青少年信息学竞赛

主　编　刘　洪
副主编　杨　娟　徐　勇

清華大學出版社
北　京

内 容 简 介

本书面向零基础的信息学竞赛初学者。全书共 6 章，主要讲解 C++编程语言基础和算法知识。第 1 章 C++语言基础，主要讲解数据类型、变量、常量、数据溢出、数据的输入和输出等；第 2 章程序设计结构，主要讲解顺序、分支和循环 3 大结构；第 3 章数组和字符串，主要讲解 C++的数组基础知识及字符串应用；第 4 章函数和结构体，主要讲解自定义函数的使用、结构体的定义和应用；第 5 章基础算法，主要讲解算法的描述方法，以及入门算法、递推和递归算法、排序算法和数值处理方法；第 6 章进阶算法，主要讲解查找算法中的顺序查找和二分查找，搜索算法中的深度优先搜索和广度优先搜索，贪心策略的应用，动态规划方法的应用。

本书内容通俗易懂，通过详尽的知识点和算法讲解，帮助初学者掌握信息学竞赛的基础知识和常用解题方法，形成编程思维和计算思维。本书可作为师范类院校编程专业的教学用书，也可以作为中小学信息技术领域教师从事编程教学的培训用书和信息学竞赛初学者的入门教材。

图书在版编目(CIP)数据

青少年信息学竞赛 / 刘洪主编. —北京：清华大学出版社，2022.8
ISBN 978-7-302-61270-4

Ⅰ. ①青… Ⅱ. ①刘… Ⅲ. ①程序设计－青少年读物 Ⅳ. ①TP311.1-49

中国版本图书馆 CIP 数据核字(2022)第 121276 号

责任编辑：王 定
封面设计：周晓亮
版式设计：思创景点
责任校对：成凤进
责任印制：丛怀宇

出版发行：清华大学出版社
网 址：http://www.tup.com.cn，http://www.wqbook.com
地 址：北京清华大学学研大厦 A 座 邮 编：100084
社 总 机：010-83470000 邮 购：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质 量 反 馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 装 者：三河市金元印装有限公司
经 销：全国新华书店
开 本：185mm×260mm **印 张**：18.25 **字 数**：433 千字
版 次：2022 年 9 月第 1 版 **印 次**：2022 年 9 月第 1 次印刷
定 价：69.80 元

产品编号：094718-01

前言

PREFACE

随着科学技术的发展，信息技术与人们的社会生活已经深度融合，物联网、云技术、人工智能等技术给人类社会带来了翻天覆地的变化，同时也带来了严峻的挑战。

信息学竞赛以算法和数据结构为核心，要求学生能运用数学知识构建模型，并采用计算机程序设计语言编写程序来解决实际问题。其主要内容包括计算机基础和编程语言基础、枚举算法、模拟问题求解、递推算法、递归算法、回溯算法、排序算法、高精度数值处理算法、查找算法、搜索算法、贪心策略、动态规划等。

本书面向零基础的信息学竞赛初学者，详尽讲解了程序的运行过程和算法的基础思想，帮助初学者完成从 0 到 1 的过程；以历年信息学竞赛真题为例，帮助初学者提升编程能力，使初学者形成计算思维，培养初学者良好的编程习惯，为后续进阶学习奠定扎实的基础。全书共分为 6 章，第 1 章为 C++语言基础，主要讲解数据类型、变量、常量、数据溢出、数据的输入和输出等；第 2 章为程序设计结构，主要讲解顺序、分支和循环 3 大结构；第 3 章为数组和字符串，主要讲解 C++的数组基础及字符串应用；第 4 章为函数和结构体，主要讲解自定义函数的使用、结构体的定义和应用；第 5 章为基础算法，主要讲解算法的描述方法，以及入门算法、递推和递归算法、排序算法和数值处理方法；第 6 章为进阶算法，主要讲解查找算法中的顺序查找和二分查找，搜索算法中的深度优先搜索和广度优先搜索，贪心策略的应用，动态规划方法的应用。

本书语言通俗，通过详尽的知识点和算法讲解，帮助初学者掌握信息学竞赛的基础知识和常用解题方法。

本书可作为师范类院校编程专业的教学用书，也可以作为中小学信息技术教师从事编程教育的培训用书，以及信息学竞赛初学者的入门教材。

由于编者水平有限，书中难免存在不足之处，敬请各位同行和读者批评、斧正。

本书免费提供教学大纲、教学课件、案例源代码、微课视频及思考练习参考答案，读者可扫下列二维码获取或学习。


教学大纲


教学课件


案例源代码


微课视频


思考练习
参考答案

编　者

2022 年 5 月

目录

CONTENTS

第1章

C++语言基础

日常生活中的语言让人类可以互相交流信息。人和计算机的交流则通过编程语言实现，常用编程语言有 BASIC、PASCAL、C、C++、Java、Python、PHP、JavaScript 等。目前在信息学竞赛中主要使用的编程语言是 C++。C++语言完全兼容 C 语言，并且 C++语言可以使用 STL（standard template library，标准模板库），可以极大地提高编程效率。此外，大多数的算法竞赛支持使用 C++语言提交代码。

1.1 编程语言

使用编程语言编写的程序在计算机中有两种运行方式，一种方式称为**解释**，就是计算机对程序的指令翻译一句执行一句，Basic 和 Python 都是典型的解释型语言。还有一种方式称为**编译**，计算机将程序的指令一次性全部翻译后，再让计算机执行。C 语言、C++语言都是编译型语言。Java 语言比较特殊，它是先编译为字节码文件，再按照解释方式执行。

1.1.1 集成开发环境

C++语言属于编译型语言，编写的程序需要先编译，再运行。C++语言使用 G++编译器完成编译过程。如果程序代码有修改，则编译器需要对代码重新编译，这个过程在编写调试代码中非常频繁。为了简化代码编辑和编译工作，很多编程语言有对应的编写开发软件工具，这种软件就称为集成开发环境（integrated development environment，IDE）。

本书采用信息学竞赛中使用的 Dev-C++作为 IDE。

注意

编程时输入的所有字符都需要在英文状态下输入，中文输入状态下的分号“；”和英文输入状态下的分号“;”是不一样的。

1.1.2 C++语言的基本结构

【例 1-1】C++语言的基本结构如下。

```
#include <iostream>
using namespace std;
int main() {
    cout << "Hello,World!" << endl;
    return 0;
}
```

第 1 行的#include <iostream>称为头文件，用于在当前程序中引用其他程序，第一个单词 include 的含义为包括。根据程序的需要，引入的头文件会有所不同。如果程序需要使用和数学有关的功能，则需要使用#include <cmath>引入数学的头文件。

此外，C++语言可以使用#include <bits/stdc++.h>引入所有的头文件。这个引用也称万能引用。

第 2 行的 using namespace std;指明程序的命名空间。命名空间用于在编写大型程序时解决名称冲突问题，如四年级 1 班和 2 班都有学生叫张三，如果只说四年级的张三，就无法区分这两位学生，具体到班，就可以区别这两位同名的学生了。

using namespace std;表示当前采用的是 std 命名空间。std 是英语 standard 的简写，含义为标准。

第 3 行的 main()是一个函数，称为主函数，这个函数可以包含很多语句，放在函数后面的{}中。在一个 C++程序中 main()函数有且只有一个，它是 C++程序的入口。

第 4 行的 cout << "Hello,World!" << endl;是一个输出语句。作用是将中间双引号中的内容输出到屏幕。

第 5 行的 return 0;表示主函数会返回一个 0，这个 0 表示程序运行正常。千万不要修改这个 0 值，其他非 0 值会导致系统误认为这个程序运行不正常，被认为是运行错误。

注意

每个语句的结束位置，都有一个分号。

在 IDE 中输入这个程序后，选择“运行”→“编译运行”命令，由 IDE 先编译然后运行。运行结果如图 1-1 所示。

图 1-1　例 1-1 的运行结果

这个窗口的标题 D:\TEST\hello.exe 表示当前这个程序所在的位置。源程序名称 hello.cpp 在目录 D:\TEST 下，IDE 编译后自动生成 hello.exe 文件，窗口中显示的就是可执行文件 hello.exe 的执行结果。

本书后续的程序运行结果都简化为如下所示，省略横线及后面的运行时间和返回值。

```
Hello,World!
```

在 C++语言的书写中，编程者可以加入自己的注释，方便下次阅读时理解，或者给其他编程者提供参考。

C++语言有两种注释方式，一种是在行首添加两个斜杠“//”，还有一种就是以“/*”开头、以“*/”结尾，把整个注释的内容包含在内。

1.1.3　调试程序

C++语言编写后，如果代码中有错误，就需要通过调试寻找并修正错误。例如，在输入过程中，将输出语句 cout 误写为 cont，在运行时就会出现错误提示，如图 1-2 所示。

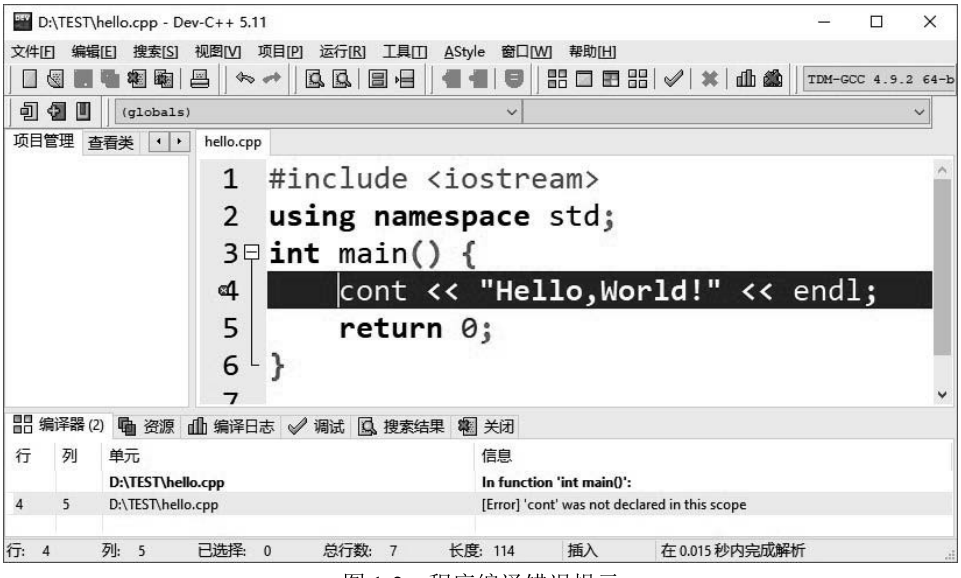

图 1-2　程序编译错误提示

在“编译器”窗口中，显示调试信息提示。其中，[Error]表示有一个错误，后面的内

容“'cont' was not declared in this scope”，意为 cont 被识别为一个变量名，并提示这个变量名还没有声明。其实，这个 cont 只是输入错误，修改为 cout 后，就可以运行了。

调试程序的过程需要编程者综合分析调试信息和源代码，判断出错的原因并修正源代码。

1.2 数据类型和运算

使用编程语言进行编程时，需要存储各种信息。不同类型的数据在内存的存储空间和存储方式各不相同。

1.2.1 常用数据类型

C++语言中常用的数据类型有布尔型、字符型、整型和浮点型，如图 1-3 所示。

（1）bool，布尔型，存放两种值：true 或 false。

（2）char，字符型，存放一个字符，如'a'。

（3）int、long long 都是整型，可以存放整数，如数字 12。

（4）float、double 和 long double 都是浮点型，其中 float 为单精度浮点型，有效位数为 8 位，如 3.402823；double 为双精度浮点型，有效位数为 16 位，如 1.797693134862318；long double 为高双精度浮点型，有效位数为 18 位。

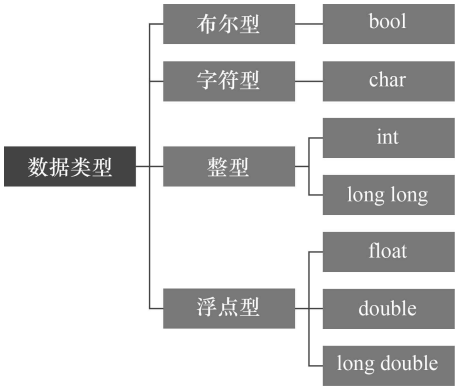

图 1-3　C++语言中常用的数据类型

注意

在不同的 C++语言编译器中，各数据类型的有效位数略有不同。

【例 1-2】查看常用数据类型占用存储空间的字节数。

```
#include <iostream>
using namespace std;
```

```
int main() {
    cout << "sizeof(bool)=" << sizeof(bool) << endl;
    cout << "sizeof(char)=" << sizeof(char) << endl;
    cout << "sizeof(int)=" << sizeof(int) << endl;
    cout << "sizeof(long long)=" << sizeof(long long) << endl;
    cout << "sizeof(float)=" << sizeof(float) << endl;
    cout << "sizeof(double)=" << sizeof(double) << endl;
    cout << "sizeof(long double)=" << sizeof(long double) << endl;
    return 0;
}
```

运行结果如下：

```
sizeof(bool)=1
sizeof(char)=1
sizeof(int)=4
sizeof(long long)=8
sizeof(float)=4
sizeof(double)=8
sizeof(long double)=16
```

【例 1-3】查看常用数据类型的最大值和最小值。

```
#include<iostream>
#include <limits>
using namespace std;
int main() {
    cout << "bool: 最小值：" << (numeric_limits<bool>::min)() << "\t 最大值：" <<
(numeric_limits<bool>::max)() << endl;
    cout << "char: 最小值：" << (int)(numeric_limits<char>::min)() << "\t 最大值：" <<
(int)(numeric_limits<char>::max)() << endl;
    cout << "int: 最小值：" << (numeric_limits<int>::min)() << "\t 最大值：" <<
(numeric_limits<int>::max)() << endl;
    cout << "long long: 最小值：" << (numeric_limits<long long>::min)() << "\t 最大值：
" << (numeric_limits<long long>::max)() << endl;
    cout << "float: 最小值：" << (numeric_limits<float>::min)() << "\t 最大值：" <<
(numeric_limits<float>::max)() << endl;
    cout << "double: 最小值：" << (numeric_limits<double>::min)() << "\t 最大值：" <<
(numeric_limits<double>::max)() << endl;
    cout << "long double: 最小值：" << (numeric_limits<long double>::min)() << "\t 最
大值：" << (numeric_limits<long double>::max)() << endl;
    return 0;
}
```

运行结果如下：

```
bool: 最小值：0 最大值：1
char: 最小值：-128      最大值：127
int: 最小值：-2147483648        最大值：2147483647
long long: 最小值：-9223372036854775808 最大值：9223372036854775807
float: 最小值：1.17549e-038     最大值：3.40282e+038
double: 最小值：2.22507e-308    最大值：1.79769e+308
long double: 最小值：3.3621e-4932       最大值：1.18973e+4932
```

常用数据类型占用存储空间及取值范围如表 1-1 所示，在编程时，需要根据数据的范围选择适合的数据类型。

表 1-1　常用数据类型占用存储空间及取值范围

数据类型	占用存储空间	取值范围
bool	1 字节，8 位	true 或 false，也可记为 1 或 0
char	1 字节，8 位	−128～127，也可记为-2^7～2^7-1
int	4 字节，32 位	−2147483648～2147483647，也可记为-2^{31}～$2^{31}-1$
long long	8 字节，64 位	−9223372036854775808～9223372036854775807 也可记为-2^{63}～$2^{63}-1$
float	4 字节，32 位	1.17549e−038～3.40282e+038
double	8 字节，64 位	2.22507e−308～1.79769e+308
long double	16 字节，128 位	3.3621e−4932～1.18973e+4932

例如，在统计我国全国人口数时，可以使用 int 数据类型吗？

int 的最大值是 2147483647，超过了我国现有人口数（14 亿+），所以可以使用 int 数据类型统计全国人口数。

再思考，可以使用 int 数据类型统计全球人口数吗？

截至 2022 年 2 月 16 日，统计到的全球人口数为 7596934179 人，这个数字已经超出了 int 数据类型的数据范围，这时就应该使用 long long 数据类型。

由于 $2147483647=2^{31}-1$，int 数据类型的数据范围也有两种表示方式：−2147483648～2147483647 或-2^{31}～$2^{31}-1$。

单精度浮点数 float 的数据范围表示为 1.17549e−038～3.40282e+038，这种表示方法称为科学计数法，1.17549e−038 意为 1.17549×10^{-38}，3.40282e+038 意为 3.40282×10^{38}。

1.2.2　整数运算

【例 1-4】把 20 支铅笔平均分给 8 位学生，每人分得几支，还剩几支？

编写一个程序，完成计算。每个人分几支，需要做除法运算，剩下几支通过求余数可以得到。

```
#include <iostream>
using namespace std;
int main() {
    cout << 20 / 8 << endl;
    cout << 20 % 8 << endl;
    return 0;
}
```

运行结果如下：

```
2
4
```

程序中出现了两个运算符“/”和“%”，分别表示除法运算和模运算。C++语言常用的运算符如表 1-2 所示。

表 1-2　C++语言常用的运算符

运算符	作用	说明	示例
+	加法	加法运算	3+2=5
−	减法	减法运算	8−6=2
*	乘法	乘法运算	6*4=24
/	除法	除法运算	8/2=4
%	模	两个数字除法运算后的余数	9%4=1

需要注意的是，运算结果的数据类型和参加运算的数据类型相关，如果两个整数进行除法运算，其结果同样也是整数。

所以，第 1 行输出的 2 是 20/8 的数学运算结果 2.5 的整数部分。第 2 行输出的 4 是整除后的余数。

程序中的cout 是 C++语言的输出语句，这个输出的目标就是系统默认的输出设备（屏幕）。

程序中的 endl 的作用是换行，英语的含义是 end of line。

注意

endl 的最后一个字母是单词 line 的第一个字母 l。

程序在输出内容时，按如下原则处理：**双引号包含的字符串，原样输出；计算表达式，先计算再输出结果。**

要注意 cout 和输出内容之间的符号“<<”，在输出多个内容时，可以连续书写。

【例 1-5】cout 语句的连续输出应用示例。

```
#include <iostream>
using namespace std;
int main() {
    cout << "20 / 8 = " << 20 / 8 << endl;
    cout << "20 % 8 = " << 20 % 8 << endl;
    return 0;
}
```

运行结果如下：

```
20 / 8 = 2
20 % 8 = 4
```

在计算表达式中，运算符*、/、%的优先级别高于+、−。相同级别的运算符按照从左到右的顺序计算。

【例 1-6】运算符的优先级别不同。

```
#include <iostream>
using namespace std;
int main() {
    cout << 9 / 8 * 4 << endl;
    cout << 18 / 3 * 3 << endl;
    cout << 18 / (3 * 3) << endl;
    return 0;
}
```

运算结果如下：

```
4
18
2
```

第 1 行，9/8 的结果虽然有小数部分，但是取整数部分 1 作为结果，所以 9/8*4 的结果是 4。

第 2 行，由于/和*的优先级别相同，所以按照从左到右的顺序依次运算。18/3=6，6*3=18。

第 3 行，由于使用了括号，改变了运算的先后顺序，所以先计算 3*3，然后计算 18/9，结果是 2。

运算符的优先级别总结如下。

（1）先算括号内，再算括号外。

（2）乘、除、模（*、/、%）。

（3）加、减（+、−）。

【例 1-7】将 8000 秒表示为小时:分钟:秒的形式。

【分析】1 小时有 60 分钟，1 分钟有 60 秒，所以 1 小时有 60*60=3600 秒。8000/3600 得到的整数就是对应的小时。8000/3600 得到的余数就是分钟和秒部分，将这个余数除以 60，得到的整数就是分钟数，得到的余数就是剩余的秒数。

按上述分析，编程如下。

```
#include <iostream>
using namespace std;
int main() {
    cout << "8000 秒=";
    cout << 8000 / 3600 << "小时";
    cout << 8000 % 3600 / 60 << "分钟";
    cout << 8000 % 3600 % 60 << "秒" << endl;
    return 0;
}
```

运行结果如下：

```
8000 秒=2 小时 13 分钟 20 秒
```

1.2.3 浮点数运算

如果参与运算的数字有小数部分，就需要使用浮点数类型。

【例 1-8】4 个工人 2 天铺了 60 米的公路，按照这个工作进度，12 个工人 3 天能铺多少米的公路？

【分析】先求 1 个工人 1 天能铺多少米的公路，即 60/2/4=7.5 米，再求 12 个工人 3 天能完成的工作量：7.5*12*3=270 米。

```
#include <iostream>
using namespace std;
int main() {
    cout << "12 个工人 3 天能铺";
    cout << 60.0/2/4*12*3 << "米";
    return 0;
}
```

在这个程序中，如果把 60.0 写为 60，得到的结果会不一样。

表达式 60.0/2/4*12*3 中，60.0/2 的结果是 30.0，60/2 的结果也是 30。

在第二步计算中：30.0/4 的结果是 7.5，而 30/4 的结果是 7，这就是浮点数运算与整数运算的不同结果。

如果需要限定输出的浮点数小数点后的位数，就需要使用格式化输出功能。

【例 1-9】指定浮点数的显示精度。

```
#include <iostream>
#include <iomanip>
using namespace std;
int main() {
    //默认格式
    cout << "10.0/6.0=" << 10.0 / 6.0 << endl;
    cout << "10.0/5=" << 10.0 / 5 << endl;
    //设置：保留小数点后位数
    cout << fixed << setprecision(2);
    cout << "10.0/6.0=" << 10.0 / 6.0 << endl;
    cout << setprecision(4);
    cout << "1000.0/6.0=" << 1000.0 / 6.0 << endl;
    cout << "10.0/6.0="  << 10.0 / 6.0 << endl;
    return 0;
}
```

运行结果如下：

```
10.0/6.0=1.66667
10.0/5=2
10.0/6.0=1.67
1000.0/6.0=166.6667
10.0/6.0=1.6667
```

程序中的 fixed << setprecision(2)是格式函数，作用是让后续输出的实数保留小数点后 2 位。

再次调整小数点后的保留位数时，只需要调用 setprecision(4)函数即可。

使用这个功能，需要注意，必须在头文件中引入#include <iomanip>。

1.3 变量、常量和函数

在编程中存储数据时，需要用到变量和常量，使用变量和常量需要遵循 C++编程语言的规定。在处理数据时，可以使用 C++语言标准库提供的内置函数，以实现特定功能，简化编码。

1.3.1 变量

程序中常常需要把各种数据保存到计算机的内存中，这时就需要使用变量，用变量表示计算机内存中的存储位置。为了方便理解，可以将变量看作数据的容器。

由于不同的数据类型所需的存储空间大小也不同，不同的数据类型在存储时的存储方式也不一致。所以 C++语言要求在声明变量时必须指定变量的数据类型。

【例 1-10】将整数存储到变量，再输出变量值。

```
#include <iostream>
using namespace std;
int main() {
    int a;                  //定义整型变量 a
    a = 12;                 //将整数 12 存入 a 中
    cout << a << endl;      //输出 a 的值
    return 0;
}
```

运行结果如下：

```
12
```

从例 1-10 可以看出，12 可以正常地存储和输出。

在例 1-10 中，如果需要保存的数字过大，会出现什么效果呢？

【例 1-11】将整数 3123456789 存储到变量中，再输出变量的值。

```
#include <iostream>
using namespace std;
int main() {
    int a;
    a = 3123456789;
    cout << a << endl;
    return 0;
}
```

运行结果如下：

```
-1171510507
```

从例 1-11 可以看出，整数 3123456789 没有正常输出，原因是超出了变量 a 的数据类型 int 的取值范围。

变量 a 的数据类型是 int，数据类型 int 在 C++语言中的存储空间是 4 字节，可以存储的数字范围为−2147483648～2147483647。如果存储的整数超出了这个范围，就会发生数据溢出的错误。

在例 1-11 中，只需要将变量 a 的数据类型定义为 long long，就可以避免发生数据溢出错误了。

1. 变量名

为了区分不同的变量，需要为变量取个名称，C++语言变量的命名要遵守以下规则。

（1）变量名只能包含字母（A～Z、a～z）、数字（0～9）或下画线。

（2）变量名不能以数字开头。

（3）变量名不能是 C++语言的关键字。

同时要注意，C++语言的变量名中的字母要区分大小写，也就是说 A 和 a 是两个不同的变量。

【例 1-12】判断以下内容，是否可以作为变量名。

（1） int。

（2）10boy。

（3）my_pen。

（4） us$。

【分析】4 个选项中，只有（3）可以作为变量名，而其他选项，（1）int 是关键字，（2）10boy 以数字开头，（4）us$包含的字符$不可以用于变量命名。

2. 布尔类型变量

C++语言的布尔值只有两种状态，即 true 和 false。但是在实际编程中，可以使用其他值为布尔类型赋值，如表 1-3 所示。

表 1-3　布尔值一览表

真	假
1	0
true	false
非零值（2，-2、1.2）	0

【例 1-13】布尔类型变量。

```
#include<iostream>
using namespace std;
int main() {
    bool a = 0;
    bool b = 1;
    bool c = true;
    bool d = false;
    bool e = 2;
    bool f = -2;
    cout << "a=" << a << endl;
    cout << "b=" << b << endl;
    cout << "c=" << c << endl;
    cout << "d=" << d << endl;
    cout << "e=" << e << endl;
    cout << "f=" << f << endl;
    return 0;
}
```

运行结果如下：

```
a=0
b=1
c=1
d=0
e=1
f=1
```

【分析】

输入数字 0，计算机记录为布尔类型的 false。

输入所有的非 0 数字（包括负数），计算机记录为布尔类型的 true。

布尔类型的 false 输出为 0，布尔类型的 true 输出为 1。

3. 字符类型变量

字符类型在计算机内部存储时，存储字符对应的 ASCII 码值，如字符'A'对应 65，'B'

对应 66，'a'对应 97，'b'对应 98。常用字符的 ASCII 码值及对应的字符如表 1-4 所示。

表 1-4　常用 ASCII 码和字符对应表（节选）

ASCII 码值	字符	ASCII 码值	字符
48	0	49	1
50	2	51	3
52	4	53	5
54	6	55	7
56	8	57	a
65	A	97	a
66	B	98	b
67	C	99	c
68	D	100	d
69	E	101	e
…	…	…	…
88	X	120	x
89	Y	121	y
90	Z	122	z

【例 1-14】字符类型对应的 ASCII 码值。

```
#include <iostream>
using namespace std;
int main() {
    int i;
    char j;
    i = 'a';
    j = 97;
    cout << i << endl;
    cout << j << endl;
    return 0;
}
```

运行结果如下：

```
97
a
```

虽然赋值给变量 i 的是字符'a'，但是输出时，系统按照变量 i 的定义类型（整数）输出。变量 j 也是同理。

4. 基本运算

C++语言中的基本运算分为算术运算、关系运算、逻辑运算和位运算，其中的算术运算符加、减、乘、除、模（+、-、*、/、%）用于数值运算；关系运算符大于、小于、等于、大于等于、小于等于、不等于（>、<、==、>=、<=、!=）用于数据之间的关系判断；逻辑运算符与、或、非（&&、||、!）用于逻辑值的运算；位运算符与、或、非、异或、左移、右移（&、|、~、^、<<、>>）用于二进制运算。

算术运算符计算右侧表达式后，赋值给左侧的变量。

【例 1-15】算术运算符的应用。

```
#include <iostream>
using namespace std;
int main() {
    float x = 7.5;  //定义浮点型变量 x,并给 x 赋值
    float y = 10.6; //定义浮点型变量 y,并给 y 赋值
    float area = x * y; //求面积
    cout << "矩形面积为: " << area << endl;
    return 0;
}
```

运行结果如下：

```
矩形面积为: 79.5
```

赋值运算符可以和常见运算符（+、−、*、/、%）组合成复合赋值，常用复合赋值运算符如表 1-5 所示。

表 1-5　常用复合赋值运算符

简单赋值	复合赋值
a = a + 2;	a += 2;
a = a − 2;	a −= 2;
a = a * 2;	a *= 2;
a = a / 2;	a /= 2;
a = a % 2;	a %= 2;

【例 1-16】复合赋值语句。

```
#include <iostream>
using namespace std;
int main() {
    int a, b;
    a = b = 3;
    a += b;
    cout << a << endl;
    cout << b << endl;
    return 0;
}
```

运行结果如下：

```
6
3
```

关系运算符用于数据之间的关系判断，计算的结果为布尔类型。

【例 1-17】使用关系运算符判断数字大小。

```
#include <iostream>
using namespace std;
int main() {
    int a, b;
    a = 3;
    b = 5;
    cout << "(a<b):" << (a < b) << endl;
    cout << "(a<=b):" << (a <= b) << endl;
    cout << "(a>b):" << (a > b) << endl;
```

```
    cout << "(a>=b):" << (a >= b) << endl;
    cout << "(a==b):" << (a == b) << endl;
    cout << "(a!=b):" << (a != b) << endl;
    return 0;
}
```

运行结果如下：

```
(a<b):1
(a<=b):1
(a>b):0
(a>=b):0
(a==b):0
(a!=b):1
```

逻辑运算与、或、非用于计算逻辑表达式，运算结果是布尔类型。

【例 1-18】使用逻辑运算符计算逻辑表达式。

```
#include <iostream>
using namespace std;
int main() {
    bool a, b;
    a = true;
    b = false;
    cout << "(a&&b):" << (a && b) << endl;
    cout << "(a||b):" << (a || b) << endl;
    cout << "!a:" << !a << endl;
    cout << "!b:" << !b << endl;
    return 0;
}
```

运行结果如下：

```
(a&&b):0
 (a||b):1
!a:0
!b:1
```

位运算与、或、非、异或、左移、右移（&、|、~、^、<<、>>）用于二进制运算。

【例 1-19】使用位运算符进行二进制运算。

```
#include <iostream>
using namespace std;
int main() {
    int a, b;
    a = 5;
    b = 3;
    cout << "(a&b):" << (a & b) << endl;
    cout << "(a|b):" << (a | b) << endl;
    cout << "~a:" << ~a << endl;
    cout << "~b:" << ~b << endl;
    cout << "(a^b):" << (a ^ b) << endl;
    cout << "(a<<b):" << (a << b) << endl;
    cout << "(a>>b):" << (a >> b) << endl;
    return 0;
}
```

运行结果如下：

```
(a&b):1
(a|b):7
```

```
~a:-6
~b:-4
(a^b):6
(a<<b):40
(a>>b):0
```

5. 变量自增和自减

程序运行时，常常需要对数值型变量值加 1 或减 1，这个操作可以通过自增运算符++或自减运算符--完成。

在使用时，要注意自增运算符++或自减运算符--和其他运算符执行的先后顺序。

【例 1-20】自增运算符++。

```
#include <iostream>
using namespace std;
int main() {
    int a = 3, b = 5;
    a++;
    cout << "a= " <<a << endl;
    ++b;
    cout << "b= " << b << endl;
    a = b++;
    cout << "a= " << a << ",b= " << b << endl;
    a = ++b;
    cout << "a= " << a << ",b= " << b << endl;
    return 0;
}
```

运行结果如下：

```
a= 4
b= 6
a= 6,b= 7
a= 8,b= 8
```

在例 1-20 中，a++的作用相当于 a=a+1，++b 相当于 b=b+1。

需要注意的是语句 a=b++，这个语句需要先将变量 b 的值赋给变量 a，再完成变量 b 的自增。

而 a=++b 语句的作用，先完成变量 b 的自增，再将变量 b 的值赋给 a。

【例 1-21】自减运算符--。

```
#include <iostream>
using namespace std;
int main() {
    int a = 3, b = 5;
    a--;
    cout << "a= " << a << endl;
    --b;
    cout << "b= " << b << endl;
    a = b--;
    cout << "a= " << a << ",b= " << b << endl;
    a = --b;
    cout << "a= " << a << ",b= " << b << endl;
    return 0;
}
```

运行结果如下：

```
a= 2
b= 4
a= 4,b= 3
a= 2,b= 2
```

在例 1-21 中，a--和 b--都是自减 1。

a=b--，在自减前先赋值给变量 a；而 a=--b 则相反，先完成自减，再赋值给变量 a。

6. 交换变量值

程序运行时，如果要交换两个变量的值，则需要借助一个临时变量。这个过程类似于两个相同的杯子 A、B，分别装满了不同的饮料，要求将两个杯子的饮料互换。这时需要准备一个空杯子 T，先把 A 中的饮料倒入 T 中，再把 B 中的饮料倒入 A 中，最后把 T 中的饮料倒入 B 中。

【例 1-22】交换变量的值。

```
#include <iostream>
using namespace std;
int main() {
    float a, b, t;
    a = 20.1;
    b = 50.7;
    cout << a << "," << b << endl;
    t = a;
    a = b;
    b = t;
    cout << a << "," << b << endl;
    return 0;
}
```

运行结果如下：

```
20.1,50.7
50.7,20.1
```

7. 数据溢出

程序运行时，需要特别注意数据在运行过程中有可能出现的最大值或最小值。如果有可能超出数据范围，就需要更换数据类型。

【例 1-23】求 2 个整数之和。

```
#include <iostream>
using namespace std;
int main() {
    int a, b, s;
    a = 1545675824;
    b = 1456874589;
    s = a + b;
    cout << s << endl;
    return 0;
}
```

运行结果如下：

```
-1292416883
```

【分析】int 整数的范围为−2147483648～2147483647，程序中，变量 a 和变量 b 的和超出了这个范围，造成数据溢出。

【例 1-24】求 2 个整数之和（修改版 1）。

```
#include <iostream>
using namespace std;
int main() {
    int a, b;
    long long s;
    a = 1545675824;
    b = 1456874589;
    s = a + b;
    cout << s << endl;
    return 0;
}
```

运行结果如下：

```
-1292416883
```

【分析】虽然先将变量 s 声明为取值范围更大的 long long 类型，但是变量 a 和变量 b 相加后会先得到一个 int 类型的值，这个加法步骤已经发生了数据溢出错误，之后再赋值给变量 s，所以结果依然不正确。

【例 1-25】求 2 个整数之和（修改版 2）。

```
#include <iostream>
using namespace std;
int main() {
    int a, b;
    long long s;
    a = 1545675824;
    b = 1456874589;
    s = (long long)a + (long long)b;
    cout << s << endl;
    return 0;
}
```

运行结果如下：

```
3002550413
```

【分析】在发生数据溢出错误的操作之前，先将变量 a 和变量 b 强制转换为 long long 类型，这样变量 a 和变量 b 相加后会得到一个 long long 类型的值，这个值的取值范围不会发生数据溢出错误，再赋值给变量 s，所以最终结果正确。

8. 数据类型转换

在 C++程序运行中，有些数据类型转换是自动完成的，有些情况下需要通过代码来完成。在数据类型转换时要注意避免丢失数据。

【例 1-26】已知三角形的底为 3、高为 5，求三角形的面积。

```
#include <iostream>
using namespace std;
```

```
int main() {
    int a, h;
    float s;
    a = 3;
    h = 5;
    s = a * h / 2;
    cout << s << endl;
    return 0;
}
```

运行结果如下：

```
7
```

这个运行结果明显不正确，因为在运算中，虽然 s 是浮点数，但是在计算 a*h/2 时，由于所有数据都是整数类型，程序会只返回一个整数值，只保留了计算式（a*h/2）的计算结果的整数部分。

正确操作是，在发生丢失小数部分的运算前，先将参与运算的数字强制转换为浮点数。

转换时，不需要全部转换，只需要确保有一个运算的数字是浮点数，其他参与运算的数值会自动转换为对应的数据类型。

【例 1-27】已知三角形的底为 3、高为 5，求三角形的面积（修改版 1）。

```
#include <iostream>
using namespace std;
int main() {
    int a, h;
    float s;
    a = 3;
    h = 5;
    s = a * h / 2.0;
    cout << s << endl;
    return 0;
}
```

运行结果如下：

```
7.5
```

在例 1-27 中，将例 1-26 中的 s=a*h/2;修改为 s=a*h/2.0;，即在除法运算之前，程序自动将 a*h 的结果转换为与 2.0 相同的浮点型，再进行除法运算，结果为浮点型。也可以采用例 1-25 的方法，将第 1 个参与运算的变量 a 强制转换为浮点型（double），后续的运算过程都会以 double 类型为标准，统一数据类型，并完成运算。

【例 1-28】已知三角形的底为 3、高为 5，求三角形的面积（修改版 2）。

```
#include <iostream>
using namespace std;
int main() {
    int a, h;
    float s;
    a = 3;
    h = 5;
    s = (double)a * h / 2;
    cout << s << endl;
    return 0;
}
```

1.3.2 常量

在程序运行过程中，不能改变值的称为常量。常量定义如下：

```
const float PI = 3.14159265;
```

其中，const 是常量说明，float 是类型说明，PI 是常量名，通常使用大写字母定义常量，方便与变量进行区别。

【例 1-29】设置圆周率为常量并参与运算。

```
#include <iostream>
using namespace std;
int main() {
    const double PI = 3.14159265;
    double radius;                          //定义存放半径变量为浮点型
    double area;                            //定义存放面积变量为浮点型
    radius = 7;                             //半径为 7
    area = PI * radius * radius;            //求圆的面积
    cout << "圆面积=" << area << endl;  //输出圆的面积
    return 0;
}
```

运行结果如下：

```
圆面积=153.938
```

如果在程序编译中，强行修改常量的值，会发生错误。如图 1-4 所示，如果添加语句 PI=3.14;，则程序编译时会报错。

在“编译器”窗口的提示信息中，Error 表示有错误发生，错误信息“assignment of read-only variable 'PI'”，意为错误原因是“对只读变量 PI 赋值”。

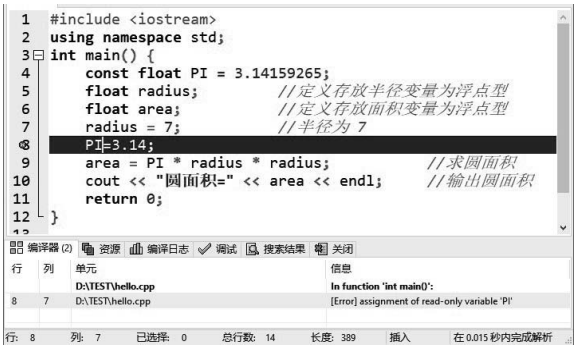

图 1-4 修改常量会导致错误

1.3.3 函数

C++语言的标准数学库中有很多和数学相关的函数，使用前需要先添加头文件引用 #include <cmath>。

【例 1-30】向上取整和向下取整。

```
#include <iostream>
#include <cmath>
using namespace std;
```

```
int main() {
    float x = 3.14;
    cout << ceil(x) << endl;
    cout << floor(x) << endl;
    return 0;
}
```

运行结果如下：

```
4
3
```

【分析】ceil 的含义是天花板，作用是向上取整，3.14 向上取整是 4。floor 的含义是地板，作用是向下取整，3.14 向下取整是 3。

1.4 输入和输出

编程中的输入和输出称为 I/O（input/output）操作。C++语言的所有输入输出都称为数据流，如果流是从设备（如键盘、磁盘驱动器、网络连接等）流向内存，就称为输入操作；如果流是从内存流向设备（如显示屏、打印机、磁盘驱动器、网络连接等），就称为输出操作。

1.4.1 标准输入输出流

在 C++编程中，cin 称为标准输入流，cout 称为标准输出流。cin 处理从标准输入设备（通常是键盘）输入的信息。

【例 1-31】老师准备给信奥班的学生购买一批铅笔，从标准输入设备输入同学人数和每位学生需要的铅笔数量，编程计算需要购买的铅笔总数。

```
#include <iostream>
using namespace std;
int main() {
    int x, y;
    cin >> x >> y;
    cout << x * y << endl;
    return 0;
}
```

运行时输入“40 3”，运行结果如下：

```
40 3
120
```

【分析】程序运行后，光标在命令行窗口中闪动，输入“40 3”，两个数字之间有空格，然后按<Enter>键，程序会输出 120。

在实际程序运行时，也有可能发生输入信息和程序中的需求信息不匹配的情况，如在运行时，输入“40 3 20 6”，观察并分析结果；再重新运行程序，输入“a b”，观察并分析结果。

从不同的输入信息和程序多次运行结果可以看出以下几点。

（1）cin 语句把空格和回车作为分隔符。

（2）cin 语句会忽略多余输入的数据。

（3）如果输入的类型和 cin 语句中的变量类型不匹配，则结果错误。

（4）cin 语句要使用>>符号，和 cout 语句的<<符号的方向相反。

【例 1-32】输入一个小写字母，输出对应的大写字母。

【分析】ASCII 码中，大写字母 A 对应 65，小写字母 a 对应 97，两个字母的差值为 32，将输入的小写字母值减去 32，就可以得到对应的大写字母。

【代码】

```
#include <iostream>
using namespace std;
int main() {
    char c;
    cin >> c;
    c = c - 32;
    cout << c << endl;
    return 0;
}
```

【思考】将上述代码中的“c=c−32;”替换为“c=c−'a'+'A';”，运行结果会变化吗？为什么？

1.4.2 重定向语句

程序运行时如果需要输入信息，程序会等待键盘输入，输出信息会自动发送到屏幕。

除了标准输入输出，C++语言还提供了重定向标准输入输出的功能。

（1）重定向标准输入。

语句 freopen("data.in", "r", stdin);可以将文件 data.in 定义为标准输入流。重定向后，所有需要从键盘读取的信息都可以从文件 data.in 中读取。

（2）重定向标准输出。

语句 freopen("data.out", "w", stdout);可以将文件 data.out 定义为标准输出流。重定向后，所有输出到屏幕的内容会自动写入文件 data.out 中。

在调试过程中，如果调试输入信息过多，使用重定向可以减少调试时的输入时间，让程序自动从文件读取；还可以避免从键盘输入时发生的人为输入错误。

【例 1-33】从文件 data.in 中读取一个小写字母，输出对应的大写字母到文件 data.out 中。

```
#include <iostream>
using namespace std;
int main() {
    freopen("data.in", "r", stdin);
    freopen("data.out", "w", stdout);
    char c;
    cin >> c;
    c = c - 32;
    cout << c << endl;
    return 0;
}
```

程序运行后，运行结果输出到文件 data.out 中，所以屏幕上没有输出大写字母。

运行时输入小写字母 a，输出到文件 data.out 的运行结果是大写字母 A。

【例 1-34】逆向输出数字（数值型处理方法）。

【题目描述】

输入一个不小于 100，且小于 1000，同时包括小数点后一位的一个浮点数。例如，输入 123.4，输出 4.321。

【输入格式】

1 行，1 个浮点数，不小于 100，且小于 1000，同时包括小数点后一位的一个浮点数。

【输出格式】

1 行，逆序输出的浮点数。

【输入样例】

```
123.4
```

【输出样例】

```
4.321
```

【分析】结合使用模运算和除法运算，分离数字的各位置，首先将数字乘以10，将小数变成整数。然后使用模运算取余数，获取各位置上的数字，再反向输出。

```
#include <iostream>
using namespace std;
int main() {
    float n;
    int a, b, c, d, m;
    cin >> n;
    m = n * 10;
    a = m / 1000;
    b = m / 100 % 10;
    c = m / 10 % 10;
    d = m % 10;
    cout << d << "." << c << b << a;
    return 0;
}
```

1.4.3 scanf 语句和 printf 语句

scanf 语句又称格式输入函数，printf 语句又称格式输出函数。在英语中，scan 的含义是扫描，print 的含义是打印。后面的字母 f 是 format 的简写，format 的含义为格式化。

在 C++语言中使用 scanf 和 printf，需要在头文件中添加**#include** <cstdio>。

scanf 语句中的变量前需要添加一个符号“&”，表示取变量的地址。

【例 1-35】输入整数（65～91），输出该 ASCII 码值对应的字符。

【分析 1】如果使用 cin 语句输入整数，则需要通过赋值，将值赋给一个字符类型的变量，再使用 cout 语句输出字符。cout 输出变量时，按照这个变量的定义类型输出。

请比较以下 3 段代码的异同。

【输出 ASCII 码值对应的字符，代码 1】

```
#include <iostream>
using namespace std;
int main() {
    int a;
    char b;
    cin >> a;
    b = a;
    cout << b;
    return 0;
}
```

【输出 ASCII 码值对应的字符，代码 2】，只能输出整数，因为 cout 输出的变量 a 的定义类型是 int。

```
#include <iostream>
using namespace std;
int main() {
    int a;
    cin>>a;
    cout<<a;
    return 0;
}
```

【输出 ASCII 码值对应的字符，代码 3】，只能输出整数的第 1 个数字字符，因为 cout 输出的变量 a 虽然是字符类型，但是 cin 会读取键盘上输入的第 1 个字符，将这个字符赋值给变量 a，并输出。

```
#include <iostream>
using namespace std;
int main() {
    char a;
    cin>>a;
    cout<<a;
    return 0;
}
```

【分析 2】使用 scanf 语句输入，使用 printf 语句输出，格式字符%d 指定输入为整数，通过格式字符%c 指定输出时，按字符类型输出。程序中只需要一个变量即可。变量 a 的声明使用 char 或 int 都可以。

```
#include <iostream>
#include <cstdio>
using namespace std;
int main() {
    char a;
    scanf("%d", &a);
    printf("%c", a);
    return 0;
}
```

【例 1-36】输入浮点数，输出时小数点后保留两位小数。

【分析】使用 printf 语句输出时，可以通过格式字符控制小数点后的位数。%.2f 表示小数点后保留 2 位。%9.3f 表示总共占 9 个字符位置，小数点后保留 3 位。输出格式函数中常用的格式字符及其含义如表 1-6 所示。

```
#include <iostream>
#include <cstdio>
using namespace std;
int main() {
    float a;
    scanf("%f", &a);
    printf("%.2f", a);
    printf("%9.3f", a);
    return 0;
}
```

运行结果如下：

```
1.2345
1.23    1.235
```

【说明】输入 1.2345；第 1 个输出保留小数点后 2 位，第 2 个输出保留小数点后 3 位，总共占位 9 位，前面补充 4 个空格。

表 1-6　输出格式函数中常用的格式字符及其含义

格式字符	含义
%d	十进制
%f、%lf	浮点数/双精度浮点数
%c	字符
%s	字符串

1.4.4　快速读取

C++语言中常用的读取方式是 cin 语句，如果要获取更快的读取速度可以使用 scanf 读入语句，特别在读取数据量比较大时更加明显。

【例 1-37】通过文件读取较大数据。

【分析】在语句 freopen("data.in", "r", stdin)中，freopen 语句重定向标准输入，使用文件作为标准输入。以下使用包含 10000000 个整数的测试文件 data.in 测试不同输入方式的区别。

制作包含 10000000 个整数的测试文件 data.in 的运行代码时，会自动生成一个包含 10000000 个随机整数的 data.in 文件。

```
#include <iostream>
#include <cstdlib>
#include <ctime>
using namespace std;
int main() {
    freopen("data.in", "w", stdout);
    srand(time(0));
    for (int i = 0; i < 10000000; i++) {
        cout << rand() << " ";
    }
    return 0;
}
```

【代码 1，通过 cin 语句读取】

```
#include <iostream>
#include <cstdio>
```

```
using namespace std;
int main() {
    int n;
    freopen("data.in", "r", stdin);
    for (int i = 1; i <= 10000000; i++) {
        cin >> n;
    }
    return 0;
}
```

运行程序后，在结果中可以看到整个运行时间如下：

```
Process exited after 7.772 seconds with return value 0
```

注意

在不同软硬件环境的计算机中，程序运行时间会不同，即使是同一台计算机，每次的运行时间也略有差异。

【代码 2，通过 scanf 语句读取】

```
#include <iostream>
#include <cstdio>
using namespace std;
int main() {
    int n;
    freopen("data.in", "r", stdin);
    for (int i = 1; i <= 10000000; i++) {
        scanf("%d", &n);
    }
    return 0;
}
```

运行程序后，在结果中可以看到整个运行时间如下：

```
Process exited after 6.477 seconds with return value 0
```

此外，还可以通过关闭同步的方式加快 cin 的读入速度，需要在程序开始加上以下语句：

```
ios::sync_with_stdio(false);
```

需要注意，使用此功能后，就不可以再使用 scanf 输入数据了。

【代码 3，通过关闭同步的方式加快 cin 的读取速度】

```
#include <iostream>
#include <cstdio>
using namespace std;
int main() {
    ios::sync with stdio(false);
    int n;
    freopen("data.in", "r", stdin);
    for (int i = 1; i <= 10000000; i++) {
        cin >> n;
    }
    return 0;
}
```

运行程序后，在结果中可以看到整个运行时间如下：

```
Process exited after 2.612 seconds with return value 0
```

【代码 4，快速读取】通过 getchar()实现读取，每次读取一个字符，需要编程处理读取的字符，如果连续读取到数字字符，则需要组合为较大的数字。具体实现过程如下。

```
#include <iostream>
#include <cstdio>
using namespace std;
int read() {
    int f = 1;
    int x = 0;
    char c = getchar();
    while (!isdigit(c)) {
        if (c == '-')
            f = -1;
        c = getchar();
    }
    while (isdigit(c)) {
        x = x * 10 + c - '0';
        c = getchar();
    }
    return x * f;
}
int main() {
    int n;
    freopen("data.in", "r", stdin);
    for (int i = 1; i <= 10000000; i++) {
        n = read();
    }
    return 0;
}
```

运行程序后，在结果中可以看到整个运行时间如下：

```
Process exited after 2.206 seconds with return value 0
```

【总结】以上 4 种读取方式都是从文件中读取 10000000 个数字，虽然每次的运行时间略有差异，多测试几次可以发现代码 1 耗费的时间最多，代码 4 耗费的时间最少。

【思考练习】

习题 1-1：按题意编程输出

车厘子成熟后需要及时安排工人采摘果实。去年，小明家有 765 棵果树，请了 12 名工人，耗时 5 天才完成采摘。今年，小明家又新种植了 1377 棵果树，假设每棵果树的结果量都和去年的结果量相当，今年由于天气原因，需要在 4 天内完成采摘任务，请问需要请多少名工人？

习题 1-2：按题意编程输出

小明和小红的跑步速度分别为 2.7 米/秒和 1.8 米/秒，小红在操场环形跑道上跑步半分钟之后小明才开始跑步，请问小明多少秒后可以超过小红？

习题 1-3：按题意编程输出

某电商网站 1 天有 3000 万～4000 万订单，技术人员在给订单编号时准备使用数据类型 int，从 1 开始依次给订单编号（订单号必须是正整数）。这种订单编号方案是否可行？如果按 1 天 4000 万订单计算，这种订单方案使用多少天会出现问题？

习题 1-4：请写出下面代码的运行结果

```
#include <iostream>
using namespace std;
int main() {
    int a = 5, b = 6;
    cout << a++ + ++a << endl;
    cout << b-- - --b << endl;
    return 0;
}
```

读书笔记