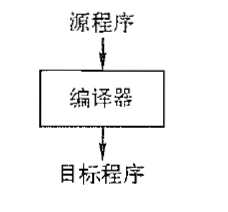
Compilerbau k1-Einleitung

1. Compiler, Interpreter, Hybridcompiler

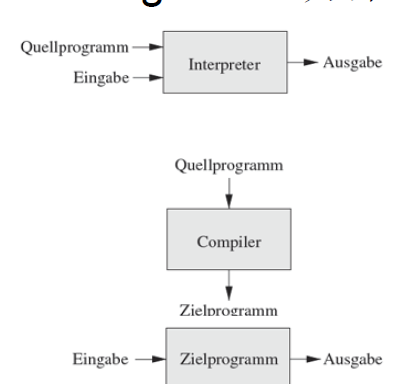
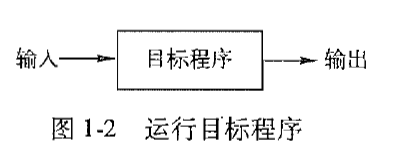
编译器, 解释器, 混合编译器 龙书p17

编译器：把程序翻译成能被计算机执行的语言

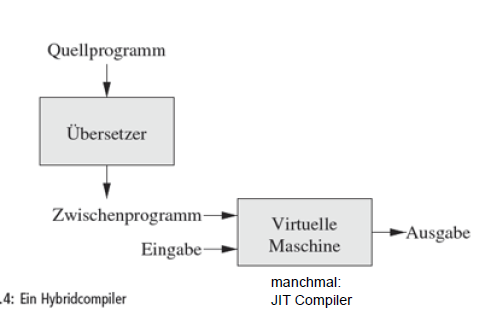
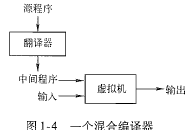
（源语言 等价的 翻译成 目标语言）

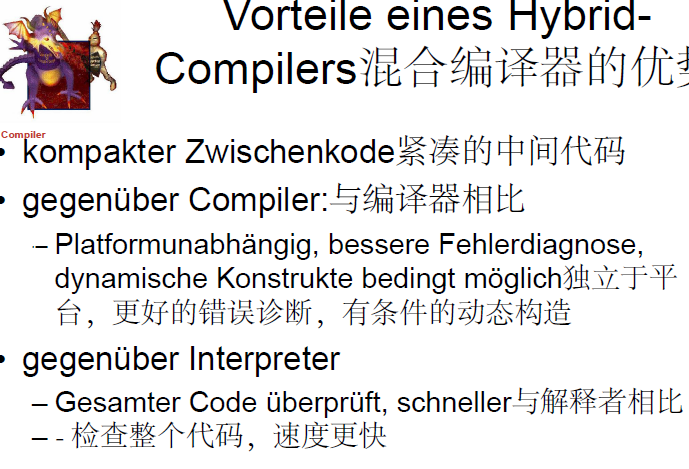


解释器：利用用户的输入执行源程序中指定的操作



混合编译器：如 java语言处理器 结合了 编译和解释功能

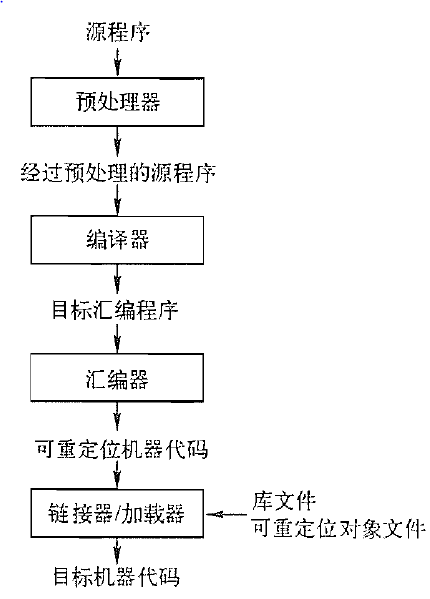


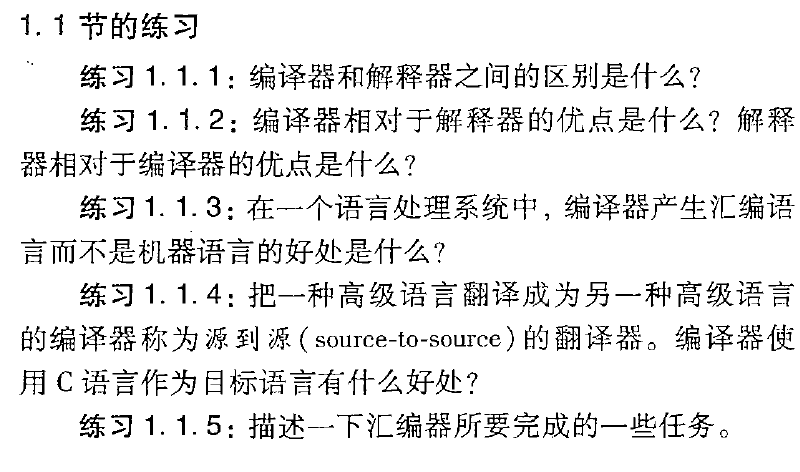


预处理器proprocessor：把源程序聚合在一起 最开始

汇编器 assembler: 编译器的输出结果 整合

链接器linker:解决外部内存地址问题

加载器 loader：把所有可执行目标文件放到内存中执行

练习：

1.1.1

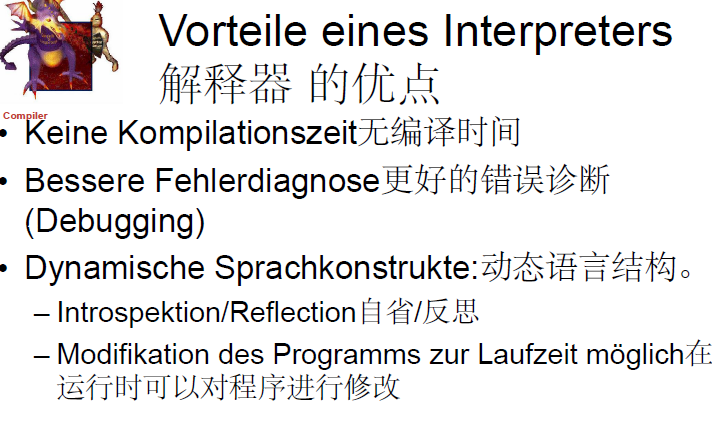
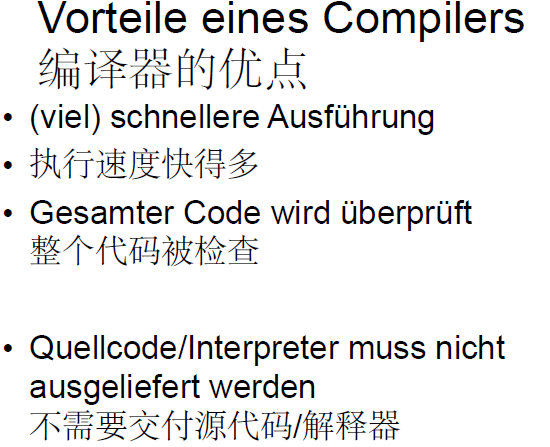
1. 编译器是一种程序，它可以读取一种语言的程序--源语言--并将其翻译成另一种语言的等效程序--目标语言，并报告它在翻译过程中发现的源程序的任何错误

2.解释器直接在用户提供的输入上执行源程序中指定的操作。

3.还有一点，编译器是把整个程序跑完在生成，而解释器是跑一行，执行一行

1.1.2

1. 编译器产生的机器语言目标程序通常比解释器在将输入映射到输出方面快得多
2. 解释器通常能比编译器提供更好的错误诊断，因为它是逐条执行源程序的语句



1.1.3

汇编语言 容易产生输出，也更容易调试。debug

1.1.4

因为C语言相对来说更好理解和调试，相比其他高级语言，C语言更快。

1.1.5

在编译的过程中，处理汇编器产生的汇编程序，并且生成可以重新定位的机器代码。

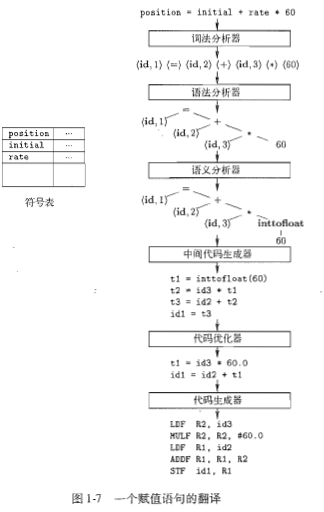
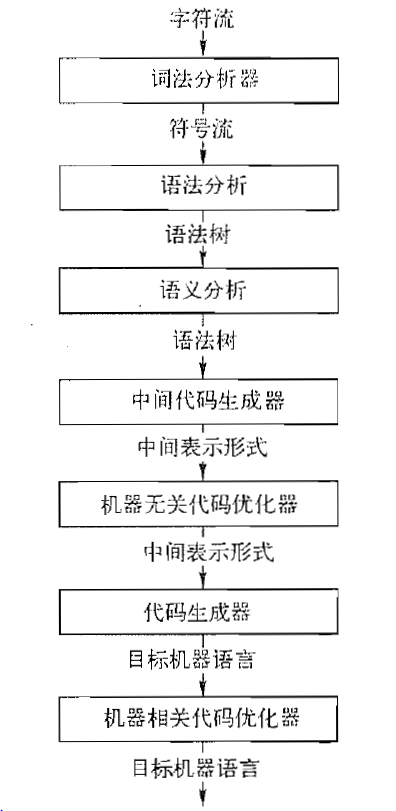
1. Compilerphasen编译阶段

• Compiler: Abfolge von Phasen 阶段的顺序

• Trennung: Back-End, Front-End分离：后端、前端

编译器可以分为2部分：

1. 分析（前端）：把源程序分解成多个组成要素，并在这些要素之上加上语法结构，用这个结构来创建一个中间表示，检查是否出错，收集信息
2. 综合（后端）：根据中间表示和符号表中的信息来构造用户期待的目标程序
3. 它是由一组固定步骤组成的

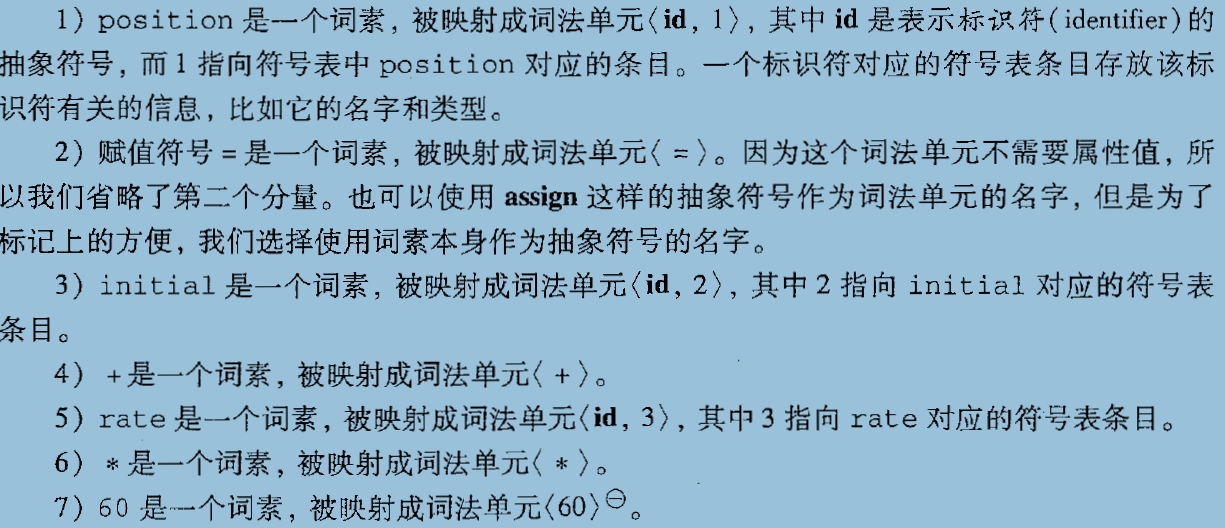


1. • Theorie und Praxis: Modellierung im Compilerdesign (Automaten, Grammatiken, reguläre Ausdrücke, Bäume)理论与实践：编译器设计中的建模

第一步：词法分析：lexical analysis/scanning 读入组成的字符，将他们组成有意义的词素（lexeme)序列，对于每个词，以元组的形式输出：

(token-name,attribute-value)

Bsp: 1.1



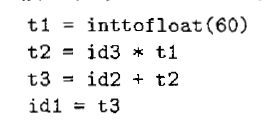
得到1.3

第二步骤 语法分析syntax analysis/parsing:弄成树syntax tree

第三步 语义分析器 semantic analyzer：使用语法树和符号表中的信息来检查源程序是否和语言定义的语义一致

其中重要的有类型检查（type checking)然后自动转换（conereion)

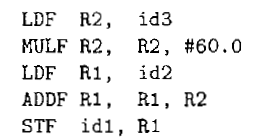
第四步 中间代码生成 三地址代码（three-adress code)



第5步 代码优化 去掉无用的中间代码

2

最后 代码生成



符号表管理

多个步骤组合成一次工序（pass) P23

4.• Einige Programmiersprachenkonzepte

• Statisch/dynamisch, “scope”, …

• 一些编程语言的概念

龙书p31

1.6 程序设计语言基础

1.6.1 设计编译器时，

a.问题：编译器能够对一个程序做出哪些判定

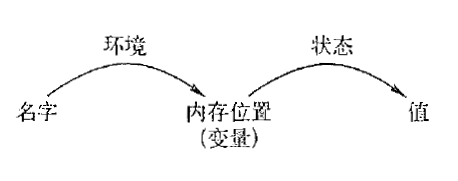
可以在编译时刻决定的：静态策略

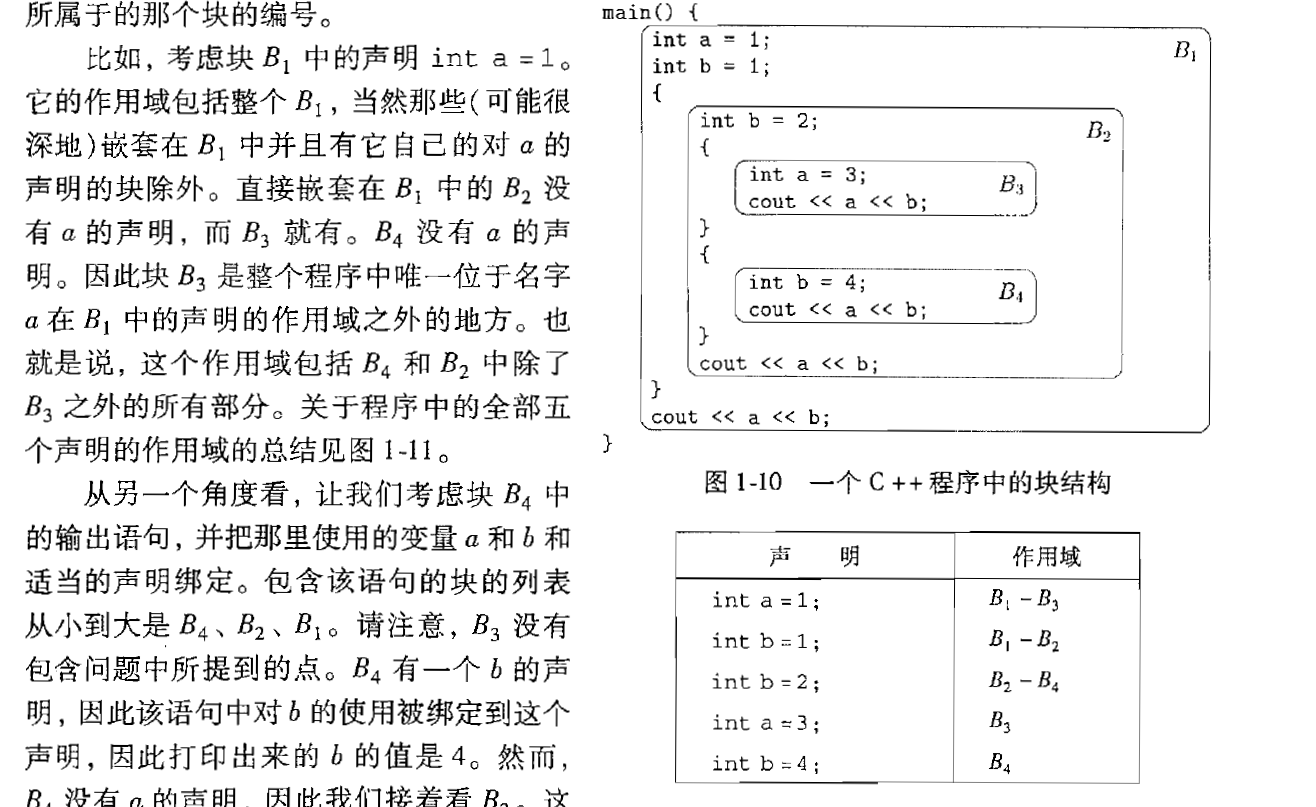
运行程序时决定的：动态策略



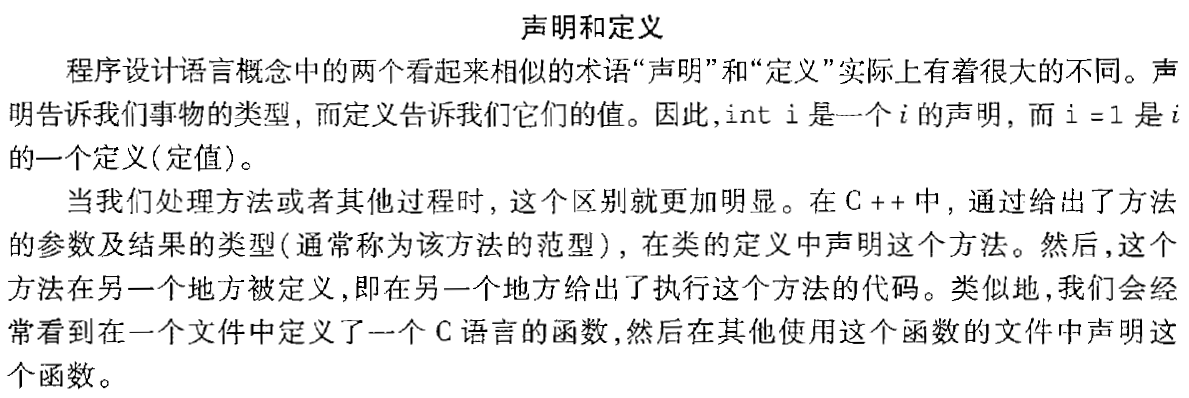
1. 另一个问题是作用域：即变量的使用范围 public private static
2. 元素值是否会迭代覆盖
3. 环境和状态 即：c中的指针，元素值和地址

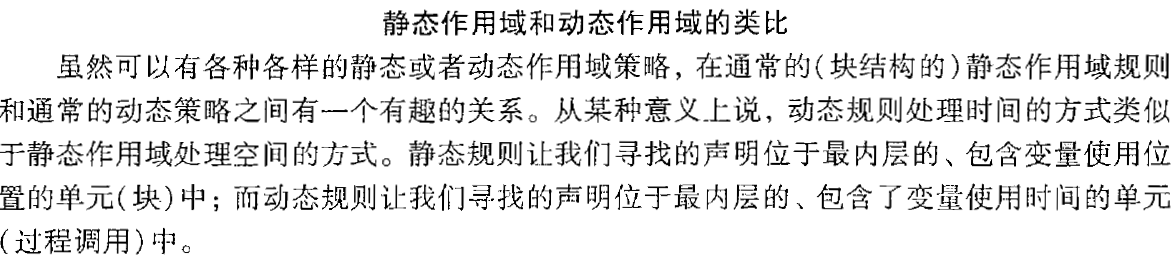
环境 ：名字到存储的映射

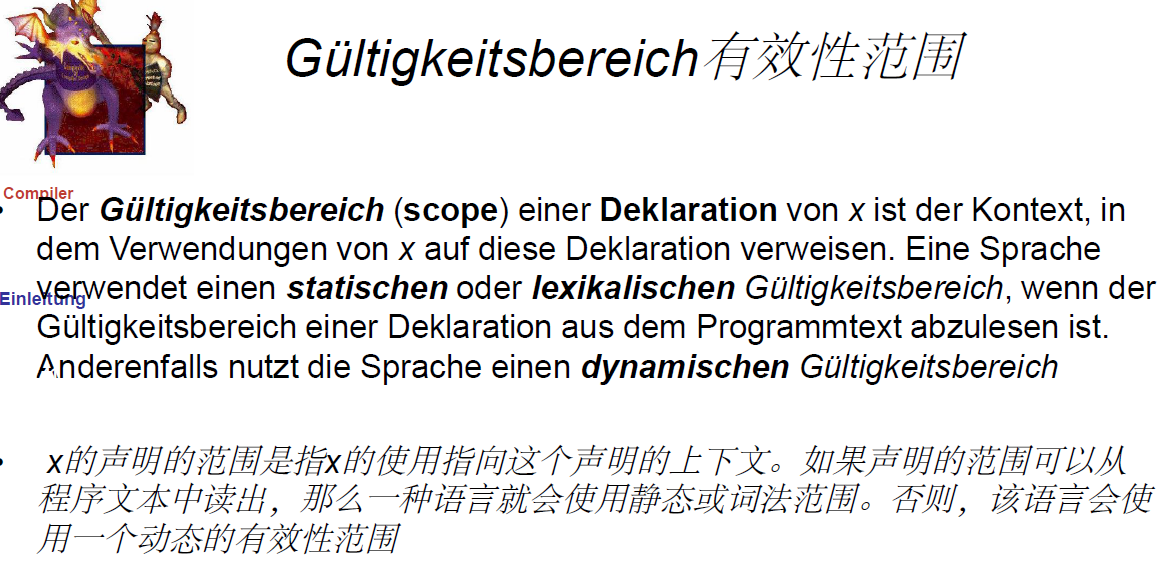
状态：内存地址位置到值的映射

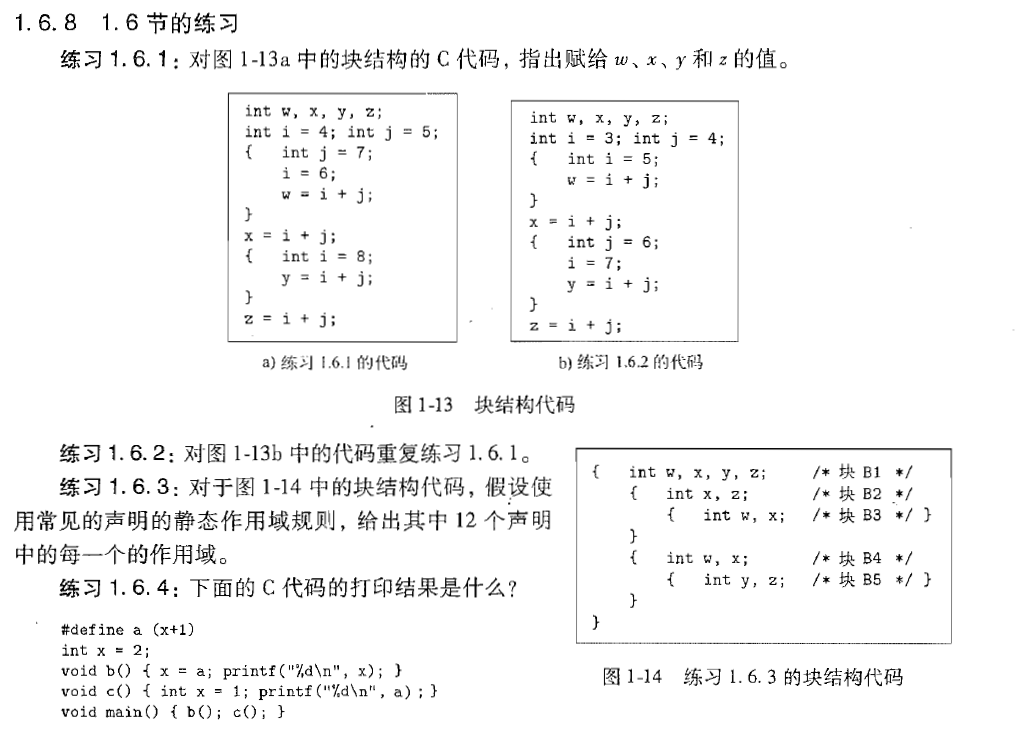
1.6 块 代码块

封装 声明 定义



动态作用域：在程序执行过程中才直到x的作用范围



练习

1. w = 13, x = 11, y = 13, z = 11.

2.w = 9, x = 7, y = 13, z = 11.

3.Block B1:

    declarations:  ->   scope

        w               B1-B3-B4

        x               B1-B2-B4

        y               B1-B5

        z               B1-B2-B5

Block B2:

    declarations:  ->   scope

        x               B2-B3

        z               B2

Block B3:

    declarations:  ->   scope

        w               B3

        x               B3

Block B4:

    declarations:  ->   scope

        w               B4

        x               B4

Block B5:

    declarations:  ->   scope

        y               B5

        z

4.3

2

总结

