普通高等教育“十一五”国家级规划教材

卓越工程师培养计划系列教材
国家级优秀教学团队教学成果

程序设计语言与编译
——语言的设计和实现
（第4版）

王晓斌　陈文宇　余盛季　唐　鸿　田　玲　编著
龚天富　审

中国工信出版集团
電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

普通高等教育“十一五”国家级规划教材

卓越工程师培养计划系列教材　国家级优秀教学团队教学成果

程序设计语言与编译

——语言的设计和实现

（第 4 版）

王晓斌　陈文宇

余盛季　屈鸿　田玲　编著

龚天富　审

電子工業出版社

Publishing House of Electronics Industry

北京 · BEIJING

内容简介

本书是一本计算机专业的宽口径教材，新版覆盖 CCC2001 教程和 CCC2002 教程中编程语言(LP)模块(除自动机理论外)的全部知识点，内容涉及语言及其编译系统的设计要素、设计思想、设计方法、设计技术和设计风格等知识，全书分为上、下篇。上篇，程序设计语言的设计包括：绪论，数据类型，控制结构，程序语言设计；下篇，程序设计语言的实现(编译)包括：编译概述，词法分析，自上而下和自下而上的语法分析，语义分析和中间代码生成，代码优化和目标代码生成，运行时存储空间的组织，MINI 编译器的设计与实现，clang/LLVM 编译器平台介绍；附录包括形式语言与自动机简介。

本书的学习目标是，使读者掌握设计和实现一种程序设计语言的基本思想和方法，具有分析、鉴赏、评价、选择、学习、设计和实现一种语言的基本能力。本书力求简明、通俗，注重可读性，是大学计算机科学和软件工程等专业高级程序设计语言概论及编译技术课程教材，也是软件开发人员的学习参考书。

图书在版编目(CIP)数据

程序设计语言与编译：语言的设计和实现 / 王晓斌等编著. —4 版. — 北京：电子工业出版社，2015.3
ISBN 978-7-121-25482-6

Ⅰ. ①程… Ⅱ. ①王… Ⅲ. ①程序语言－高等学校－教材 Ⅳ. ①TP312

中国版本图书馆 CIP 数据核字(2015)第 024426 号

策划编辑：陈晓莉
责任编辑：陈晓莉
印　　刷：三河市华成印务有限公司
装　　订：三河市华成印务有限公司
出版发行：电子工业出版社
　　北京市海淀区万寿路 173 信箱　邮编　100036
开　　本：787×1 092　1/16　印张：20.75　字数：595 千字
版　　次：1997 年 1 月第 1 版
　　2015 年 3 月第 4 版
印　　次：2015 年 3 月第 1 次印刷
定　　价：45.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：(010)88254888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

服务热线：(010)88258888。

第 4 版前言

本书是一本适合大多数学校计算机专业的宽口径教材，按照 CC2001 教程和 CC2002 教程改写，覆盖了编程语言(PL)模块(除自动机理论外)的全部内容。

作为计算机工作者，必须要与计算机进行交流、通信，所使用的工具是程序设计语言，用来告诉计算机“做什么”和“怎么做”。而程序设计语言数以千计，千姿百态，到底在大学中学习哪些语言才合适？我们的观点是，学会一两种语言的程序设计，更重要的是在此基础上了解语言的共性，这样，就具有鉴赏、评价、选择、学习和设计程序语言的能力。本书的上篇就是为达到上述目的编写的。以抽象的观点，将程序设计语言的共性抽象出来，然后用相应的语言去说明这些共性。

随着计算机技术的发展，有越来越多的人认为，编译程序的设计和实现是专家的工作领域，并非每个计算机专业的学生都需要设计和实现编译程序的知识和能力，有的学校减少了学时，有的学校更砍掉编译课程，取而代之更现代的课程。多年的教学经验告诉我们，编译系统作为计算机系统软件之一，其设计和实现的系统性，能使学生对软件系统的结构形成及系统的建立有充分的了解。因此，本书的下篇讨论了编译程序的五个阶段和每个阶段的基本实现技术。

编译原理课程内容已相对比较成熟，算法相对固定，但编译技术这些年发展迅速，特别是近几年大量编译辅助工具应运而生，且已经运用在实际编译器中。

第四版增加的内容包括 LEX、YACC 介绍、MINI 编译器的设计与实现、clang/LLVM 编译器平台介绍；本书为教师提供了教学参考资料，包括课件、教学指导书和习题答案，需要的老师可通过电子工业出版社的教材服务部获得教学支持。

本书由王晓斌、陈文宇、余盛季、屈鸿和田玲编写，龚天富教授审阅。

电子工业出版社陈晓莉编辑为本书的出版做了大量工作，在此表示衷心感谢。

若书中出现谬误，恳请读者不吝赐教。

作者

2015 年 1 月 于中国·成都

目　录

上篇　程序设计语言的设计

第1章　绪　　论

本章将讨论程序设计语言中的一些重要概念，为深入了解程序设计语言打下基础。最后一节简单介绍程序设计语言的发展历史。

1.1　引　　言

语言是人们交流思想的工具。人类在长期的历史发展过程中，为了交流思想、表达感情和交换信息，逐步形成了语言。这类语言，如汉语和英语，通常称为自然语言(Natual Language)。另一方面，人们为了某种用途，又创造出各种不同的语言，如旗语和哑语，这类语言通常称为人工语言(Artificial Language)。

1946 年出现了第一台电子数字计算机(Electronic Digital Computer)，它一问世就成为强有力的计算工具。只要针对预定的任务(问题)，告诉计算机“做什么”和“怎么做”，计算机就可以自动地进行计算，对给定的问题求解。为此，人们需要将有关的信息告诉计算机，同时也要求计算机将计算结果告诉人们。这样，人与计算机之间就要进行通信(Communication)，既然要通信，就需要信息的载体。人们设计出词汇量少、语法简单、意义明确的语言作为载体，这样的载体通常称为程序设计语言(Programmig Language)。这类语言有别于人类在长期交往中形成的自然语言，它是由人设计创造的，故属于人工语言。本书将讨论这类语言的设计(Design)和实现(Implementation)。

每当设计出一种类型的计算机，就随之产生一种该机器能理解并能直接执行的程序设计语言，这种语言称为机器语言(Machine Language)。用机器语言编写的程序由二进制代码组成，计算机可以直接执行。对人来说，机器语言程序既难编写，又难读懂。为了提高程序的可写性(Writability)和可读性(Readability)，人们将机器语言符号化，于是产生了汇编语言(Assemble Language)。机器语言和汇编语言都是与机器有关的语言(Machine-dependent Language)，通常称为低级语言(Low-level Language)。其他与机器无关的程序设计语言(Machine-independent Language)，通常称为高级语言(High-level Language)。由于计算机只能够理解机器语言，可直接执行用机器语言编写的程序，而用汇编语言和高级语言编写的程序，机器不能直接执行，必须将它们翻译成功能完全等价的机器语言程序才能执行。这个翻译工作是自动进行的，由一个特殊的程序来完成。将汇编语言的程序翻译为机器语言程序的程序称为汇编程序(Assembler)，又称为汇编器；将高级语言程序翻译为低级语言程序的程序称为编译程序(Compiler)，又称为编译器。编写一个高级语言的编译程序的工作，通常称为对这个语言的实现。

每种高级语言都有一个不大的词汇表(Vocabulary)及构造良好的语法(Syntax)规则和语义(Semantics)解释。规定这些基本属性，便于实现高级语言程序到低级语言程序的机器翻译。高级语言较接近于数学语言和自然语言，它具有直观、自然和易于理解的优点。用高级语言编写的程序易读、易写、易交流、易出版和易存档。由于易理解，使程序员容易编出正确的程

序，以便验证程序的正确性，发现错误后也容易修改。因此，用高级语言开发软件的成本比用低级语言低得多。今天，绝大多数的软件都是用高级语言开发的，因此，高级语言是软件开发最重要的工具。

由于高级语言独立于机器，用高级语言编写的程序很容易从一种机器应用到另一种机器上，因而具有较好的可移植性(Portability)。

高级语言至今还没有完全取代低级语言，在一些场合还必须使用机器语言或汇编语言，例如编译程序的目标程序和各种子程序，以及实时应用系统中要求快速执行的代码段等。但是，随着功能强大且具有高级语言和汇编语言特性的 C 语言的出现，使应用汇编语言的人越来越少。

人们在进行科学研究的过程中，总是对具体现象和事物进行观察、分析和综合，以发现它们的重要性质和特征，建立相应的模型。这种通过观察、分析和综合建立模型的过程称为抽象(Abstract)。利用抽象模型，人们可以把注意力集中在有关的性质和特征上，忽略那些不相干的因素。本书在后面的讨论中，大量使用抽象的方法阐述程序设计语言的概念和结构，然后以各种语言中的具体实例来说明这些概念和结构，从而教会读者如何去设计一个程序设计语言。事实上，程序设计语言中处处都使用了抽象概念，例如变量(Variable)是存储单元(Memory Cell，Memory Location)的抽象；子程序(Subroutine 或 Subprogram)是一段多处重复执行的程序段的抽象等。

在此，我们讨论的对象是高级语言，接下来利用抽象的方法讨论高级语言具有的共性概念和结构以及它们的属性。为了叙述简洁，在不引起混淆的情况下，以下将高级语言简称为语言。

一种语言涉及设计者、实现者和使用者，有了设计者和实现者，才可能有使用者。读者在中学或进入大学后，已经使用过这种或那种程序设计语言，也就是说，已经是使用者。本书的目标是引导读者成为语言的设计者和实现者。由于教学学时及篇幅的限制，本书仅给出入门知识和技术，读者如果要真正设计或实现一个语言，尚需查阅相关的文献资料，建议感兴趣的读者阅读参考文献[59]。通过本书的学习，读者可以提高鉴赏和评价语言(或语言设计方案)的能力；了解语言的重要概念、功能和限制，以便具有为某个目的选择一种恰当语言的能力；具有设计一种语言或扩充现有语言的能力；初步具有实现一个语言的能力。最终使读者能够鉴赏、分析、选择、设计和实现程序设计语言。

1.2 强制式语言

通常的高级语言又称为强制式语言(Imperative Language)，本书主要讨论强制式语言的设计和实现。

1.2.1 程序设计语言的分类

语言的分类没有一个统一的标准，通常按不同的尺度有不同的分类方法和结果。例如，按语言设计的理论基础来分类，可分为 4 类语言，即强制式语言，其基础是冯·诺依曼(Von Neumann)模型；函数式语言(Functional Language)的基础是数学函数；逻辑式语言(Logic Language)的基础是数理逻辑谓词演算；对象式语言(Object-oriented Language)的基础是抽象数据类型(Abstract Data Type)。

人们习惯上按语言的发展历程来对语言进行分类。

1. 第一代语言

第一代语言(First-generation Language)通常称为机器语言,它与机器孪生。实际上,它完全依赖于机器的指令系统(Instruction System),以二进制代码表示。这类语言的程序既难编写,又难读懂。

2. 第二代语言

第二代语言(Second-generation Language)通常称为汇编语言,它将机器语言符号化,用符号来代表机器语言的某些属性。例如,用符号名来代表机器语言的地址码。这样可以帮助程序员记忆,摆脱使用二进制代码的烦恼,提高了程序的可写性和可读性。

不同的机器有不同的机器语言和汇编语言,通常人们又把它们称为与机器有关的语言,或面向机器的语言。

3. 第三代语言

第三代语言(Third-generation Language)通常是指高级语言,这类语言的设计基础与冯·诺依曼体系结构有关。高级语言程序按语句顺序执行,因此又称为面向语句的语言(Sentence-oriented Language)。通常,每条语句对应机器的一组命令,因此又称命令式语言(Order Language)。用这类语言编写的程序,实际上是描述对问题求解的计算过程,因此也有人称它为过程式语言(Procedure Language)。

这类语言书写自然,具有更好的可读性、可写性和可修改性(Modifiability),读者使用过的语言大多是这种高级语言。高级语言程序就是要告诉计算机"做什么"和"怎么做"。

4. 第四代语言

第四代语言(Fourth-generation Language)是说明性语言(Declaration Language),它只需要告诉(说明)计算机"做什么",不必告诉计算机"怎么做";也就是说不需要描述计算过程,系统就能自动完成所需要做的工作。所以,这类语言又称为超高级语言或甚高级语言(Very-high-level Language),典型的例子是 SQL 语言。

5. 新一代语言

另一类不同风格的语言,如函数式和逻辑式语言,它们的理论基础和程序设计风格均不同于高级程序设计语言。它们不适合称为第五代语言或第六代语言,因此,语言学家把它们称为新一代程序设计语言。

1.2.2 冯·诺依曼体系结构

当今的计算机模型是由数学家冯·诺依曼提出来的,我们称为冯·诺依曼模型(Von Neumann Model)或冯·诺依曼机(Von Neumann Machine)。直到今天,几乎所有的计算机都是沿用这一模型设计的。1978 年,巴科斯(Backus)在获得图灵奖的颁奖大会上发表演说,批判了冯·诺依曼的体系结构和程序设计风格,称这种结构和风格影响了计算机系统的执行效率,提出了函数式程序设计风格,并发表了 FP 和 FFP 语言。今天,人们越来越多地强调使用并行体系结构和并行程序设计,以提高计算机的执行效率。

下面讨论冯·诺依曼体系结构和它对高级语言的影响。

冯·诺依曼机的概念基于以下思想：一个存储器(用来存放指令和数据)，一个控制器和一个处理器(控制器负责从存储器中逐条取出指令，处理器通过算术或逻辑操作来处理数据)，最后的处理结果还必须送回存储器中。我们可以把这些特点归结为以下4个方面。

(1) 数据和指令以二进制形式存储(数据和指令在外形上没有什么区别，但每位二进制数字有不同的含义)。

(2) "存储程序"方式工作(事先编好程序，执行之前先将程序存放到存储器某个可知的地方)。

(3) 程序顺序执行(可以强行改变执行顺序)。

(4) 存储器的内容可以被修改(存储器的某个单元一旦放入新的数据，则该单元原来的数据立即消失，且被新数据代替)。

冯·诺依曼体系结构的作用体现在命令式语言的下述三大特性上。

(1) 变量　存储器由大量存储单元组成，数据就存放在这些单元中，汇编语言通过对存储单元的命名来访问数据。在命令式语言中，存储单元及其名称由变量的概念来代替。变量代表一个(或一组)已命名的存储单元，存储单元可存放变量的值(Value)，变量的值可以被修改；也正是这种修改，产生了副作用(Side Effect)问题(参见3.3.1节)。

(2) 赋值　使用存储单元概念的另一个后果是每个计算结果都必须存储，即赋值于某个存储单元，从而改变该单元的值。

(3) 重复　指令按顺序执行，指令存储在有限的存储器中；要完成复杂的计算，有效的办法就是重复执行某些指令序列。

1.2.3　绑定和绑定时间

一个对象(或事物)与其各种属性建立起某种联系的过程称为绑定(Binding)。这种联系的建立，实际上就是建立了某种约束。绑定这个词是由英文Binding音译过来的，过去也曾翻译成"联编"、"汇集"、"拼接"或"约束"等。现在之所以选定"绑定"这个词，除了它能形象地表达上述过程外，它还与英文读音一致。

一个程序往往要涉及若干实体，如变量、子程序和语句等。实体具有某些特性，这些特性称为实体的属性(Attribute)。变量的属性有名字(Name)、类型(Type)和保留其值的存储区等。子程序的属性有名字、某些类型的形参(Formal Parameter)和某种参数传递方式的约定等。语句的属性是与之相关的一系列动作。在处理实体之前，必须将实体与相关的属性联系起来(即绑定)。每个实体的绑定信息来源于所谓的描述符(Descriptor)。描述符实际上是各种形式的表格的统称(抽象)，用来存放实体的属性。例如，程序员用类型说明语句来描述变量的类型属性，编译时将它存放在符号表(Symbol Table)中；程序员用数组说明语句来描述一个数组的属性，编译时将这些属性存放在一个专门设计的表格中，这个表格称为数组描述符，又称内情向量(Dope Vector)。

对于计算机科学来说，绑定是一个随处遇到且重复使用的重要概念，借助于它可以阐明许多其他概念。把对象(实体)与它的某个属性联系起来的时刻称为绑定时间(Binding Time)。一旦把某种属性与一个实体绑定，这种约束关系就一直存在下去，直到对这一实体的另一次绑定实现，该属性的约束才会改变。

某些属性可能在语言定义时绑定，例如，FORTRAN语言中的**INTEGER**类型，在语言定义的说明中就绑定了，它由语言编译器来确定这个类型所包含的值的集合。Pascal语言中允

许重新定义**integer** 类型，因此 integer 类型在编译时才能绑定一个具体表示。若一个绑定在运行之前(即编译时)完成，且在运行时不会改变，则称为静态绑定(Static Binding)。若一个绑定在运行时完成(此后可能在运行过程中被改变)，则称为动态绑定(Dynamic Binding)。

今后讲到的许多特性，有的是在编译时所具有的，有的是在运行时所具有的，凡是在编译时确定的特性均称为静态的(Static)；凡是在运行时确定的特性均称为动态的(Dynamic)。

1.2.4 变量

强制式语言最重要的概念之一是变量，它是一个抽象概念，是对存储单元的抽象。如前所述，冯·诺依曼机基于存储单元组成的主存储器(Main Memory)概念，它的每个存储单元用地址来标识，可以对它进行读或写操作。写操作就是指修改存储单元的值，即以一个新值代替原来的值。语言中引入变量的概念，实质上是对一个(或若干个)存储单元的抽象，赋值(Assignment)语句则是对修改存储单元内容的抽象。

变量用名字来标识，此外它还有 4 个属性：作用域(Scope)、生存期(Lifetime)、值和类型。变量可以不具有名字，这类变量称为匿名变量(Anonymous Variable)。下面将讨论上述 4 个属性，以及它们在不同语言中所采用的绑定策略。

1. 变量的作用域

变量的作用域是指可访问该变量的程序范围。在作用域内，变量是可控制的(Manipulable)。变量可以被静态地或动态地绑定于某个程序范围。在作用域内变量是可见的(Visible)，在作用域外变量是不可见的(Invisible)。按照程序的语法结构定义变量的作用域的方法，称为静态作用域绑定(Static Scope Binding)。这时，对变量的每次引用都静态地绑定于一个实际(隐式或显式)的变量说明。大多数传统语言采用静态作用域绑定规则。有的语言在程序执行中动态地定义变量的作用域，这种情况称为动态作用域绑定(Dynamic Scope Binding)。每个变量说明延伸其作用域到它后面的所有指令(语句)，直到遇到一个同名变量的新说明为止。APL，LISP 和 SNOBL4 语言是采用动态作用域规则的语言。

动态作用域规则很容易实现，但掌握这类语言的程序设计比较困难，实现的有效性也偏低。对于动态作用域语言，给定变量绑定于特定说明之后程序执行到的某个特定点，因为其不能静态确定，所以程序很难读懂。

2. 变量的生存期

一个存储区绑定于一个变量的时间区间称为变量的生存期。这个存储区用来保存变量的值。我们将使用术语“数据对象”(Data Object)，或简称“对象”(Object)来同时表示存储区和它保存的值。

变量获得存储区的活动称为分配(Allocation)。某些语言在运行前进行分配，这类分配称为静态分配(Static Allocation)，如 FORTRAN 语言；某些语言在运行时进行分配，这类分配称为动态分配(Dynamic Allocation)，如 C、C++语言。动态分配可以通过两种途径来实现：或者程序员用相关的语句显式提出请求，如 C++语言通过 new 进行；或者在进入变量的作用域时隐式自动进行分配，如 C 语言的活动记录分配。采用什么样的分配，这要看语言是如何规定的。

变量所分配的存储单元的个数，称为变量的长度(Length)。

3. 变量的值

变量在生存期内绑定于一个存储区，该存储区中每个存储单元的内容是以二进制编码方

式表示的变量值，并绑定于变量。编码表示按变量所绑定的类型来进行解释。

在某些语言中，变量的值可能是指向某个对象的指针(Pointer)，若这个对象的值也是指针，那么，可能形成一个引用链(Reference Chain)，这个引用链通常称为访问路径(Access Path)。

若两个变量都有一条访问路径指向同一对象，那么，这两个变量共享(Share)一个对象。经由某个访问路径修改一个共享对象的值时，这种修改能被所有共享这个对象的访问路径获知。多变量共享一个对象，可以节省存储空间。但是，由于一个变量的值被修改，就造成所有共享这个对象的变量的值都被修改，使程序很难读懂。

特别地，访问匿名变量的基本方法是通过访问路径来实现的。

变量的值在程序运行时可以通过赋值操作来修改，因此，变量与它的值的绑定是动态的。一个赋值操作，例如

```
b := a
```

将变量 a 绑定的值复制到变量 b 绑定的存储区内，从而修改变量 b 绑定的值，以一个新值(a 的值)来代替 b 原来的值。

然而，有的语言允许变量与它的值一旦绑定完成就被冻结(Frozen)，不能再修改。例如，在 Pascal 语言中，符号常数语句定义为

```
const pi = 3.1416
```

在 ALGOL 68 中，语句

```
real pi = 3.1416
```

定义了 pi 绑定于值 3.1416。在表达式中使用值 3.1416 可写成

```
circumference := 2 * pi * radius
```

式中的 pi 具有它所绑定的值 3.1416 。这个绑定在整个程序执行过程中不能改变，即不能向 pi 赋新值。显然，语句

```
pi := 2 * pi
```

是错误的。Pascal 语言中的符号常数(Symbolic Constant)的值可以是一个数，也可以是一个字符串，这类变量在编译时即可完成对值的绑定。同时，编译程序可以在编译过程中合法地以这个值去替代程序中出现的相应符号名。ALGOL 68 的符号常数还可以定义为

```
real pi := 3.1416 + x
```

它允许将值定义成变量(或表达式中含变量)，由于有变量，它只能在程序运行中待这些变量建立时才能完成绑定。

对一般变量而言，当它建立时，才会获得所分配的存储区，同时完成变量与存储区的绑定。此时，该变量绑定的值是什么呢？这是变量值的初始化问题，这个问题十分微妙。例如，程序段

```
procedure
    integer x,y;
    x := y + 3;
```

中的语句

```
integer x,y;
```

建立了两个整型变量，允许执行到该过程时，变量 x 和 y 绑定于不同的两个存储单元，但它们绑定的是什么整数值并不确定，原因是未对变量 x 和 y 赋初值。当执行到语句

```
x:=y+3;
```

时 y 绑定什么值是不明确的，即未对 y 赋初值，因而计算的结果 x 绑定什么值也不明确。在上述程序段中未对 y 赋初值就引用了它，程序可能出错。因此，在程序设计中，读者应当遵从对变量“先赋初值后引用”的原则。

虽然有许多方法可以解决“初值问题”，但遗憾的是，通过语言定义来解决这个问题的大多数尝试都不太成功，因为同一语言的不同实现可能采用不同的方法。例如，FORTRAN 语言定义了一个初值语句(Initial Value Statement)，不同的编译程序采用不同的方法来实现，程序员用起来很不方便。要在语言定义中设定一种方式，实现对所有变量初始化是十分困难的。

最简单也是最常用的解决办法是，在语言设计时，忽略初始化问题，在这种情况下，一旦存储区绑定于某个变量，该存储区当前的内容就是该变量绑定的初值。实际上，这个值是随机的位串。类似的方法还有，规定一个非初始化值(Uninitialized Value)，由编译程序在把某存储区分配给变量时，将这个特殊的值赋给每个已分配的存储单元。当程序运行时，每引用一个变量，先检查引用单元的值是否是这个特殊的非初始化值，若是，则出现错误，由系统报告这一错误，这种方法可以彻底解决非初始化问题，但执行效率较低。

有些实现方法提供了定义初值的强制手段，例如，若定义整型变量，初值强制置 0，若定义字符串变量，初值强制置空串。总之，程序员在使用一个语言编程时，一定要注意初值问题。

4. 变量的类型

变量的类型可以看成与变量相关联的值的类，以及对这些值进行的操作(例如，整数加、浮点数加、建立、存取和修改等操作)的说明。类型也可用来解释变量绑定的存储区的内容(二进制位串)的意义。

根据上述对类型的定义，在定义语言时，类型名通常绑定于某一个值类和某一组操作。例如，布尔类型**boolean** 绑定于值**true** 和**false**，以及操作**and**，**or** 和**not**。

语言实现时，值和操作绑定于某种机器的二进制代码表示。例如，**false** 可以绑定于位串 00000000，**true** 绑定于位串 11111111。**and**，**or** 和**not** 操作可以通过表示布尔量位串操作的机器指令来实现。

在某些语言中，程序员可以用类型说明方式来定义新类型。例如，Pascal 语言的语句

```
type t=array[1..10] of boolean
```

在编译时就能建立一个名为 t 的类型，并使它绑定于一个实现(即由 10 个布尔值组成的数组，借助于下标 1～10 可访问每个数组元素)。类型 t 继承了它所代表的数据结构(布尔数组)的所有操作，用数组内的下标能够读取和修改类型 t 的对象的每个分量(元素)。

变量可以静态或动态地绑定于类型，大多数传统语言都采用静态绑定。例如，FORTRAN，ALGOL 60，COBOL，Pascal，ALGOL 68，SIMULA 67，CLU，C 和 Ada 语言等。在这些语言中，变量和它的类型定义之间的绑定通常都是由显式的变量说明来规定的。例如，语句

```
var x,y:integer;
    z:boolean;
```

将变量 x 和 y 绑定为整型，将变量 z 绑定为布尔型。然而，在某些语言中，例如 FORTRAN 语言允许隐式说明并绑定变量的类型。一个未被说明的新变量，它的第一次出现即可