# Compiler ?

# Compiler ?

- **定义1.1：**

将某一种程序设计语言写的程序翻译成等价的另一种语言的程序的程序，称之为编译程序(compiler)。

- ✓ 编译程序是**语言处理**系统；
- ✓ 编译程序是将高级**语言翻译**成机器语言的程序；
- ✓ 人、机交互的工具；
- ✓ ……

**源语言： (source language)**

源语言是用来编写源程序的语言，一般是汇编语言或高级程序设计语言。

**源程序： (source program)**

编译程序的输入程序称为源程序。

**目标语言：(target language)**

目标程序的描述语言称为目标语言。

**目标程序：(target program)**

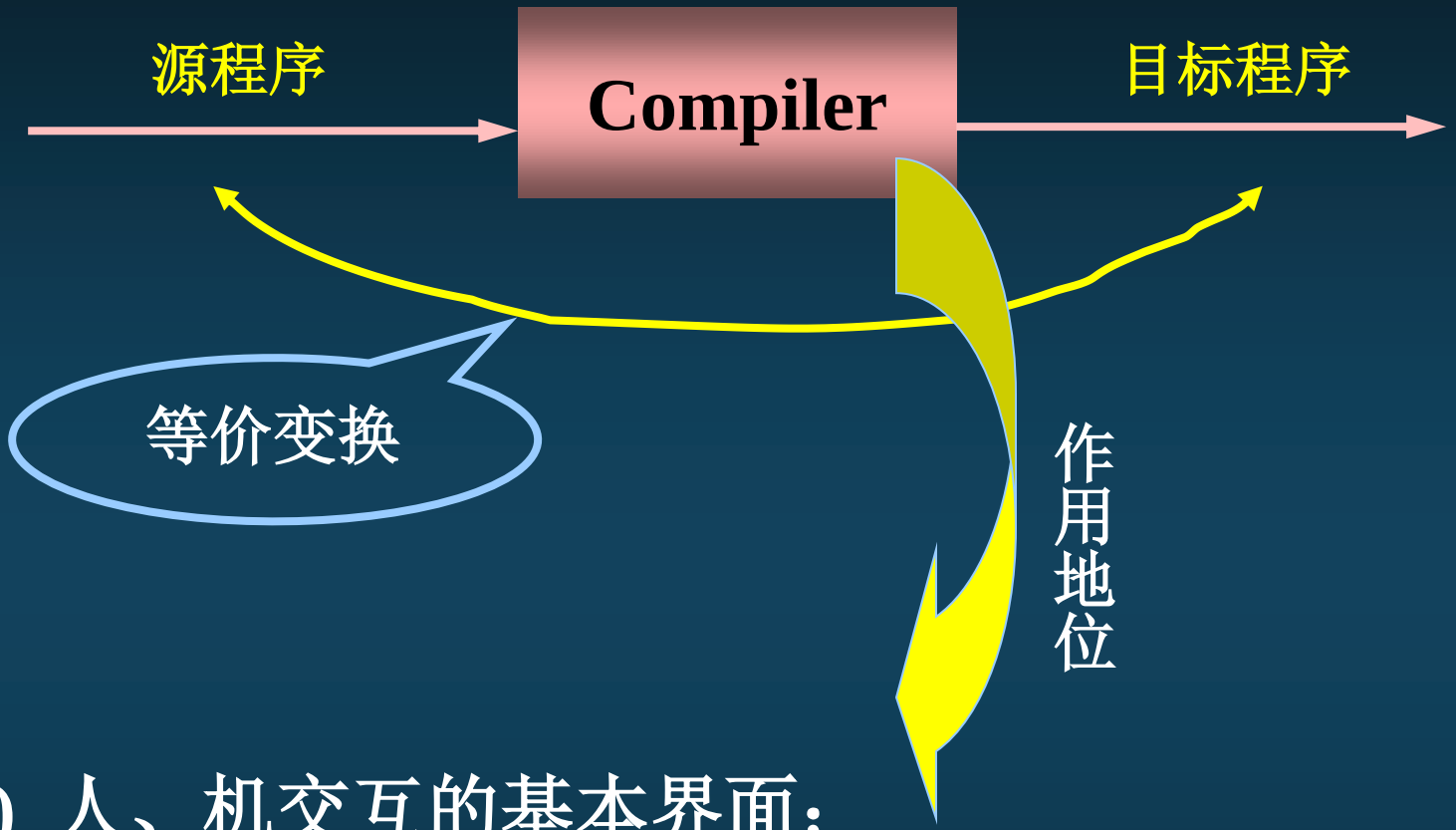
源程序经过编译后生成的程序称之为目标程序。

**宿主语言：**

编译程序的实现语言。

**宿主机（目标机）：**

编译程序的运行环境。




(1) 人、机交互的基本界面；

(2) 软件学科的重要分支，系统软件的重要组成部分；

(3) 用户直接关心的工具。

# 一. 编译程序的分类与执行

编译程序 { 解释程序 / 编译程序 }

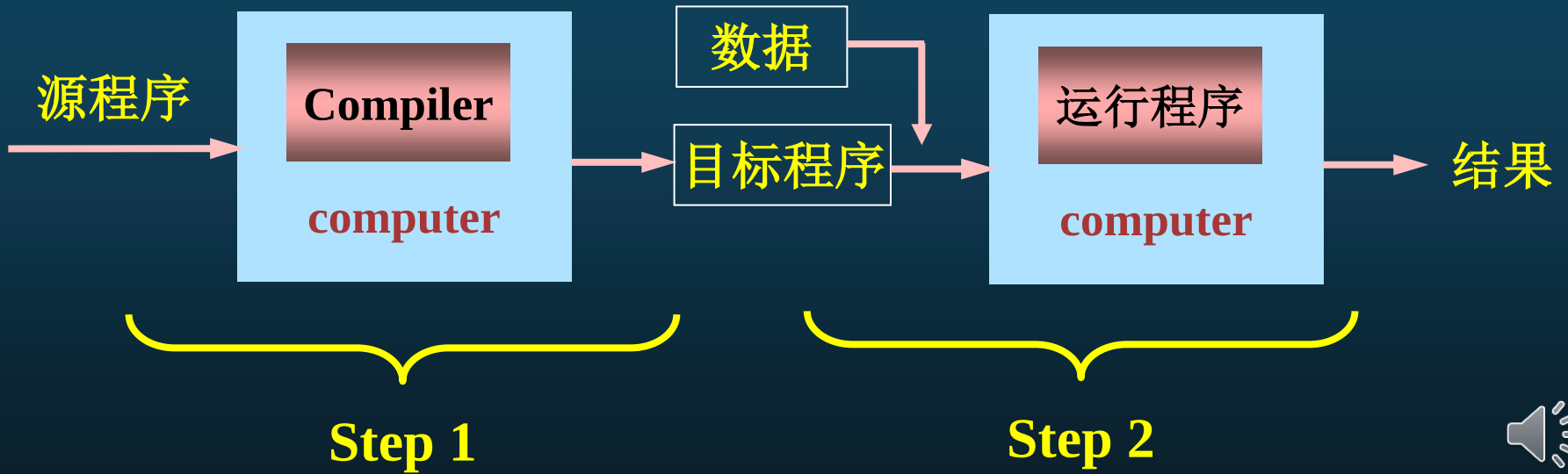
**解释程序：** 接受某语言的源程序将其直接翻译成目标代码且执行。

**编译程序：** 接受某语言的源程序将其直接翻译成等价的目标代码，然后执行且允许重复执行。

## 解释执行：


源程序
解释程序
computer
结果

## 编译执行：


源程序
Compiler
computer
数据
目标程序
运行程序
computer
结果
Step 1
Step 2

- 编译执行和解释执行的区别：

编译执行是由编译程序生成一个与源程序等价的目标程序，它可以完全取代源程序，该目标程序可以运行任意次。

解释执行不生成目标程序，仅是对源程序逐句解释逐句执行。

编译执行类似于自然语言的**笔译**；解释执行类似自然语言的**口译**。

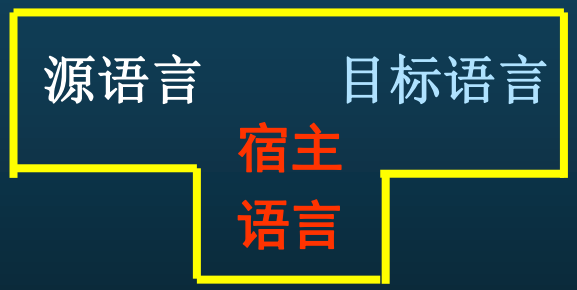
# 二. 编译程序的表示

1. 函数表示 ：

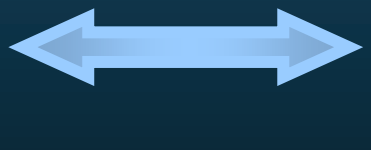
$$T = C(S)$$

T: 目标程序　　C: 编译程序　　S: 源程序

2. T型图表示 (体现编译程序的三个要素)

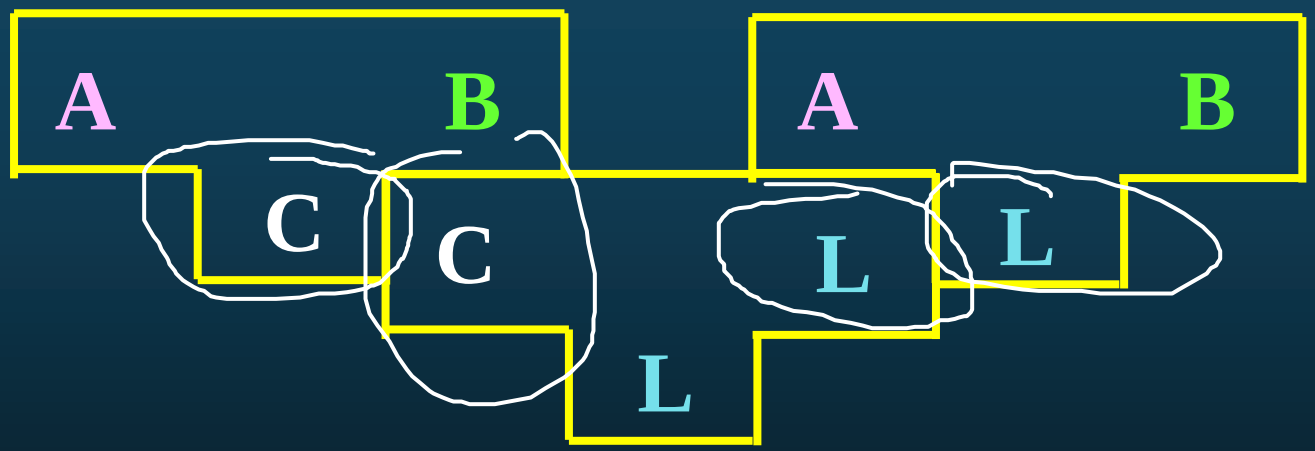



3. 符号表示



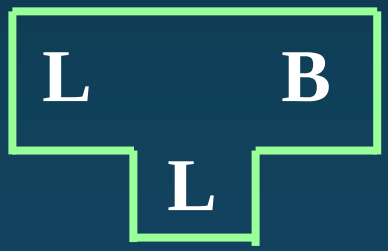
T型图的左上角表示源语言，右上角表示目标语言，而其底部表示实现语言。

- T型图联立表示 (例参见P3)

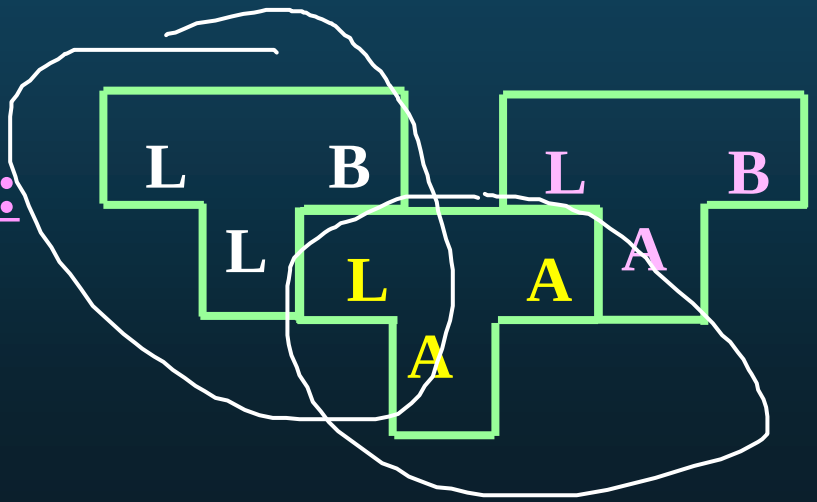


**例1.1:** 将**A**机器上已经存在的**L**语言的编译程序移植到**B**机器上。

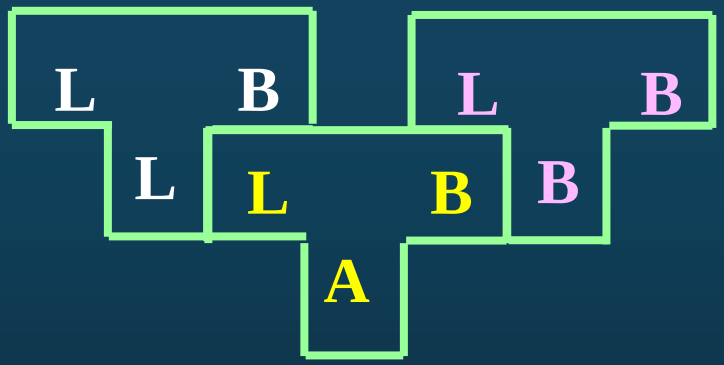
**Step1:**



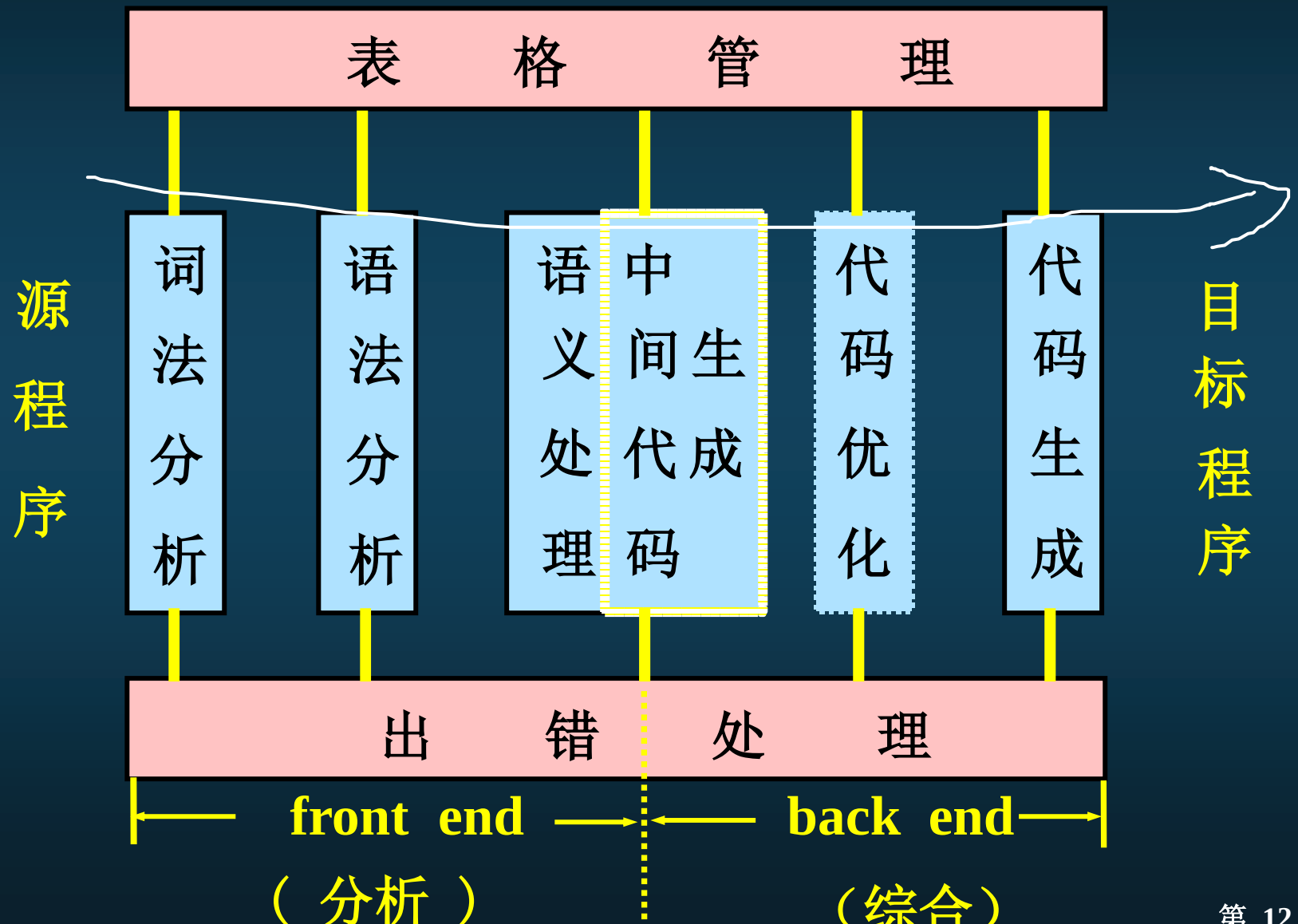

**Step2:**




**Step3:**



# 编译程序的逻辑结构 (经典划分)



## 1. 词法分析 (lexical analysis)

词法分析阶段的任务是对输入的符号串形式的源程序进行最初的加工处理。 它依次**扫描**读入的源程序中的每个字符, **识别**出源程序中有独立意义的源语言单词, 用某种特定的数据结构对它的属性予以表示和标注。词法分析实际上是一种线性分析。属性字的数据结构可据不同语言及编译程序实现方案来设计, 但一般设计成单词属性标识及单词内码两个数据项。

**例1.2:** 有如下C代码行 ：

```
a[index] = 12 * 3 ;
```

经词法分析后将产生对9个识别的单词， 如下属性标注:

源程序

属性字流

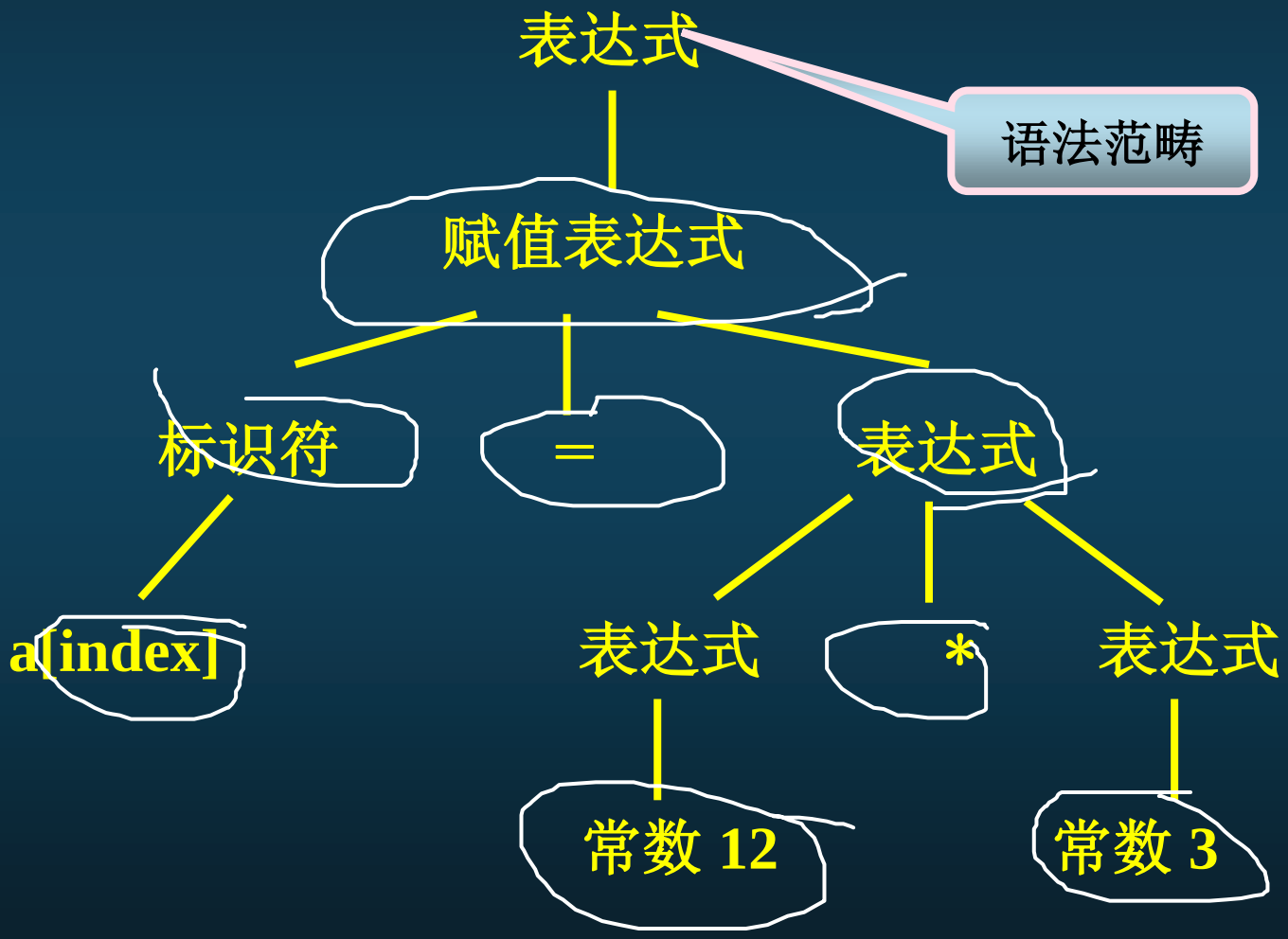
| | | |
|---|---|---|
| (1) | a | 标识符 |
| (2) | [ | 左方括号 |
| (3) | index | 标识符 |
| (4) | ] | 右方括号 |
| (5) | = | 赋值 |
| (6) | 12 | 整常数 |
| (7) | * | 乘号 |
| (8) | 3 | 整常数 |
| (9) | ; | 分号 |

## 2. 语法分析 (Syntax analysis)

语法分析的任务是依据语言文本所规定的语法规则, 对词法分析的结果进行语法检查, 并识别出单词序列所对应的语法范畴 。通常将语法分析的过程结果表示为分析树(parse tree)或语法树(syntax tree), 它是一种层次结构的形式。

例1.2中的C语言的赋值表达式语句经过语法分析，确认合乎C语言的语法规则且语法范畴为表达式语句, 产生的语法树形式的语法分析结果如下图所示。

# C语句 a[index] = 12 * 3 ； 的语法树

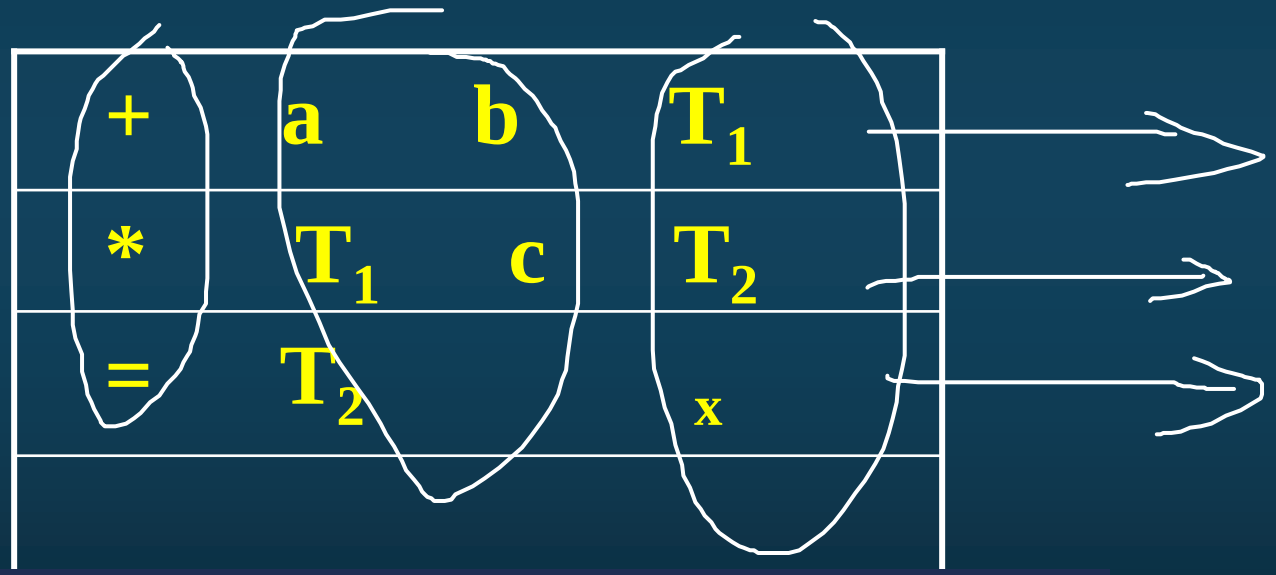


## 3. 语义分析 (semantic analysis)与中间代码生成

语义分析阶段的任务是依据语言文本限定的语义规则对语法分析所识别的语法范畴进行语义检查和处理, 直至最后翻译成与其等价的某种中间代码或目标代码。语义分析是整个编译程序完成的最实质性的翻译任务。

例1.3:　设有赋值语句　x=(a+b)*c

产生的四元式形式的中间代码：



## 4. 代码优化 (optimization)

代码优化是为了改进目标代码的质量而在编译过程中进行的工作。代码优化可以在中间代码或目标代码级上进行 ，其实质是在不改变源程序语义的基础上对其进行加工变换， 以期获得更高效的目标代码。而高效一般是对所产生的目标程序在运行时间的缩短和存贮空间的节省而言。

# 例1.4: 设有语句

```
for (i=1; i<=200; i++)
  {   y=y*i+y;
      x=x1/cos(x2)
  }
```

优化后

```
x=x1/cos(x2);
for (i=1; i<=200; i++)
  {
     y=y*i+y;
  }
```

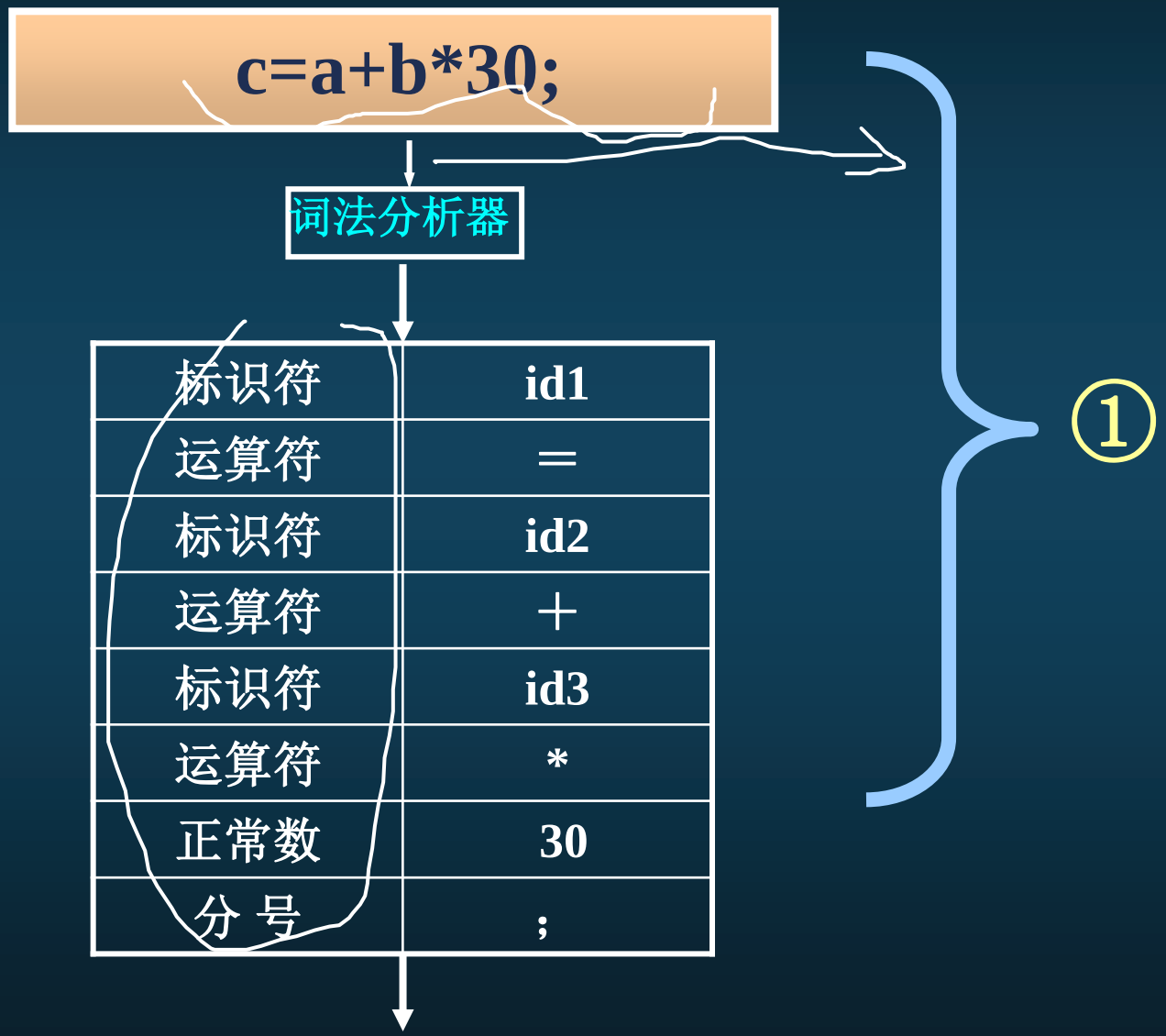
## 5. 目标代码生成 (code generator)

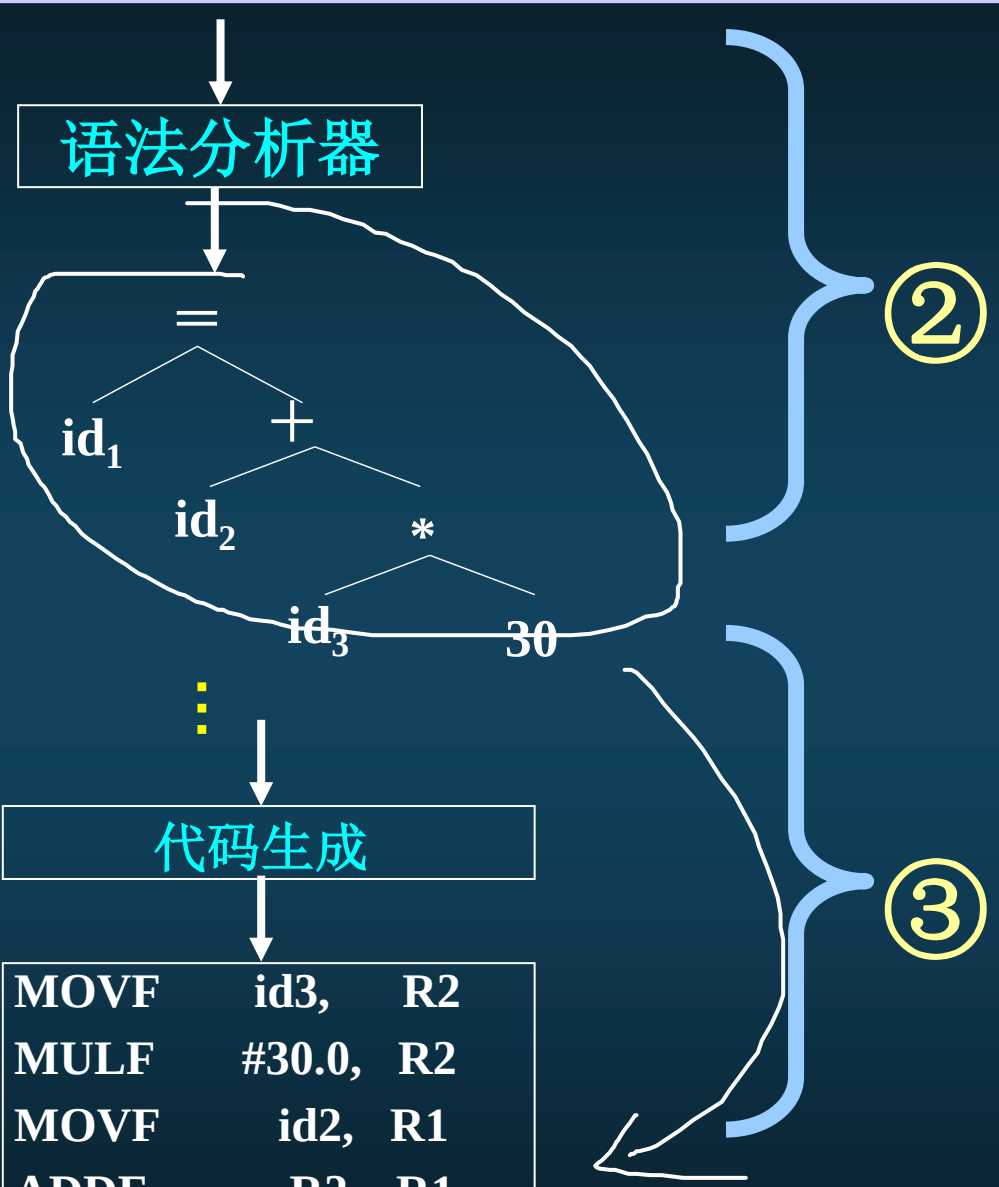
根据中间代码及编译过程中产生的各种表格的有关信息, 最终生成所期望的目标代码程序。一般为特定机器的机器语言代码或汇编语言代码，需要充分考虑计算机硬件和软件所提供的资源，以生成较高质量的目标程序。

对例1.2的语句生成代码。要考虑怎样存储整型数来为数组元素的引用生成目标代码，以假设的汇编语言描述如下：

```
MOV    R0,  index        ;; 索引值赋给寄存器R0
MUL    R0,  4            ;; 存储按字节编址，整型数
                         ;; 占4个字节
MOV    R1,  &a           ;; &a表示数组a的基地址
ADD    R1,  R0           ;; 计算a[index]的地址
MOV   [R1],  36          ;; a[index] = 36
```

# 例1.5:　设有语句　(P7)




语法分析器
=
id1
+
id2
*
id3
30
②
③
代码生成
MOVF id3, R2
MULF #30.0, R2
MOVF id2, R1
ADDF R2, R1
MOVF R1, id1

## ☺ 编译阶段与自然语言的对比

| 编译程序 | | 自然语言翻译 |
|---|---|---|
| 词法分析 | ⟺ | 查字典、识别单词 |
| 语法分析 | ⟺ | 分析识别句子 |
| 产生中间代码 | ⟺ | 翻译初稿 |
| 代码优化 | ⟺ | 对初稿润色修饰 |
| 目标代码生成 | ⟺ | 译文定稿 |