第一部分 入门编程

入门编程的学习目标在于掌握 C++程序设计的操作技能与基本编程描述技巧。

编程目标的如此定位，决定了编程学习的实践主导模式，也决定了编程能力的基本因素是技能而不仅是知识。考核编程能力的方式是一定知识背景下的上机实验，而不是以书面形式考核的、缺乏充分实践因而可能曲解的知识点。

刚开始学习编程，首先面临的是操作技能关。编程经验都来自于操作实践，没有实践基础的编程能力，从何谈起都不知道。

有了操作技能，就可以慢慢地从书本中领略所包含的知识了。操作技能应该反映在整个编程过程的实践活动中，从编辑代码到编译链接，从调试运行到代码提交，从在操作系统中对各种类型的文件创建到编程设计中对文件的输入输出等各类操作。因此，这些对于在开发环境中创建程序工程的概念，编译查错的能力，调试并发现错误的能力，包括对提交测试结果的分析能力，是一个综合的考量。

入门编程还包括学习基本的 C++程序设计的表达能力，即将算法思想转换成程序代码的能力。为了能准确、高效地描述算法思想，有必要学习诸多的编程描述技巧。其中包括对输入输出的文件操作，熟悉语言中的各种循环控制的方法和结构，从而深入理解循环的控制能力。同时学会函数的声明、定义和调用，递归函数的调用方法，直至代码优化的各种方法。编程技巧还包括学习能够降低循环嵌套层数的重复字串表示，充分利用整型数的特征，进行二进制转换和位操作带来的代码优化，浮点数的比较技巧，条件表达式和条件语句的适时转换，条件表达式中逻辑短路语句的表达技巧和使用经验，以及一些 STL 的算法和容器使用方法，慢慢地引导读者权衡不同解题方法的性能，以展开比较有深度的编程分析与逻辑描述的思考。

第一部分的学习，是学习操作技能和算法思想到程序代码的转换技能，没有这部分的学习，而直接进入对解决问题能力的追求和对运行性能的追求都是一句空话。因而第一部分的学习和领悟是展开第二部分学习的必要条件。

第一部分是能否学到 C++编程要领的关键。如果第一部分的知识点没有掌握，宁可重学，也千万不要跟读后面的内容。因为没有语言表达能力和编程验证的方法，就没有自我学习与研究的能力，即使看了许多编程方法与算法的书，独立进行了许多编程逻辑思考，但终究没有实践的经验，以致到最后论及编程能力，一无所有。

第1章　实验概述

1.1　实验目标

❑ 总体目标

掌握 C++程序设计的基本操作技能，实践程序编译、运行与调试

（1）重点掌握各种内部数据类型，数值与逻辑运算，各种表达式，函数的声明、定义及调用。

（2）掌握过程控制编程方法，正确编制多重循环过程，对简单问题能够临场加以解决。

（3）学会使用简单的 C++ 标准库。

（4）了解程序质量的相关要素，对可移植性、可维护性、可扩展性、易读性、正确性、健壮性以及时间与空间效率有初步的认识。

（5）养成良好的编程习惯，形成自己的编程风格。

❑ 具体目标

（1）掌握操作技能。学会启动编程工具，创建 console 项目，设置路径，添加程序文件，编辑文件内容，更换程序文件，建立数据文件，存储 console 项目。

（2）掌握编译链接技能。学会对程序文件进行编译，对程序进行链接，生成可执行文件，获得查找和纠正编译与链接错误的基本能力。

（3）逐步积累调试技能。能够在调试环境和 Windows 命令提示符环境下运行程序；能够在调试环境设置运行断点，单步运行程序代码，在运行中查看变量值，中止运行，查看运行结果；学会判断运行出错的位置，了解错误原因。

（4）学会控制语句的描述与使用。掌握单重循环及多重循环控制，学会处理简单的逻辑及计算问题，体会不同语句表达同一语义之间的差别，学会构造语句块、条件与赋值表达式。

（5）学习标准流、文件流及字串流操作。能够打开、读写文件，从标准流、文件流及字串流中读入各种基本类型的数据，能进行流结束的判断与控制，能进行标准流输入的终止操作。

（6）学习将代码拆分成若干函数。掌握函数的声明、定义和调用方法，正确描述参数和传递参数，正确描述返回类型和书写返回语句。

（7）初步学习分析问题的方法。能够从问题描述以及从样本输入数据与样本输出数据的关系中了解问题，能够仔细理解问题描述，并能分析测试数据的表示范围。

❑ 几个要点

（1）如何在 Borland C++ Builder 6 中创建一个 console 项目，并且正确设置路径。创建过程总是需要与 Windows 操作达成默契，即在 Windows 中创建一个自己的文件夹，以存放自己的程序代码和数据文件（☞附录 A“实验操作指南”）。

（2）在进入 C++ Builder 的代码编辑窗口后，如何输入程序代码。或许最开始是输入一些样本代码，但要逐步了解一些快捷键的功能，如 Ctrl+Y 是删除当前行，Ctrl+Z 是撤销当前操作等。

（3）在代码编写中学习如何读取数据文件中的数据和判断数据读取结束状态，同一处理目标可以由不同的代码实现。初学者应先了解各种表达方法，再模仿比较简洁高效的方法。

（4）学习解决问题的步骤，即从分析着手，先思考算法，再编写代码，然后进入编译调试的过程。在熟悉简单的编程方法后，可以直接在编写代码过程中表达算法意图，即合并思考算法和编写代码过程。

（5）代码编写完成后，即可进行编译和连接。编译中会出现一些意想不到的错误，读者需要根据编译错误信息的提示将错误逐一找出。要了解到虽然有编译信息提示，但编译系统并不足够智能，它无法了解程序员的编码目的，对于代码错误，它只是按照语言的语法机械地列举其不符规则的错误事实。因而，查找错误需要靠一定程度的经验来判断，错误查找能力与经验的积累成正比。对初学者来说，查错是一个重要的学习内容。

（6）编译完成后，即可进入运行调试阶段。在运行中很可能会发生运行异常中止或运行结果与期望不符的情况。查找运行中出现错误的位置或原因，并且修改代码直至正确的过程，称为调试。修改运行错误远比修改编译错误困难得多，因为运行中出现的运行异常或结果不符都是对程序代码整体而言的，在编译阶段尚有出错提示信息，而运行阶段则没有任何出错提示。初学者往往会对产生的错误，特别是对运行异常的错误无从下手，建议从查找出错的位置着手。读者在学习初步的编程方法之后，如何真正提高编程技能，则与调试经验密切相关。程序员很大程度上是在不断的代码调试过程中成长的。

（7）经常会发生这样的情况，针对样本数据的程序运行得到了正确结果，但提交之后被判定为错误，这说明代码只能适合其数据范围的一个子集，并不能满足包括边界值在内的测试数据集合，代码中还存在潜在的错误。这同样属于调试过程，但发现与改正错误的过程更为艰难，它与正确理解题意和测试边界数据有关。

1.2 实验环境

❑ 单机实验环境

1 操作系统环境

对个人单机的配置没有严格的要求，只要可以访问外网，可以运行 C++调试环境即可。大多数人可能在 Windows 环境下安装 C++开发环境。

2 C++环境

在 Windows 环境下，常用的软件有 Microsoft Visual C++ 6（简称 VC6）、Borland C++ Builder 6（简称 BCB6）、Dev-C++ 4（采用 G++编译器）。如果安装了微软的 .NET 套件，则支持 Visual C++ .NET 的编译器（VC7 或 VC8）。

从初学者的角度来选择 C++的工具，其标准 C++实现的程度、系统使用的方便与简洁性、Help 功能的易用性、调试功能的强大与否，都是选择使用的参考依据。表 1-1 列出了几种 C++工具的比较情况。

表 1-1　几种 C++工具对初学者的适合程度[①]

工具	标准 C++	Help 功能	调试功能	人气	方便性	适合初学者
VC6	★★	★★★	★★★	★★★★	★★★★	★★★
BCB6	★★★★	★★★★	★★★★	★★★	★★★★	★★★★
Dev-C++ 4	★★★★	★★	★★	★★	★★★	★★★★
VC++ .NET	★★★★	★★★	★★★	★	★★	★★

从表 1-1 中可以看出，当前使用各 C++工具的人气与该工具使用的方便性密切相关，相对来说，BCB6 软件规模比较庞大，甚至初学者安装成功率都比较低，但仍然具有相当的人气。

VC6 最大的影响因素也许是它的用户群，许多高校都以该软件作为学习 C++的工具，许多公司和科研机构也用它开发软件。它使用比较方便，性能也比较好，再加上得天独厚的微软 Windows 环境，在学习 C++之后，立即可以用它进行应用开发。

Dev-C++ 4 的最大特点是软件共享性，下载和安装都很方便，而且它的标准 C++程度较高，操作界面比较方便。

BCB6 最大的特点是标准 C++程度相对较高，调试功能强大，而且 Help 功能比较好。对于初学者来说，学习比较方便。

VC++ .NET 充分采用了标准 C++，但是安装环境或使用学习方面稍微复杂一些，Help 功能比较专业化，不太适合初学者学习。

综合上述描述，适合初学者的工具应以 Dev-C++和 BCB6 两款软件为好。本书附录中介绍了 BCB6 的使用方法，列出了其操作说明和错误信息，同时也在清华大学出版社网站上提供了用 VC6 解答的代码，读者可比较与标准 C++程序的差异，对于其他 C++编译器也有借鉴作用。

1.3　实验安排

❑ 第一套实验

1 实验准备

简单阅读主教材《C++程序设计教程》（第 3 版）第 1 章。

① 各项指标的评价来自作者的使用经验，仅供参考。

学习使用 C++工具 BCB6 或其他工具。

2 做第一部分的第一套实验

第一套实验共 6 个问题。

如果不会操作，则回到实验准备阶段，重新学习 C++工具或 OPS 系统。

如果不会编写代码，则阅读第 4.1 节“第一套实验”的解题指导，并仔细阅读第 1.4 节的做题步骤。

如果还不会编写代码或操作 C++环境，则停止“学习”过程，检查自己的机器环境是否完好，或反思自己的信心是否足备。

初学者在没有任何编程经验的前提下，一味看书已被证实并没有益处，此时，与人沟通是第一要务。

❑ 第二～五套实验

1 实验准备

阅读主教材《C++程序设计教程》（第 3 版）第 1～4 章的内容，了解大致的解题方法。

阅读附录 B“BCB6 常见编译错误”，了解编译错误信息的表示方式，便于实验过程中查阅。

学习第 1.4 节的实验过程，了解实验过程中的调试方法，便于模仿学习。

2 做第二套实验～第五套实验

如果不会解决问题，则阅读第 4.2 节的第二套实验的解题指导及之后的诸解题指导，学习思路并模仿代码。

如果不会操作，则回到第一套实验。

如果看到错误但不会修正，则回到实验准备阶段，求助同学和老师，学习查找编译错误和调试方法。

初学者主要学习的是调试经验，而调试经验是一个缓慢的积累过程。明白编程的难学之处在于漫长的调试经验的积累过程。只有积累了调试经验，算法思想才能开始付诸实施，编程便开始有了快乐，这是编程学习的一大秘诀。

1.4 做题步骤

❑ 分析理解

（1）推算样本输入与输出数据之间的关系（确认对题意的正确理解）。
（2）确定输入数据的读入单位（每组数据的类型与个数）。
（3）确定每组输入数据与计算过程的对应关系。
（4）确定输入、处理和输出的数据存储方案。
（5）分析数学计算过程（算法、表达式、流程图等）。
（6）将数学计算过程转换成程序代码。
（7）确定输入数据的结束条件和控制实现。
（8）构造输入过程、处理过程和输出过程，很多时候这三个过程可以合并。

（9）分析数据表示范围，有些范围是题目中明确的，有些范围是根据逻辑关系推断的。

（10）设计若干组不同于样本输入的边界数据以帮助调试。

（11）分析输出格式的实现方案。

❑ 算法描述

算法是对整个程序运行的操作顺序进行的结构性描述，编程则是将算法具体实现为能够运行的程序代码。

算法的文字描述是按有序的步骤描述操作过程，其中的步骤可以转移到其他的步骤，形成控制结构，也可以转移到另一个独立的算法文字描述，形成过程调用。

算法中的结构控制是通过转移到其他的步骤来完成的，类似程序中的 goto 语句，因而比较原始。为了使算法描述更具结构性，一般的算法往往采用某种编程语言来描述。在这里，因为算法文字描述的过渡性（以后便用 C++语言直接在代码中描述），我们仍然采用原始的方式。例如，下列分别表示有循环结构和无循环结构的不同算法文字描述。无循环的算法描述将读入数据按奇偶性打印；有循环的算法描述 1~n 的逐项求和，并打印结果的过程。

1 奇偶判断算法

第一步：创建一个输入变量 n。

第二步：输入一个整数 n。

第三步：判断 n 是否为偶数。

第四步：若 n 为偶数，则输出“偶数”并转到第六步。

第五步：若 n 为奇数，则输出“奇数”。

第六步：结束。

2 逐项求和算法

第一步：创建一个输入变量 n。

第二步：输入一个整数 n。

第三步：定义变量 sum 初值为 0，定义循环变量 i 初值为 1。

第四步：判断 i 是否等于 n。

第五步：若 i 不为 n，则将 i 值加到 sum 中，然后将 i 加 1 并转到第四步。

第六步：若 i 为 n，则输出 sum。

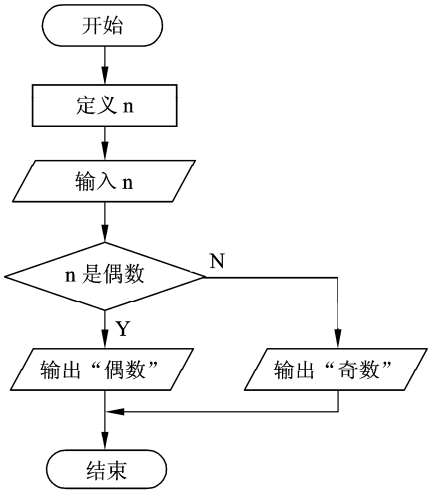
第七步：结束。

算法的流程图描述是用一些图形元素的组合来形象地描述动作序列的过程。通过箭头可以看清控制的流动，通过各种图形框可以区分各种不同性质的操作。例如，图 1-1 分别表示了上面的两个算法。

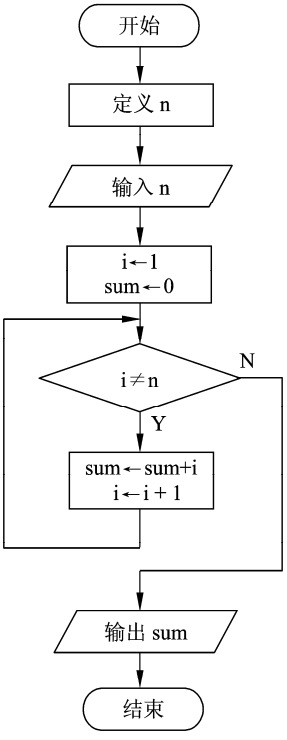
在图 1-1 中，椭圆形框表示开始和结束；平行四边形框表示输入与输出操作；菱形框表示条件判断操作；矩形框表示一般的处理。操作顺序循着箭头所指的方向流动，遇到菱形框时，流动的方向取决条件判断的真或假，在菱形框的两个出口处标有真或假的记号，以明示其出路。

算法的文字描述和流程图描述可以一一对应，它们又都可以用某种编程语言来描述，用编程语言描述比用文字描述更能体现其结构。算法的文字描述比较细密周到，而流程图描述比较直观。有的人喜欢用文字描述，有的人喜欢用流程图描述，因人而异。不管哪一

种描述都是学习编程的一个过渡，因为编程是实战，可以切实地表达算法的实现性，同时，语言可以抽象也可以具体，因而可以作为展开算法描述的较好手段。



(a) 奇偶判断算法



(b) 逐项求和算法

图 1-1　两个算法的流程

❑ 代码风格

代码描述了呈结构性的语句集合。编写代码除了把关键算法思想描述清楚外，还需要可运行性，即代码的完整性。代码需要包含必要的资源，任何对实体的声明或定义、表达式、过程调用，都必须严格符合语言的语法，保证其在编译和链接后能够正确运行，这也是代码与算法描述的区别所在。

代码除了正确性，还体现了一定的风格。因为语句的结构性描述完全可以因人而异。通俗地说，代码风格就是代码中体现的好看和易懂的个性。代码风格也是代码质量的衡量标准之一。程序员为了提升代码质量，也在努力提高代码的可读性，在编程路途中慢慢地形成自己独特的代码书写风格。程序员可以通过使用不同的语句描述同样的算法，通过独特的书写格式，简洁明快地表达代码的意图。

（1）代码就像作文，作文有标题、作者、时间和地点，代码也有相应的代码标题（反映其功能），正规的软件开发中也一定附带代码的编写者和编写时间、版本等信息。所以初级代码至少应包含标题信息。

（2）初级代码中以函数为块单位进行描述，应有区分函数之间的明显界限。

（3）块可以嵌套，块中语句也可以包含语句。为了体现其结构性，块与子块，语句与子句应该有区分，通常的做法是锯齿形代码。

（4）在算法描述的理解困难之处应有简短的代码注释。

（5）代码不能因太松散而影响可读性。

作者在本书中采用初级代码的风格，每个完整的代码头上都有一个标题，并用注释线区分代码头上说明部分和实际执行部分以及区分各个函数块。由于注释线能够明显区分各函数块，所以作者便弱化了块分隔符（花括号对）的界限作用，总是将左花括号放到函数声明头的最右端或循环结构体说明的最右端，以体现语句紧凑的效果。

同时，尽量多用 for 循环结构，少用 while 结构，也是增强代码结构性、简化代码、又不失性能的良好风格。例如，图 1-2 选用了主教材第 6 章中的 f0618.cpp 代码示例加以说明。

```
//==================================
// f0618.cpp
// 求 1~1000000 内的素数个数                    //代码标题
//==================================
#include<iostream>
#include<cmath>                                 //资源说明
using namespace std;
//---------------------------------              注释线作分隔
bool isPrime(int n){
  int sqtn=sqrt(n*1.0);  //平方根作优化          行末短注释
  for(int i=2; i<=sqtn; ++i)
    if(n%i==0) return false;
  return true;                                  //函数块
}//---------------------------------             左花括号右置
int main(){
  int num=0;                                    //主函数块
  for(int i=2; i<=1000000; ++i)
    if(isPrime(i))                              锯齿形语句
      num++;
  cout<<num<<endl;
}//==================================
```

图 1-2　代码风格样本

当把代码输入编辑窗口后，语言中的关键字（如类型名 int）就会自动加黑，这对于防止编辑中的拼写错误很有好处，所以，编辑代码最好在开发软件的集成环境中进行，而尽量不在文本编辑软件之类的窗口中进行。

按一定的风格书写，则可以避免代码的一些结构性错误（如花括号配对、循环的包含关系等），而且各功能块之间区分明显，即使发生编译和调试错误，检查和跟踪也会比较容易。

❑ 代码编译

编译错误发生时，会给出一些错误信息，根据这些错误信息，可以查找发生错误的原因。但是，编译的错误信息大多数不是直接指示错误的位置，而是“拐弯抹角”地道出因违反哪一款语法规则而报错。因此，作为初学者，要学会查找真正的错误原因。下面列举一些作者常遇到的编译错误的情况。

（1）错误信息显示的错误位置不一定为错误发生位置。例如：

```
void f(){
  int a
  int b=a;    //E2141: 声明格式错。但显然是因上句漏了分号引起出错
}
```

C++语法以分号为语句分隔符，而回车只是一个空格、一个词法分隔符而已，所以没有分号的行，与下一行视同为一条语句。这是C++继承了C的语句格式特征，给程序员带来了很大的风格变易余地，也带来了富有创意的表达灵活性，但对初学者来说，只有适应了它才能成为好事。

注意：对待编译错误，应根据行号找到标志错误的位置，先辨认该行语句是否有错，不要认为该语句一定有错，在上下句中观察一下，特别是语句格式错误，更是与上下文有关。

（2）编译发现多个错误，但并不一定存在多个错误。例如：

```
void f(){
  in a;          //E2451: 不认识名字in;   E2379: 名字in后漏了字母t
  int b=a;       //E2451: 不认识名字a。  上条语句错误，连累了本语句中的a
}
```

编译在发现错误以后，即不让该语句被正常编译，导致后续依赖于该语句的语句产生错误。编译显示三个错误，即关键字int漏了字母t。

注意：针对编译显示的多个错误，找到和改正一个错误后，不要继续找第二个错误，而是重新编译，在重新编译的基础上再查找新的错误。

（3）编译可能会针对同一处显示多个错误。例如：

```
void f(){
  for ; ; )    //E2376: 语句缺左括号;  E2188: 表达式错误
    int a=0;
}
```

编译器针对同一个位置的语句，可能同时套上多条语法规则错误，因为前一个错误导致对本语法单位的编译中止，而使本语法单位作为整体又套上另一个语法错误，结果便在同一条语句显示多个错误。

注意：与上一条策略相似，看见编译器显示许多错误，无须着急，当一个错误被纠正之后，也许所有的错误就都消失了，也许又会发现从没有出现过的编译错误。

（4）编译只显示一个错误，其实可能暗藏着其他还没有被发现的错误。例如：

```
//#define C++
#ifndef C++
#error Non-C++ error     //F1003: 编译根据指示，产生一个致命错误
#endif
```

F开头的错误号为致命错误。编译器遇到致命错误，便会立即停止编译，后面的可能错误因而就未被检查和发现。在上例程序员编写的代码中，在发现没有定义C++（#define C++）这个名字的前提下，会人为产生一个致命错误以阻止继续编译。例如，本代码是用C编写

的，但编译开关却设置成按C++编译，于是即使编译了代码，可能也是错的。尽早发现，给个致命错误，中止编译了事。

注意：遇到致命错误的情况，由于编译强行中止，无法知道本代码的其他错误，只有在改正致命错误以后，重新编译，才会正常编译代码的其他部分。

（5）警告可能是不能忽略的错误，也可能是可不予理睬的错误。例如：

```
void g(){
  const int a=3;
  int b=7/(a-3);  //W8082: 除数为0
}
```

由于a为常量，编译会对全部包含常量和字面值的表达式进行计算以优化运行过程，结果发现一个除数为0的错误。但是从语法上，不能说这是一个错误，合法而不合理的事在我们周围到处都是，编译器对其只能是警告。但是，从代码编写的角度来看，编译器确实帮助我们发现了一个编程错误。

又例如：

```
#include<vector>
void f(){
  std::vector<int> a(3);
  for(int i=0; i<a.size(); ++i)  //W8012: 有符号数与无符号数进行比较
    a[i] = i+1;
}
```

变量i是有符号整数，a.size()是无符号整数，因此两种类型的数做一个操作（i<a.size()），逻辑上带有某种不安全性。在计算机内部，负数也是作为二进制数来描述的，当有符号与无符号数进行比较时，系统先将有符号数转换成无符号数，再进行比较。结果–1转换成无符号数，就会大于整数5。因此，编译对于有潜在错误的操作，会人性化地给出警告。

对于本例来说，代码书写时已经限制了变量i的初值为0，以后只做适当次数的++操作，因此它的符号与a.size()是相同符号的，可以排除不安全性，因此该警告可以忽略。

注意：遇到警告，必须小心，应该以对待编译错误的心态来对待警告，必须知道为什么警告。

附录B给出了BCB6常见编译错误示例，可以把它当作学习和查找编译错误的案头参考，对于不同的编译器也有参考价值。很多查找编译错误的经验，是要在编程中慢慢积累的，因为编译并不仅是封闭在一个调试平台内的，它还受到操作系统甚至网络环境的制约，千差万别的计算机环境配置也会影响编译的状态。读者提高了编译查错技能后，其编程总体能力也会有所提高。

❑ 代码调试

1 建立数据文件

（1）数据文件的作用。

代码经过编译和链接后，便是一个可以运行的合法程序了。但是合法并不等于正确，必须给予一些数据，让其运行，通过运行结果来判定其正确性。