C++ Primer（第 4 版）习题解答

图灵程序设计丛书

C++ Primer（第4版）

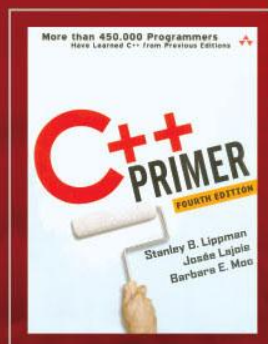
习题解答

蒋爱军　李师贤　梅晓勇　编著

■ 学习 C++ 的最佳伴侣

■ 涵盖所有习题的答案

■ 代码均配有详细注释



人民邮电出版社
POSTS & TELECOM PRESS

- 书名：**C++ Primer**（第 **4** 版）习题解答
- 作者：蒋爱军　李师贤　梅晓勇
- 来源：人民邮电出版社
- 出版时间：**2006** 年 **12** 月
- **ISBN:9787115155108**
- 定价：**45** 元

内容介绍：

C++ Primer（第 **4** 版）是 **C++**大师 **Stanley B. Lippman** 丰富的实践经验和 **C++**标准委员会原负责人 **Josée Lajoie** 对 **C++**标准深入理解的完美结合，更加入了 **C++**先驱 **Barbara E. Moo** 在 **C++**教学方面的真知灼见，

是初学者的最佳 C++指南，而且对于中高级程序员，也是不可或缺的参考书。本书正是这部久负盛名的 C++经典教程的配套习题解答。书中提供了 C++ Primer（第 4 版）中所有习题的参考答案。
本书对使用 C++ Primer（第 4 版）学习 C++程序设计语言的读者是非常理想的参考书。

C++是一门非常实用的程序设计语言，既支持过程式程序设计，也支持面向对象程序设计，因而也是目前应用极为广泛的一门程序设计语言。

在层出不穷的介绍 C++语言的书籍中，*C++ Primer* 是一本广受欢迎的权威之作。强大的作者阵容、全面的内容介绍、新颖的组织方式，使之深受 C++爱好者的青睐。本书编者在翻译 *C++ Primer*（第 4 版）的过程中也深深地感受到了这一点。

在学习一门程序设计语言的过程中，亲自动手编写代码是一种极其有效的学习方式，可以对语言的理解和应用达到事半功倍的效果，因此，*C++ Primer*（第 4 版）中提供了许多习题，以帮助读者加深对书中内容的理解。

本书试图成为 *C++ Primer*（第 4 版）的配套书籍，根据 *C++ Primer*（第 4 版）中所介绍的内容提供配套习题的解答，书中所给出的“见 xx 节”，均指参见 *C++ Primer*（第 4 版）的相应章节。

本书中给出的程序均已通过 Microsoft Visual C++ .NET 2003 的编译。源文件（实现文件）以.cpp 为扩展名，头文件为了与此对应采用.hpp 为扩展名（而没有采用编译器的默认扩展名.h）。为了节省篇幅，有些程序中将类的定义与使用类的主函数放在同一实现文件中。包含主函数的源文件根据习题编号命名。大多数模板的定义都没有区分头文件和实现文件（因为编者所用的编译器支持模板的包含编译模型）。另外，使用 Visual C++ .NET 2003 编译器的默认设置会自动连接一些默认库，因此可能有某些所用到的库函数或库类型没有显式指明相应的头文件。使用其他编译器的读者需特别注意，必要时应加上相应的 #include 指示。

衷心希望本书能对使用 *C++ Primer*（第 4 版）学习 C++语言的读者有所帮助。

由于编者水平所限，书中不当之处在所难免，恳请读者批评指正。

编　者

2006 年 10 月

习题 1.1

查看所用的编译器文档，了解它所用的文件命名规范。编译并运行本节的 main 程序。

【解答】

一般而言，C++编译器要求待编译的程序保存在文件中。C++程序中一般涉及两类文件：头文件和源文件。大多数系统中，文件的名字由文件名和文件后缀（又称扩展名）组成。文件后缀通常表明文件的类型，如头文件的后缀可以是.h 或.hpp 等；源文件的后缀可以是.cc 或.cpp 等，具体的后缀与使用的编译器有关。通常可以通过编译器所提供的联机帮助文档了解其文件命名规范。

习题 1.2

修改程序使其返回-1。返回值-1 通常作为程序运行失败的指示器。然而，系统不同，如何（甚至是否）报告 main 函数运行失败也不同。重新编译并再次运行程序，看看你的系统如何处理 main 函数的运行失败指示器。

【解答】

笔者所使用的 Windows 操作系统并不报告 main 函数的运行失败，因此，程序返回-1 或返回 0 在运行效果上没有什么区别。但是，如果在 DOS 命令提示符方式下运行程序，然后再键入 echo %ERRORLEVEL%命令，则系统会显示返回值-1。

习题 1.3

编一个程序，在标准输出上打印“Hello, World”。

【解答】

```
#include<iostream>

int main()

{

    std::cout << "Hello, World" << std::endl;

    return 0;

}
```

习题 1.4

我们的程序利用内置的加法操作符“+”来产生两个数的和。编写程序，使用乘法操作符“*”产生两个数的积。

【解答】

```
#include <iostream>
```

```
int main()
{
    std::cout << "Enter two numbers:" << std::endl;
    int v1, v2;
    std::cin >> v1 >> v2;
    std::cout << "The product of " << v1 << " and " << v2
                << " is " << v1 * v2 << std::endl;
    return 0;
}
```

习题 1.5

我们的程序使用了一条较长的输出语句。重写程序，使用单独的语句打印每一个操作数。

【解答】

```
#include <iostream>
int main()
{
    std::cout << "Enter two numbers:" << std::endl;
    int v1, v2;
    std::cin >> v1 >> v2;
    std::cout << "The sum of ";
    std::cout << v1;
    std::cout << " and ";
    std::cout << v2;
    std::cout << " is ";
```

```
    std::cout << v1 + v2 ;

    std::cout << std::endl;

    return 0;

}
```

习题 1.6

解释下面的程序段：

```
std::cout << "The sum of " << v1;

           << " and " << v2;

           << " is " << v1 + v2

           << std::endl;
```

这段代码合法吗？如果合法，为什么？如果不合法，又为什么？

【解答】

这段代码不合法。

注意，第 1、2、4 行的末尾有分号，表示这段代码包含三条语句，即第 1、2 行各为一个语句，第 3、4 行构成一个语句。“<<”为二元操作符，在第 2、3 两条语句中，第一个“<<”缺少左操作数，因此不合法。

在第 2、3 行的开头加上“std::cout”，即可更正。

习题 1.7

编译有不正确嵌套注释的程序。

【解答】

由注释对嵌套导致的编译器错误信息通常令人迷惑。例如，在笔者所用的编译器中编译 1.3 节中给出的带有不正确嵌套注释的程序：

```
#include <iostream>

/*

 * comment pairs /* */ cannot nest.
```

```
 * "cannot nest" is considered source code,
 * as is the rest of the program
 */
int main()
{
    return 0;
}
```

编译器会给出如下错误信息：

error C2143: syntax error : missing ';' before '<'

error C2501: 'include' : missing storage-class or type specifiers

warning C4138: '*/' found outside of comment （第 6 行）

error C2143: syntax error : missing ';' before '{' （第 8 行）

error C2447: '{' : missing function header (old-style formal list?)（第 8 行）

习题 1.8

指出下列输出语句哪些（如果有）是合法的。

```
std::cout << "/*";
std::cout << "*/";
std::cout << /* "*/" */;
```

预测结果，然后编译包含上述三条语句的程序，检查你的答案。纠正所遇到的错误。

【解答】

第一条和第二条语句合法。

第三条语句中<<操作符之后至第二个双引号之前的部分被注释掉了，导致<<操作符的右操作数不是一个完整的字符串，所以不合法。在分号之前加上一个双引号即可更正。

习题 1.9

下列循环做什么？sum 的最终值是多少？

```
    int sum = 0;

    for (int i = -100; i <= 100; ++i)

        sum += i;
```

【解答】

该循环求-100~100 之间所有整数的和（包括-100 和 100）。

sum 的最终值是 0。

习题 1.10

用 for 循环编程，求从 50~100 的所有自然数的和。然后用 while 循环重写该程序。

【解答】

用 for 循环编写的程序如下：

```
#include <iostream>

int main()

{

    int sum = 0;

    for (int i = 50; i <= 100; ++i)

        sum += i;

     std::cout << "Sum of 50 to 100 inclusive is "

               << sum << std::endl;

     return 0;
```

```
}
```

用 while 循环编写的程序如下：

```
#include <iostream>
int main()
{
    int sum = 0, int i = 50;
    while (i <= 100) {
        sum += i;
        ++i;
    }
    std::cout << "Sum of 50 to 100 inclusive is "
              << sum << std::endl;
    return 0;
}
```

习题 1.11

用 while 循环编程，输出 10~0 递减的自然数。然后用 for 循环重写该程序。

【解答】

用 while 循环编写的程序如下：

```
#include <iostream>
int main()
{
    int i = 10;
    while (i >= 0) {
        std::cout << i << " ";
```

```
        --i;
    }
    return 0;
}
```

用 for 循环编写的程序如下：

```
#include <iostream>
int main()
{
    for (int i = 10; i >= 0; --i)
         std::cout << i << " ";
    return 0;
}
```

习题 1.12

对比前面两个习题中所写的循环。两种形式各有何优缺点？

【解答】

在 for 循环中，循环控制变量的初始化和修改都放在语句头部分，形式较简洁，且特别适用于循环次数已知的情况。在 while 循环中，循环控制变量的初始化一般放在 while 语句之前，循环控制变量的修改一般放在循环体中，形式上不如 for 语句简洁，但它比较适用于循环次数不易预知的情况（用某一条件控制循环）。两种形式各有优点，但它们在功能上是等价的，可以相互转换。

习题 1.13

编译器不同，理解其诊断内容的难易程度也不同。编写一些程序，包含本小节“再谈编译”部分讨论的那些常见错误。研究编译器产生的信息，这样你在编译更复杂的程序遇到这些信息时不会陌生。

【解答】

对于程序中出现的错误，编译器通常会给出简略的提示信息，包括错误出现的文件及代码行、错误代码、错误性质的描述。如果要获得关于该错误的详细信息，一般可以根据编译器给出的错误代码在其联机帮助文档中查找。

习题 1.14

如果输入值相等，本节展示的程序将产生什么问题？

【解答】

sum 的值即为输入值。因为输入的 v1 和 v2 值相等（假设为 x），所以 lower 和 upper 相等，均为 x。for 循环中的循环变量 val 初始化为 lower，从而 val<=upper 为真，循环体执行一次，sum 的值为 val（即输入值 x）；然后 val 加 1，val 的值就大于 upper，循环执行结束。

习题 1.15

用两个相等的值作为输入编译并运行本节中的程序。将实际输出与你在习题 1.14 中所做的预测相比较，解释实际结果和你预计的结果间的不相符之处。

【解答】

运行 1.4.3 节中给出的程序，输入两个相等的值（例如 3,3），则程序输出为：

Sum of 3 to 3 inclusive is 3

与习题 1.14 中给出的预测一致。

习题 1.16

编写程序，输出用户输入的两个数中的较大者。

【解答】

```
#include <iostream>

int main()

{

    std::cout << "Enter two numbers:" << std::endl;

    int v1, v2;

    std::cin >> v1 >> v2; // 读入数据
```

```
    if (v1 >= v2)
        std::cout << "The bigger number is" << v1 << std::endl;
    else
        std::cout << "The bigger number is" << v2 << std::endl;
    return 0;
}
```

习题 1.17

编写程序，要求用户输入一组数。输出信息说明其中有多少个负数。

【解答】

```
#include <iostream>
int main()
{
    int amount = 0, value;
    // 读入数据直到遇见文件结束符，计算所读入的负数的个数
    while (std::cin >> value)
        if (value <= 0)
            ++amount;
    std::cout << "Amount of all negative values read is"
              << amount << std::endl;
    return 0;
}
```

习题 1.18

编写程序，提示用户输入两个数并将这两个数范围内的每个数写到标准输出。

【解答】

```
#include <iostream>
int main()
{
    std::cout << "Enter two numbers:" << std::endl;
    int v1, v2;
    std::cin >> v1 >> v2; // 读入两个数

    // 用较小的数作为下界 lower、较大的数作为上界 upper
    int lower, upper;
    if (v1 <= v2) {
        lower = v1;
        upper = v2;
    } else {
        lower = v2;
        upper = v1;
    }
    // 输出从 lower 到 upper 之间的值
    std::cout << "Values of " << lower << "to "
              << upper << "inclusive are: " << std::endl;
    for (int val = lower; val <= upper; ++val)
        std::cout << val << " ";
    return 0;
}
```

习题 1.19

如果上题给定数 1000 和 2000，程序将产生什么结果？修改程序，使每一行输出不超过 10 个数。

【解答】

所有数的输出连在一起，不便于阅读。

程序修改如下：

```
#include <iostream>

int main()

{

    std::cout << "Enter two numbers:" << std::endl;

    int v1, v2;

    std::cin >> v1 >> v2; // 读入两个数

    // 用较小的数作为下界 lower、较大的数作为上界 upper

    int lower, upper;

    if (v1 <= v2) {

        lower = v1;

        upper = v2;

    } else {

        lower = v2;

        upper = v1;

    }

    // 输出从 lower 到 upper 之间的值

    std::cout << "Values of " << lower << "to "

                 << upper << "inclusive are: " << std::endl;
```

```
    for (int val = lower, count=1; val <= upper; ++val, ++count) {
        std::cout << val << " ";
        if (count % 10 == 0)            //每行输出 10 个值
            std::cout << std::endl;
    }
    return 0;
}
```

粗黑体部分为主要的修改：用变量 count 记录已输出的数的个数；若 count 的值为 10 的整数倍，则输出一个换行符。

习题 1.20

编写程序，求用户指定范围内的数的和，省略设置上界和下界的 if 测试。假定输入数是 7 和 3，按照这个顺序，预测程序运行结果。然后按照给定的数是 7 和 3 运行程序，看结果是否与你预测的相符。如果不相符，反复研究关于 for 和 while 循环的讨论直到弄清楚其中的原因。

【解答】

可编写程序如下：

```
// 1-20.cpp
// 省略设置上界和下界的 if 测试，求用户指定范围内的数的和
#include <iostream>
int main()
{
    std::cout << "Enter two numbers:" << std::endl;
    int v1, v2;
    std::cin >> v1 >> v2;  // 读入数据
    int sum = 0;
```

```
    // 求和
    for (int val = v1; val <= v2; ++val)
            sum += val; // sum = sum + val
    std::cout << "Sum of " << v1
             << " to " << v2
             << " inclusive is "
             << sum << std::endl;
    return 0;
}
```

如果输入数据为 7 和 3，则 v1 值为 7，v2 值为 3。for 语句头中将 val 的初始值设为 7，第一次测试表达式 val <= v2 时，该表达式的值为 false，for 语句的循环体一次也不执行，所以求和结果 sum 为 0。

习题 1.21

本书配套网站（http://www.awprofessional.com/cpp_primer）的第 1 章的代码目录下有 Sales_ item.h 源文件。复制该文件到你的工作目录。编写程序，循环遍历一组书的销售交易，读入每笔交易并将交易写至标准输出。

【解答】

```
#include <iostream>
#include "Sales_item.h"
int main()
{
   Sales_item book;
   // 读入 ISBN，售出书的本数，销售价格
   std::cout << "Enter transactions:" << std::endl;
   while (std::cin >> book) {
```

```
        // 输出 ISBN，售出书的本数，总收入，平均价格
        std::cout << "ISBN, number of copies sold, "
                      << "total revenue, and average price are:"
                      << std::endl;
        std::cout << book << std::endl;
    }
    return 0;
}
```

习题 1.22

编写程序，读入两个具有相同 ISBN 的 Sales_item 对象并产生它们的和。

【解答】

```
#include <iostream>
#include "Sales_item.h"
int main()
{
    Sales_item trans1, trans2;
    // 读入交易
    std::cout << "Enter two transactions:" << std::endl;
    std::cin >> trans1 >> trans2;
    if (trans1.same_isbn(trans2))
        std::cout << "The total information: " << std::endl
                     << "ISBN, number of copies sold, "
                     << "total revenue, and average price are:"
                     << std::endl << trans1 + trans2;
```

```
    else
        std::cout << "The two transactions have different ISBN."
                  << std::endl;
    return 0;
}
```

习题 1.23

编写程序，读入几个具有相同 ISBN 的交易，输出所有读入交易的和。

【解答】

```
#include <iostream>
#include "Sales_item.h"
int main()
{
    Sales_item total, trans;
    // 读入交易
    std::cout << "Enter transactions:" << std::endl;
    if (std::cin >> total) {
        while (std::cin >> trans)
            if (total.same_isbn(trans))  // ISBN 相同
                total = total + trans;
            else { // ISBN 不同
                std::cout << "Different ISBN." << std::endl;
                return -1;
            }
        // 输出交易之和
```

```
        std::cout << "The total information: " << std::endl
                  << "ISBN, number of copies sold, "
                  << "total revenue, and average price are:"
                  << std::endl << total;
        }
    else {
        std::cout << "No data?!" << std::endl;
        return -1;
    }
    return 0;
}
```

习题 1.24

编写程序，读入几笔不同的交易。对于每笔新读入的交易，要确定它的 ISBN 是否和以前的交易的 ISBN 一样，并且记下每一个 ISBN 的交易的总数。通过给定多笔不同的交易来测试程序。这些交易必须代表多个不同的 ISBN，但是每个 ISBN 的记录应分在同一组。

【解答】

```
    #include <iostream>
#include "Sales_item.h"
int main()
{
    // 声明变量以保存交易记录以及具有相同 ISBN 的交易的数目
    Sales_item trans1, trans2;
    int amount;
    // 读入交易
```

```
    std::cout << "Enter transactions:" << std::endl;
    std::cin >> trans1;
    amount=1;
    while (std::cin >> trans2)
        if (trans1.same_isbn(trans2))// ISBN 相同
            ++amount;
        else {
            // ISBN 不同
            std::cout << "Transaction amount of previous ISBN: "
                      << amount << std::endl;
            trans1 = trans2;
            amount=1;
        }
    // 输出最后一个 ISBN 的交易数目
    std::cout << "Transaction amount of the last ISBN: "
              << amount << std::endl;
    return 0;
}
```

习题 1.25

使用源自本书配套网站的 Sales_item.h 头文件，编译并执行 1.6 节给出的书店程序。

【解答】

可从 *C++ Primer*（第 4 版）的配套网站 (http://www.awprofessional.com/cpp_primer）下载头文件 Sales_item.h，然后使用该头文件编译并执行 1.6 节给出的书店程序。

习题 1.26

在书店程序中，我们使用了加法操作符而不是复合赋值操作符将 trans 加到 total 中，为什么我们不使用复合赋值操作符？

【解答】

因为在 1.5.1 节中提及的 Sales_item 对象上的操作中只包含了+和=，没有包含+=操作。（但事实上，使用 Sales_item.h 文件，已经可以用+=操作符取代=和+操作符的复合使用。）

习题 2.1

int、long 和 short 类型之间有什么差别？

【解答】

它们的最小存储空间不同，分别为 16 位、32 位和 16 位。一般而言，short 类型为半个机器字（word）长，int 类型为一个机器字长，而 long 类型为一个或两个机器字长（在 32 位机器中，int 类型和 long 类型的字长通常是相同的）。因此，它们的表示范围不同。

习题 2.2

unsigned 和 signed 类型有什么差别？

【解答】

前者为无符号类型，只能表示大于或等于 0 的数。后者为带符号类型，可以表示正数、负数和 0。

习题 2.3

如果在某机器上 short 类型占 16 位，那么可以赋给 short 类型的最大数是什么？unsigned short 类型的最大数又是什么？

【解答】

若在某机器上 short 类型占 16 位，那么可以赋给 short 类型的最大数是 $2^{15}-1$，即 32767；而 unsigned short 类型的最大数为 $2^{16}-1$，即 65535。

习题 2.4

当给 16 位的 unsigned short 对象赋值 100000 时，赋的值是什么？

【解答】

34464。

100000 超过了 16 位的 unsigned short 类型的表示范围，编译器对其二进制表示截取低 16 位，相当于对 65536 求余（求模，%），得 34464。

习题 2.5

float 类型和 double 类型有什么差别？

【解答】

二者的存储位数不同（一般而言，float 类型为 32 个二进制位，double 类型为 64 个二进制位），因而取值范围不同，精度也不同（float 类型只能保证 6 位有效数字，而 double 类型至少能保证 10 位有效数字）。

习题 2.6

要计算抵押贷款的偿还金额，利率、本金和付款额应分别选用哪种类型？解释你选择的理由。

【解答】

利率可以选择 float 类型，因为利率通常为百分之几。一般只保留到小数点后两位，所以 6 位有效数字就足以表示了。

本金可以选择 long 类型，因为本金通常为整数。long 类型可表示的最大整数一般为 $2^{31}-1$（即 2147483647），应该足以表示了。

付款额一般为实数，可以选择 double 类型，因为 float 类型的 6 位有效数字可能不足以表示。

习题 2.7

解释下列字面值常量的不同之处。

(a) 'a', L'a', "a", L"a"

(b) 10, 10u, 10L, 10uL, 012, 0xC

(c) 3.14, 3.14f, 3.14L

【解答】

(a) 'a', L'a', "a", L"a"

'a'为 char 型字面值，L'a'为 wchar_t 型字面值，"a"为字符串字面值，L"a"为宽字符串字面值。

(b) 10, 10u, 10L, 10uL, 012, 0xC

10 为 int 型字面值，10u 为 unsigned 型字面值，10L 为 long 型字面值，10uL 为 unsigned long 型字面值，012 为八进制表示的 int 型字面值，0xC 为十六进制表示的 int 型字面值。

(c) 3.14, 3.14f, 3.14L

3.14 为 double 型字面值，3.14f 为 float 型字面值，3.14L 为 long double 型字面值。

习题 2.8

确定下列字面值常量的类型：

(a) - 10 (b) -10u (c) -10. (d) -10e-2

【解答】

(a) int 型

(b) unsigned int 型

(c) double 型

(d) double 型

习题 2.9

下列哪些（如果有）是非法的？

(a) "Who goes with F\145rgus?\012"

(b) 3.14e1L

(c) "two" L"some"

(d) 1024f

(e) 3.14UL

(f) "multiple line

comment"

【解答】

(c) 非法。因为字符串字面值与宽字符串字面值的连接是未定义的。

(d) 非法。因为整数 1024 后面不能带后缀 f。

(e) 非法。因为浮点字面值不能带后缀 U。

(f) 非法。因为分两行书写的字符串字面值必须在第一行的末尾加上反斜线。

习题 2.10

使用转义字符编写一段程序，输出 2M，然后换行。修改程序，输出 2，跟着一个制表符，然后是 M，最后是换行符。

【解答】

输出 2M、然后换行的程序段：

```
// 输出"2M"和换行字符
std::cout << "2M" << '\n';
```

修改后的程序段：

```
// 输出'2', '\t', 'M'和换行字符
std::cout << '2' << '\t' << 'M' << '\n';
```

习题 2.11

编写程序，要求用户输入两个数——底数（base）和指数（exponent），输出底数的指数次方的结果。

【解答】

```
#include <iostream>
int main()
{
    // 局部对象
    int base, exponent;
    long result=1;
```

```
    // 读入底数（base）和指数（exponent）
    std::cout << "Enter base and exponent:" << std::endl;
    std::cin >> base >> exponent;
    if (exponent < 0) {
        std::cout << "Exponent can't be smaller than 0" << std::endl;
        return -1;
    }
    if (exponent > 0) {
        // 计算底数的指数次方
        for (int cnt = 1; cnt <= exponent; ++cnt)
            result *= base;
    }
    std::cout << base
              << " raised to the power of "
              << exponent << ": "
              << result << std::endl;
    return 0;
}
```

习题 2.12

区分左值和右值，并举例说明。

【解答】

左值（lvalue）就是变量的地址，或者是一个代表“对象在内存中的位置”的表达式。

右值（rvalue）就是变量的值，见 2.3.1 节。

变量名出现在赋值运算符的左边，就是一个左值；而出现在赋值运算符右边的变量名或字面常量就是一个右值。

例如：

val1=val2/8

这里的 val1 是个左值，而 val2 和 8 都是右值。

习题 2.13

举出一个需要左值的例子。

【解答】

赋值运算符的左边（被赋值的对象）需要左值，见习题 2.12。

习题 2.14

下面哪些（如果有）名字是非法的？更正每个非法的标识符名字。

(a) int double = 3.14159;　　(b) char _;

(c) bool catch-22;　　(d) char 1_or_2 ='1';

(e) float Float = 3.14f;

【解答】

(a) double 是 C++语言中的关键字，不能用作用户标识符，所以非法。此语句可改为：double dval = 3.14159;。

(c) 名字 catch-22 中包含在字母、数字和下划线之外的字符“-”，所以非法。可将其改为：catch_22;。

(d) 名字 1_or_2 非法，因为标识符必须以字母或下划线开头，不能以数字开头。可将其改为：one_or_two;。

习题 2.15

下面两个定义是否不同？有何不同？

int month = 9, day = 7;

int month =09, day = 07;

如果上述定义有错的话，那么应该怎样改正呢?

【解答】

这两个定义不同。前者定义了两个 int 型变量，初值分别为 9 和 7；后者也定义了两个 int 型变量，其中 day 被初始化为八进制值 7；而 month 的初始化有错：试图将 month 初始化为八进制值 09，但八进制数字范围为 0~7，所以出错。可将第二个定义改为：

```
int month =011, day = 07;
```

习题 2.16

假设 calc 是一个返回 double 对象的函数。下面哪些是非法定义？改正所有的非法定义。

(a) int car = 1024, auto = 2048;

(b) int ival = ival;

(c) std::cin >> int input_value;

(d) double salary = wage = 9999.99;

(e) double calc = calc();

【解答】

(a) 非法：auto 是关键字，不能用作变量名。使用另一变量名，如 aut 即可更正。

(c) 非法：>>运算符后面不能进行变量定义。改为：

```
int input_value;

std::cin >> input_value;
```

(d) 非法：同一定义语句中不同变量的初始化应分别进行。改为：

```
double salary = 9999.99, wage = 9999.99;
```

注意，(b)虽然语法上没有错误，但这个初始化没有实际意义，ival 仍是未初始化的。

习题 2.17

下列变量的初始值（如果有）是什么？

```
std::string global_str;

int global_int;

int main()

{

    int local_int;

    std::string local_str;

    // ...

     return 0;

}
```

【解答】

global_str 和 local_str 的初始值均为空字符串，global_int 的初始值为 0，local_int 没有初始值。

习题 2.18

解释下列例子中 name 的意义：

```
extern std::string name;

std::string name("exercise 3.5a");

extern std::string name("exercise 3.5a");
```

【解答】

第一条语句是一个声明，说明 std::string 变量 name 在程序的其他地方定义。

第二条语句是一个定义，定义了 std::string 变量 name，并将 name 初始化为"exercise 3.5a"。

第三条语句也是一个定义，定义了 std::string 变量 name，并将 name 初始化为"exercise 3.5a"，但这个语句只能出现在函数外部（即，name 是一个全局变量）。

习题 2.19

下列程序中 j 的值是多少？

```
int i = 42;

int main()

{

    int i = 100;

    int j = i;

    // ...

}
```

【解答】

j 的值是 100。j 的赋值所使用到的 i 应该是 main 函数中定义的局部变量 i，因为局部变量的定义会屏蔽全局变量的定义。

习题 2.20

下列程序段将会输出什么？

```
int i = 100, sum = 0;

for (int i = 0; i != 10; ++i)

    sum += i;

std::cout << i << " " << sum << std::endl;
```

【解答】

输出为：

100 45

for 语句中定义的变量 i，其作用域仅限于 for 语句内部。输出的 i 值是 for 语句之前所定义的变量 i 的值。

习题 2.21

下列程序合法吗？

```
int sum = 0;
for (int i = 0; i != 10; ++i)
    sum += i;
std::cout << "Sum from 0 to " << i
          << " is " << sum << std::endl;
```

【解答】

不合法。因为变量 i 具有语句作用域，只能在 for 语句中使用，输出语句中使用 i 属非法。

习题 2.22

下列程序段虽然合法，但是风格很糟糕。有什么问题呢？怎样改善？

```
for (int i = 0; i < 100; ++i)
// process i
```

【解答】

问题主要在于使用了具体值 100 作为循环上界：100 的意义在上下文中没有体现出来，导致程序的可读性差；若 100 这个值在程序中出现多次，则当程序的需求发生变化（如将 100 改变为 200）时，对程序代码的修改复杂且易出错，导致程序的可维护性差。

改善方法：设置一个 const 变量（常量）取代 100 作为循环上界使用，并为该变量选择有意义的名字。

习题 2.23

下列哪些语句合法？对于那些不合法的使用，解释原因。

(a) const int buf;

(b) int cnt = 0;

    const int sz = cnt;

(c) cnt++; sz++;

【解答】

(a) 不合法。因为定义 const 变量（常量）时必须进行初始化，而 buf 没有初始化。

(b) 合法。

(c) 不合法。因为修改了 const 变量 sz 的值。

习题 2.24

下列哪些定义是非法的？为什么？如何改正？

(a) int ival = 1.01;　　　　(b) int &rval1 = 1.01;

(c) int &rval2 = ival;　　(d) const int &rval3 = 1;

【解答】

(b)非法。

因为 rval1 是一个非 const 引用，非 const 引用不能绑定到右值，而 1.01 是一个右值。可改正为：

int &rval1 = ival;

（假设 ival 是一个已定义的 int 变量）。

习题 2.25

在习题 2.24 给出的定义下，下列哪些赋值是非法的？如果赋值合法，解释赋值的作用。

(a) rval2 = 3.14159;　　(b) rval2 = rval3;

(c) ival = rval3;　　　　(d) rval3 = ival;

【解答】

(d)非法。因为 rval3 是一个 const 引用，不能进行赋值。

合法赋值的作用：

(a)将一个 double 型字面值赋给 int 型变量 ival，发生隐式类型转换，ival 得到的值为 3。

(b)将 int 值 1 赋给变量 ival。

(c)将 int 值 1 赋给变量 ival。

习题 2.26

(a)中的定义和(b)中的赋值存在哪些不同？哪些是非法的？

```
(a)  int ival = 0;              (b)  ival = ri;
     const int &ri = 0;              ri = ival;
```

【解答】

int ival = 0;　　　　定义 ival 为 int 变量，并将其初始化为 0。

const int &ri = 0;　　定义 ri 为 const 引用，并将其绑定到右值 0。

ival = ri;　　　　将 0 值赋给 ival。

ri = ival;　　　　试图对 ri 赋值，这是非法的，因为 ri 是 const 引用，不能赋值。

习题 2.27

下列代码输出什么？

```
int i, &ri = i;

i = 5; ri =10;

std::cout << i << " " << ri << std::endl;
```

【解答】

输出：

10 10

ri 是 i 的引用，对 ri 进行赋值，实际上相当于对 i 进行赋值，所以输出 i 和 ri 的值均为 10。

习题 2.28

编译以下程序，确定你的编译器是否会警告遗漏了类定义后面的分号。

```
class Foo {
```

```
    // empty
}   // Note: no semicolon
int main()
{
    return 0;
}
```

如果编译器的诊断结果难以理解，记住这些信息以备后用。

【解答】

在笔者所用的编译器中编译上述程序，编译器会给出如下错误信息：

error C2628: 'Foo' followed by 'int' is illegal (did you forget a ';'?) （第 4 行）

warning C4326: return type of 'main' should be 'int or void' instead of 'Foo'　（第 5 行）

error C2440: 'return' : cannot convert from 'int' to 'Foo'　（第 6 行）

也就是说，该编译器会对遗漏了类定义后面的分号给出提示。

习题 2.29

区分类中的 public 部分和 private 部分。

【解答】

类中 public 部分定义的成员在程序的任何部分都可以访问。通常在 public 部分放置操作，以便程序中的其他部分可以执行这些操作。

类中 private 部分定义的成员只能被作为类的组成部分的代码（以及该类的友元）访问。通常在 private 部分放置数据，以对对象的内部数据进行隐藏。

习题 2.30

定义表示下列类型的类的数据成员：

(a) 电话号码　　　　(b) 地址

(c) 员工或公司 (d) 某大学的学生

【解答】

(a) 电话号码

```
class Tel_number {
public:
    //...对象上的操作
private:
    std::string country_number;
    std::string city_number;
    std::string phone_number;
};
```

(b) 地址

```
class Address {
public:
    //...对象上的操作
private:
    std::string country;
    std::string city;
    std::string street;
    std::string number;
};
```

(c) 员工或公司

```
class Employee {
public:
```

```
    // ...对象上的操作
private:
    std::string ID;
    std::string name;
    char sex;
    Address addr;
    Tel_number tel;
};
class Company {
public:
    // ...对象上的操作
private:
    std::string name;
    Address addr;
    Tel_number tel;
};
```

(d) 某大学的学生

```
class Student {
public:
    // ...对象上的操作
private:
    std::string ID;
    std::string name;
    char sex;
```

```
        std::string dept; // 所在系
        std::string major;
        Address home_addr;
        Tel_number tel;
    };
```

注意，在不同的具体应用中，类的设计会有所不同，这里给出的只是一般性的简单例子。

习题 2.31

判别下列语句哪些是声明，哪些是定义，请解释原因。

(a) extern int ix = 1024 ;

(b) int iy ;

(c) extern int iz ;

(d) extern const int &ri ;

【解答】

(a)是定义，因为 extern 声明进行了初始化。

(b)是定义，变量定义的常规形式。

(c)是声明，extern 声明的常规形式。

(d)是声明，声明了一个 const 引用。

习题 2.32

下列声明和定义哪些应该放在头文件中？哪些应该放在源文件中？请解释原因。

(a) int var ;

(b) const double pi = 3.1416;

(c) extern int total = 255 ;

(d) const double sq2 = squt (2.0) ;

【解答】

(a)、(c)、(d)应放在源文件中，因为(a)和(c)是变量定义，定义通常应放在源文件中。(d)中的 const 变量 sq2 不是用常量表达式初始化的，所以也应该放在源文件中。

(b)中的 const 变量 pi 是用常量表达式初始化的，应该放在头文件中。

参见 2.9.1 节。

习题 2.33

确定你的编译器提供了哪些提高警告级别的选项。使用这些选项重新编译以前选择的程序，查看是否会报告新的问题。

【解答】

在笔者所用的编译器（Microsoft Visual C++ .NET 2003）中，在 Project 菜单中选择 Properties 菜单项，在 Configuration Properties→C/C++→General→Warning Level 中可以选择警告级别。

习题 3.1

用适当的 using 声明，而不用 std::前缀，访问标准库中的名字，重新编写 2.3 节的程序，计算一给定数的给定次幂的结果。

【解答】

```
#include <iostream>

using std::cin;

using std::cout;

int main()

{

    // 局部对象

    int base, exponent;

    long result=1;
```

```
    // 读入底数和指数
    cout << "Enter base and exponent:" << endl;
    cin >> base >> exponent;
    if (exponent < 0) {
        cout << "Exponent can't be smaller than 0" << endl;
        return -1;
    }
    if (exponent > 0) {
        // 计算底数的指数次方
        for (int cnt = 1; cnt <= exponent; ++cnt)
            result *= base;
    }
    cout << base
         << " raised to the power of "
         << exponent << ": "
         << result << endl;
    return 0;
}
```

习题 3.2

什么是默认构造函数？

【解答】

默认构造函数（default constructor）就是在没有显式提供初始化式时调用的构造函数。它由不带参数的构造函数，或者为所有形参提供默认实参的构造函数定义。如果定义某个类的变量时没有提供初始化式，就会使用默认构造函数。

如果用户定义的类中没有显式定义任何构造函数，编译器就会自动为该类生成默认构造函数，称为合成的默认构造函数（synthesized default constructor）。

习题 3.3

列举出三种初始化 string 对象的方法。

【解答】

(1) 不带初始化式，使用默认构造函数初始化 string 对象。

(2) 使用一个已存在的 string 对象作为初始化式，将新创建的 string 对象初始化为已存在对象的副本。

(3) 使用字符串字面值作为初始化式，将新创建的 string 对象初始化为字符串字面值的副本。

习题 3.4

s 和 s2 的值分别是什么？

```
string s;

int main() {

string s2;

}
```

【解答】

s 和 s2 的值均为空字符串。

习题 3.5

编写程序实现从标准输入每次读入一行文本。然后改写程序，每次读入一个单词。

【解答】

```
//从标准输入每次读入一行文本

#include <iostream>

#include <string>
```

```
using namespace std;
int main()
{
    string line;
    // 一次读入一行，直至遇见文件结束符
    while (getline(cin, line))
        cout << line << endl;     // 输出相应行以进行验证
    return 0;
}
```

修改后程序如下：

```
//从标准输入每次读入一个单词
#include <iostream>
#include <string>
using namespace std;
int main()
{
    string word;
    // 一次读入一个单词，直至遇见文件结束符
    while (cin >> word)
        cout << word << endl; // 输出相应单词以进行验证
    return 0;
}
```

注意，一般而言，应该尽量避免使用 using 指示而使用 using 声明（参见 17.2.4 节），因为如果应用程序中使用了多个库，使用 using 指示引入这些库中定义

的名字空间，容易导致名字冲突。但本书中的程序都只使用了标准库，没有使用其他库。使用 using 指示引入名字空间 std 中定义的所有名字不会发生名字冲突。因此为了使得代码更为简洁以节省篇幅，本书的许多代码中都使用了 using 指示 using namespace std;来引入名字空间 std。另外，本题中并未要求输出，加入输出是为了更清楚地表示读入的结果。本书后面部分有些地方与此类似处理，不再赘述。

习题 3.6

解释 string 类型的输入操作符和 getline 函数分别如何处理空白字符。

【解答】

string 类型的输入操作符对空白字符的处理：读取并忽略有效字符（非空白字符）之前所有的空白字符，然后读取字符直至再次遇到空白字符，读取终止（该空白字符仍留在输入流中）。

getline 函数对空白字符的处理：不忽略行开头的空白字符，读取字符直至遇到换行符，读取终止并丢弃换行符（换行符从输入流中去掉但并不存储在 string 对象中）。

习题 3.7

编一个程序读入两个 string 对象，测试它们是否相等。若不相等，则指出两个中哪个较大。接着，改写程序测试它们的长度是否相等，若不相等，则指出两个中哪个较长。

【解答】

测试两个 string 对象是否相等的程序：

```
#include <iostream>

#include <string>

using namespace std;

int main()

{

    string s1, s2;

    // 读入两个 string 对象

    cout << "Enter two strings:" << endl;
```

```
    cin >> s1 >> s2;
    // 测试两个 string 对象是否相等
    if (s1 == s2)
        cout << "They are equal." << endl;
    else if (s1 > s2)
        cout << "\"" << s1 << "\" is bigger than"
             << " \"" << s2 << "\"" << endl;
    else
        cout << "\"" << s2 << "\" is bigger than"
             << " \"" << s1 << "\"" << endl;
    return 0;
}
```

测试两个 string 对象的长度是否相等的程序：

```
#include <iostream>
#include <string>
using namespace std;
int main()
{
    string s1, s2;
    // 读入两个 string 对象
    cout << "Enter two strings:" << endl;
    cin >> s1 >> s2;
    // 比较两个 string 对象的长度
    string::size_type len1, len2;
```

```
    len1 = s1.size();
    len2 = s2.size();
    if (len1 == len2)
        cout << "They have same length." << endl;
    else if (len1 > len2)
        cout << "\"" << s1 << "\" is longer than"
             << " \"" << s2 << "\"" << endl;
    else
        cout << "\"" << s2 << "\" is longer than"
             << " \"" << s1 << "\"" << endl;
    return 0;
}
```

习题 3.8

编一个程序，从标准输入读取多个 string 对象，把它们连接起来存放到一个更大的 string 对象中，并输出连接后的 string 对象。接着，改写程序，将连接后相邻 string 对象以空格隔开。

【解答】

```
#include <iostream>
#include <string>
using namespace std;
int main()
{
    string result_str, str;
    // 读入多个 string 对象并进行连接
```

```
    cout << "Enter strings(Ctrl+Z to end):" << endl;
    while (cin>>str)
        result_str = result_str + str;
    // 输出连接后的 string 对象
    cout << "String equal to the concatenation of these strings is:"
         << endl << result_str << endl;
    return 0;
}
```

改写后的程序：

```
#include <iostream>
#include <string>
using namespace std;
int main()
{
    string result_str, str;
    // 读入多个 string 对象并进行连接
    cout << "Enter strings(Ctrl+Z to end):" << endl;
    cin >> result_str;//读入第一个 string 对象，放到结果对象中
    while (cin>>str)
        result_str = result_str + ' ' + str;
    // 输出连接后的 string 对象
    cout << "String equal to the concatenation of these strings is:"
         << endl << result_str << endl;
    return 0;
```

```
}
```

习题 3.9

下列程序实现什么功能？实现合法吗？如果不合法，说明理由。

```
string s;

cout << s[0] << endl;
```

【解答】

该程序段输出 string 对象 s 所对应字符串的第一个字符。

实现不合法。因为 s 是一个空字符串，其长度为 0，因此 s[0]是无效的。

注意，在一些编译器（如 Microsoft Visual C++ .NET 2003）的实现中，该程序段并不出现编译错误。

习题 3.10

编一个程序，从 string 对象中去掉标点符号。要求输入到程序的字符串必须含有标点符号，输出结果则是去掉标点符号后的 string 对象。

【解答】

```
#include <iostream>

#include <string>

#include <cctype>

using namespace std;

int main()

{

    string s, result_str;

    bool has_punct = false;//用于标记字符串中有无标点

    char ch;

    //输入字符串
```

```
    cout << "Enter a string:" << endl;
    getline(cin, s);
    //处理字符串：去掉其中的标点
    for (string::size_type index = 0; index != s.size(); ++index)
    {
        ch = s[index];
        if (ispunct(ch))
            has_punct = true;
        else
            result_str += ch;
    }
    if (has_punct)
        cout << "Result:" << endl << result_str <<endl;
    else {
        cout << "No punctuation character in the string?!" << endl;
        return -1;
    }
    return 0;
}
```

习题 3.11

下面哪些 vector 定义不正确？

(a) `vector< vector<int> > ivec;`

(b) `vector<string> svec = ivec ;`

(c) `vector<string> svec(10,"null");`

【解答】

(b)不正确。因为 svec 定义为保存 string 对象的 vector 对象，而 ivec 是保存 vector <int>对象的 vector 对象（即 ivec 是 vector 的 vector），二者的元素类型不同，所以不能用 ivec 来初始化 svec。

习题 3.12

下列每个 vector 对象中元素个数是多少？各元素的值是什么？

(a) vector<int> ivec1;

(b) vector<int> ivec2(10);

(c) vector<int> ivec3(10,42);

(d) vector<string> svec1;

(e) vector<string> svec2(10);

(f) vector<string> svec3(10,"hello");

【解答】

(a) 元素个数为 0。

(b) 元素个数为 10，各元素的值均为 0。

(c) 元素个数为 10，各元素的值均为 42。

(d) 元素个数为 0。

(e) 元素个数为 10，各元素的值均为空字符串。

(f) 元素个数为 10，各元素的值均为"hello"。

习题 3.13

读一组整数到 vector 对象，计算并输出每对相邻元素的和。如果读入元素个数为奇数，则提示用户最后一个元素没有求和，并输出其值。然后修改程序：头尾元素两两配对（第一个和最后一个，第二个和倒数第二个，以此类推），计算每对元素的和，并输出。

【解答】

//读一组整数到 vector 对象，计算并输出每对相邻元素的和

```
#include <iostream>
#include <vector>
using namespace std;
int main()
{
    vector<int> ivec;
    int ival;
    // 读入数据到 vector 对象
    cout << "Enter numbers(Ctrl+Z to end):" << endl;
    while (cin>>ival)
        ivec.push_back(ival);
    // 计算相邻元素的和并输出
    if (ivec.size() == 0) {
        cout << "No element?!" << endl;
        return -1;
    }
    cout << "Sum of each pair of adjacent elements in the vector:"
         << endl;
    for (vector<int>::size_type ix = 0; ix < ivec.size()-1;
                              ix = ix + 2) {
        cout << ivec[ix] + ivec[ix+1] << "\t";
        if ( (ix+1) % 6 == 0) // 每行输出 6 个和
            cout << endl;
    }
```

```
    if (ivec.size() % 2 != 0) // 提示最后一个元素没有求和
        cout << endl
             << "The last element is not been summed "
             << "and its value is "
             << ivec[ivec.size()-1] << endl;
    return 0;
}
```

修改后的程序：

```
//读一组整数到 vector 对象，计算首尾配对元素的和并输出
#include <iostream>
#include <vector>
using namespace std;
int main()
{
    vector<int> ivec;
    int ival;
    //读入数据到 vector 对象
    cout << "Enter numbers:" << endl;
    while (cin>>ival)
        ivec.push_back(ival);
    //计算首尾配对元素的和并输出
    if (ivec.size() == 0) {
        cout << "No element?!" << endl;
        return -1;
```

```
    }
    cout << "Sum of each pair of counterpart elements in the vector:"
         << endl;
    vector<int>::size_type cnt = 0;
    for (vector<int>::size_type first = 0, last = ivec.size() - 1;
         first < last; ++first, --last) {
        cout << ivec[first] + ivec[last] << "\t";
        ++cnt;
        if ( cnt % 6 == 0) //每行输出 6 个和
            cout << endl;
    }
    if (first == last) //提示居中元素没有求和
        cout << endl
             << "The center element is not been summed "
             << "and its value is "
             << ivec[first] << endl;
    return 0;
}
```

习题 3.14

读入一段文本到 vector 对象，每个单词存储为 vector 中的一个元素。把 vector 对象中每个单词转化为大写字母。输出 vector 对象中转化后的元素，每 8 个单词为一行输出。

【解答】

```
//读入一段文本到 vector 对象，每个单词存储为 vector 中的一个元素。
```

```
//把 vector 对象中每个单词转化为大写字母。
//输出 vector 对象中转化后的元素，每 8 个单词为一行输出
#include <iostream>
#include <string>
#include <vector>
#include <cctype>
using namespace std;
int main()
{
    vector<string> svec;
    string str;
    // 读入文本到 vector 对象
    cout << "Enter text(Ctrl+Z to end):" << endl;
    while (cin>>str)
        svec.push_back(str);
    //将 vector 对象中每个单词转化为大写字母,并输出
    if (svec.size() == 0) {
        cout << "No string?!" << endl;
        return -1;
    }
    cout << "Transformed elements from the vector:"
         << endl;
    for (vector<string>::size_type ix = 0; ix != svec.size(); ++ix) {
```

```
        for (string::size_type index = 0; index != svec[ix].size();
++index)
            if (islower(svec[ix][index]))
                //单词中下标为 index 的字符为小写字母
                svec[ix][index] = toupper(svec[ix][index]);
        cout << svec[ix] << " ";
        if ((ix + 1) % 8 == 0)//每 8 个单词为一行输出
            cout << endl;
    }
    return 0;
}
```

习题 3.15

下面程序合法吗？如果不合法，如何更正？

```
vector<int> ivec;
ivec[0] = 42;
```

【解答】

不合法。因为 ivec 是空的 vector 对象，其中不含任何元素，而下标操作只能用于获取已存在的元素。

更正：将赋值语句改为语句 ivec.push_back(42);。

习题 3.16

列出三种定义 vector 对象的方法，给定 10 个元素，每个元素值为 42。指出是否还有更好的实现方法，并说明为什么。

【解答】

方法一：

```
vector<int> ivec(10, 42);
```

方法二：

```
vector<int> ivec(10);
for (ix = 0; ix < 10; ++ix)
    ivec[ix] = 42;
```

方法三：

```
vector<int> ivec(10);
for (vector<int>::iterator iter = ivec.begin();
                         iter != ivec.end(); ++iter)
    *iter = 42;
```

方法四：

```
vector<int> ivec;
for (cnt = 1; cnt <= 10; ++cnt)
    ivec.push_back(42);
```

方法五：

```
vector<int> ivec;
vector<int>::iterator iter = ivec.end();
for (int i = 0; i != 10; ++i) {
    ivec.insert(iter, 42);
    iter = ivec.end();
}
```

各种方法都可达到目的，也许最后两种方法更好一些。它们使用标准库中定义的容器操作在容器中增添元素，无需在定义 vector 对象时指定容器的大小，比较灵活而且不容易出错。

习题 3.17

重做 3.3.2 节的习题，用迭代器而不是下标操作来访问 vector 中的元素。

【解答】

重做习题 3.13 如下：

```
//读一组整数到 vector 对象，计算并输出每对相邻元素的和
//使用迭代器访问 vector 中的元素
#include <iostream>
#include <vector>
using namespace std;
int main()
{
    vector<int> ivec;
    int ival;
    //读入数据到 vector 对象
    cout << "Enter numbers(Ctrl+Z to end):" << endl;
    while (cin>>ival)
        ivec.push_back(ival);
    //计算相邻元素的和并输出
    if (ivec.size() == 0) {
        cout << "No element?!" << endl;
        return -1;
    }
    cout << "Sum of each pair of adjacent elements in the vector:"
         << endl;
    vector<int>::size_type cnt = 0;
```

```
    for (vector<int>::iterator iter = ivec.begin();
                                iter < ivec.end()-1;
                                iter = iter + 2) {
        cout << *iter + *(iter+1) << "\t";
        ++cnt;
        if ( cnt % 6 == 0) //每行输出 6 个和
            cout << endl;
    }
    if (ivec.size() % 2 != 0) //提示最后一个元素没有求和
        cout << endl
             << "The last element is not been summed "
              << "and its value is "
             << *(ivec.end()-1) << endl;
    return 0;
}
//读一组整数到 vector 对象，计算首尾配对元素的和并输出
//使用迭代器访问 vector 中的元素
#include <iostream>
#include <vector>
using namespace std;
int main()
{
    vector<int> ivec;
    int ival;
```

```
//读入数据到 vector 对象
cout << "Enter numbers(Ctrl+Z to end):" << endl;
while (cin>>ival)
    ivec.push_back(ival);
//计算首尾配对元素的和并输出
if (ivec.size() == 0) {
    cout << "No element?!" << endl;
    return -1;
}
cout << "Sum of each pair of counterpart elements in the vector:"
     << endl;
vector<int>::size_type cnt=0;
for (vector<int>::iterator first = ivec.begin(),
                         last = ivec.end() - 1;
                         first < last;
                         ++first, --last) {
    cout << *first + *last << "\t";
    ++cnt;
    if ( cnt % 6 == 0) //每行输出 6 个和
        cout << endl;
}
if (first == last) //提示居中元素没有求和
    cout << endl
         << "The center element is not been summed "
```

```
            << "and its value is "
            << *first << endl;
    return 0;
}
```

重做习题 3.14 如下：

```
//读入一段文本到 vector 对象，每个单词存储为 vector 中的一个元素。
//把 vector 对象中每个单词转化为大写字母。
//输出 vector 对象中转化后的元素，每 8 个单词为一行输出。
//使用迭代器访问 vector 中的元素
#include <iostream>
#include <string>
#include <vector>
#include <cctype>
using namespace std;
int main()
{
    vector<string> svec;
    string str;
    //读入文本到 vector 对象
    cout << "Enter text(Ctrl+Z to end):" << endl;
    while (cin>>str)
        svec.push_back(str);
    //将 vector 对象中每个单词转化为大写字母,并输出
    if (svec.size() == 0) {
```

```
        cout << "No string?!" << endl;
        return -1;
    }
    cout << "Transformed elements from the vector:"
         << endl;
    vector<string>::size_type cnt = 0;
    for (vector<string>::iterator iter = svec.begin();
         iter != svec.end(); ++iter) {
        for (string::size_type index = 0; index != (*iter).size();
             ++index)
            if (islower((*iter)[index]))
                //单词中下标为 index 的字符为小写字母
                (*iter)[index] = toupper((*iter)[index]);
        cout << *iter << " ";
        ++cnt;
        if (cnt % 8 == 0)//每 8 个单词为一行输出
            cout << endl;
    }
    return 0;
}
```

习题 3.18

编写程序来创建有 10 个元素的 vector 对象。用迭代器把每个元素值改为当前值的 2 倍。

【解答】

```
//创建有 10 个元素的 vector 对象，
//然后使用迭代器将每个元素值改为当前值的 2 倍
#include <iostream>
#include <vector>
using namespace std;
int main()
{
    vector<int> ivec(10, 20);//每个元素的值均为 20
    //将每个元素值改为当前值的 2 倍
    for (vector<int>::iterator iter = ivec.begin();
                                iter != ivec.end(); ++iter)
        *iter = (*iter)*2;
    return 0;
}
```

习题 3.19

验证习题 3.18 的程序，输出 vector 的所有元素。

【解答】

```
//创建有 10 个元素的 vector 对象，
//然后使用迭代器将每个元素值改为当前值的 2 倍并输出
#include <iostream>
#include <vector>
using namespace std;
int main()
{
```

```
    vector<int> ivec(10, 20);//每个元素的值均为 20
    //将每个元素值改为当前值的 2 倍并输出
    for (vector<int>::iterator iter = ivec.begin();
                                iter != ivec.end(); ++iter) {
        *iter = (*iter)*2;
        cout << *iter << " ";
    }
    return 0;
}
```

习题 3.20

解释一下在上几个习题的程序实现中你用了哪种迭代器，并说明原因。

【解答】

上述几个习题的程序实现中使用了类型分别为 vector<int>::iterator 和 vector <string>::iterator 的迭代器，通过这些迭代器分别访问元素类型为 int 和 string 的 vector 对象中的元素。

习题 3.21

何时使用 const 迭代器？又在何时使用 const_iterator？解释两者的区别。

【解答】

const 迭代器是迭代器常量，该迭代器本身的值不能修改，即该迭代器在定义时需要初始化，而且初始化之后，不能再指向其他元素。若需要指向固定元素的迭代器，则可以使用 const 迭代器。

const_iterator 是一种迭代器类型，对这种类型的迭代器解引用会得到一个指向 const 对象的引用，即通过这种迭代器访问到的对象是常量。该对象不能修改，因此，const_iterator 类型只能用于读取容器内的元素，不能修改元素的值。若只需遍历容器中的元素而无需修改它们，则可以使用 const_iterator。

习题 3.22

如果采用下面的方法来计算 mid 会产生什么结果？

```
vector<int>::iterator mid = (vi.begin() + vi.end())/2;
```

【解答】

将两个迭代器相加的操作是未定义的，因此用这种方法计算 mid 会出现编译错误。

习题 3.23

解释下面每个 bitset 对象包含的位模式：

(a) bitset<64> bitvec(32);

(b) bitset<32> bv(1010101);

(c) string bstr; cin >> bstr; bitset<8> bv(bstr);

【解答】

(a) bitvec 有 64 个二进制位，（位编号从 0 开始）第 5 位置为 1，其余位置均为 0。

(b) bv 有 32 个二进制位，（位编号从 0 开始）第 0、2、4、5、7、8、11、13、14、16、17、18、19 位置为 1，其余位置均为 0。因为十进制数 1010101 对应的二进制数为 00000000000011110110100110110101。

(c) bv 有 8 个二进制位，（位编号从 0 开始）用读入的字符串的从右至左的 8 个字符对 bv 的 0~7 位进行初始化。

习题 3.24

考虑这样的序列 1,2,3,5,8,13,21，并初始化一个将该序列数字所对应的位置设置为 1 的 bitset<32>对象。然后换个方法，给定一个空的 bitset 对象，编写一小段程序把相应的数位设置为 1。

【解答】

bitset<32>对象的初始化：

bitset<32> bv(0x20212e)

方法二：

```
bitset<32> bv;

int x = 0, y = 1, z;
```

```
z = x + y;
while (z <= 21) {
    bv.set(z);
    x = y;
    y = z;
    z = x + y;
}
```

注意，设置为 1 的数位的位编号符合斐波那契数列的规律。

习题 4.1

假设 get_size 是一个没有参数并返回 int 值的函数，下列哪些定义是非法的？为什么？

```
unsigned buf_size = 1024
(a) int ia[buf_size];
(b) int ia[get_size()];
(c) int ia[4*7-14];
(d) char st[11] = "fundamental" ;
```

【解答】

(a)非法，buf_size 是一个变量，不能用于定义数组的维数（维长度）。

(b)非法，get_size()是函数调用，不是常量表达式，不能用于定义数组的维数（维长度）。

(d)非法，存放字符串"fundamental"的数组必须有 12 个元素，st 只有 11 个元素。

习题 4.2

下列数组的值是什么？

```
string sa[10];
```

```
int ia[10];

int main(){

    string   sa2[10];

    int      ia2[10];

}
```

【解答】

sa 和 sa2 为元素类型为 string 的数组，自动调用 string 类的默认构造函数将各元素初始化为空字符串；ia 为在函数体外定义的内置数组，各元素初始化为 0；ia2 为在函数体内定义的内置数组，各元素未初始化，其值不确定。

习题 4.3

下列哪些定义是错误的？

```
(a) int ia[7] = {0, 1, 1, 2, 3, 5, 8};

(b) vector<int> ivec = {0, 1, 1, 2, 3, 5, 8};

(c) int ia2[] = ia;

(d) int ia3[] = ivec;
```

【解答】

(b)错误。vector 对象不能用这种方式进行初始化。

(c)错误。不能用一个数组来初始化另一个数组。

(d)错误。不能用 vector 对象来初始化数组。

习题 4.4

如何初始化数组的一部分或全部元素？

【解答】

定义数组时可使用初始化列表（用花括号括住的一组以逗号分隔的元素初值）来初始化数组的部分或全部元素。如果是初始化全部元素，可以省略定义数组时方括号中给出的数组维数值。如果指定了数组维数，则初始化列表提供的元素个数不能超过维数值。如果数组维数大于列出的元素初值个数，则只初

始化前面的数组元素，剩下的其他元素，若是内置类型则初始化为 0，若是类类型则调用该类的默认构造函数进行初始化。字符数组既可以用一组由花括号括起来、逗号隔开的字符字面值进行初始化，也可以用一个字符串字面值进行初始化。

习题 4.5

列出使用数组而不是 vector 的缺点。

【解答】

与 vector 类型相比，数组具有如下缺点：数组的长度是固定的，而且数组不提供获取其容量大小的 size 操作，也不提供自动添加元素的 push_back 操作。因此，程序员无法在程序运行时知道一个给定数组的长度，而且如果需要更改数组的长度，程序员只能创建一个更大的新数组，然后把原数组的所有元素复制到新数组的存储空间中去。与使用 vector 类型的程序相比，使用内置数组的程序更容易出错且难以调试。

习题 4.6

下面的程序段企图将下标值赋给数组的每个元素，其中在下标操作上有一些错误，请指出这些错误。

```
const size_t array_size = 10 ;

int ia[array_size];

for (size_t ix = 1; ix <= array_size; ++ix)

    ia[ix] = ix ;
```

【解答】

该程序段的错误是：数组下标使用越界。

根据数组 ia 的定义，该数组的下标值应该是 0~9(即 array_size-1)，而不是从 1 到 array_size，因此其中的 for 语句出错，可更正如下：

```
for (size_t ix = 0; ix < array_size; ++ix)

    ia[ix] = ix ;
```

习题 4.7

编写必要的代码将一个数组赋给另一个数组，然后把这段代码改用 vector 实现。考虑如何将一个 vector 赋给另一个 vector。

【解答】

将一个数组赋给另一个数组，就是将一个数组的元素逐个赋值给另一数组的对应元素，可用如下代码实现：

```
int main()
{
    const size_t array_size = 10;
    int ia1[] = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };
    int ia2[array_size];
    for (size_t ix = 0; ix != array_size; ++ix)
        ia2[ix] = ia1[ix];
    return 0;
}
```

将一个 vector 赋给另一个 vector，也是将一个 vector 的元素逐个赋值给另一 vector 的对应元素，可用如下代码实现：

```
//将一个 vector 赋值给另一 vector
//使用迭代器访问 vector 中的元素
#include <vector>
using namespace std;
int main()
{
     vector<int> ivec1(10, 20);//每个元素初始化为 20
    vector<int> ivec2;
    for (vector<int>::iterator iter = ivec1.begin();
                                  iter != ivec1.end();  ++iter)
```

```
        ivec2.push_back(*iter);
    return 0;
}
```

习题 4.8

编写程序判断两个数组是否相等，然后编写一段类似的程序比较两个 vector。

【解答】

判断两个数组是否相等，可用如下程序：

```
//判断两个数组是否相等
#include <iostream>
using namespace std;
int main()
{
    const int arr_size = 6;
    int ia1[arr_size], ia2[arr_size];
    size_t ix;
    //读入两个数组的元素值
    cout << "Enter " << arr_size
         << " numbers for array1:" << endl;
    for (ix = 0; ix != arr_size; ++ix)
        cin >> ia1[ix];
    cout << "Enter " << arr_size
         << " numbers for array2:" << endl;
    for (ix = 0; ix != arr_size; ++ix)
        cin >> ia2[ix];
```

```
    //判断两个数组是否相等
    for (ix = 0; ix != arr_size; ++ix)
        if (ia1[ix] != ia2[ix]) {
            cout << "Array1 is not equal to array2." << endl;
            return 0;
        }
    cout << "Array1 is equal to array2." << endl;
    return 0;
}
```

判断两个 vector 是否相等，可用如下程序：

```
//判断两个 vector 是否相等
//使用迭代器访问 vector 中的元素
#include <iostream>
#include <vector>
using namespace std;
int main()
{
    vector<int> ivec1, ivec2;
    int ival;
    //读入两个 vector 的元素值
    cout << "Enter numbers for vector1(-1 to end):" << endl;
    cin >> ival;
    while (ival != -1) {
        ivec1.push_back(ival);
```

```
        cin >> ival;
    }
    cout << "Enter numbers for vector2(-1 to end):" << endl;
    cin >> ival;
    while (ival != -1) {
        ivec2.push_back(ival);
        cin >> ival;
    }
    //判断两个 vector 是否相等
    if (ivec1.size() != ivec2.size()) //长度不等的 vector 不相等
        cout << "Vector1 is not equal to vector2." << endl;
    else if (ivec1.size() == 0) //长度都为 0 的 vector 相等
        cout << "Vector1 is equal to vector2." << endl;
    else {//两个 vector 长度相等且不为 0
        vector<int>::iterator iter1, iter2;
        iter1 = ivec1.begin();
        iter2 = ivec2.begin();
        while (*iter1 == *iter2 && iter1 != ivec1.end()
                                && iter2 != ivec2.end()) {
            ++iter1;
            ++iter2;
        }
        if (iter1 == ivec1.end())//所有元素都相等
            cout << "Vector1 is equal to vector2." << endl;
```

```
        else
            cout << "Vector1 is not equal to vector2." << endl;
    }
    return 0;
}
```

习题 4.9

编写程序定义一个有 10 个 int 型元素的数组，并以元素在数组中的位置作为各元素的初值。

【解答】

```
//定义一个有 10 个 int 型元素的数组，
//并以元素在数组中的位置（1~10）作为各元素的初值
int main()
{
    const int array_size = 10;
    int ia[array_size];

    for (size_t ix = 0; ix != array_size; ++ix)
        ia[ix] = ix+1;
    return 0;
}
```

习题 4.10

下面提供了两种指针声明的形式，解释宁愿使用第一种形式的原因：

```
int *ip; // good practice
int* ip; // legal but misleading
```

【解答】

第一种形式强调了 ip 是一个指针，这种形式在阅读时不易引起误解，尤其是当一个语句中同时定义了多个变量时。

习题 4.11

解释下列声明语句，并指出哪些是非法的，为什么？

(a) int* ip;

(b) string s, *sp = 0;

(c) int i; double* dp = &i;

(d) int* ip, ip2;

(e) const int i = 0, *p = i;

(f) string *p = NULL;

【解答】

(a)合法。定义了一个指向 int 型对象的指针 ip。

(b)合法。定义了 string 对象 s 和指向 string 型对象的指针 sp，sp 初始化为 0 值。

(c)非法。dp 为指向 double 型对象的指针，不能用 int 型对象 i 的地址进行初始化。

(d)合法。定义了 int 对象 ip2 和指向 int 型对象的指针 ip。

(e)合法。定义了 const int 型对象 i 和指向 const int 型对象的指针 p，i 初始化为 0，p 初始化为 0。

(f)合法。定义了指向 string 型对象的指针 p，并将其初始化为 0 值。

习题 4.12

已知一指针 p，你可以确定该指针是否指向一个有效的对象吗？如果可以，如何确定？如果不可以，请说明原因。

【解答】

无法确定某指针是否指向一个有效对象。因为，在 C++语言中，无法检测指针是否未被初始化，也无法区分一个地址是有效地址，还是由指针所分配的存储空间中存放的不确定值的二进制位形成的地址。

习题 4.13

下列代码中，为什么第一个指针的初始化是合法的，而第二个则不合法？

```
int i = 42;

void *p = &i;

long *lp = &i;
```

【解答】

具有 void*类型的指针可以保存任意类型对象的地址，因此 p 的初始化是合法的；而指向 long 型对象的指针不能用 int 型对象的地址来初始化，因此 lp 的初始化不合法。

习题 4.14

编写代码修改指针的值；然后再编写代码修改指针所指对象的值。

【解答】

下列代码修改指针的值：

```
int *ip;

int ival1, ival2;

ip = &ival1;

ip = &ival2;
```

下列代码修改指针所指对象的值：

```
int ival = 0;

int *ip = &ival;

*ip = 8;
```

习题 4.15

解释指针和引用的主要区别。

【解答】

使用引用（reference）和指针（pointer）都可间接访问另一个值，但它们之间存在两个重要区别：(1)引用总是指向某个确定对象（事实上，引用就是该对象的别名），定义引用时没有进行初始化会出现编译错误；(2) 赋值行为上存在差异：给引用赋值修改的是该引用所关联的对象的值，而不是使该引用与另一个对象关联。引用一经初始化，就始终指向同一个特定对象。给指针赋值修改的是指针对象本身，也就是使该指针指向另一对象，指针在不同时刻可指向不同的对象（只要保证类型匹配）。

习题 4.16

下列程序段实现什么功能？

```
int i = 42, j = 1024;

int *p1 = &i, *p2 = &j;

*p2 = *p1 * * p2;

*p1 *= *p1;
```

【解答】

该程序段使得 i 被赋值为 42 的平方，j 被赋值为 42 与 1024 的乘积。

习题 4.17

已知 p1 和 p2 指向同一个数组中的元素，下面语句实现什么功能？

```
p1 += p2 - p1;
```

当 p1 和 p2 具有什么值时这个语句是非法的？

【解答】

此语句使得 p1 也指向 p2 原来所指向的元素。原则上说，只要 p1 和 p2 的类型相同，则该语句始终是合法的。只有当 p1 和 p2 不是同类型指针时，该语句才不合法（不能进行-操作）。

但是，如果 p1 和 p2 不是指向同一个数组中的元素，则这个语句的执行结果可能是错误的。因为-操作的结果类型 ptrdiff_t 只能保证足以存放同一数组中两个指针之间的差距。如果 p1 和 p2 不是指向同一个数组中的元素，则-操作的结

果有可能超出 ptrdiff_t 类型的表示范围而产生溢出，从而该语句的执行结果不能保证 p1 指向 p2 原来所指向的元素（甚至不能保证 p1 为有效指针）。

习题 4.18

编写程序，使用指针把一个 int 型数组的所有元素设置为 0。

【解答】

```
// 使用指针把一个 int 型数组的所有元素设置为 0
int main()
{
    const size_t arr_size = 8;
    int int_arr[arr_size] = { 0, 1, 2, 3, 4, 5, 6, 7 };
    // pbegin 指向第一个元素，pend 指向最后一个元素的下一内存位置
    for (int *pbegin = int_arr, *pend = int_arr + arr_size;
                                pbegin != pend; ++pbegin)
        *pbegin = 0; // 当前元素置 0
    return 0;
}
```

习题 4.19

解释下列 5 个定义的含义，指出其中哪些定义是非法的：

(a) int i;

(b) const int ic;

(c) const int *pic;

(d) int *const cpi;

(e) const int *const cpic;

【解答】

(a) 合法：定义了 int 型对象 i。

(b) 非法：定义 const 对象时必须进行初始化，但 ic 没有初始化。

(c) 合法：定义了指向 int 型 const 对象的指针 pic。

(d) 非法：因为 cpi 被定义为指向 int 型对象的 const 指针，但该指针没有初始化。

(e) 非法：因为 cpic 被定义为指向 int 型 const 对象的 const 指针，但该指针没有初始化。

习题 4.20

下列哪些初始化是合法的？为什么？

(a) int i = -1;

(b) const int ic = i ;

(c) const int *pic = ⁣

(d) int *const cpi = ⁣

(e) const int *const cpic = ⁣

【解答】

(a) 合法：定义了一个 int 型对象 i，并用 int 型字面值-1 对其进行初始化。

(b) 合法：定义了一个 int 型 const 对象 ic，并用 int 型对象对其进行初始化。

(c) 合法：定义了一个指向 int 型 const 对象的指针 pic，并用 ic 的地址对其进行初始化。

(d) 不合法：cpi 是一个指向 int 型对象的 const 指针，不能用 const int 型对象 ic 的地址对其进行初始化。

(e) 合法：定义了一个指向 int 型 const 对象的 const 指针 cpic，并用 ic 的地址对其进行初始化。

习题 4.21

根据上述定义，下列哪些赋值运算是合法的？为什么？

(a) i = ic;　　　　(b) pic = ⁣

(c) cpi = pic;　　　　(d) pic = cpic;

(e) cpic = ⁣　　　　(f) ic = *cpic;

【解答】

(a)、(b)、(d)合法。

(c)、(e)、(f)均不合法，因为 cpi、cpic 和 ic 都是 const 变量（常量），常量不能被赋值。

习题 4.22

解释下列两个 while 循环的差别：

```
const char *cp = "hello";

int cnt;

while (cp) { ++cnt; ++cp; }

while (*cp) { ++cnt; ++cp; }
```

【解答】

两个 while 循环的差别为：前者的循环结束条件是 cp 为 0 值（即指针 cp 为 0 值）；后者的循环结束条件是 cp 所指向的字符为 0 值（即 cp 所指向的字符为字符串结束符 null（即'\0'））。因此后者能正确地计算出字符串"hello"中有效字符的数目（放在 cnt 中），而前者的执行是不确定的。

注意，题目中的代码还有一个小问题，即 cnt 没有初始化为 0 值。

习题 4.23

下列程序实现什么功能？

```
const char ca[] = {'h', 'e', 'l', 'l', 'o'};

const char *cp = ca ;

while (*cp) {

    cout << *cp << endl;
```

```
    ++cp;
}
```

【解答】

该程序段从数组 ca 的起始地址（即字符'h'的存储地址）开始，输出一段内存中存放的字符，每行输出一个字符，直至存放 0 值（null）的字节为止。（注意，输出的内容一般来说要多于 5 个字符，因为字符数组 ca 中没有 null 结束符。）

习题 4.24

解释 strcpy 和 strncpy 的差别在哪里，各自的优缺点是什么？

【解答】

strcpy 和 strncpy 的差别在于：前者复制整个指定的字符串，后者只复制指定字符串中指定数目的字符。

strcpy 比较简单，而使用 strncpy 可以适当地控制复制字符的数目，因此比 strcpy 更为安全。

习题 4.25

编写程序比较两个 string 类型的字符串，然后编写另一个程序比较两个 C 风格字符串的值。

【解答】

比较两个 string 类型的字符串的程序如下：

```
//比较两个 string 类型的字符串
#include <iostream>
#include <string>
using namespace std;
int main()
{
    string str1, str2;
```

```
    //输入两个字符串
    cout << "Enter two strings:" << endl;
    cin >> str1 >> str2;
    //比较两个字符串
    if (str1 > str2)
        cout << "\"" << str1 << "\"" << " is bigger than "
             << "\"" << str2 << "\"" << endl;
    else if (str1 < str2)
        cout << "\"" << str2 << "\"" << " is bigger than "
             << "\"" << str1 << "\"" << endl;
    else
        cout << "They are equal" << endl;
    return 0;
}
```

比较两个 C 风格字符串的程序如下：

```
//比较两个 C 风格字符串的值
#include <iostream>
#include <cstring>
using namespace std;
int main()
{
    //char *str1 = "string1", *str2 = "string2";
    const int str_size = 80;
    char *str1, *str2;
```

```
//为两个字符串分配内存
str1 = new char[str_size];
str2 = new char[str_size];
if (str1 == NULL || str2 == NULL) {
    cout << "No enough memory!" << endl;
    return -1;
}
//输入两个字符串
cout << "Enter two strings:" << endl;
cin >> str1 >> str2;
//比较两个字符串
int result;
result = strcmp(str1, str2);
if (result > 0)
    cout << "\"" << str1 << "\"" << " is bigger than "
          << "\"" << str2 << "\"" << endl;
else if (result < 0)
    cout << "\"" << str2 << "\"" << " is bigger than "
          << "\"" << str1 << "\"" << endl;
else
    cout << "They are equal" << endl;
//释放字符串所占用的内存
delete [] str1 ;
delete [] str2 ;
```

```
    return 0;
}
```

注意，此程序中使用了内存的动态分配与释放（见 4.3.1 节）。如果不用内存的动态分配与释放，可将主函数中第 2、3 两行代码、有关内存分配与释放的代码以及输入字符串的代码注释掉，再将主函数中第一行代码

```
//char *str1 = "string1", *str2 = "string2";
```

前的双斜线去掉即可。

习题 4.26

编写程序从标准输入设备读入一个 string 类型的字符串。考虑如何编程实现从标准输入设备读入一个 C 风格字符串。

【解答】

从标准输入设备读入一个 string 类型字符串的程序段：

```
string str;
cin >> str;
```

从标准输入设备读入一个 C 风格字符串可如下实现：

```
const int str_size = 80;
char str[str_size];
cin >> str;
```

习题 4.27

假设有下面的 new 表达式，请问如何释放 pa？

```
int *pa = new int[10];
```

【解答】

用语句 delete [] pa;释放 pa 所指向的数组空间。

习题 4.28

编写程序由从标准输入设备读入的元素数据建立一个 int 型 vector 对象，然后动态创建一个与该 vector 对象大小一致的数组，把 vector 对象的所有元素复制给新数组。

【解答】

```
// 从标准输入设备读入的元素数据建立一个 int 型 vector 对象，
// 然后动态创建一个与该 vector 对象大小一致的数组，
// 把 vector 对象的所有元素复制给新数组
#include <iostream>
#include <vector>
using namespace std;
int main()
{
    vector<int> ivec;
    int ival;
    //读入元素数据并建立 vector
    cout << "Enter numbers:(Ctrl+Z to end)" << endl;
    while (cin >> ival)
        ivec.push_back(ival);
    //动态创建数组
    int *pia = new int[ivec.size()];
    //复制元素
    int *tp = pia;
    for (vector<int>::iterator iter = ivec.begin();
                                iter != ivec.end(); ++iter, ++tp)
```

```
        *tp = *iter;

    //释放动态数组的内存

    delete [] pia;

    return 0;

}
```

习题 4.29

对本节第 5 条框中的两段程序：

(a) 解释这两段程序实现的功能。

(b) 平均来说，使用 string 类型的程序执行速度要比用 C 风格字符串的快很多，在我们用了 5 年的 PC 机上其平均执行速度分别是：

user 0.47 # string class

user 2.55 # C-style character string

你预计的也一样吗？请说明原因。

【解答】

(a) 这两段程序的功能是：执行一个循环次数为 1000000 的循环，在该循环的循环体中：创建一个新字符串，将一个已存在的字符串复制给新字符串，然后比较两个字符串，最后释放新字符串。

(b) 使用 C 风格字符串的程序需要自己管理内存的分配和释放，而使用 string 类型的程序由系统自动进行内存的分配和释放，因此比使用 C 风格字符串的程序要简短，执行速度也要快一些。

习题 4.30

编写程序连接两个 C 风格字符串字面值，把结果存储在一个 C 风格字符串中。然后再编写程序连接两个 string 类型字符串，这两个 string 类型字符串与前面的 C 风格字符串字面值具有相同的内容。

【解答】

连接两个 C 风格字符串字面值的程序如下：

```
// 连接两个 C 风格字符串字面值，
```

```
// 把结果存储在一个 C 风格字符串中
#include <cstring>
int main()
{
    const char *cp1 = "Mary and Linda ";
    const char *cp2 = "are firends.";
    size_t len = strlen(cp1) + strlen(cp2);
    char *result_str = new char[len+1];
    strcpy(result_str, cp1);
    strcat(result_str, cp2);
    delete [] result_str;
    return 0;
}
```

相应的连接两个 string 类型字符串的程序如下：

```
// 连接两个 string 类型字符串
#include <string>
using namespace std;
int main()
{
    const string str1("Mary and Linda ");
    const string str2("are firends.");
    string result_str;
    result_str = str1;
    result_str += str2;
```

```
    return 0;
}
```

习题 4.31

编写程序从标准输入设备读入字符串，并把该串存放在字符数组中。描述你的程序如何处理可变长的输入。提供比你分配的数组长度长的字符串数据测试你的程序。

【解答】

```
// 从标准输入设备读入字符串，并把该串存放在字符数组中
#include <iostream>
#include <string>
#include <cstring>
using namespace std;
int main()
{
    string in_str;// 用于读入字符串的 string 对象
    const size_t str_size = 10;
    char result_str[str_size+1];
    // 读入字符串
    cout << "Enter a string(<=" << str_size
         << " characters):" << endl;
    cin >> in_str;
    // 计算需复制的字符的数目
    size_t len = strlen(in_str.c_str());
    if (len > str_size) {
```

```
        len = str_size;
        cout << "String is longer than " << str_size
             << " characters and is stored only "
             << str_size << " characters!" << endl;
    }
    // 复制 len 个字符至字符数组 result_str
    strncpy(result_str, in_str.c_str(), len);
    // 在末尾加上一个空字符（null 字符）
    result_str[len+1] = '\0';
    return 0;
}
```

为了接受可变长的输入，程序中用一个 string 对象存放读入的字符串，然后使用 strncpy 函数将该对象的适当内容复制到字符数组中。因为字符数组的长度是固定的，因此首先计算字符串的长度。若该长度小于或等于字符数组可容纳字符串的长度，则复制整个字符串至字符数组，否则，根据数组的长度，复制字符串中前面部分的字符，以防止溢出。

注意，上述给出的是满足题目要求的一个解答，事实上，如果希望接受可变长的输入并完整地存放到字符数组中，可以采用动态创建数组来实现。

习题 4.32

编写程序用 int 型数组初始化 vector 对象。

【解答】

```
// 用 int 型数组初始化 vector 对象
#include <iostream>
#include <vector>
using namespace std;
int main()
```

```
{
    const size_t arr_size = 8;
    int int_arr[arr_size];
    // 输入数组元素
    cout << "Enter " << arr_size << " numbers:" << endl;
    for (size_t ix = 0; ix != arr_size; ++ix)
        cin >> int_arr[ix];
    // 用 int 型数组初始化 vector 对象
    vector<int> ivec(int_arr, int_arr + arr_size);
    return 0;
}
```

习题 4.33

编写程序把 int 型 vector 复制给 int 型数组。

【解答】

```
// 把 int 型 vector 复制给 int 型数组
#include <iostream>
#include <vector>
using namespace std;
int main()
{
    vector<int> ivec;
    int ival;
    // 输入 vector 元素
    cout << "Enter numbers: (Ctrl+Z to end)" << endl;
```

```
    while (cin >> ival)
        ivec.push_back(ival);
    // 创建数组
    int *parr = new int[ivec.size()];
    // 复制元素
    size_t ix = 0;
    for (vector<int>::iterator iter = ivec.begin();
                          iter != ivec.end(); ++iter, ++ix)
        parr[ix] = *iter;
    // 释放数组
    delete [] parr;
    return 0;
}
```

习题 4.34

编写程序读入一组 string 类型的数据，并将它们存储在 vector 中。接着，把该 vector 对象复制给一个字符指针数组。为 vector 中的每个元素创建一个新的字符数组，并把该 vector 元素的数据复制到相应的字符数组中，最后把指向该数组的指针插入字符指针数组。

【解答】

```
//4-34.cpp
//读入一组 string 类型的数据，并将它们存储在 vector 中。
//接着，把该 vector 对象复制给一个字符指针数组。
//为 vector 中的每个元素创建一个新的字符数组，
//并把该 vector 元素的数据复制到相应的字符数组中，
//最后把指向该数组的指针插入字符指针数组
```

```
#include <iostream>

#include <vector>

#include <string>

using namespace std;

int main()

{

    vector<string> svec;

    string str;

    // 输入 vector 元素

    cout << "Enter strings:(Ctrl+Z to end)" << endl;

    while (cin >> str)

        svec.push_back(str);

    // 创建字符指针数组

    char **parr = new char*[svec.size()];


    // 处理 vector 元素

    size_t ix = 0;

    for (vector<string>::iterator iter = svec.begin();

              iter != svec.end(); ++iter, ++ix) {

        // 创建字符数组

        char *p = new char[(*iter).size()+1];

        // 复制 vector 元素的数据到字符数组

        strcpy(p, (*iter).c_str());

        // 将指向该字符数组的指针插入到字符指针数组
```

```
        parr[ix] = p;
    }
    // 释放各个字符数组
    for (ix =0; ix != svec.size(); ++ix)
        delete [] parr[ix];
    // 释放字符指针数组
    delete [] parr;
    return 0;
}
```

习题 4.35

输出习题 4.34 中建立的 vector 对象和数组的内容。输出数组后，记得释放字符数组。

【解答】

```
//4-35.cpp
//读入一组 string 类型的数据，并将它们存储在 vector 中。
//接着，把该 vector 对象复制给一个字符指针数组：
//为 vector 中的每个元素创建一个新的字符数组，
//并把该 vector 元素的数据复制到相应的字符数组中，
//然后把指向该数组的指针插入字符指针数组。
//输出建立的 vector 对象和数组的内容
#include <iostream>
#include <vector>
#include <string>
using namespace std;
```

```
int main()
{
    vector<string> svec;
    string str;
    // 输入 vector 元素
    cout << "Enter strings:(Ctrl+Z to end)" << endl;
    while (cin >> str)
        svec.push_back(str);
    // 创建字符指针数组
    char **parr = new char*[svec.size()];
    // 处理 vector 元素
    size_t ix = 0;
    for (vector<string>::iterator iter = svec.begin();
              iter != svec.end(); ++iter, ++ix) {
        // 创建字符数组
        char *p = new char[(*iter).size()+1];
        // 复制 vector 元素的数据到字符数组
        strcpy(p, (*iter).c_str());
        // 将指向该字符数组的指针插入到字符指针数组
        parr[ix] = p;
    }
    // 输出 vector 对象的内容
    cout << "Content of vector:" << endl;
    for (vector<string>::iterator iter2 = svec.begin();
```

```
                    iter2 != svec.end(); ++iter2)
          cout << *iter2 << endl;
     // 输出字符数组的内容
     cout << "Content of character arrays:" << endl;
     for (ix =0; ix != svec.size(); ++ix)
          cout << parr[ix] << endl;
     // 释放各个字符数组
     for (ix =0; ix != svec.size(); ++ix)
          delete [] parr[ix];
     // 释放字符指针数组
     delete [] parr;
     return 0;
}
```

习题 4.36

重写程序输出 ia 数组的内容，要求在外层循环中不能使用 typedef 定义的类型。

【解答】

```
//4-36.cpp
//重写程序输出 ia 数组的内容
//在外层循环中不使用 typedef 定义的类型
#include <iostream>
using namespace std;
int main()
{
```

```
    int ia[3][4] = {      // 3 个元素，每个元素是一个有 4 个 int 元素的数组
        {0, 1, 2, 3} , // 0 行的初始化列表
        {4, 5, 6, 7} , // 1 行的初始化列表
        {8, 9, 10, 11} // 2 行的初始化列表
    };
    int (*p)[4];
    for (p = ia; p != ia + 3; ++p)
        for (int *q = *p; q != *p + 4; ++q)
            cout << *q << endl;
    return 0;
}
```

习题 5.1

在下列表达式中，加入适当的圆括号以标明其计算顺序。编译该表达式并输出其值，从而检查你的回答是否正确。

12 / 3 * 4 + 5 * 15 + 24 % 4 / 2

【解答】

加入如下所示的圆括号以标明该表达式的计算顺序：

(((12 / 3) * 4) + (5 * 15)) + ((24 % 4) / 2)

习题 5.2

计算下列表达式的值，并指出哪些结果值依赖于机器？

-30 * 3 + 21 / 5

-30 + 3 * 21 / 5

30 / 3 * 21 % 5

-30 / 3 * 21 % 4

【解答】

各表达式的值分别为**-86**、**-18**、**0**、**-2**。其中，最后一个表达式的结果值依赖于机器，因为该表达式中除操作只有一个操作数为负数。

习题 5.3

编写一个表达式判断一个 int 型数值是偶数还是奇数。

【解答】

如下表达式可以判断一个 **int** 型数值（假设为 **ival**）是偶数还是奇数：

ival % 2 == 0

若 **ival** 是偶数，则该表达式的值为真（**true**），否则为假（**false**）。

习题 5.4

定义术语“溢出”的含义，并给出导致溢出的三个表达式。

【解答】

溢出：表达式的求值结果超出了其类型的表示范围。

如下表达式会导致溢出（假设 **int** 类型为 **16** 位）：

1000 * 1000

32766 + 5

3276 * 20

在这些表达式中，各操作数均为 **int** 类型，因此这些表达式的类型也是 **int**，但它们的计算结果均超出了 **16** 位 **int** 型的表示范围（**-32768~32767**），导致溢出。

习题 5.5

解释逻辑与操作符、逻辑或操作符以及相等操作符的操作数在什么时候计算。

【解答】

逻辑与、逻辑或操作符采用称为“短路求值”（**short-circuit evaluation**）的求值策略，即先计算左操作数，再计算右操作数，且只有当仅靠左操作数的值无法确定该逻辑运算的结果时，才会计算右操作数。

相等操作符的左右操作数均需进行计算。

习题 5.6

解释下列 while 循环条件的行为：

char *cp = "Hello World" ;

while (cp && *cp)

【解答】

该 **while** 循环的条件为：当指针 **cp** 为非空指针并且 **cp** 所指向的字符不为空字符 **null**（**'\0'**）时执行循环体。即该循环可以对字符串**"Hello World"**中的字符进行逐个处理。

习题 5.7

编写 while 循环条件从标准输入设备读入整型(int)数据，当读入值为 42 时循环结束。

【解答】

int val;

cin >> val;

while (val != 42)

或者，**while** 循环条件也可以写成

while (cin >> ival && ival != 42)

习题 5.8

编写表达式判断 4 个值 a、b、c 和 d 是否满足 a 大于 b、b 大于 c 而且 c 大于 d 的条件。

【解答】

表达式如下：

a > b && b > c && c > d

习题 5.9

假设有下面两个定义：

unsigned long ul1 =3, ul2 = 7;

下列表达式的结果是什么？

(a) ul1 & ul2 **(b) ul1 && ul2**

(c) ul1 | ul2 **(d) ul1 || ul2**

【解答】

各表达式的结果分别为 **3**、**true**、**7**、**true**。

习题 5.10

重写 bitset 表达式：使用下标操作符对测验结果进行置位（置 1）和复位（置 0）。

【解答】

```
bitset<30> bitset_quiz1;

bitset_quiz1[27] = 1;

bitset_quiz1[27] = 0;
```

习题 5.11

请问每次赋值操作完成后，i 和 d 的值分别是多少？

```
int i; double d;

d = i = 3.5;

i = d = 3.5;
```

【解答】

赋值语句 **d=i=3.5;**完成后，**i** 和 **d** 的值均为 **3**。因为赋值操作具有右结合性，所以首先将 **3.5** 赋给 **i**（此时发生隐式类型转换，将 **double** 型字面值 **3.5** 转换为 **int** 型值 **3**，赋给 **i**），然后将表达式 **i=3.5** 的值（即赋值后 **i** 所具有的值 **3**）赋给 **d**。

赋值语句 **i=d=3.5;**完成后，**d** 的值为 **3.5**，**i** 的值为 **3**。因为先将字面值 **3.5** 赋给 **d**，然后将表达式 **d=3.5** 的值（即赋值后 **d** 所具有的值 **3.5**）赋给 **i**（这时也同样发生隐式类型转换）。

习题 5.12

解释每个 if 条件判断产生什么结果？

```
if ( 42 = i ) // ...

if ( i = 42 ) // ...
```

【解答】

前者发生语法错误，因为其条件表达式 42=i 是一个赋值表达式，赋值操作符的左操作数必须为一个左值，而字面值 42 不能作为左值使用。

后者代码合法，但其条件表达式 i=42 是一个永真式（即其逻辑值在任何情况下都为 true），因为该赋值表达式的值为赋值操作完成后的 i 值（42），而 42 为非零值，解释为逻辑值 true。

习题 5.13

下列赋值操作是不合法的，为什么？怎样改正？

```
double dval; int ival; int *pi;

dval = ival = pi = 0;
```

【解答】

该赋值语句不合法，因为该语句首先将 0 值赋给 pi，然后将 pi 的值赋给 ival，再将 ival 的值赋给 dval。pi、ival 和 dval 的类型各不相同，因此要完成赋值必须进行隐式类型转换，但系统无法将 int 型指针 pi 的值隐式转换为 ival 所需的 int 型值。

可改正如下：

```
double dval; int ival; int *pi;

dval = ival = 0;

pi = 0;
```

习题 5.14

虽然下列表达式都是合法的，但并不是程序员期望的操作，为什么？怎样修改这些表达式以使其能反映程序员的意图？

```
(a) if ( ptr = retrieve_pointer() != 0 )

(b) if ( ival = 1024 )

(c) ival += ival + 1;
```

【解答】

对于表达式(a)，程序员的意图应该是将 retrieve_pointer()的值赋给 ptr，然后判断 ptr

的值是否为 **0**，但因为操作符“=”的优先级比“!=”低，所以该表达式实际上是将 **retrieve_pointer()**是否为 **0** 的判断结果 **true** 或 **false** 赋给 **ptr**，因此不是程序员期望的操作。

对于表达式(**b**)，程序员的意图应该是判断 **ival** 的值是否与 **1024** 相等，但误用了赋值操作符。

对于表达式(**c**)，程序员的意图应该是使 **ival** 的值增加 **1**，但误用了操作符“+=”。

各表达式可修改如下：

(a) if ((ptr = retrieve_pointer()) != 0)

(b) if (ival == 1024)

(c) ival += 1; 或 **ival++;** 或 **++ival;**

习题 5.15

解释前自增操作和后自增操作的差别。

【解答】

前自增操作和后自增操作都使其操作数加 **1**，二者的差别在于：前自增操作将修改后操作数的值作为表达式的结果值；而后自增操作将操作数原来的、未修改的值作为表达式的结果值。

习题 5.16

你认为为什么 C++不叫作++C？

【解答】

C++之名是 **Rick Mascitti** 在 **1983** 年夏天定名的（参见 ***The C++ Programming Language(Special Edition)*** **1.4** 节），**C** 说明它本质上是从 **C** 语言演化而来的，“++”是 **C** 语言的自增操作符。**C++**语言是 **C** 语言的超集，是在 **C** 语言基础上进行的扩展（引入了 **new**、**delete** 等 **C** 语言中没有的操作符，增加了对面向对象程序设计的直接支持，等等），是先有 **C** 语言，再进行++。根据自增操作符前、后置形式的差别（参见习题 **5.15** 的解答），**C++**表示对 **C** 语言进行扩展之后，还可以使用 **C** 语言的内容；而写成++**C** 则表示无法再使用 **C** 的原始值了，也就是说 **C++**不能向下兼容 **C** 了，这与实际情况不符。

习题 5.17

如果输出 vector 内容的 while 循环使用前自增操作符，那会怎么样？

【解答】

将导致错误的结果：**ivec** 的第一个元素没有输出，并企图对一个多余的元素进行解引用。

习题 5.18

编写程序定义一个 vector 对象，其每个元素都是指向 string 类型的指针，读取该 vector 对象，输出每个 string 的内容及其相应的长度。

【解答】

```
//定义一个 vector 对象，其每个元素都是指向 string 类型的指针，
//读取该 vector 对象，输出每个 string 的内容及其相应的长度
#include <iostream>
#include <string>
#include <vector>
using namespace std;
int main()
{
    vector<string*> spvec;
    //读取 vector 对象
    string str;
    cout << "Enter some strings(Ctrl+Z to end)" << endl;
    while (cin >> str) {
        string *pstr = new string; //指向 string 对象的指针
        *pstr = str;
        spvec.push_back(pstr);
    }
    //输出每个 string 的内容及其相应的长度
    vector<string*>::iterator iter = spvec.begin();
    while (iter != spvec.end()) {
```

```
        cout << **iter << (**iter).size() << endl;
        iter++;
    }
    //释放各个动态分配的 string 对象
    iter = spvec.begin();
    while (iter != spvec.end()) {
        delete *iter;
        iter++;
    }
    return 0;
}
```

习题 5.19

假设 iter 为 vector<string>::iterator 类型的变量，指出下面哪些表达式是合法的，并解释这些合法表达式的行为。

(a) *iter++;　　　　(b) (*iter)++;

(c) *iter.empty();　　(d) iter->empty();

(e) ++*iter;　　　　(f) iter++->empty();

【解答】

(a)、(d)、(f)合法。

这些表达式的执行结果如下：

(a)返回 iter 所指向的 string 对象，并使 iter 加 1。

(d)调用 iter 所指向的 string 对象的成员函数 empty。

(f)调用 iter 所指向的 string 对象的成员函数 empty，并使 iter 加 1。

习题 5.20

编写程序提示用户输入两个数，然后报告哪个数比较小。

【解答】

可编写程序如下：

```
//提示用户输入两个数，然后报告哪个数比较小
#include <iostream>
using namespace std;
int main()
{
    int val1, val2;
    //提示用户输入两个数并接受输入
    cout << "Enter two integers:" << endl;
    cin >> val1 >> val2;
    //报告哪个数比较小
    cout << "The smaller one is"
         << (val1 < val2 ? val1 : val2) << endl;
    return 0;
}
```

习题 5.21

编写程序处理 vector<int>对象的元素：将每个奇数值元素用该值的两倍替换。

【解答】

```
//处理 vector<int>对象的元素：
//将每个奇数值元素用该值的两倍替换
#include <iostream>
#include <vector>
```

```
using namespace std;
int main()
{
    vector<int> ivec(20,1);//ivec 包含 20 个值为 1 的元素
    //将每个奇数值元素用该值的两倍替换
    for (vector<int>::iterator iter = ivec.begin();
         iter != ivec.end(); ++iter)
        *iter = (*iter % 2 == 0 ? *iter : *iter * 2);
    return 0;
}
```

习题 5.22

编写程序输出每种内置类型的长度。

【解答】

```
//输出每种内置类型的长度
#include <iostream>
using namespace std;
int main()
{
    cout << "type\t\t\t" << "size" << endl
         << "bool\t\t\t" << sizeof(bool) << endl
         << "char\t\t\t" << sizeof(char) << endl
         << "signed char\t\t" << sizeof(signed char) << endl
         << "unsigned char\t\t" << sizeof(unsigned char) << endl
         << "wchar_t\t\t\t" << sizeof(wchar_t) << endl
```

```
        << "short\t\t\t" << sizeof(short) << endl
        << "signed short\t\t" << sizeof(signed short) << endl
        << "unsigned short\t\t" << sizeof(unsigned short) << endl
        << "int\t\t\t" << sizeof(int) << endl
        << "signed int\t\t" << sizeof(signed int) << endl
        << "unsigend int\t\t" << sizeof(unsigned int) << endl
        << "long\t\t\t" << sizeof(long) << endl
        << "sigend long\t\t" << sizeof(signed long) << endl
        << "unsigned long\t\t" << sizeof(unsigned long) << endl
        << "float\t\t\t" << sizeof(float) << endl
        << "double\t\t\t" << sizeof(double) << endl
        << "long double\t\t" << sizeof(long double) << endl;
    return 0;
}
```

习题 5.23

预测下列程序的输出，并解释你的理由。然后运行该程序，输出的结果和你预测的一样吗？如果不一样，为什么？

```
int x[10]; int *p = x;
cout << sizeof(x)/sizeof(*x) << endl;
cout << sizeof(p)/sizeof(*p) << endl;
```

【解答】

在表达式 sizeof(x)中，x 是数组名，该表达式的结果为数组 x 所占据的存储空间的字节数，为 10 个 int 型元素所占据的字节数。

表达式 sizeof(*x)的结果是指针常量 x 所指向的对象（数组中第一个 int 型元素）所占据的存储空间的字节数。

表达式 sizeof(p)的结果是指针变量 p 所占据的存储空间的字节数。

表达式 sizeof(*p)的结果是指针变量 p 所指向的对象（一个 int 型数据）所占据的存储空间的字节数。

各种数据类型在不同的系统中所占据的字节数不一定相同，因此在不同的系统中运行上述程序段得到的结果不一定相同。在 Microsoft Visual C++ .NET 2003 系统中，一个 int 型数据占据 4 个字节，一个指针型数据也占据 4 个字节，因此运行上述程序得到的输出结果为：

10

1

习题 5.24

本节的程序与 5.5 节在 vector 对象中添加元素的程序类似。两段程序都使用递减的计数器生成元素的值。本程序中，我们使用了前自减操作，而 5.5 节的程序则使用了后自减操作。解释为什么一段程序中使用前自减操作而在另一段程序中使用后自减操作。

【解答】

5.5 节的程序中必须使用后自减操作。如果使用前自减操作，则是用减 1 后的 cnt 值创建 ivec 的新元素，添加到 ivec 中的元素将不是 10~1，而是 9~0。

本节的程序中使用后自减操作或前自减操作均可，因为对 cnt 的自减操作和对 cnt 值的使用不是出现在同一表达式中，cnt 自减操作的前置或后置形式不影响对 cnt 值的使用。

习题 5.25

根据表 5-4 的内容，在下列表达式中添加圆括号说明其操作数分组的顺序（即计算顺序）：

(a) ! ptr == ptr->next

(b) ch = buf[bp++] != '\n'

【解答】

添加圆括号说明其计算顺序如下：

(a) ((! ptr) == (ptr->next))

(b) (ch = ((buf[(bp++)]) != '\n'))

习题 5.26

习题 5.25 中的表达式的计算次序与你的意图不同，给它们加上圆括号使其以你所希望的操作次序求解。

【解答】

添加圆括号获得与上题不同的操作次序如下：

(a) ! (ptr == ptr->next)

(b) (ch = buf[bp++]) != '\n'

习题 5.27

由于操作符优先级的问题，下列表达式编译失败。请参照表 5-4 解释原因，应该如何改正？

```
string s = "word";

// add an 's' to the end, if the word doesn't already end in 's'

string pl = s + s[s.size() - 1] == 's' ? "" : "s" ;
```

【解答】

由表 5-4 可知，在语句 string pl = s + s[s.size() - 1] == 's' ? "" : "s" ; 中，赋值、加法、条件操作符三者的操作次序为：先执行“+”操作，再用表达式 s + s[s.size() - 1]的结果参与条件操作，最后将条件操作的结果赋给 pl。但表达式 s + s[s.size() - 1]的结果是一个 string 对象，不能与字符's'进行相等比较，所以编译失败。

改正为：string pl = s + (s[s.size() - 1] == 's' ? "" : "s") ;。

习题 5.28

除了逻辑与和逻辑或外，C++没有明确定义二元操作符的求解次序，编译器可自由地提供最佳的实现方式。只能在“实现效率”和程序语言使用中“潜在的缺陷”之间寻求平衡。你认为这可以接受吗？说出你的理由。

【解答】

这可以接受。

因为，操作数的求解次序通常对结果没什么影响。只有当二元操作符的两个操作数涉及同一对象，并改变该对象的值时，操作数的求解次序才会影响计算结

果；后一种情况只会在部分（甚至是少数）程序中出现。在实际使用中，这种“潜在的缺陷”可以通过程序员的努力得到弥补，但“实现效率”的提高却能使所有使用该编译器的程序受益，因此利大于弊。

习题 5.29

假设 ptr 指向类类型对象，该类拥有一个名为 ival 的 int 型数据成员，vec 是保存 int 型元素的 vector 对象，而 ival、jval 和 kval 都是 int 型变量。请解释下列表达式的行为，并指出哪些（如果有的话）可能是不正确的，为什么？如何改正？

(a) ptr->ival != 0　　　　(b) ival != jval < kval

(c) ptr != 0 && *ptr++　　　　(d) ival++ && ival

(e) vec[ival++] <= vec[ival]

【解答】

表达式的行为如下：

(a) 判断 ptr 所指向的对象的 ival 成员是否不等于 0。

(b) 判断 ival 是否不等于“jval 是否小于 kval”的判断结果，即判断 ival 是否不等于 true（1）或 false（0）。

(c) 判断 ptr 是否不等于 0。如果 ptr 不等于 0，则求解&&操作的右操作数，即，ptr 加 1，且判断 ptr 原来所指向的对象是否为 0。

(d) 判断 ival 及 ival+1 是否为 true（非 0 值）（注意，如果 ival 为 false，则无需继续判断 ival+1）。

(e) 判断 vec[ival]是否小于或等于 vec[ival+1]。

其中，(d)和(e)可能不正确，因为二元操作符的两个操作数涉及同一对象，并改变该对象的值。

可改正如下：

(d) ival && ival + 1

(e) vec[ival] <= vec[ival + 1]

习题 5.30

下列语句哪些（如果有的话）是非法的或错误的？

(a) vector<string> svec(10);

(b) vector<string> *pvec1 = new vector<string>(10);

(c) vector<string> **pvec2 = new vector<string>[10];

(d) vector<string> *pv1 = &svec;

(e) vector<string> *pv2 = pvec1;

(f) delete svec;

(g) delete pvec1;

(h) delete [] pvec2;

(i) delete pv1;

(j) delete pv2;

【解答】

错误的有(c)和(f)。

(c)的错误在于:pvec2 是指向元素类型为 string 的 vector 对象的指针的指针（即 pvec2 的类型为 vector<string> **），而 new 操作返回的是一个指向元素类型为 string 的 vector 对象的指针，不能用于初始化 pvec2。

(f)的错误在于：svec 是一个 vector 对象，不是指针，不能对它进行 delete 操作。

习题 5.31

根据 5.12.2 节的变量定义，解释在计算下列表达式的过程中发生了什么类型转换？

(a) if (fval)

(b) dval = fval + ival;

(c) dval + ival + cval;

记住，你可能需要考虑操作符的结合性，以便在表达式含有多个操作符的情况下确定答案。

【解答】

(a) 将 fval 的值从 float 类型转换为 bool 类型。

(b) 将 ival 的值从 int 类型转换为 float 类型，再将 fval + ival 的结果值转换为 double 类型，赋给 dval。

(c) 将 ival 的值从 int 类型转换为 double 类型，cval 的值首先提升为 int 类型，然后从 int 型转换为 double 型，与 dval + ival 的结果值相加。

习题 5.32

给定下列定义：

```
char cval; int ival; unsigned int ui;

float fval; double dval;
```

指出可能发生的（如果有的话）隐式类型转换：

```
(a) cval = 'a' + 3;        (b) fval = ui - ival * 1.0;

(c) dval = ui * fval;        (d) cval = ival + fval + dval;
```

【解答】

(a) 'a' 首先提升为 int 类型，再将'a' + 3 的结果值转换为 char 型，赋给 cval。

(b) ival 转换为 double 型与 1.0 相乘，ui 转换为 double 型再减去 ival * 1.0 的结果值，减操作的结果转换为 float 型，赋给 fval。

(c) ui 转换为 float 型与 fval 相乘，结果转换为 double 型，赋给 dval。

(d) ival 转换为 float 型与 fval 相加，结果转换为 double 型，再与 dval 相加，结果转换为 char 型，赋给 cval。

习题 5.33

给定下列定义：

```
int ival; double dval;

const string *ps; char *pc; void *pv;
```

用命名的强制类型转换符号重写下列语句：

```
(a) pv = (void*)ps;        (b) ival = int(*pc);
```

```
(c) pv = &dval;             (d) pc = (char*) pv;
```

【解答】

```
(a) pv = static_cast<void*> (const_cast<string*> (ps));

(b) ival = static_cast<int> (*pc);

(c) pv = static_cast<void*> (&dval);

(d) pc = static_cast<char*> (pv);
```