21世纪高等学校计算机规划教材

21st Century University Planned Textbooks of Computer Science

标准C++程序设计

Standard C++ Programming

牛连强 编著

- 最新的标准与先进的体系相结合
- 精练的内容与深入的讲解相结合
- 深邃的思想与实用的经验相结合

高校系列

人民邮电出版社
POSTS & TELECOM PRESS

21世纪高等学校计算机规划教材

21st Century University Planned Textbooks of Computer Science

标准C++程序设计

Standard C++ Programming

牛连强 编著

高校系列

人民邮电出版社

北京

图书在版编目（CIP）数据

标准C++程序设计 / 牛连强编著. —北京：人民邮电出版社，2008.8
21世纪高等学校计算机规划教材
ISBN 978-7-115-18028-5

Ⅰ. 标… Ⅱ. 牛… Ⅲ. C语言—程序设计—高等学校—教材 Ⅳ. TP312

中国版本图书馆CIP数据核字（2008）第076302号

内 容 提 要

本书是作者总结教学实践经验的产物。全书主要以过程化和面向对象两条主线进行讲解，系统地介绍了C++语言的预备知识，以及两条主线内容在思想上的差异；重点介绍了C++语言的语法、面向对象的基本特征、C++异常处理机制等。全书采用C++最新的标准，内容讲述深入浅出，示例讲解精炼，并配有提示。

本书可作为大专院校程序设计课程的教材，也可作为软件开发技术人员的参考书。

21世纪高等学校计算机规划教材

标准C++程序设计

◆ 编　　著　牛连强
责任编辑　武恩玉

◆ 人民邮电出版社出版发行　　北京市崇文区夕照寺街14号
邮编　100061　　电子函件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
北京通州大中印刷厂印刷

◆ 开本：787×1092　1/16
印张：16
字数：420千字　　2008年8月第1版
印数：1－3 000册　　2008年8月北京第1次印刷

ISBN 978-7-115-18028-5/TP

定价：25.00元

读者服务热线：(010)67170985　印装质量热线：(010)67129223
反盗版热线：(010)67171154

前 言

C++语言的实用性及魅力是毋庸置疑的。时至今日，介绍C++编程的图书可谓数量众多，包括国内外学者撰写的各种设计类、教材类书籍，其中不乏学习和研究C++语言程序设计的经典，如《C++Primer中文版》、《C++语言程序设计（特别版）》等。不过，因为C++语言本身的内容很多，所涉及的部分概念也较为复杂，致使图书的篇幅激增。各种教材动辄四五百页，令人望而生畏。事实上，各类学校开设C++课程的时间、先期的课程安排、教学大纲的要求等都不尽相同，而多数学习者也不是以成为C++语言的专家为目标，注重的是掌握必备的知识，能更好地应用其从事软件开发工作。此外，C++在较长时间的改进中，形成了不同的版本，教学中也需要适时将新标准介绍给读者。

基于上述原因，我们重新组织编写了这本《标准C++程序设计》，力图以有限的篇幅透彻讲解标准C++语言的核心内容。

本书可作为第二门程序设计课程的教材，考虑到很多学校在开设C++课程之前已经先学习了C语言课程，因此，本书对C++中继承自C语言的过程化程序设计部分（包括流程控制）没有作过多讲解。

本书内容仍以过程化和面向对象两大部分构成。第1章介绍C++语言的预备知识，并用简单示例比较了过程化程序设计与面向对象程序设计在思维上的差异。第2～5章介绍C++语言的过程化语法，并对C++语言中的基本对象作了渗透。第6～12章介绍C++语言的面向对象特征，第13章简要说明了建立在类模板基础上的泛型编程技术。第14章介绍了C++的异常处理机制。标注*号的章节，授课教师可根据实际教学情况选讲。

每章开始以精炼的语言扼要说明本章的内容，难点被适当地分解在各章里。每章最后安排了若干条提示和思考题。这些提示可视为注意事项，对程序设计者积累经验、养成良好习惯具有很好的警示作用。思考题有助于理解语言的语法现象，值得认真对照教材去分析，乃至构造适当的例子去实验。每章结束部分配备了相当数量的习题。

书中所有的完整程序代码都由作者在C++Builder 6.0环境下调试通过。之所以选择C++Builder 6.0而不是Microsoft Visual C++，是因为后者含有较多非标准性的成分，对标准C++和标准类编程的支持都不理想，甚至在一些局部问题（如函数中产生的临时对象等）上采用的处理方法明显地缺乏合理性。不过，对于利用控制台程序进行语法学习的读者来说，使用哪一种环境差别并不大，本书对一些Visual C++ 6.0明显不支持的场合作了简要说明，而书后对这两种环境的使用方法都给出了介绍。

对于学习者，掌握程序设计技术最好的途径就是上机实验。为此，每章末都包含了大量的实践性题目，它们都有不同程度的应用背景，而不是没有实际应用价值的“益智问题”。

另外，本书注重程序设计的理念和实际编程技术，目标是努力提高学生的编程能力，摆脱计算机专业毕业生不会编程的现象。

作为教师，学生提出的问题、自己的编程实践和对技术的思索、特色教学的需求都是促使我们编写这本教材的源动力。参加本书编写工作的还有任义、冯海文、付博文、杜梅，在此还要感谢计算机科学与技术专业 04 级的同学，他们阅读了本书的初稿，尤其是马玉飞同学，十分认真地指出了一些文字上的错误，包括部分修改意见，我们十分感谢他的热情和劳动。

应该说明，这是一本可以用作 48～64 学时教学的教材，我们努力从程序员的角度来介绍标准 C++语言的基本技术和精华内容，但限于篇幅，有些内容无法详述，如需更深、更详细地研究时可参考前文提及的那些经典著作和参考文献中所列的书目，研读这些著作会带给人更深邃的思想。

本书为授课教师提供电子课件等较为全面的学习辅助资源，有需要者请登录人民邮电出版社（www.ptpress.com.cn）免费下载。

本书是作者多年教学实践的产物，我们希望它能够引导读者步入 C++的辉煌殿堂，也特别希望读者能够不吝指出书中的缺点和错误，与我们交流，以便将其修改得更完善。

作者的电子邮箱：niulq@sut.edu.cn。

作　者

目 录

第 1 章 C++语言基础 …… 1

1.1 C++语言概述 …… 1
1.1.1 标准 C++ …… 1
1.1.2 C++语言的简单程序 …… 2
1.2 面向对象的程序设计思想 …… 4
1.2.1 过程化的程序设计 …… 4
1.2.2 面向对象的程序设计 …… 6
1.3 面向对象程序设计的主要特点 …… 7
1.4 程序的编辑、编译与运行 …… 8
提示 …… 9
思考 …… 10
上机实验 …… 10

第 2 章 基本数据与操作 …… 11

2.1 标识符与关键字 …… 11
2.1.1 标识符 …… 11
2.1.2 关键字 …… 12
2.2 数据与数据类型 …… 12
2.2.1 程序中的数据 …… 12
2.2.2 基本数据类型 …… 13
2.2.3 整型常量 …… 14
2.2.4 浮点型常量 …… 14
2.2.5 字符型常量与字符串常量 …… 14
2.2.6 符号常量 …… 16
2.2.7 变量 …… 17
2.3 简单运算 …… 18
2.3.1 运算符和表达式 …… 18
2.3.2 算术运算 …… 18
2.3.3 赋值类运算 …… 19
2.3.4 自加和自减运算 …… 20
2.3.5 关系运算和逻辑运算 …… 20
2.3.6 位运算 …… 21
2.3.7 sizeof 运算与逗号运算 …… 23
2.4 混合运算与类型转换 …… 24
2.4.1 隐式类型转换 …… 24
2.4.2 显式类型转换 …… 24
2.5 数据输入与输出 …… 25
提示 …… 26
思考 …… 27
练习 …… 27
上机实验 …… 28

第 3 章 语句与流程控制 …… 29

3.1 语句 …… 29
3.2 选择结构 …… 30
3.2.1 条件运算符和条件表达式 …… 30
3.2.2 if 语句 …… 30
3.2.3 switch 语句 …… 32
3.3 循环结构 …… 34
3.3.1 while 语句 …… 34
3.3.2 do-while 语句 …… 36
3.3.3 for 语句 …… 37
3.3.4 流程转向语句 …… 39
提示 …… 41
思考 …… 41
练习 …… 41
上机实验 …… 43

第 4 章 指针、数组与引用 …… 44

4.1 指针 …… 44
4.1.1 指针的概念 …… 44
4.1.2 指针运算 …… 46
4.1.3 指针的安全性问题 …… 47
4.2 数组 …… 50
4.2.1 数组的定义与访问 …… 50
4.2.2 数组的指针访问 …… 51
4.3 引用 …… 53
4.4 与指针相关的其他类型 …… 54
4.4.1 指向指针的指针 …… 54

4.4.2 指针数组 …… 55
4.4.3 数组名与指向数组的指针变量 …… 56
4.5 动态内存分配 …… 57
4.5.1 分配单个对象 …… 57
4.5.2 分配一个数组 …… 58
4.5.3 malloc 与 free …… 59
提示 …… 59
思考 …… 60
练习 …… 60
上机实验 …… 61

第 5 章 函数与函数模板 …… 63

5.1 函数的定义与声明 …… 63
5.1.1 函数定义 …… 63
5.1.2 函数声明 …… 64
5.2 函数调用与参数匹配 …… 65
5.2.1 调用关系 …… 65
5.2.2 函数返回控制与函数调用表达式 … 66
5.3 参数的传递方式 …… 68
5.3.1 普通传值方式 …… 69
5.3.2 传递指针方式 …… 69
5.3.3 传递引用方式 …… 72
5.4 特殊的函数返回值 …… 73
5.4.1 返回指针的函数 …… 73
5.4.2 返回引用的函数 …… 75
5.5 递归函数 …… 77
5.6 内联函数 …… 79
5.7 函数重载 …… 79
5.8 指向函数的指针 …… 81
5.9 函数模板 …… 85
5.9.1 函数模板定义 …… 85
5.9.2 模板函数调用 …… 86
5.9.3 模板函数重载 …… 87
5.10 变量的存储属性 …… 88
5.10.1 外部变量 …… 88
5.10.2 静态变量 …… 89
5.10.3 自动变量 …… 90
提示 …… 90
思考 …… 91
练习 …… 91
上机实验 …… 93

第 6 章 类与对象 …… 95

6.1 类的定义 …… 95
6.1.1 类的含义与表述 …… 95
6.1.2 类定义的语法规则 …… 96
6.2 类对象 …… 98
6.2.1 对象定义 …… 98
6.2.2 成员访问 …… 98
6.2.3 对象存储 …… 100
6.3 类的方法 …… 101
6.3.1 为类提供必要的方法 …… 101
6.3.2 inline 方法 …… 101
6.3.3 方法重载与缺省参数 …… 102
6.3.4 常成员函数 …… 102
6.3.5 this 指针 …… 103
6.3.6 类的模板函数方法 …… 104
6.4 构造函数与对象初始化 …… 104
6.4.1 初始化的难题 …… 104
6.4.2 构造函数 …… 105
6.4.3 无名对象与临时对象 …… 107
6.4.4 对象数组与动态对象 …… 108
6.4.5 成员初始化列表与特殊成员的初始化 …… 109
6.5 拷贝构造函数 …… 111
6.5.1 用已有类对象构建新的类对象 …… 111
6.5.2 改变缺省的复制行为 …… 112
6.5.3 拷贝构造函数的实现 …… 113
6.6 析构函数与对象拆除 …… 114
*6.7 设计一个栈类 …… 114
6.8 字符串类 string …… 116
6.9 其他类型构造技术 …… 118
提示 …… 120
思考 …… 120
练习 …… 120
上机实验 …… 122

第 7 章 组织程序结构的相关技术 … 123

7.1 宏定义与条件编译 …… 123

7.1.1 宏定义……123
7.1.2 条件编译……124
7.2 头文件包含……124
7.2.1 头文件包含指令……124
7.2.2 新旧库头文件……125
7.2.3 类定义与实现的分离……125
7.2.4 头文件中的内容……125
7.2.5 一个头文件示例……126
7.3 对象的构造与析构次序……126
7.3.1 对象构造的时机和次序……126
7.3.2 对象拆除的时机和次序……127
7.4 名字冲突与名字空间……127
7.4.1 名字冲突及对策……127
7.4.2 定义和使用名字空间……129
*7.5 extern 声明及 C++与 C 的混合编程……130
7.5.1 extern 声明的作用……130
7.5.2 用 extern “C” 修饰 C 的代码……131
提示……132
思考……132
练习……133
上机实验……133
第 8 章 静态成员、友元与成员指针……134
8.1 静态成员……134
8.1.1 静态属性……134
8.1.2 静态方法……136
8.2 友元……136
*8.3 指向类成员的指针……138
8.3.1 利用普通指针访问静态成员和非静态数据成员……138
8.3.2 指向非静态方法的指针……139
提示……140
思考……141
练习……141
上机实验……142
第 9 章 继承……143
9.1 继承的概念与表示……143
9.1.1 继承与派生……143
9.1.2 继承关系的描述……144
9.2 继承的实现……145
9.2.1 继承的语法形式……145
9.2.2 对基类成员的访问……146
9.3 派生类对象的构造与析构……147
9.3.1 继承与聚集……147
9.3.2 复杂类型的构造与拆除……148
9.4 继承的工作方式……150
9.4.1 派生类对基类的覆盖……150
9.4.2 利用指针和引用的访问……151
提示……152
思考……152
练习……153
上机实验……154
第 10 章 虚函数与多态性……155
10.1 共同的基类……155
10.1.1 概念中的共性……155
10.1.2 公共基类……156
10.2 虚函数……157
10.2.1 多态性、静态联编与动态联编……157
10.2.2 用虚函数实现动态联编……158
10.2.3 构造、析构与虚函数……161
*10.2.4 虚函数的内部实现机制……161
10.2.5 重载、覆盖和隐藏……163
10.2.6 动态造型（dynamic_cast）……164
10.3 纯虚函数与抽象类……165
10.3.1 纯虚函数……165
10.3.2 抽象类……165
*10.4 多重继承……166
10.4.1 多重继承的语法……166
10.4.2 多重继承中的二义性……167
10.4.3 虚继承……168
提示……171
思考……171
练习……171
上机实验……172

第 11 章　运算符重载 ……173

11.1　重载运算符的概念 ……173
11.1.1　重载运算符的函数特征 ……173
11.1.2　类运算符重载的两种方法 ……174
11.1.3　重载运算符的限制 ……176
11.2　重载运算符函数的设计 ……177
11.3　若干常见运算符的重载 ……178
11.3.1　重载增量运算符++ ……178
11.3.2　重载赋值运算符= ……179
11.3.3　重载下标运算符[] ……181
11.3.4　重载类型转换运算符 ……182
*11.3.5　重载函数调用运算符与函数对象 ……182
提示 ……183
思考 ……184
上机实验 ……184

第 12 章　流与文件操作 ……185

12.1　理解流机制 ……185
12.1.1　流与文件 ……185
12.1.2　从函数到对象 ……186
12.1.3　源、汇和 iostream 流控制类 ……186
*12.2　构造可流的类 ……188
12.2.1　进一步探讨 cout 和 cin 对象 ……188
12.2.2　重载输出运算符<< ……189
12.2.3　重载输入运算符>> ……189
12.3　格式控制 ……190
12.3.1　使用流的方法 ……190
12.3.2　使用操控符（Manipulators） ……195
*12.3.3　内存格式化（字符串流） ……198
12.4　文件流 ……199
12.4.1　文件打开与关闭 ……199
12.4.2　文件的读写操作 ……201
12.4.3　二进制文件 ……202
12.4.4　文件的随机访问 ……203
提示 ……204
思考 ……204
上机实验 ……205

第 13 章　类模板、容器与泛型算法 ……206

13.1　类模板 ……206
13.1.1　类模板的定义与使用 ……206
*13.1.2　设计一个队列模板 ……209
*13.2　C++的标准模板库 ……211
13.2.1　从简单操作看 STL 和泛型编程 ……211
13.2.2　迭代器 ……214
13.2.3　几种主要容器类与类的方法 ……218
13.2.4　几种常用的通用算法 ……221
提示 ……223
思考 ……223
上机实验 ……223

*第 14 章　异常处理 ……224

14.1　异常处理方法 ……224
14.1.1　传统处理方法 ……224
14.1.2　try-catch 异常处理机制 ……225
14.2　用 try-catch 结构处理异常 ……226
14.3　异常处理中的其他问题 ……229
提示 ……230
思考 ……230
上机实验 ……231

附录 A　BCB 6.0 集成化环境的使用简介 ……232

附录 B　Visual C++ 6.0 编程环境简介 ……241

附录 C　常用字符与 ASCII 码对照表 ……246

附录 D　运算符的优先级与结合性 ……247

参考文献 ……248

第1章 C++语言基础

C++是从C语言基础发展起来的面向对象程序设计语言，但二者是相互独立的。本章简要说明C++语言的发展，用示例演示C++的简单程序和一般结构，并对结构化程序设计与面向对象程序设计的思想作了比较，以期使读者建立起面向对象程序设计的初步印象。最后，扼要介绍了高级语言程序的处理过程。

1.1 C++语言概述

C++是一种以C语言为基础开发的高级语言，一般认为C是C++的一个子集或基础语言，但二者又是完全独立的。如果说C++擅长大型程序的开发与设计，C则适合于更底层的项目开发。由于C++基本包含了C语言的所有主要特性，因此，了解C语言（或其他程序设计语言）对C++的学习会有较大帮助。反过来说，因为C++放弃了C语言中的某些难于理解的部分，如格式复杂的输入输出函数等，使其过程化的程序设计部分更容易掌握。不过，由C语言的传统的过程化程序设计方法过渡到新的面向对象程序设计方法总需要付出相当多的努力，而且C++的内容也远比C语言更庞杂。

考虑到目前多数的读者都有学习C语言的经历，而且C++本身所包含的内容、技术较多，本书的写作是以假设读者具有一些C语言编程知识为前提的，同时也有选择地忽略或淡化了某些不太常用或复杂的内容，以使其易于作为教材使用。

1.1.1 标准C++

C++的发明者是AT&T贝尔实验室的Bjarne Stroustrup博士，他给出了C++的第一个定义。首先，C++保留了C作为一个子集，使C++具有C语言的处理复杂的底层系统程序设计工作的能力。其次，为了增加面向对象特性，C++从Simula语言引入了类的概念，包括派生类和虚函数。此外，C++也借鉴了Algol语言的运算符重载等特性，形成了C++的早期版本，被称为“带类的C”。这期间，C++编译系统只是一个将C++代码翻译成C代码的预编译系统。

随后，C++的语法经过了若干次审查和修订，主要包括对重载的解析、连接以及存储管理，也在多重继承、static成员函数、const成员函数、protected成员、模板、异常处理、运行时类型识别和名字空间等方面进行了扩充，目的是使C++更容易编写和使用库。1988年诞生了第一个真正的C++编译系统。对于容器和泛型算法的支持，最初是以标准模板库STL形式提供，目前也已正式纳入C++标准。

随着C++应用的迅速增长，C++的标准化已不可避免。1989年，成立了旨在建立C++标准的

ANSI 委员会。到 1991 年，该委员会的 C++标准化工作正式成为 ISO 的 C++标准化工作的一部分。1998 年，ISO/ANSI C++标准正式通过并发布。

考虑到 C++标准的建立过程和市场上语言版本的应用状况，C++标准不得不保留对 C 及老版本 C++的兼容，这在外观上造成了 C++内容十分庞杂的感觉，也为使用不同时期的 C 和 C++功能带来了一些小的困扰，如头文件的处理等问题。本书将完全以 ISO/ANSI 的标准 C++为背景，但也会少量涉及一些早期语言中的问题。

总体上，C++是一种混合语言，或者说 C++是一种集过程化设计、面向对象、基于对象和泛型算法等多种技术于一体的编程语言。

在学习 C++语言时，最重要的是集中关注概念，不要迷失在语言的技术细节中。对于程序设计和设计技术的理解远比对细节的理解更重要，而这种理解的根本是时间和实践[2]。

1.1.2 C++语言的简单程序

这里给出了一个简单的 C++程序示例，功能是计算两个整数的和并将结果输出到屏幕上。

```
#include<iostream>                    //头文件包含        示例程序 1_1.cpp
using namespace std;                  //名字空间声明
//以下是 main 函数定义
int main( )                           //主函数定义
{ int x = 10, y = 20;                 //变量定义
  int z = x+y;                        //运算
  cout << "sum is " << z << endl;     //输出
  return 0;
}
```

程序运行时的输出结果为：

```
sum is 30
```

1. main 函数与程序结构

C++程序是由函数组装而成的，一个完整的程序中必须且只能有一个名字为 main 的函数，称作**主函数**，而这个面向过程的主函数来自于 C，也成了 C++是混合语言的象征。程序可以只由一个 main 函数构成，也可以由多个函数组装而成，但执行时总是从 main 函数开始。标准 C++要求 main 函数的声明类型为 int（而非 C 语言常见的 Void），只要在最末加上“return 0;”语句即可（也可不加）。

C++程序的外观结构与 C 极其相似，包括以下内容：

- 预处理指令部分；
- 函数声明部分；
- 类型定义部分；
- *main* 函数定义；
- 其他函数定义。

就一个函数（如 main 函数）来说，定义函数是指说明函数的格式和要执行的代码。这些代码放在一对花括号“{ }”内，称为**函数体**。函数体一般包括变量定义、输入、运算和输出等内容。

2. 头文件包含与名字空间

程序中的“#include <iostream>”是用于包含头文件的预处理指令，语句“using namespace std;”

说明使用名字空间 std。首先，包含 iostream 文件是因为使用了 cout 对象的原故，即 cout 定义于 iostream，没有头文件包含时，编译器不能识别 cout 的含义。其次，为了区别标准 C++、C 及老版本的 C++，也为了减少程序中不同区域定义的名字之间的冲突，C++引入了名字空间的概念。目前可以将一个名字空间理解成包含若干定义的一个区域的名字，而标准 C++的所有定义都属于名字空间 std。因此，需要向编译器表明，以下程序中出现的定义如果不是局部的，应在名字空间 std 中查找。换句话说，只要使用 C++标准库，就应该说明使用名字空间 std。在很长一段时间里，我们都使用这种固定的方法，即在程序开头加上如下代码，初学时不必深究其含义：

```
#include<iostream>                                    //头文件包含
using namespace std;                                 //名字空间声明
```

当然，也可以不在程序的开始处说明使用名字空间 std，而直接将 std 冠于一个名字之前，例如：

```
std::cout << "sum is " << z << std::endl;            //输出
```

此时，编译器也会认为 cout 和 endl 是属于名字空间 std 中定义的名字。这种作法可能会使人觉得麻烦。

3. 注释

程序注释是对程序中的代码、变量等所作的说明，在程序编译时对注释不作任何处理。程序中添加注释有助于提高程序的可读性，是非常必要的部分。在一些特殊程序中，注释可达整个篇幅的 1/3 以上。

C++中的注释分两种，包括来源于 C 的“/* 注释 */”（可称为注释对）形式和新的“//注释”形式。前者是段落形式的注释，可以包含连续的任意多行，可以在任何允许插入空格的地方插入。后者是行式注释，即从“//”开始到本行结束的内容均为注释。

通常，可以这样对程序进行注释：

（1）在程序头，添加有关程序名、功能、作者、目的、用途、修改史等的注释；

（2）在函数前，添加有关函数的功能、参数、返回值和副作用等的注释；

（3）在变量定义后，说明变量的含义；

（4）在函数体开头，解释算法思路；

（5）在类定义前或类定义中添加注释，说明类和成员的功能。

限于篇幅，本书通常只在行末添加少量、必要的注释，这些注释以中文形式给出，多用于解释所在行代码的功能、含义和应注意的事项。

4. 输入输出对象

程序中通常要从键盘接收用户的输入，并将运算结果输出到显示器上。与 C 语言的函数式输入输出技术不同，C++借助标准库中定义的对象来实现，目前可以认为这些对象是一种能力很强的变量。输入数据采用的对象是 cin，语法形式示例如下：

```
int x;
double y;
cin >> x >> y;                                       //也可写成: cin>>x; cin>>y;
```

上述语句可以将用户输入的数据保存到变量 x 和 y 中。

输出数据采用对象 cout 实现，语法形式示例如下：

```
int x = 10;
cout << "x is " << x << '.' << '\n';
```

此语句连续输出 4 部分的值，结果是“x is 10.”。示例程序 1_1.cpp 中输出语句末端的 endl 与这里的‘\n’作用基本相同，用于将光标转移到下一行的开头。

5. 编码习惯

C 和 C++的程序都以书写格式自由著称，这是指程序的书写几乎不受什么限制，程序员可以在一行上书写多个语句，也可以把一个语句写在多个行上。当然，这种灵活性可能使程序缺少一种可以遵循的规范。应该说，尽量保持好的书写风格是必须养成的习惯。因此，初学者应该注意程序的书写“格式”，如缩进格式和成对符号的对齐排列等。本书所提供的示例程序将尽量给读者提供一种可借鉴的规范，偶尔（如习题中）可能将相近的代码写在同一行内，这纯粹是出于篇幅的考虑。

1.2 面向对象的程序设计思想

从 C 向 C++过渡的实质是由过程化程序设计思想向面向对象程序设计思想的转变。本节以一个简单的程序来比较一下两种技术的差别，但这种比较仅是粗浅的、感官上的，读者并不需要完全了解描述过程中所涉及的所有技术细节。

1.2.1 过程化的程序设计

用计算机解决问题时总要设计程序，即以某种语言为工具编写控制计算机执行的动作序列，每个动作代表了一定的操作。随着人们看待问题和解决问题的思想观点的变化，程序设计方法也出现了很大差别。当然，程序设计方法的变革是以简化编程和提高软件生产率为目的，而不是以牺牲程序的正确性和效率为代价的。

由于编程问题起源于最初的科学计算，因此，长期以来，程序组织都是围绕算法的切分而展开的。面对一个问题，设计者首先要“建立需求分析和系统规格说明”，即“建立一系列规则，根据它判断任务什么时候完成，以及客户怎样才能满意”。随后，工作的重心就转移到如何把一个很大的项目分割成适当的小块，再由这些小块组织起来完成整个项目的工作，而程序员的职责就是考虑如何更好更快地实现这些小块。这样的处理方式被认为是基于过程的。在过程化程序设计中，数据和算法是问题的中心，程序设计的中心工作是研究数据的描述、存储和数据间的关系，以及作用在数据结构上的算法，这些算法通过一个一个的过程来体现，如图 1-1 所示。因此，N.Wirth 认为，**算法+数据结构＝程序**。

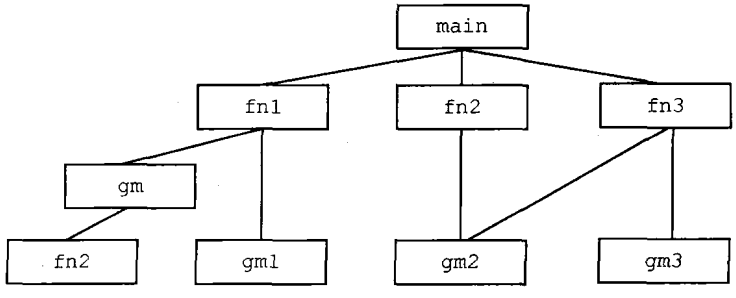

图 1-1 由函数组成的程序

过程化的问题处理思路形成了一套有效的程序设计方法，称为结构化方法，它体现在以下三个方面。

（1）程序设计采用自顶向下、逐步细分的方法展开。

（2）模块化。这是指程序中的过程体和组成部分应以模块表示，模块还可以称为过程、子程序或函数，C 和 C++中统称为函数。每个模块应具有较高的独立性，即强内聚性和弱外联性。这里的内聚性是指模块内部功能的单一性，而外联性是指同其他模块的联系。这样做的目的是使对一个模块的修改不致于对其他模块造成太大的影响。

（3）使用三种基本控制结构。这是指描述任何实体的操作序列只需要“顺序、选择、循环”这三种基本流程控制结构。从整体上看，模块中的指令（语句）总是由上到下按顺序逐个执行的，但局部可能有选择或重复（循环）。这三种结构的共同特点是每种结构只有一个入口和一个出口，从而使得程序更容易理解和维护，如图 1-2 所示。

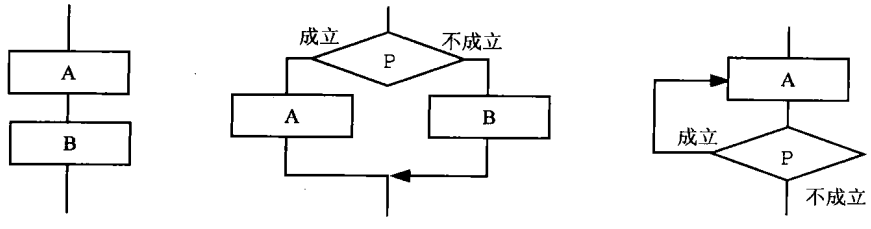

图 1-2　三种基本结构

这里考虑一个五子棋游戏的过程化程序设计示例。采用过程化方法可以将整个问题按处理过程分解如下（括号中为函数需要处理的数据）：

开始函数();

绘制画面函数（棋局数据）;

走棋函数（哪种棋子）;

判断输赢函数（棋局数据）;

输出结果函数（棋局数据）;

结束函数()。

完整程序可以按下述流程实现：

```
int main( )
{                                  //初始化工作，如假定黑棋先走等
   开始函数();
   绘制画面函数(初始棋局);
   重复下述步骤
   { 走棋函数(黑子);
     如果(判断输赢(棋局)==已定出输赢)停止重复;
     走棋函数(白子);
     绘制画面函数(棋局);
     如果(判断输赢(棋局)==已定出输赢)停止重复;
   }
   输出结果函数(棋局);
   结束函数();
}
```

由于过程化设计中的数据与过程是相互独立的，一个过程完全可以作用到并不相关的数据上，也不能保证对数据操作的合理性，数据对于算法（操作）完全是被动的。由于函数过程表示一种操作，数据是操作对象，因此，这种操作是一种“**谓语＋宾语**”结构。

1.2.2 面向对象的程序设计

面向对象设计方法将客观世界看成是由对象组成的。对象是一种实体，具有自己的属性和行为。一个程序也由对象及其相互间的协作来实现。

例如，对于五子棋游戏，完成一次比赛应由两个选手、一个裁判和一个组织者组成。选手只负责走棋，裁判负责判定输赢和确定比赛是否结束，组织者提供场所和设施。这里的每个人是一个对象，对象之间通过消息实现协作。不同对象应具有不同的行为，例如：

选手：走棋();

裁判：确定比赛开始(); 判断输赢(); 确定比赛结束();

组织者：绘制画面(); 输出结果()。

这样的程序可以按下述结构实现：

```
int main( )
{  定义对象，包括黑棋选手，白棋选手，裁判，组织者;          //初始化工作
   裁判.确定比赛开始();
   组织者.绘制画面();
   重复下述步骤
   {  黑棋选手.走棋();
      如果(裁判.判断输赢()==已定出输赢)停止重复;
      白棋选手.走棋();
      组织者.绘制画面();
      如果(裁判.判断输赢()==已定出输赢)停止重复;
   }
   组织者.输出结果();
   裁判.确定比赛结束();
}
```

其中，棋手对象负责走棋，并告知组织者对象有关棋子布局的变化，组织者对象接收到这些信息后负责在屏幕上面显示这种变化，同时利用裁判对象来对棋局进行判定。

限于篇幅，上述例子还不能全面解释面向对象程序设计的所有特性，但可以明显地看出，面向对象是以功能而不是步骤来划分问题的，这种方式与人的思维方式吻合。不同对象各司其职，对象自身行为实现方式的改变不会影响到其他对象，对象间通过互通消息实现合作，如图 1-3 所示。

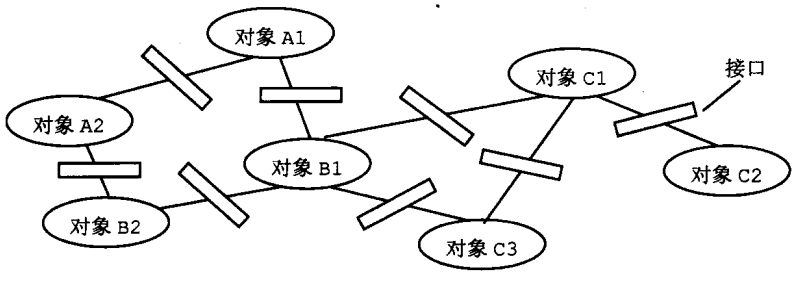

图 1-3 面向对象运行模式

从实现上看，对象的每次操作都是在该对象接收到一定消息后的自主行为(用“对象.函数名”形式表示)，具有“主语+谓语”的形式。

在采用面向对象技术解决问题时，首先要考虑的问题是整个项目的求解是由哪些对象组成的，再分析对象应具有什么样的行为，随后考虑对象之间应怎样协作。

总之，过程化与面向对象是两种分析和解决问题的不同方法，对于一些简单的问题，基于过程的解决方法是十分有效的；而对于大型、复杂的系统，采用面向对象的方法更能显示出优势，有利于利用对象构成软件“积木插件”，进而在一定程度上解决软件重用的难题。

1.3　面向对象程序设计的主要特点

面向对象程序设计（Object Oriented Programming，OOP）认为客观世界由对象组成。对象是一类概念的具体实例，有形体特征和具体行为。例如，对于“看电视”这样的一个问题来说，“人”、“遥控器”、“电视”是问题中所涉及到的概念，每种概念在 C++中称为“类”（class），而客观存在的实体对象都是某种概念的一个具体实例。例如，解决看电视问题时，需要一个“我”，这是“人”类的一个对象；一个具体的电视机，是“电视”类的一个对象。此外，还包括一个“遥控器”类的对象。

1. 类（class）与封装（encapsulation）

对某种概念的描述结果就是一个类。例如：

```
class TV
{  public:
     int color;                                  //颜色，一种属性
     int size;                                   //尺寸，一种属性
     //...                                       //其他属性
   public:
     void open();                                //打开，一种方法
     void close();                               //关闭，一种方法
     void changeChannel(int channel);            //调频道，一种方法
     //...                                       //其他方法
};
```

显然，类描述了一种概念（此例中是电视 TV）的属性特征和行为特征，属性是数据，行为是函数（算法），也称为方法，属性和方法都是类的成员。通常，类可由图 1-4 所示的方法描述。

类的定义是一种数据类型定义，因此，可以像 C++的内置类型那样生成变量，这种变量就称为一个类的实例或对象。类的所有实例具有相同的行为，但可以有不同的属性。

```
TV  aTV;       //对象定义
```

一个类定义最直接的效果是实现了对数据和函数的封装，而封装的主要目的是数据隐藏，这使对象本身所包含的内部数据不致于被无意中破坏。对电视机的使用者来说，如果需要换频道，可以通过遥控器向电视机 aTV 发送消息（调频道信号），而如何调频道是电视机本身的行为，由电视机本身来实现。使用者不必关心电视机的内部结构，也不必知道其内部工作原理。

访问对象的属性或方法一般采用“对象名.成员名”表示。例如：

```
aTV.size = 100;
aTV.open();
```

不过，由于来定义时允许指定成员对外界的公开程度，因此，外界有可能无权访问类的某些特殊的成员。

2. 继承（inheritance）

如果考虑制造新的电视机，可以有如下两种方法：

（1）从底层全新设计，即从勾画电视机的草图开始；

（2）在原设计基础上新增功能。

很明显，后一种方法会得到更多的好处，而面向对象设计方法也采取了类似的做法：从已有的类派生新类。这种处理使得新类继承了已有类的所有属性和方法，但又可以新增加必要的成员，以体现新的功能和适应新的要求，如图 1-5 所示。

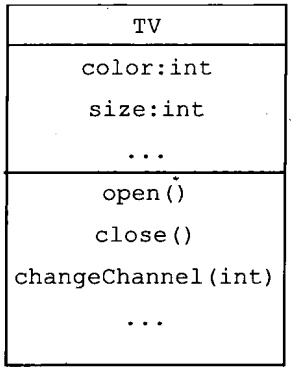

图 1-4　类的图形描述

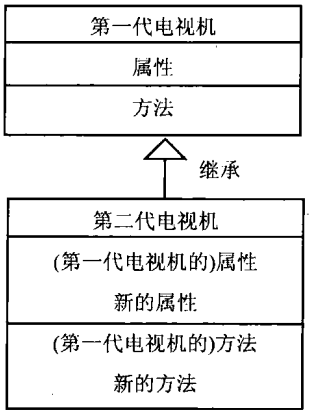

图 1-5　类的继承

支持继承可以使已经设计好的、经过考验的类得到最大限度的重用，不仅减少了新设计工作的工作量，也使新类更健壮和容易调试。可以说，继承的主要目的就是代码重用。

3. 多态（polymorphism）

多态是指多种形态，或者说不同的对象在相同的概念下能够表现出各自的特殊行为。例如，一只鸡和一条狗是两种不同的对象，都具有“走”的行为，但其走法各不相同。这是一种“低级的”多态行为，C++利用“函数名相同，但函数体不同”的函数重载来体现该行为。“高级的”多态是指在具有继承关系的类中，祖先类和子孙类之间行为上的差异，采用虚函数的方法来实现。

封装、继承和多态构成了面向对象程序设计的三个最基本的特征。通常，如果一种程序设计技术仅支持对象封装行为，则被认为是基于对象而非面向对象的。

此外，由于面向对象的程序是由对象组成的，对象之间需要通过消息传递来达到协调工作的目的。**消息**是对象之间进行通信的一种规格说明，或者说消息是指希望一个对象执行某种操作的请求，而对象执行操作称为对消息的响应。从实现代码看，消息就是通过一个对象对类的成员函数的一次调用。例如：

```
TV aTV;
aTV.changeChannel(12);                    //一条消息
```

可见，一条消息包括接收此消息的对象名、消息名和必要的参数（此例中分别是 aTV、changeChannel 和 12）。上述消息的含义是请求对象 aTV 执行调换频道操作。

1.4　程序的编辑、编译与运行

用高级语言编写的程序并不是计算机能够直接识别并执行的指令集，而是一个文本文件，称为“源程序”文件，建立源文件的过程称为“编辑”。为了能够运行源文件，必须将其转换为机器