C++程序设计

（基于C++11标准）

第1-6章部分习题答案

李长河 童恒建 叶亚琴 杨鸣 编著

电子工业出版社
北京

目　　录

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 = '\\'　(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)`

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）

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

（3）

```
1 for (k = 1, k <= n, k = k + 1);
2 {
3     f = f + (f * 1.5) % k;
4 }
```

（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？

```
1 int x = 3, y = 100;
2 while (y / x > 5)
3     if (y - x > 25)     x = x + 1;
4     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

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？

```
const int ci=5, &cri=ci;
int i=6, &ri=i, *p=&i;
auto ans1 = ci;
auto ans2 = &ci;
auto ans3 = ri;
auto &ans4 = i;
auto *ans5 = &i;
auto ans6 = &i;
decltype(i) dec1 = i;
decltype(i) dec2 = ci;
decltype(cri) dec3 = 5;
decltype(ri + 0) dec4 = 5;
decltype(*p) dec5 = i;
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。

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

```
6           str1[i] = 'a';
7       }
8       str1[i] = '\0';
9       strcpy(str, str1);
10      return 0;
11  }
```

4.11 下面的代码是否正确吗？如果不正确，请改正。

```
1  vector<int> ivec;
2  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。

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

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

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。