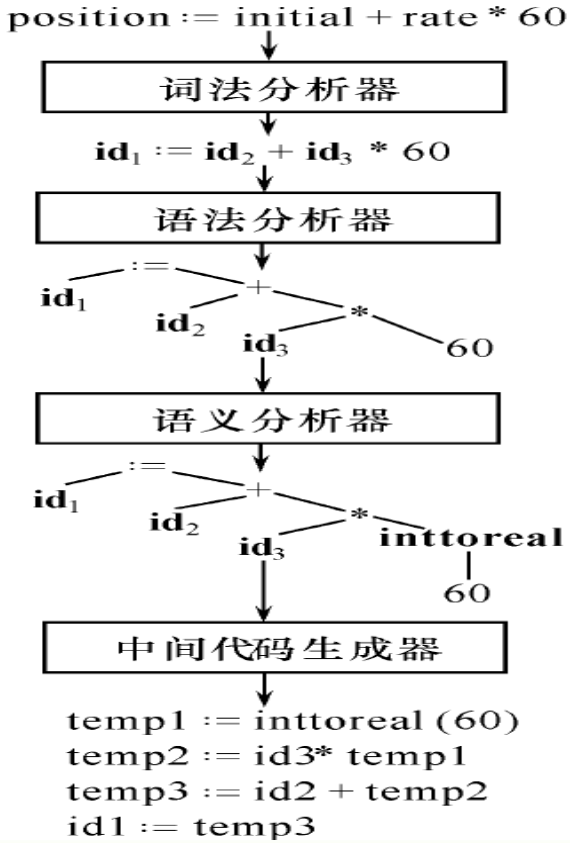


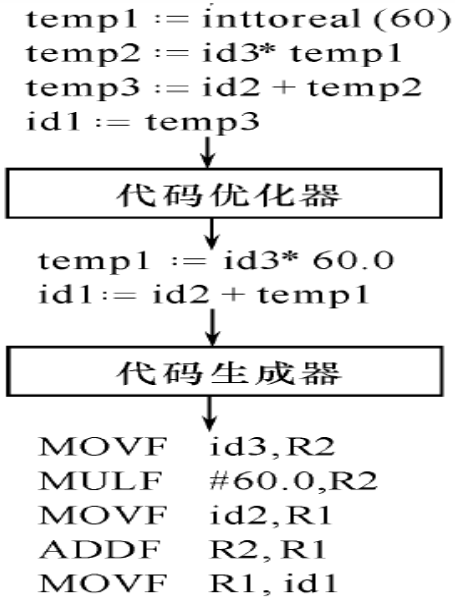
第一章：编译简介

- 引论
  - 编译器的工作可以分成若干阶段，每个阶段把源程序从一种表示变换成另一种表示。
  - 例：

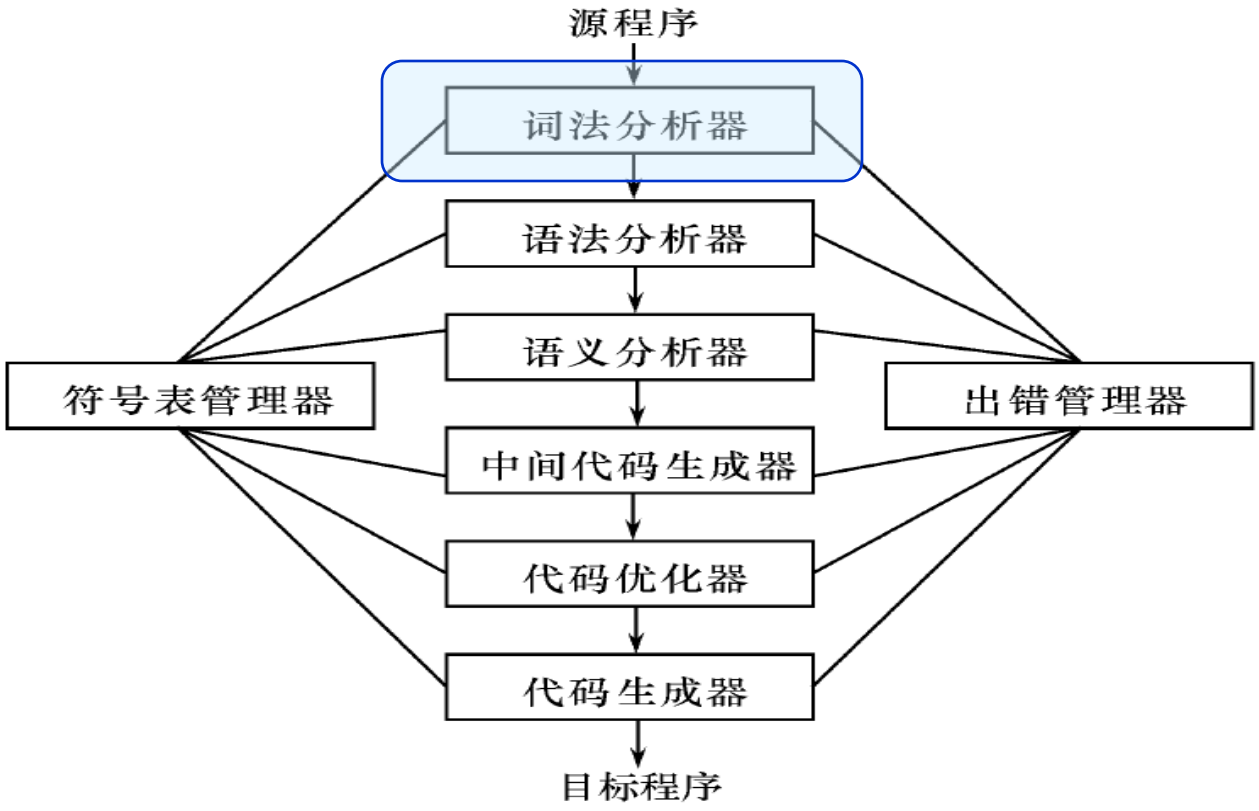
符号表

1	position	...
2	initial	...
3	rate	...
	...	...





第一章：编译简介


源程序
词法分析器
语法分析器
语义分析器
符号表管理器
出错管理器
中间代码生成器
代码优化器
代码生成器
目标程序

第一章：编译简介

- 词法分析
  - 逐个读构成源程序的字符，把它们组成词法记号(*token*)流。
  - 例：$position := initial + rate * 60$
    - （1）标识符（position）
    - （2）赋值号（:=）
    - （3）标识符（initial）
    - （4）加号（+）
    - （5）标识符（rate）
    - （6）乘号（*）
    - （7）数（60）

第一章：编译简介

- 词法分析
  - 逐个读构成源程序的字符，把它们组成词法记号(*token*)流。
  - 例：

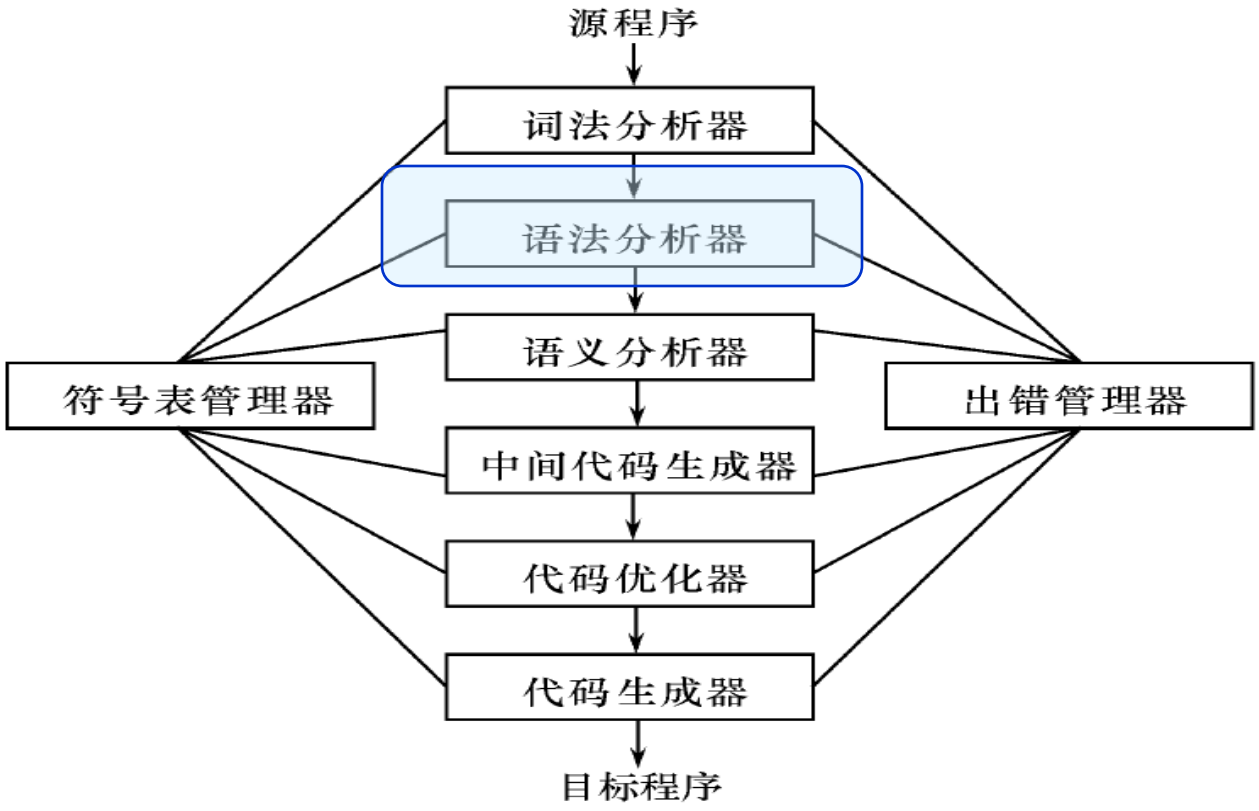
$$position := initial + rate * 60$$

词法分析

$$id_1 := id_2 + id_3 * 60$$

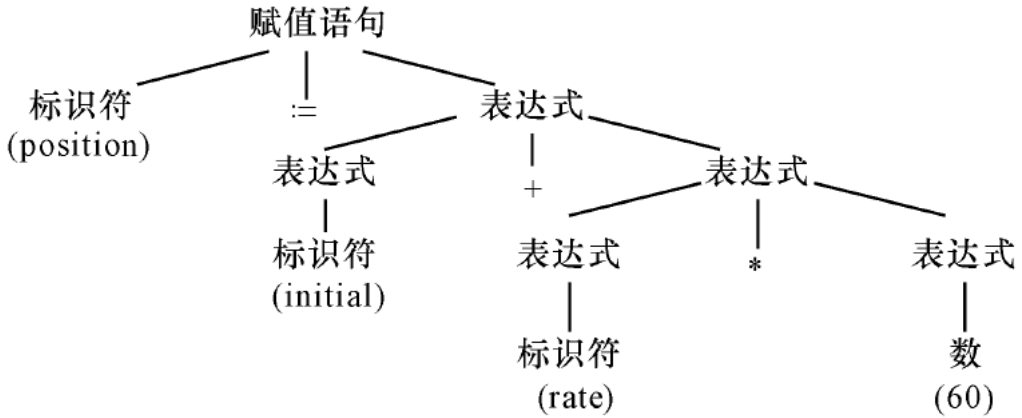
  - 编译器的词法分析也叫做线性分析或扫描。

第一章：编译简介


源程序
词法分析器
语法分析器
语义分析器
符号表管理器
出错管理器
中间代码生成器
代码优化器
代码生成器
目标程序

第一章：编译简介

- 语法分析(syntax analysis)，简称为分析(parsing)
  - 把词法记号流依照语言的语法结构按层次分组，以形成语法短语。
    - 例： $position := initial + rate * 60$ 的**分析树**：



第一章：编译简介

- 语法分析
  - 程序的层次结构通常由递归的规则表示
    - （1）任何一个标识符都是表达式；
    - （2）任何一个数都是表达式；
    - （3）如果 e_1 和 e_2 都是表达式，那么

      $e_1 + e_2$

      $e_1 * e_2$

      (e_1)

      也都是表达式。

第一章：编译简介

- 语法分析
  - 同样地，许多语言用类似如下的规则递归地定义语句
    - （１）如果identifier是标识符，expression是表达式，那么

      identifier := expression

      是语句。
    - （２）如果 expression 是表达式，statement是语句，那么

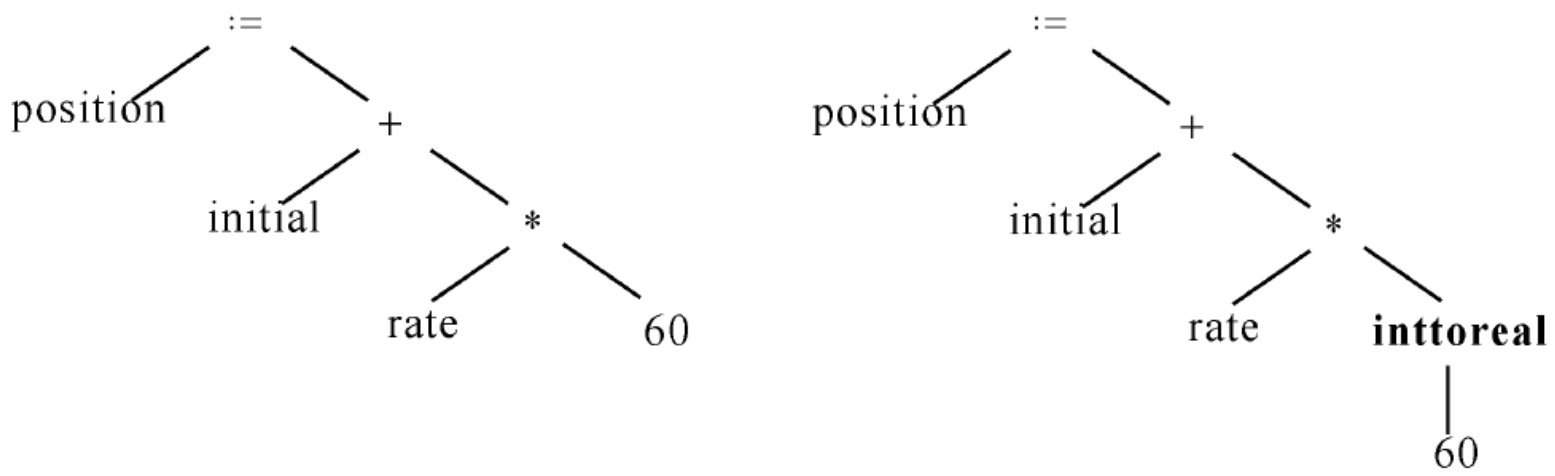
      while (expression) do statement

      if (expression) then statement

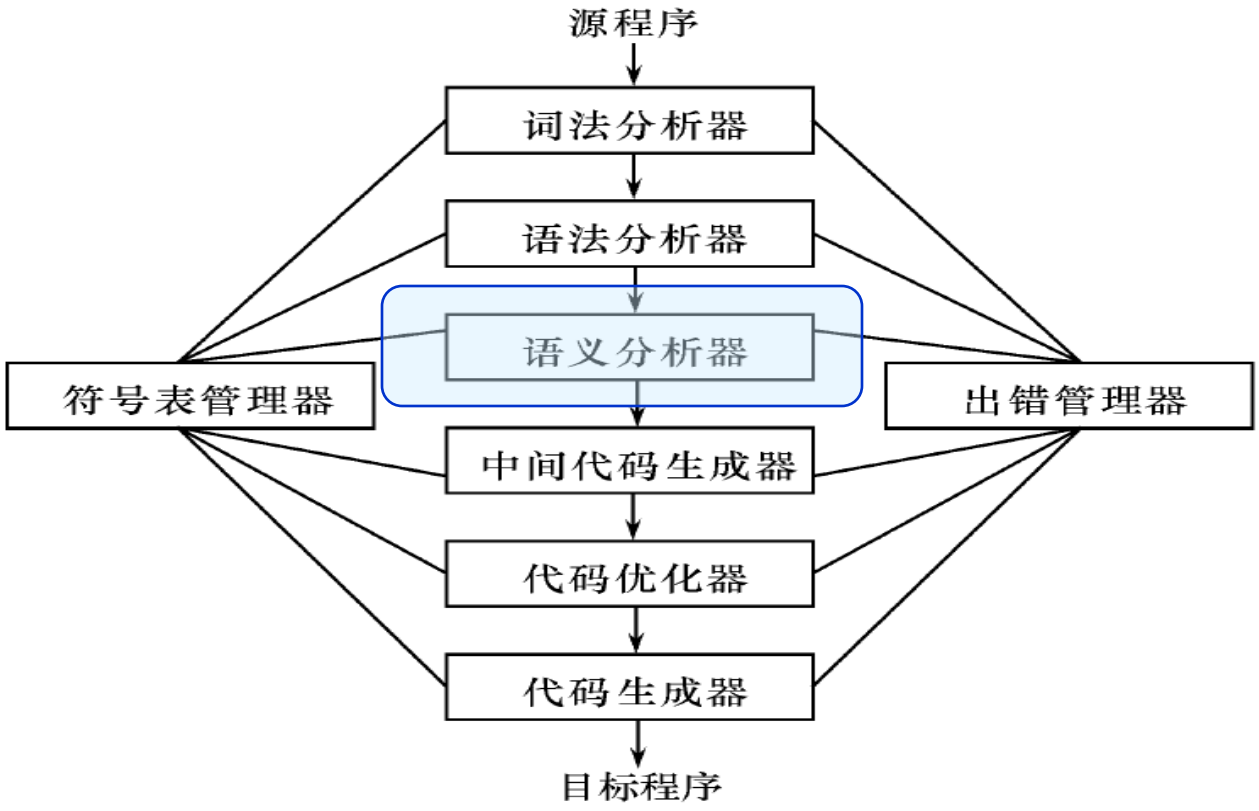
      也都是语句。

第一章：编译简介

- 语法分析(syntax analysis)，简称为分析(parsing)
  - 把词法记号流依照语言的语法结构按层次分组，以形成语法短语。
    - 例： position := initial + rate * 60 的**语法树**：

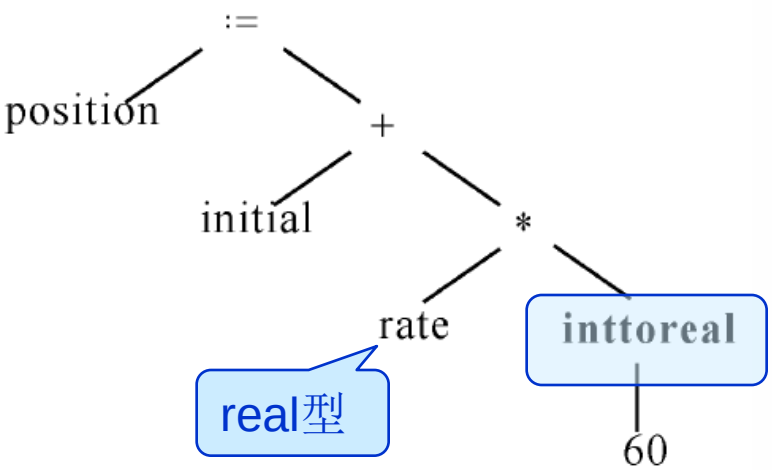


第一章：编译简介


源程序
词法分析器
语法分析器
语义分析器
符号表管理器
出错管理器
中间代码生成器
代码优化器
代码生成器
目标程序

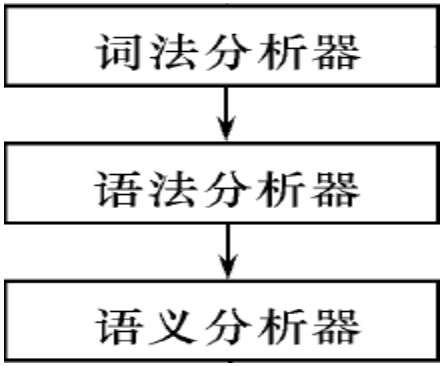
第一章：编译简介

- 语义分析
  - 检查程序的语义正确性，以保证程序各部分能有意义地结合在一起，并为以后的代码生成阶段收集类型信息。
    - 类型转换
    - 类型检查
    - 语法制导的翻译



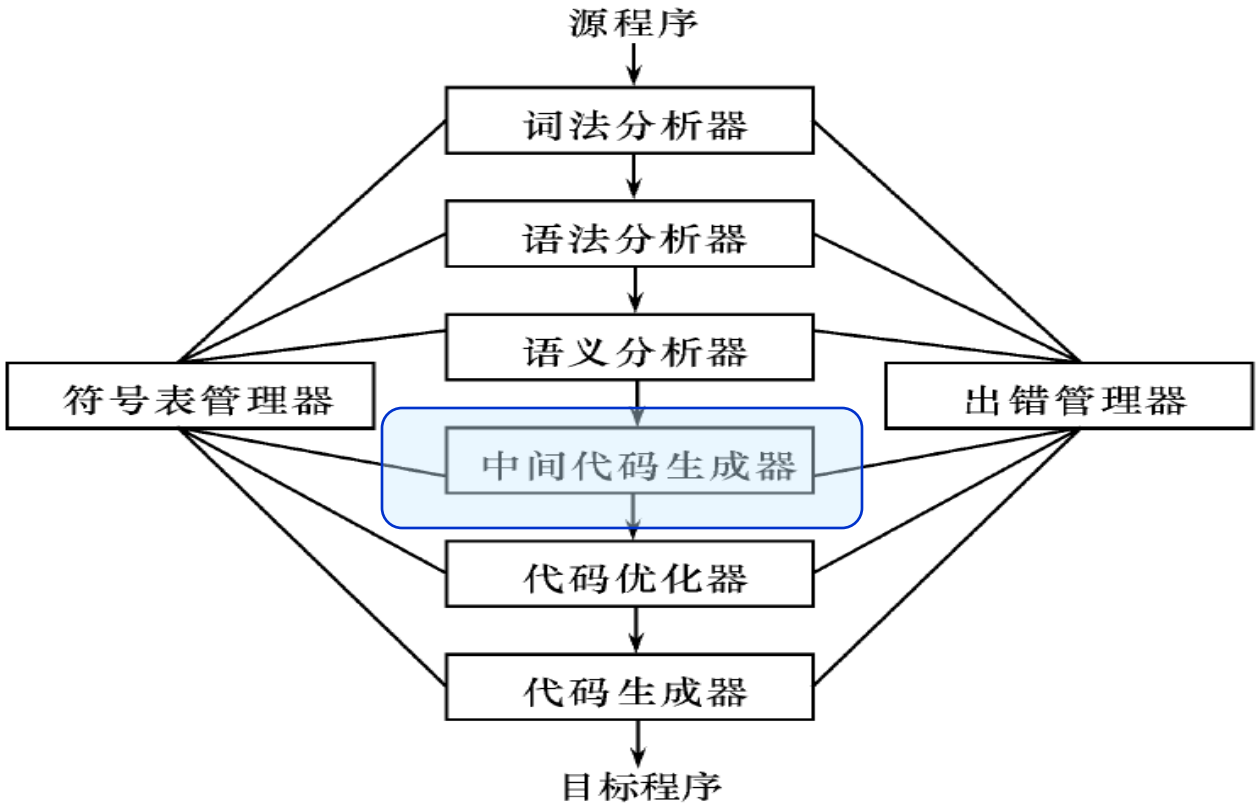
第一章：编译简介

- 编译器的前三个阶段对源程序分别进行不同的分析，以揭示源程序的基本数据和结构，决定它们的含义，建立源程序的中间表示。



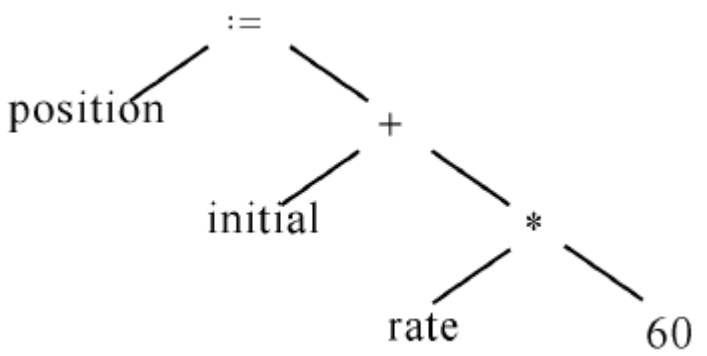
- 许多处理源程序的软件工具都要完成某类分析：
  - 格式打印程序
  - 文档抽取程序
  - 静态检查程序
  - 解释器

第一章：编译简介


源程序
词法分析器
语法分析器
语义分析器
符号表管理器
出错管理器
中间代码生成器
代码优化器
代码生成器
目标程序

第一章：编译简介

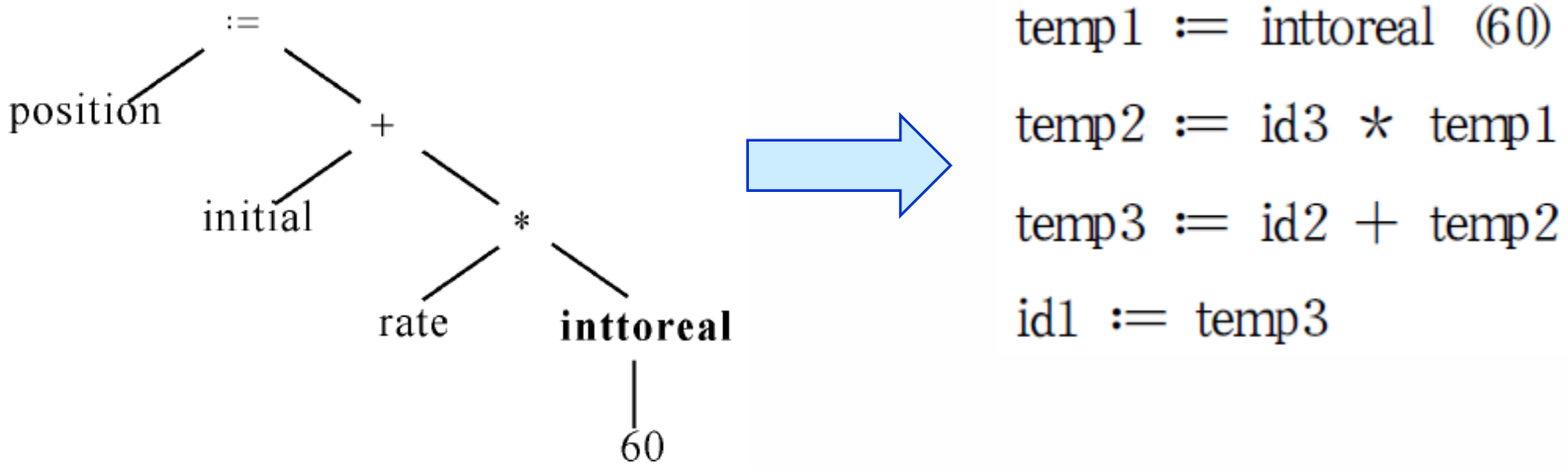
- 中间代码生成
  - 中间代码位于高级编程语言和机器语言（目标程序）之间
    - 后缀表示:　　9+5-2 → 95+2-
    - 抽象语法树:



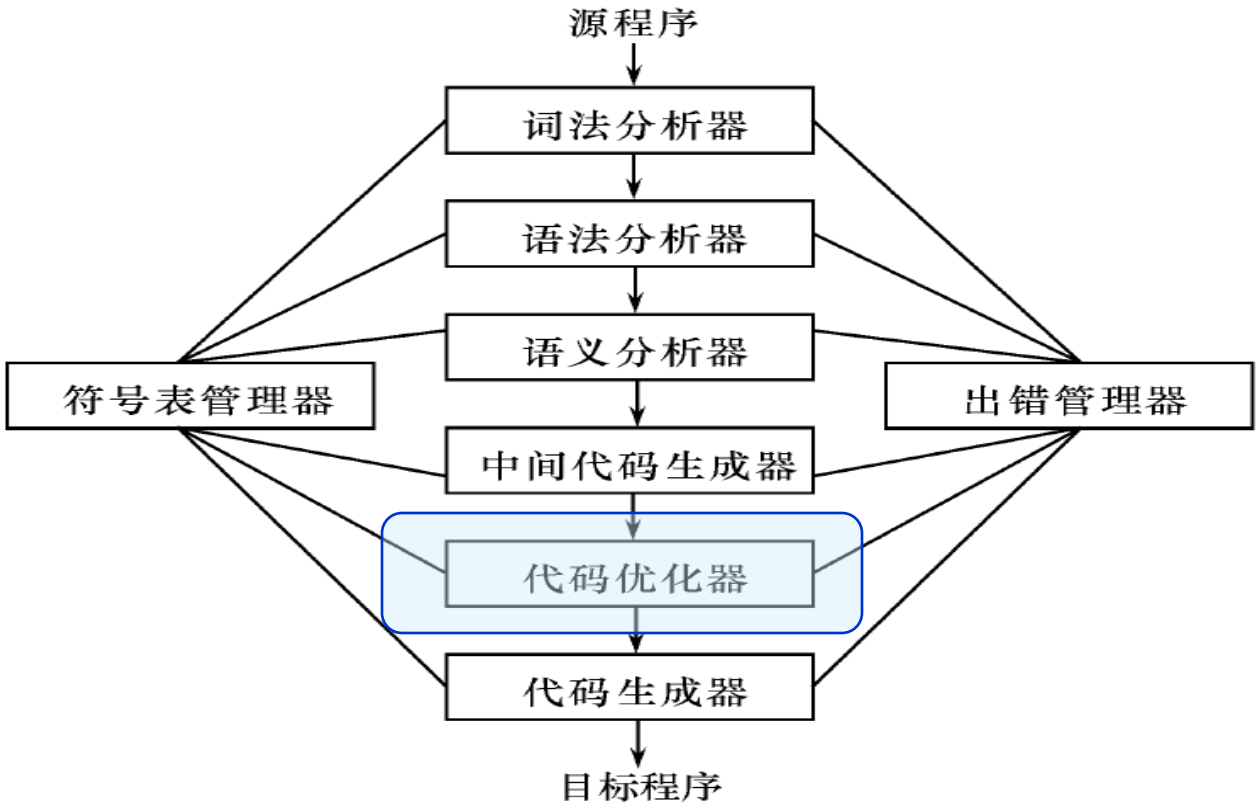
    - 三地址码

第一章：编译简介

- 中间代码生成
  - 中间代码位于高级编程语言和机器语言（目标程序）之间
    - 三地址码



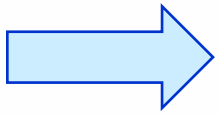
第一章：编译简介


源程序
词法分析器
语法分析器
语义分析器
符号表管理器
出错管理器
中间代码生成器
代码优化器
代码生成器
目标程序

第一章：编译简介

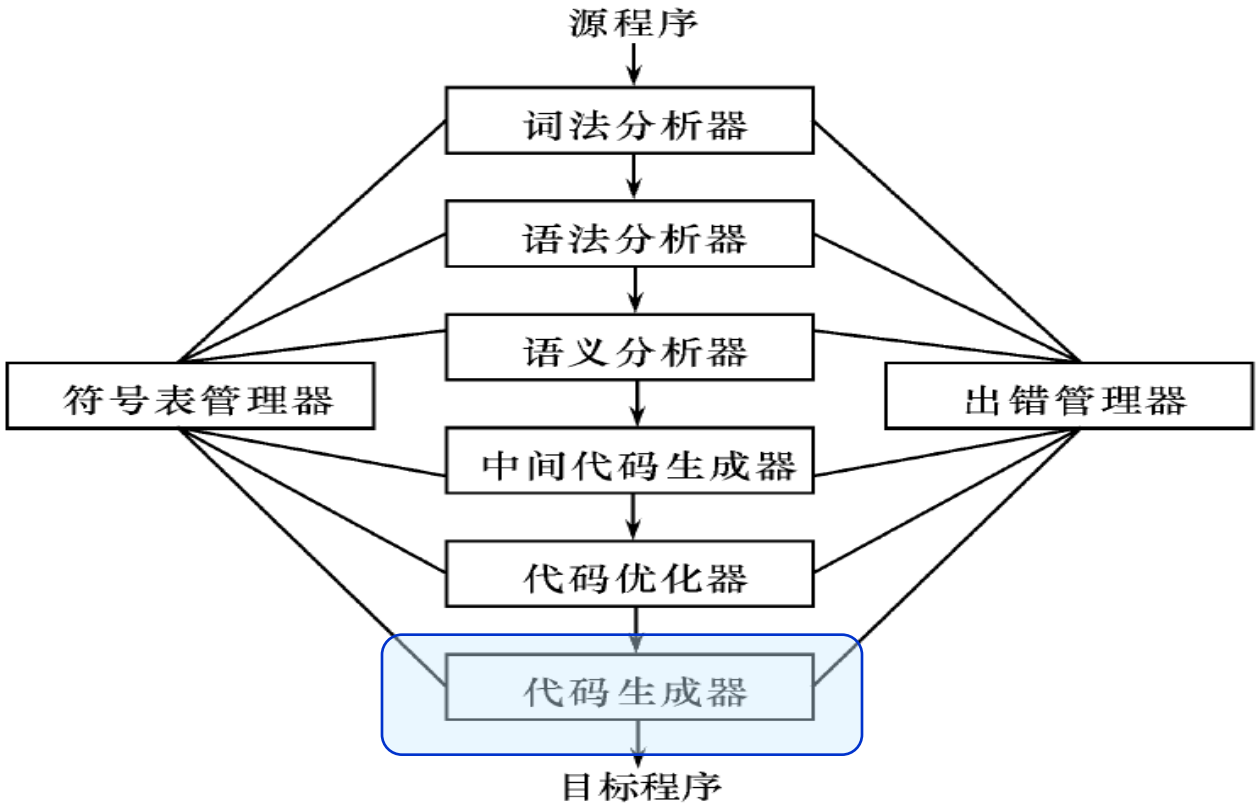
- 代码优化
  - 改进代码，以产生执行较快的机器代码。

```
temp1 := inttoreal (60)
temp2 := id3 * temp1
temp3 := id2 + temp2
id1 := temp3
```



```
temp1 := id3 * 60.0
id1 := id2 + temp1
```

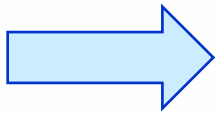
第一章：编译简介


源程序
词法分析器
语法分析器
语义分析器
符号表管理器
出错管理器
中间代码生成器
代码优化器
代码生成器
目标程序

第一章：编译简介

- 目标代码生成
  - 生成可重定位的机器代码或汇编码
    - 为源程序所用的每个变量选择存储单元，并且把中间代码翻译成等价的机器指令序列。
    - 关键问题是寄存器分配。

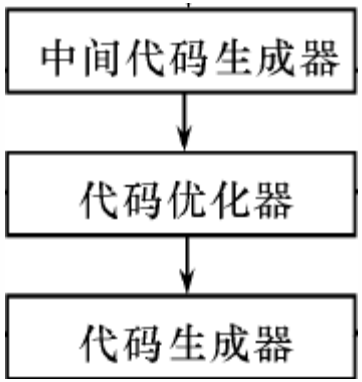
```
temp1 := id3 * 60.0
id1 := id2 + temp1
```



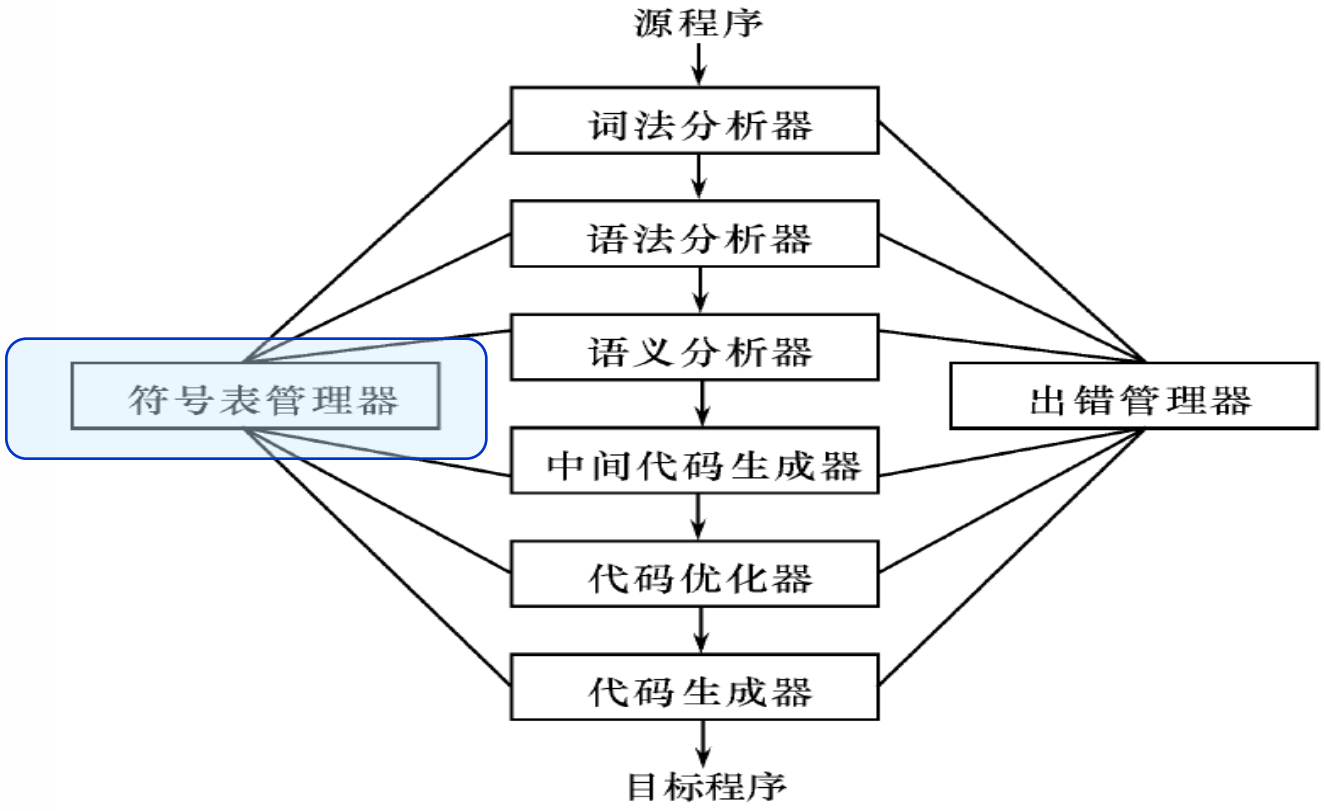
```
MOVF id3, R2
MULF #60.0, R2
MOVF id2, R1
ADDF R2, R1
MOVF R1, id1
```

第一章：编译简介

- 编译器的后三个阶段称为对源程序进行综合，它们从源程序的中间表示建立起和源程序等价的目标程序。



第一章：编译简介


源程序
词法分析器
语法分析器
语义分析器
符号表管理器
出错管理器
中间代码生成器
代码优化器
代码生成器
目标程序

第一章：编译简介

- 符号表管理
  - 编译器的一项重要工作是记录源程序中使用的标识符，并收集每个标识符的各种属性。
  - 这些属性提供标识符的存储分配、类型和作用域信息。如果是过程标识符，还有参数的个数和类型，参数传递方式和返回值类型
  - 符号表是为每个标识符保存一个记录的数据结构，记录的域是标识符的属性。该数据结构允许我们迅速地找到一个标识符的记录，在此记录中存储和读取数据。

第一章：编译简介

符号表管理

- 语句 position := initial + rate * 60 ，放入符号表中

符号表

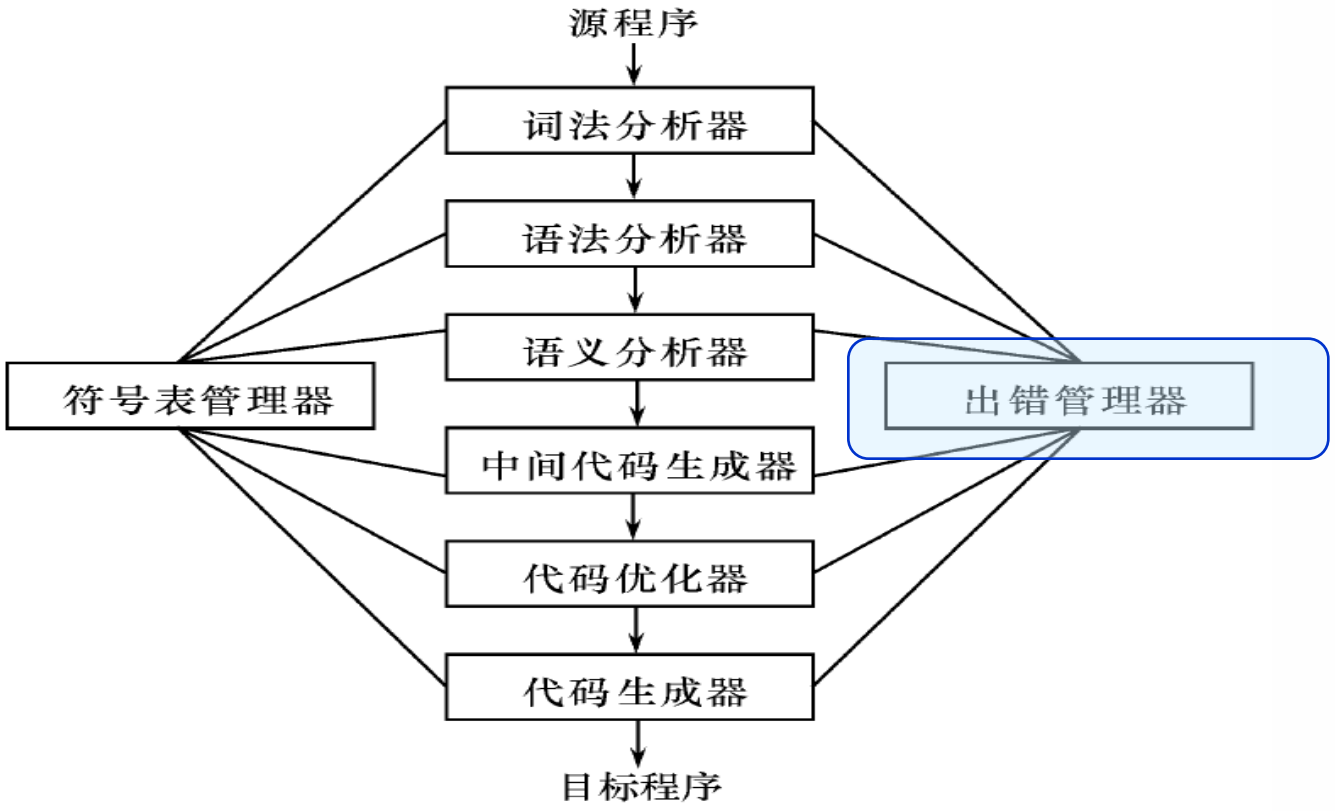
1	position	. . .
2	initial	. . .
3	rate	. . .
	. . .	. . .

- 词法分析器发现源程序的标识符时，把该标识符填入符号表。但是，词法分析期间不能确定一个标识符的属性。

  如：var position，initial，rate ：real ；

- 其余的阶段把标识符的信息填入符号表，然后以不同的方式使用这些信息

第一章：编译简介


源程序
词法分析器
语法分析器
语义分析器
符号表管理器
出错管理器
中间代码生成器
代码优化器
代码生成器
目标程序

第一章：编译简介

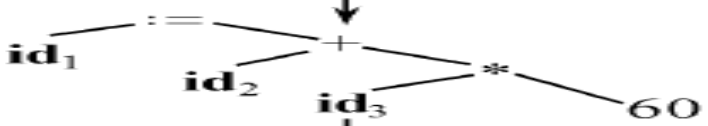
- 错误诊断和报告
  - 每个阶段都有可能发现源程序的错误。在发现错误后，该阶段必须处理此错误，使得编译可以继续进行，以便进一步发现源程序的其他错误
    - 词法分析阶段能发现的错误类型是：当前被扫描的字符串不能形成语言的词法记号。
    - 语法分析阶段诊断：记号流违反语言的语法规则。
    - 语义分析时，编译器试图找出语法正确但对所含的操作来说是无意义的结构，如相加的两个标识符，其一是数组名，另一个是过程名。

position := initial + rate * 60

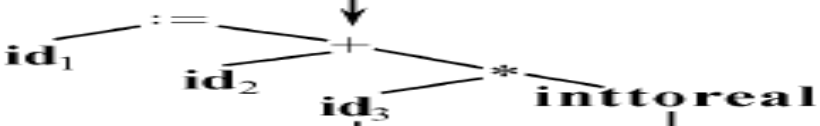
词法分析器

$id_1 := id_2 + id_3 * 60$

语法分析器



语义分析器



中间代码生成器

```
temp1 := inttoreal (60)
temp2 := id3* temp1
temp3 := id2 + temp2
id1 := temp3
```

代码优化器

```
temp1 := id3* 60.0
id1 := id2 + temp1
```

代码生成器

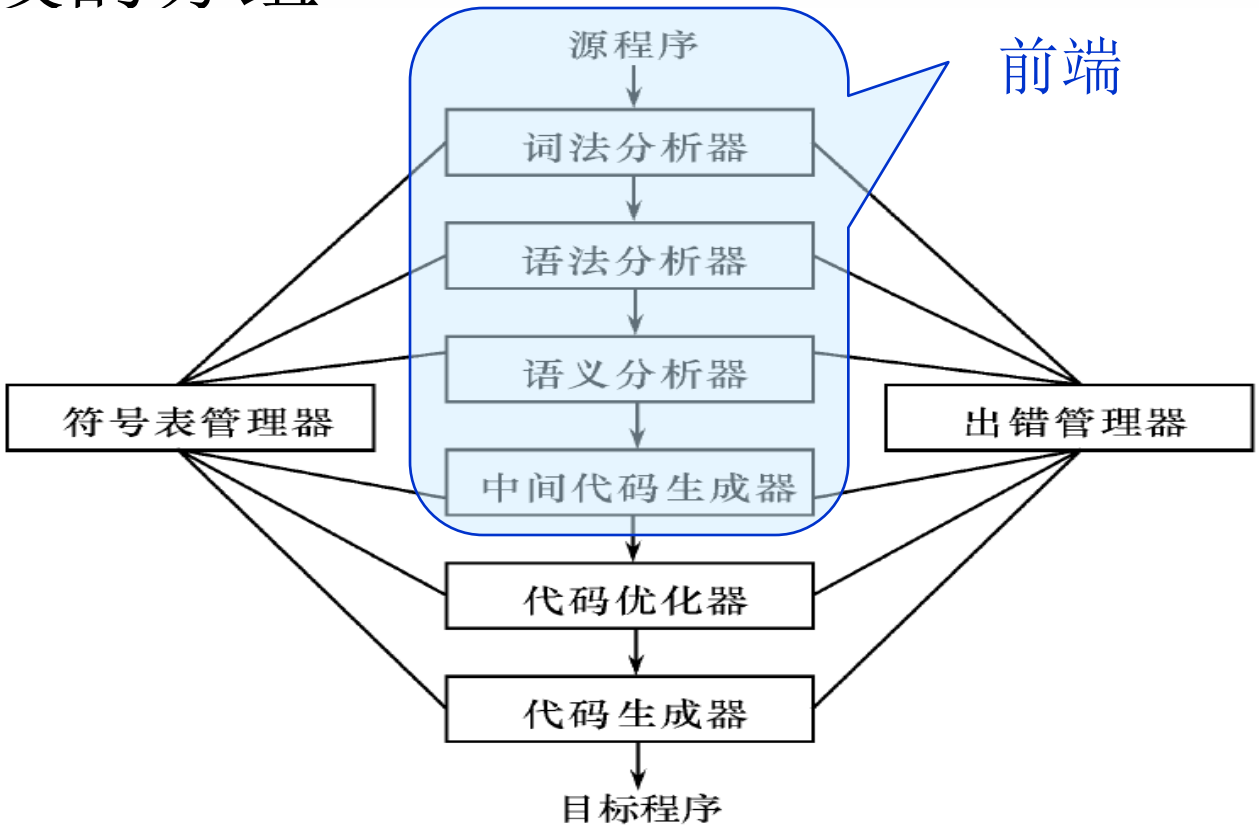
```
MOVF   id3,R2
MULF   #60.0,R2
MOVF   id2,R1
ADDF   R2, R1
MOVF   R1, id1
```

符号表

1	position	. . .
2	initial	. . .
3	rate	. . .
	. . .	. . .

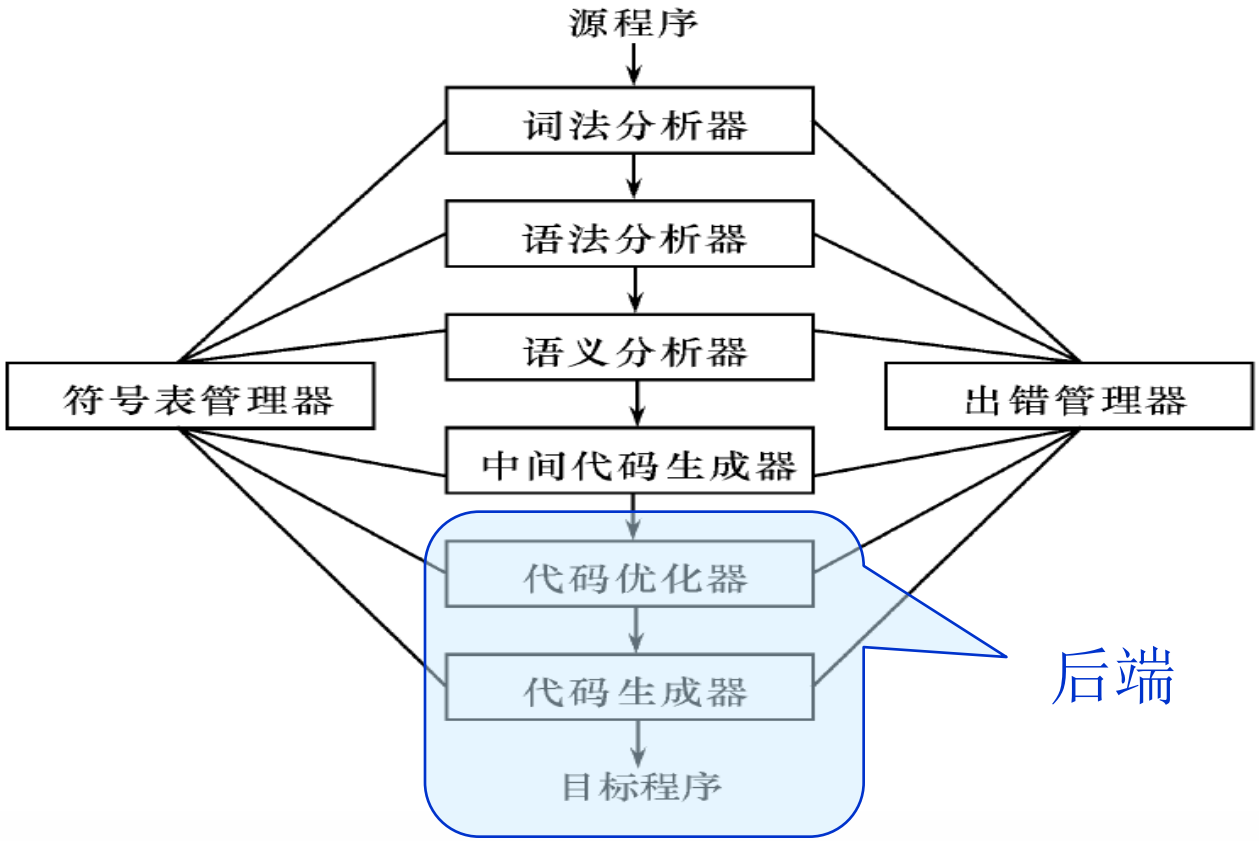
第一章：编译简介

- 阶段的分组



第一章：编译简介

- 阶段的分组



第一章：编译简介

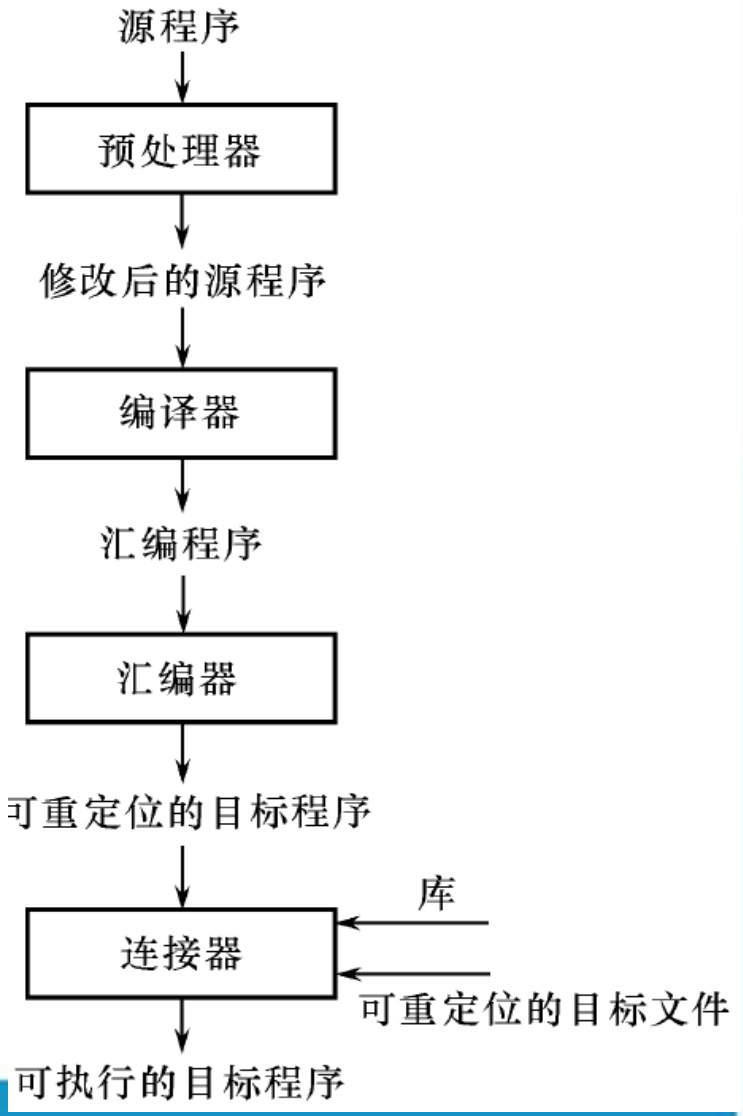
- 阶段的分组
  - 前端只依赖于源语言
  - 后端是编译器中依赖于目标机器的部分，它们一般独立于源语言，而与中间语言有关
  - **取一个编译器前端，重写它的后端以产生同一源语言在另一机器上的编译器**
  - 把几种不同的语言编译成同一种中间语言，让不同的前端使用同一后端，从而得到一台机器上的几个编译器
  - 编译的几个阶段常用一趟/遍（pass）扫描来实现，一趟/遍扫描包括读一个输入文件和写一个输出文件。

第一章：编译简介

- 编译系统
  - 除了编译器外，还需要一些其他工具的帮助，才能得到可执行的目标程序，这些工具包括预处理器、汇编器和连接器等
    - C语言的编译系统
    - Java语言的编译系统

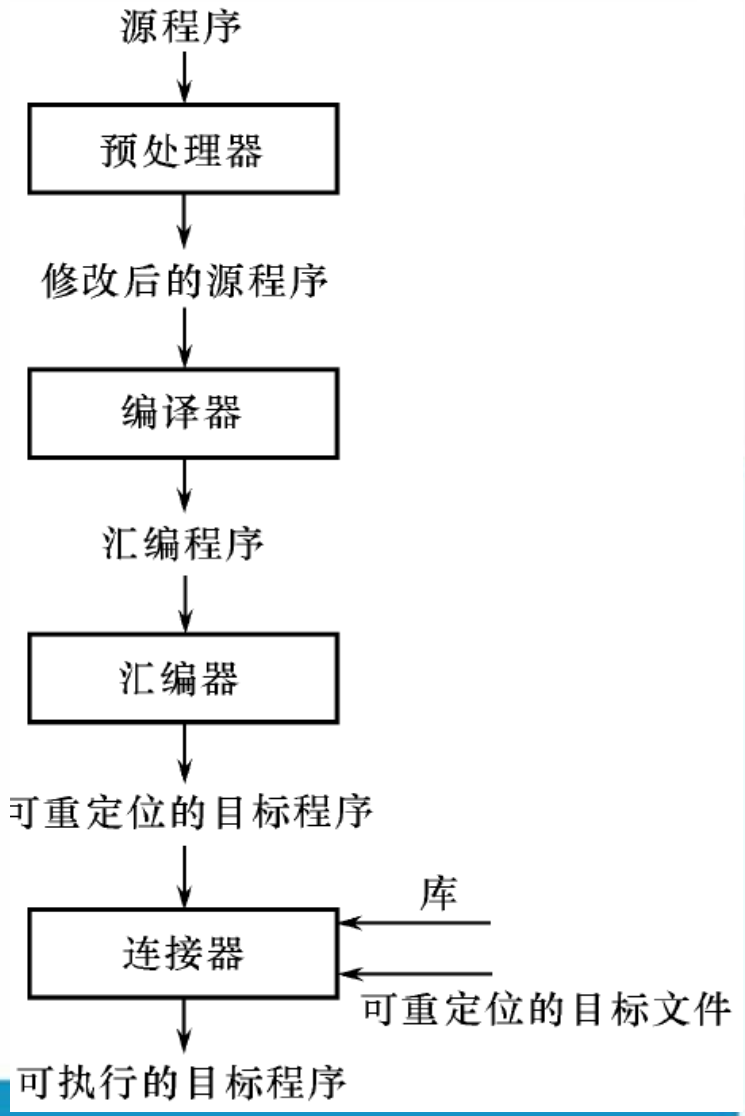
第一章：编译简介

- C语言的编译系统
  - 设有两个文件：
    - main.c
    - swap.c
  - gcc –v –o swap main.c swap.c
    - -v 可以输出该编译系统各步骤执行的命令和执行结果
    - -o 指示生成的可执行文件的名字



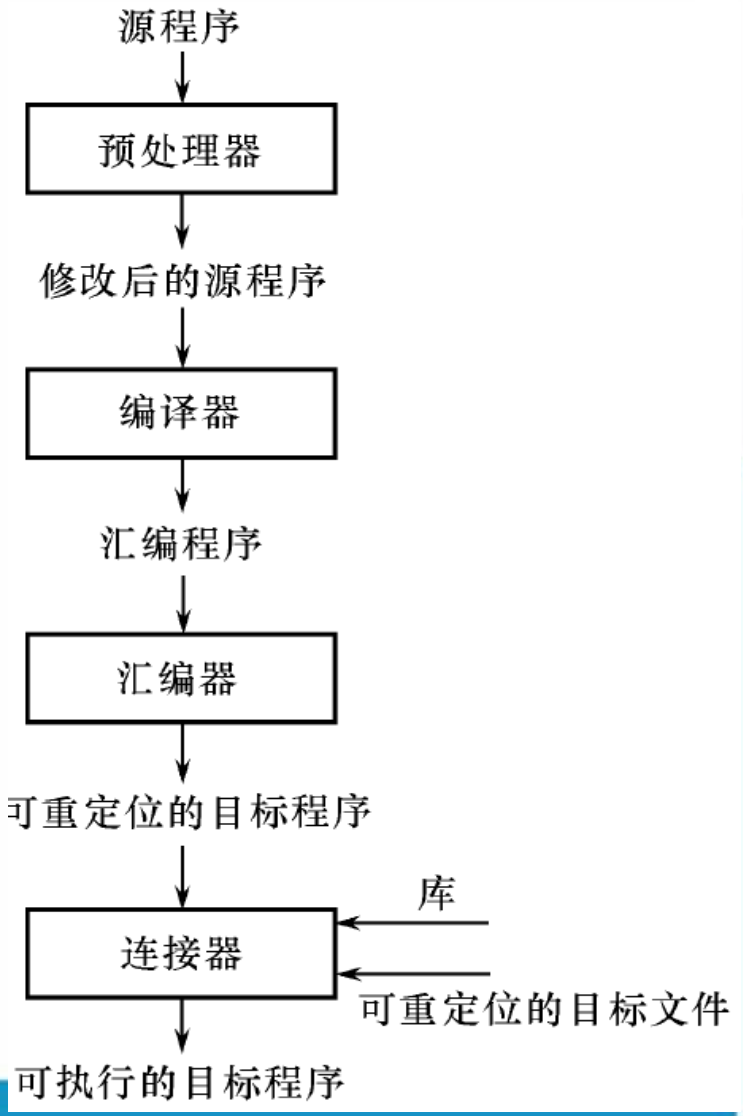
第一章：编译简介

- C语言的编译系统
  - 预处理器
    - 实现文件包含
      - #include <stdio.h>
    - 实现宏展开
      - #define pi 3.1415926
    - 条件编译
      - #if 、#ifdef



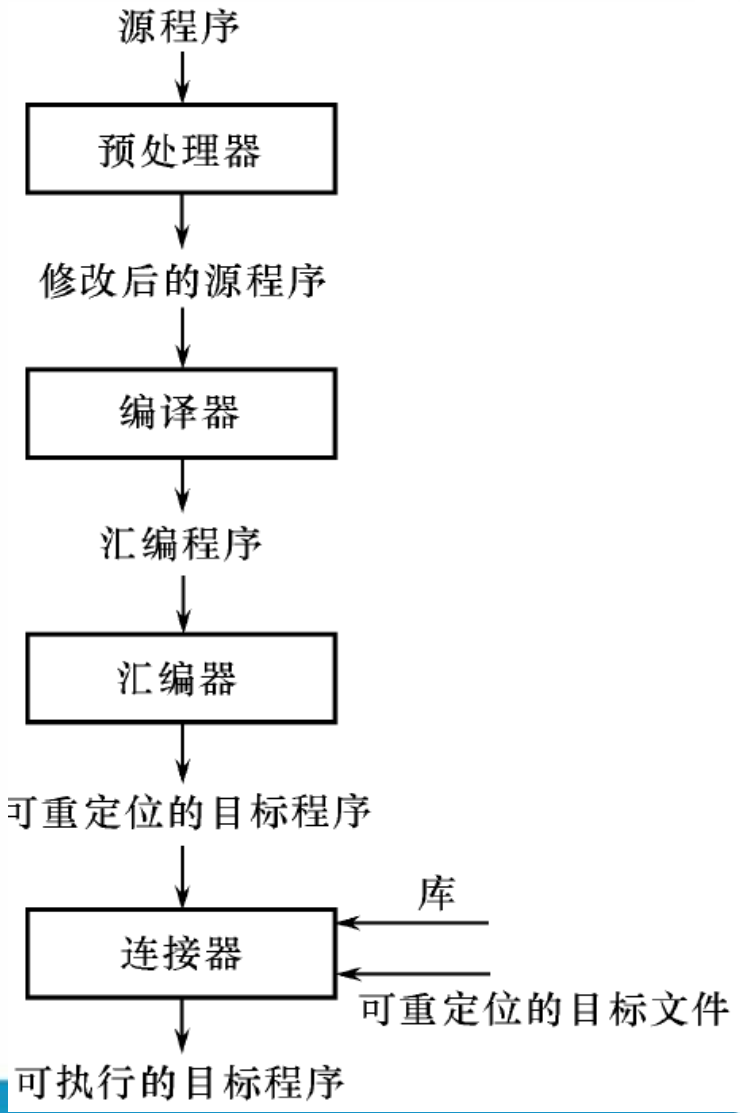
第一章：编译简介

- C语言的编译系统
  - 汇编器
    - 对输入进行两遍扫描：
      - 第一遍，汇编器扫描输入，将表示存储单元的所有标识符都存入符号表，并分配地址。
      - 第二遍，汇编器再次扫描输入，把每个操作码翻译成机器语言中代表那个操作的位串，并把代表存储单元的每个标识符翻译成符号表中为这个标识符分配的地址
    - gcc –S main.c
    - **as** –o main.o main.s



第一章：编译简介

- C语言的编译系统
  - 连接器
    - 是一个收集、组织程序所需的不同代码和数据的过程
      - 静态连接器
        - 将多个可重定位目标文件组成一个可执行目标文件（也可以组成一个可重定位目标文件）
      - 动态连接器
        - 支持在内存中的可执行程序在执行时与共享目标文件进行动态的连接。

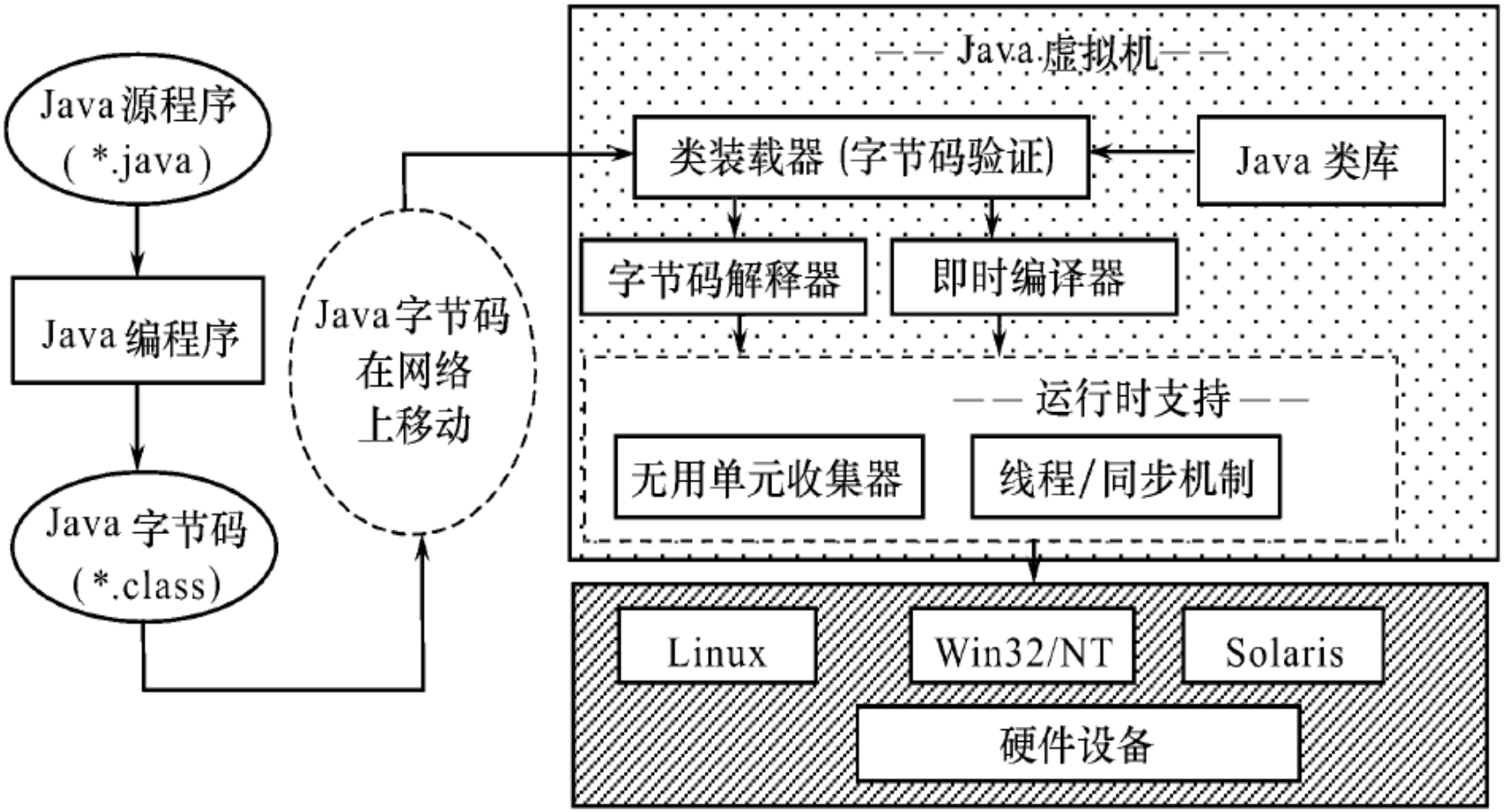


第一章：编译简介

- Java语言的编译系统
  - 一般的高级语言程序如果要在不同的平台上运行，至少需要编译成不同的目标代码。
  - Java虚拟机技术则是实现Java平台无关性特点的关键
  - Java虚拟机语言（简称JVML）
  - JVML程序只需要与虚拟机交互，不需要关心底层的硬件和操作系统。

第一章：编译简介

- Java语言的编译系统



第一章：编译简介

- 编译器发展历程
  - 编译器的编译器的编译器……？
  - 20世纪40年代：0、1序列组成的机器语言编程
  - 20世纪50年代早期：汇编语言出现
  - 20世纪50年代后5年：高级语言出现
    - Fortran、Cobol、Lisp
  - …

第一章：编译简介

- 编译器发展历程
  - C语言的产生
    - The Development of the C Language：C history – Written by Dennis Ritchie
    - BCPL → B语言 → New B语言 → C语言

Ken Thompson (left) with **Dennis Ritchie**

DEC PDP-7, as used for initial work on C and Unix

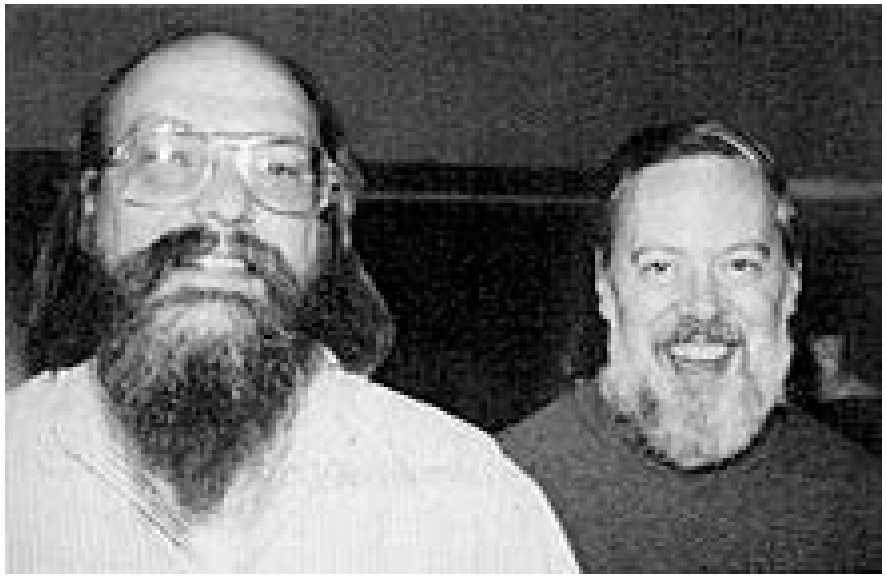
第一章：编译简介

- 第一届（1966年）图灵奖得主：Alan J．Perlis

- 贡献领域：高级程序设计技巧，编译器构造

第一章：编译简介

- 1983年图灵奖得主：Ken Thompson, Dennis Ritchie



- 贡献领域：C语言和Unix操作系统

第一章：编译简介

- 2006年图灵奖得主：Frances E. Allen

- 贡献领域：优化编译器

第一章：编译简介

- 主流编译理论会议
  - “ACM1Symposium on Programming Language Design and Implementation”（编程语言设计与实现，PLDI）
  - “ACM Symposium on Principles of Programming Languages”（编程语言原理，POPL）
  - “ACM Symposium on Principles and Practice of Parallel Programming”（并行编程原理与实践，PPoPP）
  - “ACM Conference on Object-Oriented Programming Systems,Languages and Applications”（面向对象的编程系统、语言和应用，OOPSLA）

第一章：编译简介

- 编译性语言、解释性语言和脚本语言
  - 高级语言翻译成机器语言，计算机才能执行高级语言编写的程序。
  - 翻译有两种方式：
    - 编译：一次性编译成机器语言文件，不用重新编译，效率高
    - 解释：每个语句都是执行的时候才翻译，每执行一次就翻译一次，效率比较低
  - 脚本语言是一种解释性的语言
    - JavaScript，ASP,PHP,PERL
  - Java语言
    - 既要编译，又要解释；编译只有一次，程序执行时解释执行；通过编译器，把java程序翻译成一种中间代码——字节码，然后通过JVM解释成相应平台的语言

第一章：编译简介

- 自然语言处理与编译原理相关展望
  - 词法分析
    - 正向最大匹配法从左向右匹配词典
    - 逆向最大匹配法从右向左匹配词典
      - 例子
        - 输入:企业要真正具有用工的自主权
        - MM:企业/要/真正/具有/用工/的/自主/权
        - RMM:企业/要/真正/具有/用工/的/自/主权

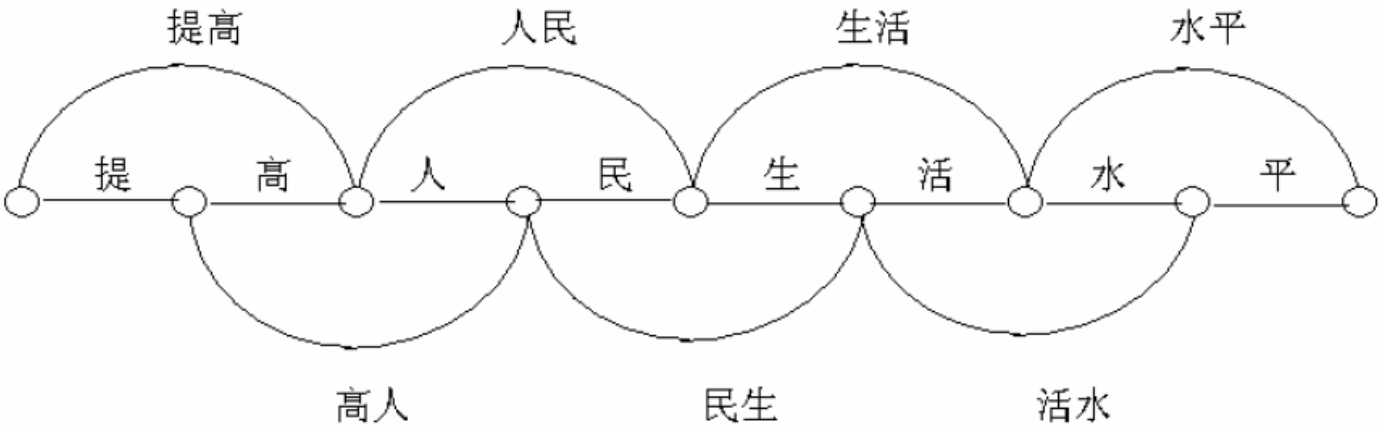
第一章：编译简介

- 自然语言处理与编译原理相关展望
  - 词法分析
    - 正向最大匹配法从左向右匹配词典

```
S←待切分的字串;
Segmentation←"";
len ← maxlen;
WHILE S≠"" DO
        W ← substr(S,0,len);
        IF (W∈D) THEN /*D 为电子词典*/
            S ← S – W;
            Segmentation ← Segmentation + W + "/";
            len ← maxlen;
        ELSE
            IF len = 1 THEN
                S ← S – W;
                Segmentation ← Segmentation + W + "/";
                len ← maxlen;
            ELSE
                len ← len – 1;
            ENDIF
        ENDIF
END WHILE
```

第一章：编译简介

- 自然语言处理与编译原理相关展望
  - 词法分析



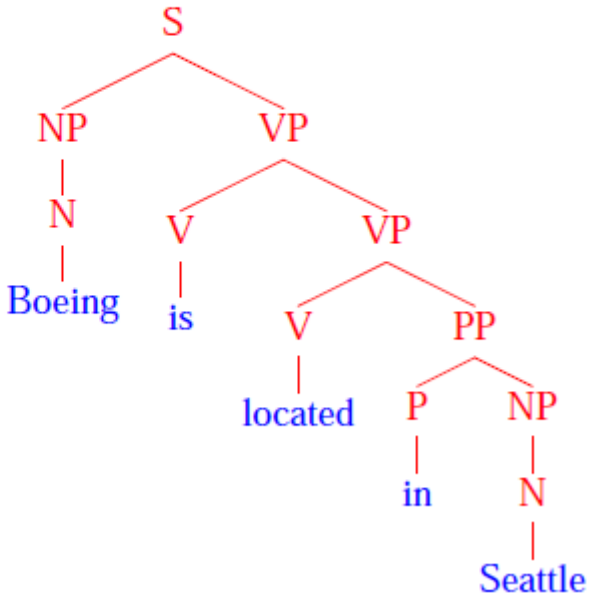
第一章：编译简介

- 自然语言处理与编译原理相关展望
  - 语法分析

INPUT:

Boeing is located in Seattle.

OUTPUT:



$N = \{S, NP, VP, PP, DT, Vi, Vt, NN, IN\}$

$S = S$

$\Sigma = \{sleeps, saw, man, woman, telescope, the, with, in\}$

$R =$

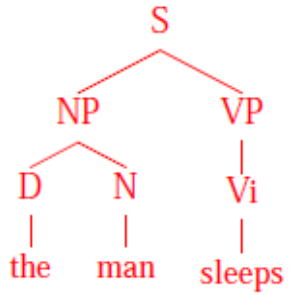
S	⇒	NP	VP
VP	⇒	Vi	
VP	⇒	Vt	NP
VP	⇒	VP	PP
NP	⇒	DT	NN
NP	⇒	NP	PP
PP	⇒	IN	NP

Vi	⇒	sleeps
Vt	⇒	saw
NN	⇒	man
NN	⇒	woman
NN	⇒	telescope
DT	⇒	the
IN	⇒	with
IN	⇒	in

Note: S=sentence, VP=verb phrase, NP=noun phrase, PP=prepositional phrase, DT=determiner, Vi=intransitive verb, Vt=transitive verb, NN=noun, IN=preposition

For example: [S], [NP VP], [D N VP], [the N VP], [the man VP], [the man Vi], [the man sleeps]

Representation of a derivation as a tree:



第一章：编译简介

- 习题
  - 解释下列名词：

    源语言　目标语言　翻译器　编译器　解释器

  - 典型的编译器可以划分成哪几个主要的逻辑阶段，各阶段的主要功能是什么？