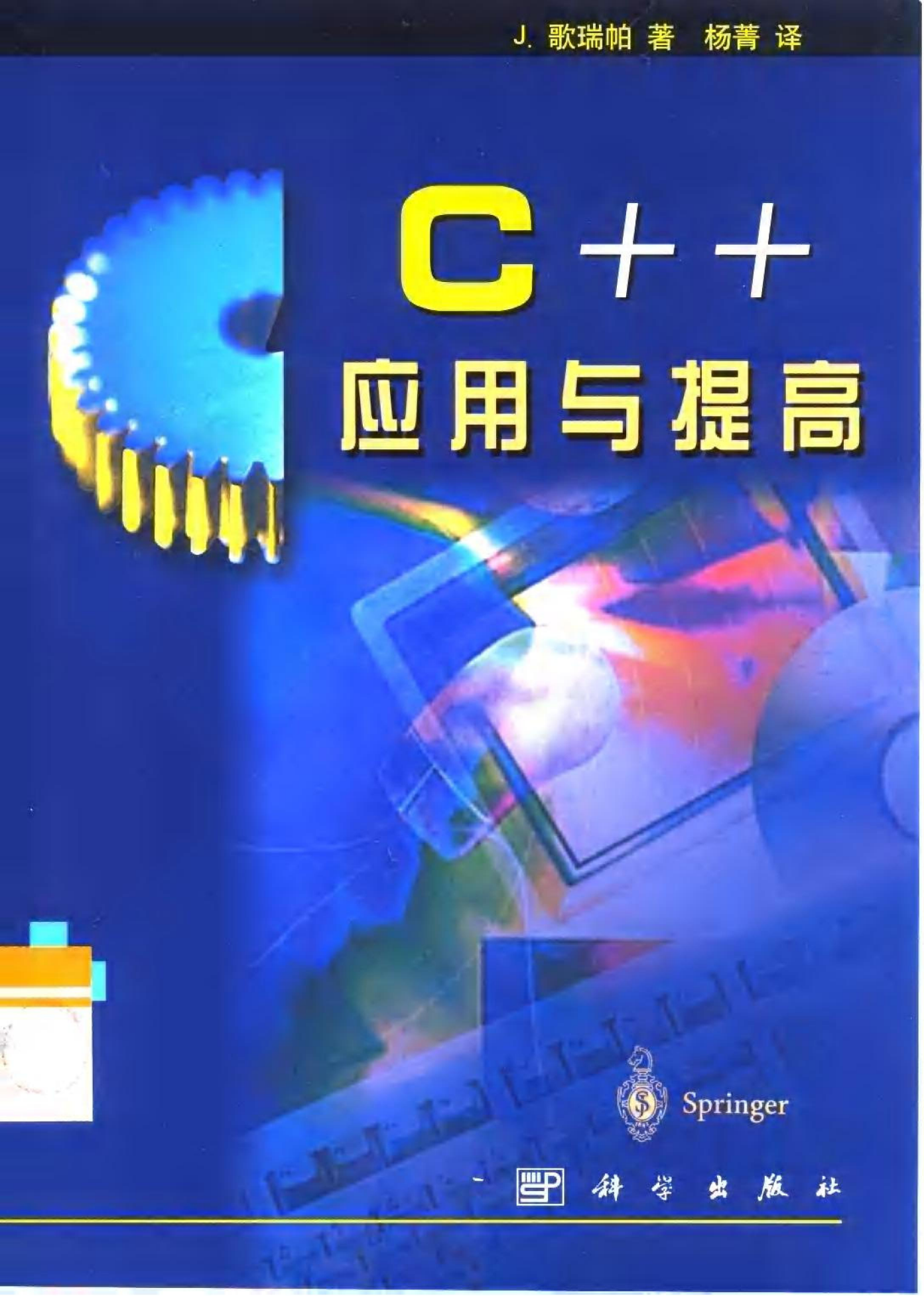
J. 歌瑞帕 著　杨菁 译

C++
应用与提高



Springer

科学出版社

C++应用与提高

J. 歌瑞帕 著

杨 菁 译

科学出版社

图字 01－1999－2391

内 容 简 介

本书内容包括数组、指针与引用，基本的输入输出，构造函数与析构函数，动态内存管理，继承性，多态性，友元函数，文件处理和模块等各个方面的知识，不仅内容全面，而且知识结构清晰、语言通俗易懂。在每一章的末尾都有相应的练习，以便于读者检验学习过程中对编程技术的掌握程度。

本书适合大专院校的学生以及想学习用 C＋＋编程的人员使用，它采用实例与学习相结合的方式，形象而直观，适合读者快速地学习，以尽快地掌握基本的编程方法、掌握编程的经验与技巧。

图书在版编目(CIP)数据

C＋＋应用与提高 / J. 歌瑞帕（Graba.J.）著；杨菁译 .－北京：科学出版社，2000

书名原文：Up and Running with C＋＋

ISBN 7-03-008340-7

Ⅰ.C… Ⅱ.①歌… ②杨… Ⅲ.C 语言-程序设计 Ⅳ.TP312

中国版本图书馆 CIP 数据核字（2000）第 04269 号

科学出版社 出版
北京东黄城根北街 16 号
邮政编码：100717
北京双青印刷厂 印刷
新华书店北京发行所发行 各地新华书店经售

2000 年 6 月第 一 版 开本：787×1092 1/16
2000 年 6 月第一次印刷 印张：19 3/4
印数：1—4 000 字数：468 000

定价：29.00 元

(如有印装质量问题,我社负责调换<环伟>)

序　言

针对很多读者反映电脑图书错误较多，尤其是编程图书经常出现所提供例子不能编译执行的情况，我们引进了本书。在引进翻译的过程中，不仅对技术问题进行了严格的控制，而且对所涉及的技术环节进行了验证，编辑并再次进行了核实，具有较高的准确性，因此程序的错误概率很小。我们认为这是对读者权益的最大保证，也是我们引进该书的主要目的所在。

该书在写作方式上，注意总结经验教训和针对不同作者群的学习特点，推陈出新，突破了计算机图书长期以来所形成的使用性较差的弊端，从实际操作入手，在各章节讲解操作流程、说明理论和概念时，均凝聚了作者多年的实践经验，该书语言流畅通俗易懂，使读者能够快速地掌握编程的技巧及加深对概念的理解。

译者虽倾心而译，但因水平有限，不足和错误之处，恳请读者不吝赐教与指正，我们定会全力改进，用更多、更好的引进版图书回报您的厚爱。

译　者

2000 年 4 月

目　录

第一章　面向对象的基本概念

目的

- 使读者了解面向对象的主要发展历史。
- 介绍和面向对象相关的技术。
- 使读者能够意识到利用面向对象技术所能带来的益处，并由此产生的对开发软件的重要性。

1.1　简短的历史

最初的 OOPL(Object-Orientated Programming Language——面向对象的程序语言)为Simula，一种主要用于模拟处理的语言。尽管 Simula 的最初发展可以追溯到 1967 年，但面向对象直到 1980 年才发挥其突出的作用。1980 年，另一个主要的 OOPL，Smalltaik 开始问世。如 Smilu 一样，在它出现后，在程序开发方面面向对象有着很大的影响。而面向对象的比较流行的且比较重要的是美国 Bell 实验室的 Bjarne Stroustrup 开发的C + +，这种语言可随意地进行扩充，并加强了现比较流行的 C 语言的结构程序，在 10 年前它自己也已面世。因此C + + 在一些传统的结构化的语言上得以发展，是一种继承的语言，而并非仅仅是一种面向对象的语言。

从 1980 年初，面向对象作为开发软件入门的流行趋势已逐渐升起，特别是在开发与用户进行通讯的 GUIs(图形用户界面)有了快速地发展。看起来这种流行比较长远，当然，在计算机软件发展方面，有许多意想不到的事情发生，但也不可忽略面向对象给软件开发所带来的重要作用。

1.2　基本的概念与术语

面向对象一词包括了几种不同的概念，并涉及到一些专业的术语，但均与以前用 C + + 开发的概念相类似，这一部分仅作为技术所必须的说明。

1.2.1　对象

在输出口所注册的基本事件就意味着短语“对象”(目录中的面向对象程序)。简单地讲，对象被定义为现实世界的物理实体与事件的软件复制。例如：飞机(物理实体)和一个人的资料(概念)的软件复制在此概念定义下均为对象。

然而，这种定义并非仅定义名词，也可意指数据的组合。一个对象不但可以含有“功能”(或方法)，也可和软件模拟的实体相联系。例如：飞机对象包含有起飞、改变速度和降落等过程。因此，许多比较精确的定义可用下面的简单方程来模拟：

对象 = 数据 + 方法

如此数据和方法之间的联系包装被称为“套装”，并且它也是面向对象中的一个基本概念。在以后也可以看见，它是一种保护来自外部的数据和由此产生少量中断所作的保护(至少在理论上)。对比之下，结构化程序只大量地单独保存数据与操作。(但是，松散的数据和操作捆绑与包装也可以通过线路、单元、模块或其他方法能够完成)。

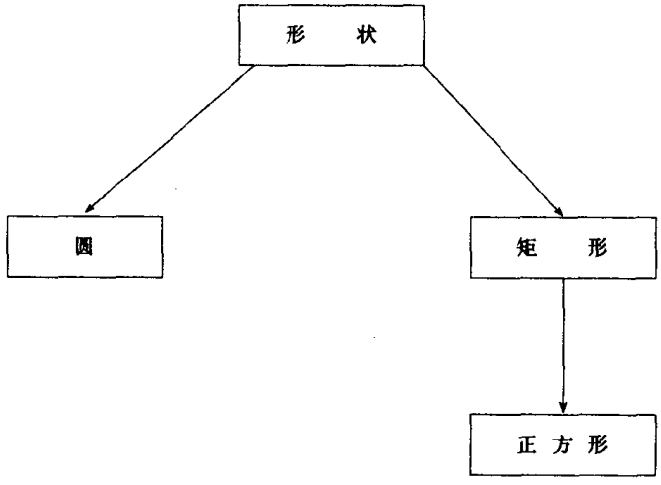
1.2.2 面向对象的程序(OOP)

基本上，面向对象主要用于现实世界的问题/解答。相同类型的对象属于一个相同的类——class。它们是实例或类的例化。对许多实际的目的，OOP 中的类可等价于结构化中的类型变量—Type。诸如，我们可以在结构化程序中定义各种类型的变量，同样也可以在 OOP 中声明类的各种对象。OOP 的一个首要特征是数据的集合，这种特征允许程序员定义自己的数据集合变量。这意味着程序员可以创造属于自己的特殊应用类型(单个变量或类变量)，它能够满足所给程序应用的需要，而且更贴近于系统内的变量。这些类型隐藏了数据和处理过程的细节，仅仅对外部世界提供对象的行为。当然，结构化语言，如 Pascal 也有创造新类型的功能，但在这些语言中缺乏灵活性，因此，所创造的类型不仅仅含有数据，而且还有方法。

1.2.3 继承性

在许多面向对象的程序中，一个类并不孤立地存在，它也是类继承的一部分，并且每一个较低级的类(一个所的类)从一个较高级类(基本类)中继承了数据与方法。

例如：



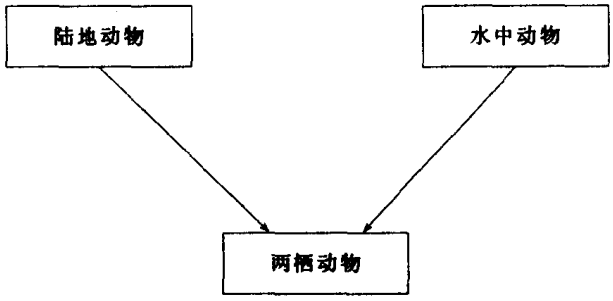
(因许多原因，多数文字显示反方向上的箭头，但作者感觉这是反方向的)

在这个例子中，圆与矩形类继承了形状类的数据和方法(并且也有它们自己特定的其他成员)，正方形直接继承矩形的成员(并且，间接地继承形状的成员)。形状是一个基本类，圆和正方形是继承类，矩形既是基本类又是继承类。

类有继承的基本原因是避免代码的重复，如果几种不同类别的对象有公共的成员，那么公共成员将被独立并放在同一个类中，那里他们仅仅声明一次。例如，在上面形状的继承中，每一个形状的参考点的坐标可作为基本类的成员，其他的类可以从基本类中继承这

些公共的成员，并不需要程序员再次声明他们。

形状例子的继承仅仅显示了单个的继承。也就是说，每一个继承类有一个间接的基本类，它也可能有多个继承，在每一个类中有多个间接的基本类。



这里，两栖动物继承了陆地和水中动物的特征(成员)。

然而，多数继承并不仅仅用于技术，也可预测以后的某些事件。

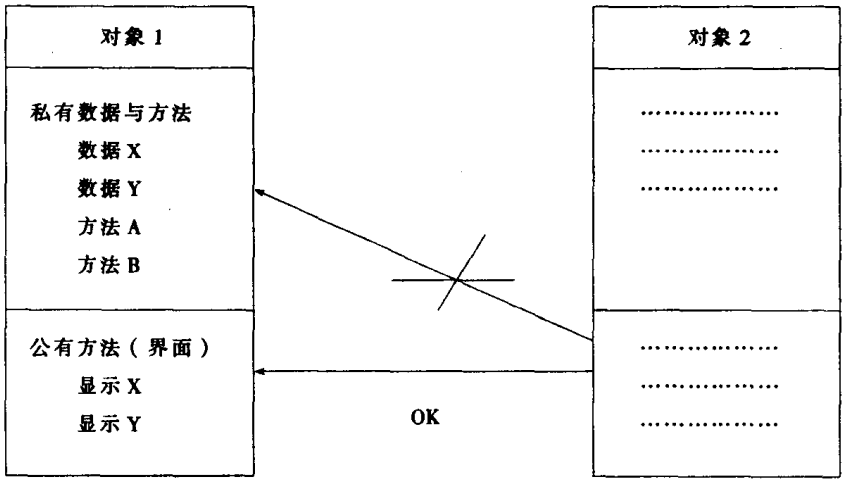
1.2.4 任务处理

套装保护了来自外部世界的对象的数据，当然对象若孤立地单独运行是无意义的。开发一个软件的目的是提供一些服务去解决问题。这不仅仅是从一个对象中读出与读进信息，他们要在软件中完成全部的工作。Grady Booch 被认为是面向对象的开发者之一，这样定义 OOP：

“程序中的执行方法均被认为对象的集体协作，他们中的每一个均代表类中的一个实例，并且他们都是通过继承关系继承了基本类中的某一员。”

两个或多个对象之间的通讯通过任务处理来完成。OOP 通过套装隐藏了来自外部程序对象的内部细节(执行过程)。然而，它通过信息处理，提供了人机交互的最小界面。一个对象的数据对于对象来说是私有的。如果其他对象想利用这些数据，他们需调用作为对象界面一部分的公共方法。界面提供了通向数据的通道，但它也可不提供通道，只对其他对象的部分数据及控制修正提供只读方法。在实际的应用中，主要依据数据的类型来定。

例如：



当收到一条信息时,所响应的方法将被执行。这里,对象 2 通过方法 X 或方法 Y 将信息发送给对象 1,因为这些方法是后边对象公共界面的一部分。然而,对象 2 不能直接采用同样的方法从对象 1 中调用其私有方法,也不能修改 X 与 Y 的数据。

一个类的界面(自以后,类的任意一个对象的界面)被类的设计者在'无须知道'的基础上被决定。外部可以合理地通过方法调用所需的数据,但不是所有。

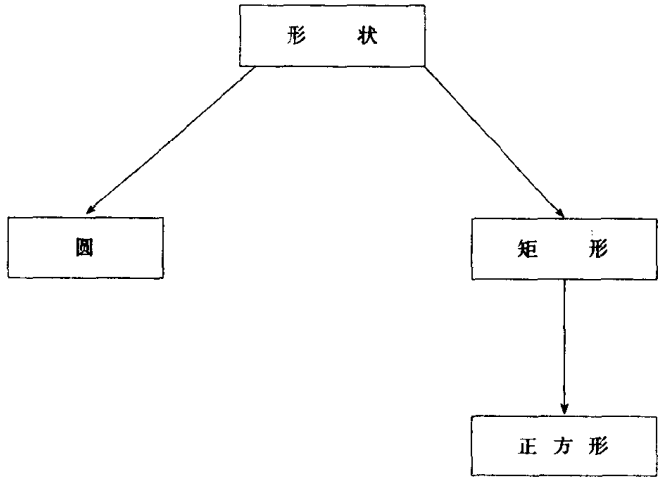
1.2.5 多态性

许多作者引用了以下面向对象的 3 个基本的概念:

- 套装性;
- 继承性;
- 多态性。

(在文献里有重要的变化,尽管一些作者以数据抽象来代替继承,但大部分的作者将继承作为完成多态性的一种方法。)

套装与继承性已经介绍,但是何谓多态性呢? 该词来源于两个希腊的单词:"polus(意味许多)"和"morphe(意味形状)"。因此,多态性意味着'采用多个形状',为了在面向对象的文本中简化'多态性'所代表的含义,可以参考我们所作的简单的形状继承的例子。



假设在"形状"类中有"画"的方法。那么在"圆"与"矩型"类中,将要继承该方法,但是在继承相同的画法任务时,将要作出不同的反应。

多态性的应用仅仅在继承遗传和允许不同的"从属类"对于相同的任务而作出不同的反应时进行应用。这是面向对象的有益的一个方面。

1.3 新方法的动机

从 60 年代中纪到现在,结构化程序在软件的开发中占据了主要的位置,结构化程序的作用是在解决所谓的"软件危机"中提供了一种方法。危机就是在软件中含有大量的预算,太多的臭虫,并且不能做用户所做的事情和难于维护。对可靠性、高质量的要求已经

呼吁了许多年。这种问题在既需要维持旧的软件系统，又要满足新的需要时所涉及到，这种结果将会把整个程序开发过程中70%的时间花在维护工作的碎片上。虽然结构化程序在软件开发的进程中起到了重要的作用，但对软件的要求已超过了该结构所起到的作用。另外，软件工程师感觉到结构化程序仍然不能满足今天许多程序开发的可靠性和维持性的需要。因此，计算机科学工作者，正在努力地寻找一种更好的方法。

1.4 什么是 OOP?

为什么面向对象的程序(OOP)，今天被许多软件开发商看作为一种结构化程序的有效方法？它的利益在于什么？主要有以下几个方面：

· 代码的可靠性

已经认识到在程序开发问题中，一次又一次的标准运算和标准数据结构所引起的问题。它由此而引起的争论，在新程序开发中每一次结构运算的调用，“重复的开发”将是非常浪费的，并且在标准库中，数据的调用是比较敏感的。OOP 提供了诸如代码的类链接库，因此降低了开发的时间并提高了软件的可靠性，这是因为代码来源于所测试与试用过的数据库。

· 更易扩展性

它可以是被标准的数据库提供，但不是全部结构或全部功能所需要的实际应用的类库，但在许多方面能更好地接近。程序员可以很自然地创建从现存的基本类中继承属于自己的类，并且可加入自己的成员。它也可以在所分的类中覆盖基本类中已存在的类，为的是优化已存在的功能。因此，程序开发者(至少在理论上)可以收获代码可靠性的益处，但是，这只能满足开发者个人的需要。

· 更易于调试

因为封套性，臭虫的原代码可被看作一个实际对象的输出，它能很快地降低所研究的领域并能缩短大量的时间。

· 更易于复杂的管理

声明这种益处来源于“分开与征服”的方法。通过继承，对单个基本对象实行逐步解决来进行。

· 更自然和直接的方案

面向对象所提供的从主要问题对象到程序对象的更直接更自然的方法是有争论的，这主要是一些人工的程序结构在许多软件开发的方法中进行对比而得出的结果。

1.5 以下几章的内容

在考虑C++面向对象的执行之前，必须对该语言基本的术语有所熟悉，这些特征基本继承了C语言。以下几章将要说明，并且O-O的执行将要持续到第七章。在这一点上，建议略过1.2和1.4节，为的是重温关于面向对象的基本规律的记忆部分。

第二章　C++程序的第一步

目的

- 使读者熟悉C++程序的基本结构。
- 为各种变量介绍基本的数据类型和语法。
- 使读者熟悉C++语言中基本的输入输出。
- 给出周期和范围概念的理解与评价。
- 介绍基本的命令符。

2.1　C++程序的基本结构

C++是一种高级语言，它较容易地进入硬件和软件的低水平的某些方面。然而，由于是一种简洁的语言，在其中能很容易地编写一些重要的并不能够读懂的程序。自然地，在程序继承上比一些冗长的语言要重要的多，特别是在变量的命名和所罗列代码的转换方面上。除O-O时间方面的考虑外，C++程序可看作一种简单的“较好的C语言”。采用时间顺序的方法，C++程序基本的组成部分(在程序中所引起的顺序)有：

- 直接处理器
- 功能的类型
- “main”功能
- 功能的模块

注意：'function'有在作为过程与功能时，与Pascal语言是相同的，在C++中并没有什么区别。

实际上，这4个组成部分在不同的文件里通常被分开，由于你看到很短的。然而，由于简单性的缘故，我们将要处理简单文件程序。

另外除了以上列出的4项外，当然任何程序都要有一定的组成部分。在C++中，解释注释符用“双斜杠”(//)。在命令的最后没有命令终止符。由于它假设运行到一行的最后。如果一些命令运行了几行，每一行要有命令符“//”。

例如：

```
//This is a C++ comment
//which runs over two lines.
```

每一个程序都要有一个所谓的功能“main”，由于系统通过调用该功能执行C++程序(然后其他的功能被调用)。在C++中，功能可以包含参数，它显示在功能mian后边的括号中。mian功能通常采用无参数的方式，因此在后面的括号中，经常显示为空的括号。以下将是围绕括号链的程序说明与语句。

```
main( )
{
```

```
.....;      //每一个说明与语句均以分号";"结束。
.....;
}
```

注意:C++(像 C,Pascal 和其他的高级语言一样)是一种块结构的语言。每一个块均被括号链所包围,并且看起来很短,也可以看作"巢"。

值得提及的是,一些程序开发者经常采用如下所示的无参数的 main 的形式。

```
main(void)
{
    .....;
    .....;
    }
```

这是 C 软件开发者经常采用的一种方法,由于通过空的括号暗示了在 C 中的任意一个参数。

有些软件开发者也采用了 main 返回整型数值的形式。

```
Int main(void)
{
  .....;
  .....;
    }
```

[返回类型的详细内容将在第四章给出。]

然而,去采用第一种方法是作者个人的习惯(也是很普遍的)。

为了简化并能够提高技能,让我们考虑一个除 main 外没有其他功能的完整例子……

```
//hellop.cpp
//显示一句对用户欢迎的词语。

#include<iostream.h>
main( )
{
  cout<<"Hello,user! \n";
    return 0;
}
```

旁白:

程序文件中的最初两行或三行的用处是文件的名字和程序目的的简短说明,这是一个较好的习惯,也是一个较好的方法。然而,作者想让读者注意的是上面例子中括号的对齐方式,每一个半括号均在它自己的上下线上。它和以下所显示的语句很容易地看出在什么地方错过了括号:

```
main(){
.....;
.....;
.....;
```