编译原理及实现技术

主讲：张鹏

Email：zhangpengccst@jlu.edu.cn

qq:809348397

邀请码： 26792075

1. 要求

- 掌握相关知识
- 学习方式
- 随时提问

2. 课程的意义

- 加深对高级程序设计语言的理解和认识
- 提高程序（尤其是大型）的设计能力
- 编译技术可以应用到许多实际开发工作中
- 可以培养抽象能力、形式化描述能力
- 是一种元级程序设计

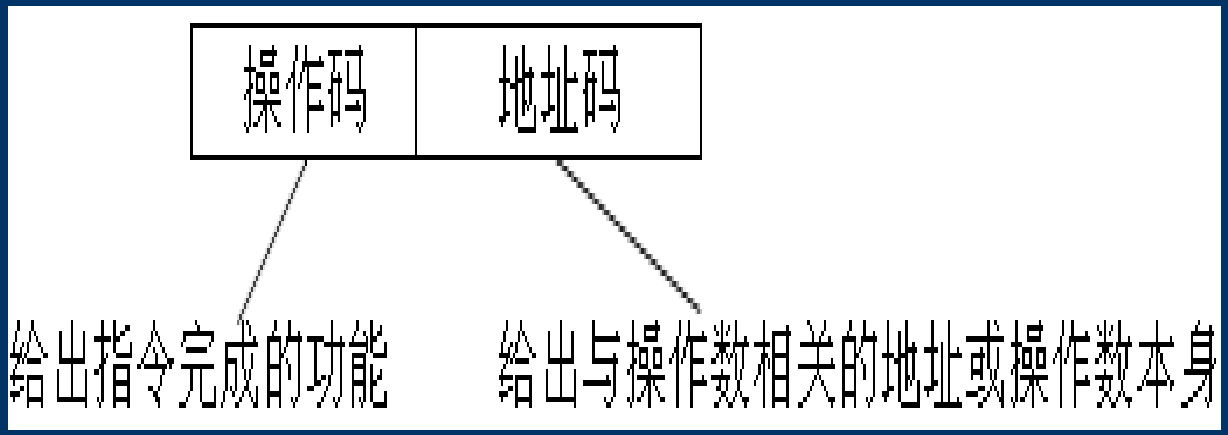
3. 程序设计语言的发展

第一代 机器语言：能够被计算机的硬件系统直接执行的指令程序，如“0001000101”。

第二代 汇编语言：将硬件指令用一些助记符表示，即符号化的机器语言，如“ADD，MOV”。

第三代 高级语言：从程序员的角度出发，对汇编语言进一步抽象，使用便于理解的“自然语言”表述。

（一）机器语言



- 某种CPU的指令1011011000000000，它表示让CPU进行一次加法操作；
- 指令1011010100000000则表示进行一次减法操作。

某种CPU指令编写如下程序片段：

- 1010 1001 0001 0110 0000 0001
- 0011 1100 0001 1000 0000 0001
- 0111 1100 0000 0101
- 0010 1101 0001 0101 0000 0000
- 1110 1010 0000 0011
- 0000 0101 0001 0101 0000 0000
- 0101 0011 0001 1000 0000 0001
- ……

  0000 0000 0000 0000

  0000 0000 0000 0000

$$Y = \begin{cases} x + 15 & x<y \\ x - 15 & \text{否则} \end{cases}$$

（二）汇编语言

用助记符(Memoni)代替操作码，用地址符号(Symbol)或标号(Label)代替地址码， 如ADD表示加法操作，SUB表示减法操作等等 。通常是为特定的计算机或系列a计算机专门设计的。

用Pentium汇编语言编程示例：

```
1.          MOV  X , AX
2.          CMP   AX , Y
3.          JL S1
4.          SUB AX ,15
5.          JMP S2
6.    S1:   ADD AX ,15
7.    S2:   MOV AX, Y
          ……..
          XDW
          YDW
```

$$Y = \begin{cases} x + 15 & x<y \\ x - 15 & \text{否则} \end{cases}$$

（三）高级语言

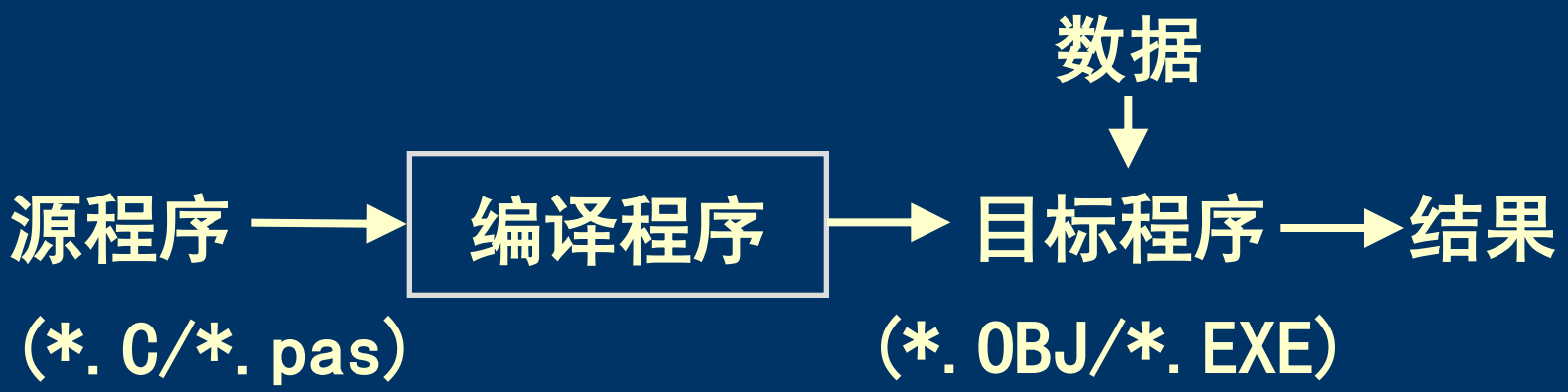
从程序员的角度出发，对汇编语言进一步抽象，使用便于理解的“自然语言”表述。

高级语言编程示例：

```
if      (X<Y )  then    Y:=X + 15
else    Y:=X - 15;
```

4. 高级语言的实现

- 编译方式：源语言为高级语言，目标语言是低级语言（汇编或机器语言）的翻译程序。



4. 高级语言的实现

- 解释方式：一边翻译一边执行，翻译完的同时也执行完了程序。

根据实际需求选择编译方式和解释方式

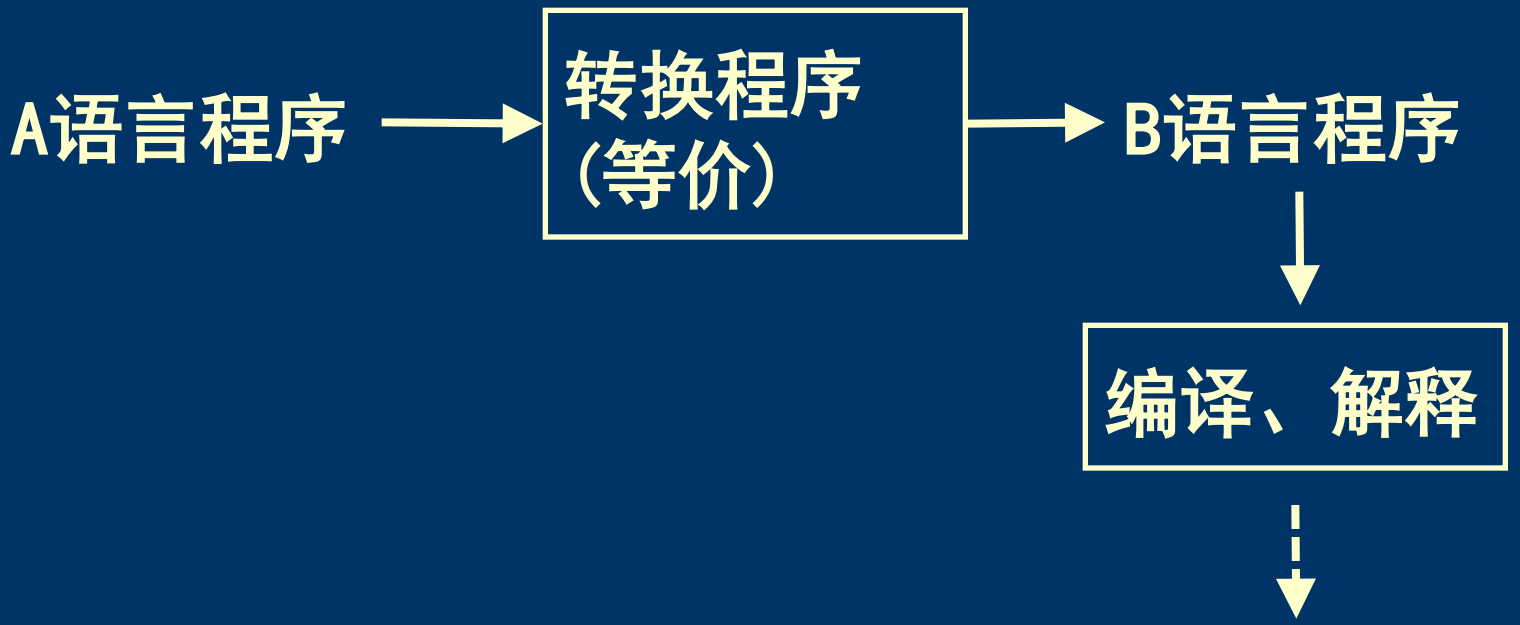
解释器和编译器的比较

- 解释器是执行系统，编译器是转换系统。
- 基于解释执行的程序可以动态修改自身，而基于编译执行的程序不易胜任。
- 基于解释方式有利于人机交互。
- 执行速度:解释器执行速度要慢。
- 空间开销：解释器需要保存的信息较多，空间开销大。
- 二者实现技术相似。

综上，大多数的高级语言采用编译方式。

4. 高级语言的实现

- 转换方式：是一种变通的方式，假如已有B语言的编译器，就可以把A语言程序转换为B语言程序，用B语言已有的编译器去编译执行。



5. 编译程序的组成

- 词法分析：识别由字符组成的高级语言程序中的单词，并将其转化成一种内部表示（TOKEN）的形式，同时检查是否存在词法错误。

  如有：x=100； 机器视角：'x' '=' '1' '0' '0' ';'

单词：符合某种规律的字符串

词法错误：7L

5. 编译程序的组成

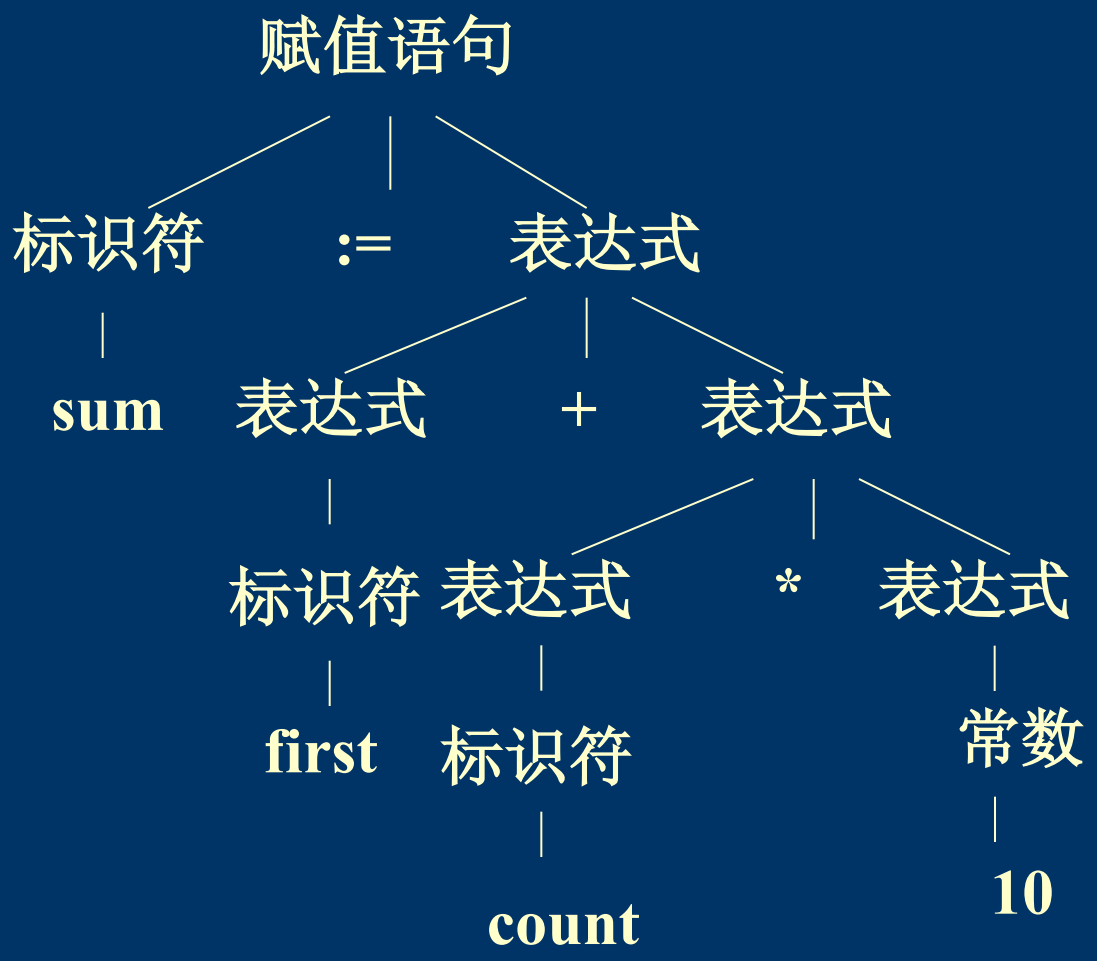
- 语法分析：根据语言定义的语法规则来验证程序中是否存在语法错误。

每个语言都有自己的语法，程序一定要对应这个语言的语法约定。

所谓语言，都包含：语法、语义、语用。

语法分析的示例

- sum:= first + count * 10



5. 编译程序的组成

- 语义分析：检查源程序有无语义错误，为代码生成阶段收集类型信息。
- 中间代码生成：将源程序转换成一种称为中间代码的内部表示形式，便于优化和移植。

语义分析的示例

```
VAR
      first: real;
      count：  char;
 BEGIN
      sum:=first + count * 10
 END.
```

词义错误：

1、sum有使用而无定义；

2、 count为字符类型变量不能进行乘法运算。

中间代码生成的示例

例：

```
VAR  sum,  first, count: real;
BEGIN
        sum:=first + count * 10
END.
```

生成如下四元式形式的中间代码序列：

1、（int-to-real, 10 , - ,t1)

2、（*, count , t1 ,t2)

3、（+, first , t2 ,t3)

4、（:=, t3 ,- , sum)

5. 编译程序的组成

- 中间代码优化：变换或改造中间代码，使生成的目标代码更为高效，即节省时间和空间。和程序算法的高效无关，更多的是针对于程序具体运行时的内部优化，尤其针对有特殊要求的编译器。

中间代码优化的示例

例：sum:=first + count * 10

生成如下四元式形式的中间代码序列：

1、（int-to-real, 10 , - ,t_1)
2、（*, count , t_1 ,t_2)
3、（+, first , t_2 ,t_3)
4、（:=, t_3 ,- , sum)

生成如下优化后的四元式形式的中间代码序列：

2、（*, count , 10.0 ,t_2)
3、（+, first , t_2 ,t_3)
4、（:=, t_3 ,- , sum)

5. 编译程序的组成

- 目标代码生成：将中间代码变换为特定机器上的机器指令代码或汇编指令代码。

例：sum:=first + count * 10
生成如下汇编代码：

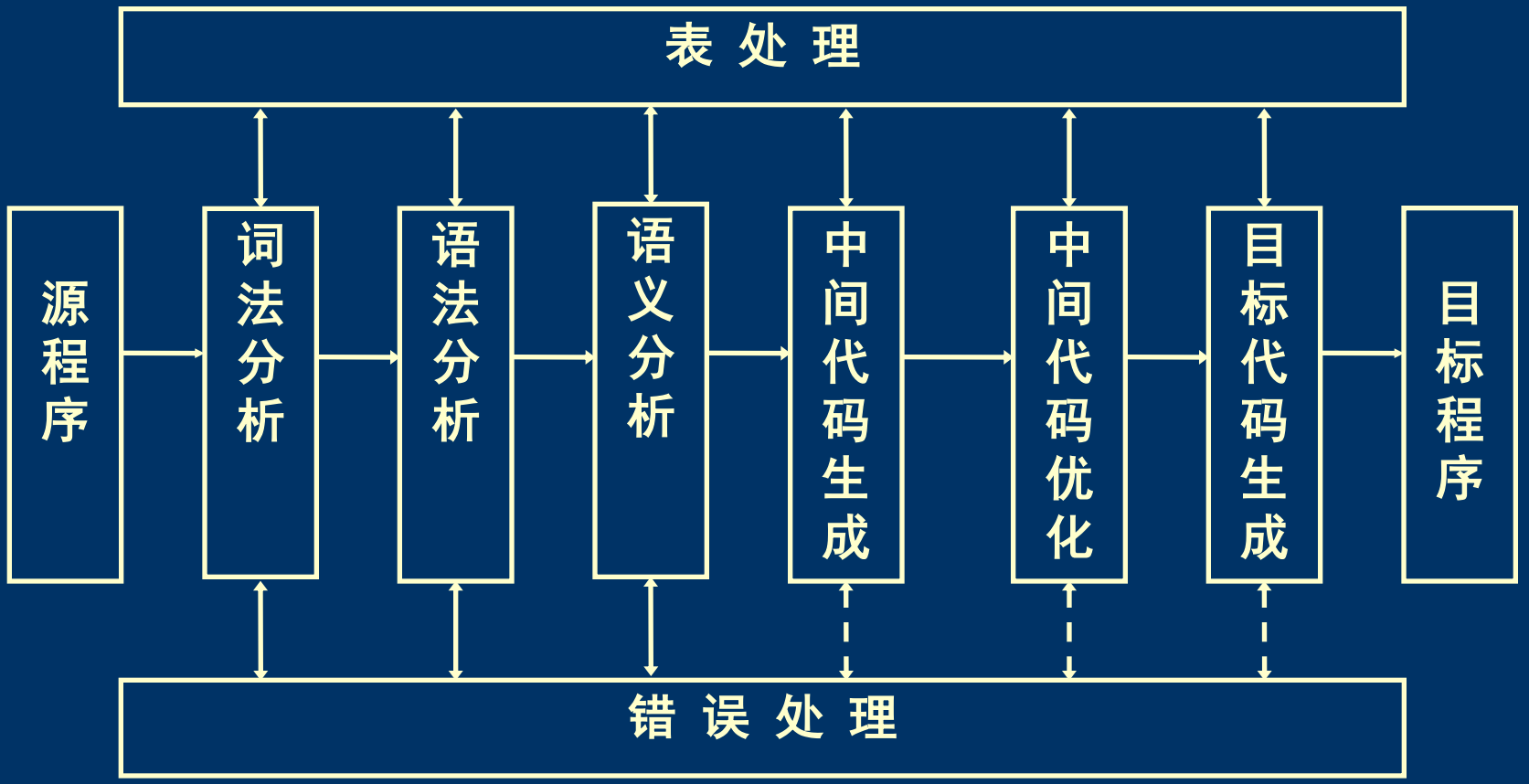
```
1. MOV   count , R1
2. MULT    R1 , #10.0
3. MOV   first , R2
4. ADD   R1, R2
5. MOV     R1,  sum
```

5. 编译程序的组成

此外，编译过程还有两部分工作贯穿始终。

- 错误处理：当编译阶段有错误出现时，由相应的错误处理模块给出解决方案，使得编译器能够继续进行下去。
- 表格管理：为了合理的管理（构造、查找、更新……）表格（符号表、类型信息表……），设立一些专门子程序称为表格管理程序。

5. 编译程序的组成



6. 编译程序的设计

- 编译程序的分遍

  所谓“遍”就是对源程序或源程序的中间表示形式从头到尾扫描一次，并作加工处理，生成新的中间结果或目标程序 ；

  分遍就是对源程序或源程序的中间表示形式从头到尾扫描几次。

a. 分遍的理由

b. 本课的分遍策略

6. 编译程序的设计

- 设计中要注意的问题

a) 准确的理解源语言

b) 确定编译要求

c) 确定目标语言

d) 分遍

e) 具体设计

7. 编译程序的自动生成

- 软件自动生成的难点
- 编译程序中可以自动生成的部分

1、基于代码大模型的编译器自动生成技术探究

2、新时代下编译技术的应用探究

3、新时代下编译器面临的机遇和挑战