C++程序设计

（基于C++11标准）

习题和答案

李长河 童恒建 叶亚琴 杨鸣 编著

电子工业出版社
北京

前言

本书是由李长河主编的《C++程序设计》教材的配套习题集和答案，全书内容由李长河主持编写，负责所有习题及答案的制定与审核。中国地质大学（武汉）王俊臣、刁义雅、肖龙、周力、刘永峰、夏海和黄祖传等博士和硕士研究生参与习题及答案的收集与撰写工作。具体分工情况如下：第 1-3 章内容由肖龙收集和编写，第 4-5 章习内容由刁义雅收集和编写，第 6 章内容由夏海收集和编写，第 7 章内容由周力收集和编写，第 8 章内容由刘永峰收集和编写，第 9-10 章内容由王俊臣收集和编写，第 11-12 章内容由黄祖传收集和编写。

李长河

lichanghe@cug.edu.cn

2018年3月

目　　录

第1章　初始C++程序 1

第2章　基本数据类型和表达式 4

第3章　语句控制结构 8

第4章　复合类型、string和vector 16

第5章　函数 31

第6章　类 42

第7章　模板与泛型编程 52

第8章　动态内存与数据结构 64

第9章　继承与多态 80

第10章　简单输入输出 89

第11章　标准模板库 92

第12章　工具与技术 105

第1章　初始C++程序

1.1 编译和调试代码清单1.2，跟踪对象 radius，height 和 volume 的值的变化情况。

答案：

```
#include <iostream>
#include <string>
int main() {
// 定义三个类型对象，存储半径、高和体积的值double
double radius, height, volume;
//屏幕终端显示 Please input radius and height:
std::cout << "please input radius and height: ";
//从键盘输入6.5 12
std::cin >> radius >> height;
//计算圆柱体体积
volume = 3.14*radius*radius*height;
//屏幕输出 the volume is 1591.98
std::cout << "the volume is " << volume << std::endl;
return 0;
}
```

定义 radius，height，volume 时，三个值都为 -9.2559631349317831e+61（随机值），程序运行到 $std :: cin >> radius >> height;$ 时，用户输入 6.5 12 并按回车后，radius，height 和 volume的值分别为6.5，12，-9.25596313493178 31e+61，程序继续往下运行，volume 的值为 1591.98。

1.2 编译和调试代码清单1.3。

答案：

```
#include<iostream>
```

```
    using namespace std;

    class Cylinder {
       double m_radius, m_height;
    public:
       double volume() {
      return 3.14*m_radius*m_radius*m_height;
       }
       Cylinder(double i = 0, double h = 0) :m_radius(i),
  m_height(h) {}
    };

    int main() {
       Cylinder object(1.0, 1.0);
       double vol = object.volume();
       cout << vol << endl;
       return 0;
    }
```

程序输出结果：3.14.

1.3 请简单描述一个类的组成。

答案：类是一种数据类型的抽象，它由类名、数据成员和成员函数组成，数据成员代表数据结构，而成员函数用来对数据成员进行操作，又被称为“方法”。

1.4 编写一个简单程序，输出：Hello, my first C++ program!。

答案：

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

int main()
{
   cout << "Hello, my first C++ program!" << endl;
   return 0;
}
```

1.5 仿照代码清单1.2，编写一个程序，用来计算圆的面积。

答案：

```
#include <iostream>
#include <string>
int main() {
    double radius, area;
    std::cout << "please input radius: ";
    //从键盘输入6.5
    std::cin >> radius;
    area = 3.14*radius*radius;
    std::cout << "the area is " << area << std::endl;
    return 0;

}
```

输入圆的半径为 6.5，屏幕输出 "the area is 132.665"。

第2章　基本数据类型和表达式

2.1 C++ 中有哪几种基本的数据类型？确定一个数据的类型有什么作用？

答案：C++ 定义了几种基本数据类型，包括算术类型（arithmetic type）和空类型（void）。算术类型也称为内置类型（build-in type），包括布尔值、字符型、整型和实型。空类型没有具体的值，仅用于特殊的场合。数据的类型决定了数据的表示方式、取值范围和可进行的操作。

2.2 一个对象的作用域和生命期指的是什么？

答案：作用域（scope）：指定了每个名字在代码中的使用范围，通常以花括号分隔。同一个名字在不同的作用域下可能指向不同的内存空间。名字的作用域起始于它的声明处，结束于声明语句所在的域块的结束处。
生命期（lifetime）：一个对象的生命期是指在其存储周期内可以访问该对象的时间。具有块域的对象，通常其生命期从定义处开始，在作用域结束时消亡。一个对象的消亡，意味着该对象生命期的结束，它在内存中占用的存储空间将被释放。具有块域的对象，它的生命期结束的时刻还与它的数据类型有关。

2.3 一个表达式由什么组成？影响表达式求值的因素有哪些？

答案：表达式是由运算符和操作数按一定语法形式组成的符号序列；影响表达式求值的因素有：运算符的语义、运算符的优先级和结合性，另外求值次序也会影响表达式求值。

2.4 什么是左值和右值？

答案：对于程序员来说，左值所在的内存空间的地址是可以获取的（用取址符& 获取），但右值的地址是无法得到的（无法用取址符&获取）。因此，左值对象既可以读又可以写，而右值对象只能对它进行读操作，不能对它进行写操作。

2.5 在 C++ 程序中什么情况下会对数据进行类型转换？转换的方式有哪些？

答案：同一个表达式中的运算对象的类型如果不一致时会发生类型转换；类型转换的方式有隐式类型转换和强制类型转换两种，C++ 提供了四种强制类型转换方式：static_cast dynamic_cast、const_cast 和 reinterpret_cast，格式如下：$cast-name<type>(expr)$。

2.6 下列哪些是合法的用户自定义标识符。

① begin ② $amount ③ new ④ _1first ⑤ Salary 94

⑥ file_name ⑦ struct ⑧ Salary94 ⑨ _while ⑩ number3.5

答案：①、④、⑥、⑧、⑨

2.7 下列哪些是C++语言中的合法常量。

① 3.14e1L ② 100L ③ 0237 ④ 'abc' ⑤ "A"

⑥ "ABC" ⑦ 0xABCD ⑧ '\581' ⑨ 1.43E3.5 ⑩ 'x0H'

答案：①、②、③、⑤、⑥、⑦、⑧、⑩

2.8 'b'，\142，和\x62分别代表什么？其ASCII值是多少？如何输出 " 和 ' ？

答案：'b' 是字符常量，表示字符 b；\142 是八进制转义字符，\x62 是十六进制转义字符，两者都表示字符 b。字符 b 的 ASCII 码值为 62。输出 " 和 ' 可用转义字符，即 \"，\'。

2.9 判断下列定义中 auto 和 decltype 推断出的类型是什么？

```
1    const int i = 42;
2    auto j = i;
3    decltype (i) j2 = i;
4    int x = 0;
5    auto j3 = x;
6    decltype (x) j4 = i;
```

答案：第 2 行 auto 推导的类型为 int；第 3 行 decltype 推导的类型为 const int；第 5 行 auto 推导的类型为 int，第 6 行 decltype 推导的类型为 int。

2.10 分别根据如下已知，求下列表达式的值：

(1)
```
float x = 2.5, y = 4.7, c = 3.5, d = 2.5;
int a = 2, b = 3;
(int)(x + y) / 24+(float)(a + b) / 2 + (int)c / (int)d
```

(2)
```
char ch1 = 'a', ch2 = '5', ch3 = '0', ch4;
ch4 = ch3 - ch2 + ch1;
```

(3) int x = 3, y = 5, z = 1, a = 0, b = 0, c = 0, d;
    d = (x + z > y)+(x < y == y < z) + (a || (b += 5) || (c -= 3));

(4) int a = 10, b = 20, c = 30;
    float x = 1.2, y = 2.1;
    a < b && x > y || a < b - !c

(5) int i = 5, j = 5, m, n;
    m = i++;
    n = ++j;

答案：(1) 3.5　(2) ch4 = '11'　(3) d = 1　(4) 1　(5) m = 5, n = 6

2.11 试根据 C++ 语言中运算符的优先级和结合性质，给下列表达式添加括号而不改变其求值结果：

(1) a = b + c * d < 2 && 8
(2) a && 077 != 3
(3) a == b || a == c && c < 5
(4) c = x != 0
(5) a < b == c == d

答案：(1) a = ((b + (c * d)) < z) && 8)

(2) a && (077 != 3)

(3) (a == b) || ((a == c) && (c < 5))

(4) c = (x != 0)

(5) ((a < b) == c) == d

2.12 将下列算式或叙述用 C++ 表达式描述。

(1) $|x| > 1$。
(2) a 和 b 之一为 0，但不同时为 0。
(3) 位于圆心在原点，内外半径分别为 a 和 b 的圆环中的点。

答案：(1) x > 1 || x < -1

(2) a == 0 && b != 0 || a != 0&& b == 0 或者a * b == 0 && a + b != 0

(3) ((x * x + y * y) >= a * a) && ((x * x + y * y) < = b)

2.13 参考代码清单1.2（第6页），编写程序。要求输入长方体的长、宽、高，计算并输出长方体的体积和表面积。

答案：

```
#include<iostream>
using namespace std;
int main()
{
    //长宽高
   double length, width, height, volume, surface_areas;
   cout << "Please input length, width and height " << endl;
   cin >> length >> width >> height;//输入三个数长宽高，空格隔开
    volume = length * width * height;
    surface_areas = 2 * (length*height + width *
      height + length * width);
   cout << "The volume of rectangle is:  " <<
  volume << endl;
   cout << "The surface areas of rectangle is:  " <<
  surface_areas << endl;
   return 0;
}
```

输入长、宽、高分别为：1 2 3
输出结果为：

```
The volume of rectangle is:  6
The surface areas of rectangle is:  22
```

第3章　语句控制结构

3.1 什么是语句块？

答案：用花括号括起来的语句序列称为复合语句（compound statement），复合语句构成一个语句块。

3.2 C++ 语言中解决嵌套 if 语句的垂悬 else 问题的原则是什么？

答案：解决嵌套 if 语句中垂悬 else 问题的原则是最近匹配，即规定 else 必须匹配没有匹配的最近的 if。

3.3 简述 continue 语句和 break 语句的异同点。

答案：相同点：两者都可以用来终止循环，仅作用于离它们最近的循环。
不同点：break 用于 switch 语句或循环语句中，用来跳出离它最近的 switch 语句或终止循环的执行。continue 语句只在循环结构中有作用，用来终止当前操作，进入下一次循环，下一次循环是否被执行取决于循环条件是否成立。

3.4 假设以下程序片段中的对象已经正确定义，请指出下列语句片段中是否存在语法或语义错误，若存在错误请改正并说明理由。

```
1  int x;
2  cin >> x;
3  if (x > 0)
4      x = x + 1
5      x /= 2;
6  else
7      x = x - 1;
```

（1）

```
1  float items;
2  cin >> items;
3  switch (items) {
4      case 0.0: cout << "Radio";
5      case 1.0: cout << "Television";
6      case 1.5: cout << "Video Camera";
7  }
```

（2）

```
p = 100;
f = 5;
do {
    p = p - 3;
    f = f + p * p;
} while (p != 0)
```

（3）

```
for (k = 1, k <= n, k = k + 1);
{
    f = f + (f * 1.5) % k;
}
```

（4）

答案：（1）x = x +1 不是一个语句，只是一个表达式，需要在后面加上分号才构成语句。else 语句无法和 if 语句匹配，将第 4 行和第 5 行语句用花括号括起来，形成 if 语句的作用域。

（2）该语句中 switch 后面的表达式是浮点型，显然不合语法，其次，每个 case 后面也是浮点常量，同样不合语法，最后每个 case 后面的语句序列应该以 break 结束，虽然没有 break 并不违犯语法，但通常应该以 break 结束，因为往往有 break 才是程序编写者真正要表达的意思。

（3）while(p!=0) 后面少了一个分号。由于 p 初始化为 100，而每次循环都减 3，所以它是不可能等于 0 的，因此循环终止条件永远也不会得到满足。

（4）for 后面括号中的逗号应该改成分号。另外在 for 后面括号后的分号表明该 for 语句的循环体是一条空语句，而编程者心目中的循环体（由花括号括起来的块语句）实际上不会循环执行。

3.5 下面语句的循环共执行几次？该语句执行完后 x 和 y 的值分别为多少？

```
int x = 3, y = 100;
while (y / x > 5)
    if (y - x > 25)     x = x + 1;
    else     y = y / x;
```

答案：循环总次数应该为14，x = 17，y = 100。

3.6 给出下面代码的输出结果。

```
int k = 1, m = 0;
for( ; k <= 50; k++){
    if(m >= 10) break;
    if(m % 2 == 0){
        m += 5;
        continue;
    }
    m -= 3;
}
cout << m << endl;
```

（1）

```
int k = 0; char c = 'A';
do{
    switch(++c) {
        case 'A': k++; break;
        case 'B': k--;
        case 'C': k += 2; break;
        case 'D': k = k % 2; continue;
        case 'E': k = k * 10; break;
        default : k = k / 3;
    }
k++;
}while(c<'G');
cout << k << endl;
```

（2）

答案：（1）m = 11　　（2）k = 2

3.7 假设有如下函数：

$$y=\begin{cases} x & x<1 \\ 2x-1 & 1\le x<10 \\ 3x-11 & x\ge 10 \end{cases}$$

编写程序，输入 x，输出 y。

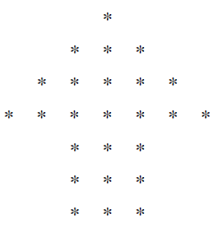
答案：

```
#include <iostream>
using namespace std;
int main() {
    double x, y;
    cout << "输入x = ";
    cin >> x;
    if (x<1)    y = x;
    else if (x<10)  y = 2 * x - 1;
    else  y = 3 * x - 11;
    cout << "y = " << y << endl;
    return 0;
```

```
}
```

输入x=0时，输出y=0；
输入x=5时，输出y=9；
输出x=15时，输出y=34。

3.8 编写程序打印如下图形：



答案：

```
#include <iostream>
using namespace std;
int main() {
    int i, j;
    for (i = 1; i <= 4; i++) {
        for (j = 4 - i; j>0; j--) cout << "  ";
        for (j = 1; j <= 2 * i - 1; j++) cout << " *";
        cout << endl;
    }
    for (i = 1; i <= 3; i++) cout << "      * * *" << endl;
    return 0;
}
```

3.9 用欧几里得算法（基本思想：两个数的最大公约数等于它们中较小的数和两数相除的余数的最大公约数）求两个整数的最大公约数。

答案：

```
#include <iostream>
using namespace std;
int main()
{
```

```
    int num1, num2, resd;
    cout << "输入两个整数：  " << endl;
    cin >> num1 >> num2;
    cout << num1 << "和" << num2 << "的最大公约数为：  ";
    while (1)
    {
        resd = num1 % num2;
        if (resd == 0)break;
        num1 = num2;
        num2 = resd;
    }
    cout << num2 << endl;
    return 0;
}
```

输入两个整数：2 6
输出为：2和6的最大公因数为：2

3.10 编程找出 1-500 之中满足除以 3 余 2，除以 5 余 3，除以 7 余 2的整数。

答案：

```
#include <iostream>
using namespace std;
int main() {
    int i;
    for (i = 1; i <= 500; i++)
        if ((i % 3 == 2) && (i % 5 == 3) && (i % 7 == 2))
            cout << i << endl;
    return 0;
}
```

输出结果为：23 128 233 338 443。

3.11 将 100 元换成用 10 元、5 元和 1 元的组合，共有多少种组合方法。

答案：

```
#include <iostream>
using namespace std;
int main() {
```

```
    int i, j, k, count = 0;
    for (i = 0; i <= 10; i++)
        for (j = 0; j <= 20; j++) {
            k = 100 - 10 * i - 5 * j;
            if (k >= 0) {
                cout << i << '\t' << j << '\t' << k << endl;
                count++;
            }
        }
    cout << count << endl;
    return 0;
}
```

共有121种组合方法。

3.12 统计输入字符流中元音字母的个数（要求：使用 switch 语句）。

答案：

```
#include <iostream>
using namespace std;
int main()
{
    unsigned acnt = 0, ecnt = 0, icnt = 0, ocnt = 0, ucnt = 0;
    char ch;
    while (cin >> ch)
    {
        switch (ch) {
        case 'a':
            ++acnt; break;
        case 'e':
            ++ecnt; break;
        case 'i':
            ++icnt; break;
        case 'o':
            ++ocnt; break;
        case 'u':
            ++ucnt; break;
```

```
        }
    }
    cout << "The number of vowel a:\t" << acnt << endl
        << "The number of vowel e:\t" << ecnt << endl
        << "The number of vowel i:\t" << icnt << endl
        << "The number of vowel o:\t" << ocnt << endl
        << "The number of vowel u:\t" << ucnt << endl;
    return 0;
}
```

输入：animals
输出结果为：

```
The number of vowel a:  2
The number of vowel e:  0
The number of vowel i:  1
The number of vowel o:  0
The number of vowel u:  0
```

3.13 用迭代法求实数 a 的平方根近似值，要求前后两个迭代根之差小于10^{-5}。提示：求平方的迭代公式为：$x_{n+1}=(x_n+a/x_n)/2$。

答案：

```
#include <iostream>
#include <cmath>
using namespace std;
int main()
{
    float x0, x1, a;
    cout << "输入一个正数：  ";
    cin >> a;
    if (a < 0)
        cout << a << "a不能开平方！" << endl;
    else if (a > 0)
    {
        x1 = a / 2;
        do {
            x0 = x1;
```

```
            x1 = (x0 + a / x0) / 2;
        } while (fabs(x1 - x0) >= 1e-5);
        cout << a << "的平方根为: " << x1 << endl;
    }
    else
        cout << a << "的平方根为: 0" << endl;
    return 0;
}
```

测试结果:
输入一个正数: 2
2的平方根为: 1.41421

3.14 利用如下反正切函数计算 π 的近似值，要求误差精度达到 10^{-5}:

$$\arctan(x) \approx x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \ldots$$

提示: 令 x=1，可计算出 $\pi/4$ 的近似值。

答案:

```
#include<iostream>
#include<cmath>
using namespace std;
int main() {
    double x = 1, a, sum;
    int i = 3;
    a = x;
    sum = x;
    do {
        a *= x * x*(-1);
        sum += a / i;
        i += 2;
    } while (fabs(a / i)>1e-6);
    cout << "pi=" << 4*sum << endl;
    return 0;
}
```

输出结果为: pi=3.14159

第4章　复合类型、string和vector

4.1 请简述指针和引用的区别，引用可以用常指针实现吗？

答案：
（1）指针是一个实体，而引用仅是个别名；（2）引用只能在定义时被初始化一次，之后不可改变绑定的对象，指针可以改变与之绑定的对象；（3）引用不能为空，指针可以为空；（4）对引用本身进行操作等价于对引用对象的操作，而对指针本身的操作，操作的是指针对象而非其指向的对象。例如“sizeof 引用”得到的是所指向的变量(对象)的大小，而“sizeof 指针”得到的是指针本身的大小，指针和引用的自增(++)运算意义不一样；（5）引用是类型安全的，而指针不是（引用比指针多了类型检查）。

引用可以用常指针实现。

4.2 左值引用和右值引用有哪些区别？请举例说明。

答案：a)左值引用，必须绑定在左值对象上。右值是不能够修改的，其不可写的特性和const 对象是一致的，因此我们可以用右值来初始化一个 const 左值引用，比如字面值常量、右值表达式等。
b)右值引用只能绑定右值对象，不能绑定左值对象，但利用标准库提供的 move 函数将一个左值显式转换成右值。
例子：

```
int i = 0;
int &lref = i;//right
int &lref1 = (i + 1);//error
const int &lref2 = (i + 1);//right
int &&rref = i;//error
int &&rref1 = move(i);//right
int &&rref2 = 10;//right
```

4.3 用对象 a 分别给出下面的定义：

（1）一个整型类型的对象。

（2）一个指向整型类型对象的指针。

（3）一个指向指针的指针，它指向的指针是指向一个整型类型的对象。

（4）一个有10个整型类型对象的数组。

（5）一个有10个指针的数组，每个指针指向一个整型类型的对象。

（6）一个指向有10个整型对象数组的指针。

答案：

1) int a; //一个整型类型的对象。

2) int *a; //一个指向整型类型对象的指针。

3) int **a; //一个指向指针的的指针，它指向的指针是指向一个整型类型的对象。

4) int a[10]; //一个有10个整型类型对象的数组。

5) int *a[10];//一个有10个指针的数组，该指针是指向一个整型类型的对象。

6) int (*a)[10]; //一个指向有10个整型对象数组的指针。

4.4 自动类型推导关键字 auto 和 decltype 有什么不同？指出下面 auto 和 decltype 推导出来的数据类型分别是什么？

```
1  const int ci=5, &cri=ci;
2  int i=6, &ri=i, *p=&i;
3  auto ans1 = ci;
4  auto ans2 = &ci;
5  auto ans3 = ri;
6  auto &ans4 = i;
7  auto *ans5 = &i;
8  auto ans6 = &i;
9  decltype(i) dec1 = i;
10 decltype(i) dec2 = ci;
11 decltype(cri) dec3 = 5;
12 decltype(ri + 0) dec4 = 5;
13 decltype(*p) dec5 = i;
14 decltype((i)) dec6 = i;
```

答案：

a) auto无法推导引用类型，而decltype可以。

b) auto会忽略const属性，但对于引用，其const属性会保留下来。，而decltype会保留const属性。

```
1  int main() {
2     const int ci = 5, &cri = ci;
3     int i = 6, &ri = i, *p = &i;
4     auto ans1 = ci;//ans1的类型是int
5     auto ans2 = &ci;//ans2的类型是const int *
```

```
    auto ans3 = ri;//ans3是int类型, 而不是引用类型
    auto &ans4 = i;//ans4是一个int类型的引用
    auto *ans5 = &i;//推断出的类型是int,ans5是指向int的指针
    auto ans6 = &i;//推断出的类型是int*,ans6是指向int的指针
    decltype(i) dec1 = i;//i的类型是int,所以dec1的类型也是int
    //dec2的类型和i是一样的,都是const int
    decltype(i) dec2 = ci;
    decltype(cri) dec3 = 5;//dec3的类型是const int &
    decltype(ri + 0) dec4 = 5;//dec4是int类型
    //dec5是int类型的引用,dec5和i绑定在一起
    decltype(*p) dec5 = i;
    //dec6的类型是int类型的引用,dec6和i绑定在了一起
    decltype((i)) dec6 = i;
    return 0;
}
```

4.5 请简述数组和 vector 的异同点？

答案：
相同点：数组和 vector 都是在一段连续内存空间内顺序存放类型相同的对象的有序集合，并且均支持元素的随机访问。
不同点：
1） 数组的大小在初始化后就固定不变，而 vector 可以通过 resize、push_back、pop 等操作进行变化。
2） 数组不能将数组的内容拷贝给其他数组作为初始值，也不能用数组为其他数组赋值；而 vector 可以。
3） 操作不同，vector中封装了一些数组没有的操作，比如插入元素push_back、emplace_back，删除元素erase等操作，使用更方便。

4.6 下面代码中 WHITE 和 BLUE 的值是多少？

```
enum color {WHITE, BACK = 100, RED, BLUE, GREEN = 300};
```

答案：WHITE和BLUE的值分别是0和102。

4.7 说明 const int *p 和 int * const p 的区别。指出下面代码中的错误，并改正。

```
int a = 248, b = 4;
int const c = 21;
int const  *d = &a;
int * const e = &b;
int const * const f = &a;
*c = 32;
*d = 43;
d = &b;
e = &a;
*e = 34;
*f = 25;
f = &a;
```

答案：

int * const p 叫常量指针，不能修改这个指针的指向，但是可以修改这个指针所指向的对象的值。const int * p 叫指向常量的指针，不能通过指针来修改其指向对象的值，但可以改变这个指针的指向。

```
    int a = 248, b = 4;
    int const c = 21;
    const int *d = &a;
    int *const e = &b;
    int const * const f = &a;
    c = 32;//错误：不能修改 const 修饰的对象的值
    *d = 43;//错误：d 是一个指向 const 对象的指针，不能修改其所指向的值
    d = &b;//正确：可以改变 d 的指向
    e = &a;//错误：不能改变 e 的指向
    *e = 34;//正确：可以改变 e 所指向对象的值
     *f = 25;//错误：不可以改变 f 指向的对象的值
    f = &a;//错误：f 是右值，不能出现在 = 符号左边
```

4.8 数组的初始化方式有哪些？请举例说明。

答案：

1） 列表初始化的方式来显式初始化数组元素：

```
int arr[5] = {1, 2, 3, 4, 5};
```

2） 显式初始化部分数组元素：

```
int arr[5] = {1, 2, 3}; //等价于arr[5] = {1, 2, 3, 0, 0}
```

3） 另外，编译器可以根据列表中提供的元素的个数，推断数组的长度：

```
int arr[] = {1, 2, 3, 4, 5}; //数组arr的长度为5 对于字符数组：
```

4） 采用字符串字面值来初始化：

```
char name[] = "Lisha"; //自动添加字符串结束符’\0’
```

这种方式等价于：

```
char name[] = {' L' , ' i' , ' s' , ' h' , ' a' , ' \0' };
```

4.9 请给出以下代码输出的结果。

```
char  a[20] = "auto567", *p = a, *p1;
for (; *p; p++);
for (p1 = p - 1; p1 >= a; p++, p1 -= 2) *p = *p1;
*p = 0;
cout << a << endl;
int s[][3] = { 10,20,30,40,50,60 };
int(*q)[3] = s;
cout << q[0][0] << "," << *(q[0] + 1) << "," << (*q)[1] << '\n';
cout << *s[0] + 1 << "," << *(*s + 1) << "," << *(s[0] + 1);
```

答案：

```
auto56775ta
10,20,20
11,20,20
```

4.10 指出下面程序有什么错误，请简要说明理由并改正。

```
#include<cstring>
int main(){
    char str[10], str1[10];
    for (int i = 0; i<10; i++){
        str1[i] = 'a';
    }
    strcpy(str, str1);
    return 0;
}
```

答案：没有添加C风格字符串结束符\0。

```
#include<cstring>
int main(){
    char str[10], str1[10];
    int i;
    for (i = 0; i<9; i++){
```

```
        str1[i] = 'a';
    }
    str1[i] = '\0';
    strcpy(str, str1);
    return 0;
}
```

4.11 下面的代码是否正确吗？如果不正确，请改正。

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

答案：
不合法。ivec 在定义时没有元素，没有被分配存储空间，无法赋值。

```
vector<int> ivec(10,0);
ivec[0]=42;
```

4.12 编写一个程序，输入两个字符串，将两个字符串连接起来，结果取代第一个字符串（提示：可以使用 string 类）。

答案：

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

输入：
aaa
bbbb
输出：
aaabbbb

4.13 设计程序：输入一段文本，包含空格和换行符，去掉文本中的空格后输出文本（包含换行符）。

答案：

```
#include <iostream>
#include<string>
#include<vector>
using namespace std;
int main() {
    vector<string> data;
    string a;
    while (getline(cin, a) ) {// getline 遇到换行符结束
        data.emplace_back(a);
    }
    for (auto & it : data) {
        int index = 0;
        if (!it.empty()){
            while ((index = it.find(' ', index)) != string::npos){
                it.erase(index, 1);
            }
        }
        cout << it << endl;
    }
    return 0;
}
```

输入：

For example, if you danced all night, you can say you danced up a storm. If you spend the whole month writing a book, you can say you wrote up a storm. If you a re at a party and meet someone who won't keep quiet, you can say they are talkin g up a storm.

I think you get the idea.

Let's look again at the verb "to storm." It can also mean to attack. If soldiers storm a military base, they are attacking it. Used another way, you could say o ne storms into a place. We only do this when we are angry and/or full of purpose .

输出：

Forexample,ifyoudancedallnight,youcansayyoudancedupastorm.Ifyouspendthewhol
hwritingabook,youcansayyouwroteupastorm.Ifyouareatapartyandmeetsomeonewhowo
epquiet,youcansaytheyaretalkingupastorm.

Ithinkyougettheidea.

Let'slookagainattheverb"tostorm."Itcanalsomeantoattack.Ifsoldiersstormamili
ase,theyareattackingit.Usedanotherway,youcouldsayonestormsintoaplace.Weonly
swhenweareangryand/orfullofpurpose.

4.14 编写一个程序，输入一行由英文单词组成的文本（以换行符结束），将这个句子中的字母按英文字典字母顺序重新排列，排列后的单词的长度要与原始句子中的长度相同，并且要求只对 A 到 Z 的字母重新排列，其它字符保持原来的状态。例如：

输入：

THE PRICE OF BREAD IS $125 PER POUND

输出：

ABC DDEEE EF HIINO OP $125 PPR RRSTU

答案：

```
#include <iostream>
#include<string>
#include<vector>
using namespace std;
int main() {
    const int totalLetters(26);
    string input;
    getline(cin, input);
    vector<int> num(totalLetters,0);//记录26个字母出现的次数
    for (auto it : input) {
        if (it >= 'A'&&it <= 'Z') {
            ++num[it - 'A'];
        }
    }

    for (auto & it : input) {
        if (it >= 'A'&&it <= 'Z') {
```

```
            for (int i(0); i < totalLetters; ++i) {
                if (num[i]) {//第i个字母出现的次数不为0
                    --num[i];
                    it = 'A' + i;
                    break;
                }
            }
        }
    }

    cout << input << endl;
    return 0;
}
```

输入：

```
THE PRICE OF BREAD IS $125 PER POUND
```

输出：

```
ABC DDEEE EF HIINO OP $125 PPR RRSTU
```

4.15 设计程序：读取一组整数，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参与求和运算，并输出其值。然后修改程序：头尾元素两两配对（每一个和最后一个，第二个和倒数第二个，以此类推），计算每对元素的和，并输出。

答案：

```
#include <iostream>
#include<string>
#include<vector>
using namespace std;
int main() {
    int a;
    vector<int> arr;
    while (cin >> a) {
        arr.push_back(a);
    }
    int  plus = 1;
```

```
12      if (arr.size() % 2) {
13          cout << "最后一个元素没有求和" << endl;
14          ++plus;
15      }
16      vector<int> ans(arr.size() / 2);
17
18      for (unsigned i(0); i < arr.size() / 2; ++i) {
19          ans[i] = arr[i] + arr[arr.size() - i - plus];
20      }
21      for (auto it : ans) {
22          cout << it << "\t";
23      }
24      cout << endl;
25      return 0;
26  }
```

输入：
1 2 3 4 1111
输出：
最后一个元素没有求和
5 5

4.16 一副扑克有 52 张牌，打桥牌时应将牌分给四个人。请设计一个程序完成自动发牌的工作（提示：利用库函数 rand 产生随机数）。要求：黑桃用 S(Spaces) 表示；红桃用 H(Hearts) 表示；方块用 D(Diamonds) 表示；梅花用 C(Clubs) 表示。

答案：

```
1  #include<iostream>
2  #include<vector>
3  #include<string>
4  using namespace std;
5  int main() {
6      const int total(13);
7      vector<char> color(total); / / 记录花色
8      vector<int> num(total); / / 牌号
9          vector<int> id(total * 4);
```

```
10      for (int i(0); i < total * 4; ++i) {
11          id[i] = i;
12      }
13      int curNum(0);
14      int randNum(0);
15      for (int i(0); i < 4; ++i) {
16          for (int j(0); j < total; ++j) {
17              randNum = rand() % id.size();
18              curNum = id[randNum];
19              id.erase(id.begin() + randNum); / / 删除这张牌
20                  num[j] = curNum%total;
21              curNum /= total;
22              switch (curNum)
23              {
24              case 0: {
25                  color[j] = 'S';
26                  break;
27              }
28              case 1: {
29                  color[j] = 'H';
30                  break;
31              }
32              case 2: {
33                  color[j] = 'D';
34                  break;
35              }
36              case 3: {
37                  color[j] = 'C';
38                  break;
39              }
40              default:
41                  break;
42              }
43          }
44          cout << "people " << i << ":\n" << endl;
45          for (int j(0); j < total; ++j) {
```

```
46              cout << color[j] << "\t";
47          }
48          cout << endl;
49          for (int j(0); j < total; ++j) {
50
51              if (num[j] >= 1 && num[j] <= 9) {
52                  cout << num[j] + 1;
53              }
54              else {
55                  switch (num[j])
56                  {
57                  case 0:
58                      cout << 'A'; break;
59                  case 10:
60                      cout << 'J'; break;
61                  case 11:
62                      cout << 'Q'; break;
63                  case 12:
64                      cout << 'K'; break;
65                  default:
66                      break;
67                  }
68              }
69              cout << "\t";
70          }
71          cout << endl;
72      }
73
74
75      return 0;
76  }
```

输出：

```
people 0:  people 0:

C S D C H D D H C C C H S
3 6 10 5 6 3 A 8 2 J 6 10 2
```

```
people 1:
H S H S S C D S S S H D D
J 8 A J Q K Q K 5 4 5 2 8
people 2:
S D D S D D D H C H H H C
3 J 7 A 9 6 5 4 9 7 2 9 8
people 3:
S C C H H S D C H D C C S
9 Q A 3 K 10 4 4 Q K 7 10 7
```

4.17 设计程序：在 1 2 3 4 5 6 7 8 9 九个数字中插入符号'+'或'-'使得结果为 100，编程求出所有的组合。要求：数字的顺序不能改变。

例如：

```
123 - 45 - 67 + 89 = 100
12 - 3 -4 + 5 - 6 + 7 + 89 = 100
```

答案：

```
#include <iostream>
#include<vector>
#include<iostream>
#include<string>
using namespace std;
int main() {
    const int totalDigit(9);//数位和
    int totalState = pow(3, totalDigit);//每个数位有3种状态，0表示作为数位，1表示前面有‘+’，2表示前面有‘-’
    vector<int> equ(totalDigit);
    string out;//输出
    const int target(100);
    for (int i(0); i < totalState; ++i) {
        if (i % 3)continue;// 算式前面有‘+ ’和没有‘+’是一样的，要去重
        int tmp(i);
        for (int j(0); j < totalDigit; ++j) {
            equ[j] = tmp % 3;
            tmp /= 3;
```

```
18          }
19          out.clear();
20          int sum(0);
21          tmp = 0;//记录当前的数字
22          int sign = 1;// 记录符号位，1 或-1
23          for (int j(0); j < totalDigit; ++j) {//计算算式的和
24              if (equ[j] == 0) {
25                  tmp = tmp * 10 + (j + 1);
26                  out += '0' + (j + 1);
27              }
28              else if (equ[j] == 1) {
29                  sum += tmp*sign;
30                  tmp = (j + 1);
31                  sign = 1;
32                  out += '+';
33                  out += '0' + (j + 1);
34              }
35              else if (equ[j] == 2) {
36                  sum += tmp*sign;
37                  out += '-';
38                  out += '0' + (j + 1);
39                  tmp = (j + 1);
40                  sign = -1;
41              }
42          }
43          sum += tmp*sign;
44          if (sum == target) {
45              cout << out << endl;
46          }
47      }
48      return 0;
49  }
```

输出：

$123-45-67+89$

$12-3-4+5-6+7+89$

$12+3+4+5-6-7+89$

$123+4-5+67-89$

$1+2+3-4+5+6+78+9$

$12+3-4+5+67+8+9$

$1+23-4+56+7+8+9$

$1+2+34-5+67-8+9$

$1+23-4+5+6+78-9$

$123+45-67+8-9$

$123-4-5-6-7+8-9$

第5章　函数

5.1 一个函数有几个要素，分别是什么？

答案：一个函数的定义由四个要素组成：返回值类型（return type）、函数名（function name）、形参列表（parameter list）和函数体（function body）。其中，返回值类型放在函数名前，形参列表放在圆括号内，每个形参必须含有一个类型说明符，形参列表可以为空。如果有多个形参，形参间用逗号隔开。函数体为一组C++语句，放在花括号内，执行特定的操作，允许为空。

5.2 什么是函数声明？它与函数定义的区别是什么？

答案：函数声明用来描述一个函数的类型，目的是告诉编译器函数的基本信息，以便其确定在程序中所调用的函数。与函数定义不同，它只包含了描述一个函数所需要的三个要素：返回值类型、函数名和形参类型。没有函数体，因此形参的名字也可以省略，只给出类型即可。

5.3 以引用方式返回的对象能否是函数的局部对象？

答案：函数的返回值不能是局部对象或者是局部变量的指针或引用

5.4 什么样的函数才能构成重载？

答案：同一作用域下具有相同名字但不同形参列表的一组函数称为重载。

5.5 内联函数和 lambda 表达式的特点是什么？

答案：a) 内联函数：编译器将在调用处嵌入内联函数的代码，不会发生函数调用，因而也不会产生函数调用的开销。优点是提高运行时间效率，缺点是增加了空间开销。
b) lambda表达式：可以理解为一个临时的匿名函数，表示一个可以调用的代码单元。lambda表达式不需要定义函数名，结构较简单。

5.6 下面哪些函数能成功完成两个数的交换？

```
void swap1(int p, int q){
    int temp(0);
    temp = p;
    p = q;
    q = temp;
}
```

```
void swap2(int *p, int *q){
    int *temp(nullptr);
    temp = p;
    p = q;
    q = temp;
}
```

```
void swap3(int *p, int *q){
    int temp(0);
    temp = *p;
    *p = *q;
    *q = temp;
}
```

```
void swap4(int &p, int &q){
    int temp(0);
    temp = p;
    p = q;
    q = temp;
}
```

答案：swap3函数和swap4函数。

5.7 已知一函数的原型是：int f(int, int = 0, double = 0.0)，下列哪个函数可以与 f 重载？请说明理由。

A. int f(int);　　B. int f(int, int);

C. int f(int, int, double);　　D. int f(int, double);

答案：D

5.8 下面程序的输出是什么？

```
#include<iostream>
using namespace std;
int i = 0;
int main(){
    int i = 9;
    cout << i << endl;
    {
        int i = 9;
        cout << i << endl;
        {
            i += 9;
            cout << i << endl;
        }
        cout << i << endl;
    }
    cout << i << endl;
    return 0;
}
```

答案：

9

```
9
18
18
9
```

5.9 下面程序的输出是什么？

```
#include <iostream>
using namespace std;
int sum(int a) {
    int c = 0;
    static int b = 1;
    c++;
    b += a + c;
    return (a + b + c++);
}
int main() {
    int i;
    int a = 2;
    for (i = 0; i < 5; ++i) {
        cout << sum(a) << '\t';
    }
    return 0;
}
```

答案：7 10 13 16 19

5.10 下面程序的输出是什么？

```
#include<iostream>
using namespace std;
#define M(x,y,z) x*y+z
int main(){
    int a = 1, b = 2, c = 3;
    cout << M(a + b, b + c, c + a);
    return 0;
}
```

答案：12

5.11 下面函数调用哪些能正确调用 fun 函数？

```
void fun(int * & b) { }
int main() {
    int a(0), *p(&a);
    fun(*p);
    fun(*a);
    fun(&a);
    fun(p);
    return 0;
}
```

答案：fun 函数的形参类型为int *类型的引用，因此实参必须为int *类型。

```
#include <iostream>
using namespace std;
void fun(int * & b) {
}
int main(){
    int a(0), *p(&a);
    fun(*p);//错误，*p是int类型，
    fun(*a);//错误，‘*’是解引用符号，只能用于指针，a是int类型
    fun(&a);//错误，只能引用左值，&a是右值
    fun(p);//正确
    return 0;
}
```

5.12 下面程序的功能是什么？输出的结果是多少？

```
#include<iostream>
#include<vector>
#include<algorithm>
using namespace std;
int main() {
    vector<int> v;
    for(int i = 1; i < 10; ++i)
        v.push_back(i);
    int a = 0;
    for_each(v.begin(), v.end(),
        [&a](int n) {
        if (n % 2 == 0)
            ++a;
        else
            cout << n << '\t';
    });
    cout << a << endl;
}
```

答案：统计偶数的个数，并输出所有的奇数。
输出：
1 3 5 7 9
4

5.13 下面程序中的函数调用正确吗？请说明理由。

```
#include<iostream>
#include<string>
using namespace std;
void print(string &s) {
    cout << s << endl;
}
void print2(const string &s) {
    cout << s << endl;
```

```
}
int main() {
    string b = "no";
    const string &a = b;
    print(b);
    print2("yes");
    print2(a);
    print2(b);
    print("yes");
}
```

答案：

```
int main() {
    string b = "no";
    const string & a = b;
    print(b); //正确
    print2("yes");//正确
    print2(a);//正确
    print2(b);//正确
    print("yes");//错误，左值引用形参需要绑定左值对象，"yes"是右值
    return 0;
}
```

5.14 函数 fun 的定义如下，问执行 fun(fun(5)) 需要调用函数 fun 多少次？

```
int fun(int n) {
    if (n <= 3) return 1;
    else  return fun(n - 2) + fun(n - 4) + 1;
}
```

答案：函数递归调用4次

5.15 给定一个整数 n，从 n 开始向外螺旋展开，如下为 n=12 的情况：

```
6  5  4  3
7  12 11 2
8  9  10 1
```

请编写程序，打印 n=100 的表。

答案：

```
#include<iostream>
#include<vector>
#include<string>
```

```
using namespace std;
bool judgeLegal(int x, int y, int maxWH,
  const vector<vector<int>>& table) {
   return x >= 0 && x < maxWH&&y >= 0 &&
  y < maxWH &&table[y][x]==0;
}
int main() {
    // 顺序是右上左下
   int next_step[4][2] = { {1,0},{0,-1},{-1,0},{0,+1} };
   const int wh(10);//长宽都是10
   vector<vector<int>> table(wh);
   for (auto & it : table) {
      it.resize(wh);
   }
   for (auto & it : table) {
      for (auto & it2 : it) {
         it2 = 0;
      }
   }
   int curX(0), curY(wh - 1);//起始位置是（0,wh-1)
   int di(0);//起始方向是向右
   int id(1);//从1开始打表
   while (true) {
      table[curY][curX] = id++;
      if (judgeLegal(curX + next_step[di][0],
  curY + next_step[di][1],wh,table)) {
         curX += next_step[di][0];
         curY += next_step[di][1];
      }
      else {
         di = (di + 1) % 4;
         if (judgeLegal(curX + next_step[di][0],
  curY + next_step[di][1],wh,table)) {
            curX += next_step[di][0];
            curY += next_step[di][1];
         }
```

```
            else {
                break;
            }
        }
    }

    for (auto &it : table) {
        for (auto it2 : it) {
            cout << it2 << "\t";
        }
        cout << endl;
    }
    cout << endl;
    return 0;
}
```

28	27	26	25	24	23	22	21	20	19
29	58	57	56	55	54	53	52	51	18
30	59	80	79	78	77	76	75	50	17
31	60	81	94	93	92	91	74	49	16
32	61	82	95	100	99	90	73	48	15
33	62	83	96	97	98	89	72	47	14
34	63	84	85	86	87	88	71	46	13
35	64	65	66	67	68	69	70	45	12
36	37	38	39	40	41	42	43	44	11
1	2	3	4	5	6	7	8	9	10

表 5.1 Output

5.16 已知 n 个人（分别以编号 1, 2, 3, ..., n 分别）围坐在一张圆桌周围。从编号为 1 的人开始，从 k 依次减 1 报数，数到 1 的那个人出列；他的下一个人又从 k 开始报数，数到 1 的那个人出列；依此规律重复下去，直到圆桌周围的人全部出列。每次有人出列后，输出圆桌上还剩下的人的编号。

答案：

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

```
int main() {
    int n, k;
    while (cin >> n >> k) {
        int id(0);
        int cur(k-1);
        vector<int> peo(n);
        for (int i(1); i <= n; ++i) {
            peo[i - 1] = i;
        }
        while (!peo.empty()) {
            cur = (cur + id + peo.size()) % peo.size();
            ++id;
            peo.erase(peo.begin() + cur);
            for (auto it : peo) {
                cout << it << "\t";
            }
            cout << endl;
        }
    }
    return 0;
}
```

输入：

```
10 3
```

输出：

```
1 2 4 5 6 7 8 9 10
1 2 4 6 7 8 9 10
1 2 4 6 7 9 10
1 4 6 7 9 10
1 4 6 7 9
4 6 7 9
4 6 9
6 9
9
```

5.17 编写程序：将一个N进制数转换成M进制数（N和M均小于26）（提示：大于10的数用

大写字母表示，例如 A 表示10， B 表示11，依次类推）。

答案：

```
#include<string>
#include<iostream>
using namespace std;
// 把N进制数转换成10进制数，再把10进制数转换成M进制数。
int  nRadix2decimal(const string & data, int n) {
    int len = data.size();
    int ten(0);
    int powN(1);
    int num;
    for (int i(len - 1); i >= 0; --i) {
        if (data[i] >= '0'&&data[i] <= '9') {
            num = data[i] - '0';
        }
        else {
            num = data[i] - 'A' + 10;
        }
        ten += num*powN;
        powN *= n;
    }
    return ten;
}
string decimal2nRadix(int ten, int n) {
    string data;
    while (ten) {
        if (ten%n >= 10) {
            data += 'A' + ten % n - 10;
        }
        else {
            data += '0' + ten % n;
        }
        ten /= n;
    }
        // 倒转数位
```

```
        for (int i(0); i < data.size() / 2; ++i) {
            swap(data[i], data[data.size() - i - 1]);
        }
    return data;
}
int main() {
    int n, m;
    string num;
    while (cin >> n >> m) {
        cin >> num;
        cout << decimal2nRadix(nRadix2decimal(num, n), m);
    }
    return 0;
}
```

输入：

```
12 6
abcd
```

输出：

```
1411153
```

5.18 利用递归实现辗转相除法求两个整数的最大公约数（提示：辗转相除法的基本思想是两个数的最大公约数等于它们中较小的数和两数相除的余数的最大公约数）。

答案：

```
int gcd(int a, int b) {
    return b ? gcd(b, a%b) : a;
}
```

输入：

```
13 3
22 11
333 3
```

输出：

```
1
```

11
3

5.19 打印 0 到 9 这 10 个数字的全排列。

答案：

```
#include<iostream>
#include<vector>
#include<string>
using namespace std;
// isIn[k]=true 表明数字k是否已经在排列中，per记录当前的排列，curPos记录当前排列的位置，
void display(vector<bool> & isIn, vector<int> & per, int curPos, int maxDig
   if (curPos == maxDigit) {
      for (auto it : per) {
         cout << it << "\t";
      }
      cout << endl;
      return;
   }
   for (int i(0); i < maxDigit; ++i) {
      if (!isIn[i]) {
         isIn[i] = true;
         per[curPos] = i;
         display(isIn, per, curPos + 1, maxDigit);
         isIn[i] = false;
      }
   }
}
int main() {
   const int maxDigit(10);
   vector<bool> isIn(maxDigit,false);
   vector<int> per(maxDigit);
   display(isIn, per, 0, maxDigit);
   return 0;
}
```

第6章　类

6.1 什么是类，它包含哪些要素？

答案：类（class）是一种非常重要的用户自定义类型，它的基本思想是抽象（abstract）和封装（encapsulation），是面向对象程序设计（object-oriented programming，OOP）的基础。抽象包含数据（属性）抽象和函数（操作）抽象，而封装将数据和操作结合在一起，构成一个对象的基本要素。

6.2 构造函数和析构函数在语法和功能上有哪些特点和不同？

答案：类类型对象的初始化过程是由一类特殊的成员函数来完成的，它们叫做构造函数（constructor）。构造函数的作用是在对象创建时为数据成员执行初始化操作，只要创建类类型对象，就会执行类的构造函数。构造函数需要符合以下三个语法条件：1）函数名必须和类名一致；2）无返回值类型说明；3）不能声明为const 成员函数。当创建一个类对象时，编译器自动调用类的构造函数来完成对象的初始化，当一个对象生命期结束时，编译器会自动调用析构函数（destructor）来销毁对象的所有成员。析构函数的名字由波浪号紧接类名构成，不能有返回值，也不能包含形参。

6.3 定义两个类 X 和 Y，其中类 X 包含一个指向 Y 类型对象的指针，而类 Y 包含一个类型为 X 的对象。

答案：

```
1  class X;//事先声明
2  class Y
3  {
4     X m_object;
5  };
6  class X
7  {
```

```
    Y * m_pointer = NULL;
};
```

6.4 请从下面的抽象概念中选择一个，思考需要哪些数据成员，提供一组合理的构造函数并阐明设计原因。

Book，Date，Employee，Vehicle，Program，Tree

答案：开放性设计题目。

6.5 编写一个名为 Person 的类，数据成员为姓名和住址，均为 string 类型，并且提供默认构造函数、复制构造函数和重载的赋值运算符。

答案：

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

class Person {
public:

    Person(const string &name = "", const string &address = "") :
        m_name(name), m_address(address) {}

    Person(const Person &p) :
        m_name(p.m_name), m_address(p.m_address) {}

    Person& operator=(const Person &p) {
        if (this != &p) {
            m_name = p.m_name;
            m_address = p.m_address;
        }
        return *this;
    }

    const string & name()const {
        return m_name;
    }

```

```
26      string & name() {
27          return m_name;
28      }
29
30      const string &  address() const {
31          return m_address;
32      }
33
34      string & const address() {
35          return m_address;
36      }
37
38      void print() const    {
39          cout << "name: " << m_name << "  address: "
40            << m_address << endl;
41      }
42
43  private:
44      string m_name;
45      string m_address;
46  };
47
48  int main()
49  {
50      Person p1("xia hai", "ying cheng"),p2;
51      p2 = p1;
52      cout << p2.name() <<" "<< p2.address() << endl;
53
54      Person p3(p2);
55      p3.print();
56
57      p3.name() = "wang xiaodong";
58      cout << p3.name() << endl;
59      return 0;
60  }
```

程序输出：

```
xia hai ying cheng
name:  xia hai address:  ying cheng
wang xiaodong
```

6.6 在上一题的 Person 类中提供两个接口函数使其能够返回姓名和住址。

答案：见第上一题。

6.7 运算符重载有哪些基本原则？

答案：1）当你考虑重载运算符时，重载运算符的含义应该与运算符本身的含义相关，不能滥用重载运算符；2）重载运算符应该继承而非改变内置版本的行为，比如赋值和复合赋值运算符；3）在重载运算符时，我们需要考虑将其声明为类成员函数还是类的辅助函数；4）我们只能重载C++ 语言已经存在的运算符，不能更改或创造新的运算符。另外，重载运算符的运算对象至少有一个是类类型。

6.8 运算符重载为类成员函数或辅助函数的依据有哪些？

答案：一般来说，我们可以根据如下规则来做出选择：1）赋值（=）、下标（[]）、函数调用（()）和成员访问箭头等运算符必须是类成员；2）改变运算对象自身状态的运算符应该是类成员，比如复合赋值、自增、自减运算符等；3）具有对称性的运算符一般应作为类的辅助函数，比如算术、关系、逻辑运算符等。

6.9 友元在什么时候有用？请说明它们的利弊。

答案：允许其他类或者函数访问其非公有成员。在类的开始或结尾集中声明友元。优点：可以灵活地实现需要访问若干类的私有或受保护成员才能完成的任务，便于与其他不支持类的语言进行混合编程；通过使用友元函数重载可以更自然第使C++语言的I/O流库。缺点：一个类将对非公有成员的访问权授予其他的函数或类，会破坏该类的封装性，降低该类的可靠性和可维护性。

6.10 什么是类的静态成员？静态成员和普通成员有什么区别？

答案：类的静态成员只与类本身有关，与类的对象无关，在声明时加上 static 关键字即可表示一个静态成员。静态成员其不与任何实例化对象绑定，我们可以使用类作用域运算符访问静态成员。static 声明在内部。在外部定义时，不加 static。与一个全局变量类似，其初始值必须是常量表达式。静态数据成员类型可以是它所属的类类型，而非静态成员只能声明为其类的指针或引用。

6.11 改进和完善第一章程序1.3（第6页），使之具有等比缩放、计算表面积、计算体积、修改底面半径和高等功能。

答案：

```
//Cylinder.h
#ifndef CYLINDER_H
#define CYLINDER_H

class Cylinder{
public:
    Cylinder(double radius=0,double height=0);
    ~Cylinder();
    double volume()const;
    double area()const;
    double radius()const;
    double height()const;
    void setRadius(double radius);
    void setHeight(double height);
    void zoom(double rate);
private:
    double m_radius;
    double m_height;
};

#endif // !CYLINDER_H
//Cylinder.cpp
#include"Cylinder.h"

Cylinder::Cylinder(double radius, double height):
  m_radius(radius),m_height(height){}

Cylinder::~Cylinder(){
}

double Cylinder::volume() const
{
    return 3.14*m_radius*m_radius*m_height;
}

```

```
36 double Cylinder::area() const{
37    return 2*3.14*m_radius*m_radius+
38   m_height*3.14*2*m_radius;
39 }
40
41 double Cylinder::radius() const{
42    return m_radius;
43 }
44
45 double Cylinder::height()const {
46    return m_height;
47 }
48
49 void Cylinder::setHeight(double height){
50    m_height = height;
51 }
52
53 void Cylinder::zoom(double rate){
54    m_radius *= rate;
55    m_height *= rate;
56 }
57
58 void Cylinder::setRadius(double radius) {
59    m_radius = radius;
60 }
61
62 //test.cpp
63 #include"Cylinder.h"
64 int main()
65 {
66    Cylinder cy1(1, 1);
67    cout << cy1.area() << endl;
68    cout << cy1.volume() << endl;
69    cy1.zoom(2);
70    cout << cy1.area() << endl;
71    cout << cy1.volume() << endl;
```

```
72
73
74 }
```

输出：

```
12.56
3.14
50.24
25.12
```

6.12 已知复数类 Complex的数据成员如下：

```
1 class Complex{
2 private:
3    double m_real;
4    double m_image;
5 };
```

其中，m_real 和 m_image 分别表示一个复数的实部和虚部。要求实现如下功能，并进行实例测试：

1）提供形参具有默认值的构造函数，实部和虚部默认值均为 0；

2）重新定义默认的赋值运算符（=），并重载复合赋值运算符（+=）、关系运算符（==）、加法运算符（+）以及减法运算符（-）；

3）以友元形式重载输入输出运算符。

答案：

```
1 //Complex.h
2 #ifndef COMPLEX_H
3 #define COMPLEX_H
4
5 #include<iostream>
6
7 class Complex{
8 public:
9   ~Complex();
10  Complex(double real=0, double image=0);
11  Complex & operator = (const Complex & rhs);
12  Complex & operator += (const Complex & rhs);
```

```
13    friend bool operator ==(const Complex & c1,
14    const Complex & c2);
15    friend Complex operator + (const Complex & c1,
16    const Complex & c2);
17    friend Complex operator - (const Complex & c1,
18    const Complex & c2);
19    friend std::ostream & operator << (std::ostream & os,
20    const Complex & rhs);
21    friend std::istream & operator >> (std::istream & is,
22    Complex & rhs);
23  private:
24     double m_real;
25     double m_image;
26  };
27
28
29
30  #endif // !COMPLEX_H
31
32
33  //Complex.cpp
34
35  #include"Complex.h"
36
37  Complex::~Complex(){
38  }
39
40  Complex::Complex(double real, double image):m_real(real),
41  m_image(image){}
42
43  Complex & Complex::operator=(const Complex & rhs){
44     m_real = rhs.m_real;
45     m_image = rhs.m_image;
46     return *this;
47  }
48
```

```
49 Complex & Complex::operator+=(const Complex & rhs){
50     m_real += rhs.m_real;
51     m_image += rhs.m_image;
52     return *this;
53 }
54
55 bool operator==(const Complex & c1,const Complex & c2){
56     return (c1.m_real == c2.m_real&&c1.m_image == c2.m_image);
57 }
58
59 Complex operator+(const Complex & c1,const Complex & c2){
60     return Complex(c1.m_real+c2.m_real,c1.m_image+c2.m_image);
61 }
62
63 Complex operator-(const Complex & c1,const Complex & c2){
64     return Complex(c1.m_real - c2.m_real,c1.m_image - c2.m_image);
65 }
66
67 std::ostream & operator<<(std::ostream & os,const Complex & rhs){
68     os << rhs.m_real << "+" << rhs.m_image << "i";
69     return os;
70 }
71
72 std::istream & operator >> (std::istream & is,Complex & rhs) {
73     is >> rhs.m_real >> rhs.m_image;
74     return is;
75 }
76 //Test.cpp
77 #include"Complex.h"
78 int main() {
79     Complex c,  c1(1, 1), c2(2, 2);
80     cout << "c1: " << c1 << endl;
81     cout << "c2: " << c2 << endl;
82     Complex c3 = c1 + c2;
83     cout << "c3: " << c3 << endl;
84     Complex c4(3, 3), c5(4, 4);
```

```
85      if (c3 == c4) {
86          cout << c3 << " == " << c4 << endl;
87      }
88      else {
89          cout << c3 << " != " << c4 << endl;
90      }
91      if (c3 == c5) {
92          cout << c3 << " == " << c5 << endl;
93      }
94      else {
95          cout << c3 << " != " << c5 << endl;
96      }
97      Complex c6;
98      cin >> c6;
99      cout << c6 << endl;
100     return 0;
101 }
```

```
输出：c1:  1+1i
c2:  2+2i
c3:  3+3i
3+3i == 3+3i
3+3i != 4+4i
输入：2 3
2+3i
```

第7章　模板与泛型编程

7.1 简述什么是泛型编程以及它是如何实现的。

答案：泛型编程是一种语言机制，它是指一种采用与数据类型无关的方式编写代码的方法，是代码重用的重要手段。但是泛型不属于面向对象，它是面向对象的补充和发展。模板是泛型编程的基础，它将数据类型参数化，为数据结构和算法的抽象提供通用的代码解决方案。

7.2 C++ 中模板分为函数模板和类模板，请分别简述二者的作用及原理。

答案：函数模板使得我们可以生成通用的函数，这些函数能够接受任意数据类型的参数，可返回任意类型的值，而不需要对所有可能的数据类型进行函数重载；类模板使得一个类可以有基于通用类型的成员，而不需要在类生成的时候定义具体的数据类型。这在一定程度上实现了宏的作用。

7.3 为什么模板的声明和定义需要放在同一个文件中？

答案：当编译器遇到一个模板定义时，并不会生成代码，只有遇到实例化语句时，模板才会生成一段实例的代码，此时编译器需要知道模板的定义，因此模板的声明和定义都在同一个头文件里。

7.4 模板参数的初始化与函数形参的初始化有何不同？

答案：函数形参的初始化是在调用该函数时进行的，而模板参数需要在编译时确定其类型或值。

7.5 分析下面一段代码，哪里出错了，请说明理由？

```
1 template <typename T>
2 const T& getMax(const T& a, const T& b){
3     return a > b ? a : b;
4 }
```

```
int main(){
    cout << getMax(2.5, 3) << endl;
    return 0;
}
```

答案：该函数模板只有一个模板参数，而所给的实参类型有两种：double 和 int。导致模板函数不能生成相应的参数列表来与其匹配。

7.6 分别简述函数模板重载和函数模板特例化，并简要说明两者的区别。

答案：函数模板重载本质上和函数重载一样，有些函数功能相同，但仅仅是参数不同，为了方便，C++允许这些函数同名。函数模板特例化是指为了某种需求将该函数模板特例成某一种固定参数类型的普通同名函数。两者区别在于前者本质还是函数模板，而后者只是普通函数。

7.7 编写一个函数模板，实现两个对象大小的比较，并按照需求返回较大者或者较小者。

答案：

```
#include<iostream>
using namespace std;
enum class type {Max,Min};
template <typename T>
const T& func(const T &a, const T &b, type c) {
   if (c==type::Max) return a > b ? a : b;
   else return a < b ? a : b;
   return 0;
}
int main() {
   cout << func(7.8, 9.5, type::Max) << endl;
   cout << func(7.8, 9.5, type::Min) << endl;
   cout << func('a', 'b', type::Max) << endl;
   cout << func('a', 'b', type::Min) << endl;
   system("pause");
   return 0;
}
```

输出结果：

```
9.5
```

7.8
b
a

7.8 设计类模板 MyVector，并用实例进行测试。要求：

1) 能直接使用 C++ 标准库类模板 vector 的实例化对象对其进行初始化；
2) 能合并类模板 vector 的实例化对象；
3) 能删除自身重复元素。

答案：

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

template<typename T>
class MyVector
{
   vector<T> m_data;
public:
   MyVector(){}
   MyVector(const vector<T> &vec);
   void delete_repetition();
   void merge(const vector<T> &vec);
   void display();
};

template<typename T>
MyVector<T>::MyVector(const vector<T> &vec):m_data(vec){}

template<typename T>
void MyVector<T>::delete_repetition() {
   vector<T> temp;
   for (size_t i = 0; i < m_data.size(); ++i){
      bool flag = false;
      for (size_t j = 0; j < temp.size(); ++j){
         if (temp[j] == m_data[i]) {
            flag = true;
```

```
28                 break;
29             }
30         }
31         if (!flag) temp.push_back(m_data[i]);
32     }
33     m_data = temp;
34 }
35
36 template<typename T>
37 void MyVector<T>::merge(const vector<T> &vec) {
38     for (auto &i : vec)
39         m_data.push_back(i);
40 }
41
42 template<typename T>
43 void MyVector<T>::display() {
44     for (auto &i : m_data)
45         cout << i << " ";
46     cout << endl;
47 }
48
49 int main() {
50     vector<int> vec1 = { 1,2,3,4,5 };
51     vector<int> vec2 = { 1,3,5,7,9 };
52     MyVector<int> mvec(vec1);
53
54     mvec.display();
55     mvec.merge(vec2);
56     mvec.display();
57     mvec.delete_repetition();
58     mvec.display();
59     return 0;
60 }
```

输出结果为：

1 2 3 4 5

```
1 2 3 4 5 1 3 5 7 9
1 2 3 4 5 7 9
```

7.9 编写函数模板，返回数组的最大元素，并使用实例进行测试。

答案：

```
#include<iostream>
using namespace std;
template <typename T, size_t N>
T& max(T(&x)[N])
{
    T maxv = x[0];
    for (size_t i = 1; i<N; ++i)
        if (maxv<x[i])
            maxv = x[i];
    return maxv;
}
int main()
{
    int a[] = { 4,5,2,8,9,3 };
    double b[] = { 3.5,6.7,2,5.2,9.2 };
    cout << "数组最大值：a" << max(a) << endl;
    cout << "数组最大值：b" << max(b) << endl;
    return 0;
}
```

输出结果：
a数组最大值：9
b数组最大值：9.2

7.10 设计一个类模板 Sample，实现排序功能。要求：
1） 分别实现选择、插入和冒泡三种排序方法，
2） 只允许提供一个接口 sort 来选择性调用哪种排序方法（选择、插入或者冒泡）和排序类型（升序或者降序）。

答案：见下一题答案。

7.11 在上一题类模板 Sample 的基础上添加功能：用二分法查找元素下标，并用实例进行测试。

答案：

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

enum class sort_type { Selection, Insertion, Bubble };
enum class order_type { Ascending, Descending };

template<typename T>
struct Less {  //小于
   bool operator()(const T &a, const T &b) {
      return a < b;
   }
};
template<typename T>
struct Greater {  //大于
   bool operator()(const T &a, const T &b) {
      return a > b;
   }
};
template <typename T, size_t N>
class Sample
{
   T m_data[N];
public:
   Sample() {}
   Sample(const T *arr);
   int seek(const T & c);
   void disp()
   {
      for (size_t i = 0; i<N; ++i)
         cout << m_data[i] << " ";
      cout << endl;
   }

   void sort(sort_type mode, order_type type);
private:
```

```
36      template<typename F >
37      void selectionSort(F f);
38      template<typename F >
39      void insertionSort(F f);
40      template<typename F >
41      void bubbleSort(F f);
42
43      void swap(int i, int j) {
44          T t = m_data[i];
45          m_data[i] = m_data[j];
46          m_data[j] = t;
47      }
48  };
49  template <typename T, size_t N>
50  Sample<T, N>::Sample(const T *arr)
51  {
52      for (size_t i = 0; i<N; ++i)
53          m_data[i] = arr[i];
54  }
55  template <typename T, size_t N>
56  int Sample<T, N>::seek(const T & c)
57  {
58      int low = 0, high = N - 1, mid;
59      while (low <= high)
60      {
61          mid = (low + high) / 2;
62          if (m_data[mid] == c)
63              return mid;
64          else if (m_data[mid]<c) low = mid + 1;
65          else high = mid - 1;
66      }
67      return -1;
68  }
69  template <typename T, size_t N>
70  template<typename F >
71  void Sample<T, N>::selectionSort(F f) {
```

```
72      for (int i = 0; i < N - 1; ++i) {
73          int min = i;
74          for (int j = i + 1; j < N; ++j) {
75              if (f(m_data[j], m_data[min]))
76                  min = j;
77          }
78          swap(i, min);
79      }
80  }
81
82  template<typename T, size_t N>
83  template<typename F >
84  void  Sample<T, N>::insertionSort(F f) {
85      for (int i = 1, j; i < N; ++i) {
86          T t = m_data[i];
87
88          for (j = i; j > 0; --j) {
89              if (f(m_data[j - 1], t))
90                  break;
91              m_data[j] = m_data[j - 1];
92          }
93
94          m_data[j] = t;
95      }
96  }
97
98  template<typename T, size_t N>
99  template<typename F>
100 void  Sample<T, N>::bubbleSort(F f) {
101     for (int i = N - 1; i >= 0; --i) {
102         for (int j = 0; j <= i - 1; ++j)
103         {
104             if (f(m_data[j + 1], m_data[j]))
105                 swap(j, j + 1);
106         }
107     }
```

```
108 }
109 template<typename T, size_t N>
110 void Sample<T, N>::sort(sort_type mode,order_type type){
111    if (type == order_type::Ascending)
112       switch (mode)
113       {
114       case 0: selectionSort(Less<T>());
115          break;
116       case 1: insertionSort(Less<T>());
117          break;
118       case 2: bubbleSort(Less<T>());
119          break;
120       default:
121          break;
122       }
123    else if (type == order_type::Descending)
124       switch (mode)
125       {
126       case 0: selectionSort(Greater<T>());
127          break;
128       case 1: insertionSort(Greater<T>());
129          break;
130       case 2: bubbleSort(Greater<T>());
131          break;
132       default:
133          break;
134       }
135
136 }
137
138 int main()
139 {
140    char a[] = "acegkmpwxz";
141    Sample<char,10> s(a);
142    cout << "元素序列: ";
143    s.disp();
```

```
144     s.sort(sort_type::Bubble,order_type::Descending);
145     cout << "冒泡排序后降序元素序列：";
146     s.disp();
147     s.sort(sort_type::Insertion,order_type::Ascending);
148     cout << "插入排序后升降序元素序列：";
149     s.disp();
150 
151     cout << "'g' 的下标:" << s.seek('g') << endl;
152 
153     return 0;
154 }
```

输出结果：
元素序列：a c e g k m p w x z
冒泡排序后降序元素序列：z x w p m k g e c a
插入排序后升降序元素序列：a c e g k m p w x z
'g' 的下标:3

7.12 设计函数模板，实现数组中数据的简单加密和解密。加密方法：若类型为数值类型，则奇数位自身的数值加 1；若类型为非数值类型，则奇数位的数据自身ASCII值加 1。解密方法反之。

答案：

```
1  #include<iostream>
2  using namespace std;
3  
4  template<typename T, size_t N>
5  void secret(T (&data)[N]) {
6      for (size_t i = 0; i < N; i += 2) {
7          data[i] += 1;
8      }
9  }
10 template<typename T, size_t N>
11 void desecret(T (&data)[N]) {
12     for (size_t i = 0; i < N; i += 2) {
13         data[i] -= 1;
14     }
```

```
}

template<typename T, size_t N>
void display(const T (&data)[N]) {
    for (size_t i = 0; i < N; ++i)
        cout << data[i];
    cout << endl;
}

int main() {
    char ch[] = "This is a secret code!";
    int in[] = { 1,2,2,3,4 };
    secret(ch);
    secret(in);
    display(ch);
    display(in);

    desecret(ch);
    desecret(in);
    display(ch);
    display(in);
    return 0;
}
```

输出结果：

```
Uhjs!it b tedrft!cpdf!☻
22335
This is a secret code!
12234
```

7.13 设计函数模板，判断数组中数据是否为“回文”，完成代码并用实例进行测试。“回文”是指正序和逆序都一样的数据串，例如：12321、madam。

答案：

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

```

```
4  template<typename T, size_t N>
5  bool is_palindrome(const T *data) {
6     for (size_t i = 0; i < N / 2; ++i)
7        if (data[i] != data[N - 1 - i])
8           return false;
9     return true;
10 }
11
12 int main() {
13    char ch1[] = "madam";
14    char ch2[] = "palindrome";
15    int in1[] = { 1,2,3,4,5 };
16    int in2[] = { 1,2,2,1 };
17    if (is_palindrome<char,5>(ch1))
18       cout << "ch1" << "是回文! " << endl;
19    else cout << "ch1" << "不是回文! " << endl;
20    if (is_palindrome<char, 10>(ch2))
21       cout << "ch2" << "是回文! " << endl;
22    else cout << "ch2" << "不是回文! " << endl;
23    if (is_palindrome<int,5>(in1))
24       cout << "in1" << "是回文! " << endl;
25    else cout << "in1" << "不是回文! " << endl;
26    if (is_palindrome<int,4>(in2))
27       cout << "in2" << "是回文! " << endl;
28    else cout << "in2" << "不是回文! " << endl;
29
30    return 0;
31 }
```

输出结果：

```
ch1是回文!
ch2不是回文!
in1不是回文!
in2是回文!
```

第8章　动态内存与数据结构

8.1 通过 new 运算符创建的动态对象的存储类型是什么？它们的生命期如何？

答案：动态内存，也称为自由存储区或堆。生命期从创建该对象的new语句时开始，到delete语句时结束。

8.2 假设有 int *p = new int[20]，则当释放该数组的内存时，应使用怎样的语句来进行释放？

答案：delete [] p;

8.3 移动语义的优势是什么？右值引用所引用的对象特性是什么？

答案：移动语义避免了对象间无谓的复制，提升了程序的性能。右值引用所引用的对象具有以下特性：1）将要消亡；2）没有其他用户。

8.4 一个类的拷贝控制成员有哪些？它们的功能是什么？

答案：一个类包含 5 个拷贝控制成员，它们分别是复制构造函数、赋值运算符、移动构造函数、移动赋值运算符、析构函数。
复制构造函数和移动构造函数利用已有对象初始化另一个同类型对象；赋值运算符和移动赋值运算符将一个对象赋值给同类型的另一个对象。使用左值对象初始化对象时，将使用复制构造函数，而使用右值对象初始化对象时，将使用移动构造函数。复制构造函数和赋值运算符会将右侧运算对象的内容拷贝到另一个对象之中，而移动构造函数和移动赋值运算符接管源对象的内存，通常不分配任何资源。析构函数用于对象销毁时释放占有的资源，通常用于清理善后，该对象生命期结束时释放资源。

8.5 指出下面代码的错误之处，说明理由并改正。

```
int* f1() {
    int x;
    return &x;
}
```

（1）

```
char* f2(char *c){
    char *t = new char[strlen(c)];
    strcpy(t, c);
    return t;
}
```

（2）

```
void f3() {
    while (true)
        auto x=new int;
}
```

（3）

```
void f4() {
    string s1("test");
    string s2(std::move(s1));
    cout << s1[3] << endl;
}
```

（4）

答案：

（1）返回了局部对象的地址，临时和局部变量在函数返回时将被析构所以返回的地址可能是无效的。可将int x;改为static int x;即可返回地址。

（2）使用new开辟的内存空间少了一个用于容纳'\0'的char大小的内存空间，调用strcpy时会在开辟的内存空间之后强行写入一个'\0'，产生内存问题。可将new char[strlen(c)] 改为new char[strlen(c)+1]

（3）没有释放x占用的内存空间。可改为：

```
        void f3() {
        while ( true ){
            auto x = new int;
            // 其他操作
            delete x;
            }
        }
```

（4）使用了移后源对象的值，由于move之后源对象虽然有效，但其值是未定义的，因此不可以访问其内容。可改为销毁操作语句或者赋值操作语句。

8.6 下列unique_ptr的定义中，哪些是有问题的，请说明理由。

```
    int ix = 1024, *pi = &ix, *pi2 = new int(2048);
    using IntP = unique_ptr<int>;
    IntP p0(ix);
```

```
4       IntP p1(pi);
5       IntP p2(pi2);
6       IntP p3(&ix);
7       IntP p4(new int(2048));
8       IntP p5(p2.get());
```

答案：尽管有些定义语法上是正确的，但存在语义问题，具体如下：
（1）语法错误。在定义一个unique_ptr 时，需要将其绑定到一个new 返回的指针上。
（2）语义错误。当p1 消亡时，p1 所指向的对象也被释放，所以导致pi 成为一个空悬指针。
（3）语义错误。当 p2 消亡时，会使得pi2 成为空悬指针。
（4）语义错误。当p3 被销毁时，它试图释放一个栈空间的对象。
（5）正确。
（6）语义错误。p5 和 p2 指向同一个对象，当p5 和p2 被销毁时，会使得同一个指针被释放两次。

8.7 在长度为 10 的数组和链表中插入一个元素，最坏的情况下分别需要移动几个元素？

答案：数组在最坏的情况下需要移动10个元素；链表则无需移动元素，只需将该元素的指针指向插入位置的下一个元素并将插入位置的上一个元素的指针指向插入元素即可。

8.8 单链表的结点结构分为哪两个部分？如果需要实现环形链表，则需要对最后一个结点进行怎样的操作？

答案：单链表的每个结点包含一个数据域和一个指针域。实现环形链表需要将最后一个节点的指针指向第一个节点。

8.9 栈在何处进行插入和删除操作？最先入栈的节点何时出栈？

答案：在栈的一端也就是栈顶进行插入和删除操作。最先入栈的结点最后出栈，也就是“后进先出”。

8.10 写出表达式 3*2^(4+2*2-6*3)-5 求值过程中当扫描到 6 时，操作数栈和运算符栈的内容（其中^为乘幂）。

答案：操作数栈：3，2，8；运算符栈：*^(-。
过程：3入栈s1（操作数栈）；*入栈s2（运算符栈）；2入栈s1；^入栈s2 （^优先级高于*）；(入栈s2，4入栈s1；+直接入栈s2；2入栈s1；*入栈s2；2入栈s1；由于-优先级小于*，所以*出栈，s1中的2 和2 也出栈，2*2 结果为4 入栈s1；由于-优先

级等于+但在+后面所以+ 出栈，s1中的4和4也出栈，4+4结果8入栈；- 入栈s2；所以最后s1 中为3，2，8，s2中为*^(-。

8.11 假设以 I 和 O 分别表示入栈和出栈操作。栈的初态和终态均为空，入栈和出栈的操作序列可表示为仅由I和O组成的序列，可以操作的序列称为合法序列，否则称为非法序列。则下面所示的序列中哪些是合法的？

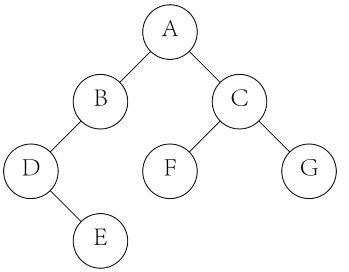
A. IOIIOIOO　　B. IOOIOIIO

C. IIIOIOIO　　D. IIIOOIOO

答案：A、D合法，而B、C不合法。在B中，先入栈1 次，再连续出栈2次，错误。在C中，入栈和出栈次数不一致，会导致最终的栈不空。而A、D均为合法序列，请自行模拟。注意：在操作过程中，入栈次数一定大于或等于出栈次数；结束时，栈一定为空。

8.12 已知二叉树的中序遍历序列为 DEBAFCG，后序遍历序列为 EDBFGCA，试画出该二叉树。

答案：



8.13 从键盘读入 n 个实数，使用动态数组存储所读入的数，并分别输出它们的和与平均值。要求使用函数实现求和与计算平均值。

答案：

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

double sum(double *s, int n) {
  double sum = 0;
  for (int i = 0; i < n; i++) {
```

```
            sum = sum + s[i];
        }
        return sum;
    }

    double avg(double *s, int n) {
        return sum(s, n) / n;
    }

    int main() {
        int n = 0;
        cin >> n;
        double *s = new double[n];
        for (int i = 0; i < n; i++) {
            cin >> s[i];
        }
        cout << sum(s,n) << ' ' << avg(s,n) << endl;

        return 0;
    }
```

测试结果：
输入5，再输入1 2 4 7 11
输出为25 5

8.14 模拟 vector，设计一个 MyVec 类模板，要求实现 5 个拷贝控制成员、容量函数（size）、下标运算符（[]）、尾插函数（push_back）、尾部删除函数（pop_back）、指定位置插入函数（insert）和指定位置删除函数（erase）。

```
    #define  SPACE_CAPACITY 5 // 比当前 size 多预留5个容量，在少量添加元素的情况下可以避免多次重复分配内存
    template<typename T>
    class MyVec {
    public:
        // 构造
        MyVec(int initSize = 0) :m_size(initSize),
              m_capacity(initSize + SPACE_CAPACITY) {
            m_p = new T[m_capacity];
```

```
9         }
10        // 拷贝构造
11        MyVec(MyVec& v) :m_size(v.m_size), m_capacity(v.m_capacity),
12              m_p(new T[m_capacity]) {
13            // 深复制
14            for (int i = 0; i < m_size; ++i) {
15                m_p[i] = v.m_p[i];
16            }
17        }
18        // 赋值运算符
19        MyVec<T> &operator=(MyVec& ths) {
20            if (this != &ths) {
21                delete[] m_p;           // 销毁原来的内存空间
22                m_size = ths.m_size;
23                m_capacity = ths.m_capacity;
24                m_p = new T[m_capacity];  // 重新分配内存空间
25                // 深复制
26                for (int i = 0; i < ths.m_size; ++i) {
27                    m_p[i] = ths.m_p[i];
28                }
29            }
30            return *this;  // 返回运算结果
31        }
32        // 移动构造
33        MyVec(MyVec&& v) :m_size(v.m_size), m_capacity(v.m_capacity),
34              m_p(v.m_p) {
35            v.m_p = nullptr;
36        }
37        // 移动赋值运算符
38        MyVec<T> &operator=(MyVec&& ths) {
39            if (this != &ths) {
40                delete[] m_p;           // 销毁原来的内存空间
41                m_size = ths.m_size;
42                m_capacity = ths.m_capacity;
43                m_p = ths.m_p;
44                ths.m_p = nullptr;    // 将原对象置于可以被析构的状态，此处
```

将其中指针设为 nullptr

```
45            }
46            return *this;  // 返回运算结果
47        }
48        // 返回当前元素数量
49        int size() {
50            return m_size;
51        }
52        // 下标
53        T& operator[](int index) {
54            return m_p[index];
55        }
56
57        // 尾插
58        void push_back(const T &item) {
59            if (m_size == m_capacity) {
60                reserve(m_capacity * 2 + 1);
61            }
62            m_p[m_size++] = item;
63        }
64        // 尾部删除
65        void pop_back() {
66            if (m_size > 0) { //TODO:
67                --m_size;
68            }
69        }
70        // 指定位置插入
71        void insert(int index, const T& item) {
72            if (m_size == m_capacity) {
73                reserve(m_capacity * 2 + 1);
74            }
75            for (size_t i = m_size - 1; i >= index; i--) {
76                m_p[i + 1] = m_p[i];
77            }
78            m_p[index] = item;
79            m_size++;
```

```
80          }
81          // 指定位置删除
82          void erase(int index) {
83              if (index < m_size) {
84                  for (int i = index; i < m_size; i++) {
85                      m_p[i] = m_p[i + 1];
86                  }
87                  m_size--;
88              }
89          }
90          // 重新分配内存空间大小
91          void reserve(int newCapacity) {
92              if (newCapacity < m_size) {
93                  return;
94              }
95              T* temp = m_p;              // 将m_p存储的地址给到过渡变量
96              m_p = new T[newCapacity];   // 为m_p分配新的内存空间
97                                          // 将原始内容拷贝回来
98              for (int i = 0; i < m_size; ++i) {
99                  m_p[i] = temp[i];
100             }
101             // 释放原来的空间
102             delete[] temp;
103         }
104         // 析构
105         ~MyVec() {
106             delete[] m_p;
107         }
108     private:
109         int m_size;     // 当前元素数量
110         int m_capacity;// 容量
111         T* m_p;         // 被管理指针
112     };
```

测试代码如下：

```
#include"Vector.h"
#include<iostream>

using namespace std;

int main() {
    // 测试构造函数及尾插函数
    MyVec<int> v1;
    for (size_t i = 0; i < 5; i++) {
        v1.push_back(i);
    }
    for (size_t i = 0; i < v1.size(); i++) {
        cout<< v1[i] <<' ';
    }
    cout << endl;
    // 测试尾部删除函数
    v1.pop_back();
    for (size_t i = 0; i < v1.size(); i++) {
        cout << v1[i] << ' ';
    }
    cout << endl;
    // 测试删除函数
    v1.erase(1);
    for (size_t i = 0; i < v1.size(); i++) {
        cout << v1[i] << ' ';
    }
    cout << endl;
    // 测试插入函数
    v1.insert(2,10000);
    for (size_t i = 0; i < v1.size(); i++) {
        cout << v1[i] << ' ';
    }
    cout << endl;

    // 测试拷贝赋值运算符以及赋值运算
    auto v2 = v1;
```

```
37          v2[1] = 123;
38          for (size_t i = 0; i < v1.size(); i++) {
39              cout << v1[i] << ' ';
40          }
41          cout << endl;
42          for (size_t i = 0; i < v2.size(); i++) {
43              cout << v2[i] << ' ';
44          }
45          cout << endl;
46          // 测试拷贝构造函数
47          MyVec<int> v3(v2);
48          for (size_t i = 0; i < v3.size(); i++) {
49              cout << v3[i] << ' ';
50          }
51          cout << endl;
52
53          // 测试移动构造函数
54          MyVec<int> v4(move(v3));
55          for (size_t i = 0; i < v4.size(); i++) {
56              cout << v4[i] << ' ';
57          }
58          cout << endl;
59          // 测试移动构造运算符
60          v3 = move(v4);
61          for (size_t i = 0; i < v3.size(); i++) {
62              cout << v3[i] << ' ';
63          }
64          cout << endl;
65
66          return 0;
67      }
```

测试结果如下：

```
0 1 2 3 4
0 1 2 3
0 2 3
```

```
0 2 10000 3
0 2 10000 3
0 123 10000 3
0 123 10000 3
0 123 10000 3
0 123 10000 3
```

8.15 在课本线性链表代码的基础之上，实现一个求线性链表大小即结点数目的成员函数size。

答案：

```
template<typename T>
int SList<T>::size() {
  Node<T> *p = m_head;
  int size = 0;
  while (p) {
     size++;
     p = p->m_next;
  }
  return size;
}
```

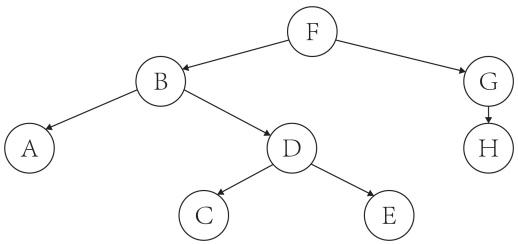
测试代码如下：

```
#include"SList.h"
using namespace std;
int main() {
  SList<int> l;
  // 尾插五个整型对象
  for (size_t i = 0; i < 5; i++) {
     l.push_back(i);
  }
  cout << l.size() << ' ';
  l.erase(2); // 删除值为2的节点
  cout << l.size() << ' ';
  l.clear();  // 清空链表
  cout << l.size() << ' ';
```

```
14          return 0;
15      }
```

输出结果：5 4 0

8.16 以书中二叉树有关代码作为基础，增加一个成员函数 leaf，其功能是打印输出如下图所示的二叉树中的所有叶子结点：：



答案：

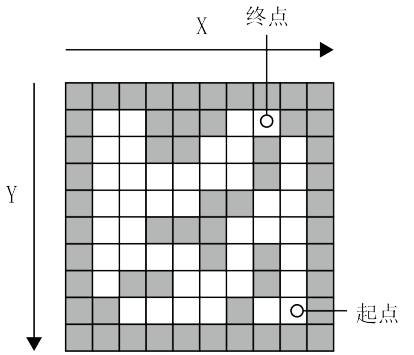
```
1       成员函数leaf：
2       template<typename T>
3       void BinaryTree<T>::leaf(Node<T> *p) {
4         if (p != nullptr) {
5            if (!(p->m_left || p->m_right)) {
6               cout << p->m_data << ' ';
7            }
8            leaf(p->m_left);
9            leaf(p->m_right);
10        }
11      }
12      主函数如下：
13      #include"Tree.h"
14      #include<iostream>
15
16      using namespace std;
17
18      int main() {
19        // 首先创建如图所示的查找树
20        char keys[8] = { 'F','B','A','D','C','E','G','H' };      //注意
```

数组中字母的顺序决定了插入二叉树的顺序，也决定了构成的树的形态

```
        BinaryTree<char> bstree;
        for (auto i : keys)
            bstree.insert(i);
        // 调用函数查找所有的叶子节点
         bstree.leaf(bstree.root());
         return 0;
    }
```

输出结果：A C E H

8.17 如图所示，我们用二维数组表示一个四周有墙壁（灰色方块）的迷宫并设置起点和终点，利用栈的知识和8.4节（第190页）的有关代码，通过回溯法寻找从起点到终点的走迷宫程序，将所找到的路径输出出来。




```
1 1 1 1 1 1 1 1 1 1
1 0 0 1 1 1 0 0 1 1
1 0 0 1 1 0 0 1 0 1
1 0 0 0 0 0 0 1 0 1
1 0 0 0 0 1 1 0 0 1
1 0 0 1 1 1 0 0 0 1
1 0 0 0 0 1 0 1 0 1
1 0 1 1 0 0 0 1 0 1
1 1 0 0 0 0 1 0 0 1
1 1 1 1 1 1 1 1 1 1
```

答案：

```
#include<iostream>
#include"Stack.h"    // 课本中栈的头文件

class Point {
public:
  int x;
  int y;
  char dir;
  Point() = default;
  Point(int x1, int y1) {
```

```
11            x = x1;
12            y = y1;
13        }
14    };
15    // 定义10*10大小的迷宫
16    int maze[10][10] = { 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
17                         1, 0, 0, 1, 1, 1, 0, 0, 1, 1,
18                         1, 0, 0, 1, 1, 0, 0, 1, 0, 1,
19                         1, 0, 0, 0, 0, 0, 0, 1, 0, 1,
20                         1, 0, 0, 0, 0, 1, 1, 0, 0, 1,
21                         1, 0, 0, 1, 1, 1, 0, 0, 0, 1,
22                         1, 0, 0, 0, 0, 1, 0, 1, 0, 1,
23                         1, 0, 1, 1, 0, 0, 0, 1, 0, 1,
24                         1, 1, 0, 0, 0, 0, 1, 0, 0, 1,
25                         1, 1, 1, 1, 1, 1, 1, 1, 1, 1
26    };
27    int startX = 8, startY = 8;   // 起点坐标
28    int endX = 7, endY = 1;       // 终点坐标
29
30    int main() {
31        Stack<Point> s;
32        // 由于栈的特性是后进先出，为了最后输出走法的正确也就是能够表示从起点到终点的路线而不是反过来，我们在这里让路线搜索从终点向起点进行。
33        int i = endX, j = endY;
34        while (!(i == startX && j == startY)) {
35            // 向右
36            if (maze[j][i + 1] == 0) { // 注意X、Y坐标向二维数组元素的转换
37                Point p(i, j);
38                s.push(p);
39                maze[j][i] = -1; // 标记走过的点
40                i = i + 1;
41            }
42            // 向下
43            else if (maze[j + 1][i] == 0) {
44                Point p(i, j);
```

```
45              s.push(p);
46              maze[j][i] = -1; // 标记走过的点
47              j = j + 1;
48          }
49          // 向左
50          else if (maze[j][i - 1] == 0) {
51              Point p(i, j);
52              s.push(p);
53              maze[j][i] = -1; // 标记走过的点
54              i = i - 1;
55          }
56          // 向上
57          else if (maze[j - 1][i] == 0) {
58              Point p(i, j);
59              s.push(p);
60              maze[j][i] = -1; // 标记走过的点
61              j = j - 1;
62          }
63          // 回溯
64          else if (!s.empty()) {
65              Point p = s.top();
66              s.pop();
67              maze[j][i] = -1;
68              i = p.x;
69              j = p.y;
70          }
71          else {
72              cout << "oohh!wrong maze" << endl;
73          }
74      }
75
76      Point p;
77      while (!s.empty()) {
78          Point p = s.top();
79          s.pop();
80          cout << "->(" << p.x << "," << p.y << ") ";
```

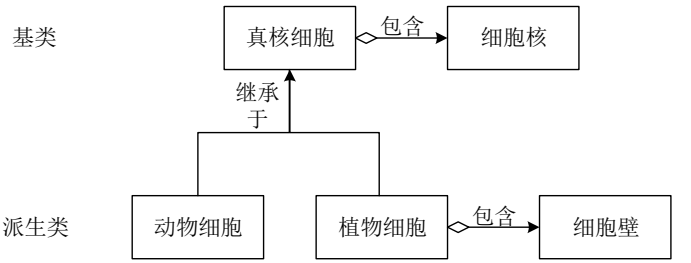
```
81          }
82            return 0;
```

输出结果：->(8,7) ->(8,6) ->(8,5) ->(7,5) ->(6,5) ->(6,6) ->(6,7) ->(5,7) ->(4,7) ->(4,6) ->(3,6) ->(2,6) ->(2,5) ->(2,4) ->(3,4) ->(4,4) ->(4,3) ->(5,3) ->(6,3) ->(6,2) ->(6,1) ->(7,1)

第9章　继承与多态

9.1 请仿照教材中图9-1（教材第206页），用简单的示意图设计真核细胞、植物细胞、动物细胞、细胞核、细胞壁之间的类关系（提示：真核细胞指含有细胞核的细胞，植物细胞含有细胞壁而动物细胞不含细胞壁）。

答案：



9.2 在保证类 A 的成员不能直接从外部访问的前提下，如何改正下面的代码，使得派生类 B 获得对基类 A 的成员的访问权限？

```
1  class A {
2  private:
3      int m_val;
4  };
5  class B : public A {
6  public:
7      void add_val() {
8          m_val++;
9      }
10 };
```

答案：将类 A 中的 private 访问限定符改为 protected。

9.3 若一组含继承关系的类的定义如下：

```
class Base {
public:
    int m_val = 0;
};
class D1:protected Base{};
class D2:private D1{
public:
    void print(){
        cout << m_val << endl;
    }
};
class D3 : public D2 { };
```

假设有 D2 a; D3 b; 请问以下语句会出现什么错误，说明理由。

```
a.print();
b.print();
```

答案：第二行编译会出现错误。类 Derived1 通过 protected 方式继承类 Base，成员 m_val 访问权限从 public 变为 protected；类 Derived2 通过 private 方式继承类 Derived1，成员 m_val 访问权限从 protected 变为 private；类 Derived3 通过 public 方式继承类 Derived2，仅继承其父类的 public 成员 print 函数，不再继承父类的 private 成员 m_val，因此实例 b 在其 print 函数内不能访问 m_val。

9.4 基类 A 和派生类 B 的定义如下：

```
class A {
public:
    int m_val = 1;
};
class B : public A {
public:
    int m_val = 2;
};
```

则下面语句的输出是什么？

```
B data;
cout << data.m_val << endl;
cout << data.A::m_val << endl;
```

答案：

2

1

9.5 什么情况下派生类对象不能被隐式自动转换到基类对象？

答案：转换的前提是公有继承，不是通过 public 方式继承的派生类对象不能被隐式自动转换到基类对象。

9.6 为什么基类对象不能被隐式自动转换为派生类对象？

答案：因为基类对象不能提供派生类对象新定义的部分。

9.7 请写出下列程序的输出。

```
struct C {
    C(){ cout << "C1";}
    virtual ~C(){cout<<"C2";}
};
struct D : public C {
    D(){ cout << "D1";}
    ~D() { cout << "D2";}
};
struct A {
    A(){ cout << "A1";}
    virtual ~A(){cout<<"A2";}
protected:
    C c1;
};
struct B : public A {
    B(){ cout << "B1";}
    ~B() { cout << "B2";}
private:
    D d1;
};

int main () { B b1; return 0; }
```

答案：

C1A1C1D1B1B2D2C2A2C2

9.8 派生类的复制构造函数的初始化列表中，是否需要调用基类的复制构造函数？说明理由。

答案：需要。如果没有显式调用基类的复制构造函数，那么编译器将调用基类的默认构造函数，基类成员执行默认初始化，而非复制被复制实例的基类成员值。

9.9 在类之间的关系设计中，有哪两种形式的代码复用？它们的区别是什么？

答案：模板与继承。模板为除了数据类型不同，其他部分都相同的函数或类的定义提供代码重用。继承为在类成员方面与已有类相似，但又不完全相同的类的定义提供代码重用。

9.10 什么是静态类型和动态类型？它们的区别是什么？

答案：静态类型指对象声明时的类型或表达式生成时的类型，在编译时确定；动态类型指的是指针指向或引用绑定到的对象的类型，仅在运行时可知。

9.11 运行时多态指什么？如何实现运行时多态？

答案：运行时多态可以简单地概括为“一个接口，多种实现”，即程序在运行时才决定调用的函数。运行时多态通过动态绑定实现，即用基类的指针或引用，调用派生类的重载了基类虚函数的成员函数。

9.12 基类的析构函数为什么通常定义为虚函数？

答案：如果基类析构函数为非虚函数，则 delete 一个指向派生类对象的基类指针将产生未定义的行为。

9.13 关键字 final 和 override 有什么作用？

答案：关键字 override 用来显式说明派生类的函数要覆盖基类的虚函数，关键字 final 用来阻止派生类覆盖基类版本的虚函数。

9.14 定义基类天体（Celestial）及其派生类行星（Planet），天体类要求包含质量（m_mass）和半径（m_radius），行星类要求包含天体类对象，以表示其卫星（m_satellite）。每个类均需定义构造函数、复制构造函数及赋值运算符函数。

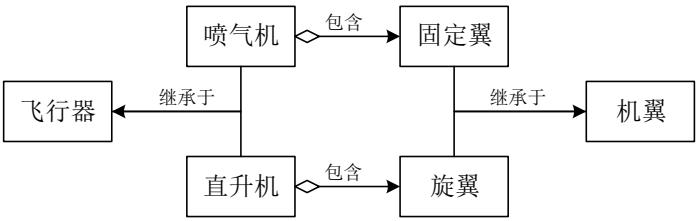
答案：

```
#include <iostream>
#include <vector>
#include <memory>

using namespace std;

class Celestial {
private:
    double m_mass;
    double m_radius;
```

```
11 public:
12     Celestial(double mass, double radius)
13         : m_mass(mass), m_radius(radius) {}
14     Celestial(const Celestial& c)
15         : m_mass(c.m_mass), m_radius(c.m_radius) {}
16     Celestial& operator=(const Celestial& c) {
17         if (this == &c) return *this;
18         m_mass = c.m_mass;
19         m_radius = c.m_radius;
20         return *this;
21     }
22 };
23
24 class Planet : public Celestial {
25 private:
26     vector<Celestial> m_satellite;
27 public:
28     void add_satellite(Celestial& s) {
29         m_satellite.push_back(s);
30     }
31     Planet(double mass, double radius) : Celestial(mass, radius) {}
32     Planet(const Planet& p)
33         : Celestial(p), m_satellite(p.m_satellite) {}
34     Planet& operator=(const Planet& p) {
35         if (this == &p) return *this;
36         Celestial::operator=(p);
37         if (!m_satellite.empty())
38             m_satellite.clear();
39         m_satellite = p.m_satellite;
40         return *this;
41     }
42 };
43
44 int main()
45 {
46     Celestial Moon1(5,5);
```

```
    Celestial Moon2(6,6);
    Planet Earth1(10,10);
    Earth1.add_satellite(Moon1);
    Planet Earth2(20,20);
    Earth2.add_satellite(Moon2);
    Planet Earth3(Earth2);
    Earth3 = Earth1;
    return 0;
}
```

9.15 请按下图中类的关系图，简单地定义飞行器（Aircraft）、喷气机（JetAircraft）、直升机（Helicopter）、固定翼（FixedWings）、旋翼（RotaryWings）和机翼（Wings）这六个类。



答案：

```
class Wings {};
class RotaryWings : public Wings {};
class FixedWings : public Wings {};
class Aircraft {};
class JetAircraft : public Aircraft {
private:
    FixedWings m_wings;
};
class Helicopter : public Aircraft {
private:
    RotaryWings m_wings;
};
```

9.16 现需要一个理财程序，其中包含四个类，分别为投资（Investment）、储蓄（Saving）、基金（Fund）和理财人（Person）。储蓄和基金为两种具体投资，都有确定的投资金额（m_capital），但它们年底结算（settle）的方式不同。储蓄的结算方式如下：

```
m_capital = m_capital * (1+1.78/100);
```

基金的结算方式如下：

```
m_capital = m_capital * (rand()%20 + 90) / 100;
```

理财人实例化时确定本金 m_cashFlow 。理财人通过其成员函数 addInvest 添加具体投资，将新建的投资类实例的地址保存在 vector 类型的成员 mv_invests 中，并从本金中减去投资额。一年后，理财人通过其成员函数 settle 结算所有投资，将投资额返回本金。实现效果如下：

```
// 初始本金十万元
Person Wang(100000);
// 储蓄、基金分别投资五万、两万
Wang.addInvest(Saving(50000));
Wang.addInvest(Fund(20000));
// 年底全部结算转入本金
cout << Wang.settle() << endl;
```

请据此给出投资、储蓄、基金、理财人这四个类的定义，并利用上述代码进行测试。

答案：

```
#include <stdlib.h>
#include <vector>
#include <iostream>
#include <memory>
using namespace std;

class Investment {
protected:
    double m_capital;
public:
    Investment(double invest) : m_capital(invest) {}
    const double capital() const { return m_capital; }
    virtual int settle() = 0;
    virtual ~Investment() {}
};
```

```
16 class Saving : public Investment {
17 public:
18     Saving(double invest) : Investment(invest) {}
19     int settle() {
20         return m_capital = m_capital * (1+1.78/100);
21     }
22 };
23 class Fund : public Investment {
24 public:
25     Fund(double invest) : Investment(invest) {}
26     int settle() {
27         return m_capital = m_capital*(rand()%20+90)/100;
28     }
29 };
30 class Person {
31 private:
32     vector<unique_ptr<Investment>> mv_invests;
33     double m_cashFlow;
34 public:
35     Person(double cashFlow) : m_cashFlow(cashFlow) {}
36     template<typename T>
37     void addInvest(const T& invest) {
38         mv_invests.push_back(unique_ptr<Investment>
39           (new T(invest)));
40         m_cashFlow -= mv_invests.back()->capital();
41     }
42     int settle() {
43         for (auto& p : mv_invests) {
44             m_cashFlow += p->settle();
45         }
46         mv_invests.clear();
47         return m_cashFlow;
48     }
49 };
50 int main() {
51    // 初始本金十万元
```

```
52      Person Wang(100000);
53      // 储蓄、基金分别投资五万、两万
54      Wang.addInvest(Saving(50000));
55      Wang.addInvest(Fund(20000));
56      // 年底全部结算转入本金
57      cout << Wang.settle() << endl;
58      return 0;
59  }
```

输出：99090

第10章　简单输入输出

10.1 对于 cout 对象，除了使用 endl 强制控制台窗口输出数据，还有什么其他操纵符，区别是什么？

答案：还有 flush 和 ends。 endl 会添加换行符，flush 不添加额外字符，ends 添加一个空字符（'\0'）。

10.2 写出代码，以八进制、八位宽度的格式输出十进制整数 25。

答案：

```
cout << oct << setw(8) << 25 << endl;
```

10.3 写出代码，以两位有效数字、科学表示法、十二位宽度的格式输出浮点数 1.0。

答案：

```
cout << setprecision(2) << setw(12) << scientific << 1.0 << endl;
```

10.4 如何判断文件流对文件的打开是否成功？

答案：检测文件流对象的值为 true 还是 false。

10.5 文件流对象在读写数据完成后为什么需要关闭操作？

答案：如果没有关闭操作，与文件流关联的文件不能被其他文件流对象访问，该文件流对象也不能访问其他文件。

10.6 写出代码，以写方式打开文件"test.txt"，写入的数据追加到文件末尾。

答案：

```
ofstream out("test.txt", ios::app);
```

10.7 编写一个程序，将键盘输入复制到通过命令行指定的文件中（输入 Ctrl+z 退出）。

答案：

```
1 #include <iostream>
2 #include <iomanip>
3 #include <fstream>
```

```
using namespace std;

int main(int argc, char* argv[])
{
    if (argc <= 1)
    {
        cout << "need file name" << endl;
        exit(0);
    }
    string fileName = argv[1];
    ofstream fo(fileName.c_str());

    char ch;
    while ( cin && cin.get(ch) )
    {
        fo << ch;
    }
    fo.close();
    return 0;
}
```

10.8 编写一个程序，将两个文件合成一个文件。

答案：

```
#include <fstream>
#include <string>
using namespace std;

int main() {
   ifstream ifile1("1.txt");
   ifstream ifile2("2.txt");
   ofstream ofile("3.txt");
   string line;
   if (ifile1) {
      while (!ifile1.eof()) {
         getline(ifile1, line);
         ofile << line << endl;
```

```
14          }
15      }
16      if (ifile2) {
17          while (!ifile2.eof()) {
18              getline(ifile2, line);
19              ofile << line << endl;
20          }
21      }
22      ifile1.close();
23      ifile2.close();
24      ofile.close();
25      return 0;
26  }
```

第11章　标准模板库

11.1 简述 STL 中迭代器与 C++ 指针的异同点。

答案：迭代器包含内存地址的获得，是面向对象版本的指针。
迭代器与指针有许多相同之处，但迭代器保存所操作的特定容器需要的状态信息，从而实现与每种容器类型相适应的迭代器。而且有些迭代器操作在所有容器中是一致的，这带来了很大的方便。如++运算符总是返回容器下一个元素的迭代器，就像数组指针++后指向下一个元素；间接引用符“*”，总是表示迭代器指向的容器元素，就像数组指针加*号后代表指针所指的元素。
迭代器在STL中起粘结剂的作用，用来将STL的各部分结合在一起。从本质上说，STL提供的所有算法都是模板，我们可以通过使用自己指定的迭代器来对这些模板实例化。
迭代器可以包括指针，但迭代器又不仅是一个指针。

11.2 顺序容器包括哪几种？它们各以什么数据结构为基础？各有哪些特点？

答案：C++标准模板库提供三种顺序容器：vector、string、list、deque、forward_list 和array。vector、string 、array 类和deque类是以数组为基础的，list 类是以双向链表为基础的，forward_list类是以单向链表为基础的。
矢量（vector）类提供了具有连续内存地址的数据结构。它和C/C++ 的数组一样通过下标运算符[ ]直接有效地访问矢量的任何元素。与数组不同，vector 的内存用尽时，vector自动分配更大的连续内存区，将原先的元素复制到新的内存区，并释放旧的内存区。内存分配由分配子（allocator）完成。
vector支持随机访问迭代子，具有最强的功能。vector的迭代子通常实现为vector元素的指针。
String是basic_string<char>的实现，在内存中是连续存放的，为了提高效率，都会有保留内存。其内存是在堆中分配的，所以串的长度可以很大，而char[] 是在栈中分配的，长度受到可使用的最大栈长度限制。和C字符数组相比较，其功能也非常强大。
Array是大小固定，顺序存储的数组。由于其内存结构简单、而且支持随机的访问。
列表（list）是由双向链表（doubly linked list）组成的。支持的迭代子类型为

双向迭代子。

双端队列（deque）（double-ended queue）类。双端队列允许在队列的两端进行操作。它是以顺序表为基础的，所以它能利用下标提供有效的索引访问，它支持随机访问迭代子。它放松了访问的限制，对象既可从队首进队，也可以从队尾进队，同样也可从任一端出队。而且除了可从队首和队尾移走对象外，也支持通过使用下标操作符“[ ]”进行访问。

当要增加双端队列的存储空间时，可以在内存块中对deque两端分别进行分配。并且新分配的存储空间保存为指向这些块的指针数组，这样双端队列可以利用不连续内存空间。因此它的迭代子比 vector 的迭代子更加智能化。为双端队列分配的存储块，往往要等删除双端队列时才释放，它比重复分配（再分配和释放）更有效，但也更浪费内存。使用双端队列容器类来实现矢量容器类所能实现的各种数据结构要更灵活，方便。

单向列表（forward_list）支持在其任何地方快速插入和删除元素，不支持快速的随机访问。它被实现为单向链表，当不需要双向迭代的时候，该容器具有更高的空间利用率。

11.3 关联容器有哪几种？简单介绍它们是怎样组成的？各有什么特点？

解：关联容器主要有：集合（set），多重集合（multiset），映射（map）和多重映射（multimap）。

集合和多重集合类提供了控制数值集合的操作，其中数值是关键字，即不必另有一组值与每个关键字相关联。集合与多重集合类的主要差别在于多重集合允许重复的关键字（key），而集合不允许重复的关键字。集合和多重集合通常实现为红黑二叉排序树。元素的顺序由比较器函数对象（comparator function object）确定。如对整型multiset，只要用比较器函数对象less<int>排序关键字，元素即可按升序排列。

映射和多重映射类提供了操作与关键字相关联的映射值（mapped value）的方法。映射和多重映射的主要差别在于多重映射允许存放与映射值相关联的重复关键字，而映射只允许存放与映射值一一对应的单一关键字。

多重集合关联容器用于快速存储和读取关键字。多重映射和映射关联容器类用于快速存储和读取关键字与相关值（关键字/数值对，key/value pair）。

11.4 向 STL 算法传递函数有几种方式？请举例说明。

答案：

（1）使用函数。

（2）使用函数对象。

（3）使用lambda表达式。

```
1 #include <vector>
2 #include <algorithm>
```

```
#include <iostream>
using namespace std;
bool fun(int a, int b){
    return a < b;
}
struct Compare{
    bool operator()(int a, int b){
        return a<b;
    }
};
int main(){
    vector<int> x={9,2,5,3,1};
    sort(x.begin(),x.begin()+2,fun); //通过函数
    for(auto i:x) cout<<i<<" "; // (2 9) 5 1 3
    cout<<endl;
    sort(x.begin()+2,x.end(),Compare()); //通过函数对象
    for(auto i:x) cout<<i<<" "; //2 9 1 3 5
    cout<<endl;
    sort(x.begin(),x.end(),[](int a,int b){ // 通过lambda表达式
        return a<b;});
    for(auto i:x) cout<<i<<" "; //1 2 3 5 9
    cout<<endl;
    return 0;
}
```

输出结果：

```
2 9 5 3 1
2 9 1 3 5
1 2 3 5 9
```

11.5 利用 cin 读入一组整数（1, 2, 3, ..., 100），把它们存入一个 vector 对象，计算它们的平均值（整型类型）并打印输出。

答案：

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

```
int main()
{
    vector<int> numbers;
    int sum = 0;
    for (int i = 0; i < 100; ++i) {
        numbers.push_back(i+1);
        }

    for (vector<int>::iterator iter = numbers.begin();
          iter != numbers.end(); ++iter) {
        sum += *iter;
    }
    cout << "\平均值n:\t"<< sum/numbers.size()<< endl;
    system("pause");
    return 0;
}
```

输出结果：
平均值:50

11.6 使用映射（map），建立阿拉伯的数字 0 到 9 和英文单词 Zero 到 Nine 的映射关系，实现输入一个阿拉伯数字，输出与之对应的英文单词。例如，输入1，输出One。

答案：

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

typedef map<int, char*> ismap;

int main() {
    int i = 1, j;
    char* str[10] = { "One","Two","Three","Four",
    "Five","Six","Seven","Eight","Nine","Ten" };
    ismap map1;
    ismap::iterator it;
    for (i = 1; i <= 10; i++)
```

```
        map1.insert(ismap::value_type(i, str[i - 1]));
    for (it = map1.begin(); it != map1.end(); it++)
        cout <<(*it).first<<'\t'<<(*it).second<< endl;
    cout << map1.size() << endl;
    cout << "请输入的数字：1~10" << endl;
    cin >> j;
    it = map1.find(j);
    cout <<(*it).first<<'\t'<<(*it).second<< endl;
    return 0;
}
```

输出结果：

```
1   One
2   Two
3   Three
4   Four
5   Five
6   Six
7   Seven
8   Eight
9   Nine
10   Ten
10
请输入数字：1～10
1 2 3 4 5 6 7 8 9 10
1 One
```

11.7 编写程序统计并输出所读入的单词出现的次数。

答案：

```
#include <iostream>
#include <string>
#include <utility>
#include <map>
using namespace std;

```

```
int main() {
    map<string, int > wordCount;
    string word;
    while (cin >> word) {
        ++wordCount[word];
        cout << "单词'" << word << "出现了'"
            << wordCount[word] << " 次。" << endl;
    }
    system("pause");
    return 0;
}
```

输出结果：

```
Whatever is worth doing is worth doing well.


Whatever is worth doing is worth doing well
单词‘Whatever’出现了1次.
单词‘is’出现了1次.
单词‘worth’出现了1次.
单词‘doing’出现了1次.
单词‘is’出现了2次.
单词‘worth’出现了2次.
单词‘doing’出现了2次.
单词‘well’出现了1次.
```

11.8 假设有一个存储了10 个元素（1,2,3, ..., 10）的 vector 对象，将其中第3 到第7 个位置上的元素以逆序复制给list 对象。

答案：

```
#include <iostream>
#include <iterator>
#include <list>
#include <vector>
using namespace std;
int main() {
    int ia[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
    vector<int> iVec(ia, ia + 10);
```

```
    for (vector<int>::iterator it = iVec.begin();
     it != iVec.end(); ++it)
        cout << *it << " ";
    cout << endl;
    list<int> iLst(iVec.rbegin()+3,iVec.rbegin()+8);
    for (list<int>::iterator it = iLst.begin();
     it != iLst.end(); ++it)
        cout << *it << " ";  cout << endl;
    system("pause");
    return 0;
}
```

输出结果：

```
1 2 3 4 5 6 7 8 9 10
7 6 5 4 3
```

11.9 编写程序将 list 容器的元素（1, 2, 3, ..., 10）逆序输出。

答案：

```
#include <iostream>
#include <iterator>
#include <list>
using namespace std;
int main() {
    list<int> ilist = { 1,2,3,4,5,6,7,8,9,10 };
    list<int>::iterator it = ilist.end();
    --it;
    for (; it != ilist.begin(); --it)
        cout << *it << endl;
    return 0;
}
```

输出结果：

```
10
9
8
7
6
```

5
4
3
2

11.10 使用迭代器将 string 对象中的字符（如:IlikeCplusplus）都改为大写字母。

答案:

```
#include <iostream>
#include <string>
using namespace std;
int main() {
    string str;
    cout << "输入字符串:";
    cin >> str;
    cout << "输入的字符串:" << str << endl;
    for (string::iterator it = str.begin();
     it != str.end(); ++it) {
        *it = toupper(*it);
    }
    cout << "处理后输出:" << str << endl;
    system("pause");
    return 0;
}
```

输出结果:
输入字符串:Ilikecplusplus

输入的字符串:Ilikecplusplus

处理后输出:ILIKECPLUSPLUS

11.11 定义一个 map 对象，其元素的键是家族姓氏，而值则是存储该家族孩子名字的 vector 对象。为这个 map 容器输入至少六个条目。通过基于家族姓氏的查询检测你的程序，查询并输出该家族所有孩子的名字。

答案:

```
#include <iostream>
#include <string>
#include <vector>
#include <utility>
#include <map>
using namespace std;

int main() {
    map< string, vector<string> > children;
    string famName, childName;
    do {
        cout<<"输入家庭名称(ctrl+z to end):"<<endl;
        cin >> famName;
        if (!cin)
            break;
        vector<string> chd;
        pair< map< string,
        vector<string> >::iterator, bool >ret=
        children.insert(make_pair(famName, chd));
        if (!ret.second) {
         cout<<"已存在的家庭名称:"<<famName<<endl;
             continue;
          }
        cout <<"输入孩子的名字(ctrl+z to end):"<<endl;
        while (cin >> childName)
            ret.first->second.push_back(childName);
        cin.clear();
    } while (cin);
     cout<<endl;
     cin.clear();
     cout << "输入要搜索的家庭名称: "<< endl;
     cin >> famName;

     map<string,vector<string>>::iterator iter=
      children.find(famName);
     if (iter == children.end())
```

```
37          cout <<"对不起，不存在该家庭名称: "
38           << famName << endl;
39      else {
40          cout <<"输出孩子的名字: " << endl;
41          vector<string>::iterator it=
42          iter->second.begin();
43          while (it != iter->second.end())
44              cout << *it++ << endl; }
45
46      system("pause");
47      return 0;
48  }
```

输出结果:
输入家庭名称(ctrl + z to end):
我爱我家
输入孩子的名字(ctrl + z to end):
张三
张华
张丽
ctrl + z
输入家庭名称(ctrl + z to end):
快乐家族
输入孩子的名字(ctrl + z to end):
李四
李思
李林
ctrl + z
输入家庭名称(ctrl + z to end):
我爱我家
已存在的家庭名称:我爱我家
输入家庭名称(ctrl + z to end):
ctrl + z

输入要搜索的家庭名称:
我爱我家

输出孩子的名字：
张三
张华
张丽

11.12 编写程序建立作者及其作品的 multimap 容器，使用 find 函数在 multimap 中查找元素，并调用 erase 将其删除。当所寻找的元素不存在时，确保你的程序依然能正确执行。

答案：

```
#include <iostream>
#include <map>
#include <utility>
#include <string>
using namespace std;
int main() {
    multimap< string, string > mmapAuthor;
    string author, book;
    cout << "输入作者名和书名(ctrl+z to end ):";
    cout<<endl;
    while (cin >> author >> book) {
        mmapAuthor.insert(make_pair(author,book));
    }

    string strSearchItem;
    cout << "输入你想查询的作者名字: ";
    cin.clear();
    cin >> strSearchItem;
    multimap<string, string >::iterator iter =
     mmapAuthor.find(strSearchItem);
    typedef multimap<string,string>::size_type sz_type;
    sz_type amount = mmapAuthor.count(strSearchItem);
    for (sz_type cnt=0;cnt!=amount;++cnt,++iter){
        cout << "查询到的作者名:" << iter->first
         << endl<<"该作者的书名:"<<iter->second
         << endl;
    }
```

```
28      amount = mmapAuthor.erase(strSearchItem);
29      if (amount) {
30          cout <<"删除该作者的"<<amount<<"本书。"<< endl;
31      }
32      cout << endl;
33      system("pause");
34      return 0;
35  }
```

输出结果:

```
输入作者名和书名( ctrl + z to end ):
鲁迅《活着》
鲁迅《呐喊》
鲁迅《彷徨》
ctrl + z
输入你想查询的作者名字:鲁迅
查询到的作者名:鲁迅
该作者的书名:《活着》
查询到的作者名:鲁迅
该作者的书名:《呐喊》
查询到的作者名:鲁迅
该作者的书名:《彷徨》
删除该作者的3本书.
```

11.13 编写程序使用 reverse_iterator 对象以逆序输出 vector 容器对象的内容。

答案:

```
1  #include <iostream>
2  #include <iterator>
3  #include <vector>
4  using namespace std;
5
6  int main() {
7      int ia[] = { 0, 1, 2, 3, 4 };
8      vector<int> iVec(ia, ia + 5);
9      for (vector<int>::reverse_iterator rit
```

```
10        = iVec.rbegin();
11        rit != iVec.rend(); ++rit){
12            cout << *rit << " ";
13      }
14       cout << endl;
15       system("pause");
16       return 0;
17  }
```

输出结果：

```
4 3 2 1 0
```

第12章　工具与技术

12.1 简述命名空间的作用。

答案：命名空间相当于一个容器，它里面包含了逻辑结构上互相关联的一组类、模板、函数等。也就是说如果某些“对象”在逻辑上有关系，我们就可以将它们放到一个命名空间里用以和外界进行区分。命名空间一个显著的特点是命名空间内的变量(类等)名可以和命名空间以外的重名。这可以用来将不同人写的代码进行整合（注意：如果一个函数的定义没有在其对应的命名空间里，必须要使用作用域解析符::来指定函数的命名空间。不可以在命名空间以外定义一个命名空间中不存在的新成员）。

12.2 简述C++语言中异常处理的基本思想和处理机制。是不是每个函数都要抛出异常？

答案：基本思想：将异常检测与异常处理分离，异常检测部分检测到异常的存在时，抛出一个异常对象给异常处理代码。通过该异常对象，独立开发的异常检测部分和异常处理部分能够就程序执行期间所出现的异常情况进行通信。

执行机制：（1）若有异常则通过throw操作创建一个异常对象并抛出。（2）将可能抛出异常的程序段放在try块之中。控制通过正常顺序执行到达try块，然后执行try子块内的保护段。（3）如果在保护段执行期间没有引发异常，那么跟在try子块后的catch字句就不执行。程序继续执行紧跟try块最后一个catch子句后面的语句。（4）catch子句按其在try 块后出现的顺序被检查。类型匹配的catch子句将被捕获并处理异常（或重抛出异常）（5）如果找不到匹配的处理代码，则自动调用terminate，默认为abort()终止程序。

如果函数对每个遇到的异常都知道该如何处理，就不必抛出异常了。

12.3 什么是虚继承？它有什么作用？

答案：为了解决从不同途径继承来的同名的数据成员在内存中有不同的拷贝造成数据不一致问题，将共同基类设置为虚基类。这时从不同的路径继承过来的同名数据成员在内存中就只有一个拷贝，同一个函数名也只有一个映射。这样不仅就解决了二义性问题，也节省了内存，避免了数据不一致的问题。

12.4 写出程序运行结果。

(1)

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

int main() {
    int s(0), a[5]{ 1,2,3,4,5 };
    for (int i(0); i <= 5; i++) {
        try {
        if (i >= 5) throw i;
        s += a[i];
        }
        catch (int i) {
            cout << "越界" << i << endl;
        }
        cout << s << endl;
    }
}
```

答案:

1
3
6
10
15
越界5
15

(2)

```
#include <iostream>
#include <iostream>
using namespace std;
class A {
    int a;
public:
    A(int i) :a(i) {};
```

```
8       void print() {
9           cout << a << endl;
10      }
11  };
12  class B {
13      int b;
14  public:
15      B(int j) :b(j) {};
16      void print() {
17          cout << b << endl;
18      }
19  };
20  class C :public A, public B {
21      int c;
22  public:
23      C(int i, int j, int k): A(i), B(j), c(k) {}
24     void get() {
25          A::print();
26          B::print();
27      }
28  };
29
30  int main() {
31      C x(5, 8, 10);
32      x.get();
33      x.A::print();
34      return 0;
35  }
```

答案:

5

8

5

(3)

```
1   #include<iostream>
```

```
#include<iostream>
#include<memory>
using namespace std;
class A {
    protected:
        int v = 1;
        public:
        virtual ~A() {}
        virtual int fun() {
            return ++v;
        }
};
 class B :public A {
    public:
        int fun() { return v += 2; }
        int test() { return v += 3; }
};
 int main() {
    unique_ptr<A> p(new B());
    cout << p->fun() << endl;
    B *q = dynamic_cast<B*>(p.get());
    cout << q->test();
    return 0;
 }
```

答案：

3

6

12.5 编写程序求输入数的平方根。要求：使用异常处理机制处理输入负数的情况。

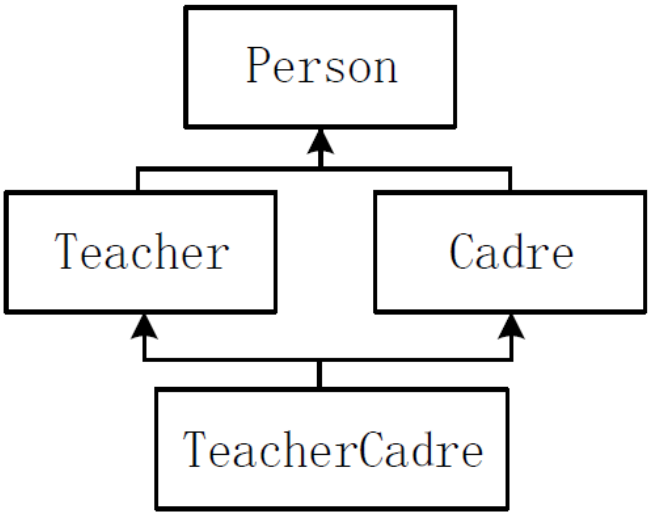
答案：

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

void main()
```

```
{
    double number;
    double result;
    cout << "请输入一个数: ";
    cin >> number;
    try
    {
        if (number < 0)
        {
            throw exception("输入的数是负数!");
         }
        result = sqrt(number);
        cout << "平方根是: "<<result<< endl;
    }
    catch (exception e)
    {
        cout << e.what() << endl;
    }
}
```

输出的结果:
(1)请输入一个数: 2
平方根是: 1.41421
(2)请输入一个数: -2
输入的数是负数!

12.6 分别设计教师（Teacher）类和干部（Cadre）类，采用多继承方式由这两个类派生出教师兼干部类（TeacherCadre）类。要求:
（1）设计一个基类Person，包含姓名、年龄、性别、地址、电话等数据成员。Teacher和Cadre都继承Person，它们的继承关系如下图所示:



（2）在Teacher 类中新增数据成员职称（title），在Cadre 类中新增数据成员职务（post）；

（3）在类体中声明成员函数，在类体外定义成员函数。

（4）在 TeacherCadre 的成员函数 display 中输出姓名、年龄、性别、地址、电话、职称和职务等信息。

答案：

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

class Person {
protected:
    string m_name;
    int m_age;
    char m_sex;
    string m_addr;
    string m_tel;
public:
    virtual void display();
};
```

```
15
16 class Teacher : virtual public Person {
17 public:
18     Teacher(string nam, int a, char s, string tit,
19         string ad, string t);
20     void display();
21 protected:
22     string m_title;
23 };
24
25 class Cadre : virtual public Person {
26 public:
27     Cadre(string nam, int a, char s, string p,
28         string ad, string t);
29     void display();
30 protected:
31     string m_post;
32 };
33
34 void Person::display() {
35     cout << "name:" << m_name << endl;
36     cout << "age:" << m_age << endl;
37     cout << "sex:" << m_sex << endl;
38     cout << "address:" << m_addr << endl;
39     cout << "tel:" << m_tel << endl;
40 }
41
42 class TeacherCadre :public Teacher, public Cadre {
43 public:
44     TeacherCadre(string nam, int a, char s,
45         string tit, string p, string ad, string t);
46     void display();
47 };
48
49 Teacher::Teacher(string nam, int a, char s,
50     string tit, string ad, string t) {
```

```
51      m_name = nam;
52      m_age = a;
53      m_sex = s;
54      m_title = tit;
55      m_addr = ad;
56      m_tel = t;
57  }
58
59  Cadre::Cadre(string nam, int a, char s,
60      string p, string ad, string t) {
61      m_name = nam;
62      m_age = a;
63      m_sex = s;
64      m_post = p;
65      m_addr = ad;
66      m_tel = t;
67  }
68
69  TeacherCadre::TeacherCadre(string nam, int a,
70      char s, string tit, string p, string ad, string t)
71      :Teacher(nam, a, s, tit, ad, t),
72      Cadre(nam, a, s, p, ad, t) {}
73
74  void Teacher::display() {
75      Person::display();
76      cout << "title:" << m_title << endl;
77  }
78
79  void Cadre::display() {
80      Person::display();
81      cout << "post:" << m_post << endl;
82  }
83
84  void TeacherCadre::display() {
85      cout << "name:" << m_name << endl;
86      cout << "age:" << m_age << endl;
```

```
87      cout << "sex:" << m_sex << endl;
88      cout << "address:" << m_addr << endl;
89      cout << "tel:" << m_tel << endl;
90      cout << "title:" << m_title << endl;
91      cout << "post:" << m_post << endl;
92  }
93
94  int main() {
95      string name, title, post, address, tele;
96      int age;
97      char sex;
98      float wages;
99      cin >> name >> age;
100     cin >> sex >> title >> post;
101     cin.ignore(2, '\n');
102     getline(cin, address);
103     cin >> tele >> wages;
104     TeacherCadre te_ca(name, age, sex, title,
105        post, address, tele);
106     te_ca.display();
107     return 0;
108 }
```

输出的结果：

```
Tom 55 M professor Dean
CUG
123123
200000
name:Tom
age:55
sex:M
address:CUG
tel:123123
title:professor
post:Dean
```