

期末60%
上机20%
作业20%

编译原理

有问题发邮件

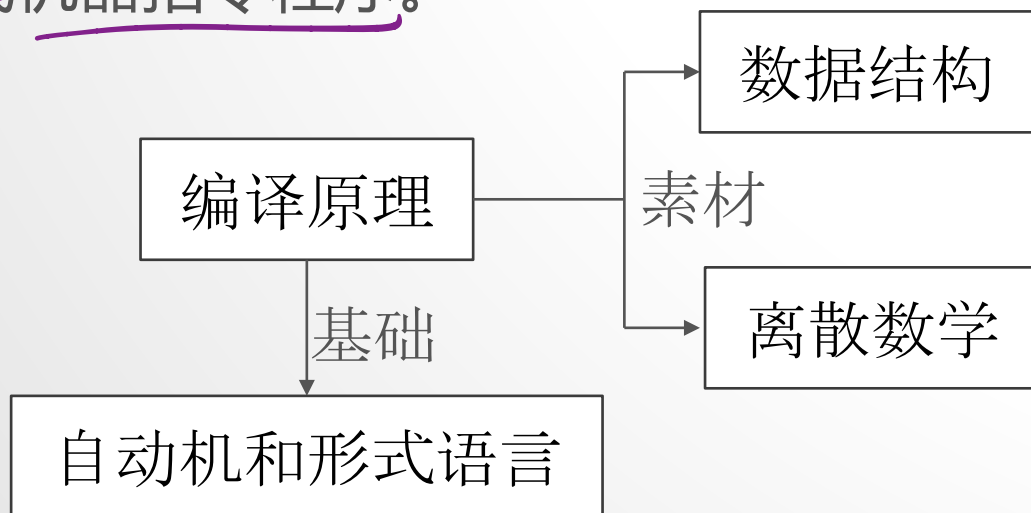
主讲：王笑琨

E_mail : wangxiaokun@ustb.edu.cn

机电101

编译课程的内容：

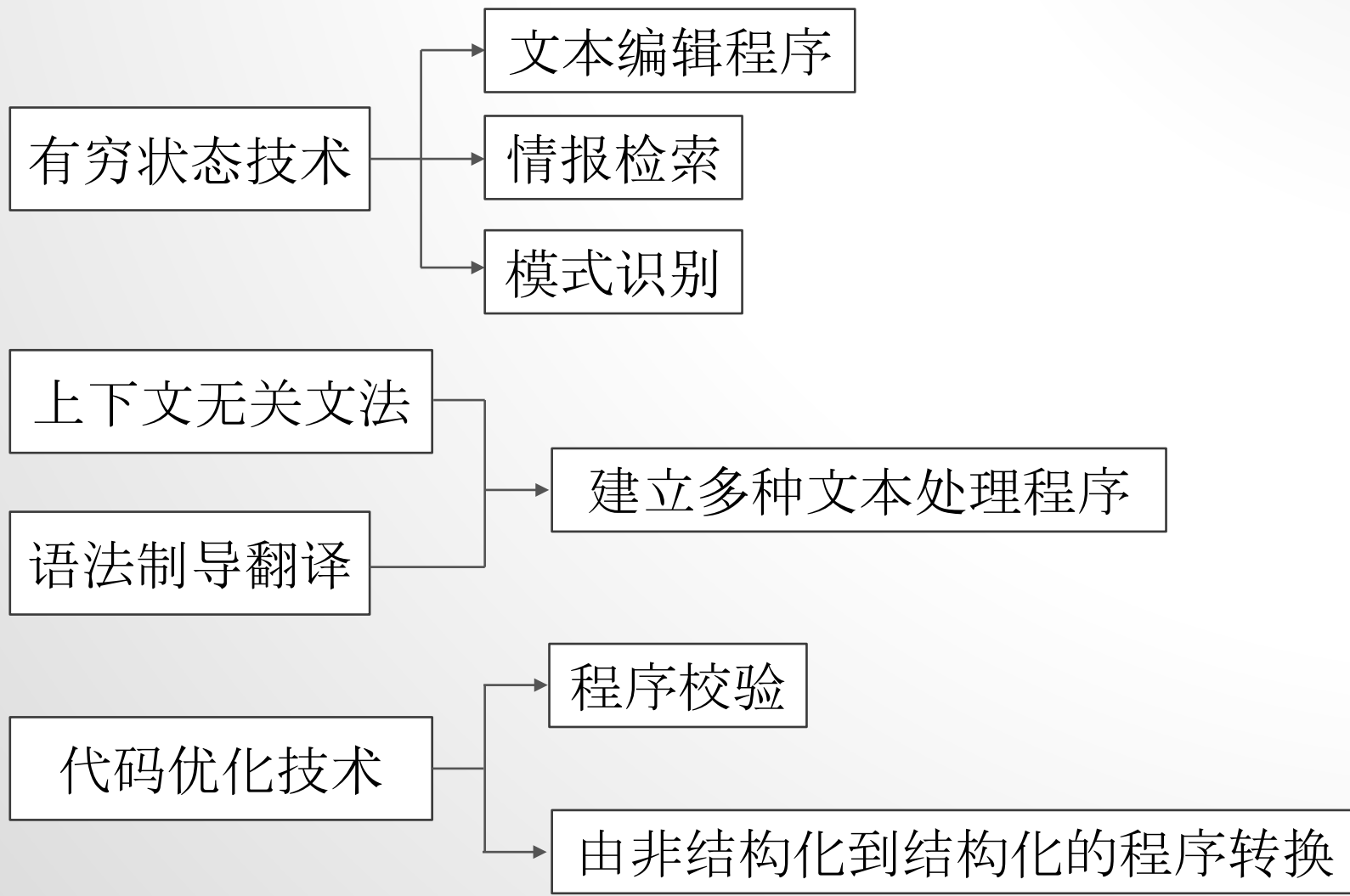
- 介绍编译程序的工作原理与构造方法；
- 详细介绍如何将一个用高级语言编写的源程序
翻译成机器指令程序。



与其他课程关系

编译理论的应用

~~编译理论的许多想法和技术~~可用于一般软件的设计。



参考书

- 吕映芝等，编译原理，清华大学出版社
- 陈意云等，编译原理，高等教育出版社
- 杜淑敏等，编译程序设计原理，北京大学出版社
- Alfred V.Aho等，编译原理，机械工业出版社
- Terrence W.Pratt, Marvin V.Zelkowitz
Programming Languages Design and
Implementation, Prentice-Hall



第一章 绪论

什么是编译

哈尔滨工业大学 陈鄞



计算机程序设计语言及编译

类似于数学定义或
自然语言的简洁形式

- 接近人类表达习惯
- 不依赖于特定机器
- 编写效率高

高级语言
(High Level Language)

引入助记符

- 依赖于特定机器，
非计算机专业人员
使用受限制

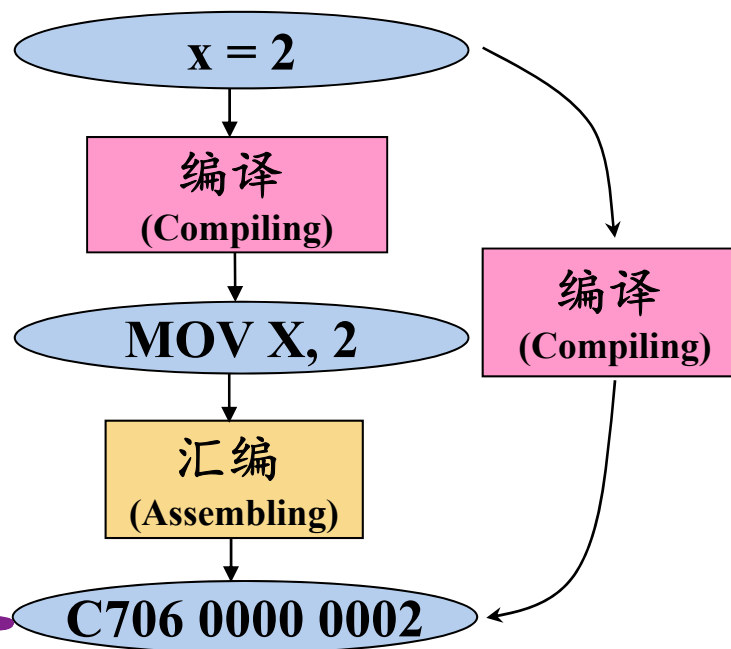
- 编写效率依然很低

汇编语言
(Assembly Language)

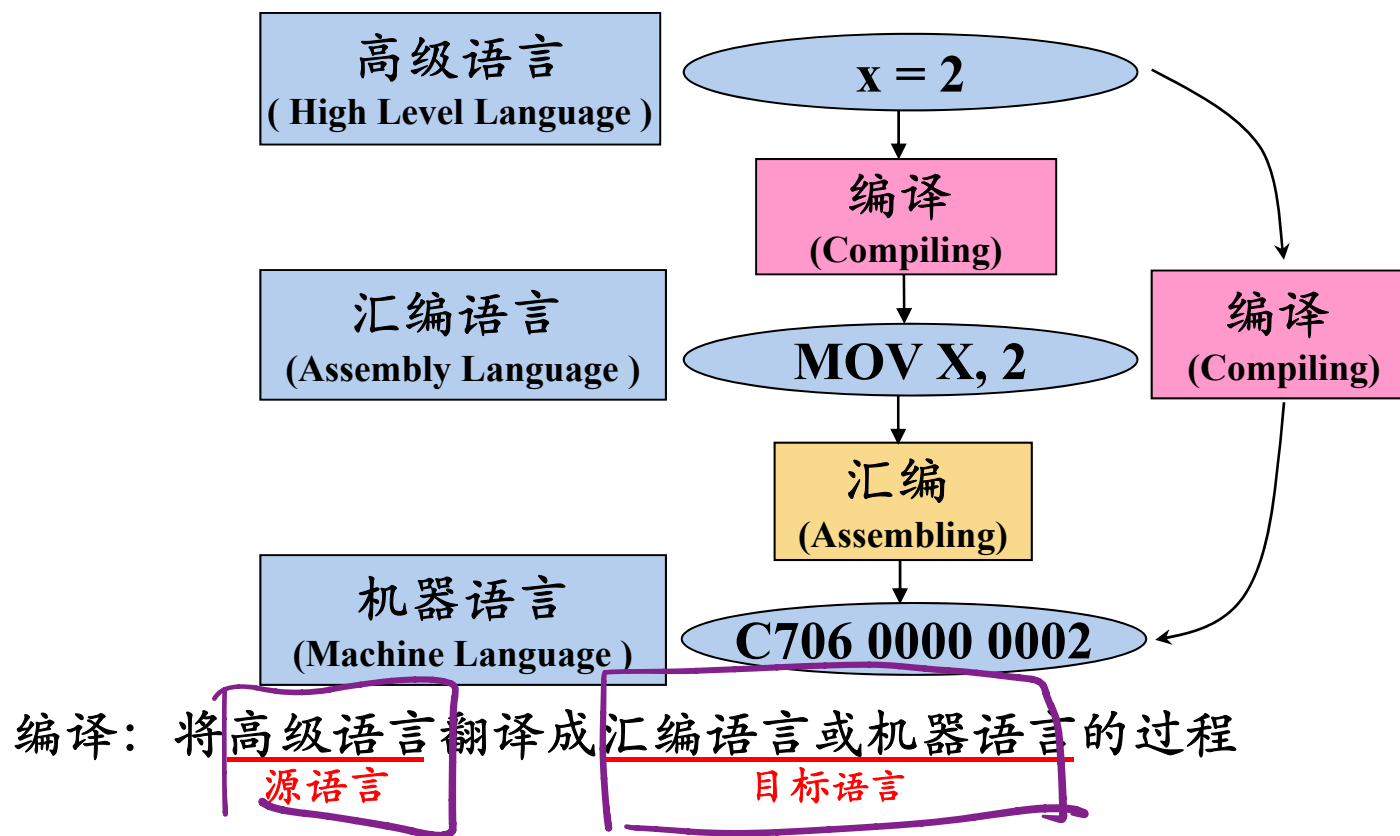
可以被计算机直接理解

- 与人类表达习惯
相去甚远
- 难记忆
- 难编写、难阅读
- 易写错

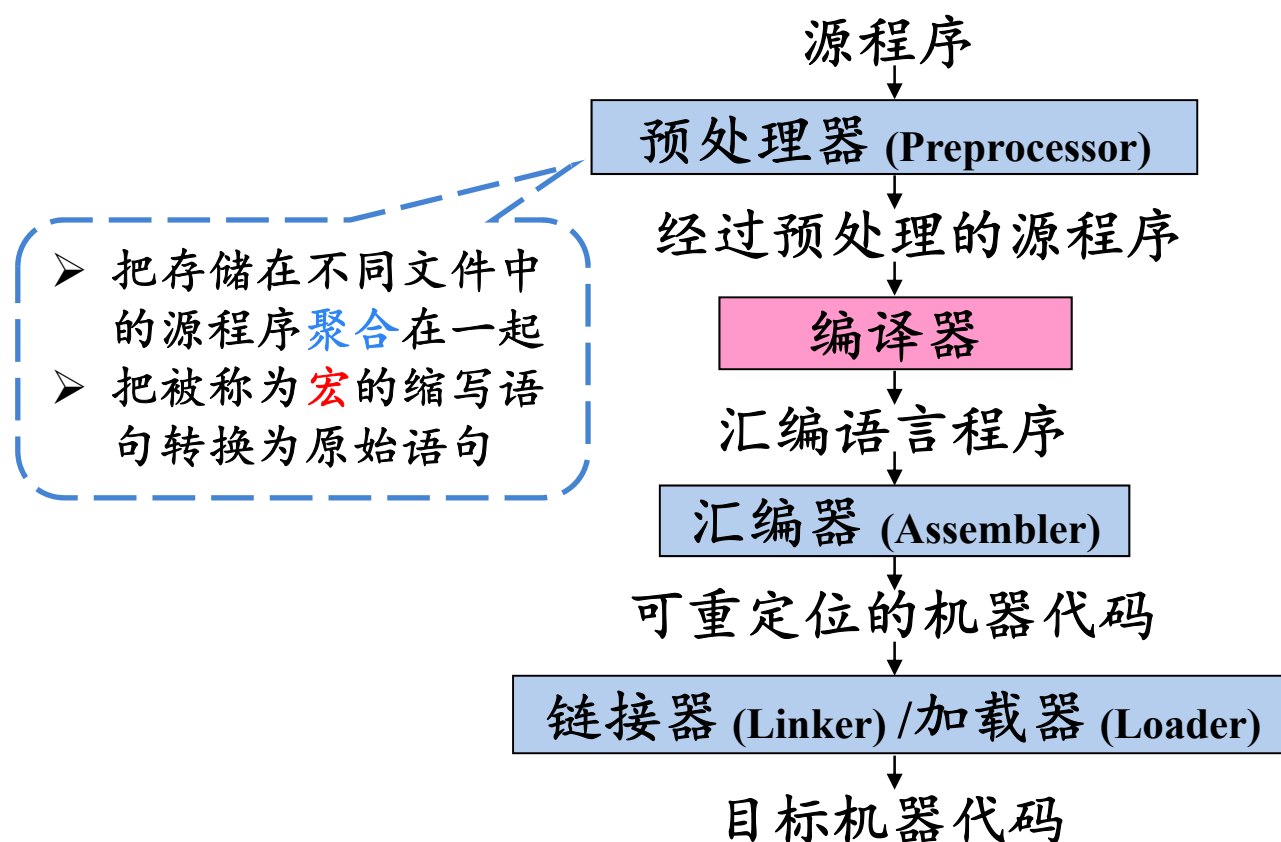
机器语言
(Machine Language)



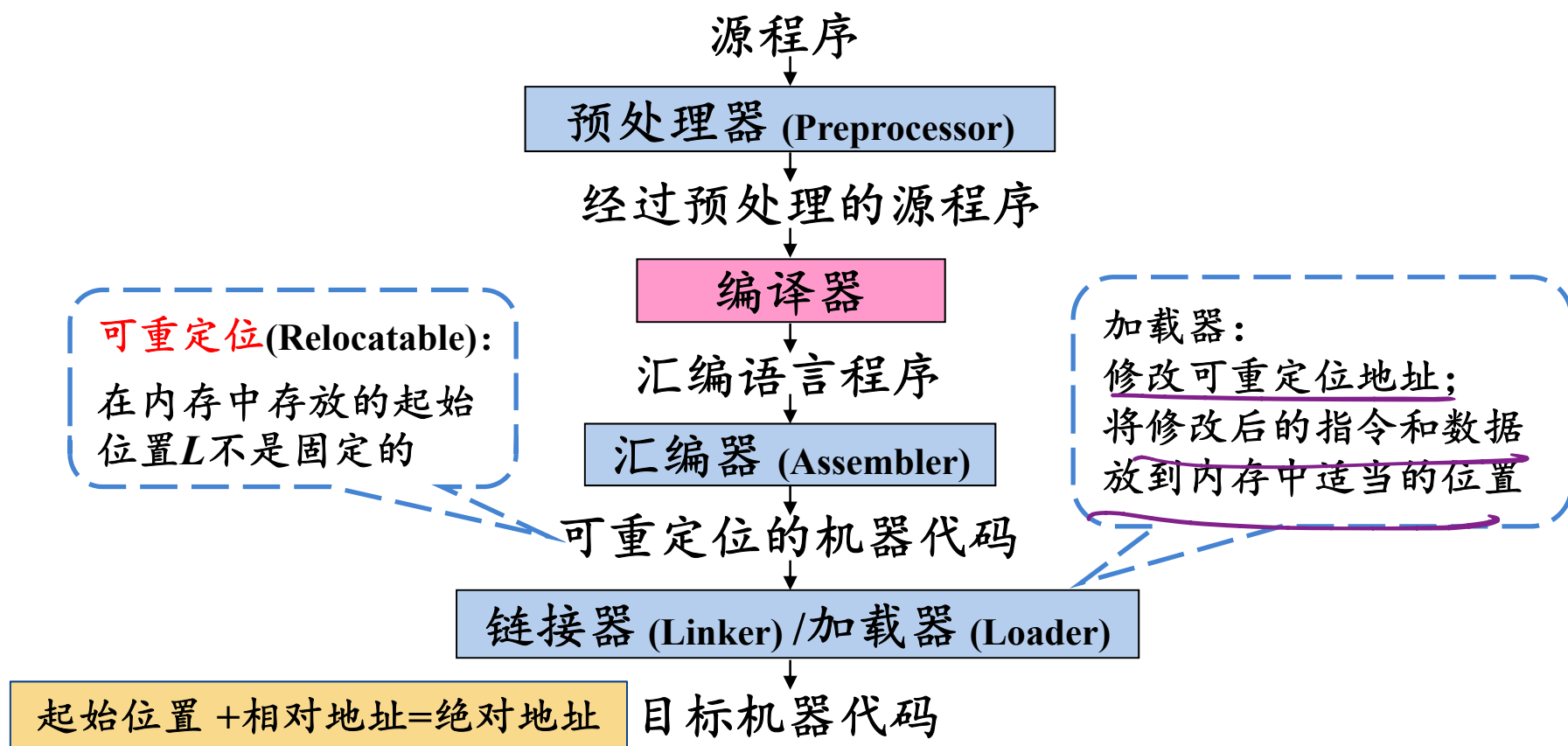
计算机程序设计语言及编译



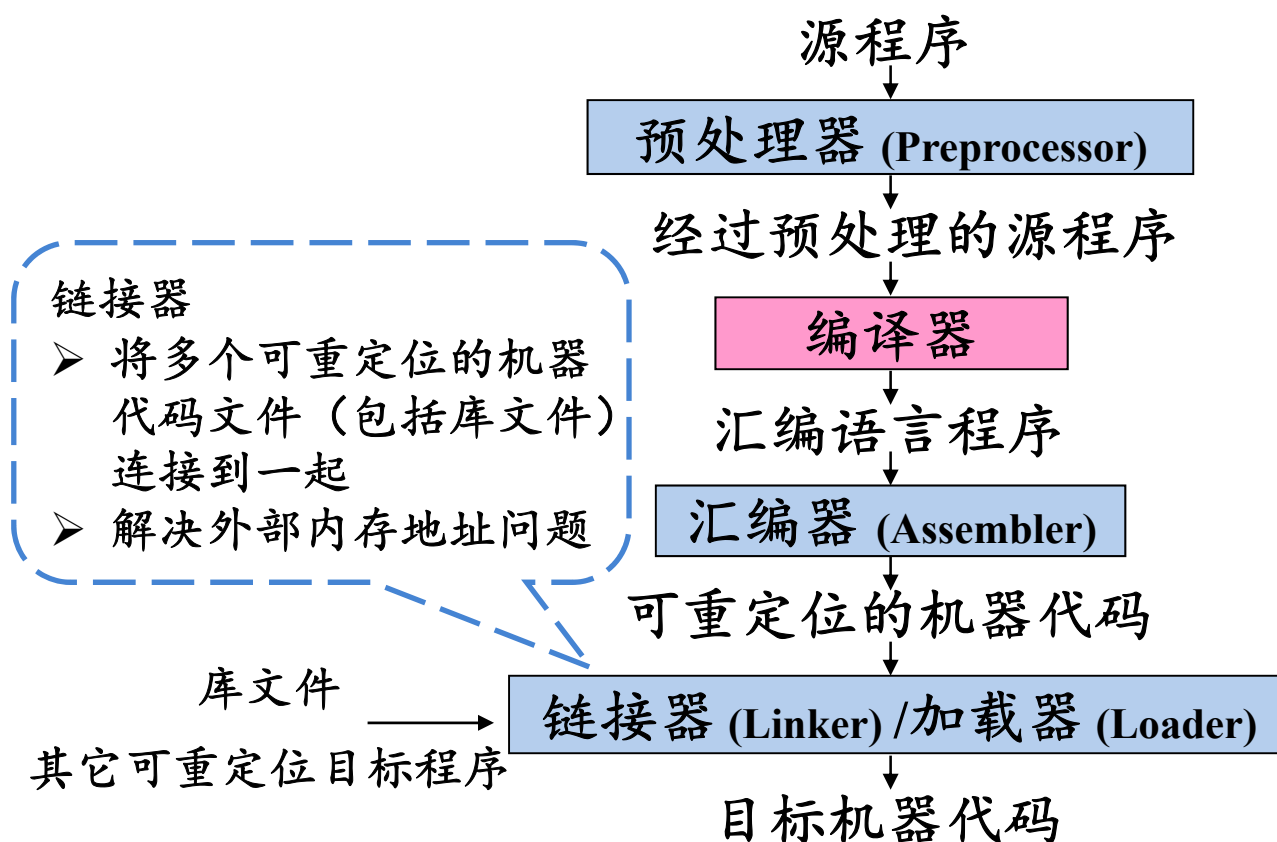
编译器在语言处理系统中的位置



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





第一章 绪论

编译系统的结构

哈尔滨工业大学 陈鄞



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)
            printf("%5d",i);
        number++;
        printf("\n");
    }
    if(number==0)
        printf("no\n");
}
```

高级语言程序

编译器

汇编语言程序/机器语言程序

机器是怎么翻译的？

```
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
75     ret
76 clearcrt endp
77 ;行, 入口参数, 行数 no
78 a proc near, no:word
79     mov cx, no
80     mov cx, 1
81     .endif
82     .repeat
83         mov ah, 02h
84         mov dl, cr
85         int 21h
86         mov ah, 02h
87         mov dl, lf
88         int 21h
89     .untilcx
90     ret
91 nextlien endp
92     end
93
94
```

人工英汉翻译的例子

在房间里，他用锤子砸了一扇窗户。

[In the room], he broke a window <with a hammer>

状语

主语

谓语

宾语

补语

源语言句子

第1步
分析源语言

句子的语义

第2步
生成目标语言

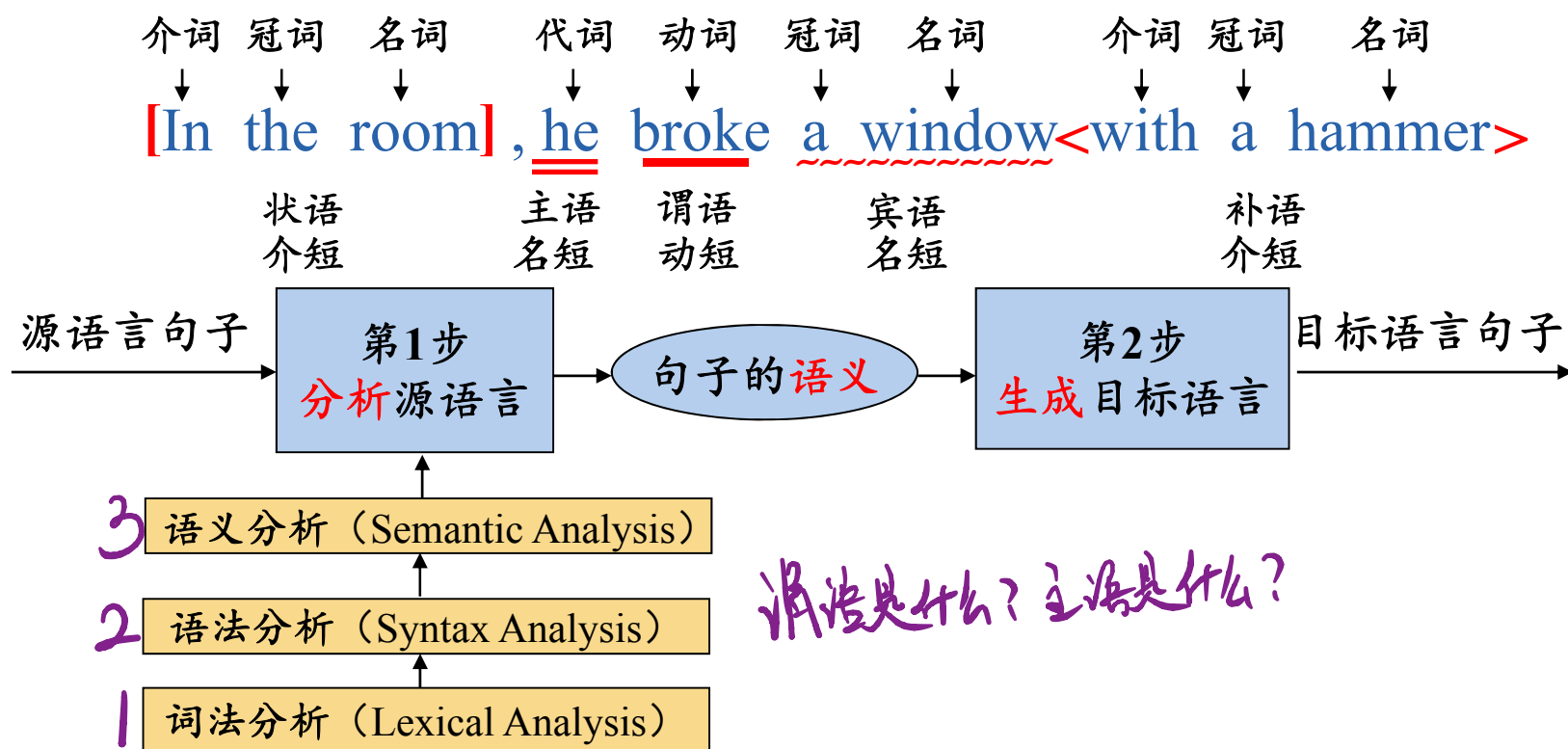
目标语言句子

语义分析 (Semantic Analysis)

中间表示，
独立于具体的语言



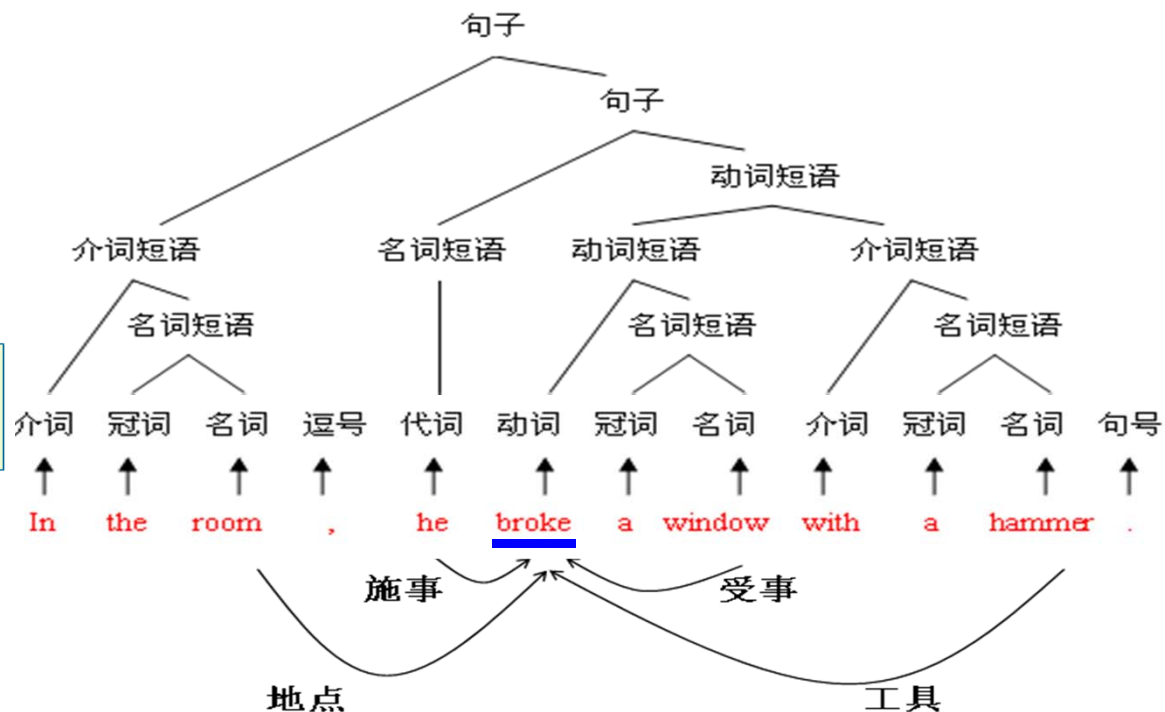
人工英汉翻译的例子



人工英汉翻译的例子

In the room , he broke a window with a hammer

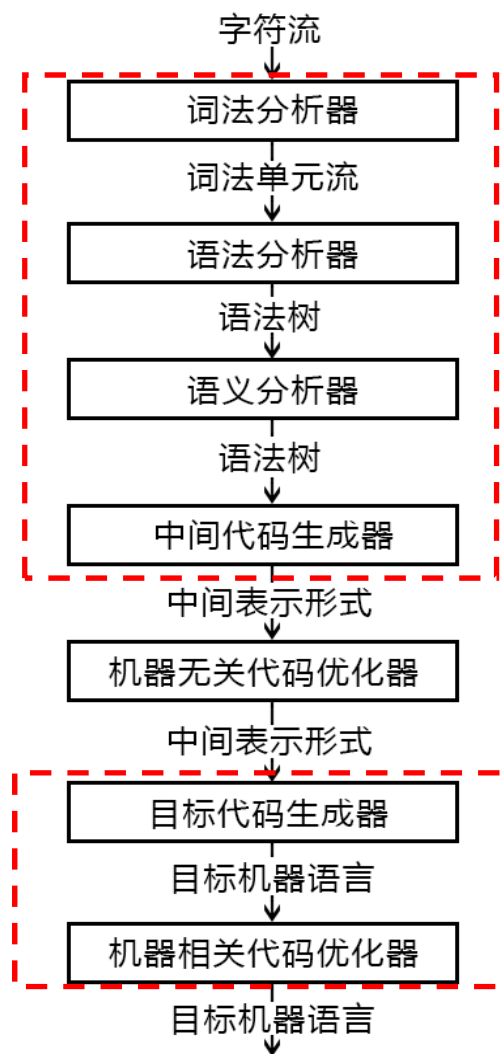
- 词法分析
- 语法分析
- 语义分析



编译器的结构

分析部分/
前端(front end):
与源语言相关

综合部分/
后端(back end):
与目标语言相关





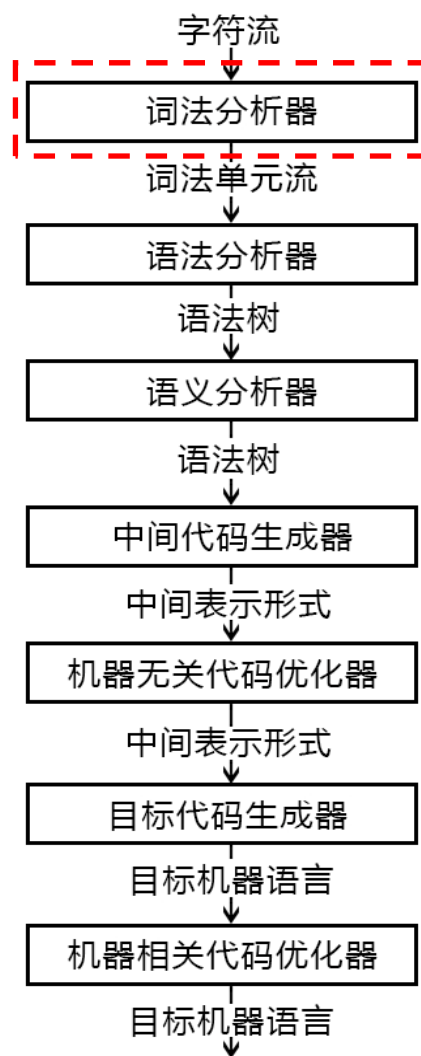
第一章 绪论

词法分析概述

哈尔滨工业大学 陈鄞



编译器的结构



词法分析/扫描(Scanning)

➤ 词法分析的主要任务

从左向右逐行扫描源程序的字符，识别出各个单词，确定单词的类型。

将识别出的单词转换成统一的机内表示——词法单元(token)形式

token: <种别码, 属性值>

	单词类型	种别	种别码
1	关键字	program、if、else、then、...	一词一码
2	标识符	变量名、数组名、记录名、过程名、...	多词一码
3	常量	整型、浮点型、字符型、布尔型、...	一型一码
4	运算符	算术 (+ - * / ++ --) 关系 (> < == != >= <=) 逻辑 (& ~)	一词一码 或 一型一码
5	界限符	; () = { } ...	一词一码

例：词法分析后得到的token序列

➤ 输入 while(value!=100){num++;}

➤ 输出

1	while	< WHILE ,	-	>
2	(< SLP ,	-	>
3	value	< IDN ,	value	>
4	!=	< NE ,	-	>
5	100	< CONST ,	100	>
6)	< SRP ,	-	>
7	{	< LP ,	-	>
8	num	< IDN ,	num	>
9	++	< INC ,	-	>
10	;	< SEMI ,	-	>
11	}	< RP ,	-	>

种别码 属性值

如何实现
词法分析器？



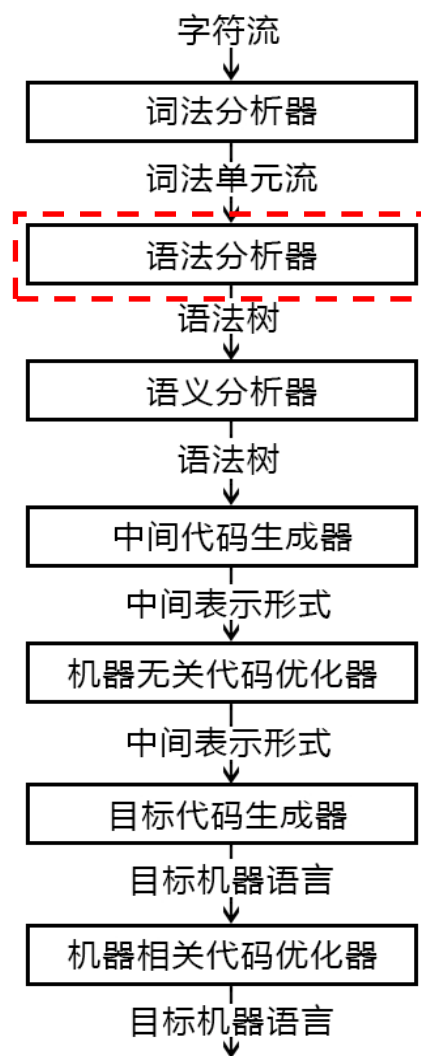
第一章 绪论

语法分析概述

哈尔滨工业大学 陈鄞

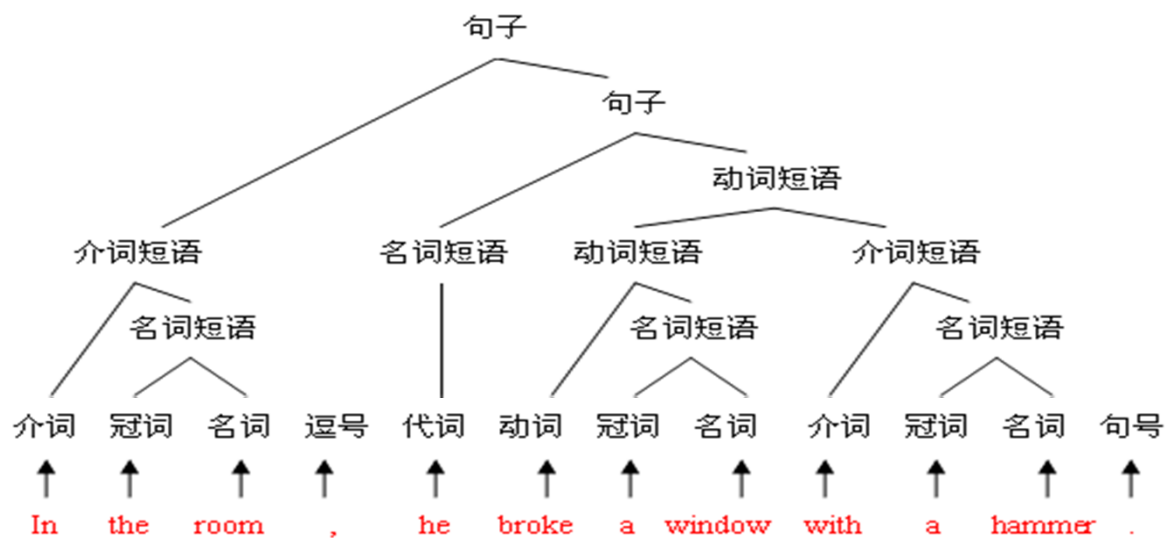


编译器的结构



语法分析 (*parsing*)

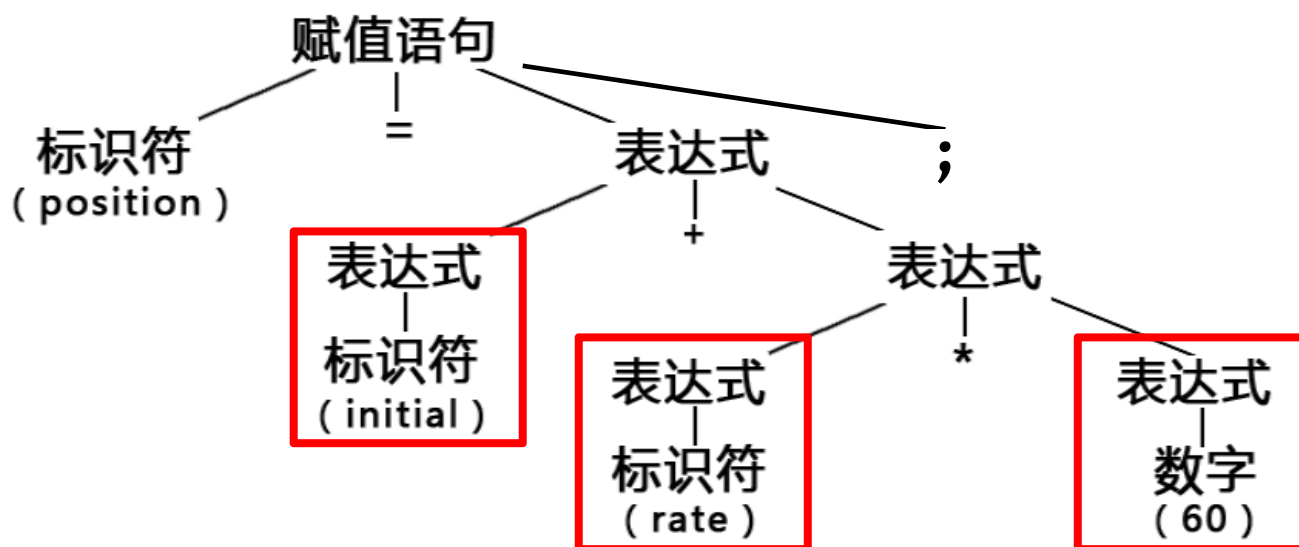
- 语法分析器(parser)从词法分析器输出的token序列中识别出各类短语, 并构造语法分析树(parse tree)
- 语法分析树描述了句子的语法结构



例1：赋值语句的分析树

position = initial + rate * 60 ;

$\langle \text{id, position} \rangle \Rightarrow \langle \text{id, initial} \rangle \langle + \rangle \langle \text{id, rate} \rangle \langle * \rangle \langle \text{num, 60} \rangle \langle ; \rangle$



例2：变量声明语句的分析树

➤ 文法：

$\langle D \rangle \rightarrow \langle T \rangle \langle IDS \rangle ;$

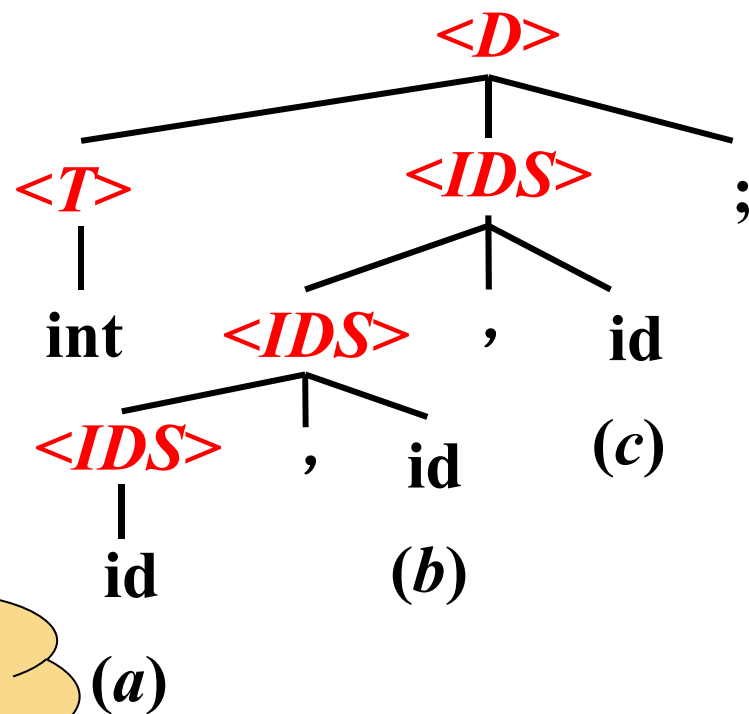
类型 $\langle T \rangle \rightarrow \text{int} \mid \text{real} \mid \text{char} \mid \text{bool}$

$\langle IDS \rangle \rightarrow \text{id} \mid \langle IDS \rangle, \text{id}$

➤ 输入：

$\text{int } a, b, c ;$

如何根据语法规则为
输入句子构造分析树？





第一章 绪论

语义分析概述

哈尔滨工业大学 陈鄞



编译器的结构



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)

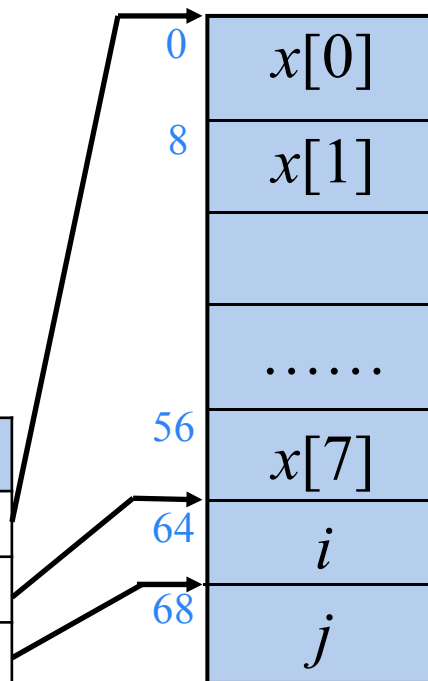
➤ 类型 (Type)

➤ 存储位置、长度

例:

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

名字	相对地址
<i>x</i>	0
<i>i</i>	64
<i>j</i>	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		
⋮	⋮	⋮	⋮	⋮

字符串表

S	I	M	P	L	E	S	Y	M	B	L	E	T	A	B	L	E		
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	--	--

TAIL


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



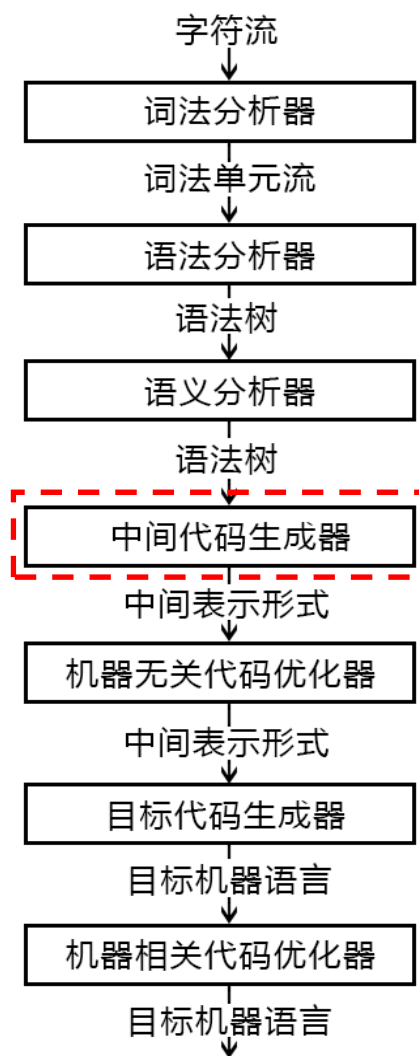
第一章 绪论

中间代码生成及 编译器后端概述

哈尔滨工业大学 陈鄞



编译器的结构



常用的中间表示形式

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)

语法结构树/语法树 (Syntax Trees)

常用的三地址指令

序号	指令类型	指令形式
1	赋值指令	$x = y \text{ op } z$ $x = \text{op } y$
2	复制指令	$x = y$
3	条件跳转	if $x \text{ relop } y$ goto n
4	非条件跳转	goto n
5	参数传递	param x
6	过程调用	call p, n
7	过程返回	return x
8	数组引用	$x = y[i]$
9	数组赋值	$x[i] = y$
10	地址及指针操作	$x = \& y$ $x = *y$ $*x = y$

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*)

三地址指令的表示

➤ 四元式 (Quadruples)

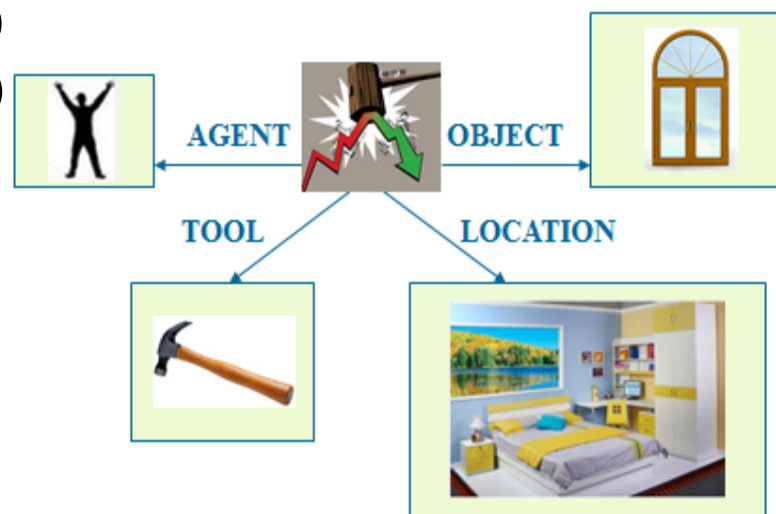
➤ (op, y, z, x)

➤ 三元式 (Triples)

➤ 间接三元式 (Indirect triples)

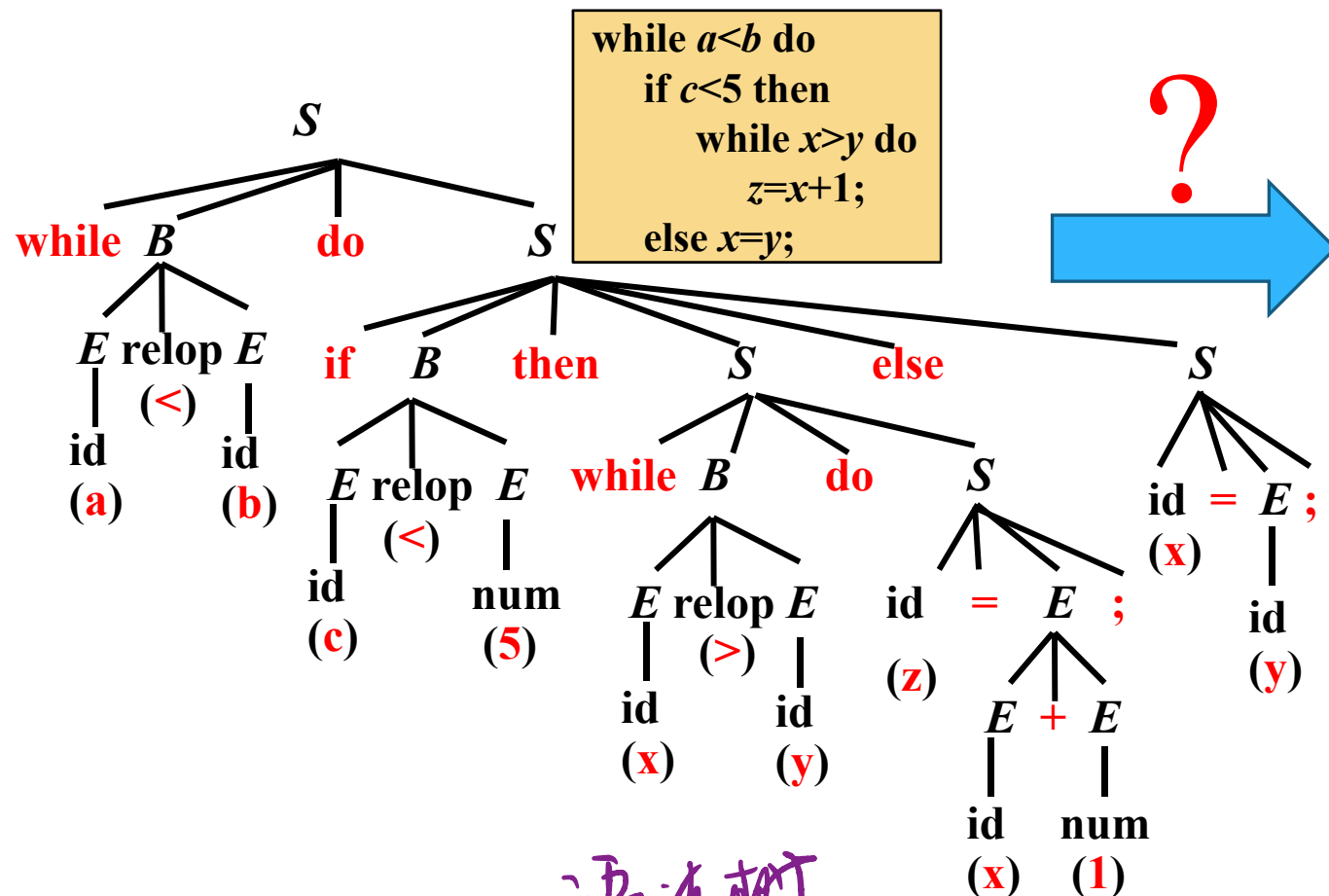
三地址指令的四元式表示

- $x = y \text{ op } z$ (**op** , y , z , x)
- $x = \text{op } y$ (**op** , y , $_$, x)
- $x = y$ (**=** , y , $_$, x)
- if $x \text{ relop } y$ goto n (**relop** , x , y , n)
- goto n (**goto** , $_$, $_$, n)
- param x (**param** , $_$, $_$, x)
- call p, n (**call** , p , n , $_$)
- return x (**return** , $_$, $_$, x)
- $x = y[i]$ (**=[]** , y , i , x)
- $x[i] = y$ (**[]=** , y , x , i)
- $x = \&y$ (**&** , y , $_$, x)
- $x = *y$ (**=*** , y , $_$, x)
- $*x = y$ (***=** , y , $_$, x)



三地址指令序列唯一确定了
运算完成的顺序

中间代码生成的例子

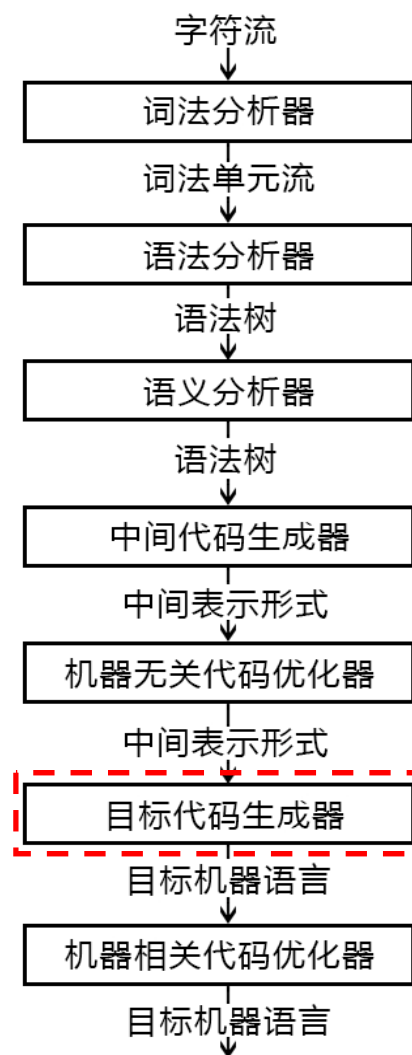


100: ($j <$, a, b , 102)
 101: (j , -, -, 112)
 102: ($j <$, c , 5, 104)
 103: (j , -, -, 110)
 104: ($j >$, x, y , 106)
 105: (j , -, -, 100)
 106: (+, x , 1, t_1)
 107: (=, t_1 , -, z)
 108: (j , -, -, 104)
 109: (j , -, -, 100)
 110: (=, y , -, x)
 111: (j , -, -, 100)
 112:

编译器的结构

➤ 目标代码生成以源程序的中间表示形式作为输入，并把它映射到目标语言

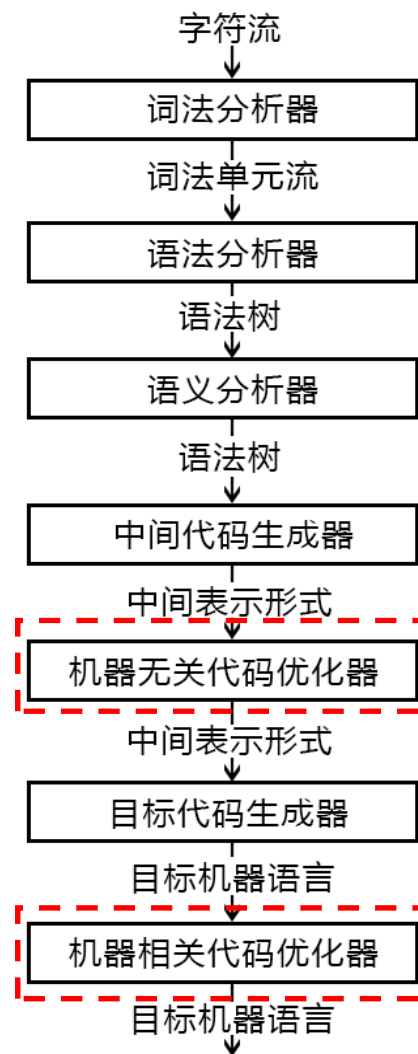
➤ 目标代码生成的一个重要任务是为程序中使用的变量合理分配寄存器



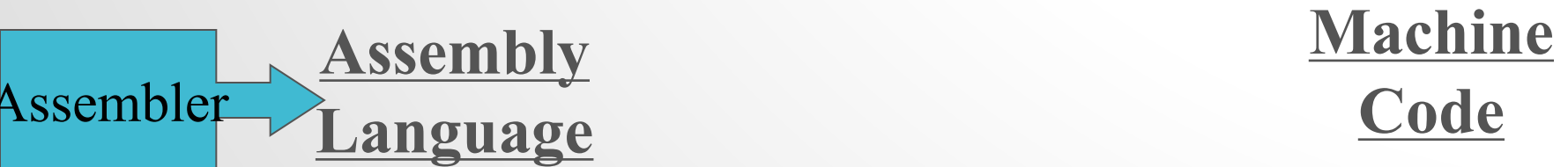
编译器的结构

3 代码优化

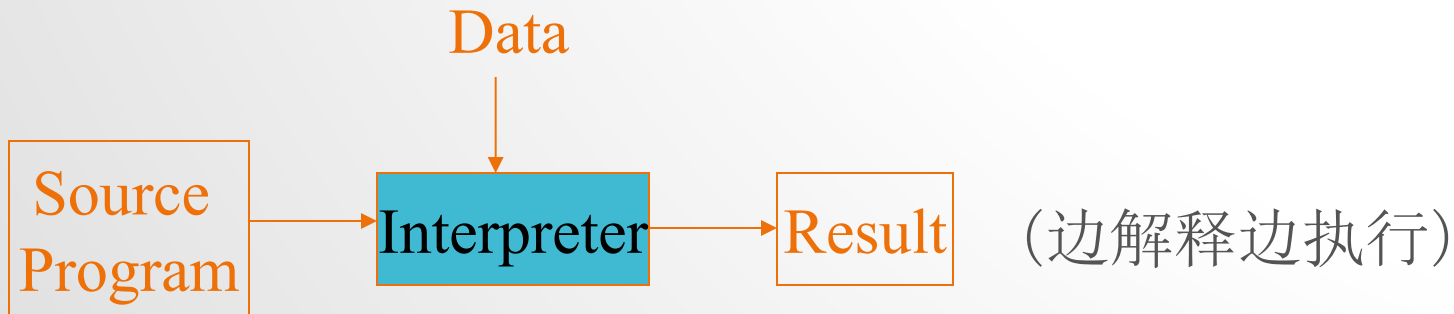
- 为改进代码所进行的等价程序变换，使其运行得更快一些、占用空间更少一些，或者二者兼顾



编译程序和汇编程序的区别：



翻译程序和解释程序的区别：



1.3 编译程序的组成

将 “I wish you success” 译成中文的过程：

1、词法分析： 单词： I, wish, you, success

词性：代， 动， 代， 名

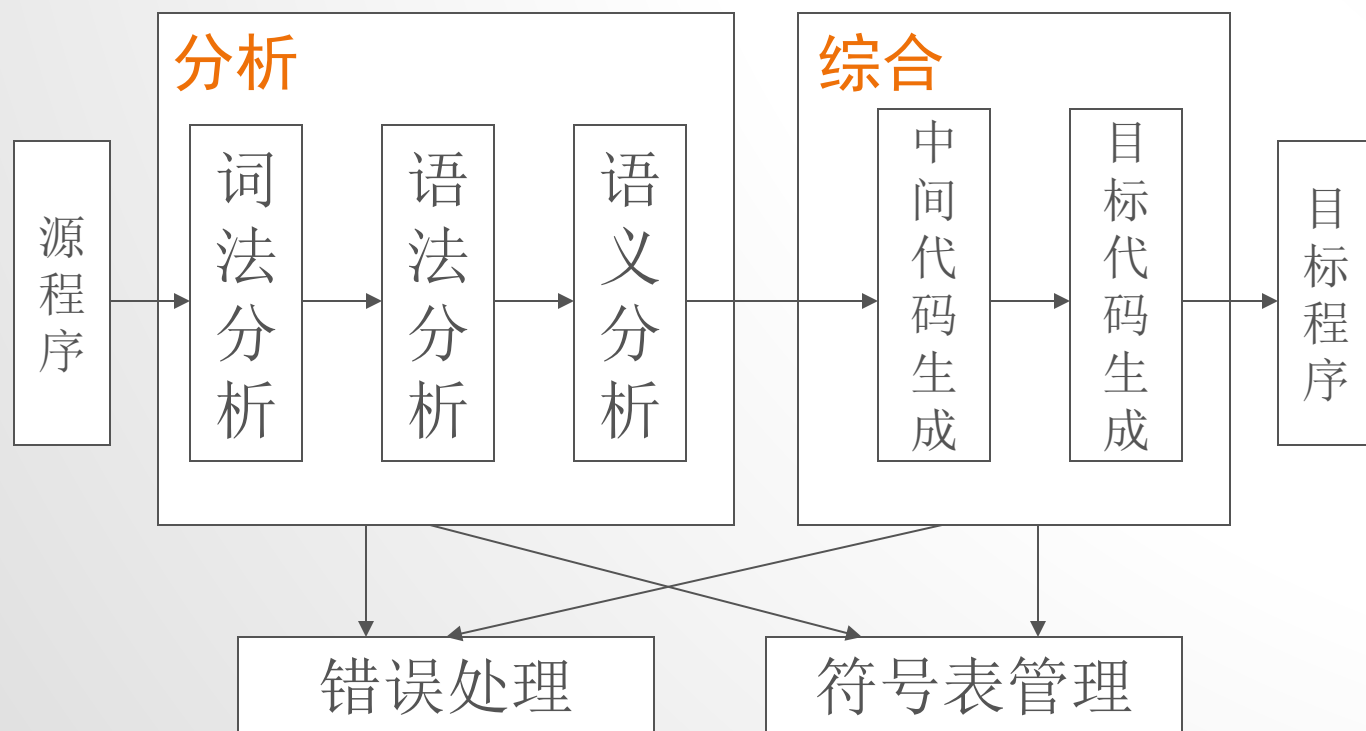
2、语法分析：语法单位：主， 谓， 间宾， 直宾

3、语义分析：语义： 我， 祝愿， 你， 成功

4、中间形式生成：我祝愿你成功

5、目标生成：祝你成功

一个典型的编译程序应具有以下的组成：



编译程序结构(components)

- 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
- 符号表管理程序
- 出错处理程序

1. 词法分析

任务：输入源程序；扫描，分解字符串，识别出一个个单词（定义符，标识符，运算符，界符，常数）


依循的规则：语言的构词规则

所做转换：

源程序字符串



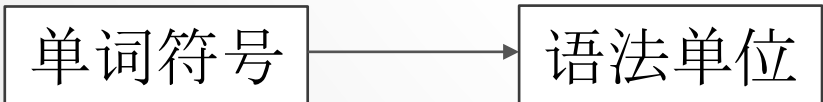
单词符号

例：Do 150 I=1,100  基本字：Do
标号：150
标示符：I
.....

2. 语法分析

任务：在词法分析基础上，将单词符号串转化为语法单位（短语，子句，句子，程序段，程序），并确定整个输入串是否构成语法上正确的程序

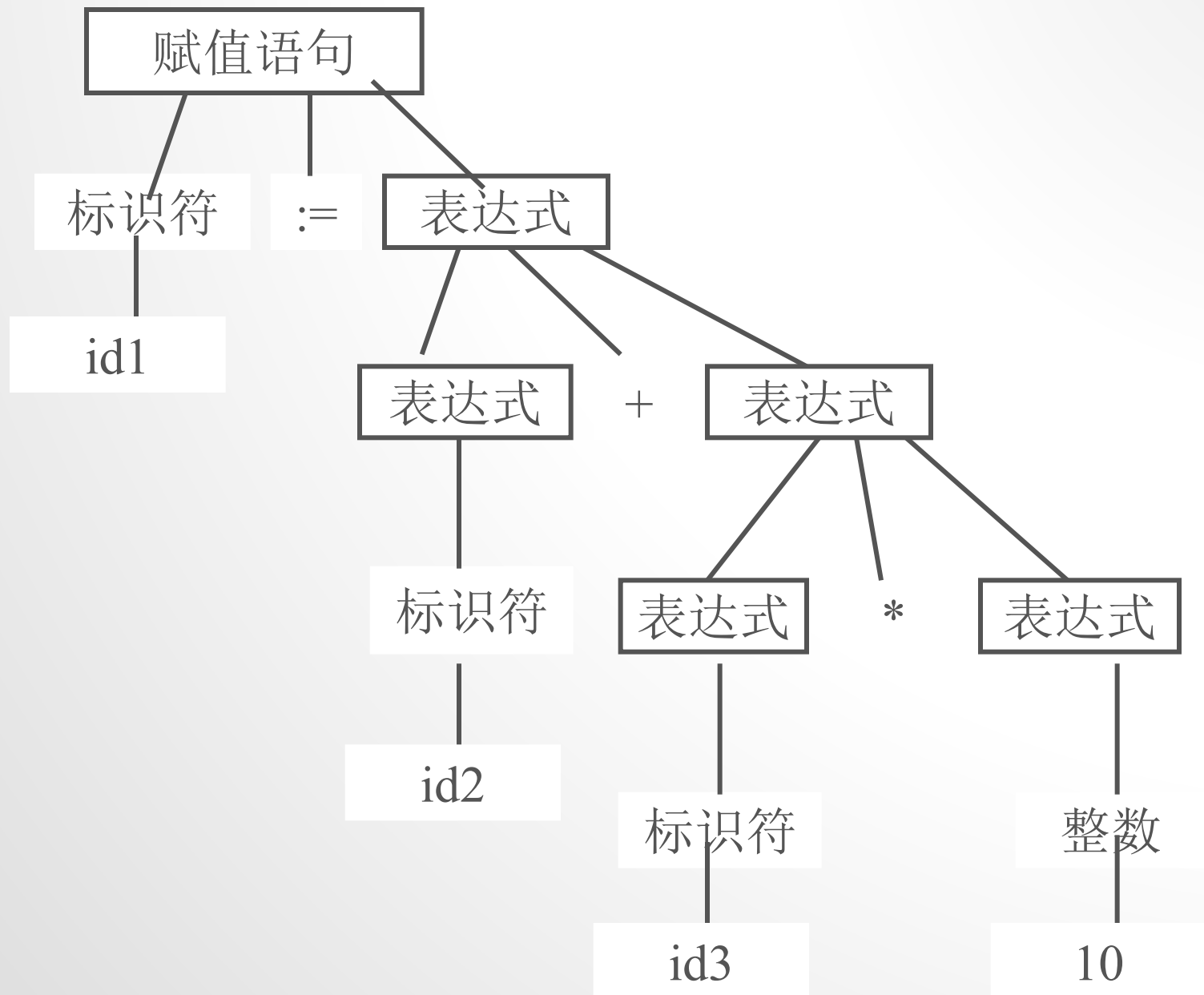
依循的规则：语法规则

所做转换：

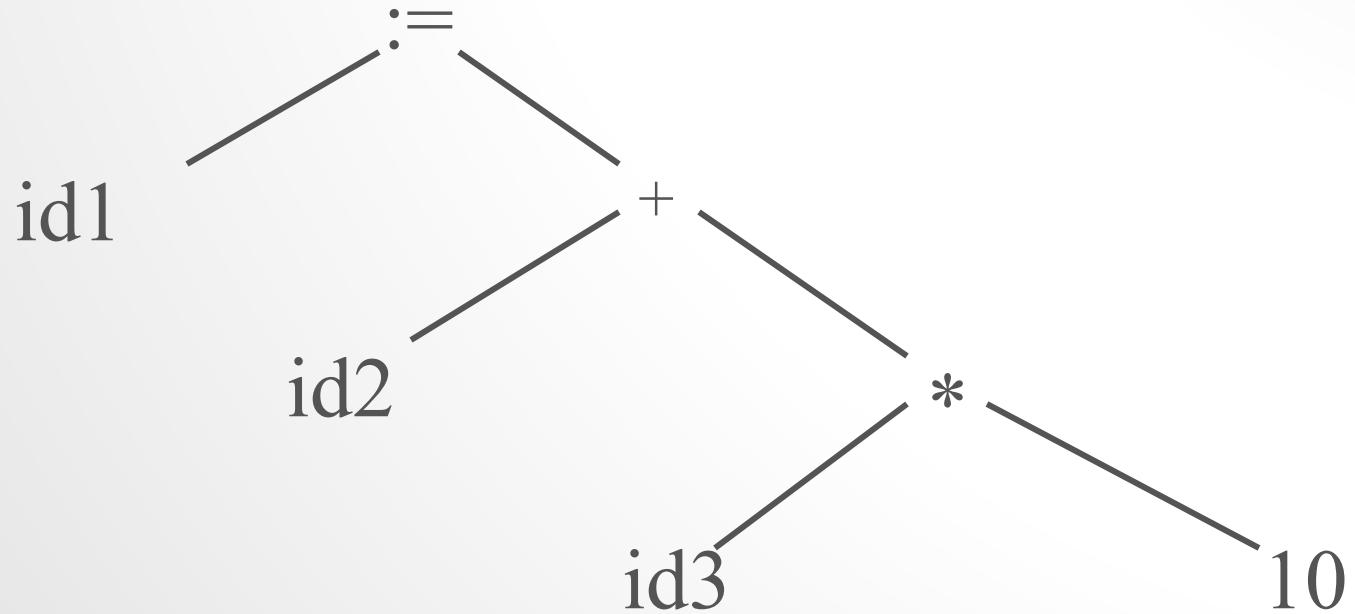
语法单位：算术表达式、短语、句子、子程序.....

例：“ $x+5y$ ”：算术表达式

语句ID1:=ID2+ID3*10的语法树



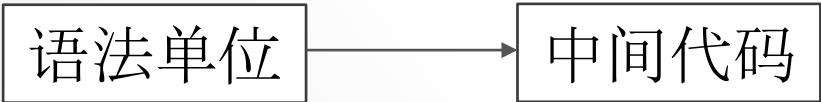
$\text{id1} := \text{id2} + \text{id3} * 10$



3. 语义分析

任务：对语法分析所识别的各种语法成分的意义(即语义)进行确定并进行初步翻译（产生中间代码）

依循的规则：语义规则

所做转换：

例，“+”：加 “*”：乘

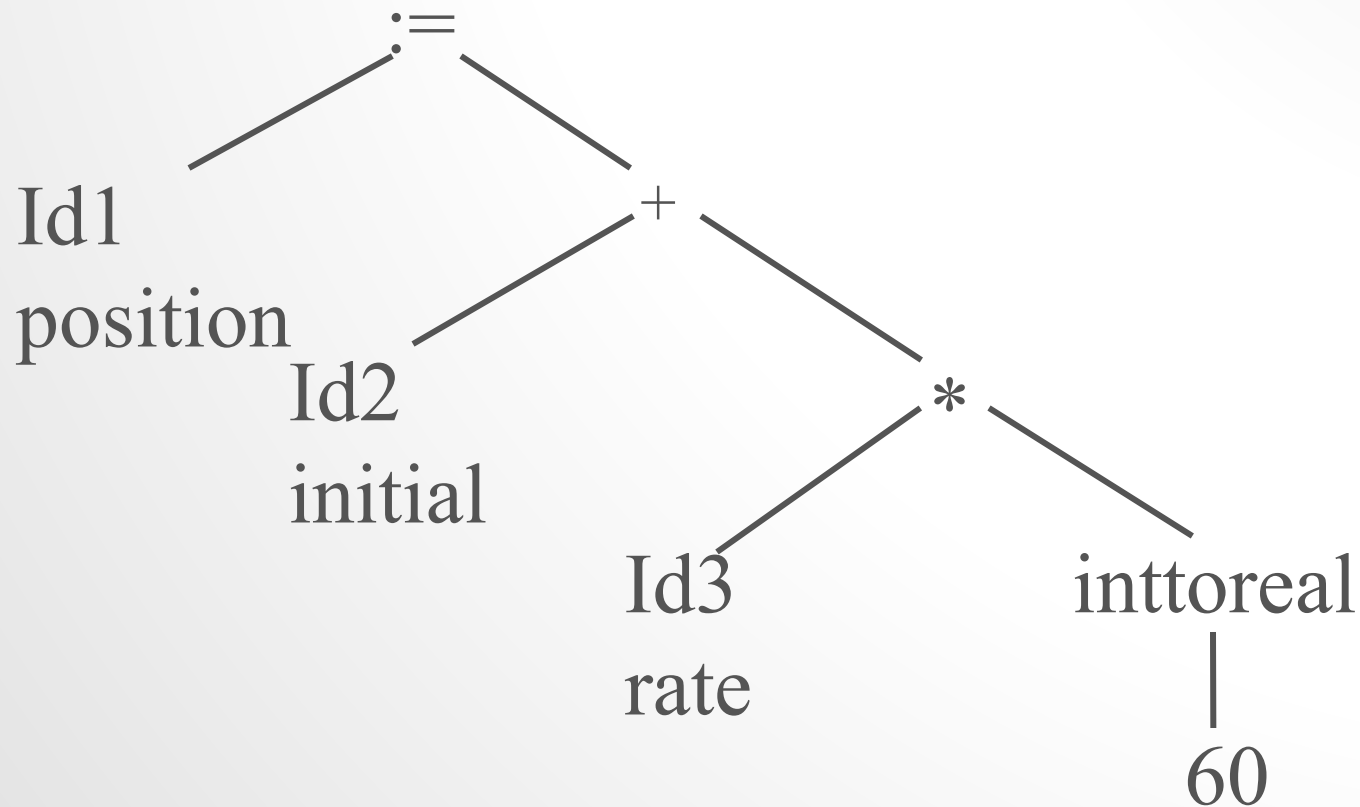
3. 语义分析

语义审查(静态语义)

- 上下文相关性
- 类型匹配
- 类型转换

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


语义分析



4. 中间代码生成

任务：从源程序的树形或其它形式，产生源程序的中间代码。

中间代码：四元式、三元式.....


例： $a := b + c$  四元式

op	arg1	arg2	result
+	b	c	T1
:=	T1		a

5. 代码优化

任务：对于代码（主要是中间代码）进行加工变换，以期能够产生更高效（节省时间和空间）的目标代码。

依循的规则：程序等价变换规则

所做转换：

```
graph LR; A[中间代码] --> B[中间代码（优化后）]
```


5. 代码优化

id1:= id2 + id3 * 60

(1) (inttoreal 60 - t1)

(2) (* id3 t1 t2)

(3) (+ id2 t2 t3)

(4) (:= t3 - id1)

变换 \Rightarrow

(1) (* id3 60.0 t1)

(2) (+ id2 t1 id1)

5. 代码优化

$t1 = b * c$

$t2 = t1 + 0$

$t3 = b * c$

$t4 = t2 + t3$

$a = t4$



$t1 = b * c$


$t2 = t1 + t1$

$a = t2$

6. 目标代码生成

任务：将中间代码转换成汇编程序或者机器语言。

依循的规则：硬件体系结构，指令系统

所做转换：

```
graph LR; A[中间代码] --> B[目标代码]
```

目标代码形式：绝对指令，汇编指令等

6. 目标代码生成

(* , id3 60.0 t1)

(+ , id2 t1 id1)



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

7、符号表管理

- 符号表管理是一个贯穿编译全过程的工作。
- 编译程序在分析源程序时，需要记录标识符的各种属性信息；
- 在语义分析和代码生成阶段，还要对建立的符号表进行检索，提取相应的属性信息。--类型、作用域、分配存储信息

Const1 常量 值: 35

Var1 变量 类型: 实 层次: 2

7、符号表管理

编译程序涉及的表格有：

符号名表

循环表

常数表

等价名表

标号表

公用链表

入口名表

格式表

过程引用表

中间代码表

8. 出错处理

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

若能自动校正错误则更好，但其代价非常高

8. 出错处理

- 错误可发生在编译的各个阶段，错误处理也是贯穿编译全过程。
- 词法分析阶段可查出的错误，如标识符的组成不符合词法规则；
- 语句结构错误是在语法分析中可查出的错误；
- 语义分析阶段可查出的错误，即结构正确，但所涉及的操作无意义或错误。
- 在编译时查出的，叫Comple-time error，在运行时表现才表现出来的错误叫Run-time error。

9. 前端和后端

前端包括编译逻辑结构中的分析部分，即词法分析、语法分析、语义分析和中间代码生成，除此还包括符号表建造及相应分析中的错误处理以及与机器无关的优化部分。

后端包括与目标机有关的部分，即综合部分，它包括目标代码生成及生成期间对符号表的相应检索操作和错误处理操作，以及与机器相关的代码优化部分。

将编译系统划分为前后端，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。

10. 遍(pass)

对源程序(或其中间形式)从头至尾扫描一次并进行有关加工处理，生成新的中间形式或最终目标程序，称为一遍。

分遍原则：

- ①目标质量高低(高则多遍)
- ②机器内存大小(小则多遍)
- ③源语言简繁(繁则多遍)
- ④设计人员多少(多则多遍)

优缺点：

11. 编译程序的生成

实现工具： Low-level Language ;
High-level Language ;
Automatic Builder。

生成方法： Self-compiler;
transplant