



同济大学软件学院

School of Software Engineering, Tongji University

编译原理

授课人：高珍

gaozhen@tongji.edu.cn

济事楼 514办公室

课程简介

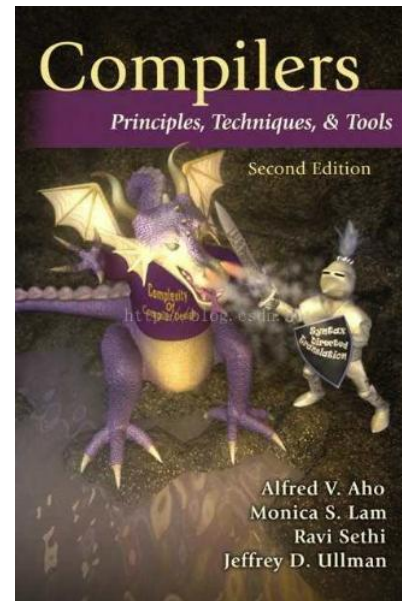
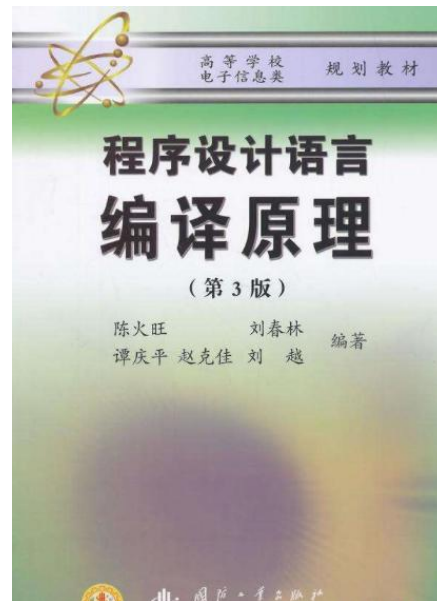
课程内容

- 介绍编译器构造的一般原理和基本实现方法
- 包括的理论知识：形式语言和自动机理论、语法制导的定义和属性文法、类型论与类型系统、程序分析原理等
- 强调形式描述技术和自动生成技术
- 强调对编译原理和技术的宏观理解，不把注意力分散到枝节算法，不偏向于任何源语言或目标机器

课程简介

教材和参考书


- 程序设计语言 编译原理（第3版）陈火旺等编著(387 pages), 国防工业出版社
- Compilers Principles Techniques and Tools(2nd Edition); Alfred V. Aho等编著(1038 pages)



课程简介

课程考核

— 成绩组成

- 出席（10%）
 - 随堂练习（20%）
 - 编译器新技术发展报告（10%）
 - 期末考试（60%）
- 
- CANVAS

内容线索

- 什么叫编译程序
- 编译过程概述
- 编译程序的结构
- 编译程序的生成
- 总结

程序语言技术的发展

表示机器实际操作的
数字代码

机器语言

C7 06 0000 0002

表示在IBM PC 上使用的
Intel 8x86处理器将
数字2移至地址0 0 0 0
(16进制) 的指令

以符号形式给出指
令和存储地址的

汇编语言

MOV X, 2

类似于数学定义或
自然语言的简洁形
式编写程序的操作

高级语言

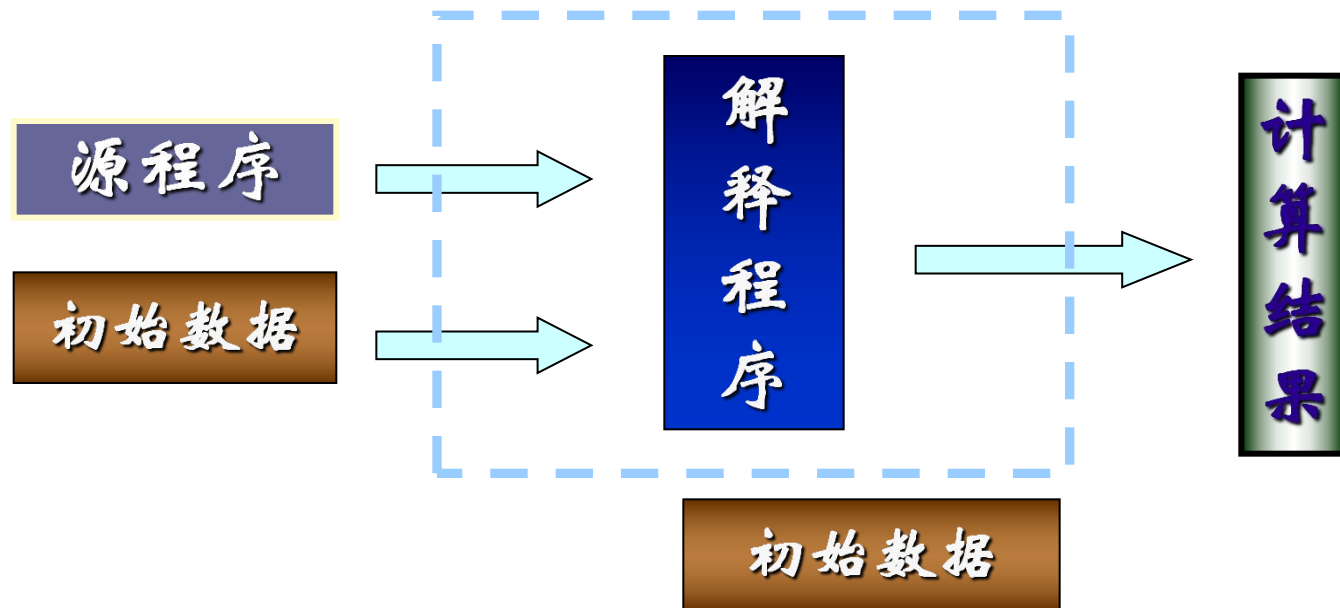
X=2

?

程序的两种执行方式

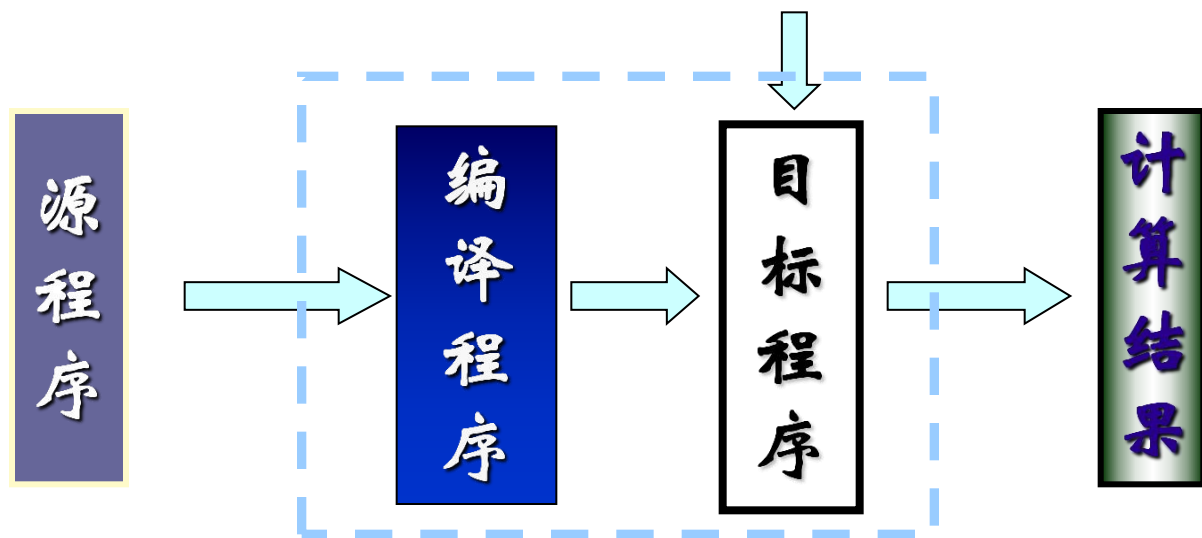
(1) 解释方式

解释程序：边解释边执行源语言程序，不产生目标语言程序。



(2) 编译方式

编译程序：能够把某种语言的程序转换成另一种语言的程序，而后者与前者在逻辑上是等价的。



程序的两种执行方式

- 高级语言程序通常采用**解释方式**和**编译方式**两种方式执行

解释方式

逐个语句地分析和执行

如Basic,Prolog,Shell,JavaScript

优点：易于查错

缺点：效率低，运行速度慢

编译方式

对整个程序进行分析，翻译成等价
机器语言程序后执行

如Pascal,Fortran,COBOL,C,C++

优点：只需分析和翻译一次，

缺点：在运行中发现的错误必须
查找整个程序确定

思考：Java属于什么类型的语言？

解释程序和编译程序的区别

	功能	工作结果
编译 程序 Compiler	源程序的一个 <u>转换</u> 系统	源程序的 <u>目标代码</u>
解释 程序 Interpreter	源程序的一个 <u>执行</u> 系统	源程序的 <u>执行结果</u>

- 简单的说，编译就是全文翻译，全部翻译完才执行。
解释就相当于同声翻译，边翻译边执行。

编译技术的发展

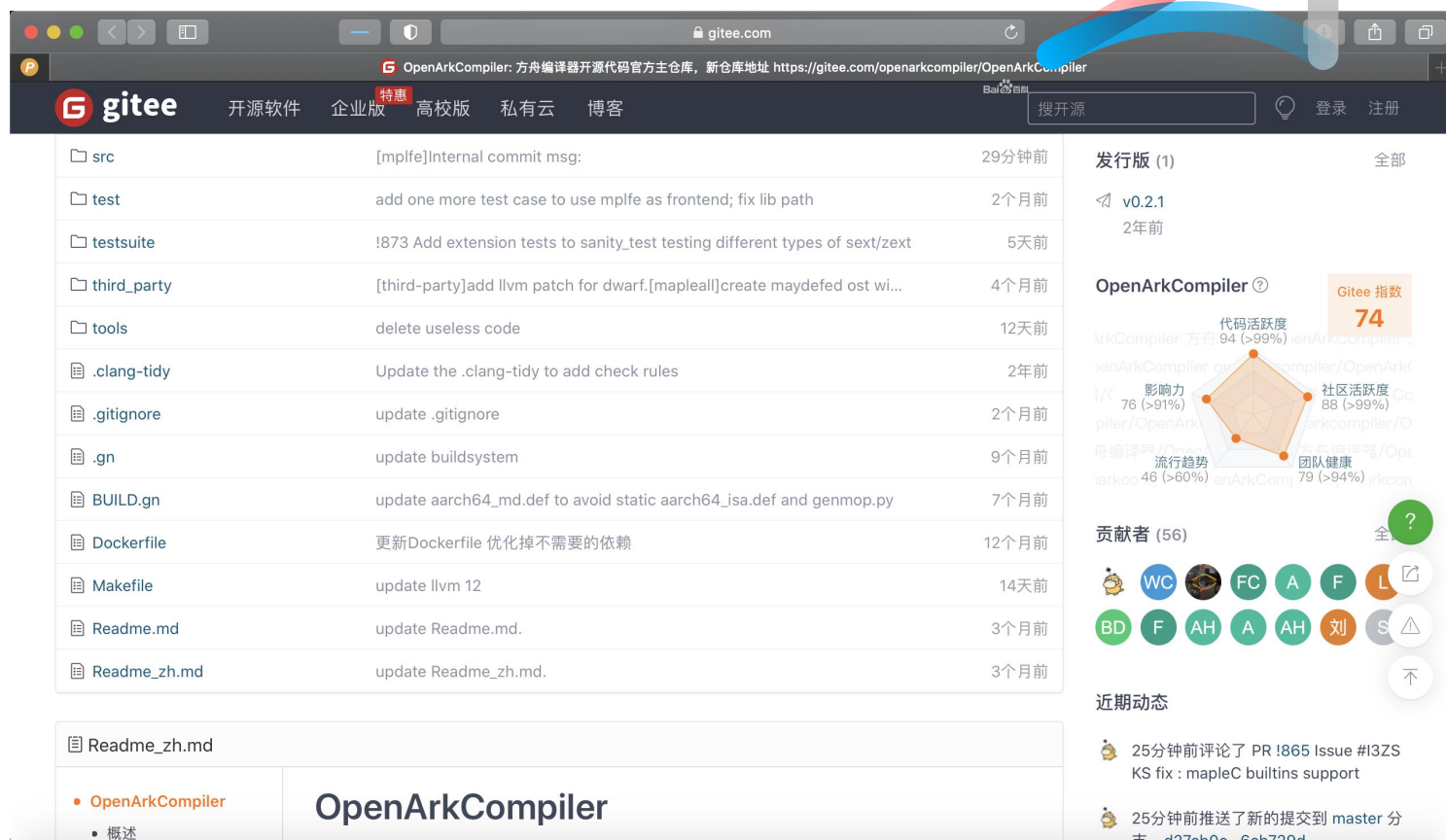
- **1954**年至1957年期间，IBM的John Backus(约翰·巴克斯)带领的一个研究小组对FORTRAN语言及其编译器的开发
 - 将算术公式翻译成机器代码
- **1956**年，Noam Chomsky (诺姆·乔姆斯基)开始了自然语言结构的研究。他的发现最终使得编译器结构异常简单
- **70年代**，有穷自动机和形式语言的研究，促进了编译器的发展
 - 1975年，Steve Johnson为Unix系统编写了**YACC** (yet another compiler-compiler)自动生成语法分析器
 - 同时期，Mike Lesk为Unix系统开发的**Lex**，自动生成词法分析器

目前已形成一套比较成熟、系统的理论与方法及开发环境

编译技术	典型系统	核心技术问题	技术发展趋势
通用处理器	1.GCC [自由软件基金会] 2.LLVM [伊利诺伊大学厄巴纳-香槟分校,美国] 3.ICC [INTEL,美国] 4.MSVC [微软,美国] 5.PGI[PGI公司,美国]	1.面向体系结构的优化,开发指令级-SIMD级、核级并行 2.数据局部性优化 3.面向的领域众多,建立完备的生态链,尤其是对于国产通用处理器亟待解决 4.编程抽象亟待适应新硬件 5.多核处理器芯片内通信调度难度大	1.针对硬件体系结构的优化,尤其是SIMD优化 2.构建新的编程模型和语言,能够刻画日益复杂的存储层次,同时降低并行程序开发难度 3.设计应用专用的编程接口 4.加强用户程序安全保障
众核处理器	1.CUDA [NVDI,美国,针对 Nvidia GPU,支持 CUDA] 2.ICC [INTEL,美国] 3.LLVM [伊利诺伊大学,美国] 4.太湖之光编译系统[江南计算技术研究所,中国]	1.片内丰富并行性的表达 2.众核处理器编程困难 3.众核结构上的软件移植困难	1.设计通用的编程框架,提供统一的并行编程接口 2.设计针对专用应用的编译优化 3.定义用户友好的高效专用库,降低编程难度
专用领域芯片	1.XLA [Google,美国] 2.TensorRT [NVIDIA,美国] 3.TVM [华盛顿大学,美国] 4.CN-CC[寒武纪,中国] 5.DNNC [深鉴科技,中国] 6.MindSpore[华为,中国]	1.中间表达形式的灵活性低 2.硬件无关的中间代码优化困难 3.硬件相关的执行代码优化困难 4.面向异构硬件的联合编译困难	1.从面向特定芯片的专用编译系统到针对不同芯片架构的通用编译系统 2.编译过程中更加重视安全保障和提供间歇计算能力 3.编译技术可充分利用异构硬件资源
FPGA	1.Vivado HLS[Xilinx, 美国] 2.Spatial [斯坦福大学,美国] 3.OpenCL [国际异构计算联盟] 4.LegUp[多伦多大学,加拿大] 5.DWARV [代尔夫特理工大学, 荷兰] 6.BAMBU[米兰理工大学,意大利]	1.可重用性和可移植性问题 2.调试问题 3.高层综合系统的低效问题 4.缺乏存储抽象和优化支持	1.提高当前 HLS 功能的易用性和调试技术 2.特定应用的高层次综合系统 3.提供高效存储访问的优化支持

编译技术的发展

华为方舟编译器(OpenArkCompiler): 十年磨一剑



The screenshot displays the Gitee.com page for the OpenArkCompiler repository. The browser address bar shows the URL: <https://gitee.com/openarkcompiler/OpenArkCompiler>. The repository name is "OpenArkCompiler: 方舟编译器开源代码官方主仓库, 新仓库地址: https://gitee.com/openarkcompiler/OpenArkCompiler".

Repository Files and Commits:

File	Commit Message	Time
src	[mplfe]internal commit msg:	29分钟前
test	add one more test case to use mplfe as frontend; fix lib path	2个月前
testsuite	!873 Add extension tests to sanity_test testing different types of sext/zext	5天前
third_party	[third-party]add llvm patch for dwarf.[mapleall]create maydefed ost wi...	4个月前
tools	delete useless code	12天前
.clang-tidy	Update the .clang-tidy to add check rules	2年前
.gitignore	update .gitignore	2个月前
.gn	update buildsystem	9个月前
BUILD.gn	update aarch64_md.def to avoid static aarch64_isa.def and genmop.py	7个月前
Dockerfile	更新Dockerfile 优化掉不需要的依赖	12个月前
Makefile	update llvm 12	14天前
Readme.md	update Readme.md.	3个月前
Readme_zh.md	update Readme_zh.md.	3个月前

Project Details:

- 发行版 (1):** v0.2.1 (2年前)
- Gitee 指数:** 74
- 代码活跃度:** 94 (>99%)
- 影响力:** 76 (>91%)
- 社区活跃度:** 88 (>99%)
- 流行趋势:** 46 (>60%)
- 团队健康:** 79 (>94%)

贡献者 (56): WC, FC, A, F, L, BD, F, AH, A, AH, 刘, S, 不

近期动态:

- 25分钟前评论了 PR !865 Issue #13ZS KS fix : mapleC builtins support
- 25分钟前推送了新的提交到 master 分支 d27eb0c_6eb729d

OpenArkCompiler

- 概述

内容线索

✓ 什么叫编译程序

- 编译过程概述

- 编译程序的结构

- 编译程序的生成

- 总结

编译过程概述

- 掌握编译过程的五个基本阶段，是我们学习编译原理课程的基本内容，把编译的五个基本阶段与英译中的五个步骤相比较，有利于对编译过程的理解：


英译

1. 识别出句子中的一个单词
2. 分析句子的语法结构
3. 初步翻译句子的含意
4. 译文修饰
5. 写出最后译文

编译

1. 词法分析
2. 语法分析
3. 语义分析中间代码生成
4. 优化
5. 目标代码生成

1,词法分析

- 词法分析程序又称扫描程序(Scanner)。
 - 任务：读源程序的字符流、识别单词（也称单词符号，或简称符号），如标识符、关键字、常量、界限符等，并转换成内部形式。
 - 输入：源程序中的字符流
 - 输出：等长的内部形式，即属性字
(Token-name, Attribute-value)


↑ ↑
抽象名字 指向符号表
- 在词法分析阶段工作所依循的是语言的词法规则。
- 描述词法规则的有效工具是正规式和有限自动机。
- 方法：状态图； DFA； NFA

1,词法分析示例

例：一个C源程序片段：

```
int a,b,c;
```

```
a=a+2;
```

词法分析后返回(如右图)：

符 号 表

1	a	...
2	b	...
3	c	...

单词类型

保留字

标识符

界符

标识符

界符

标识符

界符

标识符

算符(赋值)

标识符

算符(加)

整数

界符

单词值

int <int>

a <id,1>

, <,>

b <id,2>

, <,>

c <id,3>

; <;>

a <id,1>

= <op,EQ>

a <id,1>

+ <+>

2 <2>

; <;>

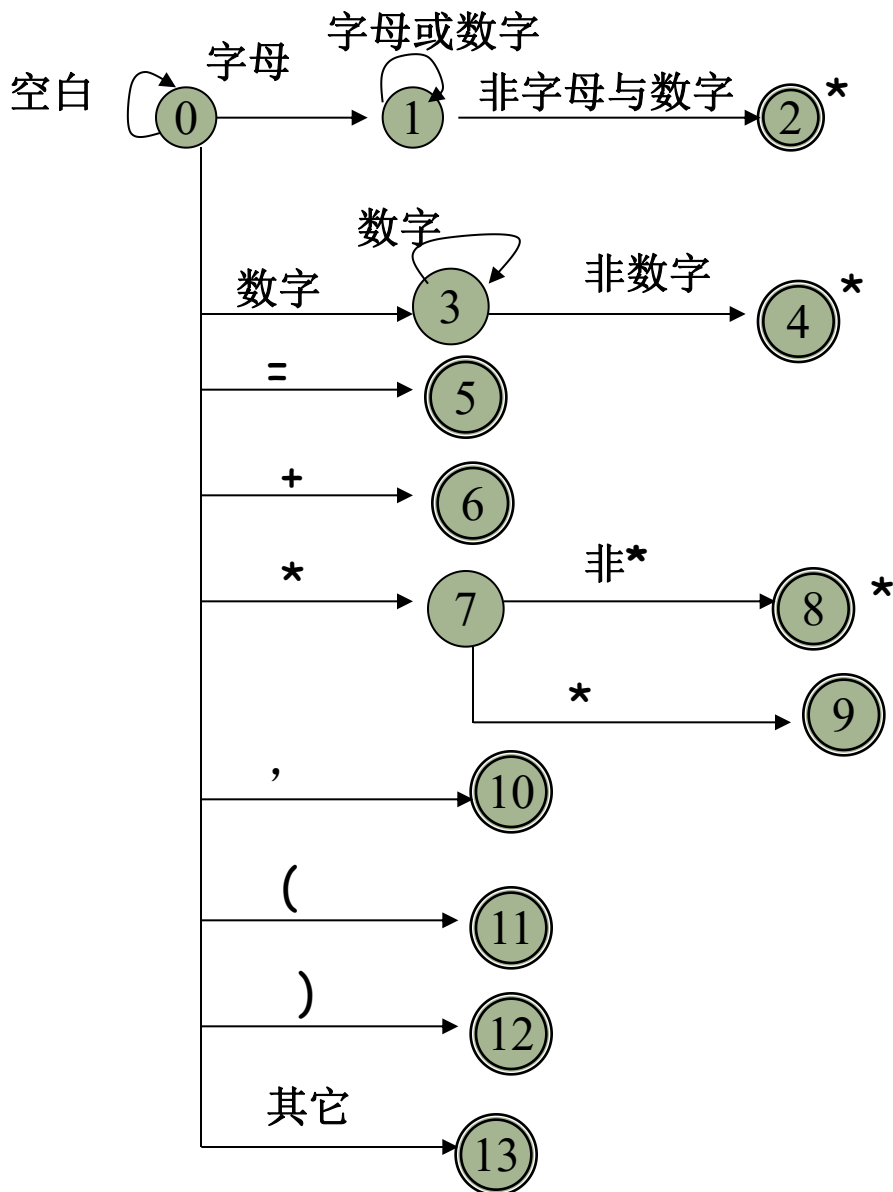
能识别小语言所有单词的状态转换图

约定（限制）：

✓ 关键字为保留字；

✓ 保留字作为标识符处理, 并使用保留字表识别；

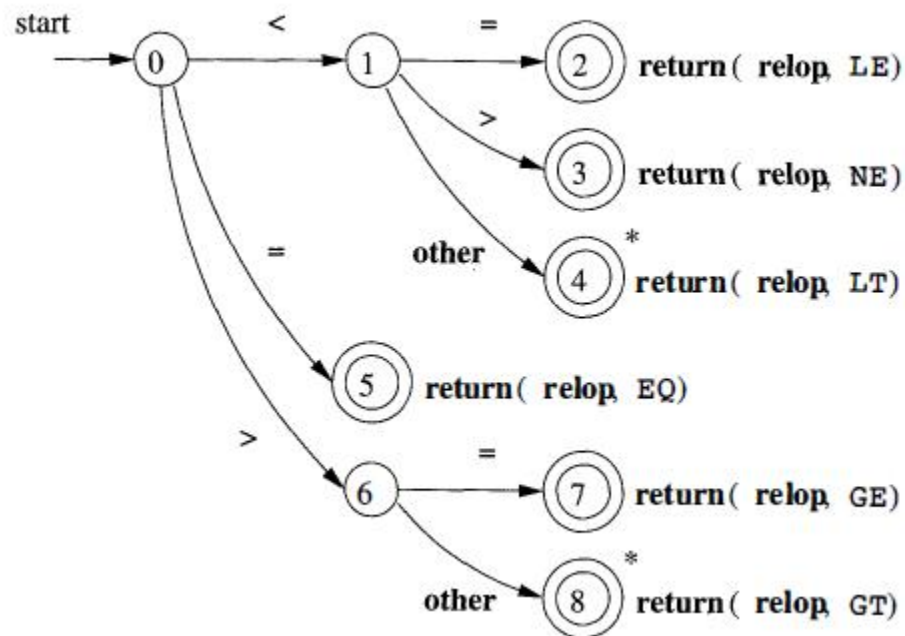
✓ 关键字、标识符、常数间若无运算符或界限符则加一空格



状态转换图示例

```

TOKEN getRelop()
{
    TOKEN retToken = new(RELOP);
    while(1) { /* repeat character proce
                or failure occurs */
        switch(state) {
            case 0: c = nextChar();
                    if ( c == '<' ) state = 1;
                    else if ( c == '=' ) state = 5;
                    else if ( c == '>' ) state = 6;
                    else fail(); /* lexeme is not a relop */
                    break;
            case 1: ...
            ...
            case 8: retract();
                    retToken.attribute = GT;
                    return(retToken);
        }
    }
}
    
```



Simulating a DFA

```
s = s0;  
c = nextChar() ;  
while ( c != eof ) {  
    s = move(s, c);  
    c = nextChar() ;  
}  
if ( s is in F ) return " yes " ;  
else return "no " ;
```

2, 语法分析

- 语法分析程序又称识别程序(Parser)。
 - 任务：读入由词法分析程序识别出的符号，根据给定语法规则，识别出各个语法单位（如：短语、子句、语句、程序段、程序），并生成另一种内部表示。
 - 输入：由词法分析程序识别出并转换的符号
 - 输出：另一种内部表示，如语法分析树或其它中间表示。
- 语法规则通常用上下文无关文法描述。
- 方法：递归子程序法、LR分析法、算符优先分析法。

例: `sum:= first + count * 10`

- 规则

<赋值语句> \rightarrow <标识符> “:=” <表达式>

<表达式> \rightarrow <表达式> “+” <表达式>

<表达式> \rightarrow <表达式> “*” <表达式>

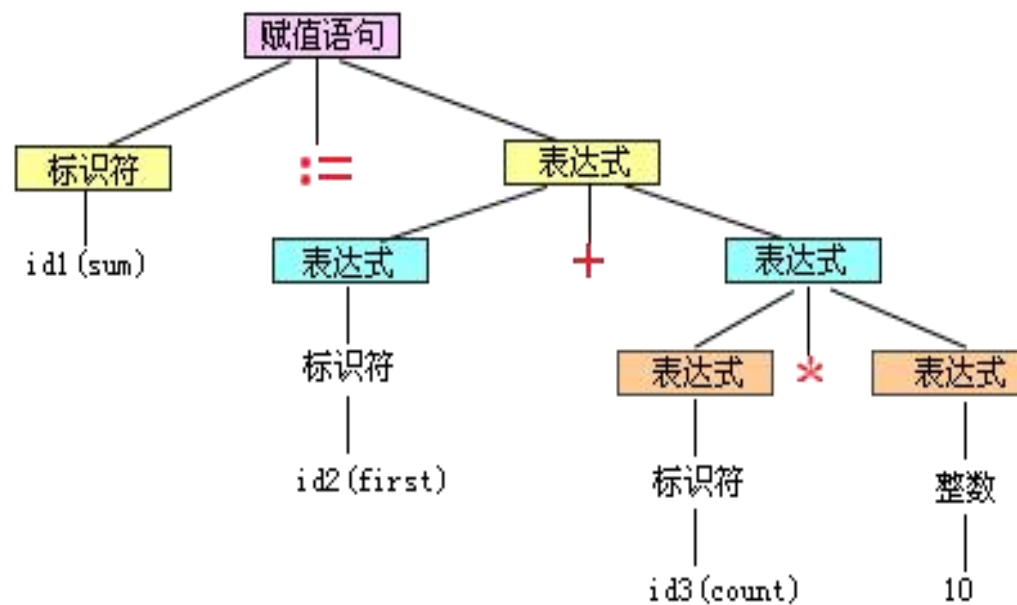
<表达式> \rightarrow “(” <表达式> “)”

<表达式> \rightarrow <标识符>

<表达式> \rightarrow <整数>

<表达式> \rightarrow <实数>

例 `sum := first + count * 10` 的语法树



3-1,语义分析

- 对语法分析树或其他内部中间表示进行静态语义检查，如果正确则进行中间代码的翻译。
 - 按照语法树的层次关系和先后次序，逐个语句地进行语义处理。
 - 主要任务：进行类型审查，审查每个算符是否符合语言规范，不符合时应报告错误。
 - 变量是否定义
 - 类型是否正确

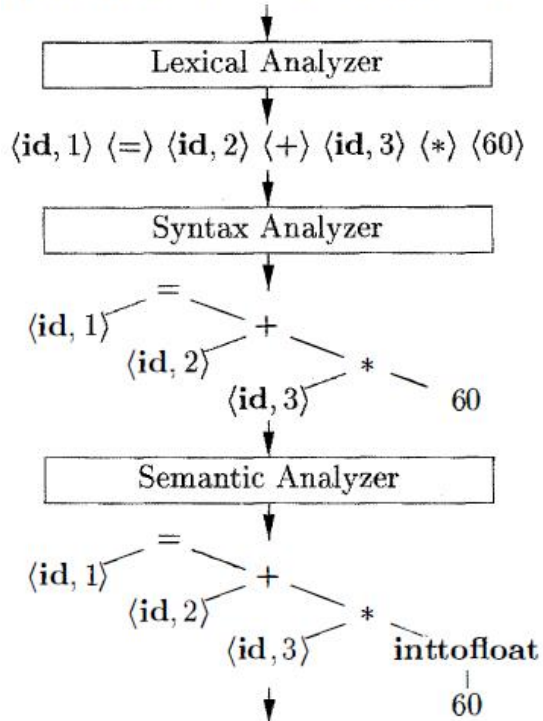
语义分析示例

插入语义处理结点的语法树

符号表

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

position = initial + rate * 60



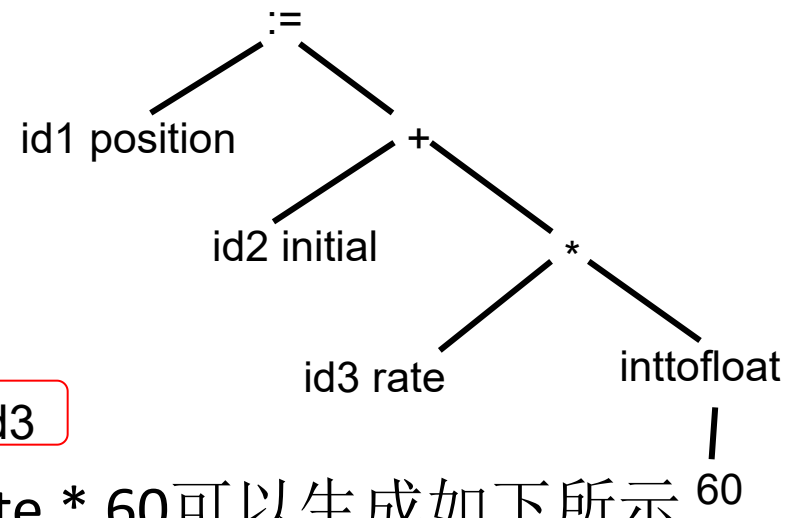
图来源: Compilers Principles
Techniques and Tools(2nd Edition)

3-2,中间代码生成

- 中间代码是一种独立于具体硬件的记号系统，或者与现代计算机的指令形式有某种程度的接近，或者能比较容易地变换成机器指令。
 - 任务：将各类语法单位，如“表达式”、“语句”、“程序”等翻译为中间代码序列。
 - 输入：句子
 - 输出：中间代码序列
- 中间代码的形式：常见的有四元式、三元式和逆波兰式等
- 方法：语义子程序；**DAG**图(有向无环图)；语法制导翻译

■ 四元式的形式为：

(算符，运算对象1，运算对象2，结果)；



- 对于源程序 `position := position + rate * 60` 可以生成如下所示的四元式：

(1) (inttofloat, 60 , — , t1)

(2) (* , id3 , t1 , t2)

(3) (+ , id2 , t2 , t3)

(4) (:= , t3 , — , id1)

t1, t2, t3 是
临时变量

- 其中：id1、id2、id3 分别表示 position、initial、rate 的机器内部表示，t1、t2、t3 是临时生成的名字，表示中间运算结果。

例2: C语言的源程序 $a = b * c + b * d$ 的三地址序列 (赋值语句形式的四元式) :

(1) $t1 := b * c$	$(*, b, c, t1)$
(2) $t2 := b * d$	$(*, b, d, t2)$
(3) $t3 := t1 + t2$	$(+, t1, t2, t3)$
(4) $a := t3$	$(:=, t3, -, a)$

四元式:

例3:源程序:

if ($a \leq b$)

$a = a - c$;

else $c = b * c$;

翻译成
→

100 ($j \leq, a, b, 102$)

101 ($j, _, _, 104$)

102 ($-, a, c, t1$)

103 ($=, t1, _, a$)

104 ($*, b, c, t2$)

105 ($=, t2, _, c$)

找错误?

4， 优化

- 优化的任务在于对前段产生的中间代码进行加工，把它变换成功能相同，但功效更高的优化了的中间表示代码，以期在最后阶段产生更为高效（省时间和空间）的代码
- 优化所依循的原则是程序的等价变换规则
- 其方法有：公共子表达式的提取、循环优化、删除无用代码等等。

举例


position = initial + rate*60

(1) (inttofloat 60 - t1)

(2) (* id3 t1 t2)

(3) (+ id2 t2 t3)

(4) (:= t3 - id1)

变换成 \Rightarrow

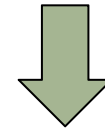
(1) (* id3 60.0 t1)

(2) (+ id2 t1 id1)

5, 目标代码生成

- 这一阶段的任务：把中间代码（或经优化处理后）变换成特定机器上的低级语言代码。它依赖于硬件系统结构和机器指令含义。
 - 与机器相关

id1 **id2** **id3**
position = initial + rate*60
(*, **id3** **60.0** **t1**)
(+, **id2** **t1** **id1**)

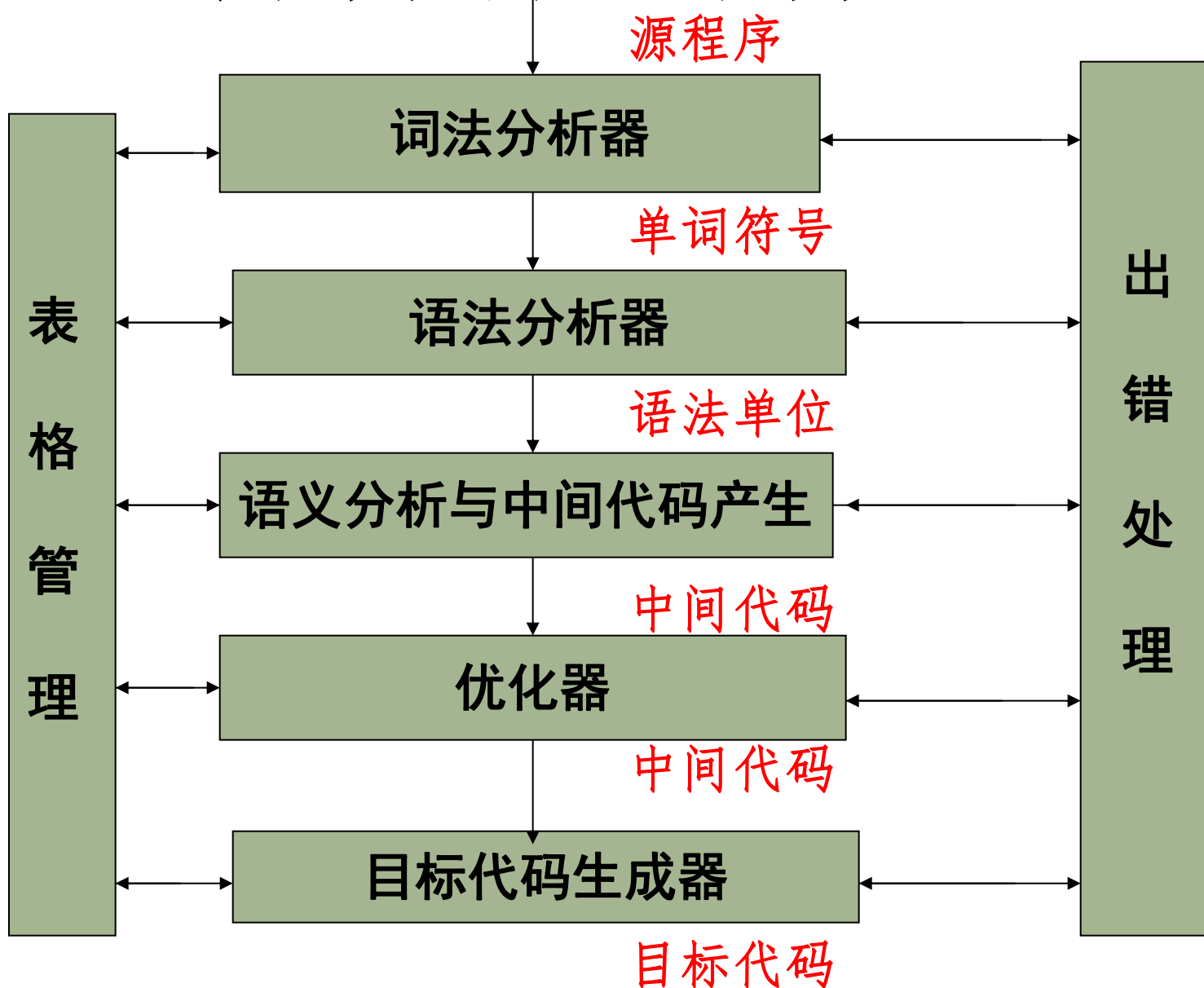


movf id3 R2
mulf #60.0 R2
movf id2 R1
addf R2 R1
movf R1 id1

内容线索

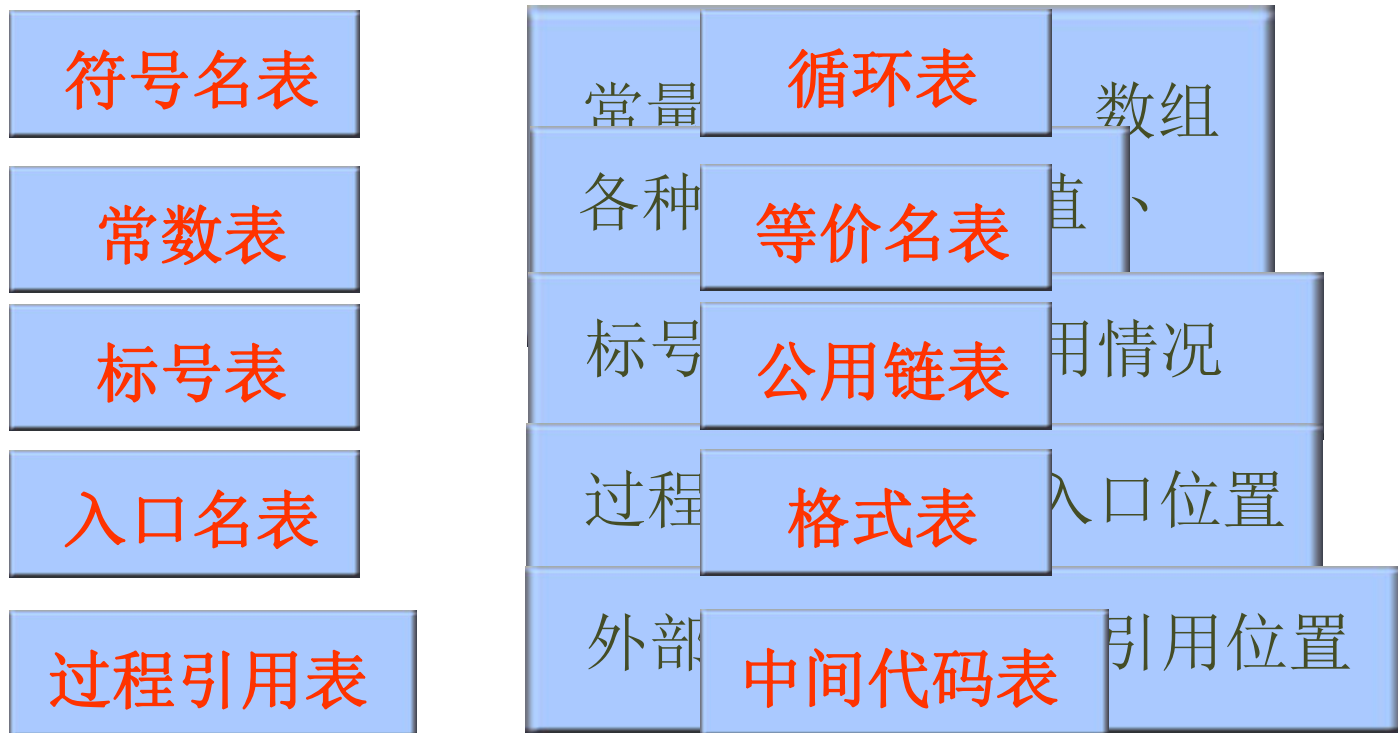
- ✓ 什么叫编译程序
- ✓ 编译过程概述
- 编译程序的结构
- 编译程序的生成
- 总结

编译程序的结构



表格与表格管理

- 编译程序涉及的表格有



符号表

- 在编译程序使用的表格中最重要的是符号表。
 - 记录源程序中使用的名字(标识符)
 - 收集每个名字的各种属性信息
 - 类型、作用域、分配存储信息

名 字	种 类	类 型	层 次	偏 移 量
m	过 程		0	
id1	变 量	real	1	d
id2	变 量	real	1	d+4
id3	变 量	real	1	d+8

出错处理

- 程序中的错误可分为语法错误和语义错误两类。
- 语法错误可在词法分析和语法分析阶段查出来。
- 例如，下列错误都属于语法错误：
 - (1) 保留字拼写错误：DIMENTOIN A(10)
 - (2) 括号不配对：
WRITE (6,10) (A (1, J) , J=1, 10) , I=1, 10)

语义错误

- 语义错误有些可在编译时查出来，有些则需在运行时才能查出来。
- 典型的语义错误：
 - 标识符没有说明就使用；
 - 标号有引用而无定义；
 - 形式参数和实在参数结合时在类型、个数、位置等方面不一致等；
- 这些错误可在编译时查出来，而另一些错误如下标越界、运算溢出、调用某些标准函数时自变量的值不符合要求等，则需要到程序运行时才能查出来。

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内

若能自动校正错误则更好，但其代价非常高

编译阶段的组合

- 编译程序可以从逻辑上分成几个阶段，对于各个阶段的划分仅仅是指其逻辑结构，而在具体实现时，经常是将几个阶段组合在一起。例如，可以将各部分组合成前端和后端。

编译过程

前端：词法分析、语法分析、语义分析、中间代码生成、优化工作。

后端：目标机有关的优化工作、目标代码生成等。

主要与源语言有关

主要与目标代码有关

遍 (Pass)

一遍

- 对源程序或源程序的中间结果从头到尾扫描一次，并做相关处理，生成新的中间结果或目标程序的过程。
- “遍”是处理数据的一个完整周期，每遍工作从外存上获得前一遍的中间结果（如源程序），完成它所含的有关工作之后，再把结果记录于外存。

语法分析器
处于核心地位

一遍 局部优化

一遍 全局优化

一遍

词法分析

语法分析

中间代码生成

代码优化

目标代码生成

遍的次数和效果

- 一个编译程序可由一遍、两遍或多遍完成。每一遍可完成不同的阶段或多个阶段的工作。

从时间
和空间
角度看

多遍编译 — 少占内存，多耗时间

一遍编译 — 多占内存，少耗时间

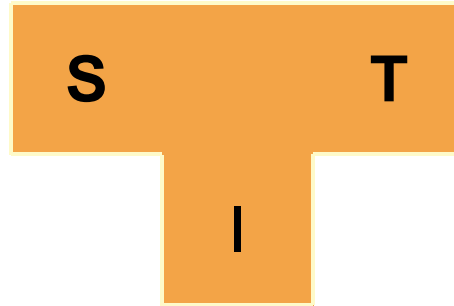
内容线索

- ✓ 什么叫编译程序
- ✓ 编译过程概述
- ✓ 编译程序的结构
- 编译程序的生成
- 总结

编译程序实现语言

- 机器语言
- 汇编语言
 - 充分发挥各种不同硬件系统的效率
 - 满足各种不同的具体要求
- 高级语言
 - 大大节省程序设计的时间
 - 构造出来的编译程序易于阅读、维护和移植

T形图

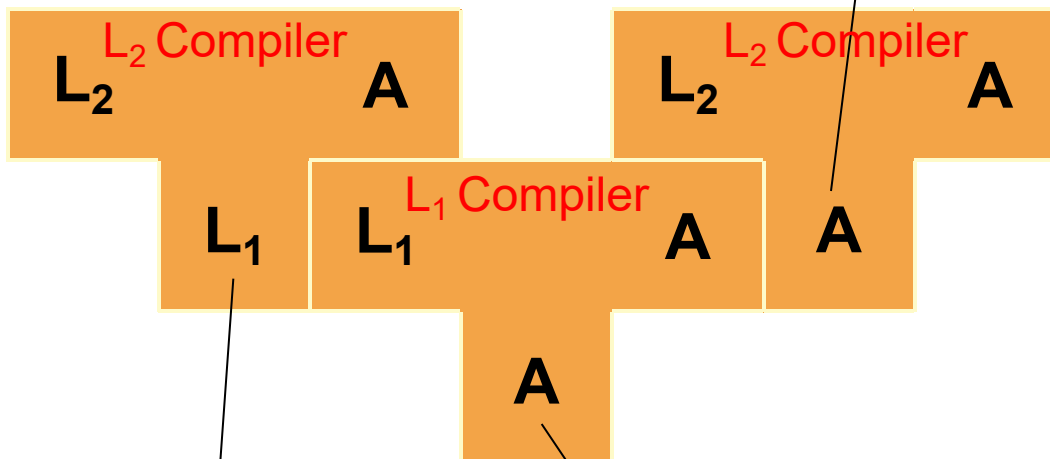


- 其中：
 - S:源语言(程序), Source language(program)
 - T:目标语言(程序), target/object language(program)
 - I:实现语言, implementation language

示例

L_1 : C
 L_2 : Java

A机器代码实现的
 L_2 编译程序



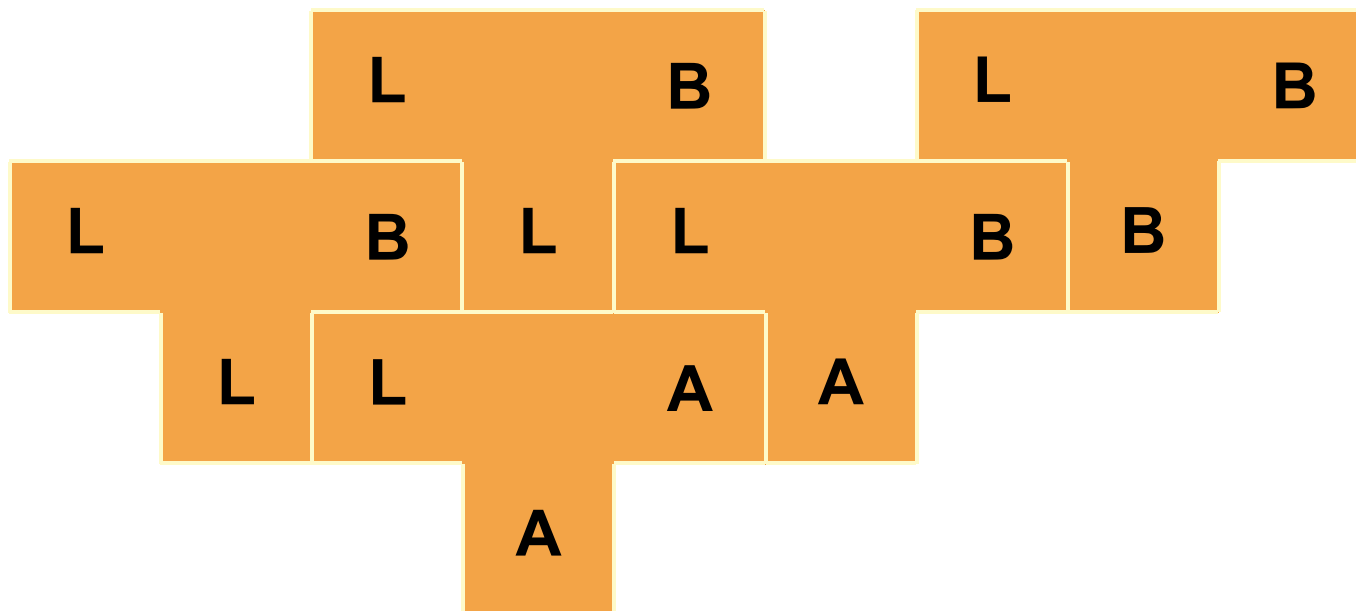
用 L_1 语言编写另
一种高级语言 L_2
的编译程序

A机器上已有一个用A
机器码实现的某高级
语言 L_1 的编译程序

如果A机器上已有一个用A机器码实现的某高级语言 L_1 的编译程序，则我们可以用 L_1 语言编写另一种高级语言 L_2 的编译程序，把写好的 L_2 编译程序经过 L_1 编译程序编译后就可得到A机器代码实现的 L_2 编译程序

示例（P9-10倒数第2段）

编译程序“移植”



编译程序的生成技术-自编译

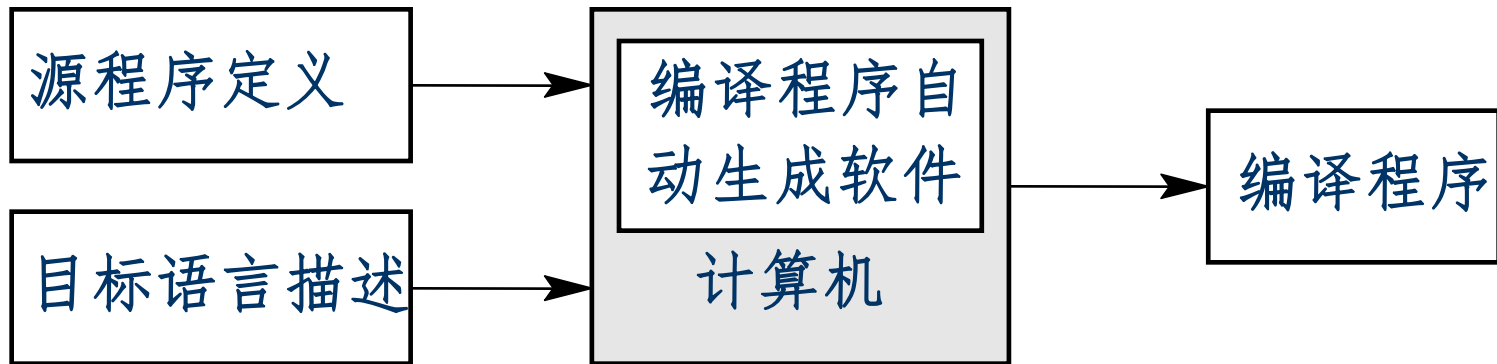
- 用某种高级语言书写自己的编译程序称为自编译。
- 例如,假定A机器上已有一个C语言编译程序,则可用C语言编写一个功能更强的C语言编译程序,然后借助于原有的编译程序对新编写的C编译程序进行编译,从而得到一个能在A机器上运行的功能更强的C编译程序。

编译程序的生成技术-交叉编译

- 用x机器上的编译程序产生可在 y 机器上运行的目标代码称为交叉编译。
- 例如, 若x机器上已有C语言编译程序,则可用x机器中的C语言书写一个编译程序, 该编译程序的源程序是C语言程序,而产生的目标程序则是基于y机器的, 即产生在y机器上执行的低级语言程序。
- 上述两种方法假定已有一个编译程序, 若没有, 则可采用自展或移植法。
- 应用场景: 嵌入式系统、手机操作系统

自动编译

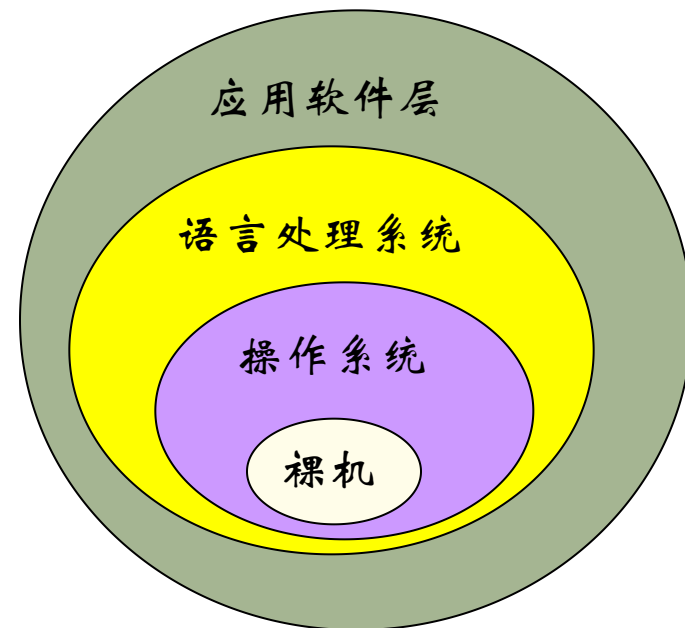
- 自动生成编译程序的软件工具, 只要把源程序的定义及目标语言的描述输入到该软件中, 就能自动生成该语言的编译程序。



- 编译程序自动生成系统, 如**lex/flex**, **bison/yacc**, **ANTLR**(Another Tool for Language Recognition)等

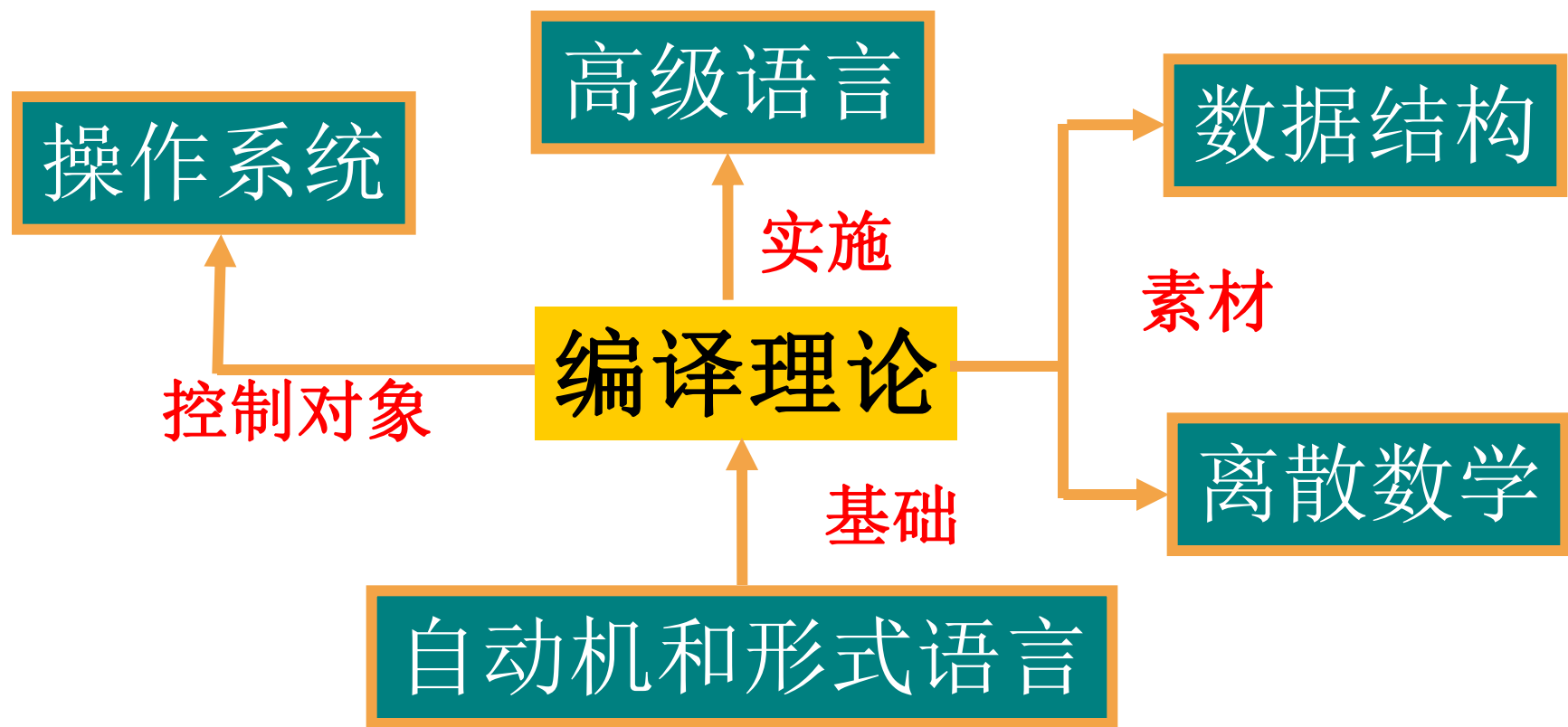
编译程序在计算机系统中的地位

- 编译系统是一种系统软件
 - 软件：计算机系统上的程序及其文档。
 - 系统软件：居于计算机系统中最靠近硬件的一层，其他软件一般都通过系统软件发挥作用。和具体的应用领域无关，如编译系统和操作系统等。
 - 语言处理系统：把软件语言书写的各种程序处理成可在计算机上执行的程序，如编译系统。



计算机系统

编译理论与其他课程的关系



内容线索

- ✓ 什么叫编译程序
- ✓ 编译过程概述
- ✓ 编译程序的结构
- ✓ 编译程序的生成
- 总结

Dank u

Dutch

Merci

French

Спасибо

Russian

Gracias

Spanish

شكراً

Arabic

감사합니다

Korean

רבה תודה

Hebrew

Tack så mycket

Swedish

धन्य व

िद

Hindi

Obrigado

Brazilian
Portuguese

Dankon

Esperanto

ありがとうございます

Japanese

Thank You !

谢谢

Chinese

Trugarez

Breton

Danke

German

Tak

Danish

Grazie

Italian

நன் ற

ஈ

Tamil

go raibh maith agat

Gaelic

děkuji

Czech

ขอบ

กณ

Thai