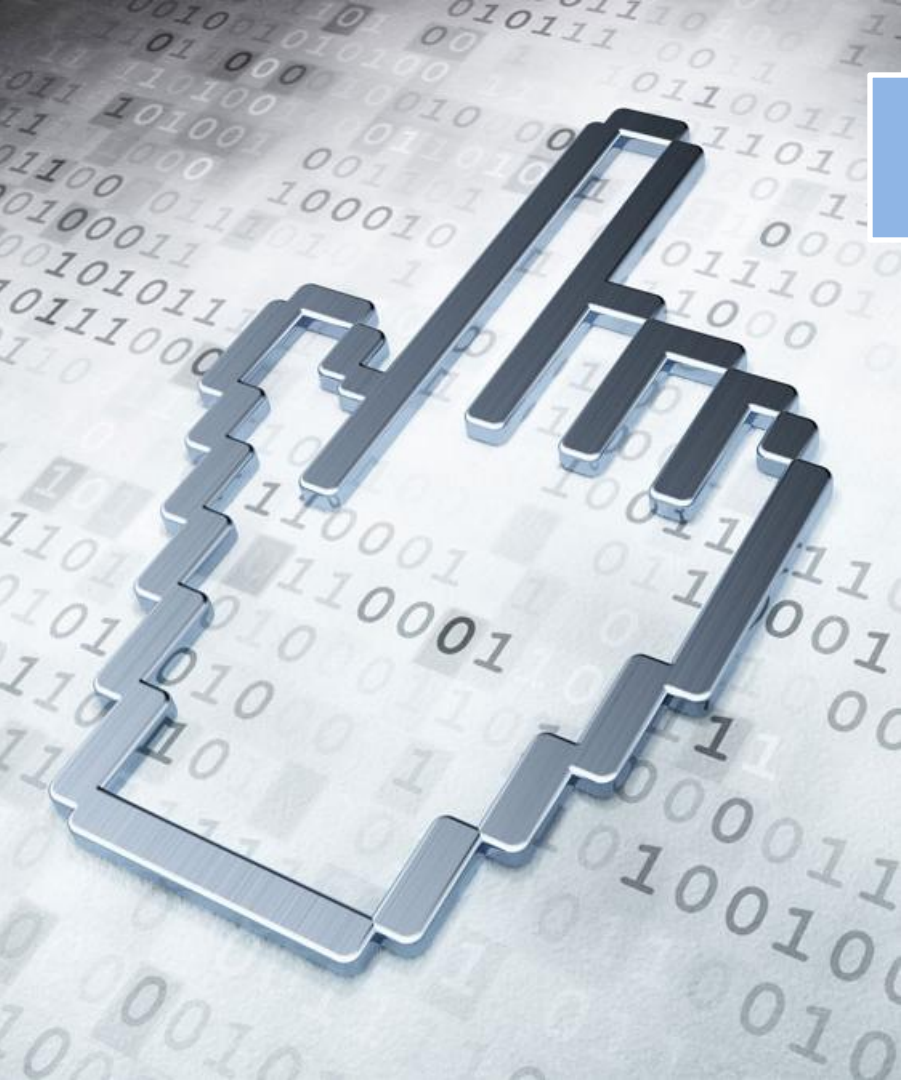
编译原理

第一章 绪论

哈尔滨工业大学 陈鄞 单丽莉



本章内容

1.1 什么是编译？

类似于数学定义或
自然语言的简洁形式

高级语言
(High Level Language)

- 接近人类表达习惯
- 不依赖于特定机器
- 编写效率高

引入助记符

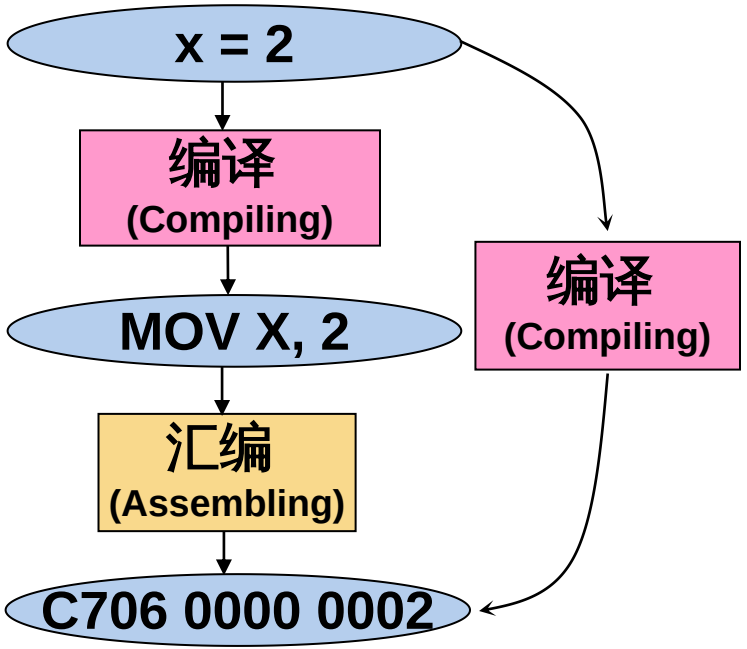
汇编语言
(Assembly Language)

- 依赖于特定机器，非计算机专业人员使用受限制
- 编写效率依然很低

可以被计算机直接执行

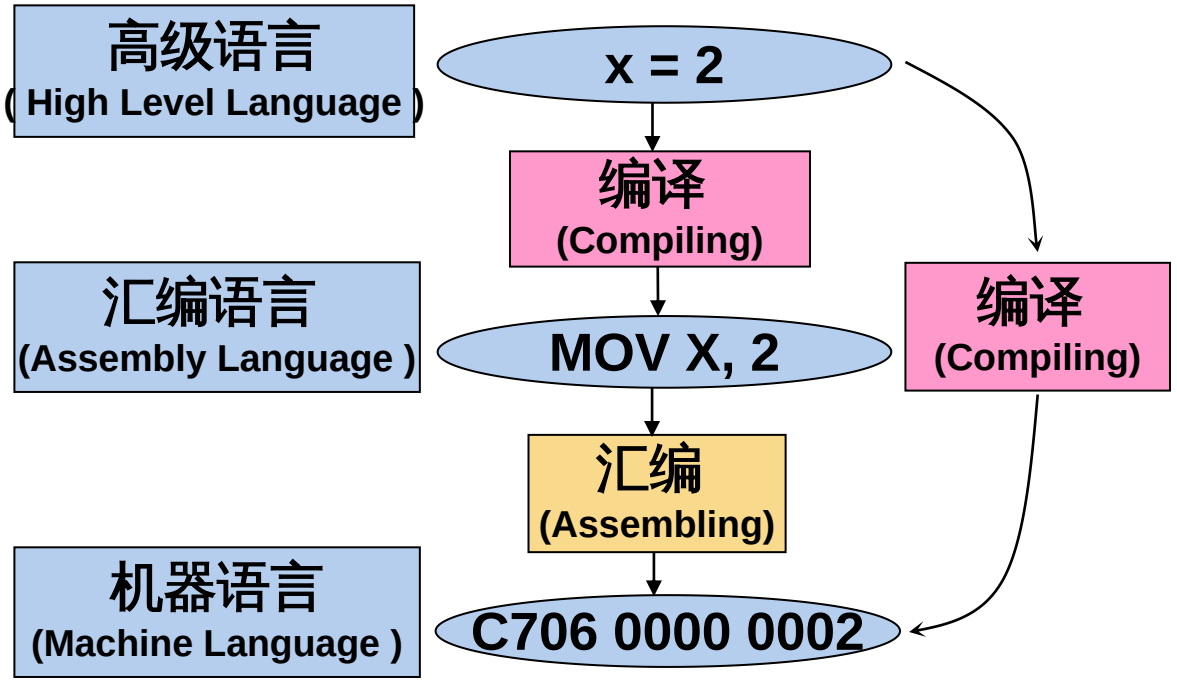
机器语言
(Machine Language)

- 与人类表达习惯相去甚远
- 难记忆
- 难编写、难阅读
- 易写错



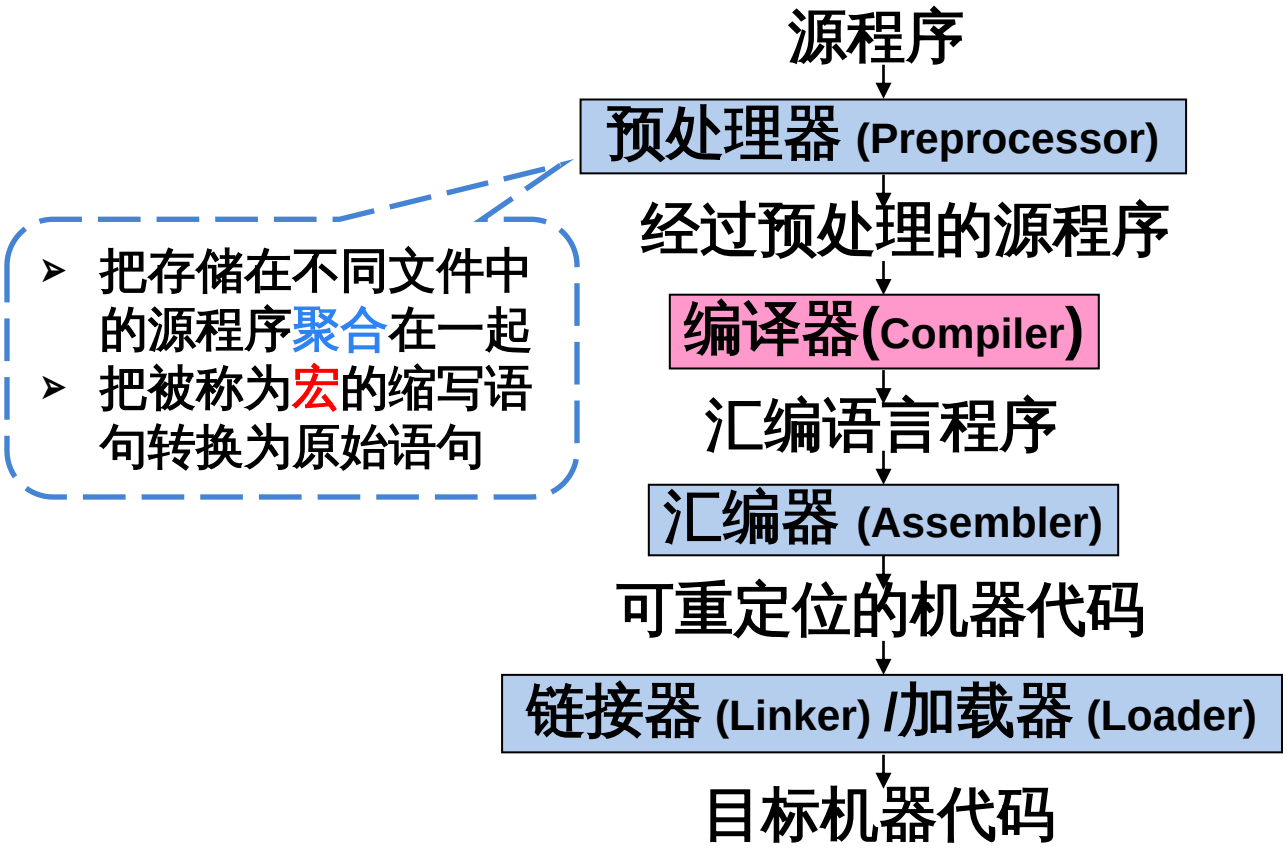
1.1 什么是编译？

编译的主要任务：翻译 和 优化
优化目标：代码规模小、执行速度快

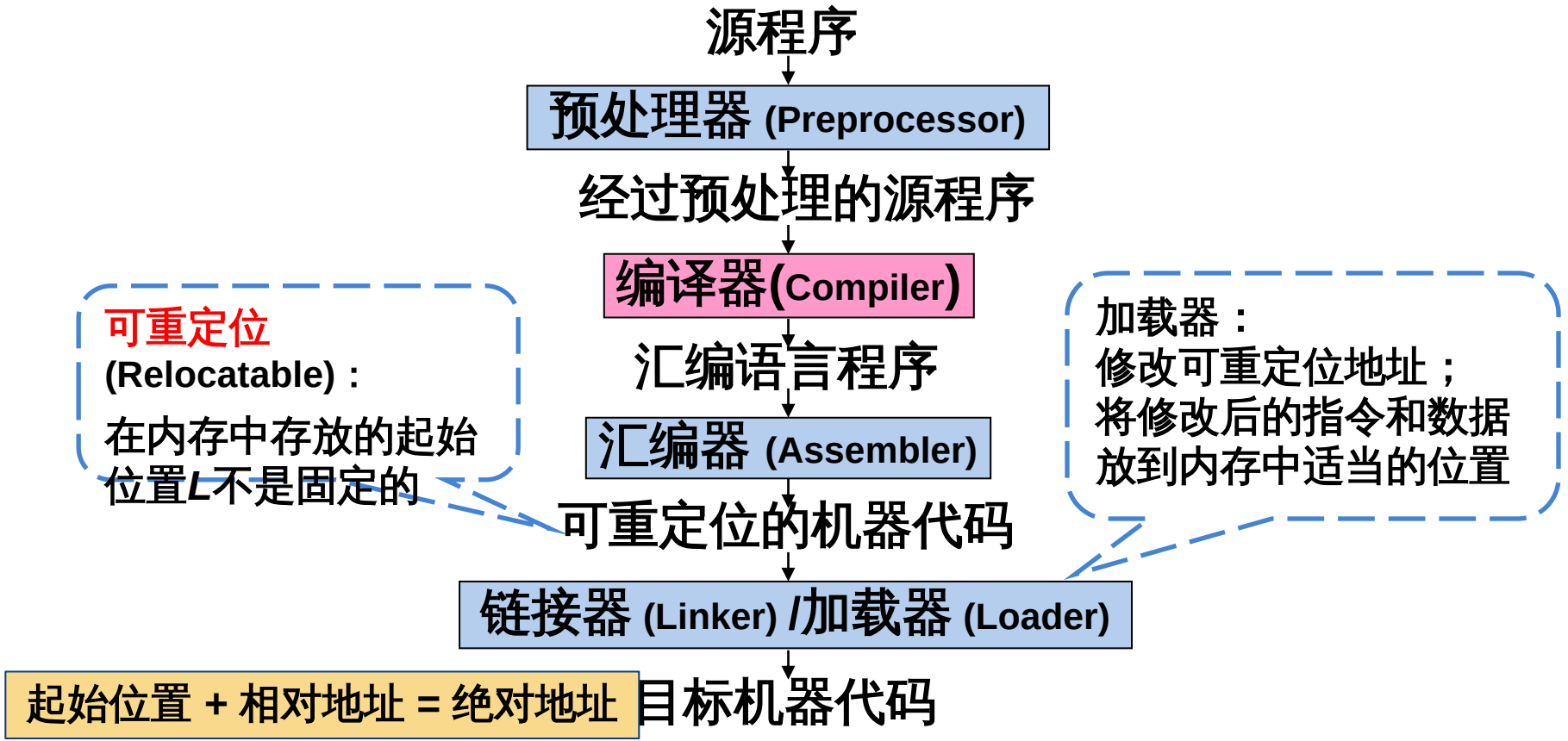


编译：将高级语言翻译成汇编语言或机器语言的过程
源语言　　目标语言

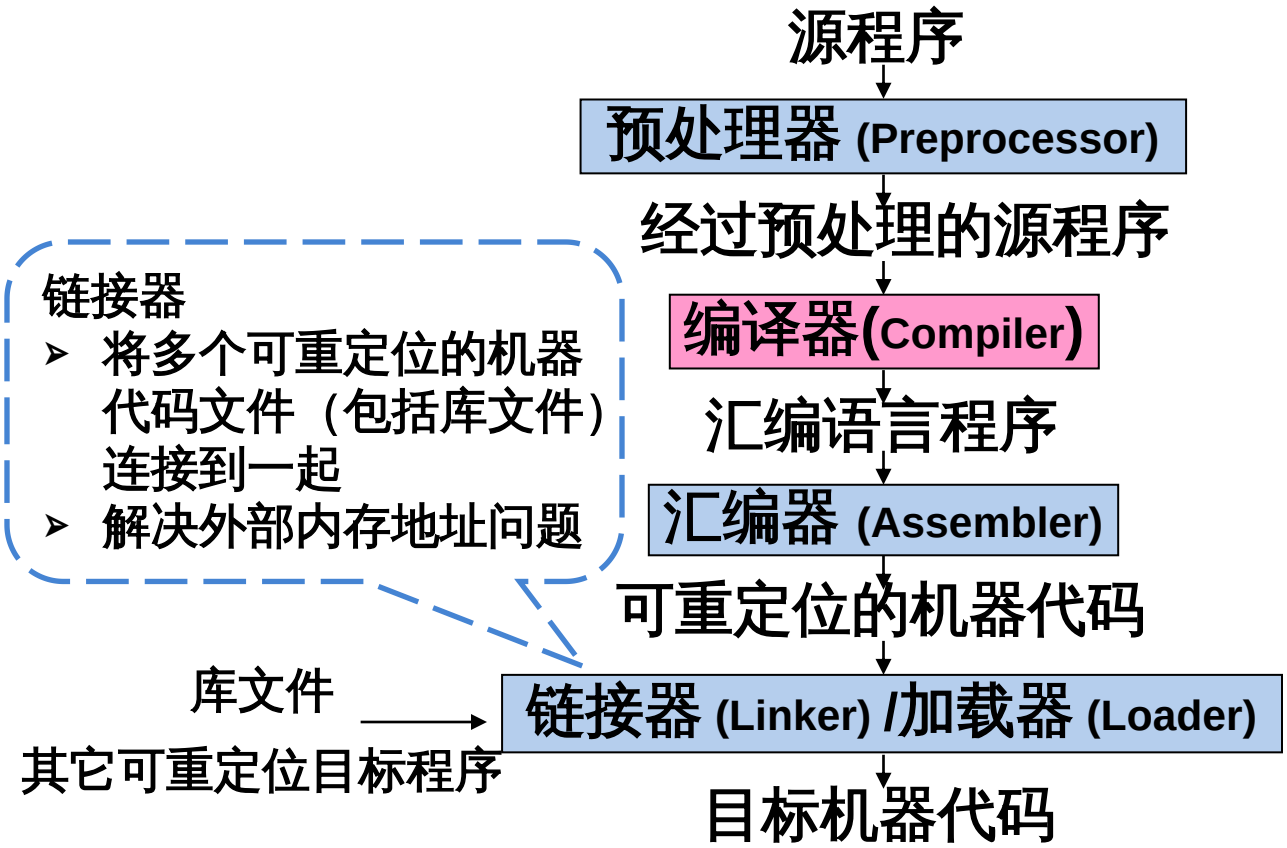
编译器在语言处理系统中的位置

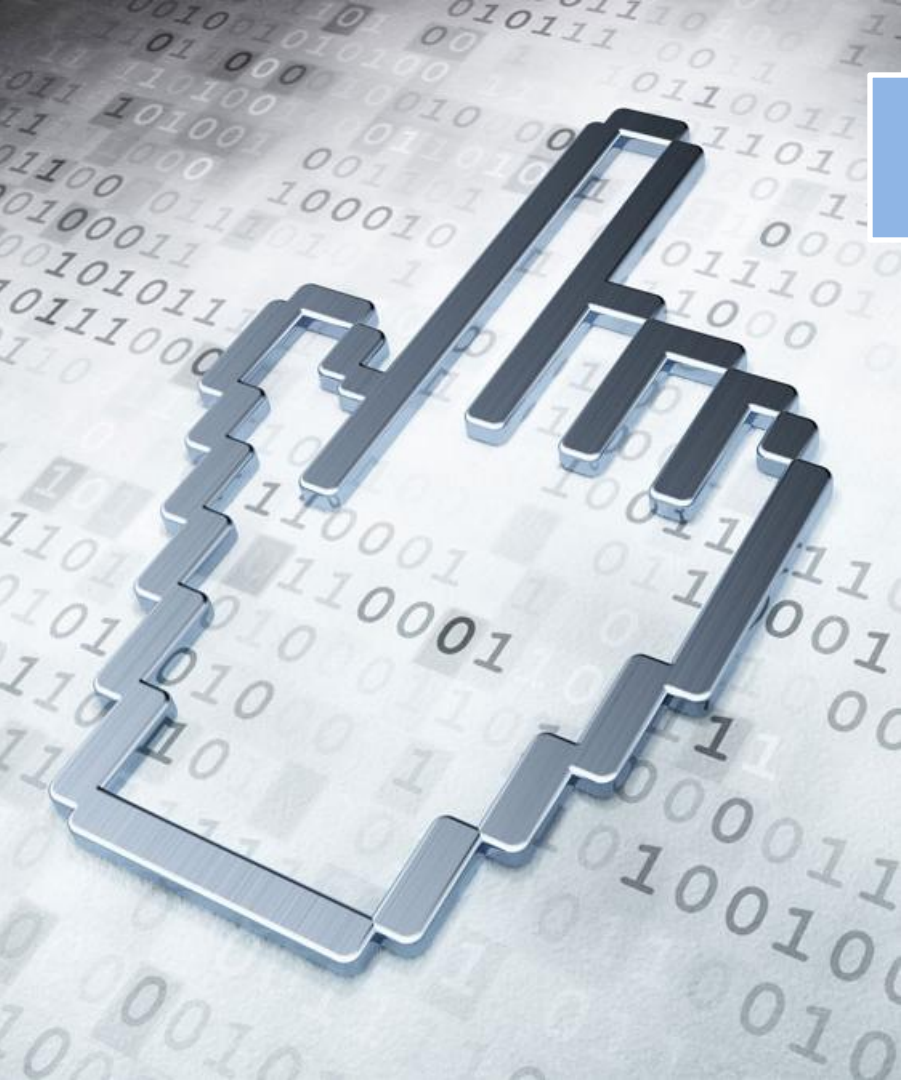


编译器在语言处理系统中的位置



编译器在语言处理系统中的位置



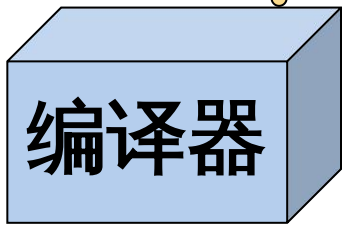


本章内容

1.2 编译系统的结构

```
#include<stdio.h>
int main()
{
    int a,b,ge,shi,bai,m,n,i,number;
    printf("请输入(输入完两个数按一次回车键): \n");
    scanf("%d %d",&a,&b);
    while(a!=0,b!=0)
        scanf("%d %d",&a,&b);
    if(a>=100,b<=999)
        if(a>b)
            m=b,n=a;
        else
            m=a,n=b;
        for(i=m;i<=n;i
        {
            bai=i/100;
            shi=(i%100)/10;
            ge=i%10;
            if(i==bai*bai*bai+shi*shi*shi+ge*ge*ge)
            printf("%5d",i);
            number++;
            printf("\n");
        }
        if(number==0)
            printf("no\n");
}
```

高级语言程序



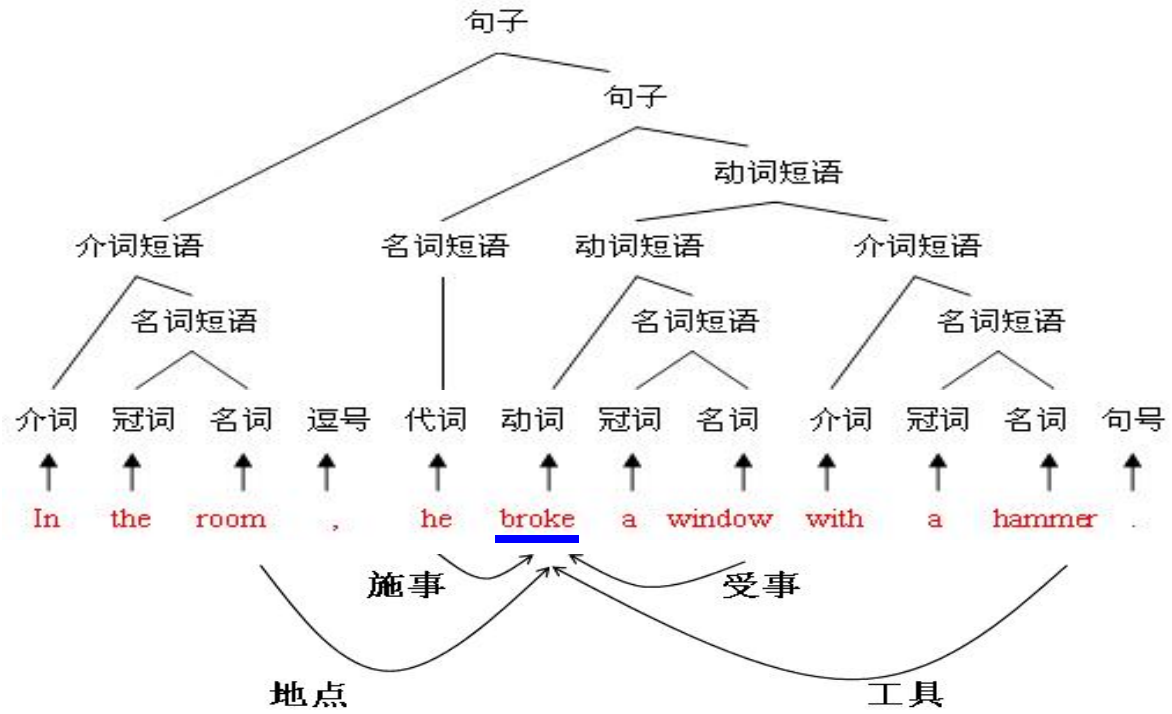

```
59      call sc
60      mov  al, [si - 1]
61      call sc
62      mov  al, [si]
63      call sc
64      mov  al, ' '
65      call sc
66      ret
67  storechr endp
68  ;清屏，无入口参数
69  clearcrt proc near
70      mov  ax, 0600h
71      mov  bh, 07h
72      mov  cx, 00h
73      mov  dx, 184fh
74      int  10h
                                    no
                                    rd
84        mov  ah, 02h
85        mov  dl, cr
86        int  21h
87        mov  ah, 02h
88        mov  dl, lf
89        int  21h
90      .untilcxz
91      ret
92  nextlien endp
93          end
94
```

汇编语言程序/机器语言程序

机器是如何自动翻译的？

人工英汉翻译的例子

In the room , he broke a window with a hammer.



初步翻译

在房间里，他砸了一扇窗用锤子。

润色

他在房间里用锤子砸了一扇窗。

1.2 编译系统的结构

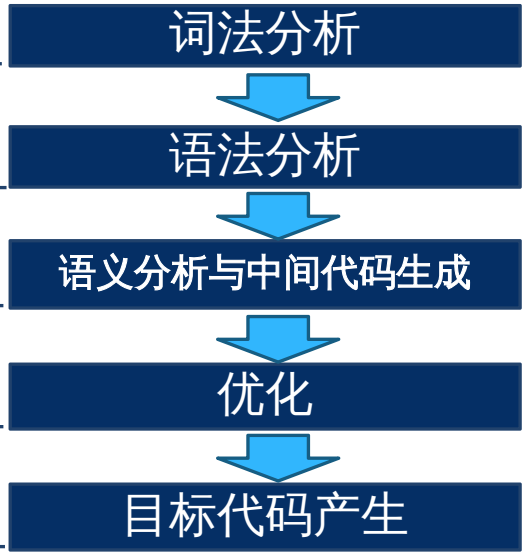
In the room , he broke a window with a hammer.

他在房间里用锤子咂了一扇窗。

把英文翻译为中文

1. 识别出文本中的一个个单词
2. 分析句子的语法结构
3. 根据句子的含义进行初步翻译
4. 对译文进行润色
5. 写出最后的译文

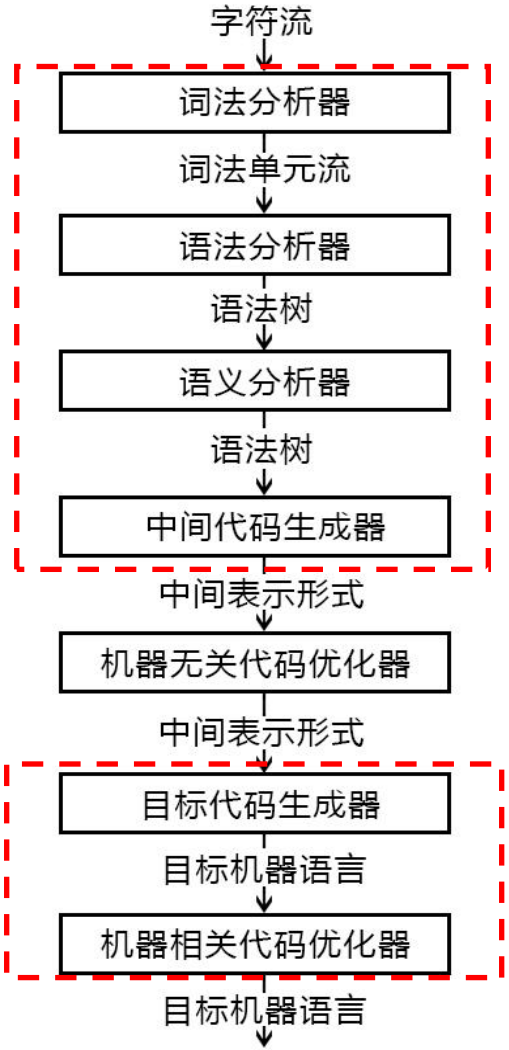
编译程序工作阶段



编译器的结构

分析部分/
前端(front end)：
与源语言相关
与目标语言无关

综合部分/
后端(back end)：
与目标语言相关



符号表

编译器的结构


字符流
词法分析器
词法单元流
语法分析器
语法树
语义分析器
语法树
中间代码生成器
中间表示形式
机器无关代码优化器
中间表示形式
目标代码生成器
目标机器语言
机器相关代码优化器
目标机器语言

符号表

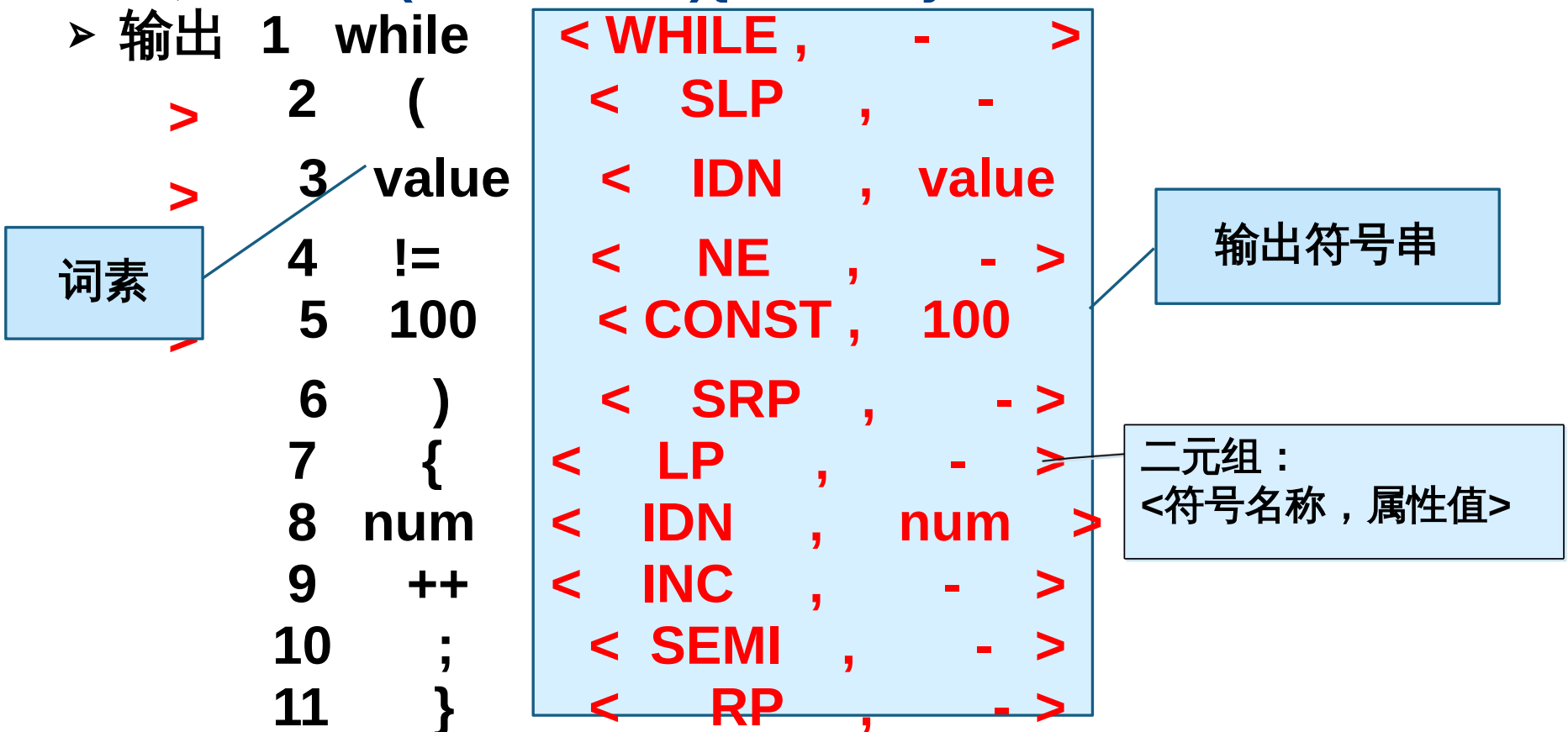
词法分析/扫描(*Scanning*)

- 词法分析由词法分析器(Lexical Analyzer)完成，又称为扫描器Scanner 。
- 任务：
  - 从左到右扫描源程序字符串，识别出一个个单词，并转换成符号表示的单词(token)串；同时要检查词法错误。
- 输入: 源程序字符串(字符流)
- 输出: 单词串（token序列，词法单元序列，符号串）
  - 每个单词(词法单元)：

    二元组序对<符号名称，属性值>

例：词法分析后得到的符号表示的单词串

- 输入 while(value!=100){num++;}
- 输出



词法分析/扫描(*Scanning*)

- 词法分析的重要准备　　token：< 符号名称，属性值 >

  设计代码中的单词与 符号名称(token) 的 编码关系

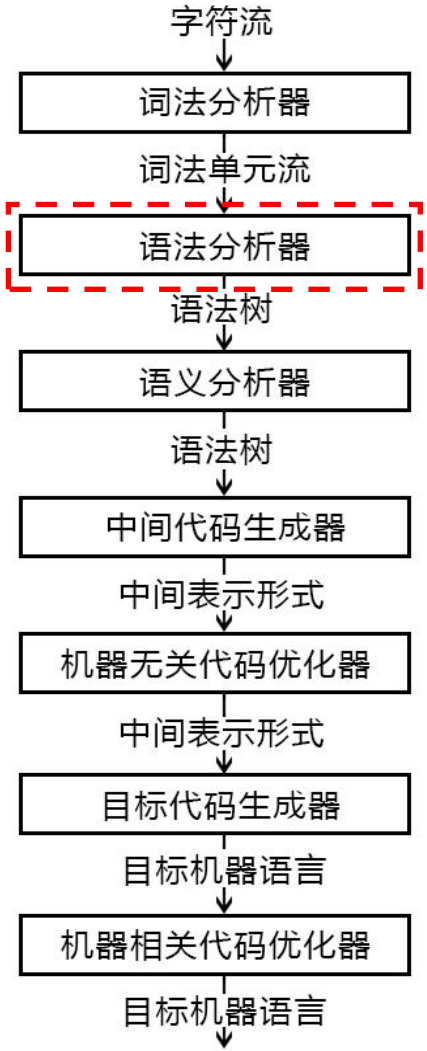
  基本依据：编码为语法分析器能处理的符号。

	单词类型	词素类别	符号名称
1	关键字	program、if、else、then、…	一词一符
2	标识符	变量名、数组名、记录名、过程名、…	一类一符
3	常量	整型、浮点型、字符型、布尔型、…	一型一符
4	运算符	算术（+ - * / ++ --） 关系（> < == != >= <=） 逻辑（&& || ！）	一词一符 或 一型一符
5	界限符	; ( ) = { } …	一词一符

词法规则可以使用正则文法来形式化描述，设计DFA来识别单词。

编译器的结构

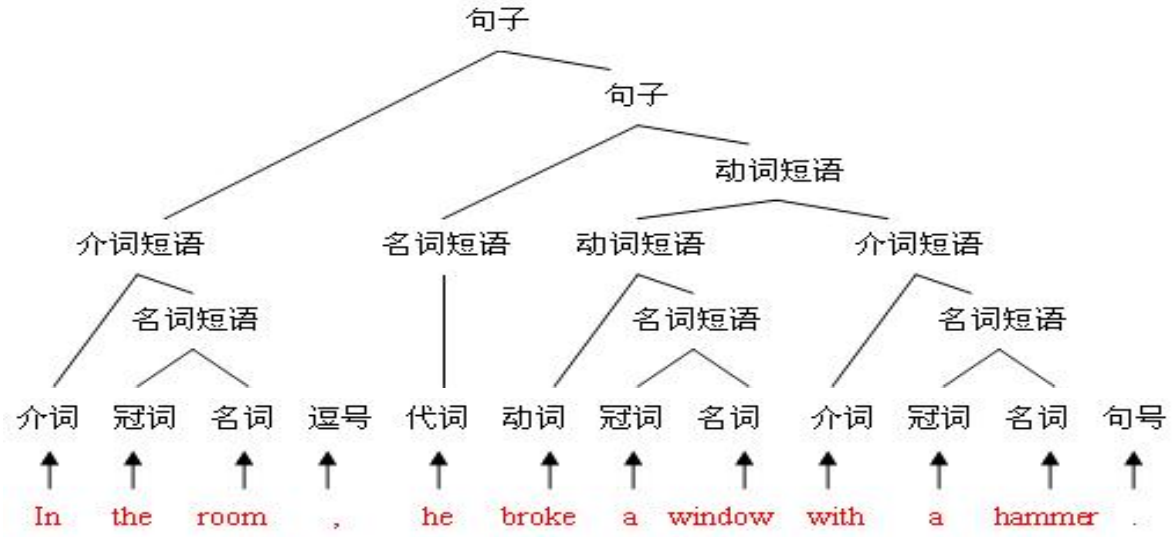
- 语法分析器：
  - 功能：组词成句
  - 输入：词法单元序列
  - 输出：语法分析树



符号表

语法分析（*parsing*）

- 语法分析器(parser)从词法分析器输出的token序列中识别出各类短语，并构造语法分析树(parse tree)
  - 语法分析树描述了句子的语法结构



任务：组词成句

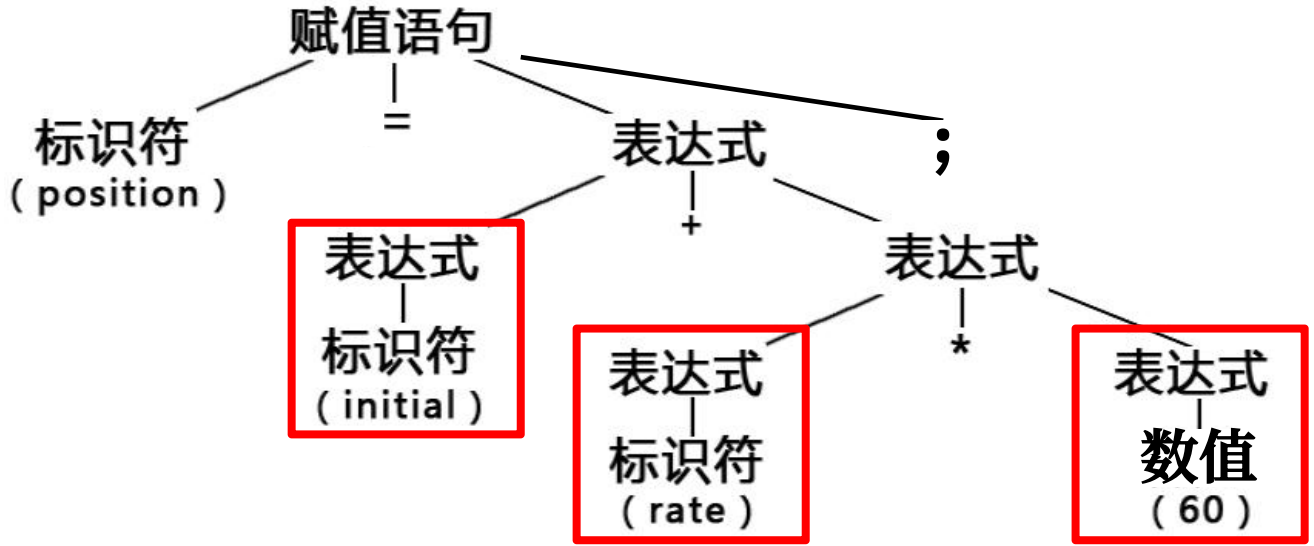
输入：单词串（token序列）

输出：语法成分（语法分析树）

例1：赋值语句的分析树

position = initial + rate * 60 ;

<标识符, position> <=> <标识符, initial> <+> <标识符, rate> <*> <数值, 60> <;>



例2：变量声明语句的分析树

- 文法：

  <*D*> → <*T*> <*IDS*>;

  <*T*> → int | real | char | bool

  <*IDS*> → id | <*IDS*>, id

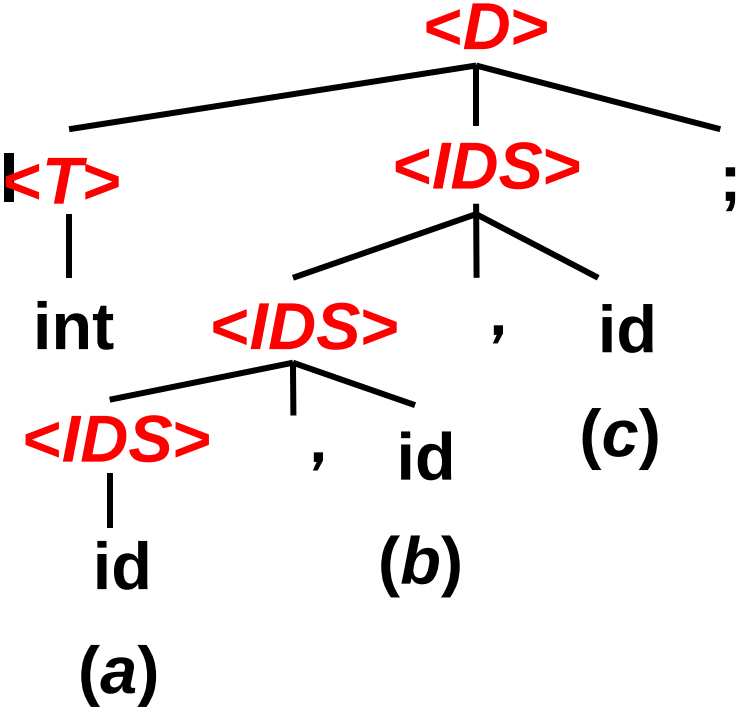
- 源代码：

  int *a* , *b* , *c* ;

- 词法分析器输出

  <int><id, a><,><id, b><,>

  <id, c><;>



语法分析依赖语言的文法，通常是上下文无关文法。

单选题 1分

设置

语法分析器的任务是：

A 分析单词是怎样构成的

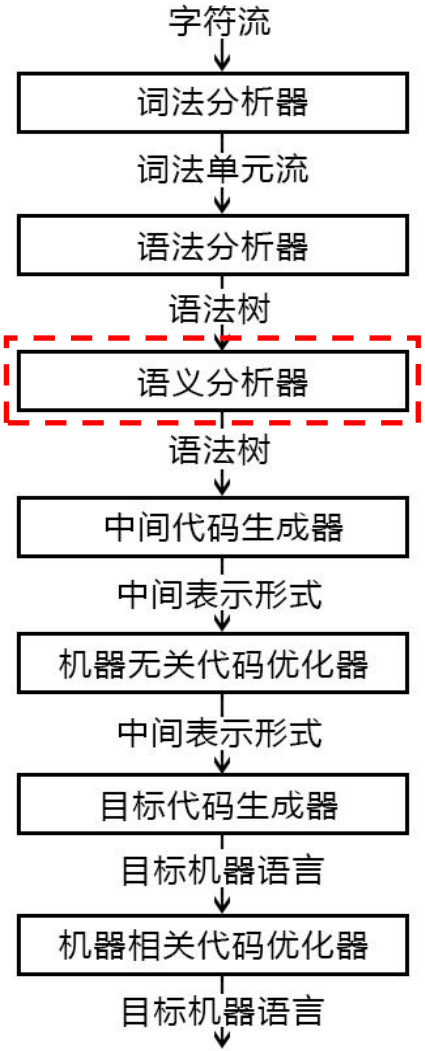
B 分析单词串是如何构成语句、声明和程序的

C 分析语句和声明是如何构成程序的

D 分析程序的结构

提交

编译器的结构


字符流
词法分析器
词法单元流
语法分析器
语法树
语义分析器
语法树
中间代码生成器
中间表示形式
机器无关代码优化器
中间表示形式
目标代码生成器
目标机器语言
机器相关代码优化器
目标机器语言

符号表

语义分析的主要任务

- 收集标识符的属性信息
  - 种属 (Kind)
    - 简单变量、复合变量（数组、记录、...）、过程、...

语义分析的主要任务

- 收集标识符的属性信息
  - 种属 (Kind)
  - 类型 (Type)
    - 整型、实型、字符型、布尔型、指针型、…

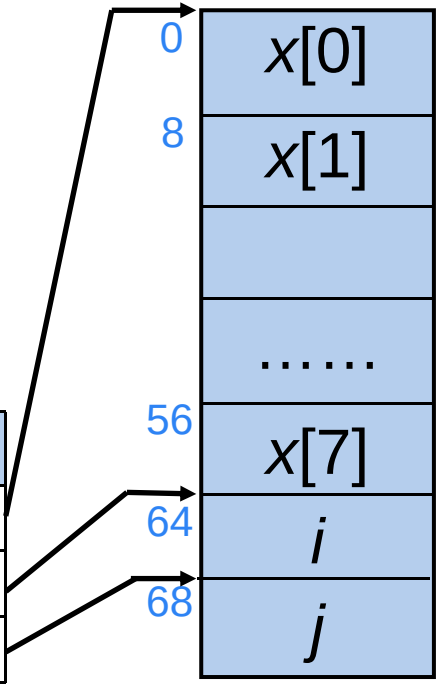
语义分析的主要任务

- 收集标识符的属性信息
  - 种属 (Kind)
  - 类型 (Type)
  - 存储位置、长度

例：

```
begin
    real  x[8];
    integer  i,
    j;
    ……
end
```

名字	相对地址
x	0
i	64
j	68



语义分析的主要任务

- 收集标识符的属性信息
  - 种属 (Kind)
  - 类型 (Type)
  - 存储位置、长度
  - 值
  - 作用域
  - 参数和返回值信息
    - 参数个数、参数类型、参数传递方式、返回值类型、…

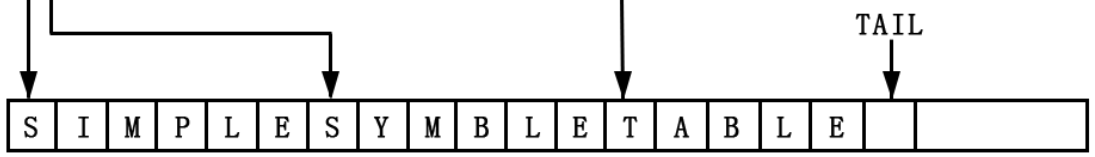
语义分析的主要任务

- 收集标识符的属性信息
  - 种属 (Kind)
  - 类型 (Type)
  - 存储位置、长度
  - 值
  - 作用域
  - 参数和返回值信息

符号表中为什么要设计字符串表这样一种数据结构？

符号表(Symbol Table)

NAME	TYPE	KIND	VAL	ADDR
6	整	简变		
6	实	数组		
5	字符	常数		
⋮	⋮	⋮	⋮	⋮



符号表是用于存放标识符的属性信息的数据结构

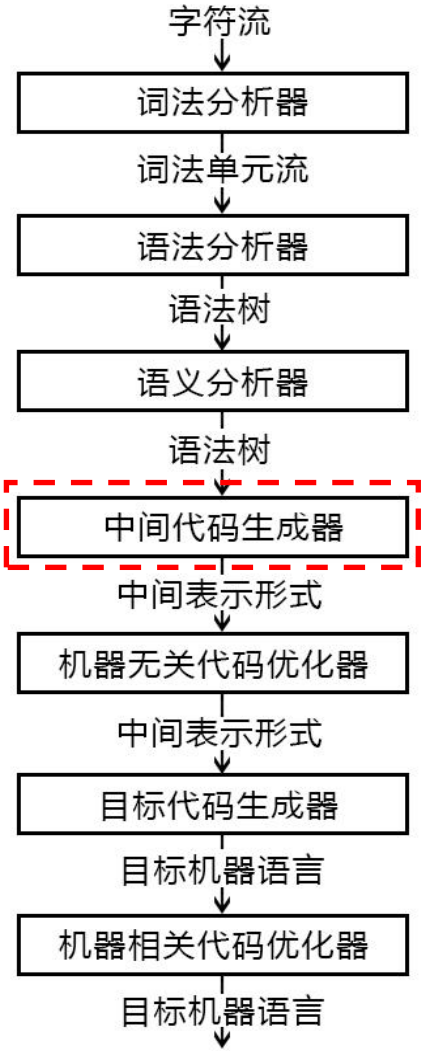
语义分析的主要任务

- 收集标识符的属性信息
- 语义检查
  - 变量或过程未经声明就使用
  - 变量或过程名重复声明
  - 运算分量类型不匹配
  - 操作符与操作数之间的类型不匹配
    - 数组下标不是整数
    - 对非数组变量使用数组访问操作符
    - 对非过程名使用过程调用操作符
    - 过程调用的参数类型或数目不匹配
    - 函数返回类型有误

上下文无关文法无法实现的上下文有关的语言特性都需要进行语义检查。

编译器的结构

- 中间代码的特点
  - 简单规范
  - 易于优化与转换
  - 与机器无关
  - 易于移植



符号表

常用的中间表示形式

- **线性**：三地址码 (Three-address Code)
  - 三地址码由类似于汇编语言的指令序列组成，每个指令最多有三个操作数(operand)
- **树型（图）**：语法结构树/语法树 (Syntax Trees)
- **混合型**：数据流图

常用的三地址指令

序号	指令类型	指令形式
1	赋值指令	*x* = *y* op *z* *x* = op *y*
2	复制指令	*x* = *y*
3	条件跳转	if *x* relop *y* goto *n*
4	非条件跳转	goto *n*
5	参数传递	param *x*
6	过程调用 函数调用	call *p*, *n* *y* =call *p*, *n*
7	过程返回	return *x*
8	数组引用	*x* = *y*[*i*]

三地址指令：一个运算符，三个地址；
三地址: (最多)两个运算分量地址，
一个结果分量地址。

地址可以具有如下形式之一

- 源程序中的名字 (*name*)
- 常量 (*constant*)
- 编译器生成的临时变量(*temporary*)

三地址指令的四元式表示

- $x = y\ op\ z$ (op , y , z , x)
- $x = op\ y$ (op , y , _ , x)
- $x = y$ (= , y , _ , x)
- if x relop y goto n (relop , x , y , n)
- goto n (goto , _ , _ , n)
- param x (param, _ , _ , x)

  y =call p, n (call , p , n , y)

  return x (return, _ , _ , x)
- $x = y[i]$ (=[] , y , i , x)

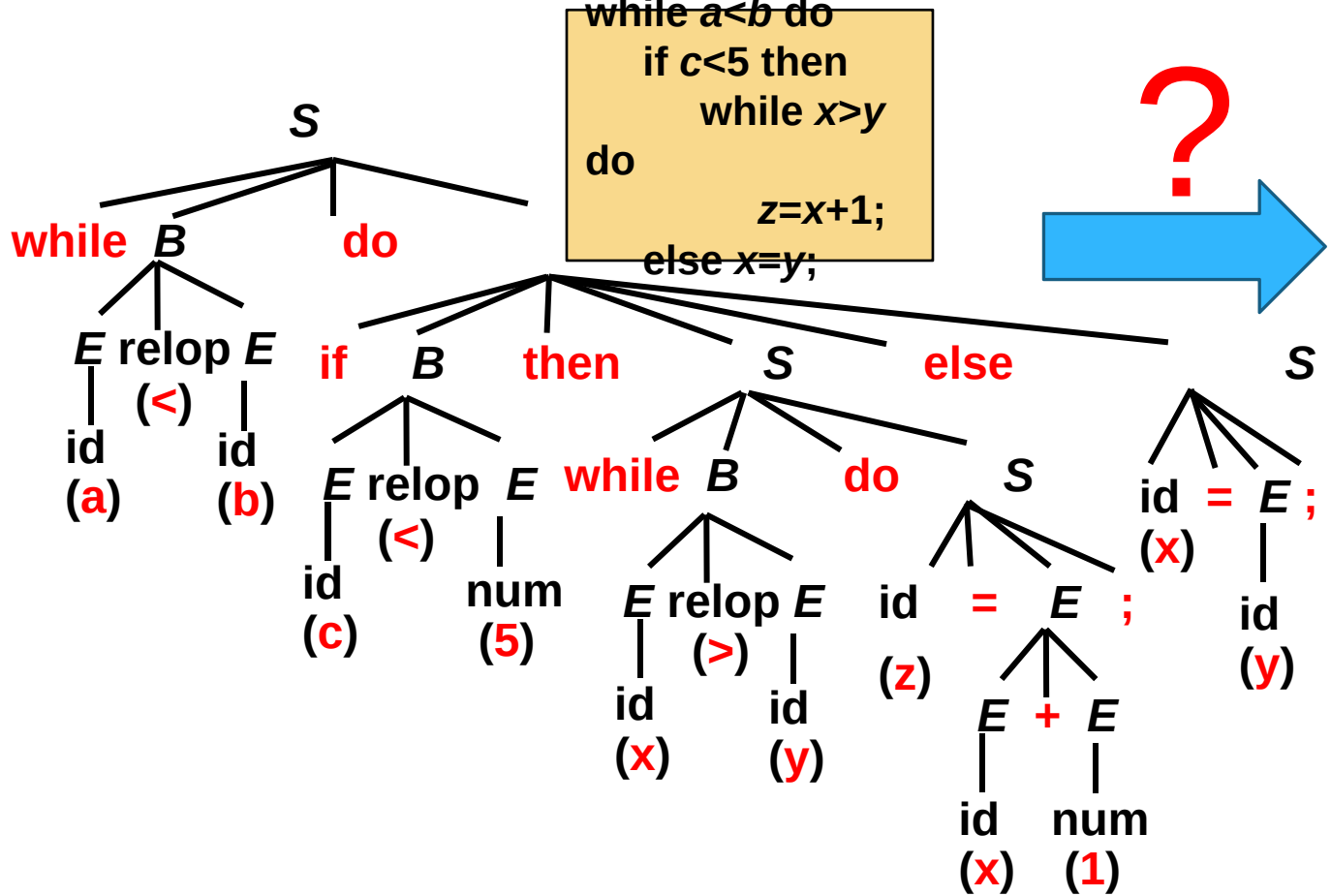
  $x[i] = y$ ([]= , y , x , i)
- $x = \&y$ (& , y , _ , x)

三地址指令序列唯一确定了运算完成的顺序

- 第一个分量是操作符
- 第二、三个分量是操作数
- 第四个分量是结果地址

中间代码生成的例子

```
while a<b do
    if c<5 then
        while x>y
do
            z=x+1;
    else x=y;
```



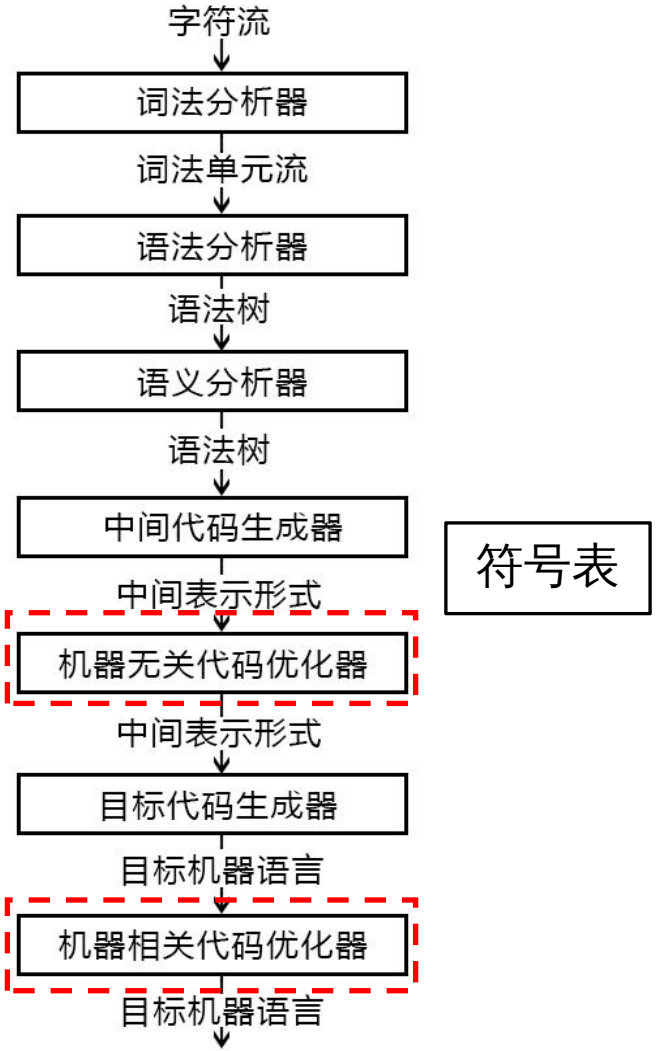
?

```
100: ( j<, a ,b ,
     102 )
101: ( j  , - ,  - ,
     112 )
102: ( j<, c , 5 ,
     104 )
103: ( j  , - ,  - ,
     110 )
104: ( j>, x , y ,
     106 )
105: ( j  , - ,  - ,
     100 )
106: ( + , x , 1 ,
     t1  )
107: ( = , t1 , - ,
```

编译器的结构

- 代码优化
  - 为改进代码所进行的等价程序变换，使其运行得更快一些、占用空间更少一些，或者二者兼顾
  - 最基本的优化原则：

    语义等价
- 优化分类
  - 机器无关优化
  - 机器相关优化



与机器无关的优化

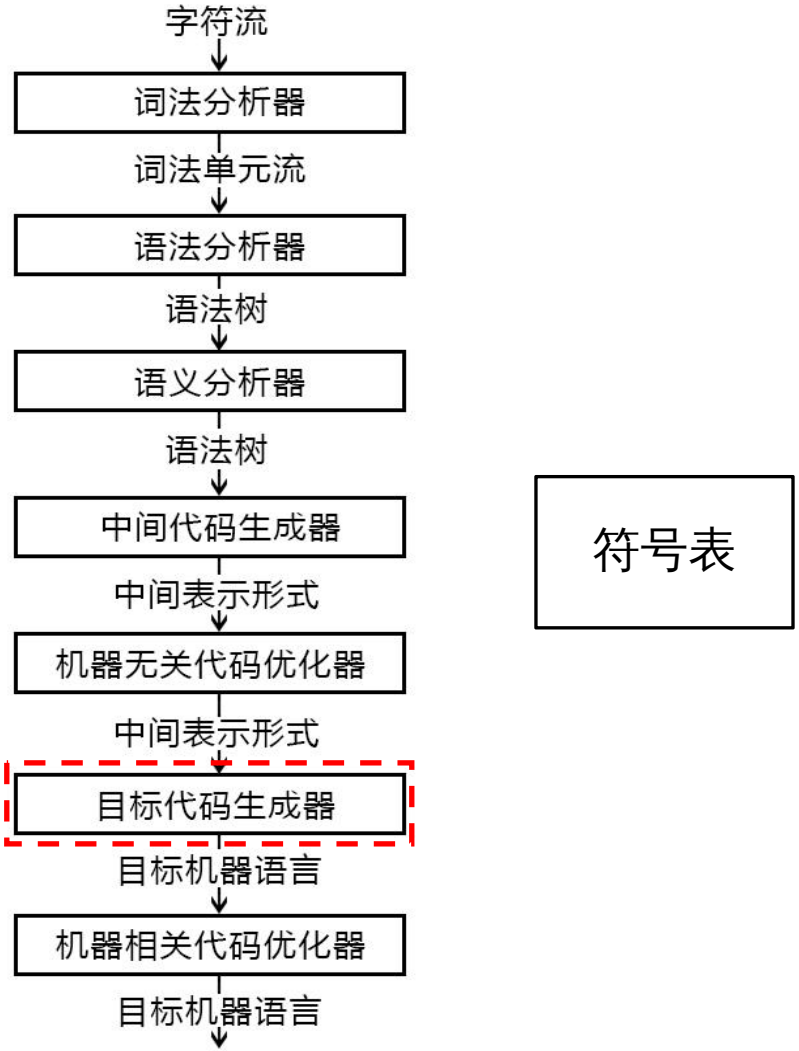
- **局部优化**
  - 常量合并：常数运算在编译期间完成，如8+9*4
  - 公共子表达式的提取: 避免重复计算相同表达式
- **循环优化**
  - 强度削减
    - 用较快的操作代替较慢的操作
  - 代码外提
    - 将循环不变计算移出循环

与机器有关的优化

- 寄存器的利用
  - 将常用量放入寄存器，以减少访问内存的次数
- 体系结构
  - MIMD、SIMD、SPMD、向量机、流水机
- 存储策略
  - 根据算法访存的要求安排：Cache、并行存储体系——减少访问冲突
- 任务划分
  - 按运行的算法及体系结构，划分子任务(MPMD)

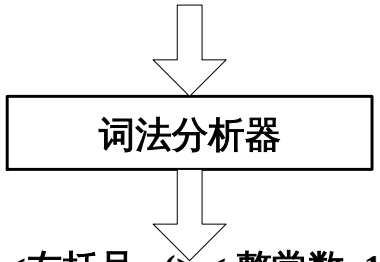
编译器的结构

- 目标代码生成
  - 输入：源程序的中间代码表示
  - 输出：目标语言程序
- 重要任务：
  - 指令选择
  - 寄存器分配
  - 指令排序



语句sum=(10+20)*(num+square);的翻译过程

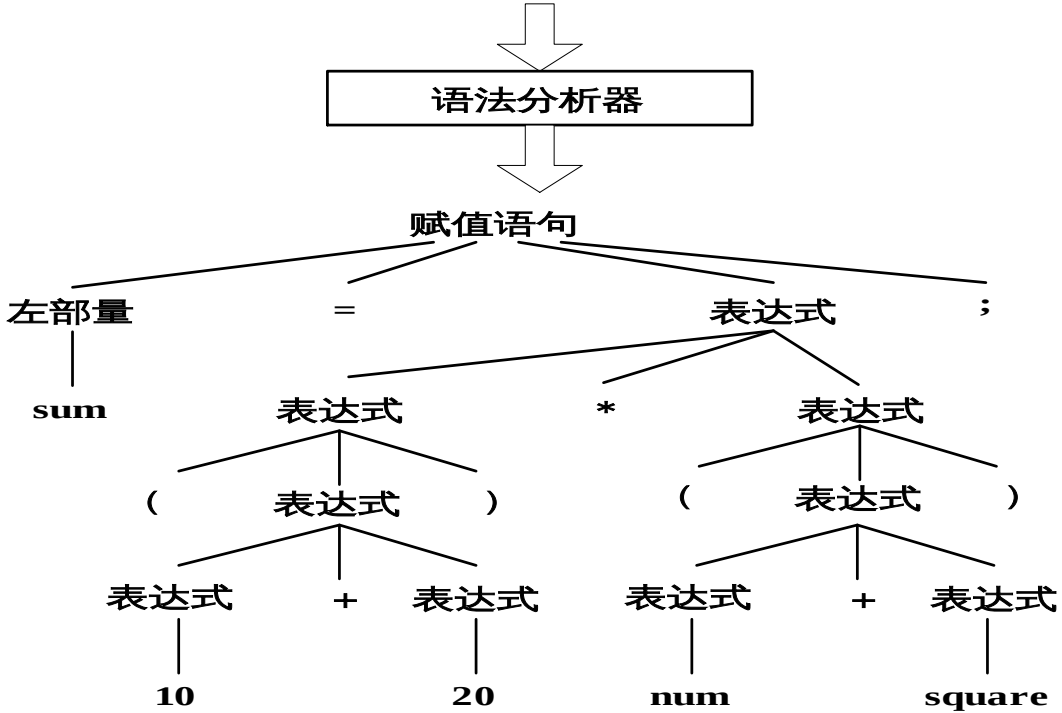
sum = (10+20)*(num+square)



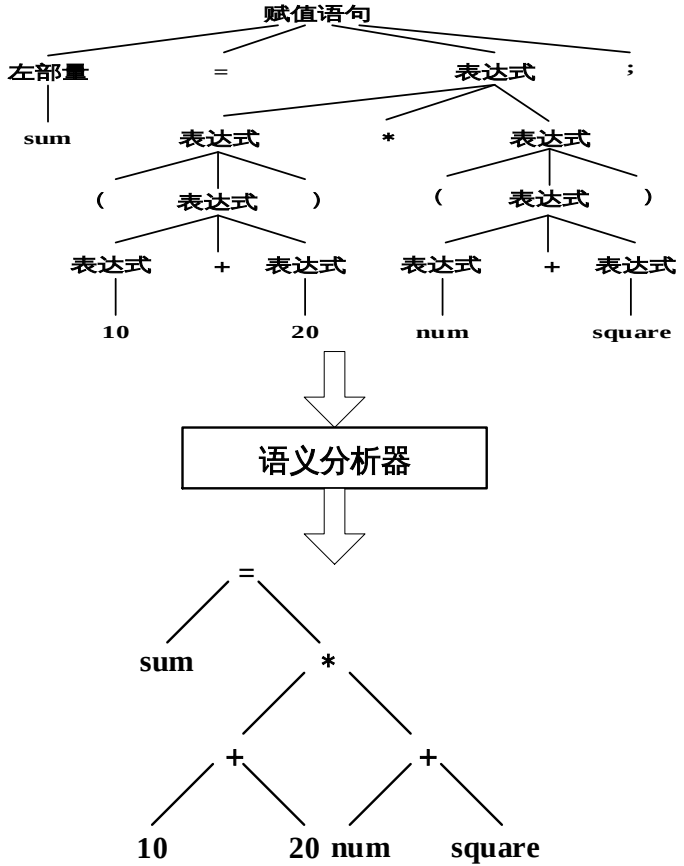
<标识符, Sum> <赋值号, => <左括号，(> < 整常数, 10 > < 加号, + >< 整常数, 20>
< 右括号，) > < 乘号, * > < 左括号，(> < 标识符, Num > <加号, + >
<标识符, Square > <右括号，> <分号, ; >

语句sum=(10+20)*(num+square);的翻译过程

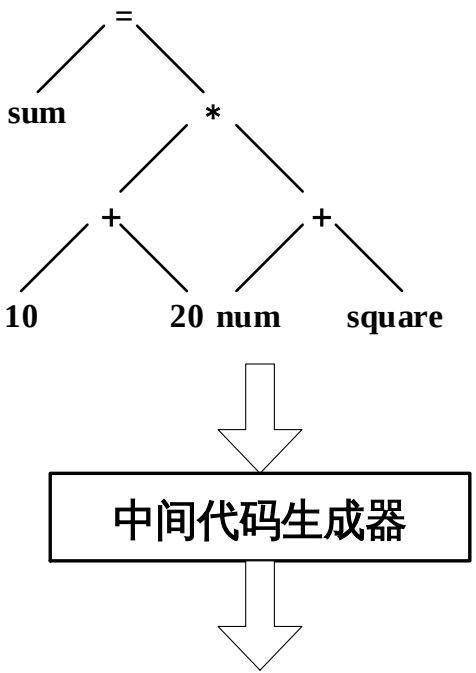
<标识符, Sum> <赋值号, => <左括号, (> < 整常数, 10 > < 加号, + >< 整常数, 20>
< 右括号,) > < 乘号, * > < 左括号, (> < 标识符, Num > <加号, + >
<标识符, Square > <右括号, > <分号, ; >



语句sum=(10+20)*(num+square);的翻译过程



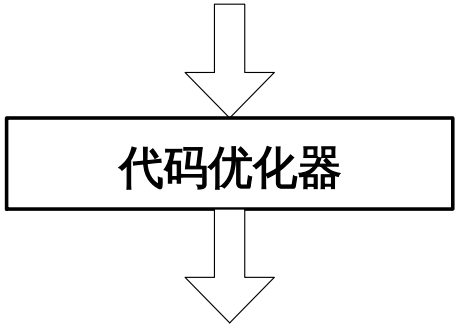
语句sum=(10+20)*(num+square);的翻译过程



T_1=10+20　　T_2=num+square　　T_3=T_1*T_2　　sum=T_3

语句sum=(10+20)*(num+square);的翻译过程

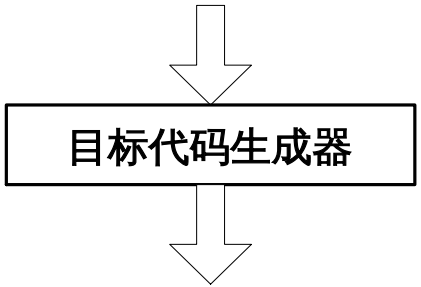
$T_1=10+20$ $T_2=num+square$ $T_3=T_1*T_2$ $sum=T_3$



$T_1=num+square$ $sum=30*T_1$

语句sum=(10+20)*(num+square);的翻译过程

T_1= num+square　　sum =30*T_1



MOV num，R1　MOV square，R2　ADD R2，R1　MUL 30，R1　MOV R1，sum

单选题 1分

设置

对代码进行优化变换时，最基本的优化原则是：

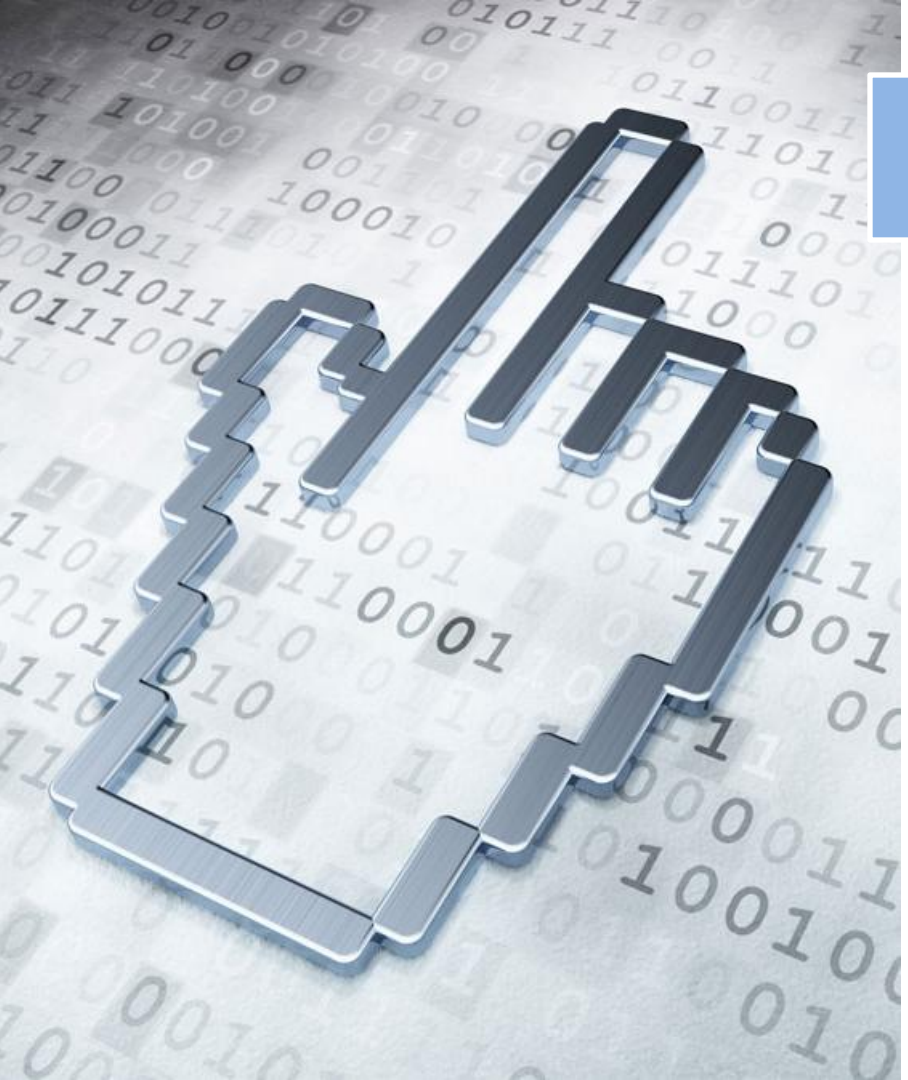
A 运行越快越好

B 占用空间越少越好

C 以上二者兼顾

D 语义等价

提交



本章内容

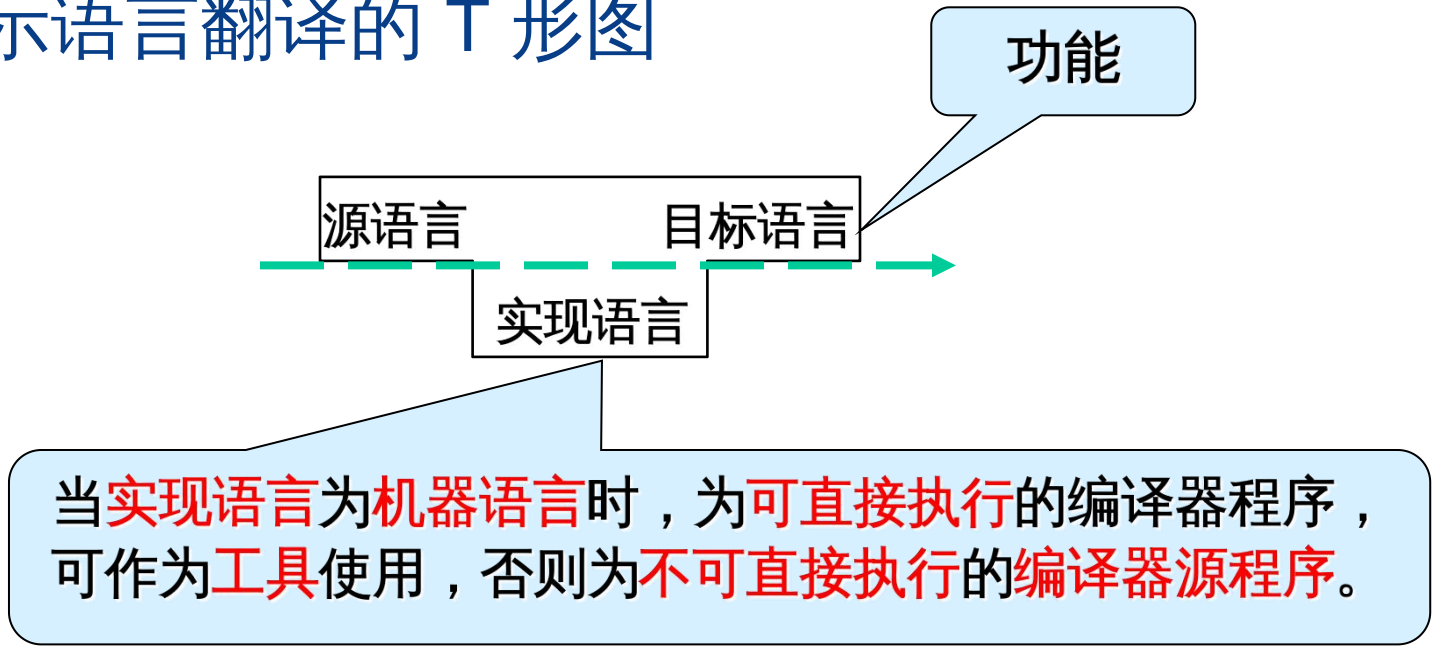
1.5 编译程序的生成

- 第一个高级语言编译器是怎样被编译的？
  - 直接用可运行的代码从头到尾编制——太费力！
- 如何利用A机器上的编译器开发B机器上的编译器？
- 如何继续开发新语言的编译器？

- 构建策略
  - 自展
  - 移植
  - 自动生成程序

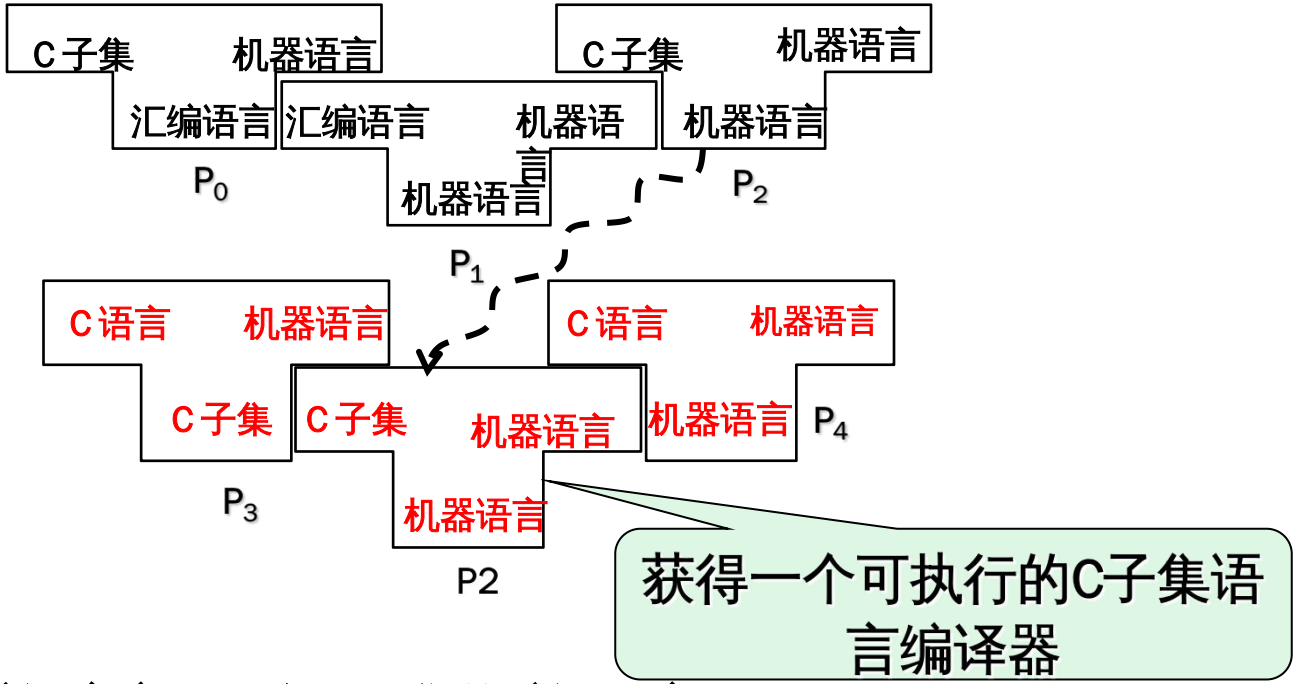
1. T形图

- 表示语言翻译的 T 形图



2. 自展

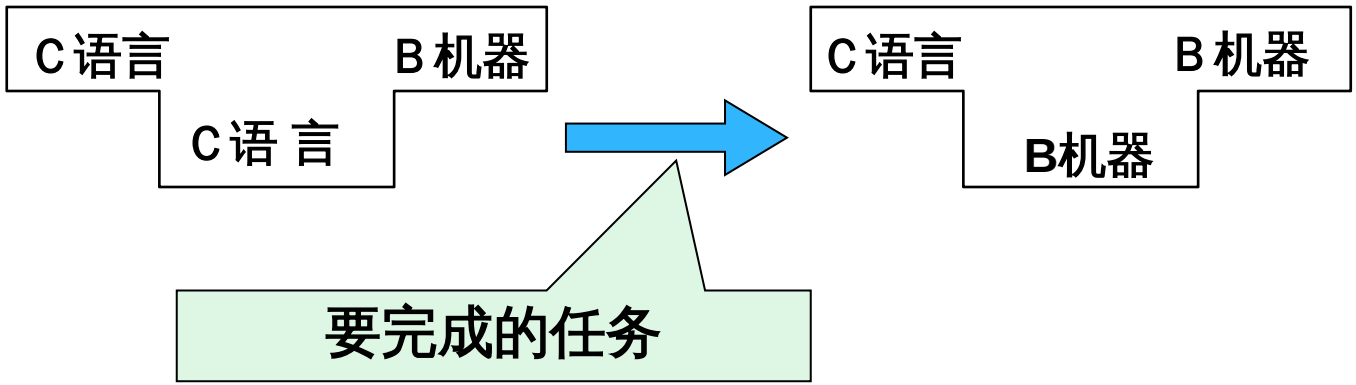
- 问题一：如何利用汇编器实现第一个C语言编译器？

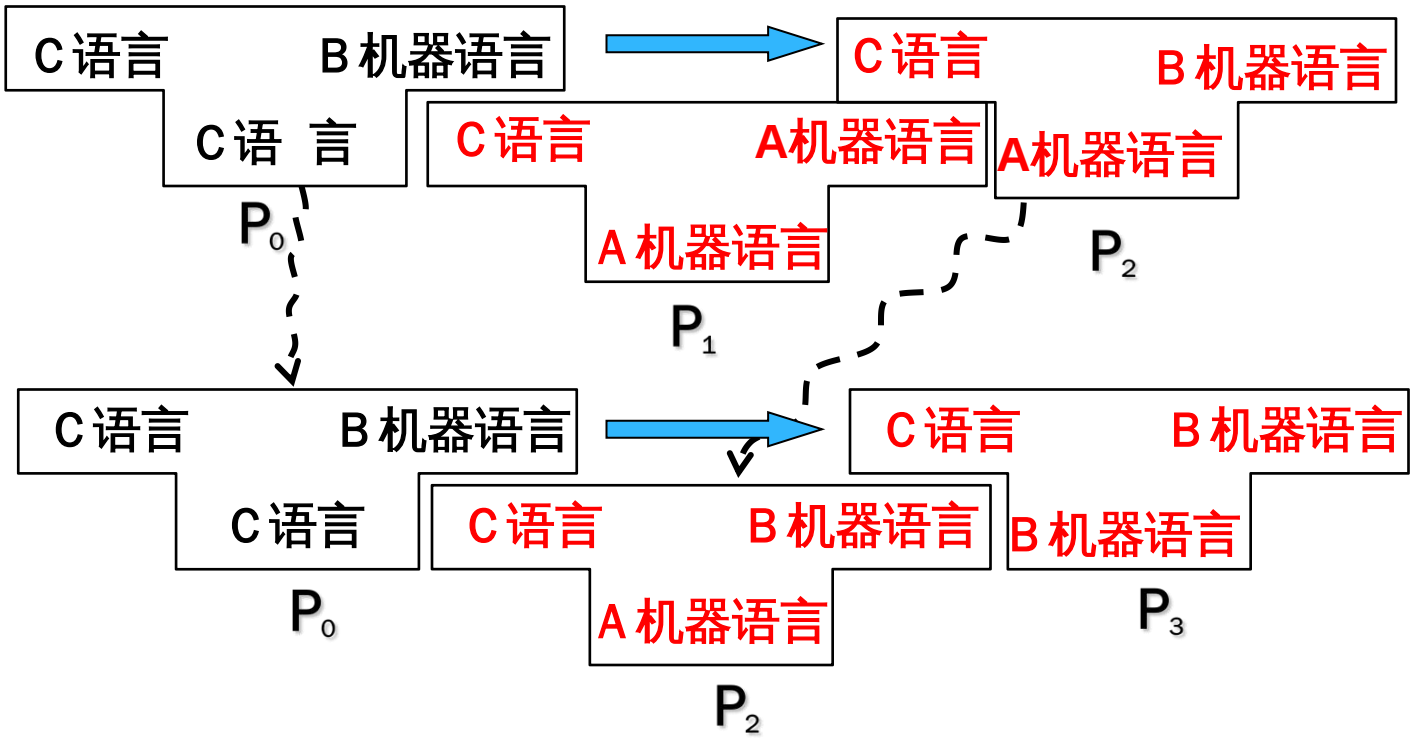


1. 用汇编语言实现一个 C子集的编译程序(P_0— 人)
2. 用汇编程序(P_1)处理该程序,得到(P_2:可直接运行)
3. 用C子集编制 C语言的编译程序(P_3—人)
4. 用P_2编译P_3，得到P_4

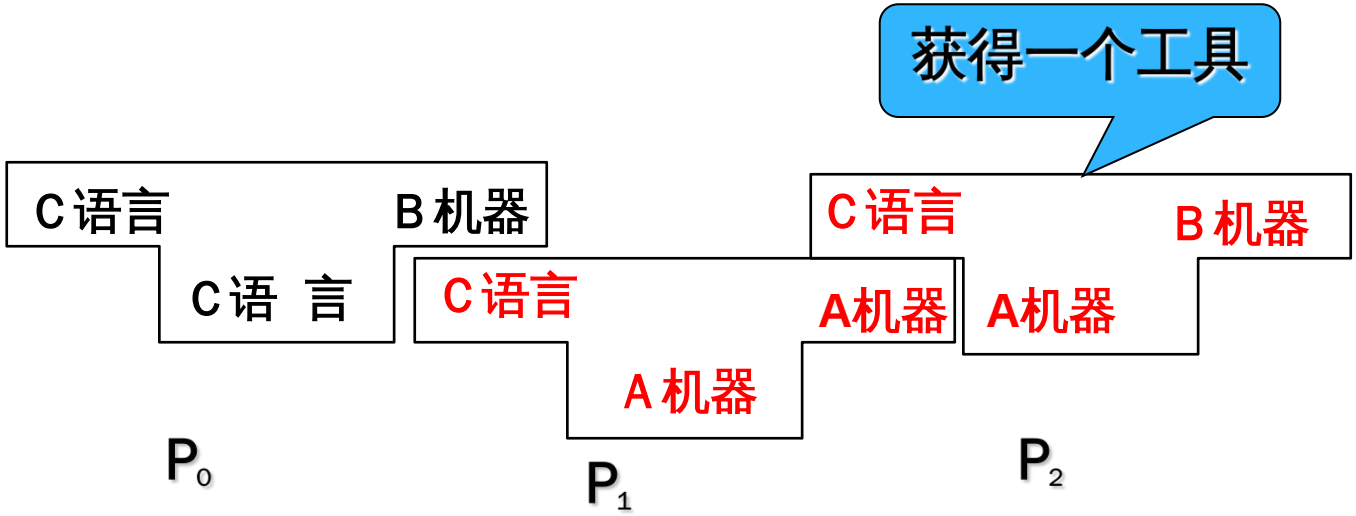
3.移植

- 问题二：A机上有一个C语言编译器，是否可利用此编译器实现B机上的C语言编译器？

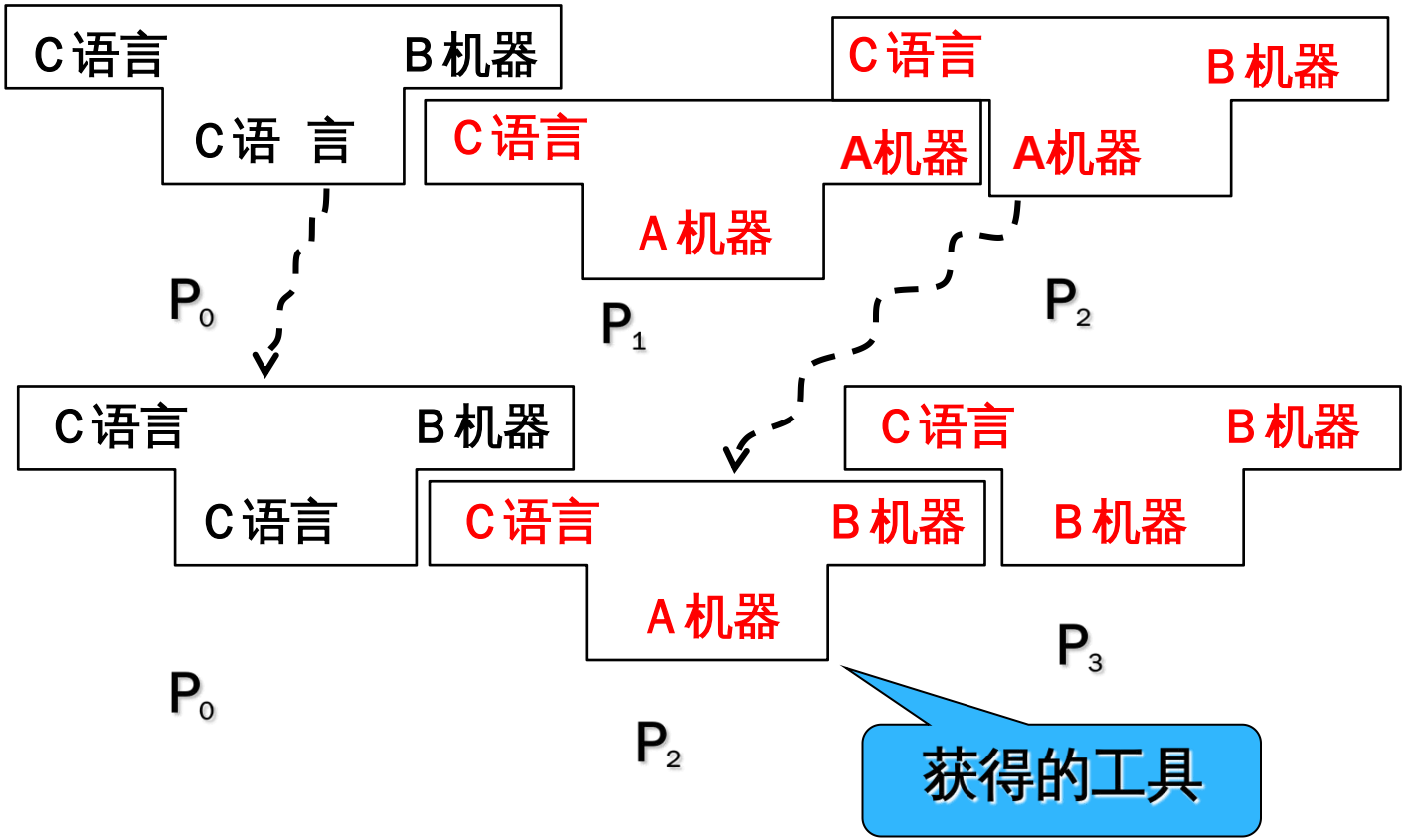



C语言
B机器语言
C语 言
P0
C语言
A机器语言
A 机器语言
P1
C语言
B机器语言
A机器语言
P2
C语言
B机器语言
C语言
P0
C语言
B机器语言
A 机器语言
P2
C语言
B机器语言
B机器语言
P3

2)问题的解决办法



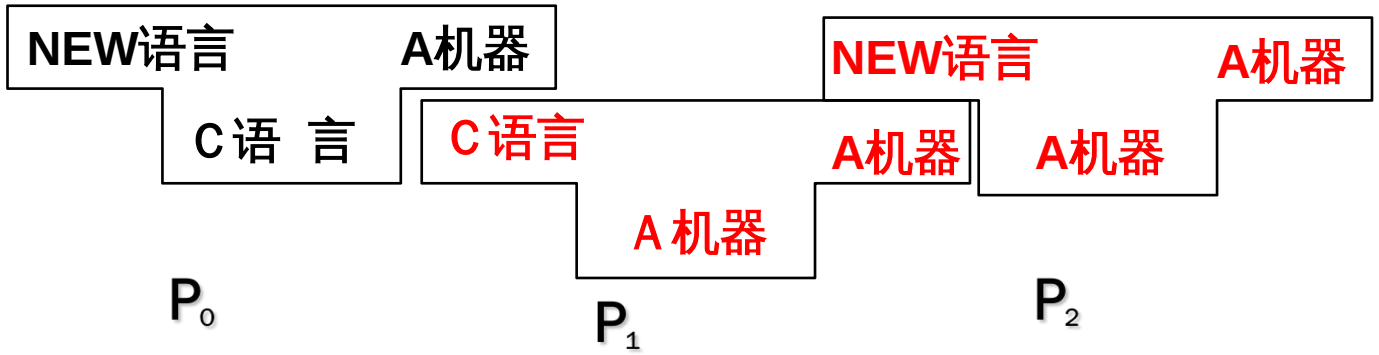
1. (人)用C语言编制B机的C编译程序P_0(C→B)
2. （A机的C编译器P_1)编译P_0，得到在A机上可运行的P_2(C→B)



3.(A 机的P_2)编译P_0，得到在B机上可运行的P_3(C→B)

4.本机编译器的利用

- 问题三： A机上有一个C语言编译器，现要实现一个新语言NEW的编译器？能利用交叉编译技术么？
- 用C编写NEW的编译，并用C编译器编译它



单选题 1分

设置

世界上第一个编译器适合通过哪种策略生成：

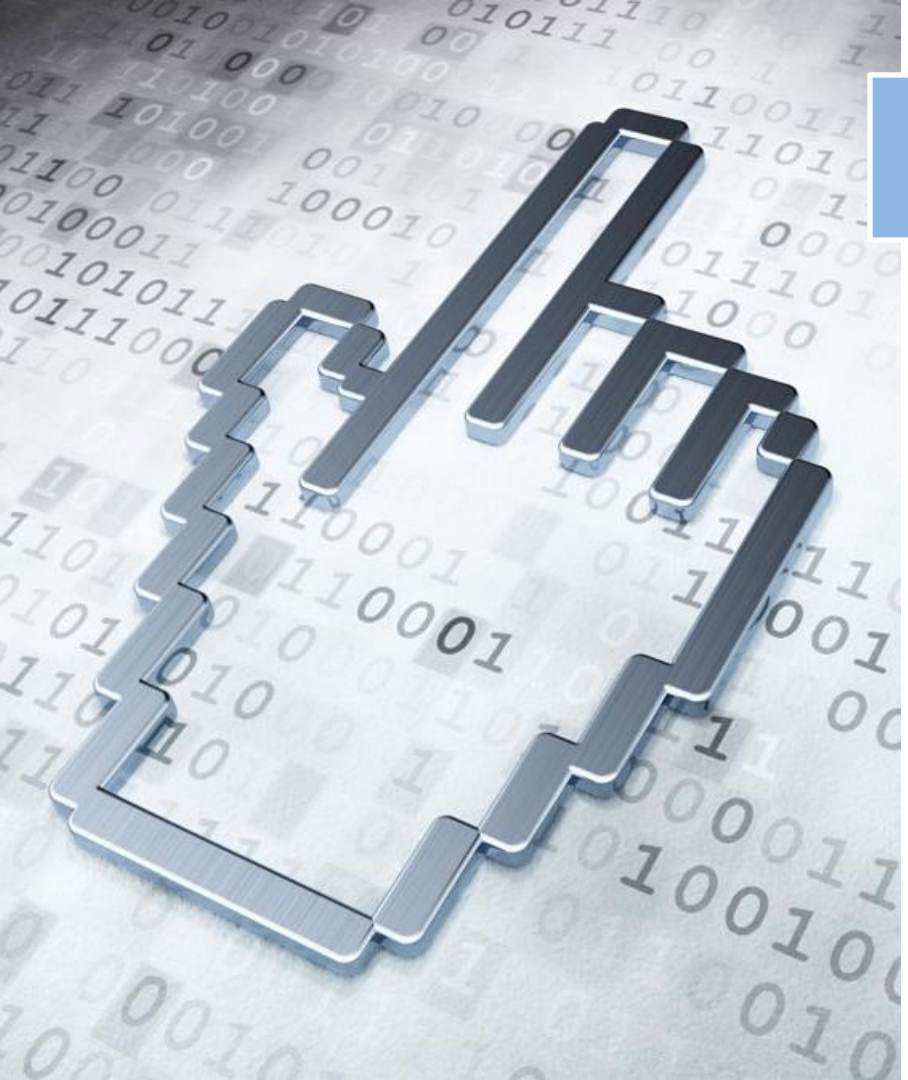
A 自展

B 移植

C 再生

D 利用已有编译器

提交



本章内容

1.1 什么是编译

1.2 编译系统的结构

1.3 编译器的生成

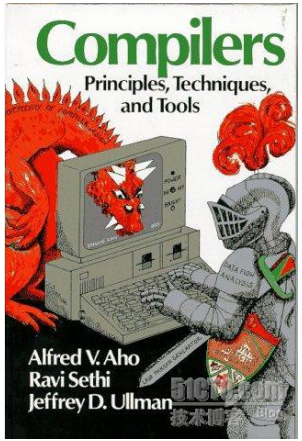
1.4 为什么要学习编译原理

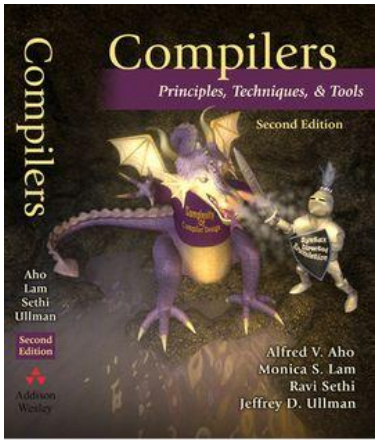
1.5 编译技术的应用

1.3 为什么要学习编译原理

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，本课程中的原理和技术都会反复用到。

——Alfred V.Aho





ACM图灵奖

https://amturing.acm.org/bysubject.cfm

编程语言、编译相关的获奖者是最多的 约占1/3

通过本课程的学习

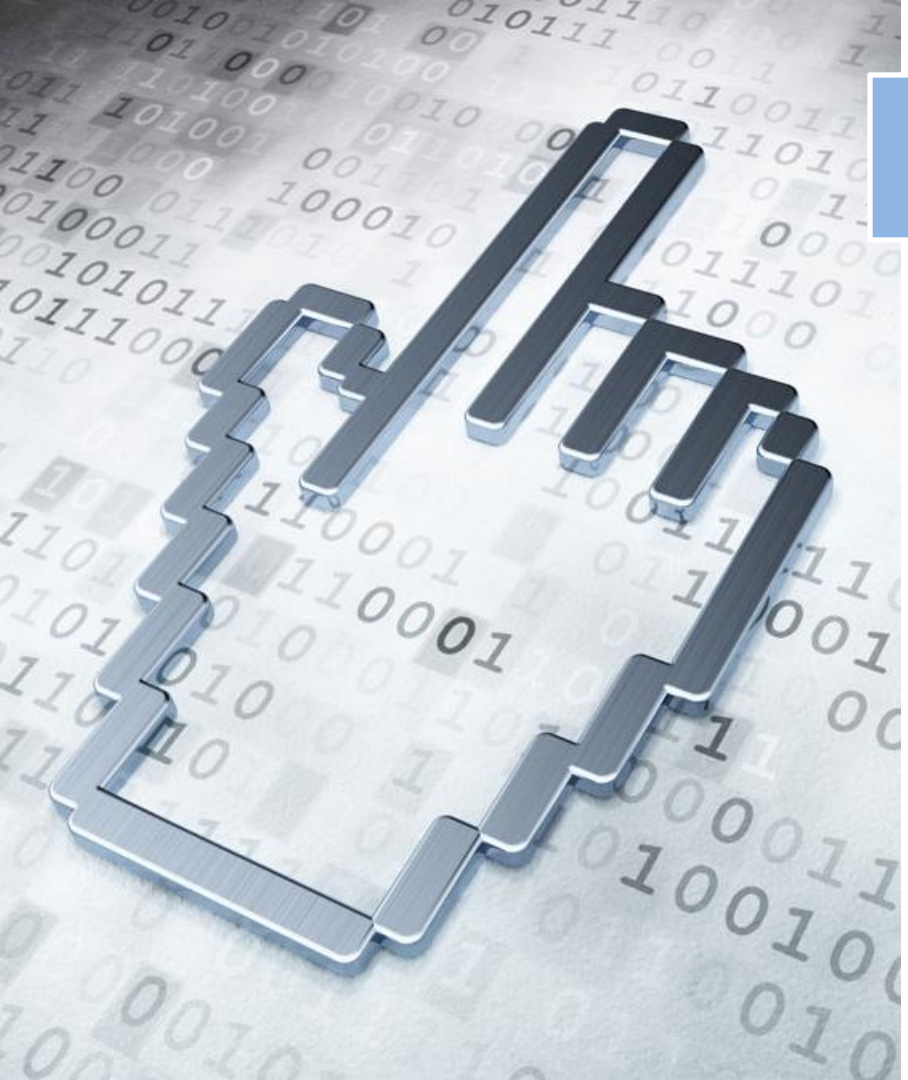
- 更深刻地理解高级语言程序的内部运行机制
  - 有利于提升正确、高效的编程能力，提升程序代码的执行效率，降低代码隐含错误、缺陷和漏洞。
- 计算机科学解决复杂问题的一般方法学；
  - “形式化→自动化”；
- 理论、技术到实践的完美诠释
- 所涉及的理论和方法在很多领域都会被用到
  - 自然语言处理、模式识别、人工智能、……
  - 句法分析、机器翻译、输入法、text2SQL解析技术

程序语言与编译系统发展的契机（国家需求）

- 人工智能技术的再次兴起
  - 人工智能加速芯片
  - 人工智能算法开发

  对程序语言和编译提出新的更高要求
- 面对美国芯片封锁，芯片自主可控迫在眉睫
  - 芯片自主研发与生产方兴未艾
  - 实现芯片自给，从芯片大国走向芯片强国

  研发和生态发展都需要编译技术
- 国产编程语言和编译器等核心基础系统软件自主需求
- 面向应用/硬件的领域特定语言DSL软硬件协同的编译系统优化



本章内容

1.1 什么是编译

1.2 编译系统的结构

1.3 编译器的生成

1.4 为什么要学习编译原理

1.5 编译技术的应用

1.4 编译技术的应用

- 很多应用软件都会用到编译技术
  - 开发新语言（DSL（Domain Specific Language）领域特定语言），扩展语言新特性
  - 语言升级新体系结构的优化支持（并行和内存层次结构）
- 提高软件开发效率的工具
  - 程序错误检测和定位：数据流分析技术
  - 内存管理错误检测
  - 专用优化编译器
- 新计算机体系结构的设计
  - 计算机系统的整体性能也依赖编译器对其特性支持的程度

1.4 编译技术的应用

- 结构化编辑器（Structure editors）
  - 引导用户在语言的语法约束下编制程序
  - 能自动地提供关键字和与其匹配的关键字

1.4 编译技术的应用

- 结构化编辑器（Structure editors）
- 智能打印机（Pretty printers）
  - 对程序进行分析，打印出结构清晰的程序
    - 注释以一种特殊的字体打印
    - 根据各个语句在程序的层次结构中的嵌套深度进行缩进

1.4 编译技术的应用

- 结构化编辑器（Structure editors）
- 智能打印机（Pretty printers）
- 静态检测器（Static checkers）
  - 静态定位程序中的错误
    - 释放空指针或已释放过的指针
    - 检测出程序中永远不能被执行的语句

1.4 编译技术的应用

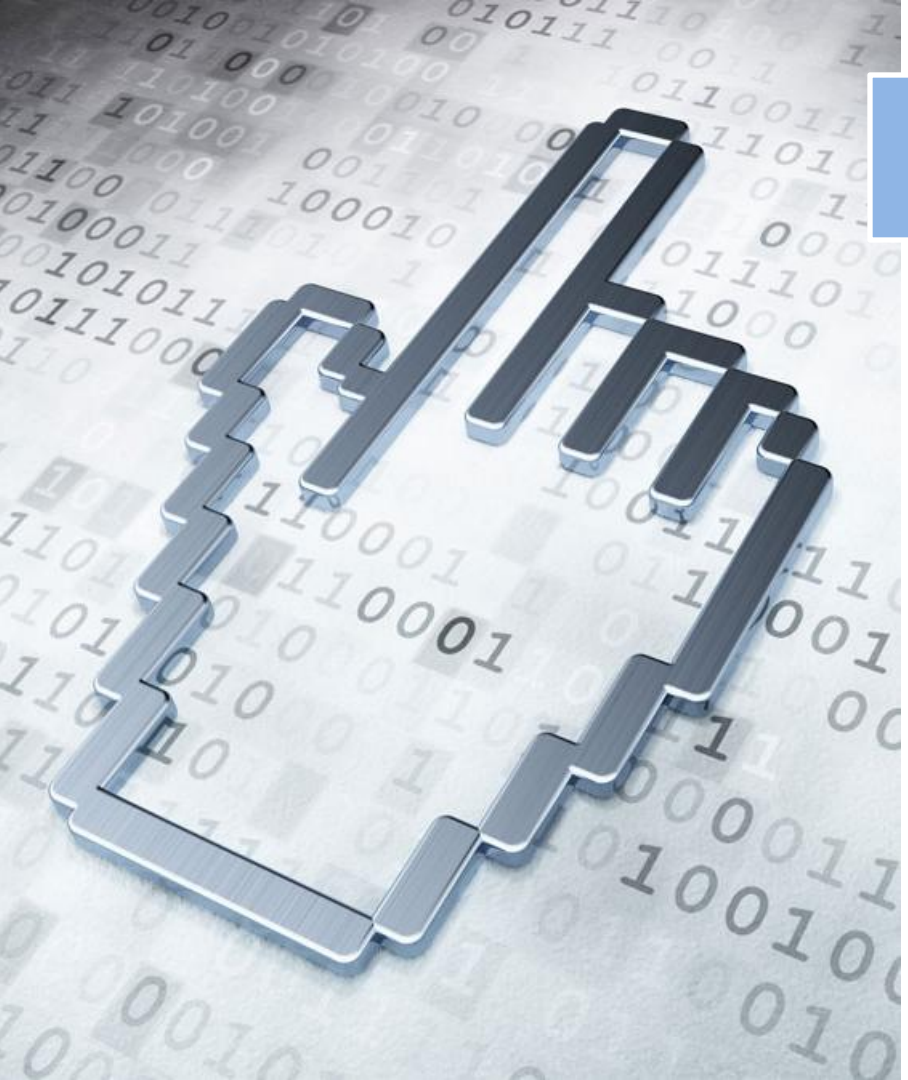
- 结构化编辑器（Structure editors）
- 智能打印机（Pretty printers）
- 静态检测器（Static checkers）
- 文本格式器（Text formatters）
  - 文本格式器处理的字符流中除了需要排版输出的字符以外，还包含一些用来说明字符流中的段落、图表或者上标和下标等数学结构的命令

1.4 编译技术的应用

- 结构化编辑器（Structure editors）
- 智能打印机（Pretty printers）
- 静态检测器（Static checkers）
- 文本格式器（Text formatters）
- 数据库查询解释器（ Database Query Interpreters ）
  - 数据库查询语句由包含了关系和布尔运算的谓词组成。查询解释器把这些谓词翻译成数据库命令，在数据库中查询满足条件的记录。

1.4 编译技术的应用

- 结构化编辑器（Structure editors）
- 智能打印机（Pretty printers）
- 静态检测器（Static checkers）
- 文本格式器（Text formatters）
- 数据库查询解释器（ Database Query Interpreters ）
- 高级语言的翻译工具



本章小结

什么是编译

编译系统的结构

编译器的生成方法

为什么要学习编译原理

编译技术的应用

课程主要内容

1．绪论	（2学时）
2. 语言及其文法	（2学时）
3．词法分析	（3学时）
4．语法分析	（9学时）
5．语法制导翻译	（6学时）
6．中间代码生成	（7学时）
7．运行时的存贮组织	（3学时）
8．代码优化	（6学时）
9．代码生成	（2学时）

结束