HZ Books
华章教育

计 算 机 科 学 丛 书

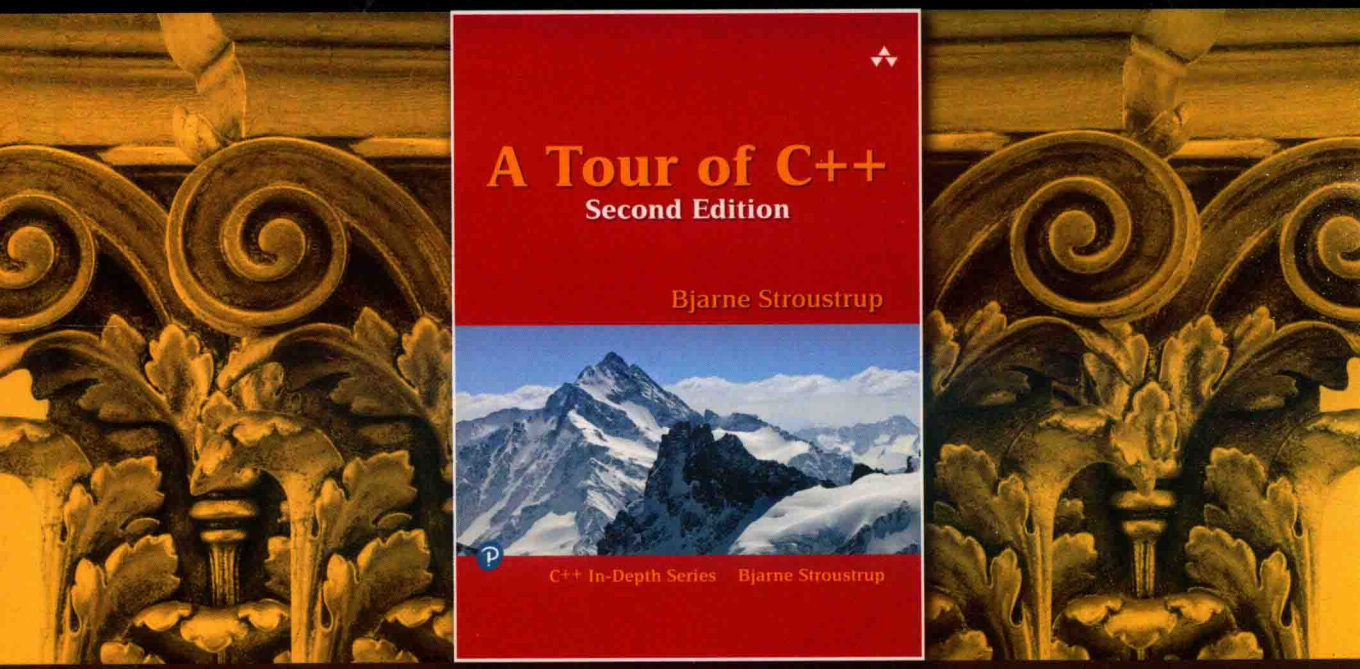
Pearson

原书第2版

C++语言导学

[美] 本贾尼·斯特劳斯特鲁普（Bjarne Stroustrup）著
王刚 译

A Tour of C++
Second Edition



C++语言创造者所著，本书精确描述了现代C++语言的构成、本质与优点

机械工业出版社
China Machine Press

计 算 机 科 学 丛 书

原书第2版

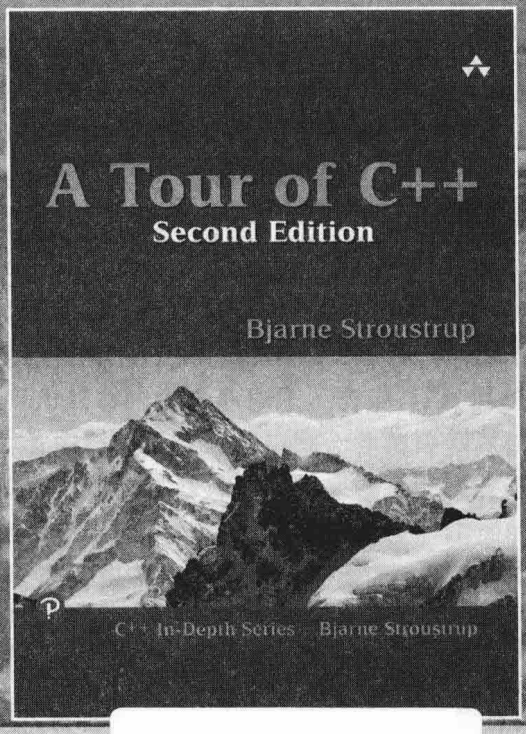
C++语言导学

[美] 本贾尼·斯特劳斯特鲁普（Bjarne Stroustrup）著

王刚 译

A Tour of C++

Second Edition



机械工业出版社
China Machine Press

图书在版编目（CIP）数据

C++ 语言导学（原书第 2 版）/（美）本贾尼·斯特劳斯特鲁普（Bjarne Stroustrup）著；王刚译．—北京：机械工业出版社，2019.8
（计算机科学丛书）
书名原文：A Tour of C++, Second Edition

ISBN 978-7-111-63328-0

I. C… II. ① 本… ② 王… III. C++ 语言 - 程序设计 IV. TP312.8

中国版本图书馆 CIP 数据核字（2019）第 164242 号

本书版权登记号：图字 01-2018-6463

本书简洁、自成体系，包含 C++ 语言大多数主要特性和标准库组件。当然，这些并未深入介绍，而是给予程序员一个有意义的语言概述、一些关键的例子以及起步阶段的实用帮助。本书的目标不是教你如何编程，它也不可能是你精通 C++ 的唯一资源。但是，如果你是一名 C 或 C++ 程序员，希望更加熟悉现在的 C++ 语言，或者你是一名精通其他语言的程序员，希望获得有关现代 C++ 语言本质和优点的精确描述，本书是最优选择。

出版发行：机械工业出版社（北京市西城区百万庄大街 22 号　邮政编码 100037）
责任编辑：张梦玲　　责任校对：李秋荣
印　　刷：三河市宏图印务有限公司　　版　　次：2019 年 9 月第 1 版第 1 次印刷
开　　本：185mm × 260mm　1/16　　印　　张：14.25
书　　号：ISBN 978-7-111-63328-0　　定　　价：79.00 元

客服电话：（010）88361066　88379833　68326294　　投稿热线：（010）88379604
华章网站：www.hzbook.com　　读者信箱：hzjsj@hzbook.com

文艺复兴以来，源远流长的科学精神和逐步形成的学术规范，使西方国家在自然科学的各个领域取得了垄断性的优势；也正是这样的优势，使美国在信息技术发展的六十多年间名家辈出、独领风骚。在商业化的进程中，美国的产业界与教育界越来越紧密地结合，计算机学科中的许多泰山北斗同时身处科研和教学的最前线，由此而产生的经典科学著作，不仅擘划了研究的范畴，还揭示了学术的源变，既遵循学术规范，又自有学者个性，其价值并不会因年月的流逝而减退。

近年，在全球信息化大潮的推动下，我国的计算机产业发展迅猛，对专业人才的需求日益迫切。这对计算机教育界和出版界都既是机遇，也是挑战；而专业教材的建设在教育战略上显得举足轻重。在我国信息技术发展时间较短的现状下，美国等发达国家在其计算机科学发展的几十年间积淀和发展的经典教材仍有许多值得借鉴之处。因此，引进一批国外优秀计算机教材将对我国计算机教育事业的发展起到积极的推动作用，也是与世界接轨、建设真正的世界一流大学的必由之路。

机械工业出版社华章公司较早意识到“出版要为教育服务”。自 1998 年开始，我们就将工作重点放在了遴选、移译国外优秀教材上。经过多年的不懈努力，我们与 Pearson、McGraw-Hill、Elsevier、MIT、John Wiley & Sons、Cengage 等世界著名出版公司建立了良好的合作关系，从它们现有的数百种教材中甄选出 Andrew S. Tanenbaum、Bjarne Stroustrup、Brian W. Kernighan、Dennis Ritchie、Jim Gray、Afred V. Aho、John E. Hopcroft、Jeffrey D. Ullman、Abraham Silberschatz、William Stallings、Donald E. Knuth、John L. Hennessy、Larry L. Peterson 等大师名家的一批经典作品，以“计算机科学丛书”为总称出版，供读者学习、研究及珍藏。大理石纹理的封面，也正体现了这套丛书的品位和格调。

“计算机科学丛书”的出版工作得到了国内外学者的鼎力相助，国内的专家不仅提供了中肯的选题指导，还不辞劳苦地担任了翻译和审校的工作；而原书的作者也相当关注其作品在中国的传播，有的还专门为其书的中译本作序。迄今，“计算机科学丛书”已经出版了近 500 个品种，这些书籍在读者中树立了良好的口碑，并被许多高校采用为正式教材和参考书籍。其影印版“经典原版书库”作为姊妹篇也被越来越多实施双语教学的学校所采用。

权威的作者、经典的教材、一流的译者、严格的审校、精细的编辑，这些因素使我们的图书有了质量的保证。随着计算机科学与技术专业学科建设的不断完善和教材改革的逐渐深化，教育界对国外计算机教材的需求和应用都将步入一个新的阶段，我们的目标是尽善尽美，而反馈的意见正是我们达到这一终极目标的重要帮助。华章公司欢迎老师和读者对我们的工作提出建议或给予指正，我们的联系方法如下：

华章网站：www.hzbook.com
电子邮件：hzjsj@hzbook.com
联系电话：（010）88379604
联系地址：北京市西城区百万庄南街 1 号
邮政编码：100037

华章教育
华章科技图书出版中心

译 者 序

A Tour of C++, Second Edition

C++ 是一门经典的程序设计语言。

Bjarne Stroustrup 是 C++ 的设计者、最初的实现者和 ISO 标准的主要制定者。

《A Tour of C++》是 Bjarne Stroustrup 推出的一本能令有经验的程序员快速了解现代 C++ 语言的小册子。与作者的其他著作相比，本书有三个特点。一是“新”：本书以快速导览的形式介绍 C++，是作者的一次新的尝试。从写作手法、章节组织到示例选取都力图推陈出新，一改语言类书籍教条枯燥的通病，文字间洋溢着新意。作为第 2 版，在内容选取、结构组织上较之前的第 1 版进行了全面更新。二是“薄”：本书篇幅短小，每个主题多则二三十页少则十余页即叙述完成，不论随身携带或者置于案头，读者都可以在较短时间内读完本书并从中受益。三是“精”：本书的文字虽少，内容却不少，甚至可以说非常丰富。不但涉及 C++ 的绝大多数语言特性以及重要的标准库组件，而且涵盖了 C++17 标准及未来的 C++20 标准中的很多新内容。

在翻译过程中我们有这样一个体会，与其说作者在书中介绍一些语法和技术，不如说他在传递思想。传递他在发明、设计和不断完善 C++ 语言的过程中的所思和所虑；当思想和编程实践产生碰撞时，他又基于丰富的实践经验给出了非常中肯的建议。

很多学习者和程序员常常会有这样的疑问：C++ 是什么？读完本书，相信你会得到满意的答案。

由于时间紧促且译者水平有限，书中的不当之处恳请广大读者批评指正。

2019 年夏

于南开园

前　言

A Tour of C++, Second Edition

教而至简，不亦乐乎。

——西塞罗

现在的 C++ 感觉就像是一种新的语言。与 C++98 相比，使用现在的 C++ 我能更清晰、更简单、更直接地表达思想。而且，编译器可以更好地检查程序中的错误，程序的运行速度也提高了。

本书给出 C++ 语言的一个概述，这里所说的 C++ 是由当前的 ISO C++ 标准 C++17 定义的，由主要的 C++ 提供商实现。此外，本书还会介绍一些目前在使用的 ISO 技术规范定义的概念和模块，但它们在 C++20 尚无计划包含进标准中。

就像其他任何一种现代编程语言一样，C++ 规模庞大且提供了非常丰富的库，这是高效编程所需的。这本小册子的目的是让一个有经验的程序员快速了解现代 C++ 语言，因此它覆盖了 C++ 大多数主要的语言特性和标准库组件。读者花费几个小时就能读完这本书，但显然要想写出漂亮的 C++ 程序绝非一日之功。好在本书的目的并非让读者熟练掌握一切，而只是给出一个概览，给出一些关键的例子，帮助读者开始自己的 C++ 之旅。

假设读者已经拥有了一些编程经验。如果没有，建议你先找一本入门教材学习，比如《Programming: Principles and Practice Using C++，Second Edition》（C++ 程序设计原理与实践（第 2 版））[Stroustrup, 2014]，然后再来学习本书。即便你曾经编写过程序，你使用的语言或者编写的应用也可能在风格或形式上与本书所介绍的 C++ 相距甚远。

我们用城市观光的例子来说明本书的作用，比如游览哥本哈根或者纽约。在短短几个小时之内，你可能会匆匆游览几个主要的景点，听一些有趣的传说或故事，然后听取建议接下来做什么。仅靠这样一段旅程，你无法真正了解这座城市，也无法完全理解听到和看到的东西，更无法熟悉这座城市正式的和非正式的生存法则。毕竟想要真正了解一座城市，你必须生活在其中，而且往往需要多年。不过如果幸运的话，此时你已经对城市的概貌有了一些了解，知道了它的某些特殊之处，并且对某些方面产生了兴趣。在这段旅程之后，你就可以开始真正的探索了。

本书的风格就像这段旅程，它会为你介绍 C++ 语言的主要特性，这是按其所支持的程序设计风格来呈现的，例如面向对象编程和泛型编程。本书不准备提供一个详细的、手册式的、逐条特性的 C++ 语言描述。遵循优秀教科书的传统，我努力在使用每个语言特性之前对其进行解释，但实际情况并不总能允许我这样做，而且并不是每个人都会严格按顺序阅读本书。因此，我鼓励读者使用交叉引用和索引。

类似地，本书以示例的方式介绍标准库，而非逐一列举标准库特性。本书没有介绍 ISO 标准之外的库，读者需要的话可以查阅相关资料，例如 [Stroustrup, 2013] 和 [Stroustrup, 2014]，网络上也有大量（质量参差不齐）的其他资料，如 [Cppreference]。例如，当我提到一个标准库函数或类时，很容易就能找到它的定义，并且通过查找其文档，能找到很多相关的资料。

本书力求把 C++ 作为一个整体呈现在读者面前，而非像千层糕一样逐层地介绍。因此，本书不细分某个语言特性是属于 C、C++98 的一部分还是新的 C++11、C++14 或 C++17。这种信息可在第 16 章（历史和兼容性）中找到。本书聚焦基础并力求简洁，但也未能完全抵抗过度阐述新特性的诱惑。这看起来也满足了很多已经了解旧版本 C++ 的读者的好奇心。

一本程序设计语言参考手册或标准会简单陈述可以做什么，但程序员通常对学习如何用好语言更感兴趣。达到这个目的一方面要靠主题的选择，另一方面要靠文字的组织，特别是建议部分。关于优秀的现代 C++ 语言是怎样构成的更多建议可在《 C++ Core Guidelines 》（C++ 核心准则）[Stroustrup, 2015] 一书中找到。对于希望继续深入探索本书介绍的思想的读者，这是一本很好的书。你可能注意到了，《 C++ Core Guidelines 》和本书在建议的呈现上甚至建议的编号方式上都惊人地相似。其中一个原因是本书第 1 版是最初的《 C++ Core Guidelines 》的主要参考资源。

致谢

本书的一些内容源自《 C++ 程序设计语言（第 4 版）》（TC++PL4）[Stroustrup, 2013]，因此要感谢帮助我完成 TC++PL4 的所有同仁。

感谢帮助我完成并校对本书第 1 版的所有同仁。

感谢 Morgan Stanley 给予我时间进行本书的写作。感谢哥伦比亚大学 2018 年春季课程“使用 C++ 设计程序”的所有学生找出了本书最初草稿中的很多拼写问题和错误并给出了很多建设性的意见。

感谢 Paul Anderson、Chuck Allison、Peter Gottschling、William Mons、Charles Wilson 和 Sergey Zubkov 审阅了本书并给出了很多改进建议。

本贾尼·斯特劳斯特鲁普

曼哈顿，纽约

第 1 章
A Tour of C++, Second Edition

基础知识

首要任务，干掉所有语言专家。
——《亨利六世》(第二部分)

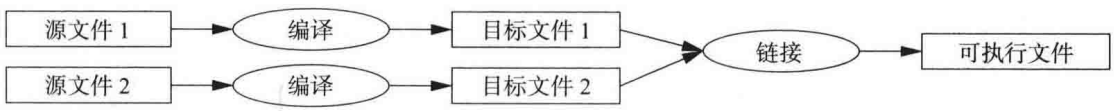
- 引言
- 程序
  Hello, World!
- 函数
- 类型、变量和算术运算
  算术运算；初始化
- 作用域和生命周期
- 常量
- 指针、数组和引用
  空指针
- 检验
- 映射到硬件
  赋值；初始化
- 建议

1.1 引言

本章简要介绍 C++ 的符号系统、C++ 的内存模型和计算模型以及将代码组织为程序的基本机制。这些语言设施支持最为常见的 C 语言编程风格，我们称之为过程式编程(procedural programming)。

1.2 程序

C++ 是一种编译型语言。为了让程序运行，首先要用编译器处理源代码文本，生成目标文件，然后再用连接器将目标文件组合成可执行程序。一个 C++ 程序通常包含多个源代码文件，通常简称为源文件（source file）。



可执行程序都是为特定的硬件 / 系统组合创建的，不具可移植性。比如说，Mac 上的可执行程序就无法移植到 Windows PC 上。当谈论 C++ 程序的可移植性时，通常是指源代码的可移植性，即源代码可以在不同系统上成功编译并运行。

ISO 的 C++ 标准定义了两类实体：

- 核心语言特性（core language feature），例如内置类型（如 `char` 和 `int`）和循环（如 `for` 语句和 `while` 语句）；
- 标准库组件（standard-library component），比如容器（如 `vector` 和 `map`）和 I/O 操作（如 `<<` 和 `getline()`）。

每个 C++ 实现都提供标准库组件，它们其实也是非常普通的 C++ 代码。换句话说，C++ 标准库可以用 C++ 语言本身实现（仅在实现线程上下文切换这样的功能时才使用少量机器代码）。这意味着 C++ 在面对大多数高要求的系统编程任务时既有丰富的表达力，同时也足够高效。

C++ 是一种静态类型语言，这意味着任何实体（如对象、值、名称和表达式）在使用时都必须已被编译器了解。对象的类型决定了能在该对象上执行的操作。

Hello, World!

最小的 C++ 程序如下所示：

```
int main() { }          // 最小的 C++ 程序
```

这段代码定义了一个名为 `main` 的函数，该函数既不接受任何参数，也不做什么实际工作。

在 C++ 中，花括号 `{}` 表示成组的意思，上面的例子里，它指出函数体的首尾边界。从双斜线 `//` 开始直到该行结束是注释，注释只供人阅读和参考，编译器会直接略过注释。

每个 C++ 程序必须有且只有一个名为 `main()` 的全局函数，它是程序执行的起点。如果 `main()` 返回一个 `int` 整数值，则它是程序返回给“系统”的值。如果 `main()` 不返回任何内容，则系统也会收到一个表示程序成功完成的值。`main()` 返回非零值表示程序执行失败。并非每个操作系统和执行环境都会利用这个返回值：基于 Linux/Unix 的环境通常会用到，而基于 Windows 的环境很少会用到。

通常情况下，程序会产生一些输出。例如，下面这个程序输出 `Hello, World!`：

```
#include <iostream>

int main()
{
    std::cout << "Hello, World!\n";
}
```

`#include<iostream>` 这一行指示编译器把 `iostream` 中涉及的标准流 I/O 设施的声明包含（include）进来。如果没有这些声明的话，表达式

```
std::cout << "Hello, World!\n"
```

无法正确执行。运算符 `<<`（“输出”）把它的第二个参数写入到第一个参数。在这个例子里，字符串字面值 `"Hello, World!\n"` 被写入到标准输出流 `std::cout`。字符串字面值是指被一对双引号包围的字符序列。在字符串字面常量中，反斜线 `\` 紧跟另一个字符组成一个“特殊字符”。在这个例子中，`\n` 是换行符，因此最终的输出结果是 `Hello, World!` 后跟一个换行。

`std::` 指出名字 `cout` 可在标准库名字空间（参见 3.4 节）中找到。本书在讨论标准特

性时通常会省略掉 std::，3.4 节将介绍如何不使用显式限定符而让名字空间中的名字可见。

基本上所有可执行代码都要放在函数中，并且被 main() 直接或间接地调用。例如：

```
#include <iostream>          // 包含（“引入”）I/O 流库的声明

using namespace std;         // 使得 std 中的名字变得可见，而无须再使用 std::（参见 3.4 节）

double square(double x)      // 计算一个双精度浮点数的平方
{
    return x*x;
}

void print_square(double x)
{
    cout << "the square of " << x << " is " << square(x) << "\n";
}

int main()
{
    print_square(1.234);     // 打印：the square of 1.234 is 1.52276（1.234 的平方是 1.52276）
}
```

“返回类型” void 表示函数 print_square( ) 不返回任何值。

1.3 函数

在 C++ 程序中完成某些任务的主要方式就是调用函数。你若想描述如何进行某个操作，把它定义成函数是标准方式。注意，函数必须先声明后调用。

一个函数声明需要给出三部分信息：函数的名字、函数的返回值类型（如果有的话）以及调用该函数必须提供的参数数量和类型。例如：

```
Elem* next_elem();       // 无参数，返回一个指向 Elem 的指针（一个 Elem*）
void exit(int);          // 接受一个 int 参数，不返回任何值
double sqrt(double);     // 接受一个 double 参数，返回的也是一个 double 类型
```

在一个函数声明中，返回类型位于函数名之前，参数类型位于函数名之后，并用括号包围起来。

参数传递的语义与初始化的语义是相同的（参见 3.6.1 节）。即，编译器会检查参数的类型，并且在必要时执行隐式参数类型转换（参见 1.4 节）。例如：

```
double s2 = sqrt(2);          // 用参数 double{2} 调用 sqrt() 函数
double s3 = sqrt("three");    // 错误：sqrt() 函数要求参数类型是 double
```

我们不应低估这种编译时检查和类型转换的价值。

函数声明可以包含参数名，这有助于读者理解程序的含义。但实际上，除非该声明同时也是函数的定义，否则编译器会简单忽略参数名。例如：

```
double sqrt(double d);   // 返回 d 的平方根
double square(double);   // 返回参数的平方结果
```

返回类型和参数类型属于函数类型的一部分。例如：

```
double get(const vector<double>& vec, int index); // 函数类型：double(const vector<double>&, int)
```

函数可以是类的成员（参见 2.3 节和 4.2.1 节）。对这种成员函数（member function），类

名也是函数类型的一部分，例如：

```
char& String::operator[](int index);                    // 函数类型：char& String::(int)
```

我们都希望自己的代码易于理解，因为这是提高代码可维护性的第一步。而令程序易于理解的第一步，就是将计算任务分解为有意义的模块（用函数和类表达）并为它们命名。这样的函数就提供了计算的基本词汇，就像类型（包括内置类型和用户自定义类型）提供了数据的基本词汇一样。C++ 标准算法（如 find、sort 和 iota）提供了一个良好开端（参见第 12 章），接下来我们就能用这些表示通用或者特殊任务的函数组合出更复杂的计算模块了。

代码中错误的数量通常与代码的规模和复杂程度密切相关，多使用一些更短小的函数有助于降低代码的规模和复杂度。例如，通过定义函数来执行一项专门任务，在其他代码中我们就不必再为其编写一段对应的特定代码，将任务定义为函数促使我们为这些任务命名并明确它们的依赖关系。

如果程序中存在名字相同但参数类型不同的函数，则编译器会为每次调用选择最恰当的
4 版本。例如：

```
void print(int);        // 接受一个整型参数
void print(double);     // 接受一个浮点型参数
void print(string);     // 接受一个字符串型参数

void user()
{
    print(42);                  // 调用 print(int)
    print(9.65);                // 调用 print(double)
    print("Barcelona");         // 调用 print(string)
}
```

如果存在两个可供选择的函数且它们难分优劣，则编译器认为此次调用具有二义性并报错。例如：

```
void print(int,double);
void print(double,int);

void user2()
{
    print(0,0);     // 错误：二义性调用
}
```

定义多个具有相同名字的函数就是我们所熟知的*函数重载*（function overloading），它是泛型编程（参见 7.2 节）的一个基本部分。当重载函数时，应保证所有同名函数都实现相同的语义。print() 函数就是一个这样的例子：每个 print() 都将其实参打印出来。

1.4　类型、变量和算术运算

每个名字、每个表达式都有自己的类型，类型决定了能对名字和表达式执行的操作。例如，下面的声明

```
int inch;
```

指定 inch 的类型为 int，也就是说，inch 是一个整型变量。

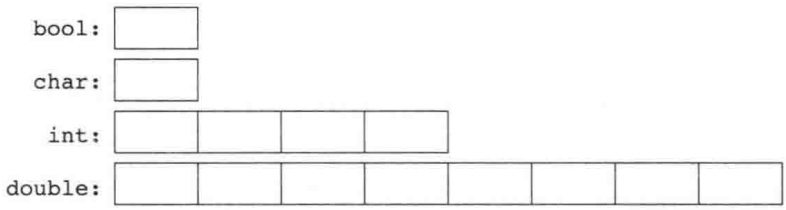
一个*声明*（declaration）是一条语句，为程序引入一个实体，并为该实体指明类型：

- 一个类型（type）定义了一组可能的值以及一组（对象上的）操作。
- 一个对象（object）是存放某种类型值的内存空间。
- 一个值（value）是一组二进制位，具体的含义由其类型决定。
- 一个变量（variable）是一个命名的对象。

C++ 就像一个小型动物园，提供了各种基本类型，但我不是一个动物学家，因此在这里不会列出全部的 C++ 基本类型。你可以在网络上的参考资料中找到它们，如 [Stroustrup,2003] 或 [Cppreference]。一些例子如下：


```
bool        // 布尔值，可取 true 或 false
char        // 字符，如 'a' 'z' 和 '9'
int         // 整数，如 -273、42 和 1066
double      // 双精度浮点数，如 -273.15、3.14 和 6.626e-34
unsigned    // 非负整数，如 0、1 和 999（用于位逻辑运算）
```

每种基本类型都直接对应硬件设施，具有固定的大小，这决定了其中所能存储的值的范围：



一个 char 变量的实际大小为给定机器上存放一个字符所需的空间（通常是一个 8 位的字节），其他类型的大小都是 char 大小的整数倍。类型的大小是依赖于实现的（即，在不同机器上可能不同），可使用 sizeof 运算符获得这个值。例如，sizeof(char) 等于 1，sizeof(int) 通常是 4。

数包括浮点数和整数。

- 浮点数是通过小数点（如 3.14）或指数（如 3e-2）来区分的。
- 整数字面值默认是十进制（如，42 表示四十二）。前缀 0b 指示二进制（基为 2）的整数字面值（如 0b10101010）。前缀 0x 指示十六进制（基为 16）整数字面值（如 0xBAD1234）。前缀 0 指示八进制（基为 8）的整数字面值（如 0334）。

为了令长字面常量对人类更易读，我们可以使用单引号（'）作为数字分隔符。例如，π 大约为 3.14159'26535'89793'23846'26433'83279'50288，如果你更喜欢十六进制，就是 0x3.243F'6A88'85A3'08D3。

1.4.1 算术运算

算术运算符可用于上述基本类型的恰当组合：

```
x+y     // 加法
+x      // 一元加法
x-y     // 减法
-x      // 一元减法
x*y     // 乘法
x/y     // 除法
x%y     // 整数取余（取模）
```

6 比较运算符也是如此：

```
x==y    // 相等
x!=y    // 不相等
x<y     // 小于
x>y     // 大于
x<=y    // 小于等于
x>=y    // 大于等于
```

除此之外，C++ 还提供了逻辑运算符：

```
x&y     // 位与
x|y     // 位或
x^y     // 位异或
~x      // 按位求补
x&&y    // 逻辑与
x||y    // 逻辑或
!x      // 逻辑非（否定）
```

位逻辑运算符对运算对象逐位计算，产生结果的类型与运算对象的类型一致。逻辑运算符 `&&` 和 `||` 根据运算对象的值返回 `true` 或者 `false`。

在赋值运算和算术运算中，C++ 会在基本类型之间进行有意义的转换，以便它们能自由地混合运算：

```
void some_function()    // 不返回值的函数
{
    double d = 2.2;     // 初始化浮点数
    int i = 7;          // 初始化整数
    d = d+i;            // 将求和结果赋给 d
    i = d*i;            // 将乘积结果赋给 i；注意，double 类型的 d*i 被截断为一个 int
}
```

表达式中使用的类型转换称为*常规算术类型转换*（usual arithmetic conversion），其目的是确保表达式以运算对象中最高的精度进行计算。例如，对一个 `double` 和一个 `int` 求和，执行的是双精度浮点数的加法。

注意，= 是赋值运算符，而 == 是相等性检测。

除了常规的算术和逻辑运算符，C++ 还提供了更特殊的修改变量的运算：

```
x+=y    // x = x+y
++x     // 递增：x = x+1
x-=y    // x = x-y
--x     // 递减：x = x-1
x*=y    // 缩放：x = x*y
x/=y    // 缩放：x = x/y
x%=y    // x = x%y
```

这些运算符简洁、方便，因此使用非常频繁。

表达式的求值顺序是从左至右的，赋值操作除外，它是从右至左求值的。不幸的是，函
7 数实参的求值顺序是未指定的。

1.4.2 初始化

在使用对象之前，必须给它赋予一个值。C++ 提供了多种表达初始化的符号，如前面用到的 =，以及一种更通用的形式——花括号限界的初始值列表：

```
double d1 = 2.3;                    //将 d1 初始化为 2.3
double d2 {2.3};                    //将 d2 初始化为 2.3
double d3 = {2.3};                  //将 d3 初始化为 2.3（使用 { ... } 初始化，= 是可选的）
complex<double> z = 1;              //标量为双精度浮点数的复数
complex<double> z2 {d1,d2};
complex<double> z3 = {d1,d2};       //使用 { ... } 初始化，= 是可选的

vector<int> v {1,2,3,4,5,6};        //整数向量
```

= 初始化是一种比较传统的形式，可追溯到 C 语言，但如果你心存疑虑，那么还是使用通用的 {} 列表形式。抛开其他不谈，这至少可以令你避免在类型转换中丢失信息：

```
int i1 = 7.8;       //i1 变成了 7（惊讶吗？）
int i2 {7.8};       //错误：浮点数向整数的转换
```

不幸的是，丢失信息的类型转换，即*收缩转换*（narrowing conversion），如 `double` 转换为 `int` 及 `int` 转换为 `char`，在 C++ 中是允许的，而且是隐式应用的。隐式收缩转换带来的问题是为了与 C 语言兼容而付出的代价（参见 16.3 节）。

我们不可以漏掉常量（参见 1.6 节）初始化，变量也只有在极其罕见的情况下可以不初始化。也就是说，在引入一个名字时，你应该已经为它准备好了一个合适的值。用户自定义类型（如 `string`、`vector`、`Matrix`、`Motor_controller` 和 `Orc_warrior`）可以定义为隐式初始化方式（参见 4.2.1 节）。

在定义一个变量时，如果它的类型可以由初始值推断得到，则你无须显式指定：

```
auto b = true;      //一个 bool
auto ch = 'x';      //一个 char
auto i = 123;       //一个 int
auto d = 1.2;       //一个 double
auto z = sqrt(y);   //z 的类型是 sqrt(y) 的返回类型
auto bb {true};     //bb 是一个 bool
```

当使用 `auto` 时，我们倾向于使用 = 初始化，因为其中不会涉及带来潜在麻烦的类型转换，但如果你喜欢始终使用 {} 初始化，也是可以的。

当没有特殊理由需要显式指定数据类型时，一般使用 `auto`。在这里，“特殊理由”包括：

- 该定义位于一个较大的作用域中，我们希望代码的读者清楚地看到数据类型；
- 我们希望明确一个变量的范围和精度（比如希望使用 `double` 而非 `float`）。

使用 `auto` 可以帮助我们避免冗余的代码，并且无须再书写长类型名。这一点在泛型编程中尤为重要，因为在泛型编程中程序员可能很难知道一个对象的确切类型，类型的名字也可能相当长（参见 12.2 节）。

8

1.5 作用域和生命周期

声明语句将一个名字引入到一个作用域中：

- *局部作用域*（local scope）：声明在函数（参见 1.3 节）或者 lambda（参见 6.3.2 节）内的名字称为*局部名字*（local name）。局部名字的作用域从声明它的地方开始，到声明语句所在的块的末尾为止。块（block）用花括号 {} 限定边界。函数参数的名字也属于局部名字。
- *类作用域*（class scope）：如果一个名字定义在一个类（参见 2.2 节、2.3 节和第 4 章）