清华松岗系列丛书

C++程序设计实用教程

张国峰　编著

清华大学出版社

C++程序设计实用教程

张国峰　编著

清华大学出版社

(京)新登字 158 号

C++程序设计实用教程

张国峰 编著

本书是关于C++语言程序设计的一本实用教程，介绍 ANSI/ISO C++。本书内容详实，系统性强。

全书共分 17 章。第一章介绍C++语言的基本概念；第二至六章介绍了C++的基本结构；第七至九章介绍C++构造复杂数据类型的机制；第十章介绍串；第十一至十三章介绍C++的高级结构，即继承和模板；第十四章介绍C++结构化的异常处理；第十五和十六章介绍枚举类型、联合类型和位操作。第十七章介绍C++标准流类库。

本书注重逻辑性和通用性，示例典型，每章附有小结和习题。本书对学习和使用C++语言的各层次人员均适用，尤其是大学生和软件开发人员。

图书在版编目(CIP)数据

C++程序设计实用教程/张国峰编著. —北京：清华大学出版社，1995.11
ISBN 7-302-01974-6

Ⅰ.C… Ⅱ.张… Ⅲ.C 语言-程序设计-教材 Ⅳ.TP312C

中国版本图书馆 CIP 数据核字（95）第 20360 号

出 版 者：清华大学出版社（北京 清华大学校内，邮政编码：100084）
责任编辑：张孟青
责任校对：李凤茹
印 刷 者：北京市海淀区清华园印刷厂
发 行 者：新华书店总店北京科技发行所
开　　本：787×1092 1/16 印张：30.5 字数：720 千字
版　　次：1996 年 1 月第 1 版 1996 年 1 月第 1 次印刷
书　　号：ISBN 7-302-01974-6/TP・911
印　　数：00001—10000
定　　价：38.00 元

前　言

C++语言是AT&T贝尔实验室的Bjarne Stroustrup博士于80年代早期开始开发的一种通用的程序设计语言，在1983年被命名为C++语言。C++语言是通过将Simula 67、Algol 68和Ada等语言中最好的特点有机地溶合到C语言中而不断完善和发展起来的。与C语言一样，C++语言也非常注重强调代码的有效性和紧凑性；与C语言不同的是，C++语言更注重强调对高级抽象的支持，所以，C++语言特别适合于开发中等和大规模的复杂程序。目前，几乎在所有的计算机应用领域，都在应用C++语言，许多大学和培训部门都在纷纷开设C++语言课程。

由于C++语言发展历史的原因，即C++语言以C语言为基础，所以，目前存在一种观点，即学习C++语言应先学习C语言，但奇怪的是，绝对没有这种观点存在，即学习C++语言应先学习Simula 67或Ada语言，虽然C++语言的精华主要来自于这两种语言。可见，持上述观点的人只注意到了C++语言与C语言语法的相近，而忽视了C++语言与C语言在构造程序的方法上存在着本质的区别。经验表明，先学C语言再学C++语言，将花十倍之精力，而未必能得一成之功夫。古人早有所云："学拳容易改拳难"。

C++语言保持了与C语言的兼容，这一方面是为了照顾广大的C程序员，C程序员只需学习C++语言新增的语言成分；另一方面也是为了保护在C语言上所做的大量投资。但这并不能支持学习C++语言应先学习C语言这一观点。作为一门语言，C++语言有自己的抽象和结构特征，虽然其中一些来自于C语言，但它们在C++语言中的地位却发生了变化(其中一些被降到次要的地位)。C++语言更强调对抽象的支持，它允许程序员为语言定义新的抽象，这些抽象使计算机解决问题的方式更加类似于人类的活动。

本书在对程序设计语言的基本概念进行讨论的基础上，围绕着C++的抽象和结构特征来介绍语言，关注于抽象在高级语言中的作用和C++实现这些抽象的结构。本书对C++进行正面介绍，由于C++程序设计的特殊性(例如，与C的兼容，使用习惯，阅读那些非C++风格的C++程序——主要来自于转向C的C++程序员等)而读者需要知道的来自于C的成分点到为止，目的在于使读者对C++有正确而全面的认识和理解。本书面向这样的读者，他们想成为真正的C++程序员，但仅要求能读懂C程序。

全书共分十七章和三个附录。第一章介绍程序和程序设计语言的相关概念，给出了几个简短的C++程序。第二章到第六章介绍C++的基本结构。第七章到第九章介绍C++构造复杂数据类型的机制。第十章介绍串。第十一章到第十三章介绍C++语言的高级结构，即继承和模板。第十四章介绍C++结构化的异常处理机制。第十五章和第十六章介绍枚举类型、联合类型和位操作。第十七章介绍C++标准流类库。附录A给出了ASCII表。附录B给出一些常用的数学库函数的使用说明，其中的一些函数在本书正文中用到而没有进行说明。附录C则对本书正文中没有讨论到的C++成分进行一些说明，并给出了我们对这些内容的看法。

本书向读者介绍ANSI/ISO C++，不要求读者有C的知识。书中的大部分示例程序都很典型，示例程序使用Borland C++系统在MS-DOS和Microsoft Windows上进行了测试。

相对于其它程序设计语言的学习来说，学习C++是需要多花一些工夫的。学习一门程序设计语言的要诀是：多在计算机上实践。将书中的程序亲自录入到计算机中，编译并运行，分析程序的运行结果。如果程序出现编译错误或运行错误，这都是好事，在纠正错误的过程中，读者将加强对C++语言的语法的记忆，或加深对C++语言相关成分的语义的理解。这也能使读者逐渐积累调试程序的经验。在这个过程中所花费的时间会在将来的程序开发项目中获得补偿。所以，对一些计算机基础比较低的读者，不能仅停留在看懂书中的示例程序或仅做一些书中的练习这一简单要求上。

在本书写作过程中，面对一些问题，作者与麦中凡教授和贾素玲副教授进行了有益的商讨。纪涌调试了本书大部分的程序。李大伟代作者整理了本书书稿。陈捷、郝威和杨晖等在本书写作过程中都给予了作者很大帮助。在此，作者向他们表示深深的感谢。作者也衷心感谢中国UNIX用户协会培训中心徐国平主任和中国计算机学会培训部王素文老师所给予的热情帮助。对本书所存在的错误和不足之处，希望广大读者批评指正。

张 国 峰

1995年6月于北京航空航天大学

目　录

第一章　程序设计语言的基本概念

高级程序设计语言的特点是什么，设计高级程序设计语言的目的是什么，这是在学习一门语言时必须回答的问题。在本章，我们将讨论程序和程序设计语言等基本概念，简要地讨论了C++的语言抽象和结构，目的是使读者能对C++语言有个总体性的认识，便于对后续章节的学习。最后，本章介绍了C++的词法元素，并给出了简单的C++程序来说明在C++程序中进行输入和输出操作的方法。

1.1　程序

读者若翻阅一下本书中给出的C++程序，那么，也许对程序这个概念就有了一个直观的理解。当有人问及程序是什么时，回答可能是程序是一种文档，这个文档给出了一系列将被计算机执行的指令。这个回答基本上是正确的，但它只说明了程序的一个方面的特性，特别是，这种回答并没有回答出程序的本质特征是什么。

那么，究竟什么是程序呢？这只需要对未通电的计算机和通电的计算机作一番考察，就可以得出一个大致的结论。对于一个未通电的计算机，它不会做任何事情，这点对任何需要使用电力驱动其工作的机器来说都是一样的。但计算机却有它更特别的地方：当它未通电时，我们甚至不能说出来它能够干什么用，或说它有什么功能可供我们利用，这是与电视机等我们常见的机器不同的地方。当我们给计算机通上电，使其工作时，我们可能得到的是一个能进行文字处理的机器，我们可以利用这台机器录入文章，按出版要求进行排版和打印；而在另一台同样的结构和配置的计算机上，我们却可以绘制工程图，或进行科学计算，或进行工资计算及人员管理等。

所以，计算机这个装置本身不是一台真正的机器，因为没有合适的软件或程序，它就不能做任何有用的事情。从这个观点来看，程序是一部机器，特别是，它是一种软件机器。程序正文代表被开动之前的机器，正执行的程序表示正在进行动作的通电的机器。借助于计算机终端，我们使用一种程序设计语言建造这种机器。我们所建造的机器依赖于工具、材料和想象力之间的相互作用，当建造软件机器时，程序设计语言定义了工具和材料。因此，也可以说，运用一定工具和材料建造各种功能的机器——即了解、掌握C++程序设计语言来设计出各种效用的程序——是我们的学习目的。

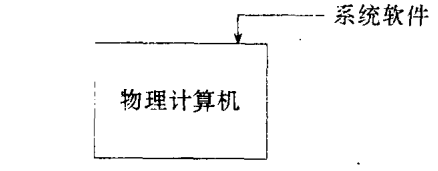
由此看来，程序是一个包含有被计算机执行的指令的文档这种说法只认识到了程序的被动方面。程序正文是被动的，但这种被动的文档可以被改变为主动的物理过程：当程序被执行时，文档中的指令被执行，正执行的程序是实时中的一个事件。而物理计算机是软件机器的动力源。虽然程序必须被装入到计算机中才能被执行，但软件机器的特征直接由语言和程序员决定，只间接地被物理计算机决定。

1.2 虚拟计算机

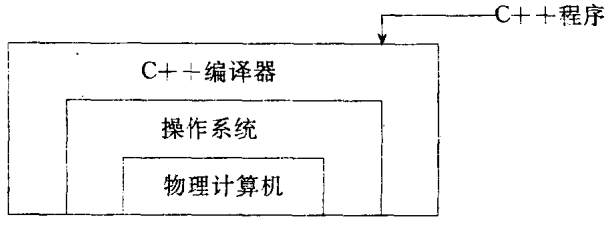
上节的讨论可能很抽象，为此，在本节，我们将探讨高级程序设计语言（简称高级语言）与它们的编译器和机器硬件（又被称为裸机）之间的关系，以便使读者能更好地理解上面的概念。

任何类型的软件为计算机用户提供了一级机器抽象，软件建立了一个新的机器，它在一个特定的应用领域的框架内比其较低层更有用。例如，操作系统（如 MS-DOS 或 Microsoft Windows）在硬件之上提供了一级机器抽象，而在操作系统之上运行的应用软件（如字处理软件）则提供了更高级的机器抽象。每种机器都使用不同的语言或命令与用户进行通信。

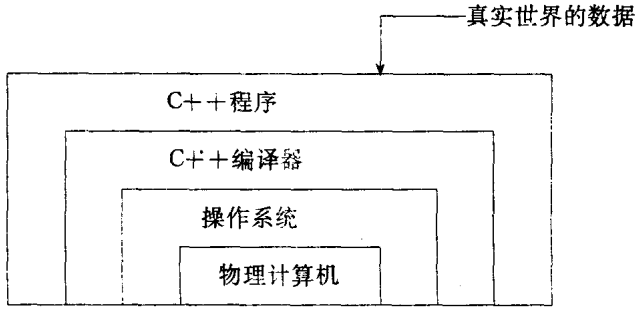
每个高级语言都有一个语言翻译程序，这个翻译程序读入高级语言的语句，然后产生等价的可执行的机器语言指令。关于高级语言的翻译程序，我们在 1.4 节进行讨论。现在我们要强调的是，高级语言和它的翻译程序将裸机变成了抽象的、高级的计算机，在使用这台计算机时，我们不必考虑裸机级的细节（例如内存地址、寄存器、程序指令计数器等），每级的软件提供了一个附加的抽象层，我们使用这个抽象层上的概念来讨论这台抽象软件机器，使用这个层上的实体构造更抽象的机器。图 1-1 给出了 C++虚拟计算机及其与实际的计算机的比较说明。



(a) 实际的计算机



(b) C++虚拟计算机



(c) 一个应用程序级的虚拟计算机

图 1-1 计算机抽象层次

这里，所谓的物理计算机本身也可以被视为一个抽象，因为，它与一些固化的软件(微程序)相关，这些固化的软件为物理计算机提供了一个机器语言接口，以便使程序员可以使用机器语言代码进行编程。

从图 1-1 可以看出，使用高级语言编写的程序可以被看作一台抽象的计算机，它是一台由应用程序的程序设计人员定义的虚拟的计算机，这样，一个正执行的程序可以被认为将裸机变成了一台某种有用的计算机，例如，工资管理计算机。

在实际计算机之上所建立的这个虚拟计算机层次结构提高了计算机的级别，使计算机能与高级人类用户"和谐相处"：用户越来越不需要学习像计算机那样去思考问题，而是尽可能地按照人类所习惯的方式思考问题和使用计算机，这使计算机技术能为广大的用户所使用。

从图 1-1 中，我们也可以看出各层抽象之间的区别和联系。在一个抽象层上的所谓的程序被其下一层次上的程序作为数据进行处理，例如，实际的硬件与它的机器语言接口一起对操作系统进行操作，而操作系统对高级语言处理器进行操作，高级语言处理器则操作高级语言语句。

现在，我们认识到了程序是一个软件机器，那么，为设计这样的机器，即进行程序设计，我们还必须对程序的内部结构进行分析。

可以从两种观点上来考虑一个程序的结构。第一种观点认为，程序是我们想让计算机执行的一系列动作的描述，这些动作是物理计算机的基本操作，或是抽象计算机的基本操作。基本操作包括将数据从寄存器中拷贝到某个存储位置处，或将某个存储器位置中的数据拷贝到另一个存储位置处，对寄存器中的数据执行某个操作，或激活某个输入或输出设备。另一种观点认为，程序是真实世界某一方面的一个模型，模型化的外部世界由对象(例如，设备、雇员和帐单等)组成，程序员的任务就是建立程序中的实体(程序对象)与真实世界中的对象之间的对应关系，以及程序中的函数与真实世界的过程的对应关系，执行一个函数表示真实世界某个对象的状态的变化。

这两种观点导致关于程序设计语言应提供什么特性的完全不同的结论。支持第一种观点的语言具备对计算机各个部分进行访问的能力，这样，为求解某个问题，程序员必须告诉计算机一步步怎样动作，这就是说，当进行问题求解时，我们必须首先学会像计算机那样思考问题。支持第二种观点的语言必须能够处理高级抽象，并提供手段来构造表达真实世界中的对象和过程的模型。显然，后者的观点免除了对基础计算机硬件细节的考虑，允许我们在一个较高的抽象层次上描述对象和过程，而选择某种特定的方法去执行特定的动作等细节的考虑则留给了编译器的设计者或编译器。

我们以构造房屋的过程为例来说明这两种观点的不同。第一种观点类似于房屋的建造者观察房屋的观点。建造者关心为完成最终的房屋所需要的方法，这种方法应该是有效的，所建立的房子应该是结构上合理的，建造者所使用的语言应包含所使用的材料和建造方法等词汇。而第二种观点类似于建筑师的观点，他关心的是房屋的整体功能和式样，它应该满足建筑师的要求，并满足用户的需要。建筑师的语言处理抽象的概念，例如空间、样式和功能等，并处理具体的单元，例如墙和窗户等。

建造者的语言的优点是能够构造高效的软件，但正如建造者在建造屋子时必须关心诸

如木材的大小、钉子的尺寸等众多细节一样，程序的建造者也必须考虑象存储分配、字节对齐、程序执行的顺序等细节。

1.3 抽象和结构

上面两节说明了这样一个问题，即任何软件为计算机用户提供了一个机器抽象，程序设计语言和它的编译器是抽象的、高级的，甚至是程序员友好的计算机，程序员使用它们，而无需关心机器级的细节。程序设计语言的发展就是沿着不断地从机器抽象出系统和语言，最终的目的是使这种抽象能达到用户的层次，而我们所学习到的为计算机建立软件的任何东西都是为建造一个特殊目的的抽象的计算机器的建造块(部件或组件)。因此，认识和理解一个程序设计语言所提供的抽象对语言的学习将是很有帮助的。

抽象是这样一种工具，借助这种工具，我们只考虑与手边的问题有关的信息，而忽略那些无关紧要的细节或事实。计算机科学中的抽象是建立可能的实现模型，这些模型抑制细节，而保留本质特征。抽象揭示了事物的本质特征，这些特征也可被其它相似的事物共享。

在计算中，抽象是体现计算的实体。抽象将干什么和如何干区别开来，抽象的使用者或称用户关心一个抽象干什么，而抽象的实现者则定义怎么干，这使程序员之间有严格的分工，但往往是，一个程序员同时扮演着这两种角色。所以，在我们学习C++时，应时刻注意自己的角色变化，以便在学习过程中更具体地理解抽象的作用和意义。

程序设计语言中的抽象不同于数学系统中的抽象。在程序设计语言中，我们必须同时考虑抽象与问题求解的联系，以及抽象与虚拟机的联系，即要考虑怎样进行计算，而这个方面在数学抽象中是不考虑的。

一个通用的程序设计语言必须提供适用于可能应用的有关抽象，以便程序员能够使用这些抽象。C++语言中的抽象是过程抽象、数据抽象和控制抽象。过程抽象的用户关心一个过程干什么，而不是如何干，如何干的问题是抽象的实现者关心的。数据抽象通过一个严格定义的操作接口的方式而在数据结构上提供了一个抽象，这个操作接口隐藏了数据结构和有关操作的实现细节。程序设计语言中的数据抽象支持是基本数据类型和抽象数据类型。基本数据类型被建造在语言中，而抽象数据类型是用户定义的一种数据类型。控制抽象则用于定义动作的执行顺序，它是使我们能够在一个数据结构上进行移动，根据需要改变或维护各个值的一种机制。利用已有的这些抽象，程序员可以构造更高级的抽象。

抽象和结构是密不可分的，结构被视为对抽象的实现或是抽象的具体化。当事物很复杂时，我们使用结构来观察和组织复杂的事物。在程序设计语言中，对每种抽象，存在相应的结构。一个数据结构由数据元素之间的关系和适用于该结构的操作来定义。控制结构在语言中则表现为顺序、选择、迭代、过程调用和转移等。程序的结构是构成程序的词法单位，它包括单词、表达式、语句、函数和类等。程序员的工作就是理解这些结构，并使用这些结构来实现各种抽象，构造各种程序。

下面给出一个典型的C++程序，来认识C++程序中的各种抽象和结构形式，细节问题将在以后的章节中进行讨论，因此，读者不必急于搞清楚这个程序的细节，我们的目的是给读者有关C++程序结构的一个感性认识，而不是讲解某个具体的C++程序。

```
class Point {
  public:
    void Init( int x, int y ) ;
    int XCoord() ;
    int YCoord() ;
    void Move(int xOffset,int yOffset) ;
  private:
    int X, Y ;
} ;
```

以 class 开始的部分实现了点数据抽象，它的构成形式是以 class 开始，在后面的一个花括号内给出了这个抽象的操作声明和为实现这些操作所使用的数据结构。数据结构由：

```
int X, Y ;
```

来表示，private 表示这个结构是隐藏的，用户不应该使用它们。而操作则由：

```
void Init( int x, int y ) ;
int XCoord() ;
int YCoord() ;
void Move(int xOffset,int yOffset) ;
```

来表示。象 Init、XCoord、YCoord 和 Move 等给出了操作的名字，这个抽象的使用者只使用这些操作名来操作一个点对象，但并不关心这些操作是怎样实现的。这种表达形式被称为过程抽象规范说明，它只给出了一个过程应该被怎样使用或能干什么的信息，但没有表示出过程的实现细节(怎样干)。

一个用户可以使用这个抽象来构造其它的抽象。例如，下面的例子使用 Point 抽象实现了一个过程抽象：

```
int Checks()
{
    Point P ;
    P. Init(2, 3 ) ;
    if ( P. XCoord() == P. YCoord() )
        cout << "Angle is 45 degree" ;
    else
        cout << "Angle is not 45 degree" ;
    return 0 ;
}
```

过程抽象的实现结构是先给出过程的名字，例如 Checks，然后在其后的一个花括号内给出这个抽象过程的一步步实现方法，每步被称为一条语句，即上例中以分号结束的句子。

像上面例子中的“if else”、“P. Init(2, 3)”、“==”、“<<”等构造都是表达某种控制抽象，而像“P”、“P. Init(2,3)”、“P. XCoord()==P. YCoord()”等都是表达式，上述程序按

C++语言的语法要求，将表达式、语句等成分有机地组织起来，构造成一个程序。

1.4 程序设计语言的实现

程序设计语言的实现是对使用高级语言所要求的符号所表达的高级程序进行解释的一种机制。高级语言的实现可以以解释方式或是以编译方式实现。对解释方式实现的高级语言，由一个称为解释器的程序读入高级语言程序的语句，然后由该解释器对语句进行执行。例如，BASIC 语言就是一种以解释方式实现的语言。由于解释方式实现的语言的程序没有进一步的翻译过程，所以，使用简单，并有一定程度的灵活性，但以这种方式实现的语言的程序运行效率很差，因此，一些被广泛使用的程序设计语言基本上都是以编译方式实现的。

对编译方式实现的高级语言，由一个称为编译器的程序读入高级语言的语句，作为其输出结果的是某种目标语言程序，所以，编译器也被称为翻译器。目标语言一般都是特定计算机的机器代码，这种代码可以被计算机硬件直接执行，因此，使用这种方式实现的高级语言的程序的运行效率很好。编译器不仅依赖于特定的计算机，也依赖于特定的操作系统，同时，它所生成的代码也依赖于该操作系统。因此，当我们开发程序时，必须根据程序的运行环境，选择相应的编译器。C++语言是以编译方式实现的高级语言。

1.4.1 编译过程

编译器的工作包括将使用高级语言书写的程序（被称为源程序或源代码）分解为各种语言成分，检查程序对这些语言成分的使用是否正确，然后，生成中间代码，最后生成机器可执行的目标程序。图 1-2 给出了编译过程的各个阶段的示意说明。

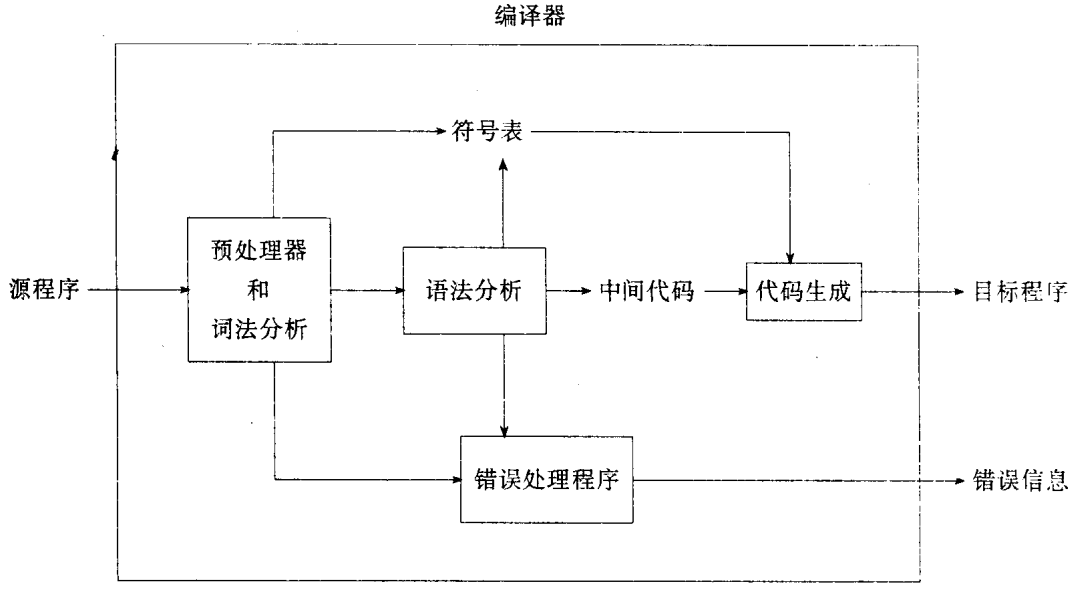


图 1-2 编译过程

词法分析和语法分析，以及生成符号表和中间代码这一过程被称为源程序分析。这个阶