编译原理-课程简介

马玲

联系方式

编译原理

编译原理-2023

南开大学

- 主讲教师：马玲
- 办公地点：泰达学院3区221

仅限企业内部成员加入

该二维码 7 天内 (9/7前)有效

课程目的

- 学习编译理论和编译技术
- 使用辅助工具
- 能用这些理论、技术和工具设计一个简单的编译器

先导课程

计算机基础

高级语言程序设计

数据结构

操作系统

*汇编语言程序设计

…

参考教材

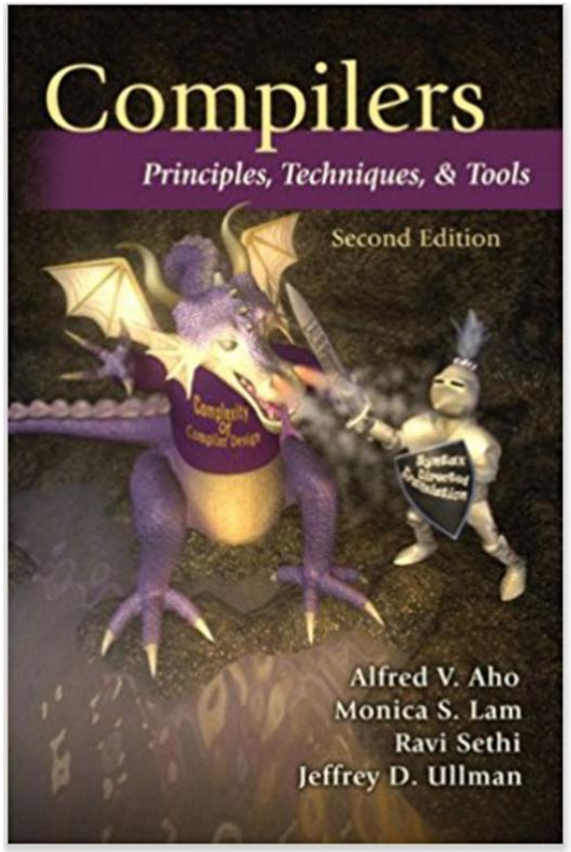
计算机科学丛书：编译原理（第2版）
[Compilers:Principle,Techniques and Tools]

[美] Alfred V.Aho，[美] Monica S.Lam，[美] Ravi Sethi 等 著；赵建华，郑滔 等 译

参考教材

Compilers: Principles, Techniques, and Tools (2nd Edition) 2nd Edition

by Alfred V. Aho, Monica S. Lam, Ravi Sethi, Jeffrey D. Ullman

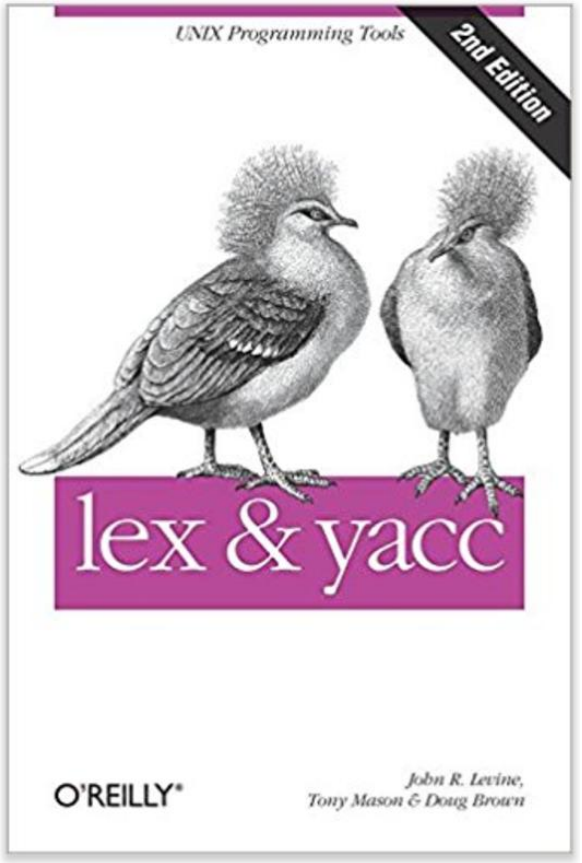


参考教材

《Lex与Yacc》（第二版）（Lex and Yacc），Levine等著，杨作梅等译，机械工业出版社，2003

lex & yacc 2nd Edition

by Doug Brown, John Levine, Tony Mason

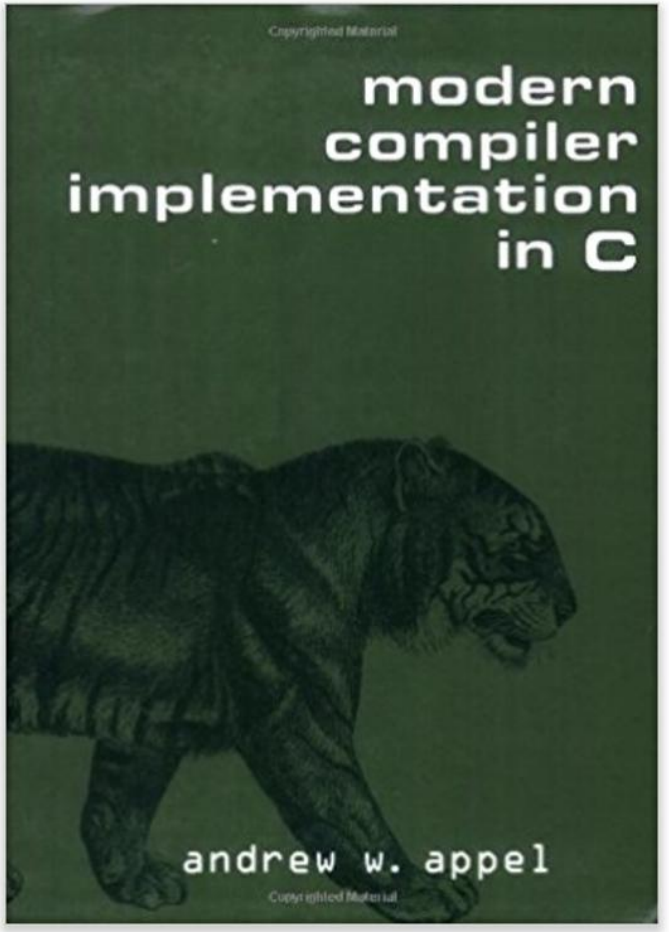


参考教材

Modern Compiler Implementation in C

by Andrew W. Appel

《现代编译原理:C语言描述》



参考教材

编译原理及实现（第2版）

孙悦红 著

清华大学出版社

参考教材

编译原理及实践教程（第2版）
黄贤英，曹琼，王柯柯 著

成绩

- 平时成绩：40％
  - 书面作业 ：15％
  - 上机实验：25％
    - 简化的C语言编译器 + 实验报告
    - 团队合作（3-5人）
    - 发现抄袭情况，成绩全为0
- 期末成绩：60％

- 上课：1 - 17周(第18周总结)
- 实验：9 - 17周

课程内容

编译器的核心功能是把源代码翻译成目标代码：

- 理解源代码
  词法分析、语法分析、语义分析
- 转化为等价的目标代码
  中间代码生成、目标代码生成
- 更好
  优化方法

第一章 介绍

学习内容

- 1.1　编译器的概述
- 1.2　编译器的结构
- 1.3　编译技术的应用

学习内容

- **1.1　编译器的概述**
- 1.2　编译器的结构
- 1.3　编译技术的应用

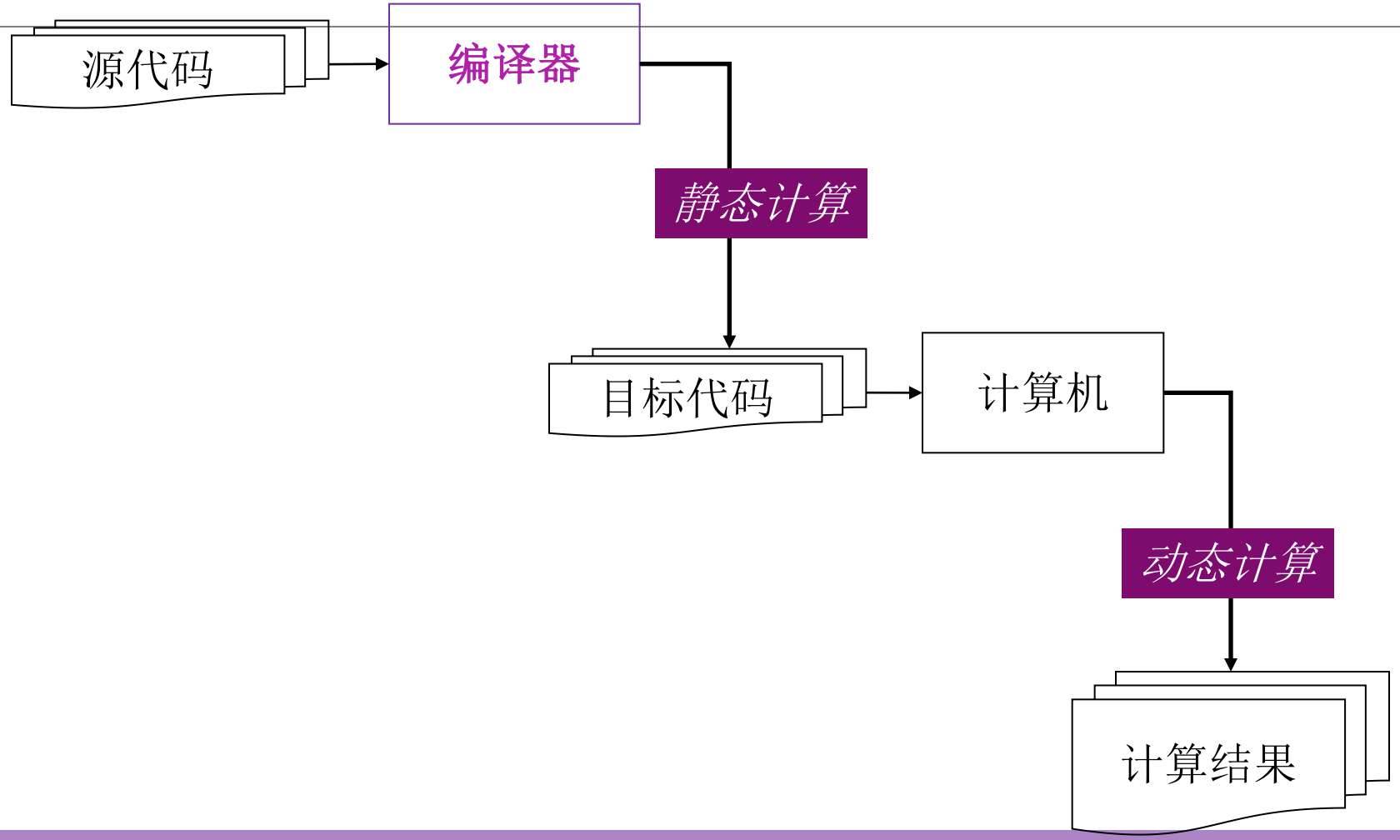
1.1 编译器的概述

- 计算设备包括个人计算机、大型机、嵌入式系统、智能设备等
- 核心的问题都是软件的构造
  - 绝大部分软件都是由高级语言书写
  - 成百种高级语言
- 这些高级语言写的程度是如何在计算机上运行呢？
  - 编译器

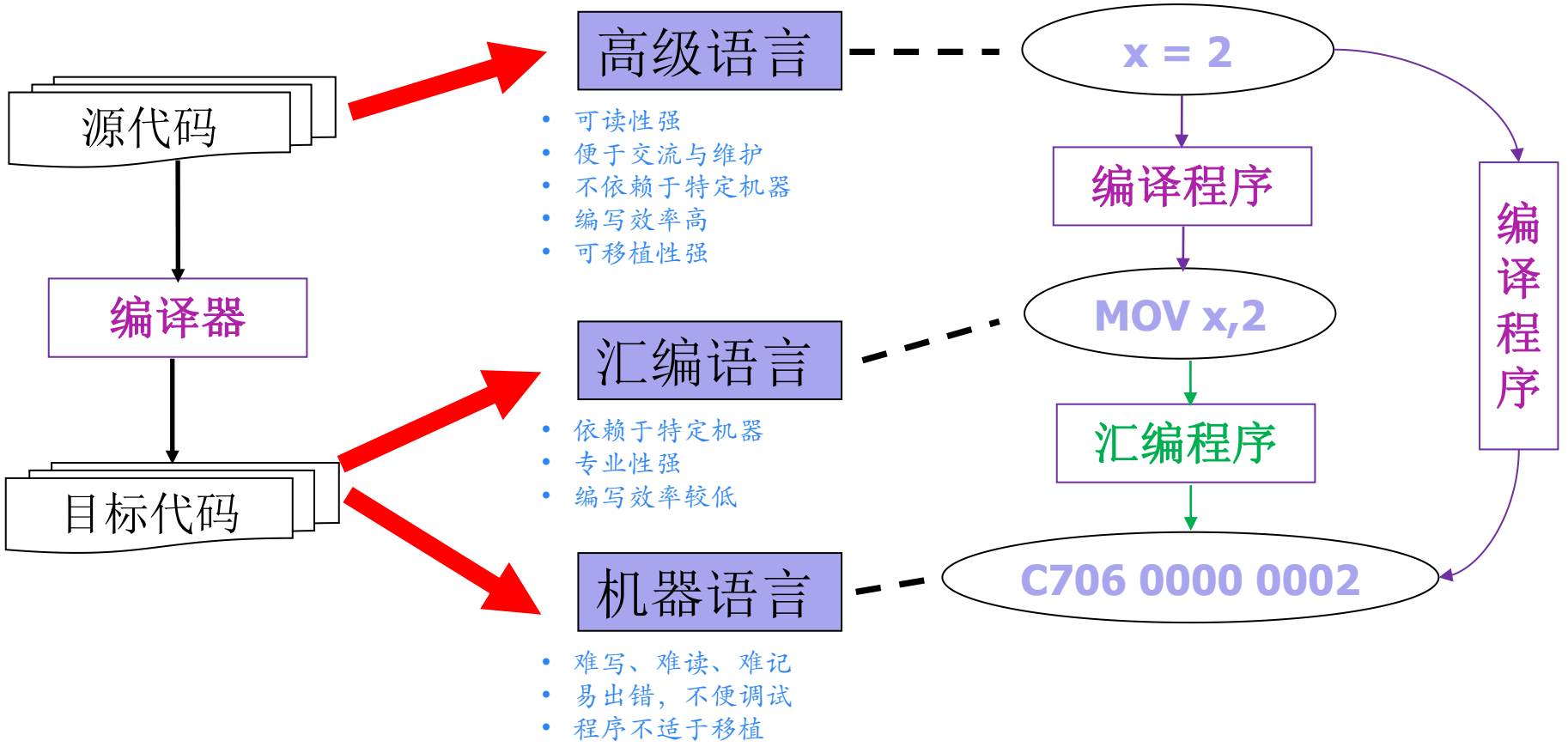
什么是编译器

- 编译器是一个程序
- 核心功能是把源代码翻译成目标代码
- 源代码:
  - C/C++, Java, C#, html, ...
- 目标代码:
  - x86, IA64, ARM, MIPS, ...

编译器的核心功能



编译器的核心功能


源代码
编译器
目标代码
高级语言
可读性强
便于交流与维护
不依赖于特定机器
编写效率高
可移植性强
汇编语言
依赖于特定机器
专业性强
编写效率较低
机器语言
难写、难读、难记
易出错，不便调试
程序不适于移植
x = 2
编译程序
MOV x,2
汇编程序
C706 0000 0002
编译程序

编译器的简历

- 计算机科学史上出现的第一个编译器是Fortran （Formula Translation）语言的编译器

- John Backus

为什么要学习编译原理

- 编译原理集中体现了计算机科学的很多核心思想
  - 算法，数据结构，软件工程等
- 编译器是其他领域的重要研究基础
- 编译器本身就是非常重要的研究领域
  - 新的语言设计
  - 大型软件的构造和维护

如何学好编译原理

- 编译器设计是理论和实践高度结合的一个领域，在学习过程中要处理好两者关系：
  - 理论：深入学习掌握各种算法和数据结构
  - 实践：理论应用于解决实际问题的能力

学习内容

- 1.1　编译器的概述
- 1.2　**编译器的结构**
- 1.3　编译技术的应用

1.2 编译器的结构

```c
#include <stdio.h>

int main()
{
    int a = 3;
    int b = 4;
    int c = a + b;
    printf("c=%d \n", c);

    return 0;
}
```

```
#include <stdio.h>

int main()
{
00E813C0  push        ebp
00E813C1  mov         ebp,esp
00E813C3  sub         esp,0E4h
00E813C9  push        ebx
00E813CA  push        esi
00E813CB  push        edi
00E813CC  lea         edi,[ebp-0E4h]
00E813D2  mov         ecx,39h
00E813D7  mov         eax,0CCCCCCCCh
00E813DC  rep stos    dword ptr es:[edi]
    int a = 3;
00E813DE  mov         dword ptr [a],3
    int b = 4;
00E813E5  mov         dword ptr [b],4
    int c = a + b;
00E813EC  mov         eax,dword ptr [a]
00E813EF  add         eax,dword ptr [b]
00E813F2  mov         dword ptr [c],eax
    printf("c=%d \n", c);
00E813F5  mov         esi,esp
00E813F7  mov         eax,dword ptr [c]
00E813FA  push        eax
00E813FB  push        0E85858h
00E81400  call        dword ptr ds:[0E89114h]
00E81406  add         esp,8
00E81409  cmp         esi,esp
00E8140B  call        __RTC_CheckEsp (0E81136h)

    return 0;
00E81410  xor         eax,eax
}
```

1.2 编译器的结构

- 编译器具有非常模块化的高层结构
- 编译器可看成多个阶段构成的“流水线”结构

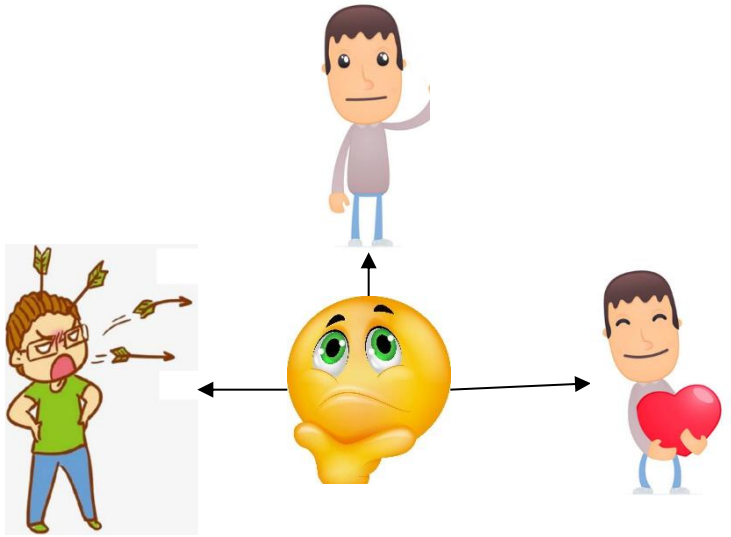
1.2 编译器的结构

英汉翻译例子：

英语句子成分有主语，谓语，宾语，宾语补足语，定语，状语等

When someone acts small,	you	just have to be	the bigger person
状语	主语	谓语	宾语

词法，语法，语义

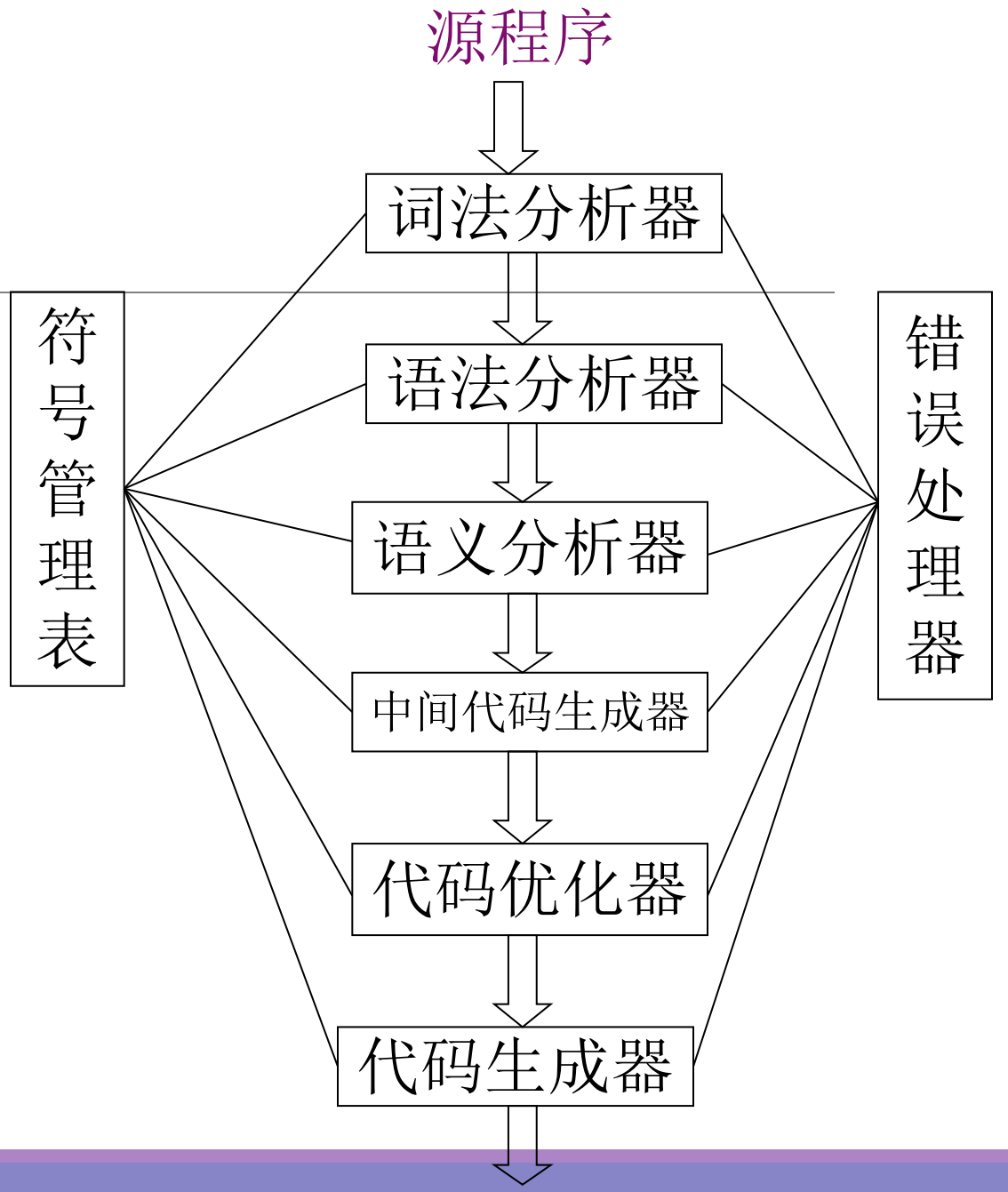


当别人对你不友好时，你只需做一个宽宏大量的人

编译器的结构

每个阶段将源程序从一种表示转换成另一种表示

随着编译器各个阶段的进展，源程序的内部表示不断地发生变化。



①

词法分析器的任务

从左到右，一个字符一个字符地读入源程序，对构成源程序的字符流进行扫描和分解，从而识别出一个个单词。

词法分析器的任务

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

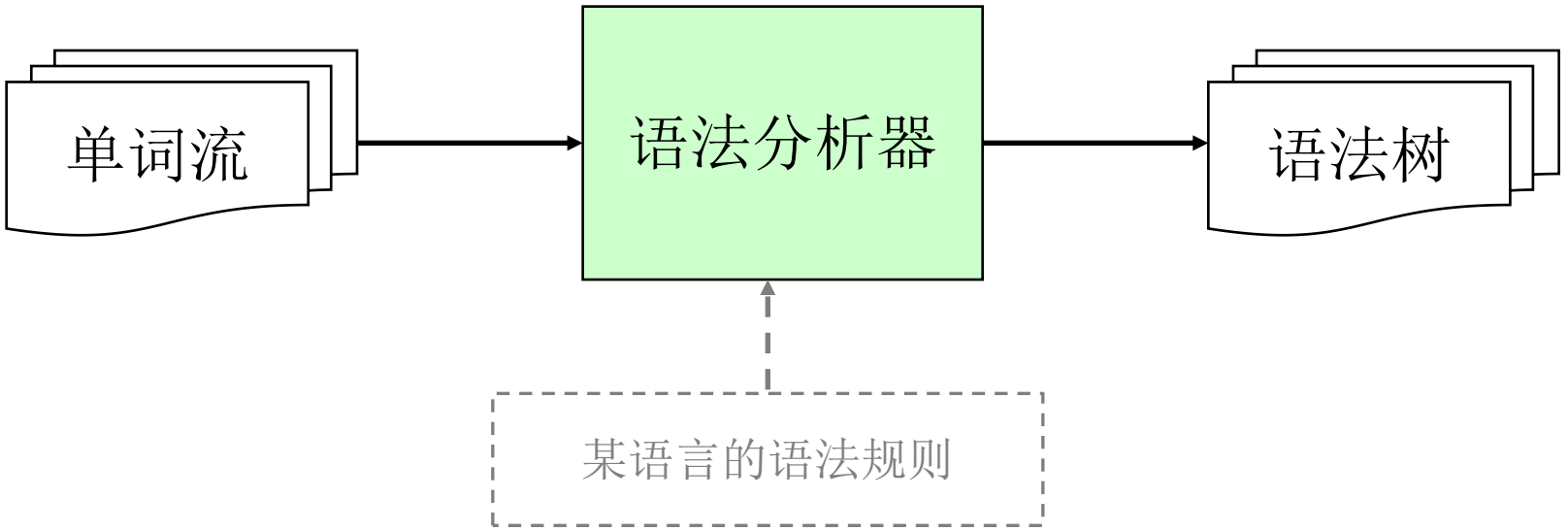
词法分析的示例

```
float limitedSquare(x)
{
    float x;
    return (x<=-10.0)||x>10.0)?100:x*x;
}
```

```
<float,->
<id,limitedSquare>
<(,->
<id, x>
<) ,->
<{,->
<float>
<id, x>
<; ,->
<return,->
<(,->
<id, x>
<op,"<=">
<num, -10.0>
<op, "||">
<id, x>
<op, ">=">
<num, 10.0>
<) ,->
<op, "?">
<num, 100>
<op, ":">
<id, x>
<op, "*">
<id, x>
<; ,->
<},->
```

②

语法分析器的任务



根据语言的语法规则，把单词符号串组成各类语法单位.

语法分析：是否符合语法

- 不符合：返回出错处理信息
- 符合：在单词流的基础上建立一个层次结构-----建立语法树

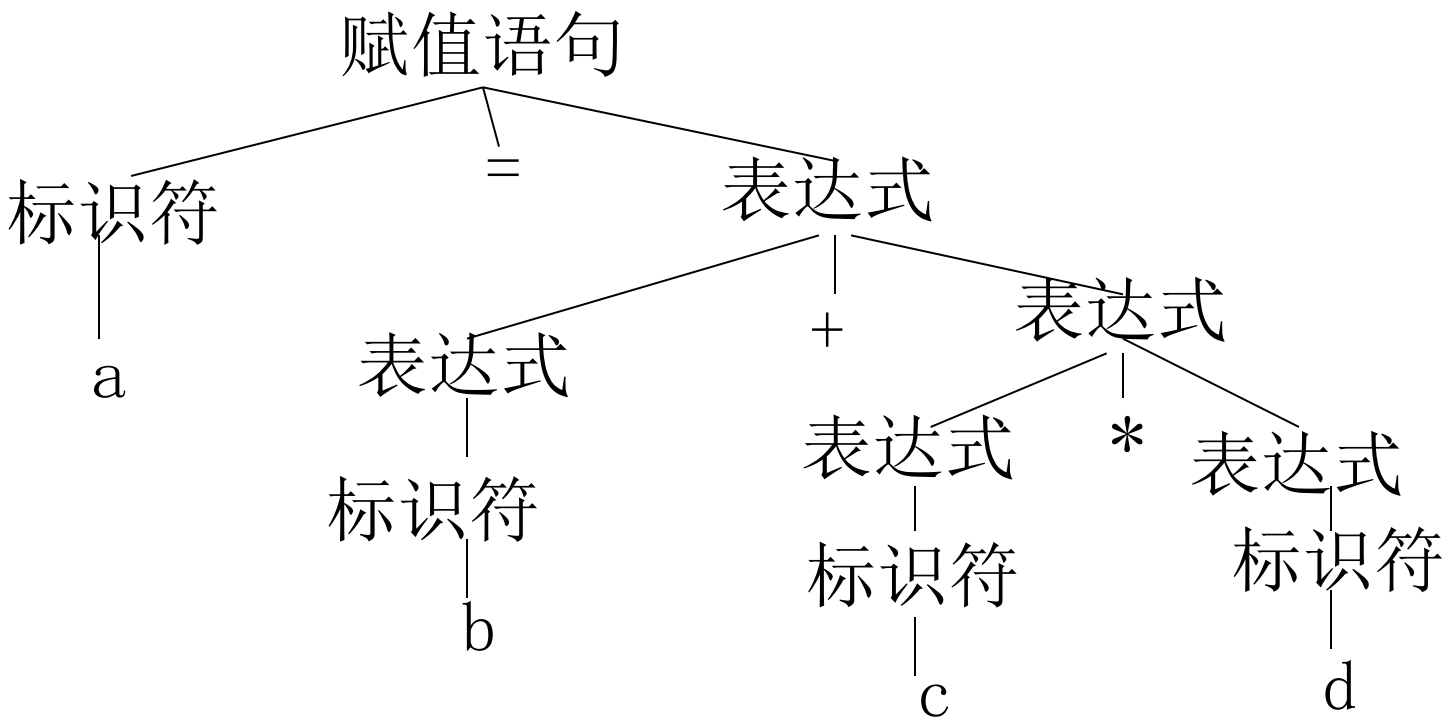
核心模块：处理程序员所写程序的输入 + 产生编译器需要的重要结构

语法分析

在单词流的基础上建立一个层次结构-----建立语法树

例：　a = b + c * d

<id,a>
<=,->
<id, b>
<+,->
<id, c>
<*,->
<id, d>



③

语义分析任务

收集标识符的属性信息

类型(Type)

种属(Kind)

存储位置、长度

值

作用域

参数和返回值信息

语义检查

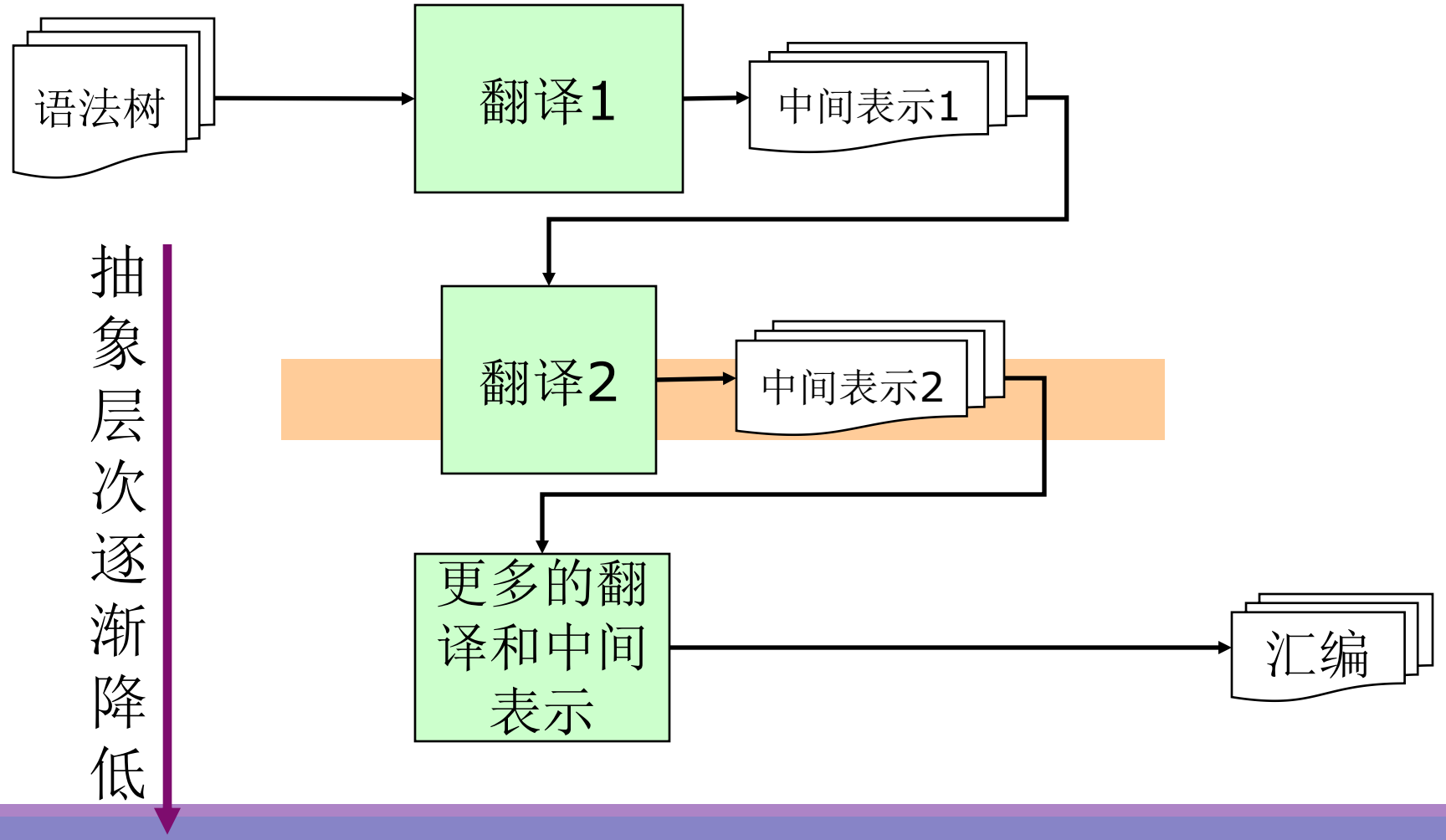
- 变量或过程未经声明就使用
- 变量或过程名重复声明
- 运算分量类型不匹配
- 操作符与操作数之间的类型不匹配

语义分析的示例

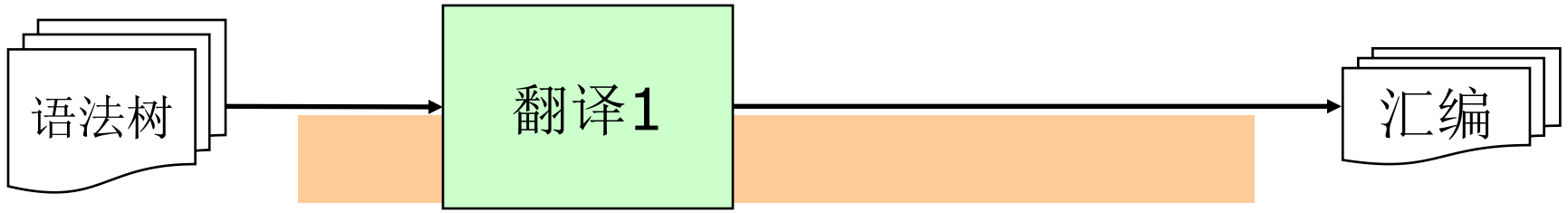
```
void f ( int *p)
{
        x += 4;
        p(23);
        “hello” + “world”;
}
int main()
{
        f() + 5;
        break;
        return;
}
```

④

中间代码



最简单的结构



（代码生成）

以前常用

目前少用：实现维护难度大

中间代码生成阶段

本阶段将产生源程序的一个显式中间表示

这种中间表示可以看成是某种抽象的程序，通常是与平台无关的

其重要性质：1. 易于产生

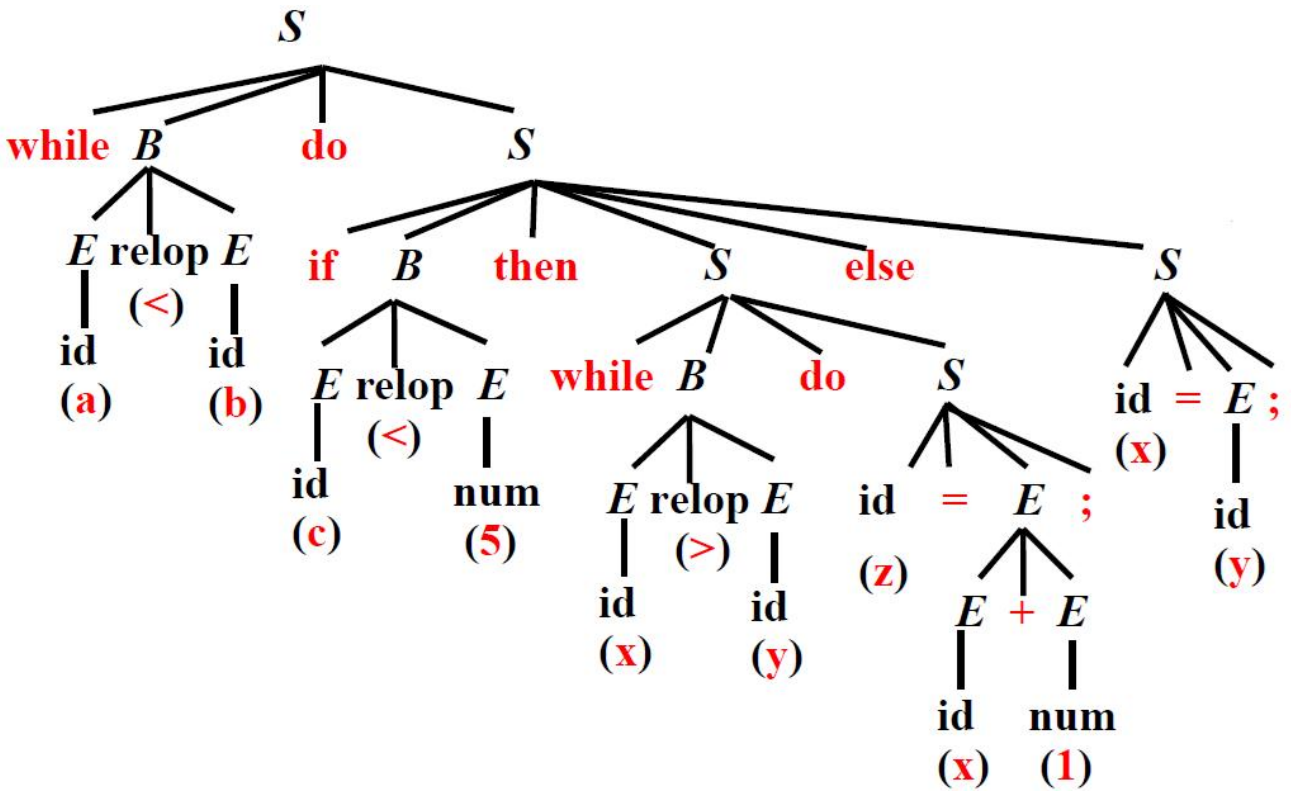
2. 易于翻译成目标程序

中间代码生成阶段

下面是用三地址码和四元式表示的例子：

	(op, arg1, arg2, result)
temp1=c*d	(* , c , d , temp1)
temp2=b+temp1	(+ , b, temp1 , temp2)
a=temp2	(= , temp2 , - , a)

```
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 , - ,  z  )
108: ( j  , - ,  - , 104 )
109: ( j  , - ,  - , 100 )
110: ( = , y , - ,  x  )
111: ( j  , - ,  - , 100 )
112:
```

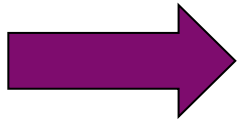
⑤

代码优化阶段

试图改进中间代码，以产生执行速度较快的机器代码

对上面中间代码进行优化处理后，产生如下的代码：

```
temp1=c*d
temp2=b+temp1
a=temp2
```



```
temp1=c*d
a=b+temp1
```

⑥

代码生成阶段

生成可重定位的机器代码或汇编代码

temp1=c*d	Mov　R2，c
a=b+temp1	Mul　R2, d
	Mov　R1, b
	Add　R2, R1
	Mov　a, R2

一个重要任务是为程序中使用的变量合理分配寄存器

符号表

```
int  a,b;

float e,f

char ch1,ch2;
```

为什么要先说明?

定义了变量的类型，也就规定了变量在内存中的存放形式，在其上所能进行的运算，解决符号地址到存贮地址上的映射

符号表

基本功能是记录源程序中使用的标识符 并收集与每个标识符相关的各种属性信息，并将它们记载到符号表中。

符号表是一个数据结构。

每个标识符在符号表中都有一条记录

例：int a,b;

名字	记号	类型	种属	……	addr
a	id1(25)	int	简变		0
b	id2(25)	int	简变		4

错误处理器

各阶段均会遇到错误

处理方式：报告错误，应继续编译

大部分错误在语法分析、语义分析阶段检测出来

词法分析：字符无法构成合法单词

语法分析：单词流违反语法结构规则

语义分析：语法结构正确，但无实际意义

编译器的结构

- 编译器由多个阶段组成，每个阶段都要处理不同的问题
  - 使用不同的理论，数据结构，和算法

- 因此，编译器设计中的重要问题是如何合理的划分组织各个阶段
  - 接口清晰
  - 编译器容易实现与维护

编译器实例

position = initial + rate * 60

词法分析

⟨id, 1⟩ ⟨=⟩ ⟨id, 2⟩ ⟨+⟩ ⟨id, 3⟩ ⟨*⟩ ⟨60⟩

符号表

1	position	id	. . .
2	initial	id	. . .
3	rate	id	. . .
4	. . .	. . .	. . .

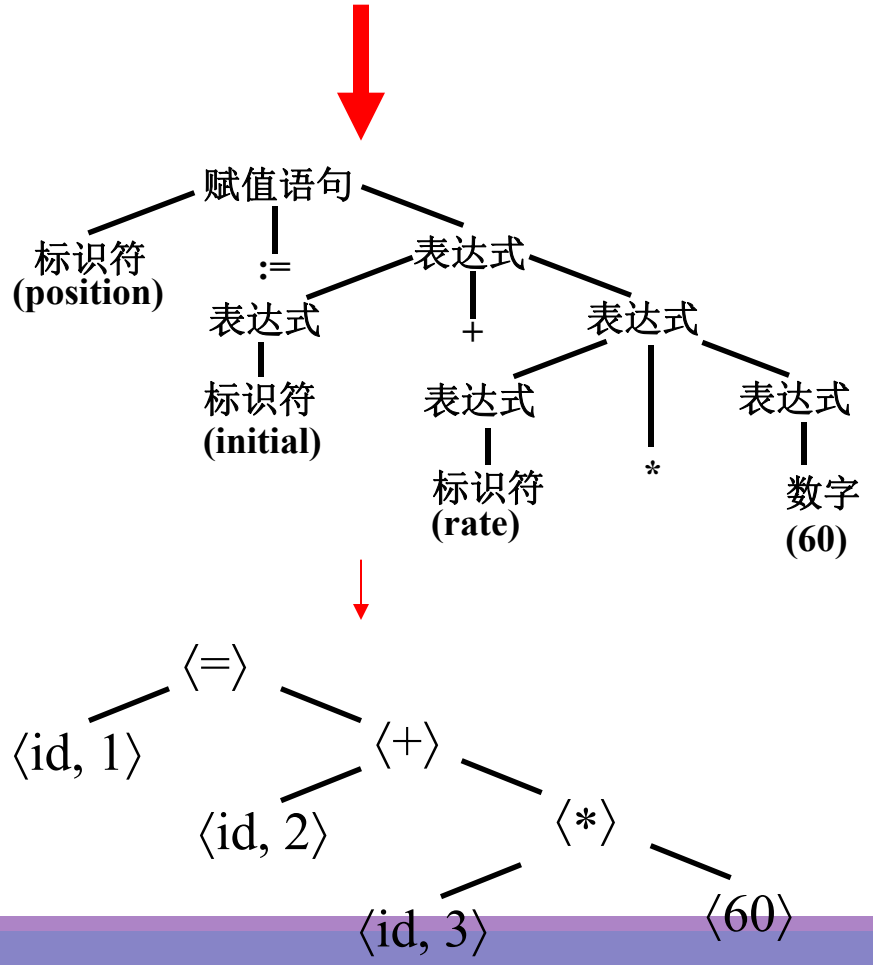
错误信息管理

position = initial + rate * 60

编译器实例

⟨id, 1⟩ ⟨=⟩ ⟨id, 2⟩ ⟨+⟩ ⟨id, 3⟩ ⟨*⟩ ⟨60⟩

语法分析



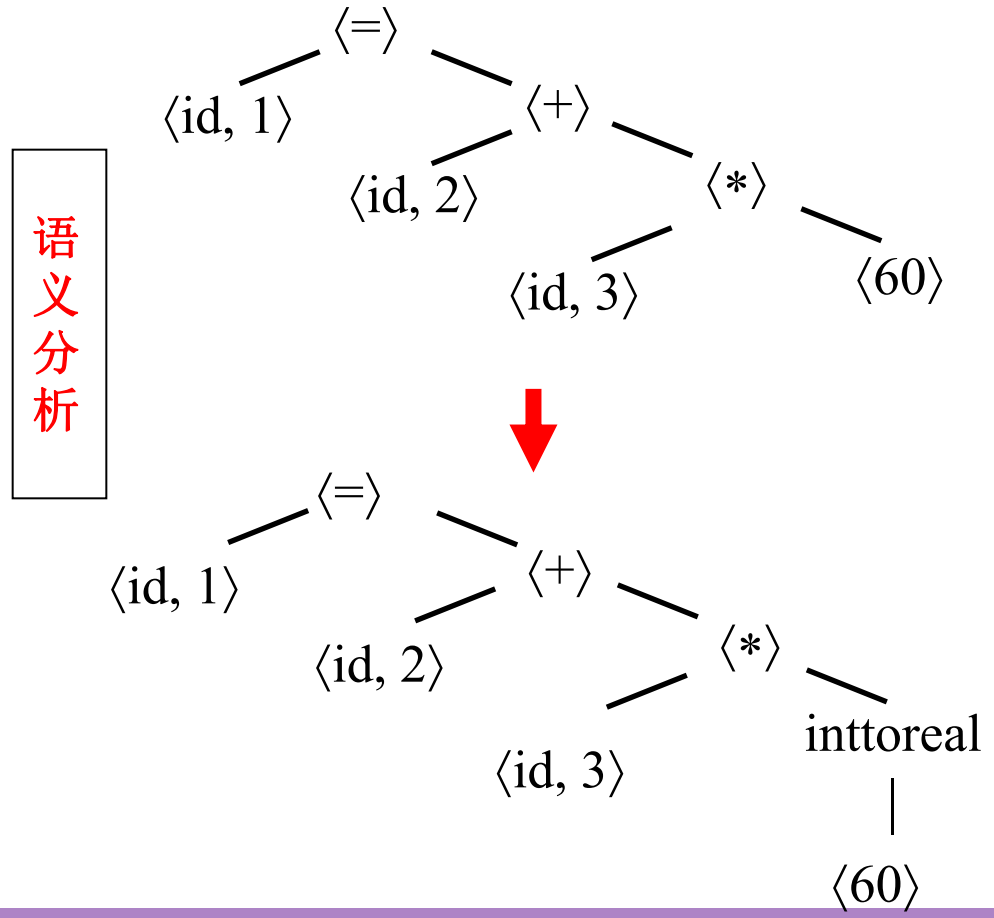
符号表

1	position	id	. . .
2	initial	id	. . .
3	rate	id	. . .
4	. . .	. . .	. . .

错误信息管理

position = initial + rate * 60

编译器实例



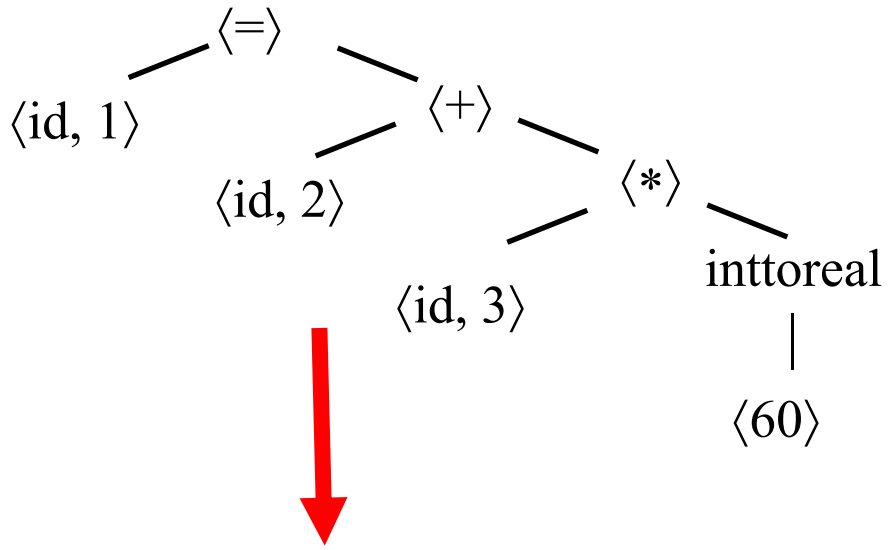
符号表

1	position	id	. . .
2	initial	id	. . .
3	rate	id	. . .
4	. . .	. . .	. . .

错误信息管理

position = initial + rate * 60

编译器实例



中间代码生成

```
t1 = inttoreal(60)
t2 = id3 * t1
t3 = id2 + t2
id1 = t3
```

符号表

1	position	id	. . .
2	initial	id	. . .
3	rate	id	. . .
4	. . .	. . .	. . .

错误信息管理

position = initial + rate * 60

编译器实例

中间代码优化

```
t1 = inttoreal(60)
t2 = id3 * t1
t3 = id2 + t2
id1 = t3
```

```
t1 = id3 * 60.0
id1 = id2 + t1
```

符号表

1	position	id	. . .
2	initial	id	. . .
3	rate	id	. . .
4	. . .	. . .	. . .

错误信息管理

position = initial + rate * 60

编译器实例

目标代码生成

t1 = id3 * 60.0
id1 = id2 + t1

MOV R2, id3
MUL R2, #60.0
MOV R1, id2
ADD R1, R2
MOV id1, R1

符号表

1	position	id	. . .
2	initial	id	. . .
3	rate	id	. . .
4	. . .	. . .	. . .

错误信息管理

学习内容

- 1.1　编译器的概述
- 1.2　编译器的结构
- 1.3　**编译技术的应用**

1.3 编译器技术的应用

高级程序设计语言的实现

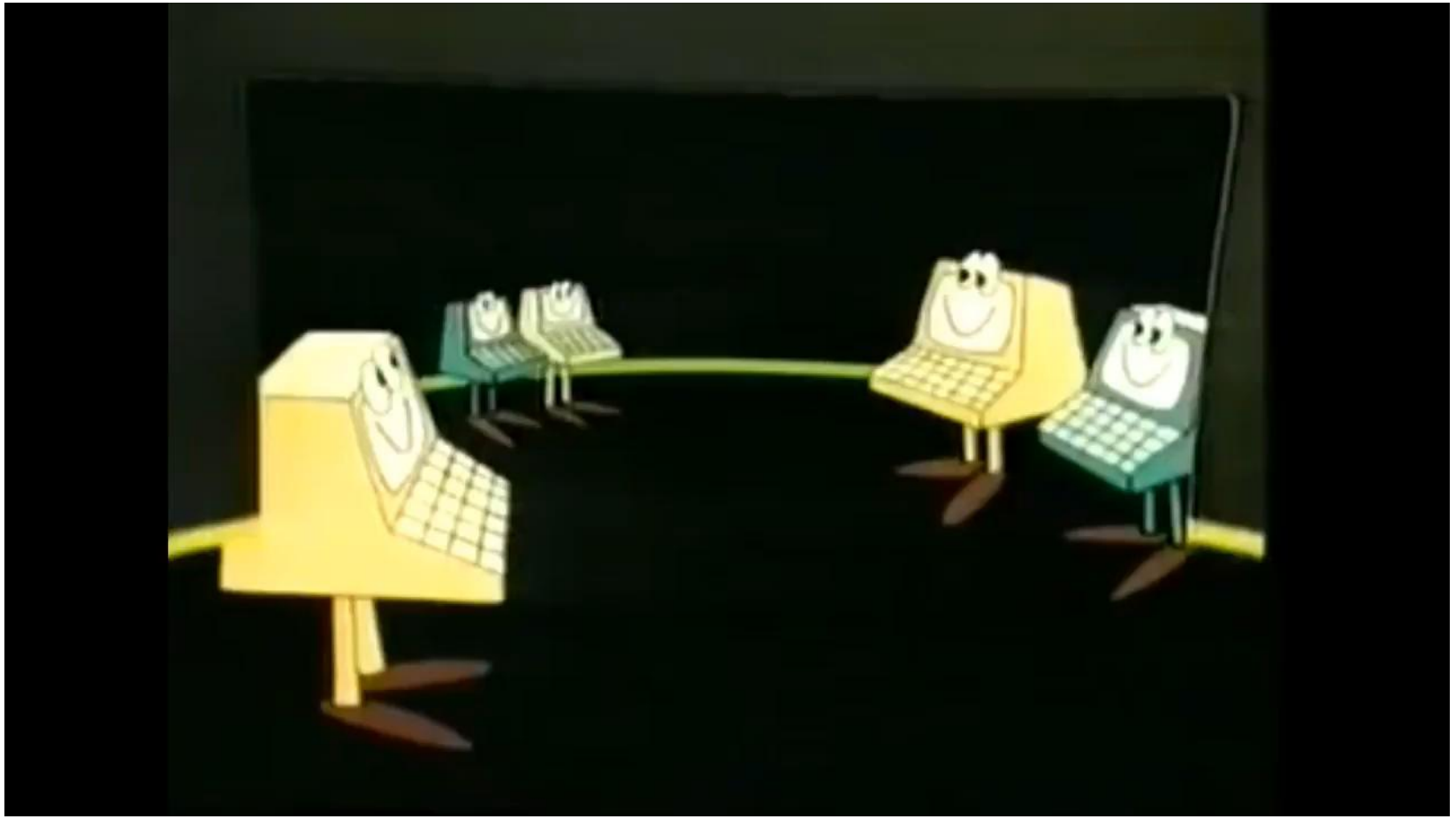
针对计算机体系结构的优化（并行性与内存层次结构）

新计算机体系结构的设计（CISC → RISC）

程序翻译

软件生产率工具（查错）

编译器与解释器



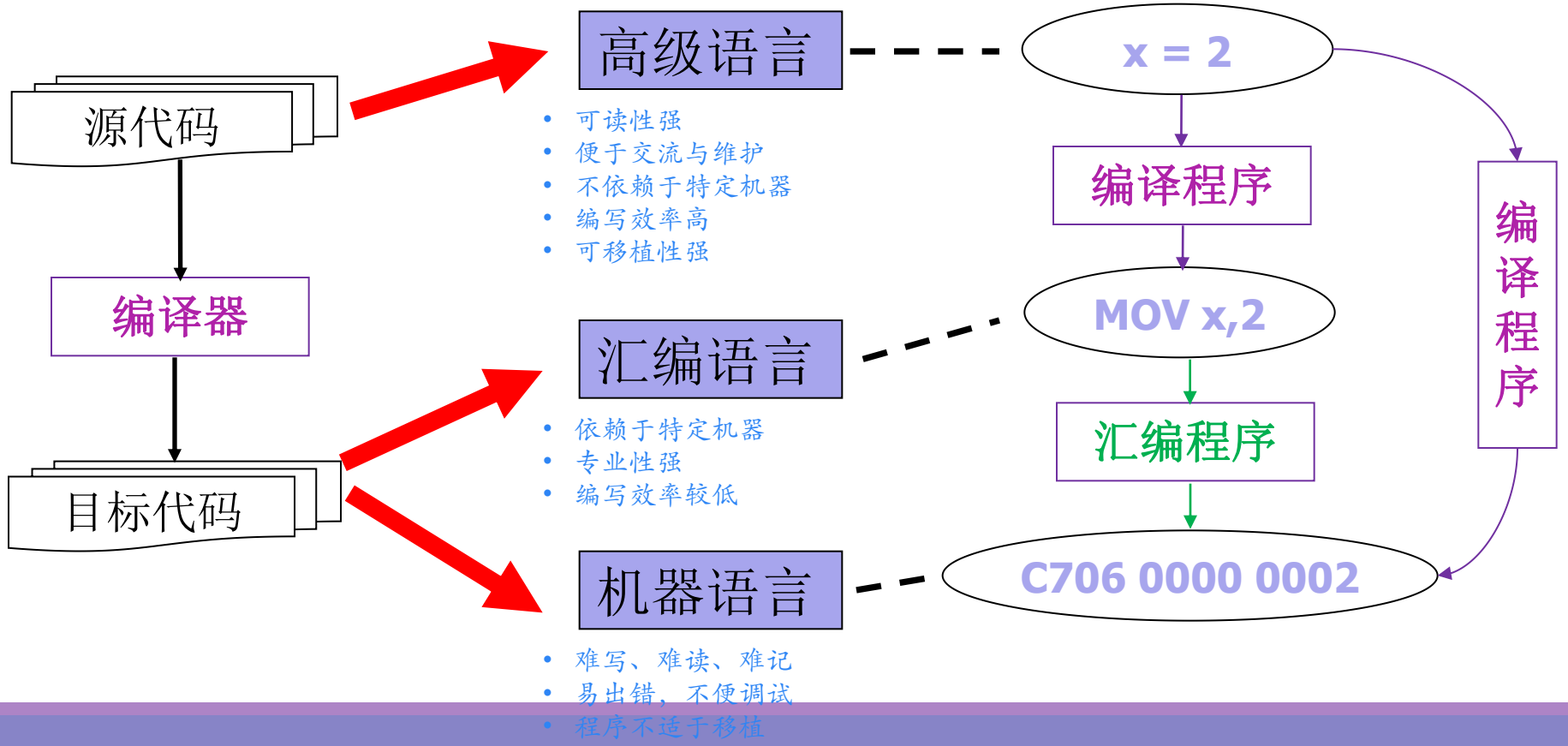
编译器与解释器

主要内容

- 1.1　编译器的概述
- 1.2　编译器的结构
- 1.3　编译技术的应用

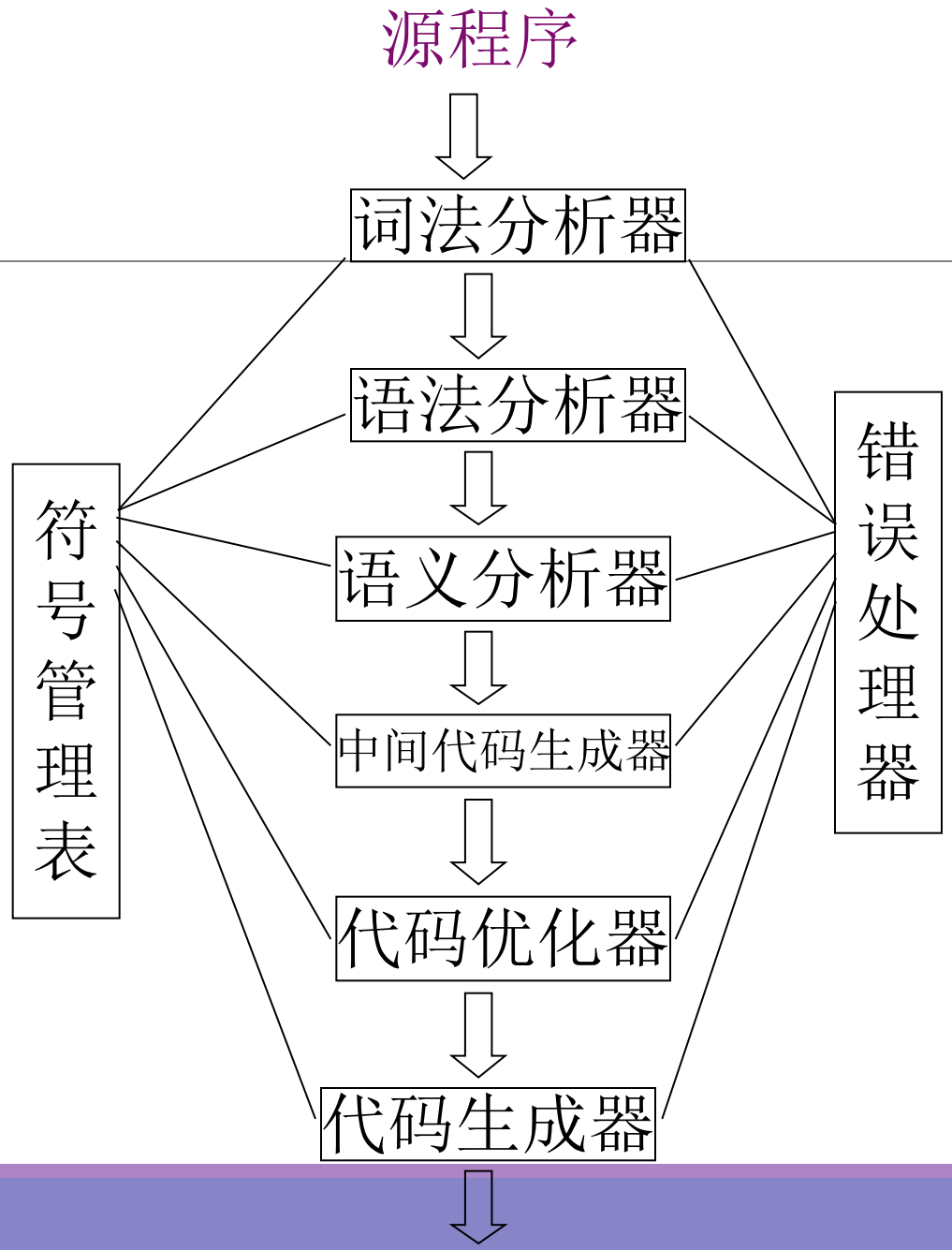
主要内容

- 1.1　编译器的概述



主要内容

- 1.1　编译器的概述
- 1.2　编译器的结构



主要内容

- 1.1　编译器的概述
- 1.2　编译器的结构
- 1.3　编译器的应用

本课程学习内容

	知其然	所以然	工具
词法分析	会用正规式描述词法结构、借助Lex实现词法分析器	简单的自动机知识 正规式→NFA→DFA →词法分析程序	Lex
语法分析	会用上下文无关文法描述语法结构、借助Yacc实现语法分析器	文法→预测分析表→预测分析器 文法→LR分析表→LR分析程序	Yacc
语法制导翻译	设计语法制导定义和翻译模式、借助Yacc实现语法制导翻译	如何在预测分析器、LR分析器中进行属性计算	Yacc

本课程学习内容

	学习内容
类型检查	类型表达式、类型系统 等价判定
中间代码生成	backpatching技术
代码生成	基本块、流图 寄存器分配与指定、窥孔优化
代码优化	数据流分析、 各种类型的优化