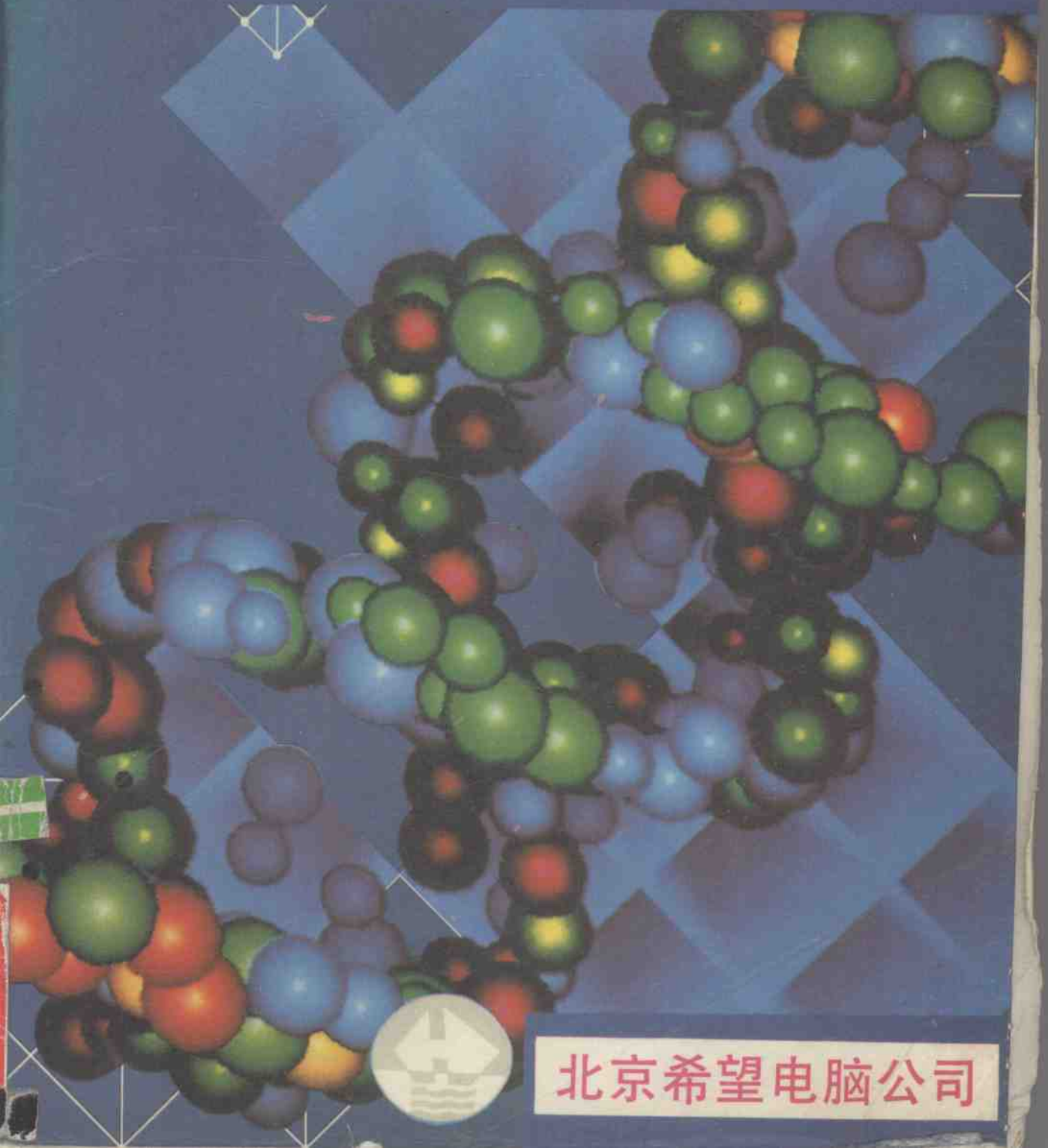
...PUTER COM...NY LTD.

Turbo C++使用手册

晓 峰 编译



北京希望电脑公司

Turbo C++使用手册

晓　峰　　编译

北京希望电脑公司

一九九一年十月

内容提要

本书是一本专为从事 C++语言程序设计人员而编写的参考教程，它将引导读者熟悉 C++语言以及 Borland 公司的 Turbo C++独具特色的内容。

与其它关于 C++的书不同，本书并不需要读者因学习这种新的面向对象语言而将 C 语言放弃，而是引导读者在基本的 C 语言基础上学习 C++，这如同在掌握了实数以后学习复数，开始时可能对实数到复数域的转变不适应，但只要熟悉了复数域，则很少有人再局限于实数这个小圈子了。

本书首先帮助读者建立 C++语言的知识，全书分为三个部分：

第一章和第二章是关于一般形式的面向对象程序设计方法以及 C++语言特点的总体介绍，它旨在使读者熟悉 C++的语法结构，并对在学习中出现的问题做一定的说明。

从第三章开始到第十章是关于 C++语法结构的详细说明，并对其中特殊的约定，包括不符合基本法则的内容加以分析，在本书中也给出了一些程序实例并简要地介绍了由源代码生成的汇编语言，在第十章中详细介绍了 Turbo C++独有的高级功能，所有这一切都将帮助读者了解 Turbo C++。

最后，在第十一章和三个附录中给出一个应用实例：一个完全由 C++语言编写成的多任务系统，此例中的 Task 类适用于多任务应用环境，包括在 DOS 环境下的信息传送以及过程控制。

目　录

引　言

《Turbo C++使用手册》是一本专为从事C语言程序设计人员而编写的参考教程，它将引导读者熟悉C++语言以及Borland公司的Turbo C++独具特色的内容。

虽然在程序语言中存在差异是不可避免的，但只有那些能够产生效益的变更才能被接受，与其他关于C++的书不同，Turbo C++使用手册并不假定读者因热衷于这种新的语言而将C语言放弃，而认为在熟悉一种新的语言过程中，一定程度的怀疑加上相当的耐心是完全必要和可以理解的。

在基本C语言的基础上来学习C++，就如同在掌握了实数后学习复数，所有实数的运算都可以在复数域进行（只是其原理常常略有不同），复数赋予这些运算以新的含义，开始时人们可能对这种从实数到复数域的变化不太适应，但只要熟悉了复数域，很少有人再局限于实数域这个小圈子里了。

《Turbo C++使用手册》首先要使读者建立起C语言的知识，全书分三个部分：第一章和第二章是关于一般形式的面向对象的程序设计方法以及C++语言特点的总体介绍，它旨在使读者熟悉C++的语法结构，并对在学习中出现的问题做一定的说明。

从第三章到第十章是关于C++语法结构的详细说明，并对其中特殊的约定，包括不符合基本法则的内容加以分析。《Turbo C++使用手册》中也给出一些程序实例并简要地介绍由这些源代码而生成的汇编语言，便于读者了解到Turbo C++是如何执行的。

最后，第十一章及三个附录给出一个应用实例：一个完全由C++写成的多任务系统。此例中的Task类适用于多任务的应用环境，包括在DOS环境下的信息传送以及过程控制。

本书的读者

《Turbo C++使用手册》适合那些对学习Turbo C++时对C已有所了解，起码已知晓基本的Kemighan和Ritchie形式的读者。本书假设读者已具备使用C编写某些短小程序的能力，读者不一定要了解面向对象的程序设计及其语法特征方面的知识。

学习C++的意义

尽管C在程序设计中很受欢迎，但它仍存在一些缺点。比如由于设计起来较困难而导致较大的成本，这类问题在目前应用软件的编写中越来越突出。

十年前结构化语言的出现带来了一场整个软件产业的革命。当时结构化技术被认为可用来控制软件设计的成本及设计进程，而实际上尽管较以前有较大改进，但并未达到人们所期望的程度。

大规模的程序设计通常需要投入大批量的程序设计人员，而程序员的增加导致相互之间信息交换的显示增多，这是因为一个模块的改动将影响其所有相关模块。系统要由很多

程序员来协调，这样当某个程序员在编写新的程序代码时，要花费更多的时间去检查和纠正各模块之间以及程序员之间可能引起的信息传递错误。

由于采用了数据抽象和封装技术，面向对象的程序设计降低了各模块间的关联程序，这就相对减少了程序员之间的相互影响。这项技术是在设计初期只有很少的程序员介入的情况下，通过在对象系统中建立一个高层次的通讯环境来实现的。它使得今后更改引起的成本大大降低。

重复调用已存在的代码是减少开发费用最有效的办法。过程程序这项技术是在实验室中设计出来的，人们曾指望通过它来提高代码的重复利用率，但事实证明这是一次失败的尝试。面向对象程序设计的思想是：详细定义用户的数据类型，将它们封装在一起以实现较高的代码利用率。

许多早期的程序设计语言只是说明性的，实际应用意义甚微。相比之下 C++能产生高效实用的机器码，它引导面向对象的程序语言走出实验室进入实际应用领域。

关于《Turbo C++》

《Turbo C++》是一个高度兼容性的编译器,它支持 ANSI C 和 C++ 2.0 版。Turbo C++标志着 Borland 公司在其编译器家族更新中迈出了新的一步。

Turbo Pascal 编译器是第一个具有集成环境的 Turbo 系统，它编译方便并能快速地定位错误位置。尽管 Turbo Pascal 没有提供产生最小或最快执行速度的程序性能，但由于它为用户提供了极为方便的操作环境，其不足之处也显得微乎其微了。

酝酿已久的 Turbo C (1.0) 版于 1987 年夏季的出版将 Turbo 引入 C 语言领域。虽然 Turbo C 比 Turbo Pascal 的软件包大许多，但编译速度丝毫不亚于 Turbo Pascal 并且操作环境更加方便。同时，Microsoft 公司也推出了 Quick C 与 Turbo C 相抗衡，引起了一场 C 语言的较量。

Turbo C (2.0) 于 1988 年秋季问世，它进一步增强了屏幕交互调试能力，使操作更为简便。Turbo C++进一步完善了这项功能，它增加了一个高性能，即支持鼠标的窗口界面。同时，它支持面向对象的程序设计语言——C++。

C++是 AT&T 公司于八十年代初期开发的程序语言。编译一个 C++程序，程序员首先要运行一个名为 cfront 的转换器，把 C++程序转变为 C 源程序，然后再把 C 源程序编译为可执行文件。而 Turbo C++则是一个完整的编译系统，程序员通过交互式环境输入命令，首先经过预处理器,然后直接编译产生可执行代码，整个过程只占用极短的时间。这使得分两步的编译方式，即选用于专门编译器进行错误处理的方法成为过去。

在进行本书的阅读之前，需要说明的是《Turbo C++手册》全书采用 Borland 公司推荐的命名规则，除以下几种情况，全部目标模块名均用小写字母。

- 结构、类和联合名称的第一个字母用大写（如 struct Base)
- 宏定义均采用大写（如#define MAXARGS)
- 由多个字组成的变量名，第一个字以后的所有字的第一个字母均用大写。(比如 .struct Base baseObject)
- 指针变量名以 Ptr 结尾（如.baseObjectPtr)

另外，本书中 C++的关键字、变量名及其它与 C++源代码相关的语法结构均用印刷体以区别其它内容.

第一章　Turbo C++内容介绍

从 Borland 推出 Turbo Pascal (1.0) 版到 Turbo C (2.0) 版，在其众多编译器中，Turbo C++是最新成果，它与其家族中其它成员的相似之处是显而易见的。

但是，Turbo C++不仅仅是 Turbo C 的环境，它还支持面向对象语言中的佼佼者 C++ (2.0) 版以及 C 语言。

后面两章将对 Turbo C++作详细的介绍。第一章首先讨论面向对象程序设计的基本概念，进一步分析 C 同 C++之间的联系，以及 Turbo C 与 Turbo C++系统的比较。第二章介绍 C++所独有的特征。第三章开始对 C++的语法结构进行剖析。

程序员在学习一种新的语言时可能会有几点疑问，为什么要用这种语言？它与自己已掌握的其它语言相比有什么优点？下面重点从两个方面出发来剖析 C++特别是 Turbo C++与 Turbo C 的优点在哪里？

1.1　面向对象的程序设计语言

C++同面向对象的程序语言（简称 OOP）有密切的内在联系，它是在 C 语言中加入面向对象的程序结构而形成的。(C++并不是唯一的收获，人们同时也在开发研究面向对象的 C 语言，虽然它的原理相似，但实现手法不同。) 要理解 C++，首先要了解有关面向对象程序设计方面的知识。

1.1.1　定义术语:

多数对面向对象的程序设计语言的定义可归结为"面向对象的程序设计好"，似乎今后在程序理论方面只有研究发展面向对象的语言才有意义。

人们通常把面向对象的程序语言与 Smalltalk 混为一谈。Smalltalk 是一种初级的，纯粹的面向对象语言，因而受到人们的关注。Smalltalk 是最早应用在 Xerox PARC / Macintosh 的语言之一，它用来处理用户接口界面"point and shoot"，这使得一些人将用户图形接口与面向对象语言混为一谈，认为面向对象的程序语言一定具有某种图形接口，而事实上 Smalltalk 在这两个方面是毫不相干的。

Pinson 和 Wiener 是这样定义面向对象的程序语言的："能够支持对象所具有的数据抽象、封装、继承及多态性这四种属性的计算机语言是面向对象的程序语言。" (Pinson and Wiener, An Intro-duction to Object-Oriented Programming and small talk 1988) 这种定义是比较合理的，但它仍需加以说明。

面向对象程序是一种程序设计范例，它可以理解为程序的方法或设计风格。但其中最根本的一点是，人们可用它来逐步解决问题，而在问题逐渐深入过程中，不必去重新修改

细笔尖

前面已完成的设计工作.

程序设计范例是用来解决程序问题的一种约定. 程序员在进行繁重的设计时必须牢记这些法则以达到最佳设计方案. 但是, 程序员有时还要审察这些法则, 看它是否便于发挥自己多年来集攒的工作经验和知识. C++鼓励读者剖析面向对象式程序的语法规则.

一种程序语言, 如果它支持面向对象的语法特征就被称为面向对象的程序语言. 当然, 支持是个相对的术语. 人们曾经争论汇编语言是否支持结构化程序, 因为可以使用汇编语言来编写结构化的代码. 但是, 通常汇编语言不被认为是一种结构化的程序语言. C++创立者 Bjarne Stroustrup 曾说过: "一种程序语言, 如果能提供方便地使用这种程序设计风格的软件, 就称这种语言支持这种程序设计风格." (Stroustrup, "what is Object-Oriented Programming?"1987).

用一种具有广泛用途的语言支持某种程序设计风格的所有特征是不可能的. "我可以用任何语言写 Fortran 代码." 这句话谁没听说过? 例如, 我们基本可以用 Pascal 来编写非结构化程序, 虽然 Pascal 被公认为标准的结构化程序语言, 但它事实上仍包括 Go To 语句, 尽管这样被普遍认为是违背结构化程序法则的.

C++是一种面向对象的程序语言, 它支持面向对象的语法特征. 做为 C 语言的一个超集, C++也支持 C 的规则. C++是目前各种语言的混合体, 包括 Apple 公司面向对象的 Pascal, Borland 的 Turbo Pascal 5.5, Forth 语言中 Neon, Flavors 以及 Lisp 语言中的 Expert Commlisp 和 Common Object. 它还基本包括 Logo 语言中的 Objective logo. 事实上, 多数程序语言都提供了面向对象的程序设计语言版本, 另一些则正在研制中. (见 Peter Wenger 的文章"Learning the Langrage", 1989, 3. 内容是关于以上语言及其它语言的分类).

1.1.2 概念的演变

早期的程序结构可以说是缺乏条理的. 在这种模式下, 人们考虑的仅仅是如何解决问题而很少去思考问题是如何解决的. 代表这种无序的语言的程序结构包括 BASIC 的初期形式 (它甚至没有子程序结构). 在这种混乱的结构中, 用户无暇顾及到如何去组织程序以及开发的消耗. 全部注意力都集中到问题本身. 这种缺乏条理的程序只能用来解决很简单的问题 (最多几百条语句). 无论多大的程序, 代码的书写方式千篇一律, 使调试和检测非常繁锁 (称为"Spaghetti code"问题).

解决这种问题的第一步是引进过程程序设计 (procedual programming). 代表语言如 Fortran 和 Cobol. 在这种结构中, 程序员试图将问题分割成一系列连续的函数. 理论上讲, 可把每个一个函数作为一个单独的部分, 每个函数的复杂性大大减化, 源代码和数据被分开存放. 人们考虑的是对代码的抽象处理, 并不直接考虑数据的处理问题.

这种主导思想下, 进一步产生出了结构化程序设计 (structured programming), 程序代码只出现在函数内, 程序的无序性也被限制在函数内部. 虽然过程程序设计也体现了这种思想, 但结构化程序设计更进一步. 它允许程序员将每个控制结构 (例如 for, while, if 程序块) 作为一个小的子函数, 使之进一步被简化. 例如: 用户可以将一个 for 循环看作是一个独立的部分. 因为用户知道出入这个函数只有一种途径, 而不必担心会有

从中途进入循环，未执行完就跳出的情况。一种语言要做到支持结构化程序，就必须具有一个完整的控制结构系列，如同我们在 Pascal 中看到的那样。

在结构化程序设计范例中，程序中的数据不再被忽视，制定了函数间数据传递的规则。函数只能接受那些做为输入变元的数据，不再提倡使用全局变量进行信息传递。

这种限定是针对过程模块中现存的缺陷制定的。如果一个函数同全局变量进行数据交换，那就很难把它作为一个抽象的整体处理。因为对这个问题的表述很困难，所以在数据传送中采用了易于描绘的形式。但这些规定只是作为参考而提出的，因此严格遵从这些规定的设计并不普遍。

很多结构化程序设计鼓励程序员采用能更准确描述实际对象的数据类型。在 C 中，简单的数据类型可采用枚举（enum）或类型定义（typedef）。组合的数据一律采用某种形式的结构。结构定义把相关数据集中到一个整体中，使它们更接近现实中的对象。

今天多数程序员都选用介乎于过程化和结构化之间的设计形式。

模块化程序设计（modular programming），是一种扩展的结构化形式。函数被进一步分成可单独进行编译的源文件——称为模块（modules）。每个模块有一系列只有它自己才能访问的数据和函数。某个系列的数据和函数可以被所有模块访问。所有相近函数存储于同一模块中。模块内的函数定义不一定被外界知道。限定全局函数的定义使模块间信息传递的方式减少，降低了整个系统的复杂性。

这种结构使数据隐藏和封装成为可能，在这种情况下，数据只可被某个函数调用而对其它模块是不可见的。程序员可以定义一种新的结构并说明一系列函数对其进行操作。对外部模块来说它是一种新的数据类型，只有调用公有函数才能访问它。

模块外可见的数据被称为公有的（public），而另一些模块使用的新数据类型对外界是不可见的，被称为私有的（private），在 C 中，静态的（static）或自动的（automatic）元素为私有的，其它则为公有的。

这是一个严格的语法规定，通过对问题的分析，程序员首先定义出所有可能用到的新数据类型。每一种类型都有一个单独的说明模块。每个模块中定义一系列函数来完成构造、删除、显示等对这类数据的操作。一旦建立了这些类，其处理程序可以很简单而易于阅读。程序员可以采用新的数学表达式而不必为操作这些数据结构担心。详细的结构定义包含在模块定义中。修改数据结构通常只需要修改其说明模块，而不用涉及核心模块。优秀的 C 程序员都采用这种语法结构。

令人遗憾的是，这种形式的使用是自发的，并非每个人都遵从。如果结构成员以一个非法的数据结束时，程序员就无法确认究竟是由于访问函数赋值造成的，还是由于其它不相干模块直接对成员函数进行了访问，具有数据封装特性的语言加强了可访问函数的作用。

新的语法规则并不一定表明它将在程序中得到应用。但支持数据抽象的程序语言使程序员可把内部运算符定义为新的类型。比如，程序员定义了一个新类型 complex，程序员可为 Complex 定义简单的数学运算符（+，−，*，/）。也就是说可按一定意义来设定。

```
complex A, B, c;
```

```
C = A + B;
```

请注意作者并没有用重定义这个词．因为用于内部数据类型的运算符仍然有效．

由于数据抽象技术，程序员在使用数学表达式时对细节问题可以更加放心．除 C++ 外，Ada 是最常用的一种支持数据抽象的实时语言．

定义新的数据类型，包括可能用到的运算符，可能是非常繁锁的过程．特别是每次定义都从头进行．另外，只使用基本的内部类型来定义往往不能直观地反映两种用户定义类型之间的关系．如果把用户已定义的类型作为基础类型派生出新的类型，那么数据类型定义就容易的多，也更易于描述．

例如，大学生是学生的一个特例．这两种类型有许多共同属性．我们可以根据 Student 的定义进一步定义出 College Student，两者的关系就很明显了．这就叫做继承 (inheritance)．

作了这个定义之后，我们仔细观察这两种类型．一个 College Student 一定可以访问为 Student 所定义的函数．可以接受 Student 类型的指针的函数也许完全可以接受指向 College Student 的指针．因为大学生属于学生，这一系列继承特性被称为多态性 (polymorphism)．

这里又回到关于面向对象的程序语言定义，一种面向对象的程序语言支持数据封装、压缩、继承和多态性．C++支持这些特性，同时也支持 C 的结构化程序．后面我们将通篇围绕四种特性进行说明．

1.1.3 为什么存在障碍?

很多关于 C++和面向对象程序设计的教材都没能注意到这样一个问题，为什么存在障碍？并不是所有的 C 程序员都能够顺利地过渡到 C++．

如果 C 程序员打算在 C++中书写同样的程序，他就没有必要做这种过渡，尽管 C 代码也可以编译出与 C++相同的程序．C++与 C 是不同的语言，虽然它具有许多相同的结构，但仍有必要强调这种过渡所引起的不同．

在所有程序规则变动引出的问题中，最基本的是成本问题．当受到内存限制，一个大程序只有几百条语句时，缺乏条理的设计方法也许还行的通．但程序一旦再扩大，这种方法便不能胜任了．同样，结构化程序作为课堂实验可以满足要求，一旦程序扩大到几 M 字节而且有多人界入时，这种方法也行不通了．

C++及其它面向对象的程序语言降低了建立高性能程序的成本．AT&T 的报告中描出，在大系统中"与其它传统方法相比，极大缩短了软件集成时间．" (Coplien et al, "C++: Evolving Toward a More Powerful Language."1988.) 另外，"减轻了程序设计后期和整个调试维护过程中各部分间的通信负担．"这种效果是通过增加程序设计初期的信息传递实现的，因为初期设计时成本很小．

软件的重复利用水平对程序成本有很大的影响．如果能利用已存在的代码，在程序设计中就会非常明显地降低所用的成本和时间．合理运用数据抽象来分隔数据及相关函数可极大地提高代码重复利用率，这是因为程序中公用的数据可以被直接使用．

但是，无论开发代价多么低，如果程序的性能不满足需要，它就没有实用价值。从无条理的设计到过程程序设计的发展过程中，一些程序员认为效率降低而提出了异议。采用函数调用方式要在程序前面加上一个说明，而过去则是采用直接插入代码的方法。随着结构化程序不断扩大，每个函数调用的定义也随之增大，效率问题也就越突出。把变元压入堆栈要占用较多的 CPU 时间，而过去则使用全局变量，省去了许多附加操作。

对于小型函数其附加成分是可计算出来的，通过记录由不同程序结构组成的函数占用的 CPU 指令是一个常用的方法。但在实际中，这种分析方法是不行的。既使在较极端的情况下，附加成分比重很大（超过 100%），除在对时间要求很严的程序中外，几乎对程序毫无影响。这就是那条普遍规律"20%的代码用去 90%的程序执行时间。"

另外，有些人认为两个程序员采用两种不同的程序形式来解决同一个较复杂的问题可以达到相同的效果。作者根据自己的经验得出完全不同的结论，程序员往往趋向于使用并非最优的解决办法。与最优设计相差多远不仅是个人设计风格以及函数处理水平问题，它还取决于程序设计范例。一个采用较为先进设计方法的程序往往能设计出水平相对高的程序。高水平的数据抽象避免了程序员只见树木不见森林，同时完全放弃了简单程序规则故有的速度优势。

C++的设计是把效率作为重点考虑的。AT&T 研究表明"只使用 C++中的 C 子集代码产生的目标与普通 C 编译器产生的目标是一样的。"程序员在掌握使用 C++的特征并加以运用之后，不仅没有失去效率反而获得了更大的成效。"

在表 1-1 中，将 Turbo C（2.0）版与 Turbo C++根据著名的 sieve of Eratosthenes 准则相比较，表明两种编译器的生成文件大小几乎相等且执行时间基本相同。唯一较明显的区别是在编译过程中 Turbo C（2.0）版速度更快。

作者在同其它语言的比较中也得出大体相似的结论。尽管 Turbo C++在编译时稍慢，在执行功能相同的程序，在执行时间和代码长度上都接近于 C 语言。

表 1-1 Sieve of Eratosthenes 定时比较

	Turbo C 2.0	*Turbo C++*
Size of .OBJ file [bytes]	1,085	1,125
Size of .EXE file [bytes]	19,072	18,323
Time to execute [sec]	58	57
Time to compile [sec]	3	7

1.2 从 C 到 C++

C 是 C++的一个起缘，C++具有 C 的一切结构特性。在 C 下编译通过的程序就能在 C++下编译，而那些遵循 Kernighan 和 Ritchie 标准的程序在 C++中可能会出现错误，并且一定会有警告信息。在 Turbo C 2.0 下无警告的程序（当然出现各种警告是允许的），在 Turbo C++下不会有编译错误。

要声明的是，除非 C++菜单 Options / Complier 中的选项处于始能状态，Turbo

C++可以通过检测其扩展名来区别 C 和 C++，如果源程序以 C 结尾，它就被认为是 C 程序；如果以 CPP 结尾，就被认为是 C++程序。

1.2.1 C++与 ANSI C

C++与 ANSI C 有许多共同点。ANSI C 中的许多扩展的函数定义形式及对类型的严格区别都是源于 C++。另外，C++扩展了包括 ANSI C 后来制定的语法规定以保持最大的兼容性。C++与 ANSI C 在下面语法规定中，可能存在着微小的差异。

1.2.1.1 函数原理:

在早期的 Kernighan 和 Ritchie C 中，一般只需要对变量的大小及函数的返回类型进行说明。如不加定义，函数就默认是 int 类型。这种规定的前提条件是，返回的整型数符合 CPU 寄存器的大小并且处理器不会因寄存器赋值而出现问题。

C++和 ANSI C 都要求严格的类型检查，程序员不能再把不同的数据类型混在一起存放（在表达式中可以同时存在字符型，整型和浮点型数据，因为 C 可以把它们区分开）。在 PC 机上，虽然指向整型的指针与指向字符的指针有着同样的大小，但下面的语言仍是不可接受的。

```
int*  a;
char* b;

a = b;              /*not acceptable without a cast*/
a = (int*)b;        /*acceptable with the cast*/
```

附加的造型（int*）把 b 从字符型指针转换为同 a 相匹配的整型指针。C 语言中造型总是以一种类型的形式出现，它由括号括起，直接位于被转换数据的前面。

严格的类型限制并不加重程序员的工作负担。一些早期的语言，如最有代表性的 PL／1，其采用的方法是使编译器尽可能地输入任何内容，以求节省程序员的工作量。如果使用一个未经定义的变量，程序会从变量的使用方法及拼法上推出其类型并予以自动说明。

这种做法在编译器试图去分析一段未知的程序时往往会导致错误。当遇到某个未定义的变量时，它很可能是由于程序员对某个已定义变量错误地拼写造成的。指出这种拼写错误可以引导程序员发现错误，而如果把它做为新的变量处理，就会给程序发现并纠正错误造成障碍。

采用严格类型区分的目的是为尽早在程序运行中发现错误。通过采用一系列严格的规则，编译器能更有效地诊断程序故障。

严格的类型检查不但用于变量而且还包括函数。ANSI C 不仅允许定义函数返回类型，同时也包括全部的变元。当调用某个函数时，要将每个变元的类型与函数的变元表进行比较。比如只定义一个函数为整型函数还不完全，还要定义函数带有一个整型和一个字

符型变元，并具有整型返回值.

这导致两个问题：(1) 函数在模块中未使用之前就要对函数做定义，这难度往往很大.(这一点可向任何一个 PASCAL 程序员询问).(2) 确定在其它模块中定义的函数类型，而这些模块只有到连接时才合在一起.要解决这个问题，C 允许无具体定义的情况下说明一个函数，这就叫做 prototype declaration，在 K&R C 中虽然允许这种形式，但通常不被采用，原型定义在 ANSI C 和 C++得到了完善，两种语言采用了新的函数说明(包括原型说明格式).在新的格式中，所有变元的名称和类型出现在一行中，例如:

```
unsigned func(unsigned a,int b);
```

而不是

```
unsigned func(a, b)
unsigned a;
int      b;
```

然后函数定义为:

```
unsigned func(unsigned a, int b) {



}
```

当程序员不对函数的变元个数和类型加以规定时，可以用省略号.下面两种原型定义是: 函数 f1 ( ) 带有一整型变元后接任意多个变元.函数 f2 ( ) 带有任意个类型可变的变元.

```
void f1(int a, ...);
void f2(...);
```

目前普遍采用的形式是在模块内为所有的已定义函数构造一个原型函数，它常放在一个包含文件.H 中或 C++的.HPP 中，这些文件由调用任意函数的模块包含.调用一个无原型的函数在 Turbo C++中会产生警告并可能导致一些奇怪的错误.

1.2.1.2 Void (空)

C++与 ANSI C 都使用关键字 void.最后，函数不做返回类型定义时被默认返回一个整数.例如: fn ( ) 和 int fn ( ) 定义是相同的，这是因为返回整型的函数把数值传

给一个寄存器（80×86机器中的AX寄存器）。调用它的函数不处理这个返回值也不会有任何影响。对于编译器而言没有必要区别这两类情况。但这样我们就不能判定函数何时被正确调用了。也许程序员认为它返回了数值而实际上没有返回数值以及程序员不知道自己是否错用了函数。

在C语言早期的演变中，void这个术语用来表示一个函数没有返回值，这样的函数被定义为void fn（）（函数的默认型仍是int）。这样编译器使可以对错误的函数调用做出判断。

从那以来，void的作用不断扩大，现在它还用来说明一个无变元的函数。例如在ANSI C中，原型函数int fn（）与int fn（…）是相同的。采用int fn（void）方式说明函数fn（）不带参数。在C++中，定义int fn（）与int fn（void）是相同的，但后者具有更好的兼容能力。

除此之外，变量也可以说明为指向void的指针。一个指向void的指针不能进行地址增加、地址减少和取值操作，但它与其它类型的指针可以进行匹配。Void的指针可以在程序还不知指针指向的情况下存贮地址（在K&R C中这种方法常用一个指向字符的指针实现，但这可能导致读者和编译器的误解）。

最后，void可作为一种转换来表示一个计算结果被有意忽略。比如：下列语句在C和C++中是合法的，而且可能指明一个问题。

1.2.1.3 volatile和const（动态变量和常量）

```
int fn(char x);
fn('z');
```

在第二行，函数fn（）返回的数值被忽略。这种情况下，应当使用void表示是程序员有意忽略，而并非错误，如采用下开形式：

```
int fn(char x);
(void)fn('z');
```

动态变量和常量

ANSI C定义了新的存贮类型动态变量volatile和常量const，把变量说明为动态，表示此变量可能随时改变，因而编译器无须对其优化。它可能意味着此变量是一个内存映象的I/O设备或是一个全局变量，其它非同时编译的任务也可以调用它。

在实际应用中，这意味着一个动态变量在每次被访问时都要从内存调入。通常当同一个变量在一小段C语句中多次使用时，编译器将把其数值存在一个寄存器中以加速其调用，而不是每次都要访问内存。这就减少了不必要的内存访问。如果把一个变量定义为动态变量，就没有上述特性了。

请看下面一小段Turbo C代码，延时等待1秒钟。它是通过读取全局时间来实现的。在时钟单元中，直接从低内存增加1秒（18个时钟周期），然后等待存贮在低内存的