第一章

【1】简述冯·诺依曼计算机的组成及工作过程。
【解】冯·诺依曼计算机由五大部分组成：运算器、控制器、存储器、输入设备和输出设备。
运算器是真正执行计算的组件。它在控制器的控制下执行程序中的指令，完成算术运算、逻辑运算和移位运算等。
控制器用于协调机器其余部分的工作。控制器依次读入程序的每条指令，分析指令，命令各其他部分共同完成指令要求的任务。
存储器用来存储数据和程序。存储器可分为主存储器和外存储器。主存储器又称为内存，用来存放正在运行的程序和程序处理的数据。外存储器用来存放长期保存的数据。
输入/输出设备又称外围设备，它是外部和计算机交换信息的渠道。输入设备用于输入程序、数据、操作命令、图形、图像和声音等信息。输出设备用于显示或打印程序、运算结果、文字、图形、图像等，也可以播放声音和视频等信息。
【2】简述寄存器、主存储器和外存储器的异同点。
【解】 寄存器、主存储器和外存储器都是用于存储信息，但级别不同。从功能来讲，寄存器存储的是运算器当前正在运算的数据或当前正在执行的那条指令。主存储器保存的是正在运行的程序代码和数据。外存储器保存的是需要长期保存的数据。从容量角度来讲，寄存器容量最小，内存次之，外存储器最大。从访问速度来讲，寄存器最快，内存次之，外存最慢。
【3】所有的计算机能够执行的指令都是相同的吗？
【解】 计算机能够执行的指令是直接由硬件完成的，与硬件设计有关。不同的硬件设计产生不同的指令系统。因此，不同类型的计算机所能执行的指令是不同的。
【4】投入正式运行的程序就是完全正确的程序吗？
【解】程序的调试及测试不可能将程序所有的路径、所有的数据都执行一遍，因此只能发现并改正程序中的某些错误，而不能证明程序是正确的。
【5】为什么需要编译？为什么需要链接？
【解】计算机硬件能“认识”的只有机器指令，它并不认识程序设计语言，如C++。要使计算机能够执行C++写的程序，必须把C++的程序翻译成计算机认识的机器语言，机器语言版的程序称为目标程序。源程序到目标程序的翻译是由编译器完成的。
程序员编写的程序通常会用到其他程序员或C++系统已经编好的一些工具，程序运行时会用到这些工具的代码。需要将目标文件和这些工具的目标文件捆绑在一起，这个过程称为**链接**。链接器就是完成这个链接工作。链接以后的代码称为一个**可执行文件**。这是能直接在某台计算机上运行的程序。
【6】调试的作用是什么?如何进行程序调试？
【解】调试的作用是尽可能多地找出程序中逻辑错误，使程序能给出正确的答案。调试一般需要运行程序，通过观察程序的阶段性结果来找出错误的位置和原因，并改正错误。
【7】试列出一些常用的系统软件和应用软件。
【解】常用的系统软件有：操作系统、编译系统、数据库系统等。应用软件又分成通用的应用软件和专用的应用软件。通用的应用软件提供一些常规的应用，如文字处理软件word，媒体播放软件Media Play等。专用的应用软件是某个领域专用的一些软件，如银行系统、证券交易系统等。
【8】为什么在不同生产厂商生产的计算机上运行C++程序需要使用不同的编译器。
【解】因为不同的生产厂商生产的计算机有不同的机器语言，所以需要不同的编译器将同样的C++程序翻译成不同的机器语言。
【9】什么是源程序？什么是目标程序？为什么目标程序不能直接运行？
【解】用某种程序设计语言写的程序称为源程序，源程序经过编译产生的机器语言的程序称为目标程序。因为程序可能用到了一些其他程序员写好的程序，没有这些工具程序的代码整

个程序就无法运行，因此需要将目标程序和这些工具的目标程序链接在一起后才能运行。
【11】为什么不直接用自然语言和计算机进行交互，而要设计专门的程序设计语言？
【解】自然语言太复杂，而计算机本身（机器语言）的功能又非常简单，如果要将自然语言作为人机交互的工具，编译器的设计与实现必将非常的复杂。另外，自然语言太灵活，理解自然语言需要一些背景知识，否则会产生二义性，这也给计算机实现带来了很大的麻烦。
【12】试列举出高级语言的若干优点（相比与机器语言）。
【解】首先高级语言更接近于自然语言和人们熟悉的数学表示，学起来比较方便。其次高级语言功能比机器语言强。一般的机器语言只能支持整数加法、移位、比较等操作，而高级语言能执行复杂的算术表达式、关系表达式和逻辑表达式。高级语言可以使程序员在较高的抽象层次上考虑问题，编程序比较容易。第三，高级语言具有相对的机器独立性，在一台机器上用高级语言编写的程序可以在另外一台不同厂商生产的计算机上运行，这使得程序有较好的可移植性，有利于代码重用。
【13】为什么不同类型的计算机可以运行同一个C++程序，而不同类型的计算机不能运行同一个汇编程序？
【解】因为不同类型的计算机上有不同的C++编译器，可以将同一个C++程序编译成不同机器上的机器语言表示的目标程序。而汇编程序仅是机器语言的另一种表现形式。不同类型的计算机有不同的机器语言，也就有不同的汇编语言。
【14】机器语言为什么要用难以理解、难以记忆的二进制比特串来表示指令，而不用人们容易理解的符号来表示？
【解】因为计算机是由逻辑电路组成的，而0、1正好对应于逻辑电路中的两种电平信号，可以直接翻译成控制信号，使计算机硬件实现比较容易。如果采用人比较容易理解的符号，如英文、中文或者数学符号，则计算机需要用硬件将这些符号翻译成控制信号，使硬件设计非常复杂，甚至无法实现。
【15】为什么电视机只能播放电视台发布的电视，DVD播放机只能播放DVD碟片，而计算机却既能当电视机用，又能当DVD机用，甚至还可以游戏机用？
【解】电视机只能播放电视台发布的电视，DVD播放机只能播放DVD碟片，这是因为设计时已经规定好它们的功能。而计算机有一个开放的平台，具有学习的功能，可以允许程序员“教会”它们新的知识和技能。只要编写了能完成相应功能的程序，计算机就具备了相应的功能。
【16】说明下面概念的异同点：
（1）硬件和软件
（2）算法与程序
（3）高级语言和机器语言
（4）语法错误与逻辑错误
【解】
（1）硬件和软件：计算机的硬件是计算机的“肉体”，是看得见、摸得着的实体，它只能做一些非常简单的工作。计算机的软件是计算机的“灵魂”。“灵魂”指导“肉体”完成一项项的工作。当你买了一台计算机后，它的硬件是不变的，但是你可以让它“学习”，即安装不同的软件，你的计算机就有了不同的能力。
（2）算法与程序：算法是按照计算机能够完成的基本功能，设计出的解决某一问题的基本步骤。用某一种程序设计语言描述的算法称为程序。
（3）高级语言和机器语言：机器语言是计算机硬件具备的功能的抽象，高级语言是面向程序员的语言，比较接近数学表示，使程序容易编写。
（4）语法错误是指程序的表述没有完全符合程序设计语言的语法规范。逻辑错误是指算法设计过程中的错误或漏洞。

第二章 简答题

【1】程序开头的注释有什么作用？

【解】程序开头的注释是对程序整体的介绍。一般包括源文件的名称、程序的功能、作者、创建日期、修改者、修改日期、修改内容等。程序注释还可以描述程序中特别复杂的部分的实现方法和过程，给出如何改变程序行为的一些建议等。当程序在将来的某一天需要修改时，程序员可以通过这些注释而不是程序本身来了解程序是如何工作的。

【2】库的作用是什么？

【解】库是一些常用工具的集合，这些工具是由其他程序员编写的，能够完成特定的功能。当程序员在编程时需要用到这些功能时，不需要再自己编程解决这些问题，只需要调用库中的工具。这样可以减少重复编程。

【3】在程序中采用符号常量有什么好处？

【解】采用符号常量主要有两个好处：提高程序可读性和可维护性。

【4】C++有哪两种定义符号常量的方法？C++建议的是哪一种？

【解】第一种是C语言的风格：#define　符号常量　值

第二种是C++语言的风格：const 类型 符号常量 = 值；

C++建议用第二种。用#define定义符号常量有两个问题：一是所定义的符号常量无法进行类型检查；二是#define的处理只是简单的字符串替换，可能会引起一些意想不到的错误。而C++的风格指明了常量的类型，同时是将右边的表达式的值计算出来后再与符号常量关联起来。例如有定义

```
#define ABC 3+5
```

程序中有语句

```
x = 3 * ABC;
```

则x 的结果是14，即3 * 3 + 5，而不是24，即3 * 8。但如果用

```
const int ABC 3+5
```

则结果是24。

【5】C++定义了一个称为cmath的库，其中包括常用的数学函数。要访问这些函数，需要在程序中引入什么语句？

【解】需要有一个编译预处理命令：#include <cmath>，表示程序用到这个库。

【6】每个C++语言程序中都必须定义的函数的名称是什么？

【解】每个C++语言程序中都必须定义的函数的名称是main。main函数是C++程序的主程序，是程序执行的入口。

【7】如何定义两个名为num1和num2的整型变量？如何定义三个名为x、y、z的实型双精度变量？

【解】定义两个名为num1和num2的整型变量可用语句

```
int num1, num2;
```

定义三个名为x、y、z的实型双精度变量

```
double x, y, z;
```

【8】简单程序通常由哪三个阶段组成？

【解】简单程序通常由输入阶段、计算阶段和输出阶段组成。输入阶段接收用户输入的需要加工的数据。计算阶段将输入的数据加工成输出数据。输出阶段将加工后的数据输出给用户。在编写程序时，最好在各阶段之间插入一个空行，使程序逻辑更加清晰。

【9】一个数据类型有哪两个重要属性？

【解】第一个是该类型的数据在内存中是如何表示的，第二个是这类数据允许执行哪些操作。

【10】两个短整型数相加后，结果是什么类型？

【解】短整型数据在执行算术运算时会先被转换成标准的整型然后执行整型运算，所以两个短整型的整数相加后的结果是整型。

【11】算术表达式 true + false 的结果是多少？结果值是什么类型的？

【解】布尔型数据在执行算术运算时会先被转换成标准的整型然后执行整型运算，true转换位1， false转换为0，所以true+false的结果是整型1。

【12】说明下列语句的效果，假设i、j和k声明为整型变量：

```
i = (j = 4) * (k = 16);
```

【解】j的值为4，k的值为16，i的值为64。

【13】用怎样的简单语句将x和y的值设置为1.0（假设它们都被声明为double型）？

【解】可以用嵌套赋值：x = y = 1.0;

【14】假如整型数用两个字节表示，写出下列各数在内存中的表示，并写出它们的八进制和十六进制表示：

10　32　240　-1　32700

【解】整型数在机器内部被表示为补码形式，所以这些整数在内存中的表示以及八进制、十六进制表示如下表所示。

	内存中的表示	八进制	十六进制
10	0000000000001010	12	a
32	0000000000100000	40	20
240	0000000011110000	360	f0
-1	1111111111111111	177777	ffff
32700	0111111110111100	77674	7fbc

【15】辨别下列哪些常量为C++语言中的合法常量。对于合法常量，分辨其为整型常量还是浮点型常量：

42　1,000,000　-17　3.1415926　123456789　-2.3　0.000001　20

1.1E+11　2.0　1.1X+11　23L　2.2E2.2

【解】这些值的情况如下表所示。

42	1,000,000	-17	3.1415926	123456789	-2.3	0.000001
整型	非法	整型	浮点型	整型	浮点型	浮点型
20	1.1E+11	2.0	1.1X+11	23L	2.2E2.2	
整型	浮点型	浮点型	非法	整型	非法	

【16】指出下列哪些是C++语言中合法的变量名？

a. x
b. formulal
c. average_rainfall
d. %correct
e. short
f. tiny
g. total output
h. aReasonablyLongVariableName
i. 12MonthTotal
j. marginal-cost
k. b4hand
l. _stk_depth

【解】这些符号的情况如下表所示。

x	合法
formulal	合法
average_rainfall	合法
%correct	不合法
short	不合法
tiny	合法
total output	不合法
aReasonablyLongVariableName	合法
12MonthTotal	不合法
marginal-cost	不合法
b4hand	合法
_stk_depth	合法

【17】在一个变量赋值之前，可以对它的值做出什么假设？

【解】可以假设它的值是一个随机值。

【19】若k已被定义为int型变量，当程序执行赋值语句

```
k = 3.14159;
```

后，k的值是什么？若再执行下面语句，k的值是什么？

```
k = 2.71828;
```

【解】在执行了k = 3.14159;后，k的值为3。在执行了k=2.71828；后，k的值为2。

【22】以下哪些是合法的字符常量？

'a' , "ab" , 'ab', '\n', '0123', '\0123', "m"

【解】

'a'	"ab"	'ab'	'\n'	'0123'	'\0123'	"m"
合法	非法	非法	合法	非法	合法	非法

【23】写出完成下列任务的表达式：

a．取出整型变量n的个位数

b．取出整型变量n的十位以上的数字

c．将整型变量a和b相除后的商存于变量c，余数存于变量d

d．将字符变量ch中保存的小写字母转换成大写字符

e．将double型的变量d中保存的数字按四舍五入的规则转换成整数

【解】

```
a. n % 10
b. n / 10
c. c = a / b, d = a % b
d. ch - 'a' + 'A'
e. int(d + 0.5)
```

【25】若变量k为int类型，x为double类型，执行了k = 3.1415; x = k; 后，x和k的值分别是多少？

【解】k的值是3。x的值为3.0。

【26】已知华氏温度到摄氏温度的转换公式为

$$C=\frac{5}{9}(F-32)$$

某同学编写了一个将华氏温度转换成摄氏温度的程序：

```
int main()
{   int c, f;

    cout << "请输入华氏温度：" ;
    cin >> f ;
    c = 5 / 9 * ( f - 32) ;
    cout << "对应的摄氏温度为：" << c;

    return 0;
}
```

但无论输入什么值，程序的输出都是0.请你帮他找一找哪里出问题了。

【解】由于该程序中将变量f定义为int型，表达式5 / 9 * (f - 32)中的所有运算数都是int类型，C++执行整型运算。5 / 9 为0,0乘任何数都为0，所以c的值永远为0。只要将5改成5.0，程序就能得到正确的结果。

第二章 程序设计题

【2】设计一个程序完成下述功能：输入两个整型数，输出这两个整型数相除后的商和余数。

【解】

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

int main()
{
    int num1, num2,quotient, remainder;

    cout << "请输入两个整型数：";
    cin >> num1 >> num2;

    quotient = num1 / num2;                          //计算商
    remainder = num1 % num2;                         //计算余数

    cout << num1 << " / " << num2 << "的商为：" << quotient << endl;
    cout << num1 << " % " << num2 << "的余数为：" << remainder << endl;

    return 0;
 }
```

【3】输入9个小于8位的整型数，然后按3行打印，每一列都要对齐。例如输入是：1、2、3、11、22、33、111、222、333，输出为

```
1      2      3
11     22     33
111    222    333
```

【解】

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

int main()
{
    int num1, num2, num3, num4, num5, num6, num7, num8, num9;

    cout << "请输入9个整型数：";
    cin >> num1 >> num2 >> num3 >> num4 >> num5 >> num6 >> num7 >> num8 >> num9;

    cout << num1 << '\t' << num2 << '\t' << num3 << endl;
    cout << num4 << '\t' << num5 << '\t' << num6 << endl;
    cout << num7 << '\t' << num8 << '\t' << num9 << endl;

    return 0;
}
```

【4】某工种按小时计算工资。每月劳动时间（小时）乘以每小时工资等于总工资。总工资扣除10%的公积金，剩余的为应发工资。编写一个程序从键盘输入劳动时间和每小时工资，输出应发工资。

【解】

```
#include <iostream>
using namespace std;
int main()
{
    int time, yuanPerHour, totalSalary, salary;

    cout << "请输入每小时工资：";
    cin >> yuanPerHour;
    cout << "请输入本月劳动时间：";
    cin >> time;
```

```
    totalSalary = time * yuanPerHour;                    //  计算总工资
    salary = totalSalary - 0.1 * totalSalary ;           //  计算应发工资

    cout << "本月应得工资为：" << salary << endl;

    return 0;
}
```

【5】编写一个程序，用于水果店售货员结账。已知苹果每斤2.50元，鸭梨每斤1.80元，香蕉每斤2元，橘子每斤1.60元。要求输入各种水果的重量，打印应付金额。再输入顾客付款数，打印应找零的金额。

【解】

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

int main()
{
    const double priceOfApple = 2.50;
    const double priceOfPear = 1.80;
    const double priceOfBanana = 2.00;
    const double priceOfOrange = 1.60;

    double apple, pear, banana, orange;
    double money, income, change;

    cout << "请输入苹果  梨  香蕉  橘子的重量：";
    cin >> apple >> pear >> banana >> orange;

    money = apple * priceOfApple + pear * priceOfPear
            + banana * priceOfBanana + orange * priceOfOrange;

    cout << "你应该付" << money << "元" << endl;
    cin >> income;
    change = income - money;
    cout << "\n找零" << change << "元";

    return 0;
}
```

【6】编写一个程序完成下述功能：输入一个字符，输出它的ASCII值。

【解】

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

int main()
{
     char ch;

     cout << "请输入一个字符：";
     cin >> ch;

     cout << static_cast<int> (ch) << endl;

     return 0;
}
```

【7】假设校园电费是0.6元/千瓦时，输入这个月使用了多少千瓦时的电，算出你要交的电费。假如你只有1元、5角和1角的硬币，请问各需要多少1元、5角和1角的硬币。例如这个月使用的电量是11，那么输出为

电费：6.6

共需6张1元、1张5角的和1张1角的

【解】

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

int main()
{
    const int FEE = 6;              // 费用以角为单位
    int amount, money;

    cout << "请输入本月的用电量：";
    cin >> amount;

    money = amount * FEE;           // 计算本月应付多少角

   cout << "你本月的电费是 " << money / 10 << "元" << money % 10 << "角" << endl;
   cout << "需要支付" << money / 10 << "个1元的硬币，";
   cout <<  money %10 / 5 << "个5角的硬币，";
   cout <<  money % 5 << "个1角的硬币，" << endl;

   return 0;
}
```

【8】设计并实现一个银行计算利息的程序。输入为存款金额和存款年限，输出为存款的本利之和。假设年利率为1.2%，计算存款本利之和公式为 本金 + 本金 * 年利率 * 存款年限。

【解】

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

int main()
{
    const double RATE = 1.2;
    double principal;
    int years ;

    cout << "请输入本金（元）和存期（年）：";
    cin >> principal >> years;

    principal = principal + principal * RATE * years / 100;

    cout << "你的本利和是：" << principal << endl;

    return 0;
}
```

【9】编写一个程序读入4个整数，输出它们的平均值。程序的执行结果的示例如下：

```
请输入4个整型数： 5 7 9 6↙
5 7 9 6的平均值是6.75
```

【解】。

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

```
int main()
{
    double avg;
    int num1, num2, num3, num4 ;

    cout << "请输入4个数：";
    cin >> num1 >> num2 >> num3 >> num4;

    avg = ( num1 + num2 + num3 + num4) / 4.0;

    cout << "4个数的平均值是：" << avg << endl;

    return 0;
}
```

【10】写一个程序，输出在你使用的C++系统中int类型的数据占几个字节，double类型的数据占几个字节，short int占几个字节，float类型占几个字节。

【解】。

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

int main()
{
    cout << "int型占用了" << sizeof(int) << "个字节" << endl;
    cout << "double型占用了" << sizeof(double) << "个字节" << endl;
    cout << "short int型占用了" << sizeof(short) << "个字节" << endl;
    cout << "float型占用了" << sizeof(float) << "个字节" << endl;

    return 0;
}
```

【11】对于一个二维平面上的两个点（x1，y1）和（x2，y2），编一程序计算两点之间的距离。

【解】。

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

int main()
{
    double x1, y1, x2, y2, distance;

    cout << "请输入点1的坐标：";
    cin >> x1 >> y1;
    cout << "请输入点2的坐标：";
    cin >> x2 >> y2;

    distance = sqrt((x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2));

    cout << "(" << x1 << ", " << y1 << ") -> (" << x2 << ", " << y2
         << ")的距离是：" << distance << endl;

    return 0;
}
```

第三章　简答题

【1】写出测试下列情况的关系表达式或逻辑表达式：

a. 测试整型变量 n 的值在 0～9 之间，包含 0 和 9。
b. 测试整型变量 a 的值是否是整型变量 b 的值的一个因子。
c. 测试字符变量 ch 中存储的是一个数字字符。
d. 测试整型变量 a 的值是否是奇数。
e. 测试整型变量 a 的值是否为 5。
f. 测试整型变量 a 的值是否为 7 的倍数

【解】

```
a. n >= 0 && n <= 9
b. b % a == 0
c. ch >= '0' && ch <= '9'
d. a % 2 == 1
e. a == 5
f. a % 7 == 0
```

【2】假设myFlag声明为布尔型变量，下面的if语句会有什么问题？

```
if (myFlag == true)...
```

【解】这个语句的语法和运行结果都是正确的，但有冗余判断。表达式myFlag == true的值与变量myFlag是一样的，所以只要写成 if (myFlag)就可以了。

【3】设a = 3，b = 4，c = 5，写出下列各逻辑表达式的值。

```
a. a + b > c && b == c
b. a || b + c && b - c
c. !(a > b) && !c
d. (a != b) || (b < c)
```

【解】

```
a. a + b > c && b == c
   = true && false
   = false
b. a || b + c && b - c
   = true || b + c && b - c
   = true
c. !(a > b) && !c
  = true && false
  = false
d. (a != b) || (b < c)
  = true || (b < c)
  = true
```

【4】用一个if语句重写下列代码

```
if (ch =='E')  ++c;
if (ch =='E') cout << c << endl;
```

【解】

```
if (ch =='E')  { ++c; cout << c << endl;}
```

或者

```
if (ch =='E') cout << ++c << endl;
```

【5】用一个switch语句重写下列代码

```
if (ch == 'E' || ch =='e')
    ++countE;
else if (ch =='A' || ch =='a')
        ++countA;
     else if (ch =='I' || ch =='i')
             ++countI;
          else   cout << "error";
```

【解】

```
switch (ch) {
   case 'E': case 'e':  ++countE; break;
   case 'A' : case 'a': ++countA; break;
   case 'I' : case 'i': ++countI;; break;
   default : cout << "error";
}
```

【6】如果 a=5，b = 0, c = 1,写出下列表达式的值，以及执行了表达式后变量 a、b、c 的值：

a. a || b +=c

b. b + c && a

c. c = (a == b)

d. a -= 5 || b++ || --c

e. b < a <= c

【解】

a. true, a=5，b = 0, c = 1

b. true, a=5，b = 0, c = 1

c. 0, a=5，b = 0, c = 0

d. false, a=0，b = 1, c = 0

e. true, a=5，b = 0, c = 1

【7】修改下面的 switch 语句，使之更简洁。

```
switch (n) {
   case 0: n += x; ++x; break;
   case 1: ++x; break;
   case 2: ++x; break;
   case 3: m = m+n; --x; n = 2; break;
   case 4: n = 2;
}
```

【解】

```
switch (n) {
   case 0: n += x;
   case 1: case 2: ++x; break;
   case 3: m = m+n; --x;
   case 4: n = 2;
}
```

【8】某程序需要判断变量 x 的值是否介于 0 到 10 之间（不包括 0 和 10），程序采用如下语句

```
if (0 < x < 10) cout << "成立";
else cout << "不成立";
```

但无论 x 的值是多少，程序永远输出“成立”。为什么？

【解】在计算0 < x < 10时，由于关系运算符是左结合的，所以先计算0 < x ，该表达式的结果可能是true也可能是false。接着计算（0 < x） < 10时，将0 < x 的结果转换成整数。true转换成1， false转换成0。0和1都小于10，所以不管0 < x 的结果是什么，整个表达式的结果都为true。所以该语句永远执行then子句。

第3章 程序设计题

【1】从键盘输入3个整数，输出其中的最大值、最小值和平均值。

【解】

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

int main()
```

```
{
    double avg;
    int num1, num2, num3, max, min;

    cout << "请输入三个整数：";
    cin >> num1 >> num2 >> num3 ;

    max = min = num1;
    if (num2 > max) max = num2;
    if (num2 < min) min = num2;
    if (num3 > max) max = num3;
    if (num3 < min) min = num3;
    avg = (num1 + num2 + num3 ) / 3.0;

    cout << "最大值是：" << max << endl;
    cout << "最小值是：" << min << endl;
    cout << "平均值是：" << avg << endl;

    return 0;
}
```

【2】编一个程序，输入一个整型数，判断输入的整型数是奇数还是偶数。例如，输入11，输出为：11是奇数

【解】

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

int main()
{
    int num;

    cout << "请输入一个整数：";
    cin >> num;

    cout << num <<(num % 2 ? "是奇数" :"是偶数") << endl;

    return 0;
}
```

【4】有一个函数，其定义如下：

$$y=\begin{cases} x & (x<1) \\ 2x-1 & (1\leqslant x<10) \\ 3x-11 & (x\geqslant 10) \end{cases}$$

编一程序，输入x，输出y。

【解】

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

int main()
{
    double x, y;

    cout << "请输入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;
}
```

【8】编写一个计算薪水的程序。某企业有3种工资计算方法：计时工资、计件工资和固定月工资。程序首先让用户输入工资类别，再按照工资类别输入所需的信息。若为计时工资，则输入工作时间及每小时薪水。若为计件工资，则输入每件的报酬和本月完成的件数。若为固定月工资，输入工资额。计算本月应发工资。职工工资需要缴纳个人收入所得税，缴个税的方法是：2000元以下免税；2000～2500元者，超过2000元部分按5%收税；2500～4000者，2000～2500的500元按5%收税，超过2500部分按10%收税；4000元以上者，2000～2500的500元按5%收税，2500～4000的1500元按10%收税，超过4000元的部分按15%收税。最后，程序输出职工的应发工资和实发工资。

【解】

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

int main()
{
    char type;
    int time, piece;
    double salary, unitSalary;

    cout << "请选择计时工资（T)、计件工资（P)或固定工资（S)：";
    cin >> type;

   // 计算应发工资
    switch (type) {
        case 'T' :                                          //  计时工资
        case 't': cout << "请输入工作时间和小时工资：";
                  cin >> time >> unitSalary;
                  salary = time * unitSalary;
                 cout << "工作时间：" << time <<" 小时，本月应发工资为：" << salary << endl;
                  break;
       case 'P' :                                          // 计件工资
       case 'p': cout << "请输入完成数量和每件报酬：";
                  cin >> piece >> unitSalary;
                  salary = piece * unitSalary;
                  cout << "完成件数：" << piece <<"，本月应发工资为：" << salary << endl;
                  break;
     case 'S':                                            // 固定工资
     case 's':   cout << "请输入月工资：";
                cin >> salary;
                cout << "本月应发工资为：" << salary << endl;
                break;
     default:   cout << "错误的工资类型！" << endl; return 1;
  }

  // 计算实发工资
   double tmp = salary;
   if (tmp > 4000) {
      salary -= (tmp  -  4000) * 0.15;
      tmp = 4000;
   }
```

```
    if (tmp > 2500) {
       salary -= (tmp - 2500) * 0.1;
       tmp = 2500;
    }
    if (tmp > 2000) salary -= (tmp - 2000) * 0.05;

    cout << “本月实发工资为：” << salary;

    return 0;
  }
```

【9】编写一个程序，输入一个字母，判断该字母是元音还是辅音字母。用两种方法实现。第一种用if语句实现，第二种用switch语句实现。

【解】

```
//用if语句判断元音字母的程序
#include <iostream>
using namespace std;

int main()
{
   char ch;

   cout << "请输入一个字母：";
   cin >> ch;

   if (ch >= 'a' && ch <='z') ch = ch - 'a' + 'A';       // 全部换成大写字母
   if (ch >'Z' || ch < 'A') cout << "不是字母" << endl;
   else if (ch == 'A' || ch == 'E' || ch == 'I' || ch == 'O' || ch == 'U')
            cout << "是元音" << endl;
        else cout << "是辅音" << endl;

   return 0;
}

//用switch语句判断元音字母的程序
#include <iostream>
using namespace std;

int main()
{
   char ch;

   cout << "请输入一个字母：";
   cin >> ch;

   if (ch >= 'a' && ch <='z') ch = ch - 'a' + 'A';       // 全部换成大写字母
   if (ch >'Z' || ch < 'A') cout << "不是字母" << endl;
   else
       switch (ch) {
          case 'A' : case 'E' : case 'I': case 'O': case 'U':
               cout << "是元音" << endl; break;
          default: cout << "是辅音" << endl;
       }

   return 0;
}
```

【10】编写一个程序，输入三个非0正整数，判断这三个值是否能够成一个三角形。如果能

构成一个三角形，判断这三角形是否是直角三角形。

【解】

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

int main()
{
    int a, b, c, tmp;

    cout << "请输入三角形的三条边长：";
    cin >> a >> b >> c;

    // 将三条边长按递减次序排序
    if (a < b) { tmp = a; a = b; b = tmp; }
    if (a < c) { tmp = a; a = c; c = tmp; }

    // 判三角形
    if (b + c > a)      //是三角形
        if (a * a == b * b + c * c)
            cout << "是三角形且为直角三角形"  << endl;
        else cout << "是三角形" << endl;
    else cout << "不能构成三角形" << endl;

    return 0;
}
```

【11】凯撒密码是将每个字母循环后移3个位置后输出。如'a'变成'd','b'变成'e '，'z'变成了'c'。编一个程序，输入一个字母，输出加密后的密码。

【解】

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

int main()
{
    char ch;

    cout << "请输入一个字母：";
    cin >> ch;

    if (ch >= 'a' && ch <= 'z') ch = (ch - 'a' + 3) % 26 + 'a';
    else if (ch >= 'A' && ch <= 'Z') ch = (ch - 'A' + 3) % 26 + 'A';

    cout << ch << endl ;

    return 0;
}
```

【13】二维平面上的一个与x轴平行的矩形可以用两个点来表示。这两个点分别表示矩形的左下方和右上方的两个角。编一程序，输入两个点（x1，y1)、(x2，y2)，计算它对应的矩形的面积和周长，并判断该矩形是否是一个正方形。

【解】矩形的宽度为|x2 - x1|，高度为|y2 - y1|。矩形面积为|(x2 - x1) * (y2 - y1)|，周长为|2 * (x2 - x1 + y2 - y1)|。判断该矩形为正方形，可用|x2 - x1|等于|y2 - y1|。

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

```
int main()
{
    double x1, y1, x2, y2;
    const double EPSILON = 1e-10;

    cout << "请输入左下角的坐标（x1  y1）：";
    cin >> x1 >> y1;
    cout << "请输入右上角的坐标（x2  y2）：";
    cin >> x2 >> y2;

    cout << "矩形的面积为：" << fabs((y2 - y1) * (x2 - x1)) << endl;
    cout << "矩形的周长为：" << fabs(2 * (y2 - y1 + x2 - x1)) << endl;
    cout << (fabs(y2 - y1 - x2 + x1) < EPSILON ? "是正方形" : "不是正方形" ) << endl;

    return 0;
}
```

【14】设计一停车场的收费系统。停车场有三类汽车，分别用三个字母表示。C代表轿车，B代表客车，T代表卡车。收费标准如下

车辆类型	收费标准
轿车	三小时内，每小时10元。三小时后，每小时15元
客车	两小时内，每小时20元。两小时后，每小时35元
卡车	一小时内，20元，一小时后，每小时30元

编一程序，输入汽车类型和入库、出库的时间，输出应交的停车费。

【解】如果所有的车都是当天进场当天出场，停车时间就是出库时间减去入库时间。假设每辆车的停车时间都不会超过24小时，那么当出库时间小于入库时间时，说明这辆车停过夜了，停车时间应该是：出库时间 + 24小时 - 入库时间。假如停车时间用分钟计算，这两种情况可以用一个公式计算：(outHour * 60 + outMinute - inHour * 60 - outMinute + 1440) % 1440。

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

int main()
{
    const int Fee3HourCar = 10, FeeForCar = 15;
    const int Fee2HourBus = 20, FeeForBus = 35;
    const int Fee1HourTruck = 20, FeeForTruck = 30;
    const int minAllDay = 1440;
    char type;
    int inHour, inMinute, outHour, outMinute, minute, hour, fee;

   // 输入车型
    cout << "请输入汽车类型（轿车C、客车B、卡车T）：";
    cin >> type;
    if (type != 'c' && type !='C' && type != 'B' && type != 'b' && type !='t' && type != 'T')
       { cout << "汽车类型错误！" << endl;  return 0; }

    // 输入入库和出库时间
    cout << "请输入入库时间（时 分）：";
    cin >> inHour >> inMinute;
    if (inHour < 0 || inHour > 23 || inMinute < 0 || inMinute > 59)
        { cout << "入库时间错误!" << endl; return 0; }

    cout << "请输入出库时间（时 分）：";
    cin >> outHour >> outMinute;
    if (outHour < 0 || outHour > 23 || outMinute < 0 || outMinute > 59)
```

```
        { cout << "出库时间错误!" << endl; return 0; }

    // 计算停车时间
      minute = (outHour * 60 + outMinute - inHour * 60 - outMinute + minAllDay) % minAllDay;
      hour = minute / 60;
     if (minute % 60 != 0) ++hour;        // 停车不足1小时按1小时计算

   // 根据不同的车型计算停车费
     switch (type) {
       case 'c':case 'C':                                  // 轿车
         if (hour <= 3) fee = Fee3HourCar * hour;
         else fee = 3 * Fee3HourCar + FeeForCar * (hour - 3);
         break;
       case 'b': case 'B':                                 // 客车
         if (hour <= 2) fee = Fee2HourBus * hour;
         else fee = 2 * Fee2HourBus + FeeForBus * (hour - 2);
         break;
       case 't': case 'T':                                 // 卡车
         if (hour <= 1) fee = Fee1HourTruck;
         else fee = Fee1HourTruck + FeeForTruck * (hour - 1);
     }

     cout << "停车费为：" << fee << endl;

     return 0;
}
```

第4章 简答题

【1】假设在while语句的循环体中有这样一条语句：当它执行时while循环的条件值就变为false。那么这个循环是将立即中止还是要完成当前的循环周期呢？

【解】循环体继续执行。在完成了当前的循环周期后再次检查循环条件，此时循环将终止。

【2】当遇到下列情况时，你将怎样编写for语句的控制行。

a. 从1计数到100。

b. 从2, 4, 6, 8, …计数到100。

c. 从0开始，每次计数加7，直到成为三位数。

d. 从100开始，反向计数，99, 98, 97, …直到0。

e. 从'a'变到'z'。

【解】

```
a. for (k = 1; k <=100; ++k)
b. for (k = 2; k <= 100; k += 2)
c. for (k = 0; k < 100; k += 7)
d. for (k = 100; k >= 0; --k)
e. for (ch = 'a'; ch <= 'z'; ++ch)。
```

【3】为什么在for循环中最好避免使用浮点型变量作为循环变量？

【解】因为循环变量通常用来记录循环执行次数，用整型数表示更加合适。

【4】在程序

```
for (i = 0; i < n; ++i)
    for (j = 0; j < i; ++j) cout << i << j;
```

中，cout << i << j; 执行了多少次？

【解】执行次数是0 + 1 + 2 +……+ n-2，即（n-2）*(n-3)/2。

【10】下面这个循环是死循环吗？

```
for  ( k = -1; k < 0; --k);
```

【解】整型数在计算机内部是用补码表示的。如果整型数在计算机内占用2个字节，那么-1在机器内的表示是16个1。当执行了32767个循环周期后，i值的最高位为1，而其他位都为0，即-32768。这时再执行一次减法，最高位变成了0，而其他位全为1。由于最高位为0，C++把它看成正数32767，循环条件不满足，循环被终止。

第4章 程序设计题

【2】编写这样一个程序：先读入一个正整数*N*，然后计算并显示前*N*个奇数的和。例如，如果*N*为4，这个程序应显示16，它是1＋3＋5＋7的和。

【解】

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

int main()
{
    int n, sum = 0;

    cout << "请输入一个整数：";
    cin >> n;

    for (int i = 0; i < n; ++i)
        sum += 2 * i + 1;

    cout << "前" << n << "个奇数和为：" << sum << endl;

    return 0;
}
```

【4】写一个程序，提示用户输入一个正整数，然后输出这个整型数的每一位数字，数字之间插一个空格。例如当输入是12345时，输出为：1 2 3 4 5。

【解】首先获取整数的位数，然后依次取出一个数的个位、十位、……。

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

int main()
{
    int n,  i;

    cout << "请输入一个整数：";
    cin >> n;

    for (i = 10; n >= i; i *= 10);      //计算n的位数，i是n的位数的10倍
    do {                                // 依次取每一位的值
        i /= 10;                        // 获得最高位的数量级
        cout << n / i << ' ';           // 输出最高位
        n %= i;                         // 去掉最高位
    } while (i > 1);
    cout << endl;

    return 0;
}
```

【5】在数学中，有一个非常著名的斐波那契数列，它是按13世纪意大利著名数学家Leonardo Fibonacci的名字命名的。这个数列的前两个数是0和1，之后每一个数是它前两个数的和。因此斐波那契数列的前几个数为：

$F_0=0$

$F_1=1$

F_2=1　　(0 + 1)
F_3=2　　(1 + 1)
F_4=3　　(1 + 2)
F_5=5　　(2 + 3)
F_6=8　　(3 + 5)

编写一个程序，顺序显示F_0到F_{15}。

【解】在斐波那契数列中，F_0和F_1是定值，F_2到F_{15}的生成方式是一样的，都是它前面的两个数的和。因此F_0和F_1可以直接赋值，F_2到F_{15}用重复n次的循环，从2循环到15，每个循环周期计算并输出一个斐波那契数。程序用到3个变量：前两个斐波那契数f0和f1，当前正在生成的斐波那契数f2。

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

int main()
{
    int f0 = 0, f1 = 1, f2;

    cout << f0 << ' ' << f1 << ' ';
    for ( int i = 2; i <= 15; ++i) {
        f2 = f0 + f1;
        cout << f2 << ' ';
        f0 = f1;
        f1 = f2;
    }
    cout << endl;

    return 0;
}
```

【6】编写一个程序，要求输入一个整型数N，然后显示一个由N行组成的三角形。在这个三角形中，第一行有一个“*”，以后每行比上一行多两个“*”，三角形像下图这样尖角朝上。

```
          *
         ***
        *****
       *******
      *********
     ***********
    *************
   ***************
```

【解】用重复n次的循环，每个循环周期打印一行。每一行有两部分组成：前面的空格部分和后面的*部分。第i行有n-i个空格和2*i-1个*。打印第i行可以先用一个for循环打印n-i个空格，再用一个for循环打印2*i-1个*。

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

int main()
{
    int n, i, j;

    cout << "请输入三角形的行数：";
    cin >> n;

    for ( i = 0; i < n; ++i) {           // 打印每一行
        cout << endl;
        for (j = 0; j < n - i - 1; ++j) cout << ' ';   // 打印前面的空格
        for (j = 0; j < 2 * i + 1; ++j)  cout << '*';   // 打印一行*
```

```
    }
    cout << endl;

    return 0;
}
```

【8】编写一个程序求$\sum_{i=1}^{10} n!$，要求只做*10*次乘法和10次加法。

【解】这是一个重复10次的循环。在第i个循环周期中，先计算i！的值，再将i！的值加入总和。计算i!时，在前一个循环周期已经计算过(i-1)!。所以计算i!时，只需要执行(i-1)! * i。这样就可以用10次乘法和10次加法完成$\sum_{i=1}^{10} n!$的计算。

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

int main()
{
   int sum = 0, i, fact = 1;           // sum是总和，fact是i！，初值是0！

   for (i = 1; i <= 10; ++i) {
       fact = fact * i;
       sum += fact;
    }

    cout << "总和为： " << sum << endl;

   return 0;
}
```

【9】设计一程序，求1 - 2 + 3 - 4 + 5 - 6 + …… +/- N的值。

【解】方法一：用一个重复n次的循环。

方法二：这个公式的第1、2项的和是-1，第3、4项的和也是-1。推而广之，如果i是奇数，则第i项和i+1项的和是-1。根据这个观察结果可以知道，如果N是偶数，正好可以凑成N/2个-1，所以结果是- N/2。如果N是奇数，则前N-1项可以凑成（N-1）/2个-1，所以结果是-（N - 1）/ 2 + N 。这个方法的时间性能要优于采用重复n次循环。

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

int main()
{
   int sum , n ;

   cout << "请输入n：";
   cin >> n;

   if (n % 2) sum = n - (n-1)/ 2;
   else sum = -n / 2;

    cout << "总和为： " << sum << endl;

    return 0;
}
```

【12】编写一个程序，输入一个句子（以句号结束），统计该句子中的元音字母数、辅音字母数、空格数、数字数及其他字符数。

【解】依次读入句子中的每个字符，如果读到句号，统计结束，输出统计结果，否则按照不

同的字符进行不同的处理。

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

int main()
{
   char ch;
   int numVowel = 0, numCons = 0, numSpace = 0, numDigit = 0, numOther = 0;

    cout << "请输入句子：";
    while (true) {                        //  处理每个字符
         cin.get(ch);                        // 读入一个字符
         if (ch == '.') break;               // 如果读入的是句号，退出循环
         if (ch >= 'A' && ch <= 'Z' ) ch = ch - 'A' + 'a';
         if (ch >= 'a' && ch <= 'z')
              if (ch == 'a' || ch == 'e' || ch == 'i' || ch == 'o' || ch == 'u') ++numVowel;
              else ++numCons;
         else if (ch == ' ') ++numSpace;
               else if (ch >= '0' && ch <= '9') ++numDigit;
                    else ++numOther;
    }

    cout << "元音字母数：" << numVowel << endl;
    cout << "辅音字母数：" << numCons << endl;
    cout << "空格数：" << numSpace << endl;
    cout << "数字字符数：" << numDigit << endl;
    cout << "其他字符数：" << numOther << endl;

   return 0;
}
```

【13】猜数字游戏：程序首先随机生成一个1到100的整数，然后由玩家不断输入数字来猜这个数字的大小。猜错了，计算机会给出一个提示，然后让玩家继续猜。猜对了就退出程序。例如：随机生成的数是42，开始提示的范围是1~100，然后玩家猜是30，猜测是错误的，计算机告诉你太小了。然后，玩家继续输入60，猜测依然错误，计算机告诉你太大了。直到玩家猜到是42为止。用户最大的猜测次数是10次。

【解】生成一个1到100的整数可以用C++的随机数生成器rand。程序的主体是用户猜测部分。用户最多可以猜10次，这可以用一个重复10次的循环来控制。每次猜测有两个阶段：先输入用户的猜测，然后判别猜测正确与否。如果猜测正确，则退出循环，显示“猜对了”，程序终止。如果猜测错误，则给出相应的信息，继续循环。如遇到for循环的终止条件，表示用户已猜了10次，输出失败信息，程序终止。

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

int main()
{
    int num, guess;

    srand(time(NULL));
    num = rand() % 100 + 1;             // 生成1 - 100之间的整数
    for (int i = 0; i < 10; ++i) {
        cout << "请输入你的猜测：";
        cin >> guess;
        if (guess == num) break;          // 猜对了
```

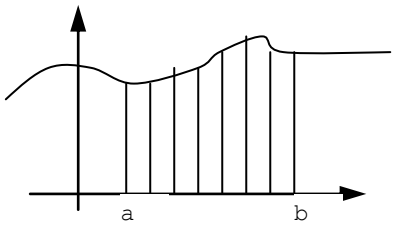
```
        if (guess < num) cout << "太小了！" << endl; else cout << "太大了！" << endl;// 猜错
    }
    if (guess == num) cout << "猜对了！" << endl; else cout << "你失败了！" << endl;

    return 0;
}
```

【14】设计一个程序，用如下方法计算x的平方根：首先猜测x的平方根root是x/2，然后检查root * root与x的差，如果差很大，则修正$root = \frac{root + \frac{x}{root}}{2}$，再检查root * root与x的差，重复这个过程直到满足用户指定的精度。

【解】这个算法只需要一个while循环就可以完成。循环条件是判断有没有达到精度，循环体修正root的值。

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

int main()
{
    double x, root, epsilon;

    cout << "请输入x：";
    cin >> x;
    cout << "请输入精度：";
    cin >> epsilon;

    root = x / 2;
    while (fabs(x - root * root) > epsilon)
        root = (root + x / root ) / 2;

    cout << x << "的平方根是：" << root << endl;

  return 0;
}
```

【15】定积分的物理意义是某个函数与x轴围成的区域的面积。定积分可以通过将这块面积分解成一连串的小矩形，计算各小矩形的面积的和而得到，如图4-1所示。小矩形的宽度可由用户指定，高度就是对应于这个x的函数值f(x)。编写一个程序计算函数$f(x)=x^2+5x+1$在区间[a, b]间的定积分。a、b及小矩形的宽度在程序执行时由用户输入。



【解】如果小矩形的宽度是delt，起点坐标是x，则矩形的近似面积为delt*f(x+delt/2)。计算定积分只需要沿着x轴将一个个小矩形的面积相加即可。

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

int main()
{
    double x, a, b, s = 0, h, delt;                 // s为积分值
```

```
    cout << "请输入积分区域（a  b）： ";
    cin >> a >> b;
    cout << "请输入小矩形的宽度： ";
    cin >> delt;

    for ( x = a + delt / 2; x <= b; x += delt) {
        h = x * x + 5 * x + 1;
        s += h * delt;
    }


    cout << "函数的积分是：" << s << endl;

    return 0;
}
```

【16】用第15题的方法求π 的近似值。具体思想如下：在平面坐标系中有一个圆心在原点，半径为1的圆，用矩形法计算第一象限的面积S，4 * S就是整个圆的面积。圆面积也可以通过π r^2来求，因此可得π = 4 * S。尝试不同的小矩形宽度，以得到不同精度的π 值。

【解】在第一象限中，x的值从0变到1。圆上的每个点的坐标为（x，$\sqrt{1-x^2}$ ）。如果矩形的宽度为delt，计算第一象限中的圆面积就是沿着x轴，从delt/2到1计算每个小矩形面积并相加。

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

int main()
{
    double x, s = 0, delt;

    cout << "请输入小矩形的宽度： ";
    cin >> delt;

    for ( x = delt / 2; x <= 1; x += delt)          // 计算第一象限内的圆面积
        s += sqrt(1 - x * x) * delt;

    cout << "π 的近似值是：" << 4 * s << endl;

    return 0;
}
```

【17】 编一程序，计算方程 $x^3+2x^2+5x-1=0$ 在区间[-1, 1]之间的根。

【解】这是一个do……while的实例。循环控制行判断是否达到指定精度。循环体计算根的近似值，修正区间。

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

int main()
{
    double x, x1 = -1, x2 = 1, f2, f1, f, epsilon;

    cout << "请输入精度： ";
    cin >> epsilon;

    do {f1 = x1 * x1 * x1 + 2 * x1 * x1 + 5 * x1 - 1;      // 计算f(x1)
        f2 = x2 * x2 * x2 + 2 * x2 * x2 + 5 * x2 - 1;      // 计算f(x2)
```

```
        x = (x1 * f2 - x2 * f1) / (f2 - f1); // 计算(x1,f(x1))和(x2,f(x2))的弦交与x轴的交点
        f = x * x * x + 2 * x * x + 5 * x - 1;
        if (f * f1 > 0) x1 = x;  else x2 = x;              // 修正区间
    } while (fabs(f) > epsilon);

    cout << "方程的根是: " << x << endl;

    return 0;
}
```

第5章　简答题

【1】数组的两个特有性质是什么?

【解】第一个特性是所有数组元素的类型是相同的，第二个特性是数组元素之间是有序的。

【2】写出以下的数组变量的定义。

a. 一个含有100个浮点型数据的名为realArray的数组。

b. 一个含有16个布尔型数据的名为inUse的数组。

c. 一个含有1000个字符串，每个字符串的最大长度为20的名为lines的数组。

【解】

```
a.  double realArray[100];
b.  bool inUse[16];
c.  char lines[1000][21];
```

【3】用for循环实现下述整型数组的初始化操作。

squares

0	1	4	9	16	25	36	49	64	81	100
0	1	2	3	4	5	6	7	8	9	10

【解】在数组squares中，每个元素的值正好是对应下标的平方，因此可用语句

```
for (int i = 0; i <= 10; ++i) squares[i] = i * i;
```

【4】. 用 for 循环实现下述字符型数组的初始化操作。

array

a	b	c	d	……	v	w	x	y	z
0	1	2	3		21	22	23	24	25

【解】可用语句

```
for (int i = 0; i <= 10; ++i) array[i] = 'a' + i;
```

【5】什么是数组的配置长度和有效长度?

【解】有时在编程序时无法确定所要处理的数据量，因此无法确定数组的大小。这时可以按可能的最大的数据量定义数组。定义数组时给定的数组规模称为配置长度。在执行时真正存入数组中的元素个数称为有效长度。

【6】什么是多维数组?

【解】数组元素本身又是一个数组的数组称为多维数组。

【7】. 要使整型数组 a[10] 的第一个元素值为 1 ，第二个元素值为 2 ，……，最后一个元素值为10 ，某人写了下面语句，请指出错误。

```
for (i = 1; i <= 10; ++i) a[i] = i;
```

【解】C++数组的下标是从0开始而不是1开始。

【9】写出定义一个整型二维数组并赋如下初值的语句

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 4 \end{bmatrix}$$

【解】int a[4][4] = { {1}, {0,2}, {0,0,3}, {0,0,0,4}};

或 int a[4][4] = {0};

```
for (int i = 0; i < 4; ++i) a[i][i] = i+1;
```

【10】定义了一个 26×26 的字符数组，写出为它赋如下值的语句

a b c d e f … x y z
b c d e f g … y z a
… … … …
y z a b c d … v w x
z a b c d e … w x y

【解】

```
for (i = 0; i<26; ++i)
     for ( j = 0; j < 26; ++j)
          a[i][j] = (i+j) % 26 + 'a';
```

第5章 程序设计题

【9】编写一个程序，从键盘上输入一篇英文文章。文章的实际长度随输入变化，最长有10行，每行最多80个字符。要求分别统计出其中的字母、数字、空格和其他字符的个数。（提示：用一个二维字符数组存储文章。）

【解】用二维字符数组存储输入的文章。

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

int main()
{
    char article[10][80];
    int i, j, alph = 0, digit = 0, space = 0, other = 0;

    cout << "请输入十行文字，每行以回车结束,满80个字符自动换行： " << endl;
    for ( i = 0; i < 10; ++i)  {
        cout << "请输入第" << i << "行：";
        for (j = 0; j < 80; ++j) {
            cin.get(article[i][j]);                          // 输入一个字符
            if (article[i][j] == '\n') {                     // 如果是换行符，当前行结束
                article[i][j] = '\0';
                break;
            }
        }
    }

    for ( i = 0; i < 10; ++i)                                // 统计过程
        for (j = 0; j < 80; ++j) {
            if (article[i][j] == '\0') break;                // 当前行结束
            if (article[i][j] == ' ') ++space;
            else if (article[i][j] >= '0' && article[i][j] <= '9') ++digit;
                else if (article[i][j] <= 'z' && article[i][j] >= 'a' || article[i][j] <= 'Z' && article[i][j] >= 'A')
                        ++alph;
                    else ++other;
        }

    cout << "一共有 " << alph << "个字母，" << digit << "个数字, " ;
    cout << space << "个空格，" << other << "个其他字符。" << endl;

    return 0;
}
```

【10】在公元前3世纪，古希腊天文学家埃拉托色尼发现了一种找出不大于*n*的所有自然数中

的素数的算法，即埃拉托色尼筛选法。这种算法首先需要按顺序写出2～*n*中所有的数。以*n*＝20为例：

2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

然后把第一个元素画圈，表示它是素数，然后依次对后续元素进行如下操作：如果后面的元素是画圈元素的倍数，就画×，表示该数不是素数。在执行完第一步后，会得到素数2，而所有是2的倍数的数将全被画掉，因为它们肯定不是素数。接下来，只需要重复上述操作，把第一个既没有被圈又没有画×的元素圈起来，然后把后续的是它的倍数的数全部画×。本例中这次操作将得到素数3，而所有是3的倍数的数都被去掉。依次类推，最后数组中所有的元素不是画圈就是画×。所有被圈起来的元素均是素数，而所有画×的元素均是合数。编写一个程序实现埃拉托色尼筛选法，筛选范围是2～1000。

【解】埃拉托色尼筛选法首先将所有的数都认为是素数，然后排除一个个合数，因此需要一个变量保存每个数是否为素数。这可以用一个2001个元素的bool类型的数组prime来保存。如果i是素数，prime[i]为true，否则为false。元素0和1没有使用。初始时，将所有元素值都设为true。然后下标从2开始，将下标为2的倍数的元素都设为false，将下标为3的倍数的元素设为false，将下标为5的倍数的元素设为false，……。最后检查数组prime，对应元素值为true的下标值都是素数。

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

int main()
{
	int i, j;
	bool prime[2001];

	for (i = 2; i < 2001; ++ i) prime[i] = true;                    // 将所有数都设为素数

	for (i = 2; i < 1001; ++i)                                      // 将i的倍数设为非素数
		if (prime[i])
			for (j = 2; i * j < 2001; ++j) prime[i*j] = false;

	cout << "2 - 2000之内的素数有" << endl;
	for (i = 2; i < 2001; ++i)                                      // 输出所有素数
		if (prime[i]) cout << i << '\t';
	cout << endl;

	return 0;
}
```

【11】设计一个井字游戏，两个玩家，一个打圈(O)，一个打叉(X)，轮流在3乘3的格上打自己的符号，最先以横、直、斜连成一线则为胜。如果双方都下得正确无误，将得和局。

【解】用一个3*3的二维整型数组table来保存棋盘。圈和叉用两个数字来区分，如1和-1。如果第1行的第2列是个圈，则table[0][1]=1。如果是个叉，则table[0][1]=-1。表示这个位置是空的可以用除了1和-1以外的任何数字，最简单的是用0。

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

int main()
{
    int table[3][3] = { 0 };                                    // 将棋盘初始化为0
    int i, j, k, judgeRow, judgeCol, judgeDig1, judgeDig2, player = 1, row, col;

    for (k = 0; k < 9; ++k) {
        do {                                                    // 输入player的选择
```

```
            for (i = 0; i < 3; ++i) {                    // 显示当前的状态
                for (j = 0; j < 3; ++j)
                    switch( table[i][j] ) {
                        case 1: cout << "[O]"; break;
                        case -1: cout << "[×]"; break;
                        default: cout << "[  ]";
                    }
                cout << endl;
            }
            cout << "请输入player" << player << "的选择：";
            cin >> row >> col;
            if (row < 1 || row > 3 || col < 1 || col > 3 || table[row - 1][col - 1])
                cout << "输入有误，请重新输入" << endl;
            else break;
        } while (true);
        table[row - 1][col - 1] = (player == 1 ? 1 : -1);    // 根据选择修改棋盘的状态
        for (i = 0; i < 3; ++i) {                            // 判断是否有玩家赢了
            judgeRow = table[i][0] + table[i][1] + table[i][2];
            judgeCol = table[0][i] + table[1][i] + table[2][i];
            if (judgeRow == 3 || judgeCol ==3 ){
                cout << "玩家1赢了， 游戏结束" << endl;
                return 0;
            }
            else if (judgeRow == -3 || judgeCol == -3) {
                cout << "玩家2赢了， 游戏结束" << endl;
                return 0;
            }
        }
        judgeDig1 = table[0][0] + table[1][1] + table[2][2];
        judgeDig2 = table[0][2] + table[1][1] + table[2][0];
        if (judgeDig1 == 3 || judgeDig2 ==3) {
            cout << "玩家1赢了， 游戏结束" << endl;
            return 0;
        }
        else if (judgeDig1 == -3 || judgeDig2 == -3 ){
            cout << "玩家2赢了， 游戏结束" << endl;
            return 0;
        }

        player = player % 2 + 1;                    // 交换玩家
    }

    cout << "平局！" << endl;
    return 0;
}
```

【12】国际标准书号ISBN用来唯一标识一本合法出版的图书。它由十位数字组成。这十位数字分成4个部分。例如，0-07-881809-5。其中，第一部分是国家编号，第二部分是出版商编号，第三部分是图书编号，第四部分是校验数字。一个合法的ISBN号，10位数字的加权和正好能被11整除，每位数字的权值是它对应的位数。对于0-07-881809-5，校验结果为（0×10+0×9+7×8+8×7+8×6+1×5+8×4+0×3+9×2+5×1） % 11 = 0。所以这个ISBN号是合法的。为了扩大ISBN系统的容量，人们又将十位的ISBN号扩展成13位数。13位的ISBN分为5部分，即在10位数前加上3位ENA(欧洲商品编号)图书产品代码“978”。 例如，978-7-115-18309-5。13位的校验方法也是计算加权和，检查校验和是否能被10整除。但所加的权不是对应的位数而是根据一个系数表：1313131313131。对于978-7-115-18309-5，校验的结果是：

（9×1+7×3+8×1+7×3+1×1+1×3+5×1+1×3+8×1+3×3+0×1+9×3+5×1）% 10 = 0。编写一个程序，检验输入的ISBN号是否合法。输入的ISBN号可以是10位，也可以是13位。

【解】程序定义一个14个元素的字符数组来保存一个ISBN号。根据输入的ISBN号的长度区分两种形式的ISBN号，采用不同的方法进行校验。如果长度不为10或13，则肯定是非法的ISBN号。ISBN号的每一位都是数字。只要有1位是非数字，则为非法的ISBN号。10位的ISBN号的校验首先计算sum = $\sum_{i=1}^{10} i \times (isbn[10-i]-'0')$，然后检验sum能否被11整除。13位的ISBN号的检验稍微麻烦一点，每一位加的权不是它的位置，而是1或3。为此程序定义了一个数组check，记录每一位的权值。这样计算13位的ISBN号的校验和可以用一个重复13次的for循环计算sum = $\sum_{i=1}^{13} check[i] \times (isbn[10-i]-'0')$，然后检验sum能否被10整除。

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

int main()
{
    char isbn[14];
    int i, sum = 0. check[13] = {1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1};

    cout << "请输入ISBN号：";
    cin >> isbn;

    if (strlen(isbn) != 10 && strlen(isbn) != 13) {          // 检查长度，排除非法的ISBN号
            cout << "非法的ISBN" << endl; return 0;
    }

    if (strlen(isbn) == 10) {                                    // 10位的ISBN号的检验
        for (i = 10; i > 0; --i) {
            if (isbn[10 - i] < '0' || isbn[10 - i] > '9') {     // 检查ISBN号中是否包含非数字
                    cout << "非法的ISBN" << endl; return 0;
            }
            sum += i * (isbn[10 - i] - '0');
        }
        if (sum % 11 == 0) cout << "合法的ISBN号" << endl;
        else cout << "非法的ISBN" << endl;
    }
    else {                                                       // 13位的ISBN号的检验
         for (i = 0; i < 13; ++i) {
            if (isbn[i] < '0' || isbn[i] > '9') {                // 检查ISBN号中是否包含非数字
                cout << "非法的ISBN" << endl; return 0;
            }
            sum += check[i] * (isbn[i] - '0');
         }
         if (sum % 10 == 0) cout << "合法的ISBN号" << endl;
         else cout << "非法的ISBN" << endl;
    }

    return 0;
}
```

第6章 简答题

【1】说明函数原型声明和函数定义的区别。

【解】函数原型声明只是说明了该函数应该如何使用，函数调用时应该给它传递哪些数据，

函数调用的结果又应该如何使用。函数定义除了给出函数的使用信息外，还需要给出了函数如何实现预期功能，即如何从输入得到输出的完整过程。

【2】什么是形式参数？什么实际参数？

【解】函数的参数一般可以看成是函数运行时的输入。形式参数指出函数调用时应该给它传递几个数据，这些数据是什么类型的。实际参数是函数某次调用时的真正的输入数据，是形式参数的初值。

【3】传递一个数组为什么需要两个参数？

【解】因为数组传递本质上只是传递了数组的起始地址，数组中的元素个数需要另一个变量来指出。

【4】对于如下的函数声明 `char f(int a, int b = 80, char c = '0');` 下面的调用哪些是合法的，哪些是不合法的？

```
f( )          f(10, 20)      f(10, '*')
```

【解】f( )不合法的。f(10, 20)合法。f(10, '*')合法的。

【5】. 什么是值传递？

【解】在值传递中，形式参数有自己的存储空间，实际参数是形式参数的初值。参数传递完成后，形式参数和实际参数再无任何关联。

【6】什么是函数模板？什么是模板函数？函数模板有什么用途？

【解】函数模板就是函数中的某个参数或返回值的类型是不确定的，是可变的，这些不确定的类型称为模板参数。如果给函数模板的模板参数指定了一个具体的类型，就得到了一个可以执行的函数，这个函数称为模板函数。函数模板可以节省程序员的工作量，若干个被处理的数据类型不同，但处理流程完全一样的函数可以写成一个函数模板。

【7】C++是如何实现函数重载的？

【解】编译器为每个函数重新取一个内部名字。当遇到调用重载函数时，编译器分析实际参数的个数和类型，确定一个形式参数表与实际参数表一致的重载函数，将这个重载函数的内部名字替代函数调用时的函数名。

【8】全局变量和局部变量的主要区别是什么？使用全局变量有什么好处？有什么坏处？

【解】局部变量是函数内部或语句块中定义的变量，当函数调用时局部变量被生成，函数执行结束时局部变量被消亡。全局变量是在所有函数外定义的变量它可以一直生存到整个程序执行结束。所有定义在该变量后面的函数都能使用该全局变量。使用全局变量可以加强函数之间的联系。函数之间的信息交互不用通过参数传递。但全局变量也破坏的函数的独立性。用同样参数对同一个函数的多次调用可能因为执行时全局变量值不一样而导致函数执行的结果不同。

【9】变量定义和变量声明有什么区别？

【解】变量定义和变量声明的最主要的区别在于有没有分配空间。变量定义要为所定义的变量分配空间，而变量声明仅指出本源文件中的程序用到了某个变量，该变量的类型是什么。至于该变量的定义可能出现在其他源文件中，也可能出现在本源文件中尚未编译到的部分。

【1】. 为什么不同的函数中可以有同名的局部变量？为什么这些同名的变量不会产生二义性？

【解】局部变量的生命周期是在对应的函数的执行期间。函数调用时，系统会分配一块内存空间（称为一个帧），所有的局部变量都存储在这块空间中。当函数执行结束时，系统回收这块空间，这些变量就消失了。每个函数有自己的存储局部变量的空间，每个函数只能访问自己的帧及全局变量。所以不同的函数可以有同样的局部变量名，而不会有二义性。

【11】静态的局部变量和普通的局部变量有什么不同？

【解】普通的局部变量随函数的执行而生成，函数执行的结束而消亡。静态的局部变量在该函数第一次执行时生成，到整个程序执行结束时消亡。当再次执行该函数时，其他的局部变量又重新被生成了，而静态局部变量不重新生成。当函数用到静态的局部变量时，还是到第

一次为该静态局部变量分配的空间中进行操作。这样，函数上一次执行时的某些信息可以被用在下一次函数执行中。

【12】如何让一个全局变量或全局函数成为某一源文件独享的全局变量或函数？

【解】将该全局变量或全局函数设为静态的。

【13】如何引用同一个project中的另一个源文件中的全局变量？

【解】用外部变量声明。如果源文件A中定义了一个全局变量x，源文件B也要用这个x，那么在源文件B中可以写一个外部变量声明：extern x;

【14】在汉诺塔问题中，如果初始时第1根柱子上有64个盘子，将这64个盘子移到第3根柱子需要移动多少次盘子。假如计算机每秒钟可执行1000万次盘子的移动，完成64个盘子的搬移需要多少时间？如果人每秒钟可以搬移一个盘子，那么完成64个盘子的搬移需要多少年？

【解】根据递归的思想，可以将64个盘子的汉诺塔问题转换为63个盘子的汉诺塔问题。可先将上面的63个盘子从第一根柱子移到第二根柱子，再将最后一个盘子直接移到第三根柱子，最后再将63个盘子从第二根柱子移到第三根柱子。依次类推，63个盘子的问题可以转化为62个盘子的问题，62个盘子的问题可以转化为61个盘子的问题，直到1个盘子的问题。如果只有一个盘子，就可将它直接从第一根柱子移到第三根柱子，这就是递归终止的条件。根据上述思路，可得汉诺塔问题的递归程序，

```
void Hanoi(int n, char start, char finish, char temp)
{   if (n==1)  cout << start << "->" << finish << '\t';
    else {
        Hanoi(n-1, start, temp, finish);
        cout << start << "->" << finish << '\t';
        Hanoi(n-1, temp, finish, start);
    }
}
```

设移动n个盘子需要时间h(n)，从程序中可以看出移动n个盘子需要2次移动n-1个盘子，再执行一次盘子的移动。移动一次盘子需要O(1)的时间，移动n-1个盘子需要h(n-1)的时间，所以

$$h(n)=2h(n-1)+1=2(2h(n-2)+1)+1=2^2h(n-2)+2+1$$
$$=2^3h(n-3)+2^2+2+1=2^nh(0)+2^{n-1}+2^{n-2}+\cdots+2^0$$
$$=2^{n-1}+2^{n-2}+\cdots+2^0=2^n-1$$

所以，h(64) = $2^{64}-1$

如果计算机每秒钟可执行1000万次盘子的移动，完成64个盘子的搬移需要

$$\frac{2^{64}-1}{10000000}\approx\frac{2^{64}}{10\times2^{20}}=0.1\times2^{44}$$

如果人每秒钟可以搬移一个盘子，那么完成64个盘子的搬移需要

$$\frac{2^{64}-1}{365\times24\times3600}\approx3.29934\times10^{11}$$ 年

第6章 程序设计题

【1】设计一个函数，判别一个整数是否为素数。

【解】该函数有一个整型的参数，函数的返回值是bool类型的值。

方法一：直接从素数的定义出发。如果一个整数n正好有两个因子：1和n，那么n是素数。

```
bool IsPrime(int n)
{
    int divisors = 0;                  // 因子个数
    for (int i =1; i<=n; ++i)
      if ( n % i == 0) ++divisors;
    return (divisors == 2);
```

```
}
```

方法二：IsPrime没有必要检查所有的因子。只要发现任何一个大于1小于n的因子，就能停下来报告n不是素数。

如果n能被2整除，直接报告n不是素数。如果n不能被2整除，那么它也不可能被4或6或其他偶数整除。因此，IsPrime只需要检查2和奇数。但注意有个特例，2能被2整除，但2是素数。如果m不是素数，则必有一个因子小于$\sqrt{n}$。因此不需要检查到n为止。只需检查到$\sqrt{n}$ 。

```
bool IsPrime(int n)
{
  int limit;
  if (n <= 1) return false;             // 小于等于1的整数不可能是素数
  if (n == 2) return true;              // 2 是素数
  if (n % 2 == 0) return false;          // 能被2整除的其他整数都不是素数
  limit = sqrt(n) + 1;
  for ( int i = 3 ; i <= limit ; i += 2 ) //最坏情况下是根号n
     if (n % i == 0) return false;
  return true ;
}
```

【2】设计一个函数，使用以下无穷级数计算sin*x*的值。$\sin x = \frac{x}{1!} - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots$。舍去的绝对值应小于$\varepsilon$，$\varepsilon$的值由用户指定。如果用户不指定$\varepsilon$的值，则假设为$10^{-6}$。

【解】函数的有一个double型的参数x以及指定的精度，返回值也是double类型。函数体是一个循环。每次循环周期将数列的一项加入总和。数列的通项是$(-1)^i \frac{x^{2i-1}}{(2i-1)!}$。如果第i-1项的值是item，则第i项的值为$-item \times \frac{x \times x}{(2 \times i - 1)(2 \times i - 2)}$。

```
double Sin(double x, double epsilon = 1e-6)
{
    double sin = 0, item = x;

    for (int i = 2; fabs(item) > epsilon; ++i) {
        sin += item;
        item = -item * x * x / (2 * i - 1) / (2 * i - 2);
    }

    return sin;
}
```

【3】写一个函数stringCopy将一个字符串复制到另一个字符串。

【解】由于字符串变量是用一个字符数组保存，而数组是不能直接赋值。数组间的赋值是数组元素之间互相赋值，所以字符串变量的赋值也是对应数组元素之间的互相赋值。函数的参数是两个字符串，即两个字符类型的数组。由于字符串都有结束符号'\0'，所以传递字符串只需要传递一个字符数组的起始地址，即数组名。数组的赋值是用一个循环。循环终止条件是检查是否已经复制到元素'\0'。

```
void stringCopy(char des[], char src[])
{
    for (int i = 0; src[i] != '\0' ; ++i) des[i] = src[i];
    des[i] = '\0';
}
```

【4】设计一个支持整型、实型和字符型的气泡排序的函数模板。

【解】

```
template <class T>
void bubbleSort( T data[], int size)
```

```
{
    int i, j;
    bool flag;
    T tmp;

    for (i = 1; i < size; ++i) {        // 完成size-1次起泡
        flag = false;
        for (j = 0; j < size-i; ++j)
            if (data[j] > data[j+1]) {
                tmp = data[j];
                data[j] = data[j+1];
                data[j+1] = tmp;
                flag = true;                // 本轮气泡过程中发生过交换
            }
        if (!flag) break;                // 排序提前结束
    }
}
```

【5】设计一函数求两个正整数的最大公约数。

【解】求两个整数的最大公约数最简单的方法是采用蛮力算法。如果要找x和y的最大公约数，首先简单地“猜测”最大公约数是x，因为x的因子不可能大于x。然后将你的假设被x和y除，检查能否整除。如果能整除，答案就有了。如果不能，将这个假设值减1，再继续测试，直到找到一个x和y都能整除的数或假设值减到了1。。

```
int gcd(int a, int b)
{
    for (int r = (a < b ? a : b); r >= 1; --r)
        if (a % r == 0 && b % r == 0)
            return r;
}
```

古希腊的数学家欧几里得提出了一个解决这个问题的非常出色的算法，称为**辗转相除法**，也被称为**欧几里得算法**。该算法描述如下：

取x除以y的余数，称余数为r。

如果r是0，过程完成，答案是y。

如果r非0，设x等于原来y的值，y等于r，重复整个过程。

```
int gcd(int a, int b)
{
    int r;

    while(true) {
        r = a % b;
        if (r == 0) return b;
        a = b;
        b = r;
    }
}
```

同样对于两个数1,000,005和1,000,000，用欧几里得算法只需要执行两次循环体。

【6】设计一个用于整型数的二分查找的递归函数。

【解】在二分查找中，如果中间元素等于要查找的元素，则查找完成。否则，确定要找的数据是在前一半还是在后一半，缩小范围，在前一半或后一半内继续用同样的方法进行查找，即递归调用。

```
int binarySearch(int data[], int start, int end, int x)
{
    int mid = (start + end) / 2;                // mid为中间元素的下标
```

```
    if (start > end) return -1;                // 查找区间不存在，即没有找到
    if (x == data[mid]) return mid;             // 找到
    if (x < data[mid]) return binarySearch(data, start, mid - 1, x);
    else return binarySearch(data, mid + 1, end, x);
}
```

二分查找的包裹函数

```
int binarySearch(int data[], int size, int x)
{   return binarySearch(data, 0, size-1, x);  }
```

【7】设计一个支持整型、实型和字符型数据的快速排序的函数模板。

【解】

```
  //快排函数
  template <class T>
  void quickSort(T data[], int start, int end)
  {
    int mid;

    if (start >= end) return;

     mid = divide(data, start, end);
    quickSort(data, start, mid - 1);
    quickSort(data, mid + 1, end);
  }

//划分函数模板
template <class T>
int divide(T data[], int start, int end)
{
    int i = start, j = end;
    T sample = data[start];

    while (i < j) {
        while (i < j && data[j] >= sample) --j;
        if (i == j) break;
        data[i++] = data[j];
        while (i < j && data[i] <= sample) ++i;
        if (i == j) break;
        data[j--] = data[i];
    }
    data[i] = sample;
    return i;
}

//快速排序的包裹函数
template <class T>
void quickSort(T data, int size)
{   quickSort(data, 0, size - 1);   }
```

【8】硬币找零问题：对于一种货币，有面值为C_1，C_2，…，C_n（分）的硬币，最少需要多少个硬币以及哪些硬币来找出K分钱的零钱。

【解】教材中已介绍如何采用动态规划解决找零问题，本题主要解决如何给出找零方案。用数组lastCoin[i]记录找零i分钱时最后用到的硬币。从lastCoin[maxChange]可以知道最后加入方案的硬币是哪一个，同时也知道最优方案是找零maxChange - lastCoin[maxChange]再加上一个lastCoin[maxChange]的硬币。于是接着检查构成找零maxChange - lastCoin[maxChange]最优方案中最后引入的是哪一个硬币。这个

回溯过程可以得到最优方案使用的每种硬币的数量。

```
void makechange( int coins[ ],   int differentCoins,   int maxChange )
{
    const int MAX = 100;                                          //  最大的找零值为100
    const int MAXCOINS = 10;                                      //  不同值的硬币最多有10个
    int coinUsed[MAX+1], lastCoin[MAX+1], coinNum[MAXCOINS] = {0} ;

    coinUsed[0] = 0;                                              //  找零0分钱需要0个硬币
    for (int cents = 1; cents <= maxChange; ++cents) {            //尝试所有找零
        int minCoins = cents;                                     //最坏情况是都用1分找零
        int newCoin = 1;
        for (int j = 0; j < differentCoins; ++j) {                //尝试所有硬币
            if (coins[j] > cents) continue;                       //coin[j]硬币不可用
            if (coinUsed[ cents - coins[j] ] + 1 < minCoins) {    //用此硬币
                minCoins = coinUsed[ cents - coins[j] ] + 1;
                newCoin = coins[j];
            }
        }
        coinUsed[cents] = minCoins;                               //  记录找零sents分的最小硬币数
        lastCoin[cents] = newCoin;                                //  记录该方案的最后加入的硬币值
    }
    cout << "找零" << maxChange << "  需要" << coinUsed[maxChange]
         << "个硬币" << endl;

   //统计方案中的各硬币数
   for (cents = maxChange; cents > 0; cents -= lastCoin[cents])   //  回溯
        for (int j = 0; j < differentCoins; ++j)
            if (lastCoin[cents] == coins[j]) {
                ++coinNum[j];
                break;
            }
   for (int j = 0; j < differentCoins; ++j)                       //  输出各种硬币的数量
        cout << coins[j] << '\t' << coinNum[j] << endl;
}
```

【9】设计一函数，用动态规划求Fibonacci(n)。

【解】用动态规划求Fibonacci数的方法如下：定义一个数组f存放Fibonacci数，首先设f[0]= 0, f[1] = 1，然后由f[0]、f[1]生成f[2]，再由f[1]、f[2]生成f[3]，……，直到生成f[n]。f[n]就是Fibonacci(n)的值。

```
int Fibonacci(int n)
{
    const int MAX = 100;
    int f[MAX];

    // 设置f0和f1
    if ( n == 0) return 0;
    if (n == 1) return 1;

    f[0] = 0; f[1] = 1;
    for (int i = 2; i <= n; ++i)       // 逐个生成fi
        f[i] = f[i - 1] + f [i - 2];

    return f[n];
}
```

【10】设计一个函数，输出小于100的所有的Fibonacci数。

【解】本题的实现思想与第9题相同，从小到大逐个生成Fibonacci数。每生成一个就显示一个。

```
void Fibonacci()
{
    double f[101];

    cout << 0 << '\t' << 1 << '\t';
    f[0] = 0; f[1] = 1;
    for (int i = 2; i <= 100; ++i)
        cout << (f[i] = f[i - 1] + f [i - 2]) << '\t';
}
```

【11】设计一函数，在一个M个元素的整型数组中找出第N大的元素（N < M)。

【解】对数组a进行N次选择。第一次从a[0]到a[m-1]中选出最大的元素，将它交换到a[0]。第二次从a[1]到a[m-1]中选出最大的元素，将它交换到a[1]。依此类推，第n次从a[n-1]到a[m-1]中选出最大的元素，将它交换到a[n-1]。这个数就是数组中第n大的数。

```
int findN(int a[], int size, int n)
{
    const int MININT = -100000000;
    int i, j, max, index;

    for (i = 0; i < n; ++i) {                    // 执行n次选择
        max = MININT;
        for (j = i; j < size; ++j)                  // 执行第i次选择
            if (a[j] > max) { max = a[j]; index = j; }
        a[index] = a[i];
        a[i] = max;
    }

    return max;
}
```

【12】编写一个函数，按如下格式打印你的名字。

```
*************************
*                       *
*            你的名字                 *
*                                      *
*************************
```

【解】函数有个参数，即要打印的名字，没有返回值。前两行和后两行都只需要直接输出就可以了。将名字打印在中间只需要前后空格数相同。可以计算一下空格数，平均分配一下、

```
void printName(char name[])
{
    int len = strlen(name), i;

    cout << "*****************************" << endl;
    cout << "*                            *" << endl;
    cout << '*';
    for (i = 0; i < 14 - len / 2; ++i) cout << ' ';
    cout << name;
    for (i = 0; i < 14 - len + len / 2; ++i) cout << ' ';
    cout << '*' << endl;
    cout << "*                            *" << endl;
    cout << "*****************************" << endl;
}
```

【13】写一个将英寸转换为厘米的函数（1英寸等于2.54厘米)。

【解】

```
  double inchToCm(double inch)
```

```
{
  return inch * 2.54;
}
```

【14】编写一个函数，要求用户输入一个小写字母。如果用户输入的不是小写字母，则要求重新输入，直到输入了一个小写字母。返回此小写字母。

【解】

```
char getLowerLetter()
{
  char ch;

  while (true) {
      cout << "请输入一个小写字母：";
      ch = cin.get();
      if (ch <= 'z' && ch >= 'a') return ch;
  }
}
```

【15】写三个函数，分别实现对一个双精度数向上取整、向下取整和四舍五入的操作。

【解】

```
//双精度数向上取整
int ceil(double x)
{
    int floor = x;
    return (x - floor < 1e-300 ? floor : floor + 1) ;
}

// 双精度数向下取整
int floor(double x)
{    return int(x);   }

// 双精度数四舍五入
int round(double x)
{
    return int(x+0.5);
}
```

【16】编写一个递归函数reverse，它有一个整型参数。reverse函数按逆序打印出参数的值。例如，参数值为12345时，函数打印出54321。

【解】把逆序打印一个正整数分解成两个问题：打印个位数和逆序打印除去个位数后的数字。

```
void reverse(int n)
{
  cout << n % 10;
  if (n < 10) return;
  reverse(n / 10);
}
```

【17】编写一个函数reverse，它有一个整型参数和一个整型的返回值。reverse函数返回参数值的逆序值。例如，参数值为12345时，函数返回54321。

【解】首先取出参数的个位数，暂且把它当作返回值rev。然后再检查参数是不是有十位数。如果有，参数的十位数才是返回值的个位数，而原来的返回值是十位数。真正的返回值是rev*10再加上参数的十位数。依此类推，直到处理了参数的所有数字。

```
int reverse(int n)
{
  int rev = 0;

  for (; n > 0; n /= 10 )
      rev = rev * 10 + n % 10;
```

```
    return rev;
  }
```

【18】编写一个函数模板，判断两个一维数组是否相同。模板参数是数组的类型。

【解】两个数组相同指的是两个一维数组对应的元素都相同，这需要一个重复n次的循环来判别。只要发现一个对应元素不相等，就可以断定两个数组不相等。如果比较了所有元素，没有发现不相等的元素，可以断定两个数组是相等的。

```
  template <class T>
  bool comp(T a[], T b[], int size)
  {
    for (int i = 0; i < size; ++i)
        if (a[i] != b[i]) return false;

    return true;
  }
```

【19】设计一个用直接选择排序法排序n个整型数的递归函数。

【解】递归过程首先找出从第k个元素到第size-1个元素中的最小值，交换到第k个位置，然后递归调用这个过程找出从第k+1个元素到第size-1个元素中的最小值，交换到第k+1个位置。递归终止条件是k等于size-1。

```
//直接选择法排序整型数组的递归函数
void sort(int a[], int size, int k)
{
    int min = a[k], index;

    if (k == size - 1) return;                    // 递归终止条件
    for (int j = k; j < size; ++j)
        if (a[j] < min) { min = a[j]; index = j; }
    a[index] = a[k];
    a[k] = min;
    sort(a, size, k+1);
}

//直接选择法递归函数的包裹函数
void sort(int a[], int size)
{   sort(a, size, 0);   }
```

【20】编写一函数int count()，使得第一次调用时返回1，第二次调用时返回2。即返回当前的调用次数。

【解】设置一个初值为0的静态局部变量，每次调用函数都将此变量加1，这个变量的值就是函数被调用的次数。

```
  int count()
  {
    static int cnt = 0;

    return ++cnt;
  }
```

【21】假设系统只支持输出一个字符的功能，试设计一个函数void print(double d)输出一个实型数d，保留8位精度。如果d大于10^8或小于10^{-8}，则按科学计数法输出。

【解】打印实数可以分为打印符号位和后面的数字部分。数字部分按数字的范围分别选择按十进制方式输出或按科学计数法输出。如果有两个函数：printFix可以十进制方式打印一个正实数，printFloat可以科学计数法打印一个正实数，那么print函数可以通过调用这两个函数实现

```
// 输出实型数的函数
```

```
void print(double d)
{
    if (d < 0) { cout << '-'; d = -d; }           // 打印符号位，将d设为d的绝对值

    if (d > 1e8 || d < 1e-8)     // 选择打印方式
        printFloat(d);
    else printFix(d);
}
```

以十进制方式输出一个实数可以分成输出整数部分和输出小数部分。由于输出整数的工作在科学计数法输出指数时也要用到，所以将它独立出来，设计成一个函数printInt。以十进制方式输出一个实数可以由下列步骤组成：①调用printInt打印d的整数部分；②统计整数部分的长度；③输出小数点；④打印除最后一位外的小数部分；⑤处理最后一位的四舍五入问题，并打印。

```
//以十进制打印一个正实型数
void printFix(double d)
{
    int scale = 1, digit;       // scale是d的数量级, digit当前处理位的值
    int len = 0;               // 已打印位数
    double epsilon = 1e-8;


    printInt(d);                // 打印整数部分

    if ( d - int(d) <= 1e-8) return;      // d没有小数部分

    // 确定整数部分的长度len
    while (scale < d) { scale *= 10; ++len; epsilon *= 10; }
    scale /= 10;

    cout << '.';

    d = d - int(d);                    // 取d的小数部分

    // 打印小数部分
    while (len < 7 && d > epsilon) {
        digit = d *= 10;
        cout << char(digit + '0');
        d -= digit;
        ++len;
        epsilon *= 10;
    }

    if (d > 1e-8) {                    // 处理四舍五入
        digit = d *= 10;
        if (d - digit > 0.5) ++digit;
        cout << char(digit + '0');
    }
}
```

printInt函数首先检查n的符号。如果是负数，先输出一个'-'再取n的绝对值。然后用递归的方法打印n。递归打印整数分成两个阶段：调用递归函数打印n/10，打印n的个位数。

```
  //打印一个整数
  void printInt(int n)
  {
    if (n < 0) { cout << '-'; n = -n; }
    if (n < 10) cout << char(n +'0');
```

```
    else {
        printInt(n / 10);
        cout << char(n % 10 + '0');
    }
  }
```

以科学计数法打印一个实数时，首先将实数变成d.ddddddd*10^{len}的形式，然后调用printFix打印尾数部分，再打印‘e’，最后调用printInt打印指数部分。

```
  //以科学计数法打印一个正实数
  void printFloat(double d)
  {
    int len = 0;                            // 已打印位数

     // 求指数的值
    if (d > 1)
        while (d > 10) { d /= 10; ++len;}
    else
        while (d < 1) { d *= 10; --len; }

    printFix(d);
    cout << 'e';
    printInt(len);
  }
```

【22】用级数展开法计算平方根。根据泰勒公式

$$f(x)\cong f(a)+f'(a)(x-a)+f''(a)\frac{(x-a)^2}{2!}+f'''(a)\frac{(x-a)^3}{3!}+\cdots+f^{(n)}(a)\frac{(x-a)^n}{n!}$$，

可求得

$$\sqrt{x}\cong 1+\frac{1}{2}(x-1)-\frac{1}{4}\frac{(x-1)^2}{2!}+\frac{3}{8}\frac{(x-1)^3}{3!}-\frac{15}{16}\frac{(x-1)^4}{4!}+\cdots$$

设计一个函数计算$\sqrt{x}$的值。要求误差小于10^{-6}。

【解】如果第n-1项的值是item，则第n项的值为$item\times\exp\times\frac{(x-1)}{n}$。item的初值为1。将误差作为数列求和的终止条件。如果数列当前的和值是s，则终止条件可以是x-s*s的绝对值小于10^{-6}。

```
//计算平方根的函数
double Sqrt(double x)
{
    double s = 0, item = 1, exp = 0.5;
    int n = 1;

    while (x - s * s > 1e-6 || x - s * s < -1e-6) {
        s += item;
        item = item * (x - 1) / n * exp;
        ++n;
        --exp;
    }
    return s;
}
```

【23】可以用下列方法计算圆的面积：考虑四分之一个园，将它的面积看成是一系列矩形面积之和。每个矩形都有固定的宽度，高度是圆弧通过上面一条边的中点。设计一个函数double area(double r, int n)；用上述方法计算一个半径为r的圆的面积。计算时将四分之一个圆划分成n个矩形。

【解】如果矩形中点的x坐标为x，则矩形面积为$\frac{r}{n}\times\sqrt{r^2-x^2}$。每个循环周期将一个矩形面积

加入到面积和s，循环结束时，s保存的是四分之一的圆面积。4s就是整个圆面积。

```
double area(double  r, int n)
{
    double s = 0, delt = r / n, x;          // delt是矩形的宽度

    for ( x = delt / 2; x <= r; x += delt)
        s += sqrt(r * r - x * x) * delt;

   return 4 * s;
}
```

【24】创建一个函数Fib。每调用一次就返回Fibonacci序列的下一个值。即第一次调用返回1，第二次调用返回1，第三次调用返回2，第四次调用返回3，……。

【解】在第n次调用时，Fib函数必须知道F_{n-1}和F_{n-2}。这可以通过静态的局部变量来实现。

```
int Fibonacci()
{
    static int f0 = 0, f1 = 0;              // 保存Fn-1和Fn-2
    int f2;

    if (f1 == 0) { f1 = 1; return f1; }       // 第一次调用，返回1
    f2 = f0 + f1;                           // 计算Fn
    f0 = f1; f1 = f2;                       // 为下一次调用做准备
    return f1;
}
```

【25】设计一个基于分治法的函数，找出一组整型数中的最大值。

【解】按照分治法，将该问题分解成找出前一半和后一半的最大值，返回两个值中较大者。

```
int max(int data[], int start, int end)
{
     int max1, max2;

     if (start == end) return data[start];
     if (end - start == 1) return (data[start] > data[end] ? data[start] : data[end]);

      max1 = max(data, start, (start +    end) / 2);
     max2 = max(data, (start +    end) / 2 + 1, end);

     return (max1 > max2 ? max1 : max2);
}
```

第7章 简答题

【1】下面的定义所定义的变量类型是什么？

```
    double *p1, p2;
```

【解】p1是指向double类型的指针变量， p2是一个double类型的变量。

【2】如果arr被定义为一个数组，描述以下两个表达式之间的区别。

```
    arr[2]
    arr + 2
```

【解】arr[2]是数组arr的第3个元素， arr + 2是数组arr第3个元素的地址。

【3】. 假设double类型的变量在你使用的计算机系统中占用8个字节。如果数组doubleArray的基地址为1000，那么doubleArray + 5的值是什么？

【解】doubleArray + 5的值是1040。

【4】定义int array[10], *p = array;后，可以用p[i]访问array[i]。这是否意味着数组和指针是等同的？

【解】数组和指针是完全不同的，数组变量存放了一组同类元素，指针变量中存放了一个地址。由于在C++中数组名是数组的起始地址，因此在将一个数组名赋给一个指针后，该指针

指向了数组的起始地址，因此可以和数组名有同样的行为。

【5】字符串是用字符数组来存储的。为什么传递一个数组需要两个参数（数组名和数组长度），而传递字符串只要一个参数（字符数组名）？

【解】数组传递通常需要两个参数：数组名和元素个数。字符串在C++中是存储在一个字符类型的数组中，所以传递一个字符串也是传递一个数组。但由于C++规定每个字符串必须以'\0'结束，所以传递字符串时就不需要指出元素个数了。

【6】值传递、引用传递和指针传递的区别是什么？

【解】值传递主要用作为函数的输入。在引用传递中，形式参数是实际参数的一个别名，形式参数并没有自己的空间，它操作的是实际参数的空间。指针传递时，形式参数是一个指针变量。函数中可以通过间接访问调用函数中的某个变量。引用传递和指针传递通常可将被调用函数中的运行结果传回调用函数。

【7】如何检查new操作是否成功？

【解】new操作成功，返回该动态变量的内存地址。如果不成功，返回一个空指针，即0。

【8】返回引用的函数和返回某种类型的值的函数在用法上有什么区别？计算机在处理这两类返回时有什么区别？

【解】普通的函数调用只能作为右值，而返回引用的函数可以作为左值。普通函数返回时，计算机用return语句中的表达式构造一个临时变量，用该临时变量替代主函数中的函数调用，所以该函数的调用只能作为右值。返回引用的函数的return语句后面一定是一个变量，而且该变量在函数调用结束后还存在。函数返回时，将return后面的变量替代主函数中函数调用。由于该函数调用等价于一个变量，所以可以作为左值，对其赋值。

【9】. 如果p是变量名，下面表达式中那些可以作为左值？请解释。

p　　*p　　&p　　*p+2　　*(p+2)　　&p+2

【解】p、*p、*（p+2）是左值，其他不是左值。

【10】如果p是一个指针变量，取p指向的单元中的内容应如何表示？取变量p本身的地址应如何表示？

【解】取p指向的单元中的内容应表示为*p，取变量p本身的地址应表示为&p。

【11】. 如果一个new操作没有对应的delete操作会有什么后果？

【解】如果一个new操作没有对应的delete操作会造成内存泄漏。

7.2.2　程序设计题

【1】用原型int getDate(int &dd, int &mm, int &yy);写一个函数从键盘读入一个形如dd-mmm-yy的日期。其中dd是一个1位或2位的表示日的整数，mmm是月份的三个字母的缩写，yy是两位数的年份。函数读入这个日期，并将它们以数字形式传给三个参数。

【解】

```
int getDate(int &dd, int &mm, int &yy)
{
    char *month[12] = {"Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"};
    char ch[10];
    int i;

    dd = mm = yy = 0;
    cout << "请输入日期：  ";
    cin >> ch;

    if (ch[1] == '-')  {                                  // 将一位表示的日用两位表示
        for (i = 9; i > 0; --i) ch[i] = ch[i-1];
        ch[0] = '0';
    }
```

```
    // 处理日
    if   (ch[0] >= '0' && ch[0] <= '9') dd = ch[0] - '0';   else return 1;
    if (ch[1] >= '0' && ch[1] <= '9') dd = 10 * dd + ch[1] - '0'; else return 1;

     if (ch[2] != '-') return 1;

    // 处理月
    for (mm = 0; mm < 12; ++mm)
         if (strncmp(&ch[3], month[mm], 3) == 0) break;
    if (mm == 12) return 1;       else ++mm;

    if (ch[6] != '-') return 1;

    // 处理年
    if (ch[7] >= '0' && ch[7] <= '9') yy = ch[7] - '0';   else return 1;
    if (ch[8] >= '0' && ch[8] <= '9') yy = 10 * yy + ch[8] - '0'; else return 1;

     if (ch[9] != '\0') return 1; else return 0;
}
```

【3】设计一个函数char *itos(int n)，将整型数n转换成一个字符串。

【解】假设在VC6.0中实现此函数。由于在VC6.0中，整数占用的空间是4位，最大的表示范围是±10位十进制，因而在程序中定义了一个长度为12个字符的动态字符数组s存放转换后的字符串。

```
char *itos(int n)
{
    char *s = new char[12];
    int i = 0, scale = 1000000000;

    if (n == 0) {             // 处理特殊情况0
        s[0] = '0';
        s[1] = '\0';
        return s;
    }
    if (n < 0) {              // 处理-号
        s[0] = '-';
        n = -n;
        ++i;
    }
    while (n / scale == 0) scale /= 10;            // 压缩整数前面的0
    while (scale > 0) {                           // 从高到低处理每一位
        s[i] = n / scale + '0';
        n %= scale;
        ++i;
        scale /= 10;
    }
    s[i] = '\0';

    return s;
}
```

【4】用带参数的main函数实现一个完成整数运算的计算器。如果该函数对应的可执行文件名为calc，在命令行界面输入

```
    calc  5 * 3
```

可得到执行结果为15。

【解】首先将argv[1]和argv[3]中的字符串转换成数字，然后根据argv[2]的第一个字符分成四个分支，分别执行+、-、*、/运算。

```
int main(int argc, char *argv[])
{
    int num1 = 0, num2 = 0, i;

    for ( i = 0 ; argv[1][i] != '\0'; ++i)       // 转换左运算数
        num1 = num1 * 10 + argv[1][i] - '0';
    for ( i = 0 ; argv[3][i] != '\0'; ++i)       // 转换左运算数
        num2 = num2 * 10 + argv[3][i] - '0';

    switch(argv[2][0]) {                        // 执行四则运算
       case '+': cout << num1 + num2 << endl; break;
       case '-': cout << num1 - num2 << endl; break;
       case '*': cout << num1 * num2 << endl; break;
       case '/': cout << num1 / num2 << endl; break;
       default: cout << "error" << endl;
    }

   return 0;
}
```

【5】编写一个函数，判断作为参数传入的一个整型数组是否为回文。所谓回文，就是第一个数与最后一个数相同，第二个数与倒数第二个数相同，依此类推。例如，若数组元素值为10、5、30、67、30、5、10就是一个回文。

【解】函数的参数是一个整型数组，因此它有两个形式参数：数组名和数组规模。函数判别数组是否为回文，所以返回值是一个bool类型的值。

```
bool plalindrome(int a[], int size)
{
    for (int i = 0; i < size / 2; ++i)
        if (a[i] != a[size - 1 - i]) return false;

    return true;
}
```

【6】. Julian历法是用年及这一年中的第几天来表示日期。设计一个函数将Julian历法表示的日期转换成月和日，如Mar 8（注意闰年的问题）。函数返回一个字符串，即转换后的月和日。如果参数有错，如天数为第370天，返回NULL。

【解】函数的参数是年份和日期，返回值是一个字符串，即指向字符的指针，指向一个动态字符数组。

```
char *Julian(int year, int day)
{
    char *date = new char[7];              // 存放返回值
    int dayInMonth[12] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};   //每月的天数
    char *month[12] = { "Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"};

    if ((year % 400 == 0) || (year % 4 == 0 && year % 100 != 0))       // 如果是闰年，二月份的天数加1
        ++dayInMonth[1];
    for (int i = 0; day > dayInMonth[i]; ++i)    day -= dayInMonth[i];   // 获取月份
    strcpy(date, month[i]);                                              // 将月份保存到date
    date[3] = ' ';
    if (day > 9)   {                                                    // 将日转换成字符串存入date
        date[4] = day / 10 + '0';
        date[5] = day % 10 + '0';
        date[6] = '\0';
    }
    else {
        date[4] =    day + '0';
        date[5] = '\0';
```

```
    }

    return date;
}
```

【7】编写一个魔阵生成的函数。函数的参数是生成的魔阵的阶数，返回的是所生成的魔阵。

【解】函数的参数是魔阵的阶数，因而函数有一个整型参数scale。函数的返回值是生成的魔阵，是一个二维数组，而二维数组的名字是一个二级指针。所以函数的返回值可以设计成指向整型的二级指针。函数首先申请一个二维的动态数组，并按照没阵填写规则填写魔阵，最后返回保存魔阵的二维数组。

```
int **magicArray(int scale)
{
    int row, col, count, i, j;
    int **magic;                        // 存放魔阵的二维数组

    // 申请动态的二维数组
    magic = new int *[scale];
    for (i = 0; i < scale; ++i) {
        magic[i] = new int[scale];
        for (j = 0; j < scale; ++j) magic[i][j] = 0;
    }

   // 填写魔阵
   row=0; col = (scale - 1) / 2; magic[row][col] = 1;
   for (count = 2; count <= scale * scale; count++) {
     if (magic[(row - 1 + scale) % scale][(col + 1) % scale] == 0) {
        row = ( row - 1 + scale ) % scale;
        col = ( col + 1 ) % scale;
     }
     else  row = ( row + 1 ) % scale;
     magic[row][col] = count;
   }

    return magic;
 }
```

【8】在统计学中，经常需要统计一组数据的均值和方差。均值的定义为$\bar{x} = \sum_{i=1}^{n} x_i / n$，方差的定义为$\sigma = \frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n}$。设计一个函数，对给定的一组数据返回它们的均值和方差。

【解】函数有一个数组参数。函数的执行结果是均值和方差。由于函数需要两个返回值，我们将这两个返回值设计成输出参数。均值和方差的计算只需按标准公式计算即可。

```
void statistic(double data[], int size, double &means, double &dev)
{
    int i;

    means = dev = 0;

    // 计算均值
    for (i = 0; i < size; ++i) means += data[i];
    means /= size;

    // 计算方差
    for (i = 0; i < size; ++i) dev += (data[i] - means) * (data[i] - means);
    dev /= size;

    return;
 }
```

【9】设计一个用弦截法求函数根的通用函数。函数有三个参数：第一个是指向函数的指针，指向所要求根的函数；第二、三个参数指出根所在的区间。返回值是求得的根。

【解】

```
double root(double (*f)(double), double x1, double x2)
{
    double x, f2, f1, fx;

    do {
        f1 = f(x1);
        f2 = f(x2);
        x = (x1 * f2 - x2 * f1) / (f2 - f1);
        fx = f(x);
        if (fx * f1 > 0) x1 = x;  else x2 = x;
    } while (fabs(fx) > 1e-10);

    return x;
}
```

【10】. 设计一个计算任意函数的定积分的函数。函数有三个参数：第一个是指向函数的指针，指向所要积分的函数；第二、三个参数是定积分的区间。返回值是求得的积分值。定积分的计算方法是采用矩形法。

【解】

```
double integral(double (*f)(double), double x1, double x2)
{
    const double EPSILON = 0.01;
    const int NUM = 1000;
    double x, s = 0, delt = EPSILON;

    if ((x2 - x1) / num < delt) delt = (x2 - x1) / num;   // 决定小矩形的宽度
    for (x = x1 + delt / 2; x <= x2; x +=delt) s += delt * f(x);

    return s;
}
```

第8章 简答题

【1】判断：数组中的每个元素类型都相同。

【解】数组是一组同类元素的有序集合，所以数组中的元素类型都是相同的。

【2】判断：结构体中每个字段的类型都必不相同。

【解】结构体用于需要多个元素描述一个对象的情况，它并不在乎字段类型是否一致。例如在学生记录中，每个学生的信息包括学号、姓名、各科成绩，这时各字段的类型就不一致。如果要描述的记录是二维平面上的一个点，描述二维平面上的一个点需要两个字段，即x坐标和y坐标，这两个字段的类型是一样的，都是double类型。

【3】使用一个结构体类型的变量的必要步骤是什么？

【解】定义结构体类型、定义结构体变量、使用结构体类型的变量。

【4】从结构体类型的变量中选取某个字段使用哪个运算符？

【解】使用“.”运算符。

【5】判断：C++语言中结构体类型的变量本身是左值。

【解】C++语言中结构体类型的变量本身是左值。

【6】如果变量p被定义为一个结构体指针，结构体中包括字段cost，要通过指针p选取cost字段时，表达式*p.cost有何错误？在C++语言中完成该操作应该用什么表达式？

【解】因为“.”运算符比“*”运算符优先级高，因此*p.cost会被认为cost是一个指针，该表达式取结构体变量p的cost字段指向的空间的值。正确的表示方法是：（*p）.cost或

p->cost。尽管两种表示方法都可以，但C++程序员习惯用后者。

【7】引入结构体有什么好处？

【解】引入结构体可以将一组逻辑上相关的变量组合成一个有机的整体，表达更复杂的对象。

【8】单链表中为什么要引入头结点？

【解】消除插入或删除第一个结点的特殊情况。

【9】. 结构体类型定义的作用是什么？结构体类型的变量定义的作用是什么？

【解】结构体类型定义是告诉C++这种类型的变量是如何组成的，需要多少空间。结构体变量的定义是告诉C++程序中需要存储和处理一个某结构体类型的变量，这时C++根据结构体类型的定义为这个变量分配空间。

第8章 程序设计题

【1】用结构体表示一个复数，编写实现复数的加法、乘法、输入和输出的函数，并测试这些函数。

加法规则：$(a+bi)+(c+di)=(a+c)+(b+d)i$。

乘法规则：$(a+bi)\times(c+di)=(ac-bd)+(bc+ad)i$。

输入规则：分别输入实部和虚部。

输出规则：如果a是实部，b是虚部，输出格式为a + bi。

【解】首先需要定义一个保存复数的结构体类型。复数由两部分组成：实部和虚部。

```
struct complex {                    // 复数类型的定义
    double real;
    double imag;
};

complex input()                     // 输入复数
{
    complex data;

    cout << "请输入实部: ";
    cin >> data.real;
    cout << "请输入虚部: ";
    cin >> data.imag;

    return data;
}

void output(complex data)           // 输出复数
{
    cout << "(" << data.real << " + " << data.imag << "i )";
}

complex add(complex d1, complex d2)    // 两个复数相加
{
    complex result;

    result.real = d1.real + d2.real;
    result.imag = d1.imag + d2.imag;

    return result;
}

complex multi(complex d1, complex d2)   // 两个复数相乘
{
    complex result;
```

```
    result.real = d1.real * d2.real - d1.imag * d2.imag;
    result.imag = d1.imag * d2.real + d1.real * d2.imag;

    return result;
}
```

【2】编写函数Midpoint(p1, p2)，返回线段p1、p2的中点。函数的参数及结果都应该为pointT类型，pointT的定义如下：

```
    struct pointT {
       double x, y;
    };
```

【解】

```
pointT mid(pointT p1, pointT p2)
{
    pointT p;

    p.x = (p1.x + p2.x) / 2;
    p.y = (p1.y + p2.y) / 2;

    return p;
}
```

【3】可以用两个整数的商表示的数称为有理数。因此1.25是一个有理数，它等于5除以4。很多数不是有理数，比如π 和2的平方根。在计算时，使用有理数比使用浮点数更有优势：有理数是精确的，而浮点数不是。因此，设计一个专门处理有理数的工具是非常有用的。

试定义类型rationalT，用来表示一个有理数。

函数CreateRational(num, den)，返回一个rationalT类型的值。

函数AddRational(r1,r2)，返回两个有理数的和。

函数MultiplyRational(r1,r2)，返回两个有理数的乘积。

函数GetRational(r)，返回有理数r的实型表示。

函数PrintRational(r)，以分数的形式将数值显示在屏幕上。

关于有理数的所有计算结果都应化到最简形式，例如，1/2乘以2/3的结果应该是1/3而不是2/6。

【解】首先定义一个表示有理数的结构体。表示一个有理数可以分别表示它的分子和分母。

```
struct RationalT {                                  // 有理数类型的定义
    int num;                                       // 分子
    int den;                                        // 分母
};

RationalT CreateRational(int num, int den)        // 创建一个有理数
{
    RationalT r;

    r.num = num;
    r.den = den;

    return r;
}

RationalT AddRational(RationalT r1, RationalT r2)  // 有理数加法
{
    RationalT r;
    int fac;
```

```
    r.num = r1.num * r2.den + r2.num * r1.den;
    r.den = r1.den * r2.den;

    // 将结果化成最简分式
    fac = r.num < r.den ? r.num : r.den;
    while (r.num % fac != 0 || r.den % fac != 0) -- fac;
    r.num /= fac;
    r.den /= fac;

    return r;
}

RationalT MultiplyRational(RationalT r1, RationalT r2)      // 有理数乘法
{
    RationalT r;
    int fac;

    r.num = r1.num * r2.num;
    r.den = r1.den * r2.den;

    // 将结果化成最简分式
    fac = r.num < r.den ? r.num : r.den;
    while (r.num % fac != 0 || r.den % fac != 0) -- fac;
    r.num /= fac;
    r.den /= fac;

    return r;
}

double GetRational(RationalT r)                 // 将有理数转换成小数
{   return double(r.num) / r.den;    }

void PrintRational(RationalT r)                  // 以分数形式输出有理数
{   cout << r.num << " / " << r.den;   }
```

【4】编一个程序用数组解决约瑟夫环的问题。

【解】根据输入的参数申请了一个动态数组arr。初始时，arr[i]的值为i，表示第i个人在第i个位置。在报数阶段，报到3的人被删除，后面的人往前移。直到最后剩下一个人。为此必须有一个变量size记住目前还剩多少人。报数阶段另一个要解决的问题是如何模拟一个圈，当报到最后一个人时，下一个人是在0号位置。如果报数为1的人在位置i，则报到3的人在位置(i+2) % size。

```
int josephus(int n)
{
    int *arr = new int[n];
    int i, j, size = n;

    for (i = 0; i < n ; ++i) arr[i] = i ;

    i = 0;                                  // 第一个报数的人的位置
    while (size > 1) {                        // 剩下的人数大于1时继续报数
        i = (i+2) % size;                       // 报到3的人的位置
        --size;
        for (j = i; j < size; ++j) arr[j] = arr[j + 1];  // 删除报到3的人
    }

    i = arr[0];                             // 最后一个人的编号
```

```
    delete [] arr;

    return i;
}
```

5．模拟一个用于显示时间的电子时钟。该时钟以时、分、秒的形式记录时间。试编写三个函数：setTime函数用于设置时钟的时间。increase函数模拟时间过去了1秒。showTime显示当前时间，显示格式为HH：MM：SS。

【解】定义一个表示时间的结构体。包括3个信息：时、分、秒。

```
struct clockT {
   int hh;
    int mm;
    int ss;
};

void setTime(clockT &c, int h, int m, int s)
{
    c.hh = h;
    c.mm = m;
    c.ss = s;
}

void increase(clockT &c)
{
    ++c.ss;
    if ( c.ss == 60) {         // ss的修改引起了mm的变化
        ++c.mm;
        c.ss = 0;
    }
    if ( c.mm == 60) {         // mm的变化引起了hh的变化
        ++c.hh;
        c.mm = 0;
    }

    if ( c.hh == 24) c.hh = 0;   // hh的变化使时间进入第二天
}

void showTime(clockT c)
{   cout << c.hh << " : " << c.mm << " : " << c.ss;   }
```

【6】在动态内存管理中，假设系统可供分配的空间被组织成一个单链表，链表的每个结点表示一块可用的空间,用可用空间的起始地址和终止地址表示。初始时,链表只有一个结点，即整个堆空间的大小。当遇到一个new操作时，在链表中寻找一个大于new操作申请的空间的结点。从这个结点中扣除所申请的空间。当遇到delete操作时，将归还的空间形成一个结点，连入链表。经过了一段时间的运行，链表中的结点会越来越多。设计一个函数完成碎片的重组工作。即将一系列连续的空闲空间组合成一块空闲空间。

【解】单链表的每个结点保存一块内存信息，用起始和终止地址表示。整理工作分成两个阶段。首先将链表中的内存块按起始地址的递增次序排列，然后检查相邻两结点的的内存块是否连续，如果连续则合并成一个结点。排序链表采用了直接选择排序。

```
struct memo {                          // 单链表的结点类
      int start;
      int finish;
      memo *next;
};
```

```
void arrage(memo *head)
{
    memo *insPos = head, *curPos, *minPos;    // insPos：插入位置的前一结点，
            //minPos：起始地址最小的结点的前一结点，curPos：当前正在检查的结点的前一结点

    while (insPos->next != NULL) {                    // 直接插入排序
        curPos = minPos = insPos;
        while (curPos->next != NULL) {                // 找起始地址最小的节点
            if (curPos->next->start < minPos->next->start)    minPos = curPos;
            curPos = curPos->next;
        }
        // 插入最小节点
        curPos = minPos->next;
        minPos->next = curPos->next;
        curPos->next = insPos->next;
        insPos->next = curPos;
        insPos = insPos->next;
    }

    for (curPos = head->next; curPos->next != NULL; curPos = curPos->next) {    // 合并连续的内存块
        while (curPos->finish + 1 == curPos->next->start) {       // 当前块与下一块连续，执行合并
            minPos = curPos->next;
            curPos->finish = minPos->finish;
            curPos->next = minPos->next;
            delete minPos;
            if (curPos->next == NULL) return;
        }
    }
}
```

第9章 简答题

【1】判断题：每个模块对应于一个源文件。

【解】对。

【2】用自己的话描述逐步细化的过程。

【解】逐步细化就是将一个大问题分成若干个小问题，小问题分解成小小问题，直到一个问题可以用一段小程序实现为止。每个问题的解决过程是一个函数，实现小问题的函数通过调用解决小小问题的函数来实现。解决大问题的函数通过调用解决小问题的函数来实现。在解决一个较大的问题时，只需要知道有哪些可供调用的解决小问题的函数，而不必关心这些解决小问题的函数是如何实现的。这样可以在一个更高的抽象层次上解决大问题。

【3】为什么库的实现文件要包含自己的头文件？

【解】保证实现文件中的原型和提供给用户程序员用的函数原型完全一致。

【4】为什么头文件要包含#ifndef...#endif这对编译预处理指令？

【解】这对编译预处理命令表示：如果#ifndef后的标识符已经定义过，则跳过中间的所有指令，直接跳到#endif。一个程序可能由很多源文件组成，每个源文件都可能调用到库中的函数，因此每个源文件都需要包含库的头文件。如果没有#ifndef...#endif这对编译预处理指令，头文件中的内容在整个程序中可能出现很多遍，将造成编译或连接错误。有了#ifndef...#endif这对编译预处理指令可以保证头文件的内容在整个程序中只出现一遍。

【5】什么是模块的内部状态？内部状态是怎样保存的？

【解】模块的内部状态就是模块内多个函数需要共享的信息，这些信息与其他模块中的函数无关。内部状态通常被表示为源文件中的全局变量，以方便模块中的函数共享。

【6】为什么要使用库？

【解】库可以实现代码重用。某个项目中各个程序员需要共享一组工具函数时可以将这组函

数组成一个库，这些函数的代码在项目中得到了重用。如果另一个项目中也许要这样的一组工具函数，那么这个项目的程序员就不必重新编写这些函数而可以直接使用这个库，这样这组代码在多个项目中得到了重用。

第9章 程序设计题

【1】哥德巴赫猜想指出：任何一个大于6的偶数都可以表示成两个素数之和。编写一个程序，列出指定范围内的所有偶数的分解。

【解】本题可以枚举法解决。主程序先接受用户输入的范围，然后枚举该范围中的每一个偶数i。对于每一个偶数，尝试把它分成两个3以上的奇数：j和i-j。检查j和i-j是否为素数，直到找到一对都为素数的j和i-j，输出这一对数。

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

bool isPrime(int n);                // 判别n是否为素数

int main()
{
    int start, end, i, j;

    cout << "请输入一个6以上的偶数范围：";
    cin >> start >> end;

    for ( i = start; i <= end; i += 2)          // 枚举范围内所有的偶数
        for (j = 3; j <= i / 2; j += 2)           // 枚举i的每一种分解
            if (isPrime(j) && isPrime(i-j)) {
                cout << i << " = "  << j << " + " << i-j << '\t';
                break;
            }

    return 0;
}

bool isPrime(int n)
{
    if (n <= 1) return false;
    if ( n == 2) return true;

    for (int i = 3; i <= sqrt(n); i += 2)
        if (n % i == 0) return false;

    return true;
}
```

【2】在每本书中，都会有很多图或表。图要有图号，表要有表号。图号和表号都是连续的，如一本书的图号可以编号为：图1、图2、图3、……。设计一个库seq，它可以提供这样的标签系列。该库提供给用户3个函数：void SetLabel(const char *)、void SetInitNumber(int)和char * GetNextLabel()。第一个函数设置标签，如果SetLabel("图")，则生成的标签为图1、图2、图3、……；如果SetLabel("表")，则生成的标签为表1、表2、表3、……；如不调用此函数，则默认的标签为"lable"。第二个函数设置起始序号，如SetInitNumber(0)，则编号从0开始生成；SetInitNumber(9)，则编号从9开始生成。第三个函数是获取标签号。例如，如果一开始设置了SetLabel("图")和SetInitNumber(0)，则第一次调用GetNextLabel()返回“图0”，第二次调用

GetNextLabel()返回“图1”，依次类推。

【解】设计一个库包括两个方面：设计接口文件（.h文件）和设计实现文件(.cpp文件)。根据题意，库包含三个函数：void SetLabel(const char *)、void SetInitNumber(int)和char * GetNextLabel()。

```
//seq库的接口文件Seq.h
#ifndef _SEQ
#define _SEQ

// 设置标签
void SetLabel(const char *s);

// 设置编号
void SetInitNumber(int);

// 获取标签
char * GetNextLabel();

#endif

//seq库的实现文件Seq.cpp
#include "seq.h"
#include <cstring>

char label[6] = "label";              // 保存标签
int no = 0;                           // 保存序号

void SetLabel(const char *s)
{   strcpy(label, s);  }

// 设置编号
void SetInitNumber(int num)
{   no = num;   }

// 获取标签
char *GetNextLabel()
{
    char tmp[11] = {'\0'};
    int i, tmpNo = no;
    static char *result = NULL;          // 保存返回值

    if (no == 0) {
        tmp[9] = '0';
        i = 8;
    }
    else for (i = 9; tmpNo != 0; --i, tmpNo /= 10)
             tmp[i] = tmpNo % 10 + '0';

    if (result != NULL) delete result;

    int len = strlen(label) + 10 - i;
    result = new char[len];
    strcpy(result, label);
    strcat(result, tmp + i + 1);
    result[len - 1] = '\0';
    ++no;
```

```
    return result;
}
```

【3】实现一个支持复数运算的库，库的功能有：复数的加法、乘法、输入和输出的函数。

【解】首先定义存储复数的结构体类型。它有两个实型的成员，分别表示实部和虚部。

```
//复数库的接口文件complex.h
#ifndef _COMPLEX
#define _COMPLEX

struct complex {
    double real;
    double imag;
};

complex input();
void output(complex data);
complex add(complex d1, complex d2);
complex multi(complex d1, complex d2);

#endif

//复数库的实现文件Complex.cpp
#include "complex.h"
#include <iostream>
using namespace std;

complex input()
{
    complex data;

    cout << "请输入实部：";
    cin >> data.real;

    cout << "请输入虚部：";
    cin >> data.imag;

    return data;
}

void output(complex data)
{
    cout << "(" << data.real << " + " << data.imag << "i )";
}

complex add(complex d1, complex d2)
{
    complex result;

    result.real = d1.real + d2.real;
    result.imag = d1.imag + d2.imag;

    return result;
}


complex multi(complex d1, complex d2)
{
```

```
    complex result;

    result.real = d1.real * d2.real - d1.imag * d2.imag;
    result.imag = d1.imag * d2.real + d1.real * d2.imag;

    return result;
}
```

【4】实现一个支持二维平面上点类型的库。支持的功能有设置点的值、输出点的值以及返回线段p1、p2的中点。

【解】首先定义一个表示二维平面上点的结构体，结构体有两个实型的成员，分别表示x和y坐标值。

```
//二维平面上点运算库的头文件point.h
#ifndef _POINT
#define _POINT

struct point {
    double x;
    double y;
};

void setPoint(point &p, double xx, double yy);
void printPoint(point p);
point midPoint(point p1, point p2);

#endif
```

```
//二维平面上点运算库的实现文件 point.cpp
#include "point.h"
#include <iostream>
using namespace std;

void setPoint( double xx, double yy,  point &p)
{
    p.x = xx;
    p.y = yy;
}

void printPoint(point p)
{   cout << "(" << p.x << ", " << p.y << ")";   }

point midPoint(point p1, point p2)
{
    point p;
    p.x = (p1.x + p2.x) / 2;
    p.y = (p1.y + p2.y) / 2;
    return p;
}
```

5．实现一个支持有理数运算的库。支持的功能有：

函数CreateRational(num, den)，返回一个rationalT类型的值。

函数AddRational(r1,r2)，返回两个有理数的和。

函数MultiplyRational(r1,r2)，返回两个有理数的乘积。

函数GetRational(r)，返回有理数r的实型表示。

函数PrintRational(r)，以分数的形式将数值显示在屏幕上

【解】定义表示有理数的结构体类型。它有两个整型的成员，分别表示分子和分母。

```
//有理数库的接口文件rational.h
#ifndef _RATIONAL
#define _RATIONAL

struct rational {
   int num;         // 分子
    int den;          // 分母
};

rational CreateRational(int num, int den);
rational AddRational(rational r1, rational r2);
rational MultiplyRational(rational r1, rational r2);
double GetRational(rational r);
void PrintRational(rational r);

#endif

// 有理数库的实现文件
#include "rational.h"
#include <iostream>
using namespace std;

rational CreateRational(int num, int den)
{
    rational r;
    int fac;

    r.num = num;
    r.den = den;

    // 化简
    fac = r.num < r.den ? r.num : r.den;
    while (r.num % fac != 0 || r.den % fac != 0) -- fac;
    r.num /= fac;
    r.den /= fac;

    return r;
}

rational AddRational(rational r1, rational r2)
{
    rational r;
    int fac;

    r.num = r1.num * r2.den + r2.num * r1.den;
    r.den = r1.den * r2.den;

    // 化简
    fac = r.num < r.den ? r.num : r.den;
    while (r.num % fac != 0 || r.den % fac != 0) -- fac;
    r.num /= fac;
    r.den /= fac;

    return r;
}
```

```
rational MultiplyRational(rational r1, rational r2)
{
    rational r;
    int fac;

    r.num = r1.num * r2.num;
    r.den = r1.den * r2.den;

    fac = r.num < r.den ? r.num : r.den;
    while (r.num % fac != 0 || r.den % fac != 0) -- fac;
    r.num /= fac;
    r.den /= fac;

    return r;
}

double GetRational(rational r)
{   return double(r.num) / r.den;  }

void PrintRational(rational r)
{   cout << r.num << " / " << r.den;  }
```

【6】设计一个字符串处理库，该库提供一组常用的字符串的操作，包括字符串复制、字符串拼接、字符串比较、求字符串长度和取字符串的子串。

【解】

```
//字符串库的头文件 string.h
#ifndef _STRING
#define _STRING

void stringCopy(char *des, const char *src);
void stringNCopy(char *des, const char *src, int len);
void stringCat(char *des, const char *src);
void stringNCat(char *des, const char *src, int len);
int stringCmp(const char *s1, const char *s2);
int stringLen(const char *s);
char *stringSub(const char *s, int start, int len);

#endif

//字符串库的实现文件
#include "string.h"
#include <iostream>
using namespace std;

void stringCopy(char *des, const char *src)
{
    while (*src != '\0') {        // 复制src到des，直到src结束
        *des = *src;
        ++des;
        ++src;
    }
    *des = '\0';
}

void stringNCopy(char *des, const char *src, int len)
{
    int i;
    for (i = 0; i < len && *src != '\0'; ++i) {
```

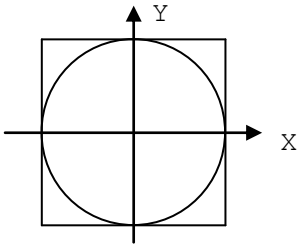
```
        *(des+i) = *src;
        ++src;
    }
    *(des+i) = '\0';
}

void stringCat(char *des, const char *src)
{
    char *p1 = des;

    while (*p1 != '\0') ++p1;           // 找des尾
    while (*src != '\0') {
        *p1 = *src;
        ++p1;
        ++src;
    }
    *p1 = '\0';
}

void stringNCat(char *des, const char *src, int len)
{
    char *p1 = des;

    while (*p1 != '\0') ++p1;
    for (int i = 0; i < len && *src != '\0'; ++i)
        *(p1++) = *(src++);
    *p1 = '\0';
}

int stringCmp(const char *s1, const char *s2)
{
    while (*s1 != '\0' && *s2 !='\0') {          // 比较s1和s2的对应元素
        if (*s1 == *s2) { ++s1; ++s2;}
        else return *s1 - *s2;
    }
    if (*s1 == *s2) return 0;                  // s1和s2长度相同，且对应字符完全相同
    if (*s1 =='\0') return -1; else return 1;      // s1必s2长
}

int stringLen(const char *s)
{
    int len = 0;

    while (*(s++) != '\0') ++len;

    return len;
}

char *stringSub(const char *s, int start, int len)
{
    int length = 0;

    if (len <= 0) return NULL;
    while (*(s+length) != '\0') ++length;     // 求字符串s的长度
    if (start > length) return NULL;          // 检查参数的合法性
    int subLen = (length - start < len? length - start : len);
    char *result = new char[subLen + 1];        // 存放取出的子串
```

```
    for (int i = start; i < subLen; ++i) result[i-start] = s[i];
    result[i-start] = '\0';

    return result;
}
```

【7】π 的近似值的计算有很多种方法，其中之一是用随机数。对于图9-1中的圆和正方形，如圆的半径为r，它们的面积之比有如下关系

$$\frac{\text{圆面积}}{\text{正方形面积}}=\frac{\pi r^2}{4r^2}=\frac{\pi}{4}$$

从中可得 $$\pi=\frac{4\text{倍的圆面积}}{\text{正方形的面积}}$$



可以通过如下的方式计算π 的近似值：假设圆的半径为1，产生-1到1之间的两个随机实数x和y。这个点是正方形中的一个点。如果 $x^2+y^2\le 1$，则点落在圆内。重复n次上述动作，并记录点落在圆内的次数m。则通过 $\pi=\frac{4m}{n}$ 可得π 的近似值。重复的次数越大，得到的π 值越精确。这种技术被称为蒙特卡洛积分法。用主教材实现的随机函数库实现该程序。

【解】

```
#include<iostream>
#include "random.h"
using namespace std;

int main()
{
    int num, inCircle = 0;
    double x, y;

    cout << "输入模拟的点数：";
    cin >> num;

    RandomInit();
    for (int i = 0; i < num; ++i) {
        x = RandomDouble(-1, 1);
        y = RandomDouble(-1, 1);
        if (x * x + y * y <= 1) ++inCircle;
    }

    cout << "Pi = " << 4.0 * inCircle / num << endl;

    return 0;
}
```

第10章　简答题

【1】用 struct 定义类型与用 class 定义类型有什么区别？

【解】在结构体中，缺省的访问特性是公有的。在类中，缺省的访问特性是私有的。

【2】构造函数和析构函数的作用是什么？它们各有什么特征？

【解】构造函数是在对象定义时自动执行，为对象赋初值。析构函数是对象销毁时自动调用，

做一些善后工作。构造函数的名字就是类名，析构函数的名字是波浪号加类名。构造函数和析构函数都不需要写函数的返回类型。对象可能有不同的构造方法，所以类可以有一组重载的构造函数，但析构函数只能有一个。构造函数还可以有一个初始化列表。

【3】. 友元的作用是什么？

【解】使全局函数或其他类的成员函数、甚至是某个类定义所有的成员函数可以直接访问当前类的私有成员。

【4】静态数据成员有什么特征？有什么用途？

【解】不管这个类有多少个对象，静态数据成员都只有一份拷贝。静态数据成员一般用来保存整个类所有对象共享的信息。

【5】在定义一个类时，哪些部分应放在头文件（.h文件）中，哪些部分应放在实现文件（.cpp文件）中？

【解】类的定义放在头文件中，类中成员函数的实现放在实现文件中。

【6】什么情况下类必须定义自己的复制构造函数？

【解】如果类的数据成员中含有指针，而指针指向的是一个动态变量，必须自己定义复制构造函数。或对复制构造有其他特殊的要求也需要定义复制构造函数。

【7】什么样的成员函数应被说明为公有的？什么样的成员函数应被设为私有的？

【解】根据对象行为提取的成员函数应该设为公有的成员函数。公有函数的实现时分解出一些小函数通常被设计为私有的成员函数。

【8】常量数据成员和静态常量数据成员有什么区别？如何初始化常量数据成员？如何初始化静态常量数据成员？

【解】常量的数据成员指得是那些在对象生成时给定了初值，在整个对象的生命周期中，该数据成员的值是不能变的。常量数据成员的值必须在构造函数的初始化列表中进行初始化。静态的常量数据成员是整个类所有对象共享的一个常量。对整个类而言，不管定义了多少个对象，该成员永远只有一份拷贝。静态常量数据成员的值是在定义类时给定。

【9】什么是this指针？为什么要有this指针？

【解】每个对象包含两部分内容：数据成员和成员函数。不同的对象有不同的数据成员值，因此每个对象都拥有一块保存自己数据成员值的空间。但同一类的所有对象的成员函数的实现都是相同的，因此所有的对象共享了一份成员函数的代码。这又带来了另一个问题：成员函数中的涉及到的数据成员到底是哪个对象的数据成员？为此C++让每个成员函数都包含了一个隐含的参数this，该参数是一个指针，指向当前调用该成员函数的对象。成员函数中涉及的数据成员都是this指针指向的对象的数据成员。

【11】复制构造函数的参数为什么一定要用引用传递，而不能用值传递？

【解】值传递的参数在参数传递时有一个构造过程，即用实际参数的值构造形式参数，这个构造过程是由复制构造函数完成的。如果将复制构造函数的参数设计成值传递，会引起复制构造函数的递归调用。

【12】下面哪个类必须定义复制构造函数：

a．包含3个int类型的数据成员的point3类。

b．处理动态二维数组的Matrix类。其中存储二维数组的空间在构造函数中动态分配，在析构函数中释放。

c．教材中定义的有理数类。

d．处理一段文本的word类。所处理的文本存储在一个字符数组中。

【解】 a．不需要 b．需要 c．不需要 d．不需要

【13】静态成员函数不能操作非静态的数据成员，那么非静态的成员函数能否操作静态的数据成员？为什么？

【解】因为静态成员函数没有隐含的this指针，所以不能够操作非静态的数据成员。静态的数据成员是所有对象共享的数据成员，在逻辑上是属于每一个对象的，所以非静态的成员函

数可以操作静态的数据成员。

【14】构造函数为什么要有初始化列表？

【解】构造函数的初始化列表可以将数据成员的构造和赋初值一起完成，提高对象构造的时间性能。除此之外，还有两种情况必须用初始化列表。第一种情况是数据成员中含有一些不能用赋值操作进行赋值的数据成员，例如常量数据成员或对象数据成员，这时必须在初始化列表中调用数据成员所属类型的构造函数来构造它们。第二种情况是在用派生的方法定义一个类时，派生类对象中的基类部分必须在构造函数的初始化列表中调用基类的构造函数完成。

【15】某程序员用本章定义的Goods类编写了一个程序。请分析这段程序有什么问题。

```
void print (Goods obj)
    { cout << obj.weight() << '\t' << obj.totalweight() << endl; }
int main()
    {  Goods g1(30), g2(50), g3(70);
       print(g1);
       cout << Goods::totalweight() << endl;
       return 0;
    }
```

【解】这段程序将会使得totalweight中保存的总重量有误。在调用print函数时，会调用Goods类的复制构造函数。由于Goods没有定义复制构造函数，C++调用的是缺省的复制构造函数，即将实际参数的weight赋给形式参数的weight，totalweight的值不变。但print函数结束时，会析构形式参数obj，析构函数将会从totalweight中减去obj的weight值。

第10章 程序设计题

【1】编一个程序，验证对象的空间中不包含静态的数据成员。

【解】定义一个简单的类Test。Test类有两个数据成员，一个是普通的字符类型的数据成员，另一个是静态的整型数据成员。在main函数中直接用sizeof运算获得Test类对象所占的字节数输出。程序运行的结果是1，表示Test类的对象只占1个字节，即数据成员id占用的空间。

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

class Test {
    char id;
    static int global;
public:
    Test(char t) {id = t;}
};

int Test::global = 0;
int main()
{
    cout << sizeof(Test) << endl;
    return 0;
}
```

【2】创建一个处理任意大的正整数的类 LongLongInt，用一个动态的字符数组存放任意长度的正整数。数组的每个元素存放整型数的一位。例如，123 被表示为“321”。注意，数字是逆序存放，这样可以使得整型数的操作比较容易实现。提供的成员函数有构造函数(根据一个由数字组成的字符串创建一个 LongLongInt 类的对象)、输出函数、加法函数、把一个 LongLongInt 类的对象赋给另一个对象的赋值函数。为了比较 LongLongInt 对象，提供了等于比较、大于比较、大于等于比较。

【解】LongLongInt类将数字以字符串形式存储，所以有一个指向字符的指针的数据成员。

LongLongInt类应该有构造函数、输出函数、加法函数、赋值函数和比较函数，这些都是LongLongInt类的公有成员函数。由于LongLongInt类含有一个指针的数据成员，还必须增加复制构造函数和析构函数。

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

class LongLongInt {
private:
    char *num;
public:
    LongLongInt(const char *n = "");
    LongLongInt(const LongLongInt &);
   ~LongLongInt() {delete num; }
    void print() const;
    void add(const LongLongInt &, const LongLongInt &);
    void assign(const LongLongInt &);
    bool equal(const LongLongInt &) const;
    bool greater(const LongLongInt &) const;
    bool greaterOrEqual(const LongLongInt &) const;
};
```

构造函数有一个字符串类型参数n，表示所要处理的数字。构造函数根据数字的长度为数据成员num申请空间，将n中的数字存储到num中。复制构造函数根据参数other的num的长度为当前正在构造的对象的数据成员num申请空间，并将other的num的内容复制到当前正在构造的对象的num中。

```
LongLongInt::LongLongInt(const char *n )
{
    int len = strlen(n);

    num = new char[len+1];
    for (int i = 0; i < len; ++i) num[len - i - 1] = n[i];
    num[len] = '\0';
}

LongLongInt::LongLongInt(const LongLongInt &other)
{
    num = new char[strlen(other.num) + 1];
    strcpy(num, other.num);
}
```

输出函数从字符串的最后一个数字开始往前，依次输出每一个字符。

```
void LongLongInt::print() const
{
    for (int i = strlen(num); i >= 0; --i) cout << num[i];
    if (strlen(num) == 0) cout << 0;
}
```

加法函数模拟人工的加法过程，从个位开始将对应位及上一位的进位相加。

```
void LongLongInt::add(const LongLongInt &n1, const LongLongInt &n2)
{
    int len1 = strlen(n1.num), len2 = strlen(n2.num);
    int minLen = (len1 > len2 ? len2 : len1) ;
    int len = (len1 > len2 ? len1 : len2) + 1;
    int carry = 0, result;                          // carry: 进位

    num = new char[len + 1];
    for (int i = 0; i < minLen; ++i) {              // n1和n2都有数字
```

```
        result = n1.num[i] - '0' + n2.num[i] - '0' + carry;
        num[i] = result % 10 + '0';
        carry = result / 10;
    }

    while (i < len1) {                              // n2已结束
        result = n1.num[i] - '0' + carry;
        num[i] = result % 10 + '0';
        carry = result / 10;
        ++i;
    }

    while (i < len2) {                               // n1已结束
        result = n2.num[i] - '0' + carry;
        num[i] = result % 10 + '0';
        carry = result / 10;
        ++i;
    }


    if (carry != 0) num[i++] = carry +'0';             // 处理最高位的进位
    num[i] = '\0';

    if (i != len)  {                                // 最高位无进位处理
        char *tmp = num;
        num = new char[len];
        strcpy(num, tmp);
        delete tmp;
    }
}
```

赋值函数首先检查right是否就是当前对象，如果是则不用赋值，直接返回。否则先释放被复制对象的空间，按照right中保存的数字重新申请适当大小的空间，将right中的数字复制到当前对象。

```
void LongLongInt::assign(const LongLongInt &right)
{
    if (this == &right) return;

    delete num;
    num = new char[strlen(right.num) + 1];
    strcpy(num, right.num);
}
```

两个数相等，则两个对象的num中保存的字符串相同，否则就是不相等。据此可以设计出equal函数。大于比较可以先比较两个字符串的长度。较长的数一定较大。如果长度相等，则从高位到低位比较对应位，直到能分出大小。大于等于与大于比较基本类似。

```
bool LongLongInt::equal(const LongLongInt &n) const
{   return strcmp(num, n.num) == 0;  }

bool LongLongInt::greater(const LongLongInt &n) const
{
    int len1 = strlen(num), len2 = strlen(n.num);

    if (len1 > len2) return true;
    else if (len1 < len2) return false;

    for (int i = len1 - 1; i >= 0; --i) {
        if (num[i] > n.num[i]) return true;
```

```
        else if (num[i] < n.num[i]) return false;
    }

    return false;
}

bool LongLongInt::greaterOrEqual(const LongLongInt &n) const
{
    int len1 = strlen(num), len2 = strlen(n.num);

    if (len1 > len2) return true;
    else if (len1 < len2) return false;

    for (int i = len1 - 1; i >= 0; --i) {
        if (num[i] > n.num[i]) return true;
        else if (num[i] < n.num[i]) return false;
    }

    return true;
}
```

【3】用单链表实现程序设计题2中的LongLongInt类。

【解】单链表的每个节点保存一个数字，以逆序存放。即第一个节点是个位数。采用带头结点的单链表。LongLongInt类有构造函数、输出函数、加法函数、赋值函数和比较函数，这些都是LongLongInt类的公有成员函数。由于LongLongInt类含有一个指针的数据成员，还必须增加复制构造函数和析构函数。在公有函数的实现过程中，进一步分解出了4个工具函数move、copy、print和compare。

```
class LongLongInt {
private:
    struct Node{
        char data;
        Node *next;

        Node(char c = ' ', Node *n = NULL) : data(c), next(n) {}
    };
    Node *num;

    void remove();
    void copy(const LongLongInt &other);
    void print(Node *p) const;
    int compare(const Node *n1, const Node *n2) const;
public:
    LongLongInt(const char *n = "");
    LongLongInt(const LongLongInt &);
    void print() const;
    void add(const LongLongInt &, const LongLongInt &);
    void assign(const LongLongInt &);
    bool equal(const LongLongInt &) const;
    bool greater(const LongLongInt &) const;
    bool greaterOrEqual(const LongLongInt &) const;
    ~LongLongInt();
};
```

构造函数根据一个字符串构造一个单链表。复制构造函数就是复制一个单链表。

```
LongLongInt::LongLongInt(const char *n )
{
    num = new Node;
```

```
    while (*n) {
        num->next = new Node(*n, num->next);
        ++n;
    }
}

LongLongInt::LongLongInt(const LongLongInt &other)
{
    num = new Node;
    copy(other);
}

void LongLongInt::copy(const LongLongInt &other)
{
    Node *p = other.num->next, *tail = num;

    while (p != NULL) {
        tail->next = new Node(p->data);
        tail = tail->next;
        p = p-> next;
    }
}
```

输出采用递归实现。函数print可以打印某个以逆序方式存储为单链表的数字，该函数的实现由两个步骤组成。先打印出从第二个结点开始的数字，然后打印第一个结点中的数字。前者可以通过递归调用来实现。

```
void LongLongInt::print() const
{
    if (num->next == NULL) cout << 0;
    else print(num->next);
}

void LongLongInt::print(Node *p) const
{
    if (p != NULL) {
        print(p->next);
        cout << p->data;
    }
}
```

加法函数首先将当前对象的单链表清空，为存放结果做好准备，然后执行加法过程。清空操作设计为一个工具函数remove。

```
void LongLongInt::add(const LongLongInt &n1, const LongLongInt &n2)
{
    Node *num1 = n1.num->next, *num2 = n2.num->next, *tail,  *tmp;
    int carry = 0, result;

    remove();
    tail = num;
    while (num1 != NULL && num2 != NULL) {       // 两个数字对应位都存在
        result = num1->data - '0' + num2->data - '0'  + carry;
        tail->next = new Node(result % 10 +'0' );
        tail = tail->next;
        carry = result / 10;
        num1 = num1->next;
        num2 = num2->next;
    }
```

```
    tmp = ( num1? num1: num2);
    while (tmp != NULL) {                    // 处理尚未结束的运算数
        result = tmp ->data - '0' + carry;
        tail->next = new Node(result % 10 +'0');
        tail = tail->next;
        carry = result / 10;
        tmp = tmp ->next;
    }

    if (carry != 0) tail->next = new Node(carry + '0');   // 最后有进位
}

void LongLongInt::remove()
{
    Node *p = num->next, *q;

    num->next = NULL;
    while (p != NULL) {
        q = p->next;
        delete p;
        p = q;
    }
}
```

赋值函数首先检查right是否就是当前对象，如果不是，先清空当前对象，然后调用copy函数实现赋值。

```
void LongLongInt::assign(const LongLongInt &right)
{
    if (this == &right) return;
    remove();
    copy(right);
}
```

比较采用递归实现。个位数的大小只有在高位数字完全相等时才有意义。如何比较高位数？这正好是原问题的再现。于是可得出compare函数的实现。

```
int LongLongInt::compare(const Node *n1, const Node *n2) const
{
    int result;

    if (n1 != NULL && n2 != NULL) {
        result = compare(n1->next, n2->next);
        if (result != 0) return result;
        if (n1->data > n2->data) return 1;
        if (n1->data < n2->data) return -1;
        return 0;
    }

    if (n1 == NULL && n2 == NULL) return 0;
    if (n1 != NULL) return 1;  else return -1;
}

bool LongLongInt::equal(const LongLongInt &n) const
{   return compare(num->next, n.num->next) == 0;  }

bool LongLongInt::greater(const LongLongInt &n) const
{   return compare(num->next, n.num->next) > 0;   }

bool LongLongInt::greaterOrEqual(const LongLongInt &n) const
{   return compare(num->next, n.num->next) >= 0;   }
```

析构函数先调用remove函数清空单链表，再释放头结点。

```
LongLongInt::~LongLongInt()
{
  remove();
  delete num;
}
```

【4】完善本章提到的SavingAccount类。该类的属性有账号、存款金额和月利率。账号自动生成。第一个生成的对象账号为1，第二个生成的对象账号为2，依次类推。所需的操作有修改利率、每月计算新的存款额（原存款额+本月利息）和显示账户金额。

【解】

```
class savingAccount {
private:
    int no;
    double balance;
    static double rate;
    static int totalNo;

public:
    savingAccount(double deposit);
    void updateMonthly();
    void print() const;
    static void setRate(double);
    static int generateNo();
};
```

构造函数有一个表示存入金额的参数。构造函数首先调用generateNo为帐户生成一个账号存入no，再记录存入的金额。函数updateMonthly将帐户本月所得的利息加入到帐户余额中。函数print输出帐户的账号和余额。函数setRate设置新的月利率。generateNo将totalNo的值加1。

```
int savingAccount::totalNo = 0;
double savingAccount::rate = 0;

savingAccount::savingAccount(double deposit)
{
    no = generateNo();
    balance = deposit;
}

void savingAccount::updateMonthly()
{   balance += balance * rate;   }

void savingAccount::print() const
{   cout << no << "\t" << balance << endl;   }

void savingAccount::setRate(double newRate)
{   rate = newRate;   }

int savingAccount::generateNo()
{   return ++totalNo;   }
```

【5】试定义一个string类，用以处理字符串。它至少具有两个数据成员：字符串的内容和长度。提供的操作有显示字符串、求字符串长度、在原字符串后添加一个字符串等（不能用cstring库）。

【解】

```
class String {
private:
```

```
    int len;
    char *data;

public:
    String(const char *s = "");
    String(const String &s);
    void Copy(const String &src);
    void Cat(const String &src);
   int Len() const { return len; };
   void Print() const { cout << data; }
    ~String() { delete data; }
};

String::String(const char *s)
{
    for (len = 0; s[len] != 0; ++len);

    data = new char[len + 1];
    for (int i = 0; i < len; ++i) data[i] = s[i];
    data[len] = '\0';
}

String::String(const String &s)
{
    len = s.len;

    data = new char[len + 1];
    for (int i = 0; i < len; ++i) data[i] = s.data[i];
    data[len] = '\0';
}

void String::Copy(const String &s)
{
    delete data;

    len = s.len;

    data = new char[len + 1];
    for (int i = 0; i < len; ++i) data[i] = s.data[i];
    data[len] = '\0';

}

void String::Cat(const String &s)
{
    char *tmp = data;
    int i;

    data = new char[len + s.len + 1];

    for (i = 0; i < len; ++i) data[i] = tmp[i];
    for (i = 0; i < s.len; ++i) data[len + i] = s.data[i];
    len += s.len;
    data[len] = '\0';
}
```

【6】为学校的教师提供一个工具，使教师可以管理自己所教班级的信息。教师所需了解和处理的信息包括课程名、上课时间、上课地点、学生名单、学生人数、期中考试成绩、期末

考试成绩和平时的课堂练习成绩。每位教师可自行规定课堂练习次数的上限。考试结束后，该工具可为教师提供成绩分析，统计最高分、最低分、平均分及优、良、中、差的人数。

【解】为了逻辑上更加明确，可以将学生信息组成一个结构体Student，并作为Teacher类的私有内嵌类。选修本课程的学生信息可以用一个Student类型的动态数组存储。教师要有一个能够提供成绩分析的工具analysis，还需要一个输入本班的学生名单的函数inputStudent和输入每次考试的成绩的函数inputScore。

```
class Teacher {
private:
    char className[30];
    char time[20];
    char place[20];
    int numOfStudent;          // 学生人数
    int totalQuiz;             // 总的小测验次数
    int curQuiz;               // 已进行的小测验次数

    struct Student {
        char name[10];
        int scoreMid;
        int scoreFinal;
        int *scoreQuiz;          // 保存每次测验成绩的数组
    };
    Student *sInfo;

public:
    enum Type {MID, FINAL, QUIZ };

    Teacher(char *cName, char *cTime, char *cPlace, int noS, int noQuiz);
    ~Teacher();
    void inputStudent();
    void inputScore(Type);
    void analysis(Type);
};

Teacher::Teacher(char *cName, char *cTime, char *cPlace, int noS, int noQuiz)
  :numOfStudent(noS), totalQuiz(noQuiz), curQuiz(0)
{
    strcpy(className, cName);
    strcpy(time, cTime);
    strcpy(place, cPlace);
    sInfo = new Student[noS];
    for (int i = 0; i < noS; ++i) sInfo[i].scoreQuiz = new int[noQuiz];
}

Teacher::~Teacher()
{
    for (int i = 0; i < numOfStudent; ++i) delete [] sInfo[i].scoreQuiz;
    delete [] sInfo;
}

void Teacher::inputStudent()
{
    for (int i = 0; i < numOfStudent; ++i) {
        cout << "请输入第" << i << "个学生姓名：";
        cin >> sInfo[i].name;
    }
}
```

```
void Teacher::inputScore(Type scoretype)
{
    int i;

    switch (scoretype) {
    case MID:
        for (i = 0; i < numOfStudent; ++i) {
            cout << "请输入" << sInfo[i].name << "的期中成绩：";
            cin >> sInfo[i].scoreMid;
        }
        break;
    case FINAL:
        for (i = 0; i < numOfStudent; ++i) {
            cout << "请输入" << sInfo[i].name << "的期末成绩：";
            cin >> sInfo[i].scoreFinal;
        }
        break;
    case QUIZ:
        if (curQuiz > totalQuiz) {
            cout << "所有测验都已完成！无法输入。" << endl;
            return;
        }
        for (i = 0; i < numOfStudent; ++i) {
            cout << "请输入" << sInfo[i].name << "的测验成绩：";
            cin >> sInfo[i].scoreQuiz[curQuiz];
        }
        ++curQuiz;
    }
}

void Teacher::analysis(Type scoretype)
{
    int i;
    int high = 0, low = 100, total = 0, execellent = 0, good = 0, pass = 0, fail = 0;

    switch (scoretype) {
    case MID:
        for (i = 0; i < numOfStudent; ++i) {
            total += sInfo[i].scoreMid;
            if (sInfo[i].scoreMid > high) high = sInfo[i].scoreMid;
            if (sInfo[i].scoreMid < low) low = sInfo[i].scoreMid;
            if (sInfo[i].scoreMid >= 90) ++ execellent;
            else if (sInfo[i].scoreMid >= 75) ++good;
            else if (sInfo[i].scoreMid >= 60) ++pass;
            else ++fail;
        }
        break;
    case FINAL:
        for (i = 0; i < numOfStudent; ++i) {
            total += sInfo[i].scoreFinal;
            if (sInfo[i].scoreFinal > high) high = sInfo[i].scoreFinal;
            if (sInfo[i].scoreFinal < low) low = sInfo[i].scoreFinal;
            if (sInfo[i].scoreFinal >= 90) ++execellent;
            else if (sInfo[i].scoreFinal >= 75) ++good;
            else if (sInfo[i].scoreFinal >= 60) ++pass;
            else ++fail;
```

```
            }
            break;
        case QUIZ:
            int no;
            cout << "要统计第几次测验成绩？";
            cin >> no;
            if (no >= curQuiz) {
                cout << "还没有进行测验，无法统计。" << endl;
                return ;
            }
            for (i = 0; i < numOfStudent; ++i) {
                total += sInfo[i].scoreQuiz[no];
                if (sInfo[i].scoreQuiz[no] > high) high = sInfo[i].scoreQuiz[no];
                if (sInfo[i].scoreQuiz[no] < low) low = sInfo[i].scoreQuiz[no];
                if (sInfo[i].scoreQuiz[no] >= 90) ++execellent;
                else if (sInfo[i].scoreQuiz[no] >= 75) ++good;
                else if (sInfo[i].scoreQuiz[no] >= 60) ++pass;
                else ++fail;
            }
        }
        cout << "本次考试的最高分是 " << high << endl;
        cout << "本次考试的最低分是 " << low << endl;
        cout << "本次考试的平均分是 " << total / numOfStudent << endl;
        cout << "共有 " << execellent << "个同学得优，" << good << "个同学得良，" << pass
             << "个同学及格，" << fail << "个同学不及格" << endl;
}
```

【7】将第 8 章的第一题改成用类实现。

【解】

```
class complex {
private:
    double real;
    double imag;

public:
    complex(double r = 0, double i = 0): real(r), imag(i) {}
    void input();
    void output() const;
    void add(const complex &d1, const complex &d2);
    void multi(const complex &d1, const complex &d2);
};

void complex::input()
{
    cout << "请输入实部：";
    cin >> real;
    cout << "请输入虚部：";
    cin >> imag;
}

void complex::output() const
{   cout << "(" << real << " + " << imag << "i )";  }

void complex::add(const complex &d1, const complex &d2)
{
    real = d1.real + d2.real;
    imag = d1.imag + d2.imag;
}
```

```
void complex::multi(const complex &d1, const complex &d2)
{
    real = d1.real * d2.real - d1.imag * d2.imag;
    imag = d1.imag * d2.real + d1.real * d2.imag;
}
```

【8】设计并实现一个解决约瑟夫环问题的类Joseph。当需要解决一个n个人的约瑟夫环问题，可以构建一个对象Joseph obj(n)，然后调用obj.simulate()输出删除过程。

【解】约瑟夫环最合适的存储方案是单循环链表，存储约瑟夫环只需要一个指向存储第0个人的结点的指针。约瑟夫环只有一个行为就是输出最后一个人是谁，这个过程由成员函数simulate完成。由于约瑟夫环是用一个单循环链表保存，所以应该有构造函数和析构函数。

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

class  Joseph{
private:
    struct node {
       int data;
       node  *next;

       node(int d, node *n = NULL):data(d),next(n) {}
    };
    node *head;

public:
    Joseph(int n);
    void simulate();
};

Joseph::Joseph(int n)
{
    node *tail;

    //建立链表
   head = tail = new node(0);             //创建第一个结点，head指向表头，p指向表尾
   for (int i = 1; i < n; ++i) {
       tail->next = new node(i);
       tail = tail->next;
   }
   tail->next = head;                       // 头尾相连
}

Joseph::~Joseph()
{
    node *p;

    if (head == NULL) return;
    while (head->next != head) {
        p = head->next;
        head->next = p->next;
        delete p;
    }
    delete head;
}
```

simulate函数的实现过程：如果head指针指向报数为1的人，那么head往后移一下就是指向报数数为2的人，head后面的人即为要被删除的人，删除head后面的人。重复这个过

程，直到循环链表中只剩下一个结点，这就是最后剩下的人。

```
void Joseph::simulate()
{
    node *p;    //被删结点
    while (head->next != head) {                                    //表中元素多于一个
       head = head->next;  p = head->next;                        //p报数为2， q报数为3
       //删除p
       head->next = p->next;                                       //绕过结点q
       cout << p->data << '\t';                                    //显示被删者的编号
       delete p;                                                   //回收被删者的空间
       head = head->next;                                          //让q指向报1的结点
    }

    // 打印结果
    cout << "\n最后剩下： " <<  head->data << endl;
    delete head;
    head = NULL;
}
```

【9】设计并实现一个英汉词典类Dictionary。类的功能有：添加一个词条、删除一个词条、查找某个单词对应的中文含义。

【解】将词条以字母序的升序保存在一个数组中。每个词条有两个内容：英文单词和中文解释，为此定义了一个保存词条的结构体item。

Dictionary类必须有插入、删除、查找和输出4个公有的成员函数。还必须有构造和析构函数，最好还应该有一个复制构造函数。在插入时可能遇到数组满的情况，为此定义了一个私有的成员函数doubleSpace，用于在数组满时扩大数组空间。还定义了一个私有的find函数在表中查找某个词条的正确位置。

```
class dictionary {
private:
    struct item {                          // 词条类
        char english[15];
        char chinese[20];
    };
    int size;                              // 数组规模
    int curLen;                            // 词条数
    item *data;

    int find(int low, int high, char *word) const;    // 找出word在词典中的位置
    void doubleSpace();

public:
    dictionary(int s):curLen(0), size(s)
    {       data = new item[size];      }
    ~dictionary() { delete [] data; }
    void insert(char *e, char *s);
    void remove(char *e);
    char *find(char *e) const;
    void print() const
    {
        for (int i = 0; i < curLen; ++i)
            cout << data[i].english << '\t' << data[i].chinese << endl;
    }
};
```

私有的find函数采用了二分查找算法。如果找到了对应的词条，返回该词条的下标。如果没有找到，返回该词条应该插入的位置。

```
char *dictionary::find(char *e) const   // 共有find函数
{
    int pos;

    if (curLen == 0) return NULL;
    pos = find(0, curLen - 1, e);
    if (strcmp(e, data[pos].english) != 0) return NULL;
    return data[pos].chinese;
}

int dictionary::find(char *word) const  // 私有的find函数
{
    int mid, result;
    int low = 0, high = curLen - 1;

    while (low <= high) {
        mid = (low + high ) / 2;
        result = strcmp(word, data[mid].english);
        if (result == 0) return mid;
        if (result > 0) low = mid + 1; else high = mid -1;
    }
    return (low + high) / 2 + 1;
}
```

插入函数insert先调用私有的find函数找出该词条的位置，然后执行插入过程。如果数组已满，则调用doubleSpace扩大数组空间。

```
void dictionary::insert(char *e, char *s)
{
    int pos;
    if (curLen == 0) {                          // 处理空表
        strcpy(data[0].english, e);
        strcpy(data[0].chinese, s);
        ++curLen;
        return;
    }
    pos = find(0, curLen - 1, e);
    if (strcmp(e, data[pos].english) == 0) {
        cout << "词条已存在！" << endl;
        return;
    }
    if (curLen == size) doubleSpace();
    for (int i = curLen; i > pos; --i) data[i] = data[i - 1];
    strcpy(data[pos].english, e);
    strcpy(data[pos].chinese, s);
    ++curLen;
}

void dictionary::doubleSpace()
{
    item *tmp = data;

    size *= 2;
    data = new item[size];
    for (int i = 0; i < curLen; ++i) data[i] = tmp[i];
    delete tmp;
}
```

删除函数首先通过私有的find函数找到被删单词的位置，将后面的所有单词向前移一个位

置，词条数减1。

```
void dictionary::remove(char *e)
{
    int pos;

    if (curLen == 0) {
        cout << "词条不存在！" << endl;
        return;
    }

    pos = find(0, curLen - 1, e);
    if (strcmp(e, data[pos].english) != 0) {
        cout << "词条不存在！" << endl;
        return;
    }
    for (int i = pos; i < curLen - 1; ++i) data[i] = data[i + 1];
    --curLen;
}
```

【10】在本章实现的有理数类中增加减法和除法运算。

【解】

```
void Rational::sub(const Rational &r1, const Rational &r2)
{   num = r1.num * r2.den - r2.num * r1.den;
    den = r1.den * r2.den;
    ReductFraction();
}

void Rational::div(const Rational &r1, const Rational &r2)
{   num = r1.num * r2.den;
    den = r1.den * r2.num;
    ReductFraction();
}
```

【11】有理数是一类特殊的实型数。在有理数类中增加一个将有理数类对象转换成一个double类型的数据的功能。

【解】

```
double Rational::getDouble() const
{  return double(num) / den;  }
```

【12】设计并实现一个有序表类，保存一组正整数。提供的功能有：插入一个正整数，删除一个正整数，输出表中第n小的数，按序输出表中的所有数据。

【解】保存一个动态有序表需要一个动态数组。有序表类必须提供插入、删除、查找和输出函数，由于使用了动态数组，还需要有构造和析构函数。有序表的插入和删除时都需要知道插入或删除的位置，为此提供了一个私有的find函数完成此任务。在设计私有的find函数时，发现私有的find函数和公有的find函数有同样的原型！为了解决这个问题，我们在私有的find函数中增加了一个无意义的整型参数nil。有序表插入时可能遇到表满的情况，此时需要扩大数组空间，为此又设计了一个私有的doubleSpace函数。

```
class List {
private:
    int size;
    int curLen;
    int *data;

    int find(int key,  int nil) const;
    void doubleSpace();
```

```
public:
    List(int s):curLen(0), size(s) {   data = new int[size];   }
    ~List() { delete [] data; }
    void insert(int key);
    void remove(int key);
    int find(int n) const
    {  if (n > curLen || n <= 0) return -1; else return data[n-1];   }
    void print() const
    {
        for (int i = 0; i < curLen; ++i) cout << data[i] << '\t' ;
        cout << endl;
    }
};

void List::remove(int key)
{
    int pos;

    if (curLen == 0) {                // 空表
        cout << "数值不存在！" << endl;
        return;
    }

    pos = find(key, 0);               // 查找被删元素的位置
    if (data[pos] != key) {
        cout << "数值不存在！" << endl;
        return;
    }
    for (int i = pos; i < curLen - 1; ++i) data[i] = data[i + 1];
    --curLen;
}

int List::find(int key, int nil) const
{
    int mid, low = 0, high = curLen - 1;

    while (low <= high) {
        mid = (low + high ) / 2;
        if (data[mid] == key) return mid;
        if (key > data[mid]) low = mid + 1; else high = mid - 1;
    }
    return (low + high) / 2 + 1;
}

void List::insert(int key)
{
    int pos;

    if (curLen == 0) {              // 处理空表
        data[curLen++] = key;
        return;
    }
    pos = find(key, 0);          // 查找插入元素的位置
    if (key == data[pos]) {
        cout << "数值已存在！" << endl;
        return;
    }
```

```
    if (curLen == size) doubleSpace();
    for (int i = curLen; i > pos; --i) data[i] = data[i - 1];
    data[pos] = key;
    ++curLen;
}

void List::doubleSpace()
{
    int *tmp = data;

    size *= 2;
    data = new int[size];
    for (int i = 0; i < curLen; ++i) data[i] = tmp[i];
    delete tmp;
}
```

【13】将第9章中的随机函数库改用类实现。如果类名为Random，当需要用到随机数时，可定义一个对象：Random r；如果要产生一个a到b之间的随机整数，可以调用r.RandomInt(a, b)。如果要产生a到b之间的随机实数，可调用r.RandomDouble(a, b)。

【解】

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

class Random {
public:
    Random() { srand(time(NULL)); }
   int RandomInt(int low, int high)
    {      return (low + (high - low + 1) * rand() / (RAND_MAX + 1)); }
   double RandomDouble(double low, double high)
    {
        double d = (double)rand() / (RAND_MAX + 1);
        return low + (high - low ) * d;
    }
};
```

【14】将第 9 章的程序设计题 2 改用类实现。

【解】Label类的对象需要保存2个信息：标号和目前的序号。函数SetLabel设置label的值，函数SetInitNumber设置起始的序号。GetNextLabel函数首先将no转换成字符串，然后和label拼接在一起返回。

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

class Label {
private:
    char label[10] ;
    int no;
    char returnLabel[20];
public:
    Label(char *lb = "label"):no(1) { strcpy(label, lb); }
   void SetLabel(const char *s) { strcpy(label, s); }
   void SetInitNumber(int num) {   no = num; }
   char *GetNextLabel();
```

```
};

char *Label::GetNextLabel()
{
    char noChar[10];
    int tmpNo = no, i, j;

    // 将no转换成字符串
    if (no == 0) {
        noChar[0] = '0';
        noChar[1] = '\0';
    }
    else {
        for (i = 9; tmpNo != 0; --i, tmpNo /= 10)  noChar[i] = tmpNo % 10 + '0';
        for (j = 0, ++i; i < 10; ++i, ++j) noChar[j] = noChar[i];
        noChar[j] = '\0';
    }

    // 组成返回字符串
    strcpy(returnLabel, label);
    strcat(returnLabel, noChar);
    ++no;

    return returnLabel;
}
```

第11章 简答题

【1】重载后的运算符的优先级和结合性与用于内置类型时有何区别?

【解】与内置类型完全一样。

【3】如何区分++和--的前缀用法和后缀用法的重载函数?

【解】后缀++的重载函数中增加了第二个参数：一个整型的参数。这个参数只是为了区分两个函数，它的值没有任何意义。当程序中出现前缀++时，编译器调用不带整型参数的operator++函数。如果是后缀的++，则调用带整型参数的operator++函数。--重载也是如此。

【4】为什么要使用运算符重载?

【解】运算符重载可以使内置的运算符用于类的对象，使C++的功能得到了扩展。

【5】如果类的设计者在定义一个类时没有定义任何成员函数，那么这个类有几个成员函数?

【解】四个函数：默认的构造函数、默认的复制构造函数、析构函数和赋值运算符重载函数。

【6】如何实现类类型对象到内置类型对象的转换？如何实现内置类型到类类型的转换？

【解】内置类型到类类型的转换是通过构造函数。如果类有一个只带一个参数的构造函数，则可以实现参数类型到类类型的隐式转换。类类型到内置类型或其他类类型的转换必须通过类型转换函数。

【7】. 如何禁止内置类型到类类型的自动转换?

【解】在构造函数原型前加一个保留词explicit。

【8】下标运算符为什么要重载成成员函数？下标运算符重载函数为什么要用引用返回？

【解】下标运算符的第一个运算数是数组名，即当前类的对象。将下标运算符重载成成员函数时，编译器会将程序中诸如a[i]的下标变量的引用改为a.operator[](i)。如果a不是当前类的对象，编译器就会报错。下标变量是左值，所以必须用引用返回。

【13】如何禁止同类对象之间的相互赋值?

【解】将赋值运算符重载函数设为类的私有成员函数就可以了。

第11章 程序设计题

【1】定义一个时间类Time，通过运算符重载实现时间的比较（关系运算）、时间增加/减少若干秒（+=和-=）、时间增加/减少1秒（++和--）、计算两个时间相差的秒数（-）以及输出时间对象的值（时-分-秒）。

【解】保存时间可以用秒保存，Time类要一个保存秒数的数据成员，需要的操作有比较、减法、++、--、+=、-=和输出操作。比较、减法和输出被重载成全局函数，其他的被重载成成员函数。

```
class Time {
    friend int operator-(const Time &t1, const Time &t2) { return t1.second - t2.second; }
    friend ostream &operator<<(ostream &os, const Time &t);
    friend bool operator>(const Time &t1, const Time &t2) { return t1.second > t2.second; }
    friend bool operator>=(const Time &t1, const Time &t2) { return t1.second >= t2.second; }
    friend bool operator==(const Time &t1, const Time &t2) { return t1.second == t2.second; }
    friend bool operator!=(const Time &t1, const Time &t2) { return t1.second != t2.second; }
    friend bool operator<(const Time &t1, const Time &t2) { return t1.second < t2.second; }
    friend bool operator<=(const Time &t1, const Time &t2) { return t1.second <= t2.second; }

    int second;
public:
    Time(int tt = 0, int mm = 0, int ss = 0)    {   second = ss + mm * 60 + tt * 3600;   }
    Time &operator++() {++second; return *this; }            // 前缀++
    Time operator++(int x) {                                 // 后缀++
        Time tmp = *this;
        ++second;
        return tmp;
    }
    Time &operator--() {--second; return *this; }            // 前缀--
    Time operator--(int x) {                                 // 后缀--
        Time tmp = *this;
        --second;
        return tmp;
    }
    Time &operator+=(const Time &other)      { second += other.second;   return *this; }
    Time &operator-=(const Time &other)      { second -= other.second; return *this; }
};

ostream &operator<<(ostream &os, const Time &t)
{
    int tt, mm, ss;

    tt = t.second / 3600;
    mm = t.second % 3600 / 60;
    ss = t.second % 60;

    os << tt << '-' << mm << '-' << ss;
    return os;
}
```

【2】用运算符重载完善第10章程序设计题第2题中的LongLongInt类，并增加++和-操作。

【解】

```
class LongLongInt {
    friend LongLongInt operator+(const LongLongInt &, const LongLongInt &);
    friend ostream &operator<<(ostream &, const LongLongInt &);
    friend bool operator==(const LongLongInt &, const LongLongInt &);
    friend bool operator !=(const LongLongInt &, const LongLongInt &) ;
    friend bool operator>(const LongLongInt &, const LongLongInt &);
```

```
    friend bool operator>=(const LongLongInt &, const LongLongInt &) ;
    friend bool operator<(const LongLongInt &, const LongLongInt &) ;
    friend bool operator<=(const LongLongInt &, const LongLongInt &);
    friend LongLongInt operator-(const LongLongInt &, const LongLongInt &);
private:
    char *num;
public:
    LongLongInt(const char *n = "");
    LongLongInt(const LongLongInt &);
    LongLongInt &operator=(const LongLongInt &);
     LongLongInt &operator++();
     LongLongInt operator++(int);
};

LongLongInt &LongLongInt::operator++()
{    return *this = *this + "1";   }

LongLongInt LongLongInt::operator++(int t)
{
    LongLongInt returnObj = *this;

    *this = *this + "1";
    return returnObj;
}
```

其他函数实现与第10章中的类似。

【3】定义一个保存和处理十维向量空间中的向量的类型，能实现向量的输入/输出、两个向量的加以及求两个向量点积的操作。

【解】保存一个向量可以用一个实型数组。向量类应该有输入重载、输出重载、加法重载和乘法重载。每个类一般都要有构造函数。

```
class Vector {
    friend ostream &operator<<(ostream &, const Vector &);
    friend istream &operator>>(istream &, Vector &);
    friend Vector operator+(const Vector &, const Vector &) ;
    friend double operator*(const Vector &, const Vector &) ;
private:
    double data[10];

public:
    Vector(double *dat = NULL);
};

Vector::Vector(double *dat)
{
   if (dat != NULL)
      for (int i = 0; i < 10; ++i) data[i] = dat[i];
   else for (int i = 0; i < 10; ++i) data[i] = 0;
}

ostream &operator<<(ostream &os, const Vector &obj)
{
    os << '(' << obj.data[0];
    for (int i = 1; i < 10; ++i) os << ',' << obj.data[i];
    os << ')';
    return os;
}
```

```
istream &operator>>(istream &is, Vector &obj)
{
    for (int i = 0; i < 10; ++i) is >> obj.data[i];
    return is;
}

Vector operator+(const Vector &v1, const Vector &v2)
{
    Vector result(v1);

    for (int i = 0; i < 10; ++i) result.data[i] += v2.data[i];
    return result;
}

double operator*(const Vector &v1, const Vector &v2)
{
    double result = 0;

    for (int i = 0; i < 10; ++i) result += v1.data[i] * v2.data[i];

    return result;
}
```

【4】实现一个处理字符串的类String。它用一个动态的字符数组保存一个字符串。实现的功能有：字符串连接（+和+=），字符串赋值（=），字符串的比较（>，>=，< ，<=，!=，= =），取字符串的一个子串，访问字符串中的某一个字符，字符串的输入输出（>>和<<）。

【解】String类用动态数组保存字符串，所以必须有一个指向字符的指针的数据成员。在字符串操作中经常会用到字符串长度，为避免每次需要长度信息时再去计算，我们增加了一个整型的数据成员len记录字符串的长度。String类必须提供字符串连接（+和+=），字符串赋值（=），字符串的比较（>，>=，< ，<=，!=，= =），取字符串的一个子串，访问字符串中的某一个字符，字符串的输入输出（>>和<<）。

```
class String {
    friend String operator+(const String &s1, const String &s2);
    friend ostream &operator<<(ostream &os, const String &obj);
    friend istream &operator>>(istream &is, String &obj);
    friend bool operator>(const String &s1, const String &s2);
    friend bool operator>=(const String &s1, const String &s2);
    friend bool operator==(const String &s1, const String &s2);
    friend bool operator!=(const String &s1, const String &s2);
    friend bool operator<(const String &s1, const String &s2);
    friend bool operator<=(const String &s1, const String &s2);
private:
    int len;
    char *data;

public:
    String(const char *s = "");
    String(const String &other);
    String &operator+=(const String &other);
    String &operator=(const String &other);
    String operator()(int start, int end);
    char &operator[](int index) { return data[index];}
    ~String() { delete data; }
};
```

```
String::String(const char *s)
{
    for (len = 0; s[len] != 0; ++len);

    data = new char[len + 1];
    for (int i = 0; i < len; ++i) data[i] = s[i];
    data[len] = '\0';
}

String::String(const String &s)
{
    len = s.len;

    data = new char[len + 1];
    for (int i = 0; i < len; ++i) data[i] = s.data[i];
    data[len] = '\0';
}

String &String::operator=(const String &other)
{
    if (this == &other) return *this;

    delete data;
    len = other.len;
    data = new char[len + 1];
    for (int i = 0; i < len; ++i) data[i] = other.data[i];
    data[len] = '\0';
    return *this;
}

bool operator>(const String &s1, const String &s2)
{
    for (int i = 0; i < s1.len; ++i) {
        if (s1.data[i] > s2.data[i]) return true;
        if (s1.data[i] < s2.data[i]) return false;
    }
    return false;
}

bool operator>=(const String &s1, const String &s2)
{
    for (int i = 0; i < s1.len; ++i) {
        if (s1.data[i] > s2.data[i]) return true;
        if (s1.data[i] < s2.data[i]) return false;
    }
    return true;
}

bool operator==(const String &s1, const String &s2)
{
    if (s1.len != s2.len) return false;
    for (int i = 0; i < s1.len; ++i)
        if (s1.data[i] != s2.data[i]) return false;

    return true;
}

bool operator!=(const String &s1, const String &s2)
```

```
{    return !(s1 == s2);   }

bool operator<(const String &s1, const String &s2)
{    return !(s1 >= s2);    }

bool operator<=(const String &s1, const String &s2)
{    return !(s1 > s2);   }
```

字符串的连接用+和+=重载来实现。

```
String &String::operator+=(const String &other)
{
    char *tmp = data;
    int i;

    data = new char[len + other.len + 1];

    for (i = 0; i < len; ++i) data[i] = tmp[i];
    for (i = 0; i < other.len; ++i) data[len + i] = other.data[i];
    len += other.len;
    data[len] = '\0';
    return *this;
}

String operator+(const String &s1, const String &s2)
{
    String tmp(s1);

    tmp += s2;
    return tmp;
}
```

取子串的功能用函数调用运算符重载来实现。

```
String String::operator()(int start, int end)
{
    if (start > end || start < 0 || end >= len) return String("");

    String tmp;

    delete tmp.data;
    tmp.len = end - start + 1;
    tmp.data = new char[len + 1];
    for (int i = 0; i <= end - start; ++i) tmp.data[i] = data[i + start];
    tmp.data[i] = '\0';
    return tmp;
}
```

输出重载的实现比较容易。输出一个String类的对象就是输出它的data值。输入重载有些麻烦，因为我们不知道用户输入的字符串有多长，无法为data申请合适的空间。输入重载采用了一个块状链表作为过渡。

```
ostream &operator<<(ostream &os, const String &obj)
{
    os << obj.data;
    return os;
}
```

```
istream &operator>>(istream &is, String &obj)
{
    struct Node {                          // 块状链表的结点类
        char ch[10];
        Node *next;
    };

    Node *head, *tail, *p;                 // head：块状链表的首指针，tail：块状链表的尾指针
    int len = 0, i;

    head = tail = new Node;
    while ((tail->ch[len % 10] = is.get()) != '\n') {          // 输入一个字符存入尾结点
        ++len;
        if (len % 10 == 0) {                                   // 申请一个新结点
            tail->next = new Node;
            tail = tail->next;
        }
    }

    obj.len = len;
    delete obj.data;
    obj.data = new char[len + 1];
    for (i = 0; i < len; ++i) {                                // 将块状链表的内容复制到data
        obj.data[i] = head->ch[i % 10];
        if (i % 10 == 9) {
            p = head;
            head = head->next;
            delete p;
        }
    }
    delete head;
    obj.data[len] = '\0';
    return is;
}
```

【5】设计一个动态的、安全的二维double型的数组Matrix。可以通过

```
Matrix table(3, 8);
```

定义一个3行8列的二维数组table，通过table(i, j)访问table的第i行第j列的元素。例如：table（i, j）= 5； 或 table（i, j）= table（i, j+1）+ 3；行号和列号从0开始。

【解】二维数组可以看成元素是一维数组的一维数组。如table是一个3行4列的数组，则table可以看成有3个元素：table[0]、table[1]和table[2]。table[i]是一个由4个元素组成的一维数组的名字，因而是一个指向double的指针。而table是指向table[0]的指针，因而是一个指向double的二级指针。

```
class Matrix {
    int row;
    int col;
    double **data;

public:
    Matrix(int r = 1, int c = 1);
    Matrix(const Matrix &other);
    double &operator()(int r, int c);
    ~Matrix();
};
```

构造函数有两个参数：行数和列数。构造函数将这两个参数分别赋给row和col，再根据这两个参数申请一个存储二维的空间。首先申请一个有row个元素的指向double的指针数组data，每个元素指向一行。然后再为data的每个元素申请一行的空间。复制构造函数也是如此。析构函数分两步释放动态数组的空间。首先释放每一行的空间，然后释放指向每一行的指针数组的空间。

```
Matrix::Matrix(int r, int c ):row(r), col(c)
{
    data = new double*[r];            // 指向每一行第一个元素的指针数组
    for (int i = 0; i < r; ++i)       // 保存每一行元素的一维数组
        data[i] = new double[c];
}

Matrix::Matrix(const Matrix &other)
{
    int i, j;

    row = other.row;
    col = other.col;
    data = new double*[row];          // 指向每一行第一个元素的指针数组
    for (i = 0; i < row; ++i)         // 保存每一行元素的一维数组
        data[i] = new double[col];
    for (i = 0; i < row; ++i)         // 复制二维数组的每个元素
        for (j = 0; j < col; ++j) data[i][j] = other.data[i][j];
}

Matrix::~Matrix()
{
    for (int i =0; i < row; ++i) delete [] data[i];
    delete [] data;
}
```

数组第r行第c列的元素存放在data[r][c]中。函数调用运算符重载函数只需要检查一下下标的合法范围，返回data[r][c]。

```
double &Matrix::operator()(int r, int c)
{
    assert( r < row && c < col);
    return data[r][c];
}
```

【6】C++的布尔类型本质上是一个枚举类型。对布尔类型的变量只能执行赋值、比较和逻辑运算。试设计一个更加人性化的布尔类型。它除了支持赋值、比较和逻辑运算外，还可以直接输入输出。如果要输入true为，直接输入true。如果某个布尔类型的变量flag的值为false，直接cout << flag将会输出false。同时还支持到整型的转换，true转换成1，false转换成0。

【解】优化的布尔类型具有的行为有：输入、输出、赋值、比较、逻辑运算和到整型的转换。

```
class boolean {
    friend ostream &operator<<(ostream &os, const boolean &obj);
    friend istream &operator>>(istream &is, boolean &obj);
private:
    bool data;
public:
    boolean(bool d = false) : data(d) {}
    boolean(const char *s) { data = (strcmp(s, "true") == 0 ? true : false);  }
```

```
    operator int() const { return (data ? 1 : 0); }
};

ostream &operator<<(ostream &os, const boolean &obj)
{
    os << (obj.data ? "true" : "false") ;
    return os;
}

istream &operator>>(istream &is, boolean &obj)
{
    char tmp[6];

    is >> tmp;
    obj.data = (strcmp(tmp, "true") == 0 ? true : false);
    return is;
}
```

【7】. 设计一个可以计算任意函数定积分的类integral。当要计算某个函数f的定积分时，可以定义一个integral类的对象，将实现数学函数f的C++函数g作为参数。例如，integral obj(g)；如果要计算函数f在区间[a, b]之间的定积分，可以调用obj(a, b)。定积分的计算采用第5章程序设计题18中介绍的矩形法

【解】integral类的对象需要一个指向函数的指针作为数据成员。对象的行为除了构造外，只有一个求某个区间内的定积分，该行为用函数调用运算符重载来实现。

```
class integral {
    enum { numOfCalc = 1000 };                  // 小矩形的个数
    double (*f)(double);
public:
    integral(double (*g)(double)): f(g) {}
    double operator()(double a, double b);
};

double integral::operator()(double a, double b)
{
    double sum = 0, dlt = (b-a) / numOfCalc;

    for (double h = a + dlt / 2; h < b; h += dlt)  sum += dlt * f(h);

    return sum;
}
```

【8】改写第 10 章的程序设计题 13，用重载函数调用运算符实现 RandomInt 和 RandomDouble。对于 Random 类的对象 r，调用 r(a, b)将得到一个随机数。如果 a、b 是整型数，将产生一个随机整数。如果 a、b 是 double 型的数，将产生一个随机实数。

【解】

```
int Random::operator()(int low, int high)
{       return (low + (high - low + 1) * rand() / (RAND_MAX + 1)); }
double Random::operator()(double low, double high)
{
    double d = (double)rand() / (RAND_MAX + 1);
    return low + (high - low ) * d;
}
```

第12章 简答题

【1】什么是组合？什么是继承？is-a的关系用哪种方式解决？has-a的关系用哪种方法解决？

【解】组合是将某个类的对象作为当前正在定义的类的数据成员，反映的是has-a的关系。继承是在一个类的基础上，通过增加数据成员或成员函数而得到一个功能更强大的类，反映的是is-a的关系。

【2】protected成员有什么样的访问特性？为什么要引入protected的访问特性？

【解】protected成员是允许派生类的成员函数可以访问的私有成员。可以提高派生类成员函数的效率。

【4】. 什么是抽象类？定义抽象类有什么意义？抽象类在使用上有什么限制？

【解】包含有纯虚函数的类称为抽象类。定义抽象类的主要用途是规范从这个抽象类派生的这些类的行为。在使用时，不能定义抽象类的对象，只能定义抽象类的指针。

【5】. 为什么要定义虚析构函数？

【解】将析构函数定义成虚函数可以防止内存泄漏。

【6】. 试说明派生类对象的构造和析构次序。

【解】构造时，先执行基类的构造函数，再执行派生类自己的构造函数。析构时，先执行派生类的析构函数，再执行基类的析构函数。

【7】试说明虚函数和纯虚函数有什么区别。

【解】虚函数有函数体，可以执行，也可以作为实现多态性的一种手段。而纯虚函数没有函数体，是不可以执行的。

【8】基类指针可以指向派生类的对象，为什么派生类的指针不能指向基类对象？

【解】由于派生类对象中包含了一个基类对象，当基类指针指向派生类对象时，通过基类指针可访问派生类中的基类部分。但如果用一个派生类指针指向基类对象，通过派生类指针访问派生类新增加的成员时，将无法找到这些成员。

【9】多态性是如何实现程序的可扩展性？

【解】派生类是一类特殊的基类，例如本科生、硕士生和博士生都是从学生类派生的。由于所有学生都要写论文，所以基类（学生类）中有一个“写论文”的虚函数。但每类学生写论文的要求是不一样的，因此在本科生、硕士生和博士生类中都有一个对应于虚函数的“写论文”函数。当需要向所有学生布置写论文的任务时，可以用一个学生类的指针遍历所有的学生。由于多态性，每类学生执行的是自己类新增的“写论文”函数，按照自己类规定的要求写论文。如果学校决定要招收工程硕士，那么系统中必须有一个“工程硕士”的类。工程硕士也是一类学生，所以也从学生类派生。工程硕士也要写论文，论文要求和其他几类学生不同，所以工程硕士类也要有一个“写论文”的函数。如果建好了工程硕士这个类，向全校学生布置写论文的工作流程还和以前一样。由此可见，当扩展系统时，只需要增加一些新的类，而主程序不变。

【10】如果一个派生类新增加的数据成员中有一个对象成员，试描述派生类的构造过程。

【解】构造派生类对象时，先调用基类的构造函数，再调用对象成员的构造函数，最后执行派生类的构造函数。

【11】写出下列程序的执行结果，并说明该程序有什么问题，应该如何修改程序解决此问题？

```
class CBase {
public:
     CBase(int i)
     { m_data = i;
       cout << "Constructor of CBase. m_data=" << m_data <<endl;
    }
     ~CBase() { cout << "Destructor of CBase. m_data=" << m_data <<endl; }
protected:
    int m_data;
};
```

```
class CDerived: public CBase {
public:
    CDerived(const char *s): CBase(strlen(s))
      { m_data = new char[strlen(s) + 1];
        strcpy(m_data, s);
        cout << "Constructor of CDerived. m_data = " << m_data << endl;
      }
    ~CDerived()
     { delete m_data;
       cout << "Destructor of CDerived. m_data = " << m_data <<endl;
   }
private:
    char  *m_data;
};

int main()
{
    CBase  *p ;
    p = new CDerived ("abcd");
    delete p;

    return 0;
}
```

【解】执行结果：

```
Constructor of CBase. m_data=4
Constructor of CDerived. m_data =abcd
Destructor of CBase. m_data=4
```

该程序会造成内存泄漏。

第12章 程序设计题

【1】. 定义一个Shape类记录二维平面上的任意形状的位置，在Shape类的基础上派生出一个Rectangle类，在Rectangle类的基础上派生出一个Square类，必须保证每个类都有计算面积和周长的功能。

【解】Shape类有两个数据成员x和y，表示形状的x和y的坐标。为了保证每个类都有计算面积和周长的功能，可在Shape类中定义两个纯虚函数area和circum。Rectangle类在Shape类的基础上派生。增加保存矩形的高度和宽度两个数据成员。行为有构造函数以及两个纯虚函数的重定义。正方形是一类高和宽相等的矩形。Square类从Rectangle类派生时不需要增加新的数据成员，也不需要增加新的成员函数。Square类只有一个构造函数。

```
class Shape {
private:
      double x;
      double y;
public:
      Shape(int xx, int yy) : x(xx), y(yy) {}
      virtual double area() = 0;
      virtual double circum () = 0;
};

class Rectangle : public Shape {
private:
      double height;
     double width;
public:
      Rectangle(double xx, double yy, double w, double h) : Shape(xx, yy), height(h), width(w) {}
      double area() {return height * width ; }
```

```
    virtual double circum () { return 2 * (height + width) ; }
};

class Square : public Rectangle {
public:
    Square(double xx, double yy, double s) : Rectangle(xx, yy, s, s) {}
};
```

2．定义一个安全的动态的一维整型数组。所谓安全，就是在数组操作中会检查下标是否越界。所谓动态，就是定义数组时，数组的规模可以是变量或某个表达式的执行结果。在这个类的基础上，派生出一个可指定下标范围的安全的动态数组。

【解】

```
class Array {
private:
    int size;
    int *storage;
public:
    Array(int s = 1) :size(s) {storage = new int[s]; }
    Array(const Array &other);
    ~array() {delete [] storage; }
   int &operator[](int index) {
        assert(index >=0 && index <size);
        return storage [index];
    }
};

Array::Array(const Array &other)
{
    size = other.size;
    storage = new int[size];
    for (int i = 0; i < size; ++i) storage[i] = other.storage[i];
}
```

可指定下标范围的安全的动态数组ArrayAdvanced类从Array类派生。ArrayAdvanced类只需保存数组的下标范围low和high即可。需要一个构造函数。下标运算符重载函数首先检查了下标范围的合法性，然后将下标值映射到0 – high-low+1，返回基类中对应的下标变量。

```
class ArrayAdvanced : public Array {
    int low;
    int high;
public:
    ArrayAdvanced(int l = 0, int h = 0) :Array(h - l + 1), low(l), high(h) {}
    int &operator[](int index) {
        assert(index >= low && index <= high);
        Array &data = *this;
        return data[index - low];
    }
};
```

【3】. 例12.3中给出了一个图书馆系统中的读者类的设计，在教师读者类和学生读者类中，借书信息使用一个数组表示。试修改这些类，将借书信息用一个单链表表示。

【解】用到单链表，就必须定义一个结点类。由于教师读者和学生读者类都要用到结点类，因此可将结点类定义在基类reader中，而且作为保护成员以方便学生读者类和教师读者类的访问。不管是教师读者或学生读者，都必须具备借书、还书和显示所借的所有图书的功能，为此在读者类中还加了3个纯虚函数以规范教师读者和学生读者类的功能。

```
class reader{
    int no;
    char name[10];
    char dept[20];
 protected:
     struct Node {
        int record;
         Node *next;
         Node(int d = 0, Node *n = NULL) : record(d), next(n) {}
     };

public:
    reader(int n, char *nm, char *d)
    {   no = n;
        strcpy(name, nm);
        strcpy(dept, d);
    }
    virtual bool bookBorrow(int bookNo) = 0;       //借书成功，返回true，否则返回false
    virtual bool bookReturn(int bookNo) = 0;      //还书成功，返回true，否则返回false
    virtual  void show() const = 0;
};
```

教师读者类需要两个数据成员：已借书的数量borrowed和指向单链表的头结点的指针head。教师读者类还需要一个整个类共享的常量：最大的借书数量。教师读者类的行为有构造、析构、借书、还书和显示借书清单。

```
class readerTeacher :public reader{
      enum {MAX = 10};                          //最多允许借的数量，是整个类共享的常量
      int borrowed;
      Node *head;
public:
      readerTeacher(int n, char *nm, char *d):reader(n, nm, d)
       {
             borrowed = 0;
              head = new Node;
       }
      bool bookBorrow(int bookNo);              //借书成功，返回true，否则返回false
      bool bookReturn(int bookNo);              //还书成功，返回true，否则返回false
      void show() const;                                          //显示已借书信息
};
bool readerTeacher::bookBorrow(int bookNo)
{
    if (borrowed == MAX) return false;
    head->next = new Node(bookNo, head->next);
    ++borrowed;

    return true;
}

//还书成功，返回true，否则返回false
bool readerTeacher::bookReturn(int bookNo)
{
      for (Node *p = head; p->next != NULL; p = p->next)
             if (p->next->record == bookNo) {
                    Node *q = p->next;
                    p->next = q->next;
                    delete q;
                    --borrowed;
                    return true;
```

```
        }

    return false;
}

//显示已借书信息
void readerTeacher::show() const
{
    for (Node *p = head->next; p != NULL; p = p->next)    cout << p->record << '\t';
    cout << endl;
}
```

【4】在第11章程序设计题第2题实现的LongLongInt类的基础上，创建一个带符号的任意长的整数类型signedInt。该类型支持输入输出、比较操作、加法操作、减法操作、++操作和- -操作。用组合和继承两种方法实现。

【解】组合的方式：有两个数据成员：保存符号的成员sign和保存无符号整数的成员data。支持的操作有：输入输出、比较操作、加法操作、减法操作、++操作和- -操作。尽管LongLongInt类是用一个动态数组保存一个正整数，但这是LongLongInt类内部的事情，作为LongLongInt类的用户不用考虑，因此signedInt类不需要析构函数。

```
class signedInt {
    friend ostream &operator<<(ostream &os, const signedInt &obj);
    friend istream &operator>>(istream &is, signedInt &obj);
    friend signedInt operator+(const signedInt &d1, const signedInt &d2);
    friend signedInt operator-(const signedInt &d1, const signedInt &d2);
    friend bool operator==(const signedInt &d1, const signedInt &d2);
    friend bool operator!=(const signedInt &d1, const signedInt &d2);
    friend bool operator>(const signedInt &d1, const signedInt &d2);
    friend bool operator>=(const signedInt &d1, const signedInt &d2);
    friend bool operator<(const signedInt &d1, const signedInt &d2);
    friend bool operator<=(const signedInt &d1, const signedInt &d2);
private:
    char sign;
    LongLongInt data;

public:
    signedInt(const char *n = "");
    signedInt &operator++();
    signedInt operator++(int t);
    signedInt &operator--();
    signedInt operator--(int t);
};
```

构造函数首先检查参数是否是空字符串。空串表示0，则将sign设为'+'，data设为0。如果为非空串，检查第一个字符。当第一个字符为'+'或'-'时，用第一个字符设置sign,否则设置sign为'+'。用余下的字符串设置data。

```
signedInt::signedInt(const char *n)
{
    if (strcmp(n, "") == 0) {                // 0总是认为是+0
        sign = '+';
        data = LongLongInt("");
    }
    else if (n[0] == '+' || n[0] == '-') {
            sign = n[0];
            data = LongLongInt(++n);
         }
         else {                              // 第一个字符不是'+'或'-'，则认为是正数
               sign = '+';
```

```
                data =LongLongInt(n);
        }
}
```

如果当前对象是正整数，++操作就是将绝对值加1。如果当前对象是一个负整数，++操作就是将绝对值减1。而在LongLongInt类中已经重载了+和-操作，可以直接使用。

```
signedInt &signedInt::operator++()         // 前缀++
{
    if (sign == '+') data = data + "1";
    else {
        data = data - "1";
        if (data == "0") sign = '+';         // 减到0时修改符号
    }
    return *this;
}

signedInt signedInt::operator++(int t)     // 后缀++
{
    signedInt result = *this;
    if (sign == '+') data = data + "1";
    else {
        data = data - "1";
        if (data == "0") sign = '+';         // 减到0时修改符号
    }
    return result;
}

signedInt &signedInt::operator--()        // 前缀--
{
    if (sign == '-') data = data + "1";
    else if (data == "0") {              // --0的情况
           sign = '-';
           data = "1";
         }
         else   data = data - "1";
    return *this;
}

signedInt signedInt::operator--(int t)    // 后缀--
{
    signedInt result = *this;
    if (sign == '-') data = data + "1";
    else if (data == "0") {              // --0的情况
             sign = '-';
             data = "1";
         }
         else  data = data - "1";
    return result;
}
```

输出一个signedInt类的对象就是输出sign和data。由于LongLongInt类重载了输出，所以data可以直接用<<输出。

由于输入时我们并不知道用户输入的数字有多长，无法为这些输入的数字准备存储空间。于是我们采用了块状链表暂存输入的数字，然后根据输入数字的实际长度申请一个字符数组tmp，将块状链表中的数据拷贝到tmp。用tmp设置对象obj。

```
ostream &operator<<(ostream &os, const signedInt &obj)
{
    os << obj.sign << obj.data;
```

```
    return os;
}

istream &operator>>(istream &is, signedInt &obj)
{

    struct Node {
        char ch[10];
        Node *next;
    };

    Node *head, *tail, *p;
    int len = 0, i;                                          // len:输入的整数长度

    head = tail = new Node;
    while ((tail->ch[len % 10] = is.get()) != '\n') {        // 输入到块状链表
        ++len;
        if (len % 10 == 0) {
            tail->next = new Node;
            tail = tail->next;
        }
    }

    char *tmp = new char[len + 1];
    for (i = 0; i < len; ++i) {                              // 将块状链表拷贝到tmp
        tmp[i] = head->ch[i % 10];
        if (i % 10 == 9) {
            p = head;
            head = head->next;
            delete p;
        }
    }
    delete head;
    tmp[len] = '\0';

     obj = signedInt(tmp);
    delete tmp;
    return is;
}
```

由于LongLongInt类重载了+、-和关系运算，signedInt类的可以利用这些函数。

```
signedInt operator+(const signedInt &d1, const signedInt &d2)
{
    signedInt result;
     if (d1.sign == d2.sign) {                   // 运算数符号相同
        result.sign = d1.sign;
        result.data = d1.data + d2.data;
    }
    else if (d1.data == d2.data) {               // 运算数符号相反，绝对值相同
        result.sign = '+';
        result.data = "";
    }
    else if (d1.data > d2.data) {                // 运算数符号相反，d1的绝对值大于d2的绝对值
        result.sign = d1.sign;
        result.data = d1.data - d2.data;
    }
    else {                                       // 运算数符号相反，d1的绝对值小于d2的绝对值
        result.sign = d2.sign;
```

```
            result.data = d2.data - d1.data;
        }
        return result;
}

signedInt operator-(const signedInt &d1, const signedInt &d2)
{
        signedInt tmp = d2;

        if (tmp.sign == '+') tmp.sign = '-'; else tmp.sign = '+';
        return d1 + tmp;
}
```

signedInt类的关系运算可以借助于LongLongInt类的关系运算实现。

```
bool operator==(const signedInt &d1, const signedInt &d2)
{   return d1.sign == d2.sign && d1.data == d2.data ;   }

bool operator!=(const signedInt &d1, const signedInt &d2)
{   return d1.sign != d2.sign || d1.data != d2.data ;   }

bool operator>(const signedInt &d1, const signedInt &d2)
{
    if (d1.sign == '+' && d2.sign == '-' ) return true;
    if (d1.sign == '-' && d2.sign == '+' ) return false;
    if (d1.sign == '+') return d1.data > d2.data;
    else return d1.data < d2.data;
}

bool operator>=(const signedInt &d1, const signedInt &d2)
{
    if (d1.sign == '+' && d2.sign == '-' ) return true;
    if (d1.sign == '-' && d2.sign == '+' ) return false;
    if (d1.sign == '+') return d1.data >= d2.data;
    else return d1.data <= d2.data;
}

bool operator<(const signedInt &d1, const signedInt &d2)
{    return !(d1 >= d2);   }

bool operator<=(const signedInt &d1, const signedInt &d2)
{    return !(d1 > d2);    }
```

用继承的方法：

带符号的任意长的整数类型可以在一个无符号整数的基础上扩展一个符号位，因此用继承方法实现时，新添加一个保存符号的数据成员sign。signInt类支持的操作有输入输出、比较操作、加法操作、减法操作、++操作和- -操作，我们将输入输出、加减运算和关系运算重载成友元函数，将++、--重载成成员函数。由于signedInt类没有增加指针类型的数据成员，也没有用动态变量，因此不需要析构函数。

```
class signedInt :public LongLongInt {
    friend ostream &operator<<(ostream &os, const signedInt &obj);
    friend istream &operator>>(istream &is, signedInt &obj);
    friend signedInt operator+(const signedInt &d1, const signedInt &d2);
    friend signedInt operator-(const signedInt &d1, const signedInt &d2);
    friend bool operator==(const signedInt &d1, const signedInt &d2);
    friend bool operator!=(const signedInt &d1, const signedInt &d2);
    friend bool operator>(const signedInt &d1, const signedInt &d2);
    friend bool operator>=(const signedInt &d1, const signedInt &d2);
```

```
    friend bool operator<(const signedInt &d1, const signedInt &d2);
    friend bool operator<=(const signedInt &d1, const signedInt &d2);
private:
    char sign;

public:
    signedInt(const char *n = "");
    signedInt &operator++();
    signedInt operator++(int t);
    signedInt &operator--();
    signedInt operator--(int t);
};

signedInt::signedInt(const char *n)
{
    LongLongInt &data = *this;

    if (strcmp(n, "") == 0) {
        sign = '+';
        data = LongLongInt("");
    }
    else if (n[0] == '+' || n[0] == '-') {
        sign = n[0];
        data = LongLongInt(n+1);
    }
    else {
        sign = '+';
        data =LongLongInt(n);
    }
}
```

如果当前对象是正整数，++操作就是将绝对值加1。如果对象是一个负整数，++操作就是将绝对值减1。

```
signedInt &signedInt::operator++()
{
    LongLongInt &data = *this;

    if (sign == '+') data = data + "1";
    else {
        data = data - "1";
        if (data == "0") sign = '+';
    }
    return *this;
}

signedInt signedInt::operator++(int t)
{
    signedInt result = *this;
    LongLongInt &data = *this;

    if (sign == '+') data = data + "1";
    else {
        data = data - "1";
        if (data == "0") sign = '+';
    }
    return result;
}
```

```
signedInt &signedInt::operator--()
{
    LongLongInt &data = *this;

    if (sign == '-') data = data + "1";
    else if (data == "0") {
        sign = '-';
        data = "1";
    }
    else    data = data - "1";
    return *this;
}

signedInt signedInt::operator--(int t)
{
    signedInt result = *this;
    LongLongInt &data = *this;

    if (sign == '-') data = data + "1";
    else if (data == "0") {
           sign = '-';
           data = "1";
         }
         else    data = data - "1";
   return result;
}
```

输出一个signedInt类的对象就是输出符号和绝对值。signedInt的输入还是采用了块状链表暂存输入的数字，然后根据输入数字的实际长度申请一个字符数组tmp，将块状链表中的数据拷贝到tmp。用tmp设置对象obj。

```
ostream &operator<<(ostream &os, const signedInt &obj)
{
    os << obj.sign << LongLongInt(obj);
    return os;
}

istream &operator>>(istream &is, signedInt &obj)
{
    struct Node {
        char ch[10];
        Node *next;
    };

    Node *head, *tail, *p;
    int len = 0, i;

    head = tail = new Node;
    while ((tail->ch[len % 10] = is.get()) != '\n') {
        ++len;
        if (len % 10 == 0) {
            tail->next = new Node;
            tail = tail->next;
        }
    }

    char *tmp = new char[len + 1];
    for (i = 0; i < len; ++i) {
        tmp[i] = head->ch[i % 10];
```

```
        if (i % 10 == 9) {
            p = head;
            head = head->next;
            delete p;
        }
    }
    delete head;
    tmp[len] = '\0';

    obj = signedInt(tmp);
    delete tmp;
    return is;
}

signedInt operator+(const signedInt &d1, const signedInt &d2)
{
    signedInt result;
    LongLongInt &data = result;

    if (d1.sign == d2.sign) {
        result.sign = d1.sign;
        data = LongLongInt(d1) + LongLongInt(d2);
    }
    else if (LongLongInt(d1) == LongLongInt(d2)) {
        result.sign = '+';
        data = "";
    }
    else if (LongLongInt(d1) > LongLongInt(d2)) {
        result.sign = d1.sign;
        data = LongLongInt(d1) - LongLongInt(d2);
    }
    else {
        result.sign = d2.sign;
        data = LongLongInt(d2) - LongLongInt(d1);
    }
    return result;
}

signedInt operator-(const signedInt &d1, const signedInt &d2)
{
    signedInt tmp = d2;

    if (tmp.sign == '+') tmp.sign = '-'; else tmp.sign = '+';
    return d1 + tmp;
}

bool operator==(const signedInt &d1, const signedInt &d2)
{   return d1.sign == d2.sign && LongLongInt(d2) == LongLongInt(d1) ;   }

bool operator!=(const signedInt &d1, const signedInt &d2)
{   return d1.sign != d2.sign || LongLongInt(d2) != LongLongInt(d1);   }

bool operator>(const signedInt &d1, const signedInt &d2)
{
    if (d1.sign == '+' && d2.sign == '-' ) return true;
    if (d1.sign == '-' && d2.sign == '+' ) return false;
    if (d1.sign == '+') return LongLongInt(d1) > LongLongInt(d2);
```

```
    else return LongLongInt(d1) < LongLongInt(d2);
}

bool operator>=(const signedInt &d1, const signedInt &d2)
{
    if (d1.sign == '+' && d2.sign == '-' ) return true;
    if (d1.sign == '-' && d2.sign == '+' ) return false;
    if (d1.sign == '+') return LongLongInt(d1) >= LongLongInt(d2);
    else return LongLongInt(d1) <= LongLongInt(d2);
}

bool operator<(const signedInt &d1, const signedInt &d2)
{    return !(d1 >= d2);   }

bool operator<=(const signedInt &d1, const signedInt &d2)
{    return !(d1 > d2);    }
```

【5】在第11章程序设计题第5题定义的安全的、动态的二维数组Matrix的基础上定义一个可指定下标范围的安全的、动态的二维数组MatrixAdvanced。

【解】MatrixAdvanced在Matrix的基础上增加了4个数据成员：rowLow、rowHigh、colLow和colHigh，分别表示行和列的下标范围。MatrixAdvanced类重载了函数调用运算符。构造MatrixAdvanced类对象时要给出行列的下标范围。构造函数用这些值初始化新增的数据成员，并根据下标范围计算二维数组的行数和列数，将行数和列数传给基类的构造函数构造一个基类对象。

```
class MatrixAdvanced :    public Matrix {
private:
      int rowLow;
      int rowHigh;
      int colLow;
      int colHigh;
public:
      MatrixAdvanced(int rl, int rh, int cl, int ch)
         : Matrix(rh – rl + 1, ch – cl + 1), rowLow(rl), rowHigh(rh), colLow(cl), colHigh(ch) {}
      double &operator()(int r, int c)
      {    return Matrix::operator()(r - rowLow, c - colLow); }
};
```

6．在程序设计题2定义的安全的动态的一维整型数组Array的基础上，用组合的方式定义一个安全的动态的二维整型数组。

【解】二维数组可以看成是一维数组的数组。保存一个二维数组可以用一个指针数组，数组的每个元素指向一个一维数组。所以Matrix类有3个数据成员：行数row、列数col和指向数组Array起始地址的指针storage。所需的成员函数有构造、析构和函数调用运算符重载。

```
class Matrix {
    int row;
    int col;
    Array **storage;
public:
    Matrix(int r, int c);
    ~Matrix() ;
    int &operator()(int r, int c);
};

Matrix::Matrix(int r, int c): row(r), col(c)
```

```
{
    storage = new Array *[row];
    for (int i = 0; i < row; ++i) storage[i] = new Array(col);
}

Matrix::~Matrix()
{
    for (int i = 0; i < row; ++i) delete storage[i];

    delete [] storage;
}

int &Matrix::operator()(int r, int c)
{
    assert(r >= 0 && r < row);
    return (*storage[r])[c];
}
```

【7】利用程序设计题3中实现的读者类、教师读者类和学生读者类完成一个简单的图书馆管理系统。该图书馆最多有20个读者。实现的功能有：添加一个读者（可以是教师读者，也可以是学生读者）、借书、还书和输出所有读者各借了些什么书。

【解】一个图书馆系统可以看成图书馆类的一个对象，为此可以先设计一个图书馆类。按照题意，图书馆包含一组读者，可以是教师读者，也可以是学生读者，我们用了一个基类的指针数组来表示。在基类中添加借书、还书和输出读者所借书的纯虚函数。图书馆的功能有：添加一个读者、借书、还书和输出所有读者各借了些什么书，每个功能被设计成一个成员函数。

```
class library {
    enum { MAX = 20 };          // 读者数上限
    reader *record[MAX];
    int noOfReader;
public:
    library() : noOfReader(0) {}
   void AddReader();
   void Borrow();
   void Return();
   void Show();
};
```

AddReader函数首先检查读者数是否以达到20，如果没有达到20 ，则添加一个读者。用户可以输入读者类别、姓名和部门。然后根据读者类别申请一个动态的学生读者对象或教师读者对象，将对象地址存入数组record。

```
void library::AddReader()
{
    char type;
    char name[10], dept[20];

    if (noOfReader == MAX) {
        cout << "不能添加读者，读者数已到上限" << endl;
        return;
    }
    cout << "请输入读者类别（t--教师， s--学生）：";
    cin >> type;
    cout << "请输入读者的姓名和部门：";
    cin >> name >> dept;
    if (type == 's') record[noOfReader] = new readerStudent(noOfReader, name, dept);
    else record[noOfReader] = new readerTeacher(noOfReader, name, dept);
```

```
    ++noOfReader;
    cout<< "添加读者成功！" << endl;
}
```

借书需要输入读者号和所借的书号。根据读者号在record数组中找到该读者对象的地址，对该对象调用bookBorrow函数。由多态性可知，当该读者是教师读者时调用的是教师读者类的bookBorrow，当该读者是学生读者时调用的是学生读者类的bookBorrow。还书功能也是如此。显示借书清单对每个读者调用show函数显示各自所借的书目。

```
void library::Borrow()
{
    int readerNo, bookNo;

    cout << "请输入读者编号：";
    cin >> readerNo;

    if (readerNo < 0 || readerNo >= noOfReader) {
        cout << "非法的读者号！" << endl;
        return;
    }
    cout << "请输入所借的书号：";
    cin >> bookNo;
    if (record[readerNo]->bookBorrow(bookNo))
        cout << "借书成功！" << endl;
    else cout << "此读者所借的书已达上限，不能再借！" << endl;
}

void library::Return()
{
    int readerNo, bookNo;

    cout << "请输入读者编号：";
    cin >> readerNo;

    if (readerNo < 0 || readerNo >= noOfReader) {
        cout << "非法的读者号！" << endl;
        return;
    }
    cout << "请输入所还的书号：";
    cin >> bookNo;
    if (record[readerNo]->bookReturn(bookNo))
        cout << "还书成功！"  << endl;
    else cout << "此读者没借过这本书，不能还！" <<  endl;
}

void library::Show()
{
    for (int i= 0; i < noOfReader; ++i) {
        cout << "读者" << i << "所借的书有：";
        record[i]->show();
    }
}
```

有了上述工具，实现图书馆管理系统只需定义一个图书馆类的对象obj，然后显示菜单，根据用户的选择调用obj的各项功能。

```
int main()
{
    int selector;
    library obj;
```

```
    while (true) {
        cout << "0 --  退出" << endl;
        cout << "1 --  添加读者" << endl;
        cout << "2 --  借书" << endl;
        cout << "3 --  还书" << endl;
        cout << "4 --  显示借书记录" << endl;

        cout << "请选择: ";
        cin >> selector;

        switch (selector) {
        case 0: return 0;
        case 1: obj.AddReader(); break;
        case 2: obj.Borrow(); break;
        case 3: obj.Return(); break;
        case 4: obj.Show();
        }
    }
    return 0;
}
```

【8】定义一个基本的银行账户类。在基本账户类的基础上派生出1年期定期账户、2年期定期账户、3年期定期账户和5年期定期账户。定义一个n个基类指针组成的数组，随机生成n个各类派生类的对象。让每个指针指向一个派生类的对象。这些对象可以是1年期定期账户、2年期定期账户、3年期定期账户，也可以是5年期定期账户。输出每个账户到期的利息。

【解】基本帐户类有账号、账户名、开户日期和帐户余额4个数据成员。这些数据成员都设为保护成员。基本帐户类只需要一个构造的功能，但为了规范派生类的功能以及实现多态性，设计了一个纯虚函数show。

```
#include <iostream.h>
#include <cstring>
#include <cstdlib>
#include <ctime>
#include <iomanip.h>

class account{
protected:
    int no;
    char name[10];
    char date[7];
    double balance;
public:
    account(int n, char *nm, char *d, double b);
    virtual void show() const = 0;
};

account::account(int n, char *nm, char *d, double b):no(n), balance(b)
{
    strcpy(name, nm);
    strcpy(date, d);
}
```

一年期帐户增加一个静态的数据成员：一年期利率，所需的功能有显示帐户信息及到期利率，另外还需要一个设置利率的静态成员函数setRate。2年期定期账户、3年期定期账户，和5年期定期账户的设计基本类似。

```
class account1Year : public account {
```

```
    static double rate;
public:
    account1Year(int n, char *nm, char *d, double b):account(n, nm,  d, b) {}
    void show() const  {
       cout << setw(10) << no << setw(15) << name << setw(10) << balance << setw(10)
            << rate << setw(10) << balance * rate << endl; }
    static void setRate(double r) {rate = r; }
};

class account2Year : public account {
    static double rate;
public:
    account2Year(int n, char *nm, char *d, double b):account(n, nm,  d, b) {}
    void show() const
      { cout << setw(10) << no << setw(15) << name << setw(10) << balance << setw(10)
            << rate << setw(10) << balance * rate * 2 << endl; }
    static void setRate(double r) {rate = r; }
};

class account3Year : public account {
    static double rate;
public:
    account3Year(int n, char *nm, char *d, double b):account(n, nm,  d, b) {}
    void show() const
      { cout << setw(10) << no << setw(15) << name << setw(10) << balance << setw(10)
            << rate << setw(10) << balance * rate * 3 << endl; }
    static void setRate(double r) {rate = r; }
};

class account5Year : public account {
    static double rate;
public:
    account5Year(int n, char *nm, char *d, double b):account(n, nm,  d, b) {}
    void show() const
     { cout << setw(10) << no << setw(15) << name << setw(10) << balance << setw(10)
           << rate << setw(10) << balance * rate * 5 << endl; }
    static void setRate(double r) {rate = r; }
};
```

按照题意，系统可以保存一组帐户，假设为30个，程序中的*data就是用于保存指向这组对象的指针。然后随机生成30个帐户。对于每个帐户，先生成帐户类别以及存入金额，我们用0-3分别表示1年期、2年期、3年期和5年期帐户，然后根据帐户类别生成不同的帐户对象，将地址存入数组data。最后输出每个帐户的到期利息。

```
double account1Year::rate = 0;
double account2Year::rate = 0;
double account3Year::rate = 0;
double account5Year::rate = 0;

int main()
{
    account1Year::setRate(0.01);
    account2Year::setRate(0.02);
    account3Year::setRate(0.03);
    account5Year::setRate(0.04);

    account *data[30];
    int type;
    char date[7] = "051112";   //12年11月5日
```

```
    char name[10] = "aaaaaaaaa";
    double deposit;

    srand(time(NULL));
    for (int i = 0; i < 30; ++i) {
        type = rand() % 4;                  // 生成不同的帐户类型
        ++name[type];                       // 生成不同的账户名称
        deposit = rand() / 100.0;             // 生成存入的金额
        switch (type) {
          case 0: data[i] = new account1Year(i, name, date, deposit); break;
          case 1: data[i] = new account2Year(i, name, date, deposit); break;
          case 2: data[i] = new account3Year(i, name, date, deposit); break;
          case 3: data[i] = new account5Year(i, name, date, deposit);
        }
    }

    for ( i = 0; i < 30; ++i) {
        data[i]->show();
        delete data[i];
    }
    return 0;
}
```

第13章 简答题

【1】什么是模板的实例化？

【解】类模板只是个设计图纸，不是一个真正的类。要使得类模板变成一个真正的类，必须用真正的类型名或常量替换类模板的形式参数。这个过程称为类模板的实例化。实例化后，类模板成为了一个真正的类，可以定义这个类的对象了。

【2】为什么要定义模板？定义类模板有什么好处？

【解】有了类模板，可以将一组功能类似、存储方式也类似的类定义成一个类模板，可以进一步减少程序员的工作量。

【3】同样是模板，为什么函数模板的使用与普通的函数完全一样，而类模板在使用时还必须被实例化？

【解】在函数模板中，如果模板的形式参数出现在函数形式参数表中，那么当函数调用时，编译器可以根据函数的实际参数的类型确定模板的实际参数，然后对函数模板进行实例化。在定义类模板的对象时，无法确定模板形式参数对应的实际参数值，因此只能在程序中显式地指出模板实际参数的值。

【4】. 什么时候需要用到类模板的声明？为什么？

【解】一般来说，当定义一个类模板是另一个类模板的友元时必须要用到类模板的声明。当类模板A声明类模板B是它的友元时，编译器必须知道有这样的一个类模板B存在。如果类模板B的定义出现在类模板A的定义前面，则没有问题。但如果类模板B定义在类模板A后面时，编译器就不知道B是什么，也无法确定类模板A中对类模板B的名是否合法，这时可以通过类模板的声明来告诉编译器B是一个类模板。

【5】. 类模板继承时的语法与普通的类继承有什么不同？

【解】类模板继承时，凡是涉及到基类的地方，都必须在基类名后面跟上模板的形式参数名。即：基类名<形式参数1，形式参数2，……>。

【6】. 定义了一个类模板，在编译通过后为什么还不能确保类模板的语法是正确的？

【解】由于类模板包含有模板参数，这些模板参数对应的实际参数在编译时尚未确定，所以编译器无法确定那些类型为模板参数的数据或函数的用法是否正确，只好暂时不检查。

第13章 程序设计题

【1】设计一个处理栈的类模板，要求该模板用一个数组存储栈的元素。数组的初始大小由模板参数指定。当栈中的元素个数超过数组大小时，重新申请一个大小为原来数组一倍的新数组存放栈元素。

【解】栈类有3个数据成员。data为数组的起始地址，size为数组的规模，top_p为栈顶位置。有5个成员函数：创建一个空栈（由构造函数完成）、进栈（push）、出栈（pop）、读栈顶元素(top)和判栈空(isEmpty)。还需要一个析构函数和一个工具函数doubleSpace。

```
template <class T>
class stack {
    private:
        T *data;
        int size;
        int top_p;
        void doubleSpace();
    public:
        stack(int S): size(S), top_p(-1) {data = new T[size]; }
        ~stack() { delete [] data;  }
        void push(T d)
        {
            if (top_p == size - 1) doubleSpace();
            data[++top_p] = d;
        }
        T pop() {return data[top_p--]; }
        bool isEmpty() { return top_p == -1;  }
        T top() { return data[top_p];  }
};

template <class T>
void stack<T, S>::doubleSpace()
{
    T *tmp = data;

    size *= 2;
    data = new T[size];
    for (int i = 0; i <= top_p; ++i) data[i] = tmp[i];
    delete [] tmp;
}
```

【2】. 设计一个处理集合的类模板，要求该类模板能实现集合的并、交、差运算。

【解】模板的参数是集合中的元素类型。保存一个集合可以用一个动态数组，类模板的行为有并、交、差，我们用运算符+、*和-表示这3个运算。

```
template <class T>
class set {
    friend set<T> operator+(const set<T> &s1,  const set<T> &s2);
    friend set<T> operator*(const set<T> &s1,  const set<T> &s2);
    friend set<T> operator-(const set<T> &s1,  const set<T> &s2);
    friend ostream &operator<<(ostream &os, const set<T> &obj)
    {
        for (int i = 0; i < obj.size; ++i) os << obj.data[i] << '\t';
        return os;
    }
    private:
        T *data;
        int size;
    public:
        set(T *d = NULL, int s = 0): size(s)
```

```
        {
            if (s == 0) return;
            data = new T[size];
            for (int i = 0; i < s; ++i) data[i] = d[i];
        }
        set(const set &obj) : size(obj.size) {
            if (size == 0) return;
            data = new T[size];
            for (int i = 0; i < size; ++i) data[i] = obj.data[i];
        }
        ~set() { delete [] data;  }
        set &operator=(const set &obj) {
            if (&obj == this) return *this;
            if (size != 0)  delete [] data;
            size = obj.size;
            if (size == 0) return *this;
            data = new T[size];
           for (int i = 0; i < size; ++i) data[i] = obj.data[i];
        }
};

template <class T>
set<T> operator+(const set<T> &s1,  const set<T> &s2)
{
    int i, j, k, s = s1.size + s2.size;
    T *tmp = new T[s];

    for (i = 0; i < s1.size; ++i) tmp[i] = s1.data[i];
    for ( j = 0 ;  j < s2.size; ++j) {
        for (k = 0; k < s1.size; ++k)
            if (s1.data[k] == s2.data[j]) break;
        if (k == s1.size) tmp[i++] = s2.data[j];
    }

    set<T> obj(tmp, i);
    delete [] tmp;
    return obj;
}

template <class T>
set<T> operator*(const set<T> &s1,  const set<T> &s2)
{
    int i = 0, j, k;
    T *tmp = new T[s1.size];

    for ( j = 0 ;  j < s2.size; ++j) {
        for (k = 0; k < s1.size; ++k)
            if (s1.data[k] == s2.data[j]) break;
        if (k < s1.size) tmp[i++] = s2.data[j];
    }

    set<T> obj(tmp, i);
    delete [] tmp;
    return obj;
}

template <class T>
```

```
set<T> operator-(const set<T> &s1,  const set<T> &s2)
{
    int i = 0, j, k;
    T *tmp = new T[s1.size];

    for ( j = 0 ;  j < s1.size; ++j) {
        for (k = 0; k < s2.size; ++k)
            if (s1.data[j] == s2.data[k]) break;
        if (k == s2.size) tmp[i++] = s1.data[j];
    }

    set<T> obj(tmp, i);
    delete [] tmp;
    return obj;
}
```

【3】在教材例13.1定义的Array类中，当生成一个对象时，用户程序员必须提供数组的规模，构造函数根据程序员给出的数组规模申请一个动态数组。在析构函数中释放动态数组的空间。如果用户程序员定义了一个对象后一直没有访问这个对象，动态数组占用的空间就被浪费了。一种称为“懒惰初始化”的技术可以解决这个问题。所谓懒惰初始化就是在定义对象时并不给它申请动态数组的空间，而是到第一次访问对象时才申请。修改类模板Array，完成懒惰初始化。

【解】在构造函数中不再为storage申请空间了，而仅把storage置为空指针。在下标运算符重载函数中，在检查了下标范围后还需检查storage是否为空指针。如果是空指针，则先为storage申请空间

```
// 构造函数
template <class T>
Array<T>::Array(int lh = 0, int rh = 0):low(lh),high(rh), storage(NULL) {}

// 下标运算符重载函数
template <class T>
T & Array<T>::operator[](int index)
{
    assert (index >= low && index <= high);
    if (storage == NULL) storage = new T[high - low + 1];

    return storage[index - low];
}
```

【4】. 改写教材代码清单13-6中的单链表类，将结点类作为链表类的内嵌类。

【解】

```
template <class elemType>
class linkList {
   friend ostream &operator<<( ostream &, const linkList<elemType> &);
private:
   void makeEmpty();                        // 清空链表
   class Node {
     public:
        elemType  data;
        Node  *next;
        Node(const elemType &x, Node *N = NULL) { data = x; next = N;}
        Node( ):next(NULL) {}
        ~Node() {}
    };
    Node *head;
```

```
public:
    linkList() { head  = new Node;  }
    ~linkList() {makeEmpty(); delete head;}
    void create(const elemType &flag);
};
```

第14章 简答题

【1】什么是打开文件？什么是关闭文件？为什么需要打开和关闭文件？

【解】打开文件就是将一个文件流对象和一个文件关联起来。关闭文件是切断文件流对象与文件的关系，表示不再需要访问此文件了。

【2】为什么要检查文件打开是否成功？如何检查？

【解】一旦文件打开失败，程序中的后续操作将无法进行。因此执行文件打开后必须检查打开是否成功。当文件打开失败时，文件流对象会包含值0。所以检查文件打开是否成功，只需要检查文件流对象的值是否为0。

【3】ASCII文件和二进制文件有什么不同？

【解】ASCII文件是将存储在文件中的每个字节解释成一个ASCII字符，二进制文件是将文件内容解释成一个二进制的比特流，由程序解释这些比特流的意义。

【4】既然程序执行结束系统会关闭所有打开的文件，为什么程序中还要用close关闭文件？

【解】在有些大系统中，某些文件会被反覆地打开或同时打开。如果某次打开后没有关闭可能会使某些文件操作的结果不正确。

【5】C++有哪4个预定义的流？

【解】C++预定义了四个流对象：cin、cout、cerr和clog，分别对应于标准输入流、标准输出流、无缓冲的标准错误流以及有缓冲的标准错误流。

【6】什么时候用输入方式打开文件？什么时候应该用输出方式打开文件？什么时候该用app方式打开文件？

【解】程序需要从文件读取信息时，以输入方式打开。如果需要把程序中的某些信息写到文件中去，以输出方式打开文件。当以输出方式打开某个文件时，文件中原有的内容会被清除。如果想保留文件中原有的内容，将新加入的内容添加在原内容后面，可以用app方式打开。

【7】哪些流操纵符只对下一次输入/输出有效？哪些流操纵符是一直有效直到被改变？

【解】常用的流操纵符中，setw和setFill只对下一次输入输出有效。设置整型数的基数和设置浮点数的精度是一直有效，直到被改变为止。

【8】. 各编写一条语句完成下列功能：

a．使用流操纵符输出整数100的八进制、十进制和十六进制的表示。

b．以科学计数法显示实型数123.4567。

c．将整型数a输出到一个宽度为6的区域，空余部分用'$'填空。

d．输出char *类型的变量ptr中保存的地址。

【解】

```
a、  cout << oct << 100 << '\t' << dec << 100 << '\t' << hex << 100 << endl;
b、 cout << scientific << 123.456 << endl;
c、 cout << setw(6) << setfill('$') << a << endl ;
d、 cout << (void *)ptr << endl;
```

第14章 程序设计题

【1】. 编写一个文件复制程序copyfile，要求在命令行界面中通过输入

```
copyfile src_name obj_name
```

将名为src_name的文件复制到名为obj_name的文件中。

【解】采用main函数的参数，源文件名存放在argv[1]中，目标文件名在argv[2]中。

```
#include <iostream>
```

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

int main(int argc, char *argv[])
{
    ifstream in(argv[1]);
    ofstream out(argv[2]);

   if (in == 0) {                    // 检查源文件打开是否成功
        cout << "输入文件打开失败" << endl;
        return 1;
    }
   if (out == 0) {                    // 检查目标文件打开是否成功
        cout << "输出文件打开失败" << endl;
        return 1;
    }

    char ch;

    while ((ch = in.get()) != EOF)  out.put(ch);     // 复制文件
    in.close();
    out.close();

    return 0;
}
```

2．编写一个文件追加程序addfile，要求在命令行界面中通过输入

addfile src_name obj_name

将名为src_name的文件追加到名为obj_name文件的后面。

【解】源文件名和目标文件名都作为main函数的参数，源文件名存放在argv[1]中，目标文件名存放在argv[2]中。

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

int main(int argc, char *argv[])
{
     ifstream in(argv[1]);
     ofstream out(argv[2], ofstream::app);

    if (in == 0) {
        cout << "输入文件打开失败" << endl;
        return 1;
     }
    if (out == 0) {
        cout << "输出文件打开失败" << endl;
        return 1;
     }

    char ch;

    while ((ch = in.get()) != EOF) out.put(ch);
    in.close();
    out.close();

     return 0;
```

```
}
```

【4】. 编写一个程序，打印1～100的数字的平方和平方根。要求用格式化的输出，每个数字的域宽为10，实数用5位精度右对齐显示。

【解】

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

int main()
{
	cout << "              n         " << "      n * n      " << " sqrt(n) " << endl;
	for (int n = 1; n <= 100; ++n)
		cout << setw(10) << n << setw(10) << n * n << setw(10) << setprecision(5) << sqrt(n) << endl;
	return 0;
}
```

【5】. 编写一个程序，打印所有英文字母（包括大小写）的ASCII值。要求对于每个字符，程序都要输出它对应的ASCII值的十进制、八进制和十六进制表示。

【解】

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

int main()
{
	char ch;

	cout << "  字母     八进制       十进制         十六进制\n";
	for (ch = 'a'; ch <= 'z'; ++ch)                          // 输出小写字母对应的ASCII值
		cout << setw(5) << ch << setw(8) << oct << int(ch)
			<< setw(10) << dec << int(ch) << setw(10) << hex << int(ch) << endl;
	for (ch = 'A'; ch <= 'Z'; ++ch)                          // 输出大写字母对应的ASCII值
		cout << setw(5) << ch << setw(8) << oct << int(ch) << setw(10) << dec
			<< int(ch) << setw(10) << hex << int(ch) << endl;

	return 0;
}
```

【6】利用第10章的程序设计题的第4题中定义的SavingAccount类，设计一个银行系统，要求该系统为用户提供开户、销户、存款、取款和查询余额的功能。账户信息存放在一个文件中。账号由系统自动生成。已销户的账号不能重复利用。

【解】程序定义了文件account保存帐户信息。account文件由两部分组成。第一部分是两个总体信息：帐户总数和利率。第二部分是一个个账户信息，即一个个savingAccount类的对象。系统初始化时创建account文件，将帐户总数（初始时为0）和利率写入文件。银行系统的功能有：开户、销户、存款、取款和查询余额，每个功能被设计成一个函数，每次启动银行系统时，首先以输入方式打开文件account，读入帐户总数和利率，存入savingAccount类的静态数据成员totalNo和rate，关闭文件。然后显示功能菜单，根据用户的选择调用相应的函数，直到用户选择退出。最后将静态数据成员totalNo和rate的值重新写回文件。

```
#include “savingAccount.h”
#include <iostream>
```

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

void updateMonthly();       // 每月更新帐户余额
void addAccount();          // 添加帐户
void deleteAccount();       // 删除帐户
void deposit();             // 存钱
void withdraw();            // 取钱
void query();               // 查询
void modifyRate();          // 修改利率
void initialize();          // 系统初始化

int main()
{
    ifstream account("account");
    int selector, total;
    double rate;

    if (!account)
        cout << "请先执行初始化操作" << endl;
    else {
        account.read(reinterpret_cast<char * >(&total), sizeof(int));
        account.read(reinterpret_cast<char * >(&rate), sizeof(double));
        savingAccount::setTotal(total);
        savingAccount::setRate(rate);
    }

    account.close();

    while (true) {
        cout << "0 -- 退出\n";
        cout << "1 -- 初始化\n";
        cout << "2 -- 月更新\n";
        cout << "3 -- 开户\n";
        cout << "4 -- 销户\n";
        cout << "5 -- 存款\n";
        cout << "6 -- 取款\n";
        cout << "7 -- 查询余额\n";
        cout << "8 -- 修改利率\n";
        cout << "请选择（0-8）： "; cin >> selector;
        if (selector == 0) break;
        switch (selector) {
            case 1: initialize(); break;
            case 2: updateMonthly(); break;
            case 3: addAccount(); break;
            case 4: deleteAccount();break;
            case 5: deposit(); break;
            case 6: withdraw(); break;
            case 7: query(); break;
            case 8: modifyRate();
        }
    }

    fstream accountw("account");
    total = savingAccount::getTotal();
    rate = savingAccount::getRate();

    accountw.seekp(0);
```

```
    accountw.write(reinterpret_cast<char * >(&total), sizeof(int));
    accountw.write(reinterpret_cast<char * >(&rate), sizeof(double));

    account.close();

    return 0;
}
```

系统初始化时需要生成文件account，然后将帐户总数（目前为0）和利率（目前未知，暂设为0）写入文件。

```
void initialize()
{
    ofstream account("account");
    int total = 0;
    double rate = 0;

    account.write(reinterpret_cast<char * >(&total), sizeof(int));
    account.write(reinterpret_cast<char * >(&rate), sizeof(double));
    savingAccount::setTotal(0);
    savingAccount::setRate(0);

    account.close();
}
```

月更新函数为每个帐户加上本月的利息。

```
void updateMonthly()
{
    savingAccount obj;
    int len = sizeof(int) + sizeof(double), n = savingAccount::getTotal();

    fstream account("account");
    if (!account) {
        cout << "文件打开错，无法继续操作" << endl;
        return;
    }

    for (int i = 1; i <= n; ++i) {
       account.seekg((i - 1) * sizeof(obj) + len);
       account.read(reinterpret_cast<char *>(&obj), sizeof(obj));
       if (obj.legalAccount()) obj.updateMonthly();
       account.seekp((i - 1) * sizeof(obj) + len);
       account.write(reinterpret_cast<char *>(&obj), sizeof(obj));
    }
    account.close();
}
```

添加帐户功能向系统中添加一个新的帐户。新的帐户信息要写入文件，所以以app方式打开文件。然后输入帐户的存款金额，按照存款金额和帐户总数生成一个新的帐户对象，显示对象信息以供检查，将对象信息写入文件。更新静态成员totalNo。

```
void addAccount()
{
    savingAccount *op;
    ofstream account("account", ofstream::app);
    int totalNo = savingAccount::getTotal();
    double deposit;

    if (!account) {
        cout << "文件打开错，无法继续操作" << endl;
        return;
```

```
    }

    cout << "请输入存款金额：";
    cin >> deposit;
    op = new savingAccount(++totalNo, deposit);
    op->print();
    account.write(reinterpret_cast<char * >(op), sizeof(*op));
    savingAccount::setTotal(totalNo);

    account.close();
}
```

deleteAccount函数首先以输入输出方式打开文件，然后输入被删除的账号，检查账号是否在合法的范围之中。如果是合法范围中的账号，从文件中读入记录，进一步检查该帐户是否已被销户。如果没有，对该帐户对象调用remove函数销户，将记录写回文件。

```
void deleteAccount()
{   savingAccount obj;
    fstream account("account");
    int no, len = sizeof(int) + sizeof(double);

    if (!account) {
        cout << "文件打开错，无法继续操作" << endl;
        return;
    }

    cout << "请输入被删除的账号：";
    cin >> no;
    if (no < 1 || no > savingAccount::getTotal()) {
        cout << "非法账号" << endl;
        return ;
    }
    account.seekg((no - 1) * sizeof(obj) + len);
    account.read(reinterpret_cast<char * >(&obj), sizeof(obj));
    if (obj.legalAccount()) obj.remove(); else cout << "该账户已被删除" << endl;
    account.seekp((no - 1) * sizeof(obj) + len);
    account.write(reinterpret_cast<char * >(&obj), sizeof(obj));
    account.close();
}
```

存、取钱函数和查询函数的实现基本类似。

```
void deposit()                          // 存款函数
{
    savingAccount obj;
    fstream account("account");
    int no, len = sizeof(int) + sizeof(double);
    double d;

    if (!account) {
        cout << "文件打开错，无法继续操作" << endl;
        return;
    }

    cout << "请输入存款账号和金额：";
    cin >> no >> d;
    if (no < 1 || no > savingAccount::getTotal()) {
        cout << "非法账号" << endl;
        return ;
    }
```

```
        account.seekg((no - 1) * sizeof(obj) + len);
        account.read(reinterpret_cast<char * >(&obj), sizeof(obj));
        if (obj.legalAccount()) { obj.deposit(d); obj.print(); }
        else cout << "该账户已被删除" << endl;

        account.seekp((no - 1) * sizeof(obj) + len);
        account.write(reinterpret_cast<char * >(&obj), sizeof(obj));
        account.close();
}

void withdraw()                         //  取款函数
{       savingAccount obj;
        fstream account("account");
        int no, len = sizeof(int) + sizeof(double);
        double d;

        if (!account) {
                cout << "文件打开错，无法继续操作" << endl;
                return;
        }

         cout << "请输入取款账号和金额：";
        cin >> no >> d;
        if (no < 1 || no > savingAccount::getTotal()) {
                cout << "非法账号" << endl;
                return ;
        }

        account.seekg((no - 1) * sizeof(obj) + len);
        account.read(reinterpret_cast<char * >(&obj), sizeof(obj));
        if (obj.legalAccount()) { obj.withdraw(d); obj.print(); }
        else cout << "该账户已被删除" << endl;
        account.seekp((no - 1) * sizeof(obj) + len);
        account.write(reinterpret_cast<char *>(&obj), sizeof(obj));
        account.close();
}

void query()
{
        savingAccount obj;
         int no, len = sizeof(int) + sizeof(double);
        fstream account("account");

        if (!account) {
                cout << "文件打开错，无法继续操作" << endl;
                return;
        }

         cout << "请输入查询账号：";
        cin >> no;
        if (no < 1 || no > savingAccount::getTotal()) {
                cout << "非法账号" << endl;
                return ;
        }

        account.seekg((no - 1) * sizeof(obj) + len);
        account.read(reinterpret_cast<char * >(&obj), sizeof(obj));
        if (obj.legalAccount())    obj.print(); else cout << "该账户已被删除" << endl;
```

```
    account.close();
}
```

利率被保存在savingAccount类的静态数据成员rate中，静态成员函数setRate可以设置rate的值。

```
void modifyRate()
{
    double rate;

    cout << "请输入新的利率：";
    cin >> rate;
    savingAccount::setRate(rate);
}
```

【7】文件a和文件b都是二进制文件，其中包含的都是一组按递增次序排列的整型数。编一个程序将文件a、b的内容归并到文件c。在文件c中，数据仍按递增次序排列。

【解】首先以输入方式打开文件a和文件b，以输出方式打开文件c，读入文件a和文件b中的第一个整数。归并过程可以分为两个阶段。第一阶段是文件a和文件b都有数据，第二阶段是文件a读完或文件b读完。第一阶段反复执行下列工作：比较两文件的第一个整数，将小的写入文件c，并读入该文件的下一个整数，直到某个文件结束。第二阶段将尚未读完的文件中的剩余整数复制到文件c。本程序需要注意的是所处理的文件是二进制文件，不能用>>和<<输入输出，要用read和write函数。

```
int main()
{
    ifstream fa("a"), fb("b");
    ofstream fc("c");
    int a, b;

    fa.read(reinterpret_cast<char *>(&a), sizeof(int));
    fb.read(reinterpret_cast<char *>(&b), sizeof(int));

    while (!fa.eof() && !fb.eof()) {              // 第一阶段：文件a、b都有数据
        if (a < b) {
            fc.write(reinterpret_cast<char *>(&a), sizeof(int));
            fa.read(reinterpret_cast<char *>(&a), sizeof(int));
        }
        else {
            fc.write(reinterpret_cast<char *>(&b), sizeof(int));
            fb.read(reinterpret_cast<char *>(&b), sizeof(int));
        }
    }
    while (!fa.eof()) {                           // 文件b结束
            fc.write(reinterpret_cast<char *>(&a), sizeof(int));
            fa.read(reinterpret_cast<char *>(&a), sizeof(int));
    }
    while (!fb.eof()) {                           //  文件 a 结束
            fc.write(reinterpret_cast<char *>(&b), sizeof(int));
            fb.read(reinterpret_cast<char *>(&b), sizeof(int));
    }

    fa.close();
    fb.close();
    fc.close();

    return 0;
}
```

【8】假设文件txt中包含一篇英文文章。编一程序统计文件txt中有多少行，多少个单词，有多少字符。假定文章中的标点符号只可能出现逗号或句号。

【解】行和行之间是以'\n'分割，统计行数只需统计'\n'的个数。但要注意最后一行可能没有回车。当读到一个标点符号、空格或回车时，如果前一个字符不是标点符号、空格或回车，表示一个单词结束，可以以此来统计单词数。字符数统计最方便，每读入一个字符，字符数加1。但要注意，回车不是可显示字符，应该扣除。

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

int main()
{
    ifstream in("txt.txt");
    char ch, prev = ' ';              // ch:当前读入的字符，prev：ch的前一字符
    int line = 0, word = 0, cha = 0;   // 分别表示行数、单词数和字符数

    if (!in) { cout << "文件打开错" << endl; return 1; }
    while ((ch = in.get() ) != EOF) {  // 逐个读入字符，直到文件结束
        ++cha;
        switch (ch) {
           case '\n': ++line;  --cha;
           case ' ': case ',': case '.':
             if (prev != '\n' && prev != ',' && prev != '.' && prev != ' ') ++word;
      }
       prev = ch;
    }

    if (prev != '\n') ++line;          // 最后一行无回车
    if (prev != '\n' && prev != ',' && prev != '.' && prev != ' ') ++word;

    cout << "共" << line << " 行，" << word << "个字，" << cha << "个字符" << endl;

   in.close();
   return 0;
}
```

【9】编写一程序输入十个字符串(不包含空格)，将其中最长的字符串的长度作为输出域宽，按右对齐输出这组字符串。

【解】假设输入的字符串最长不能超过80个字符，程序定义了一个10行、81列的二维字符数组存放输入的字符串。程序首先用一个重复10次的for循环输入10个字符串。每输入一个字符串都会计算它的长度，从中选出最长的长度max。然后利用C++的格式化输出设置域宽、设置右对齐，逐个输出字符串。

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

int main()
{
    char txt[10][81];
    int max = 0;

    for (int i = 0; i < 10; ++i) {      // 输入10个字符串，并记录最长的字符串的长度
        cout << "请输入一个字符串，长度不超过80，不能包含空白字符：";
        cin >> txt[i];
```

```
        if (strlen(txt[i]) > max) max = strlen(txt[i]);
    }

    for (i = 0; i < 10; ++i) {          // 格式化输出10个字符串
        cout << setw(max) << right << txt[i] << endl;
    }
    return 0;
}
```

【10】编写一程序，用sizeof操作来获取你的计算机上各种数据类型所占空间的大小，将结果写入文件size.data。直接显示该文件就能看到结果。例如，显示该文件的结果可能为

```
char          1
int           4
long int      8
  ……
```

【解】要直接显示文件的内容，文件必须设置成ASCII文件。程序首先以输出方式打开文件size.data。为了使文件中的信息能够对齐，程序利用了C++的格式化输出设置了两列信息的输出宽度，并设置类型名是左对齐。

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


int main()
{
    ofstream out("size.data");

    out << setw(10) << left << "int" << setw(5) << sizeof(int) << endl;
    out << setw(10) << left << "short" << setw(5) << sizeof(short) << endl;
    out << setw(10) << left << "long" << setw(5) << sizeof(long) << endl;
    out << setw(10) << left << "float" << setw(5) << sizeof(float) << endl;
    out << setw(10) << left << "double" << setw(5) << sizeof(double) << endl;
    out << setw(10) << left << "char" << setw(5) << sizeof(char) << endl;
    out << setw(10) << left << "bool" << setw(5) << sizeof(bool) << endl;
    out << setw(10) << left << "pointer" << setw(5) << sizeof(void *) << endl;

    out.close();

    return 0;
}
```

【11】编写一个程序，读入一个由英文单词组成的文件，统计每个单词在文件中的出现频率，并按字母序输出这些单词及出现的频率。假设单词与单词之间是用空格分开的。

【解】定义一个保存这些信息的数组result，数组元素由两个部分组成：单词和出现的频率。由于事先并不知道文件中有多少不同的单词，无法确定数组的规模，程序先定义了一个规模为10的动态数组，随着发现的单词数量的增加，再动态扩大数组规模。程序中定义了函数doubleSpace实现扩大数组规模。

程序首先以输入方式打开文件，然后反复调用函数getWord从文件中读入一个单词，直到文件结束。对每一个读入的单词，在数组result中查找该单词是否出现。为了加快查找速度，把数组result中的单词按升序排列。如果单词在result中存在，将该单词的出现频率加1。如果不存在，则将该单词加入result。

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

struct word {
	char data[20];
	int count;
};

void getWord(ifstream &fp, char ch[]);                    // 从文件中读入一单词
void doubleSpace(word *& list, int &size);                // 扩大数组空间

int main()
{
	ifstream in("txt.txt");
	char ch[20];
	int size = 10, len = 0, i, j;                         // len：不同单词数
	word *result = new word[10];

	if (!in) { cout << "文件打开错" << endl; return 1; }
	while (true) {                                        // 读文件直到结束
		getWord(in, ch);                                  // 读入一个单词ch
		if (ch[0] == '\0') break;
		for (i = 0; i < len; ++i)                         // 在result中查找单词ch是否出现
			if (strcmp(result[i].data, ch) >= 0) break;
		if (i < len && strcmp(result[i].data, ch) == 0) {     // 单词在result中出现
			result[i].count += 1;
			continue;
		}

		// 将单词加入result
		if (size == len) doubleSpace(result, size);
		for (j = len++; j > i; --j) result[j] = result[j-1];
		 strcpy(result[i].data, ch);
		result[i].count = 1;
	}

	for (i = 0; i < len; ++i)                             // 输出统计结果
		cout << setw(20) << left << result[i].data << setw(5) << result[i].count << endl;

	in.close();

	return 0;
}
```

函数getWord从文件fp中读入一个单词，存入字符数组ch。函数doubleSpace将数组list的空间扩大一倍。

```
void getWord(ifstream &fp, char ch[])
{
	int wLen = 0;

	while ((ch[wLen] = fp.get() ) != EOF) {
		if (ch[wLen] != ' ' && ch[wLen] != '\n') { ++wLen;	continue; }
		if (wLen != 0) break;                                 // 跳过单词前的空格
	}
	ch[wLen] = '\0';
}
```

```
void doubleSpace(word *& list, int &size)
{
      word *tmp = list;

      list = new word[2 * size];
      for (int j = 0; j < size; ++j) list[j] = tmp[j];
      size *= 2;
      delete [] tmp;
}
```

【14】编写一个程序输入任意多个实型数存入文件。将这批数据的均值和方差也存入文件。

【解】程序需要一个文件记录输入的数据，我们将这个文件命名为file。程序首先以输出方式打开文件file，从键盘接收用户输入的一个个实型数存入文件，并同时记录输入的数据个数count，计算输入数据的和sum，直到用户输入EOF。输入结束后，计算sum/count就得到了均值avg。要计算方差，必须重新访问输入的数据。为此，先将文件file关闭，再重新以输入方式打开。逐个读入文件中的实数num，计算$(num - avg)^2$的和sum。计算sum/count得到了方差。最后，关闭文件后再重新以app方式打开文件file，将均值和方差写入文件。

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

int main()
{
    ofstream out("file");

    int count = 0;
    double sum = 0, avg, num;

    if (!out) {
        cout << "文件打开错！ " << endl;
        return 1;
    }

    while (cin >> num) {              // 输入数据存入文件，并计算总和
        ++count;
        sum += num;
       out.write(reinterpret_cast<char *>(&num), sizeof(double));
    }

    avg = sum / count;                // 计算均值
    out.close();

    ifstream in("file");
    if (!in) {
        cout << "文件打开错！ " << endl;
        return 1;
    }

    sum = 0;
    while (true) {                    // 计算方差
       in.read(reinterpret_cast<char *>(&num), sizeof(double));
        if (in.eof()) break;
        sum += (num - avg) * (num - avg);
    }
    sum /= count;
```

```
    in.close();

    out.open("file", ios::app);
    if (!out) {
        cout << "文件打开错！ " << endl;
        return 1;
    }

    // 将均值和方差写入文件
    out.write(reinterpret_cast<char *>(&avg), sizeof(double));
    out.write(reinterpret_cast<char *>(&sum), sizeof(double));
    out.close();
    return 0;
}
```

第15章 简答题

【1】采用异常处理机制有什么优点？

【解】异常处理机制将异常检测和异常处理分开，由工具程序负责异常检测，而由使用工具的程序进行异常处理，更适合大型程序设计的需要。采用异常处理机制可以将异常集中在一起处理，使程序的主线条更加清晰，更能突出程序解决问题的思想。

【2】是不是每个函数都需要抛出异常？

【解】不是。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。

【3】如何让函数的使用者知道函数会抛出哪些异常？

【解】可以在函数原型声明时明确指出该函数可能抛出哪几类异常。

【4】抛出一个异常一定会使程序终止吗？

【解】不一定。如果A函数抛出了一个异常，当调用A函数的函数B对此异常有相应的异常处理，程序会正常执行。如果B函数没有对此异常进行处理，则B函数会将此异常抛给调用B函数的函数C。如果C函数也没有处理，则继续上抛，最后抛给了main函数。如果main函数也没有处理，此时程序会异常终止。

【5】在哪种情况下，异常捕获时可以不指定异常类的对象名？

【解】如果异常处理时不需要用到抛出的异常对象的值，异可以不指定异常类的对象名。

【6】为什么catch(…)必须作为最后一个异常捕获器？放在前面会出现社么问题？

【解】catch(…)表示捕获任意类的异常。如果将catch(…)作为第一个异常处理器，则跳出try块后抛出的任何异常就会被catch(…)捕获，其他异常处理器都无效了。

【7】是不是处理了异常规格说明中指定的所有异常，程序就是安全的，不会异常终止？

【解】不一定。A函数的异常规格中指定的异常一般是A函数明确可以检测到的异常。但A函数也可能调用了一些其它函数，如函数B。如果B函数没有明确指出会抛异常，但事实上却抛出了一些异常，A函数将不知道如何处理，只能将此异常抛给调用A函数的函数。此时抛出的异常就不是A函数在函数规格说明中指定的异常，调用A函数的函数也没有对此异常作出处理，此时可能会导致程序异常终止。

【8】写出下面程序执行的结果

```
class up
{ public:     up() {  cout << "It is up" << endl; }
};
class down
{ public:     down() { cout << "It is down" << endl;}
};
int f(int i) throw(up, down)
{ switch(i) {  case 1: throw up();
               case 2: throw down();
               default: return i;
```

```
            }
  }
  int main()
  {    for (int i = 1; i <= 3; i++)
          try { cout << f(i) << endl; }
               catch (up) { cout << "up catched" << endl;}
               catch (down) {cout << "down catched" << endl;}

       return 0;
  }
```

【解】

```
  It is up
  up catched
  It is down
  down catched
  3
```

第15章　程序设计题

【1】修改教材第11章中的DoubleArray类，使之在下标越界时抛出一个异常。

【解】

```
class indexOverflow {};    // 异常类的定义

double & DoubleArray::operator[](int index)
{ if (index < low || index > high) throw indexOverflow();
  return storage[index - low];
}

const double & DoubleArray::operator[](int index) const
{ if (index < low || index > high) throw indexOverflow();
  return storage[index - low];
}

DoubleArray DoubleArray::operator()(int start, int end, int lh)
{ if (start > end || start < low || end > high ) throw indexOverflow();      // 判断范围是否正确

    DoubleArray tmp(lh, lh + end - start);                                   // 为取出的数组准备空间
    for (int i = 0; i < end - start + 1; ++i)       tmp.storage[i] = storage[start + i - low];

    return tmp;
}
```

【2】写一个安全的整型类，要求可以处理整型数的所有操作，且当整型数操作结果溢出时，抛出一个异常。

【解】可能使整数操作出现溢出的运算有+、-、*、/。其他运算，如比较或输入输出都不会引起整数溢出。因此安全的整数类型必须对+、-、*、/运算中的溢出进行处理，于是重载了+、-、*、/运算。其他运算和普通整数运算完全相同，不需另外处理。于是在安全的整数类integer中设计了一个到int的类型转换函数，这样就不用重载其他的运算符了。当遇到其他运算时，自动转换成int型进行运算。

安全的整数类integer只需要一个int类型的数据成员data，除了上述函数外，还需要一个构造函数。

```
#include <iostream.h>

const int MAX = 2147483647;                          // VC中的整数范围
const int MIN = -2147483648;
```

```
class overflow {};                                  // 异常类定义

class integer {
    friend integer operator+(integer d1, integer d2);
    friend integer operator-(integer d1, integer d2);
    friend integer operator*(integer d1, integer d2);
    friend integer operator/(integer d1, integer d2);

    int data;
public:
    integer(int d) : data(d) {}
    operator int() const { return data; }
};
```

加、减、乘法运算基本过程都一样。首先检查两个运算数运算的结果是否溢出，即运算结果小于MIN或大于MAX。如果溢出，抛出异常，否则返回运算的结果。对于加法运算，溢出有两种可能。一种是两个运算数都是负数，加的结果可能小于最小整数。另一种是两个运算数都是正数，加到结果可能大于最大的整数。对于减法，溢出也有两种可能。一种是被减数是正数，减数是负数，相减的结果可能大于最大的整数。另一种是被减数是负数，减数是正数，相减的结果可能小于最小的整数。对于乘法，也同样可以分几种情况来讨论。

除法最简单，两数相除，结果的绝对值总会变小。只要除数不为0，就不会溢出。

```
integer operator+(integer d1, integer d2)
{
      if ((d1.data < 0 && d2.data < MIN - d1.data) || (d1.data > 0 && d2.data > MAX - d1.data))
                throw overflow();
      return d1.data + d2.data;
}

integer operator-(integer d1, integer d2)
{
     if ((d1.data < 0 && d2.data > d1.data - MIN) || (d1.data > 0 && d2.data < d1.data - MAX))
          throw overflow();
     return d1.data - d2.data;
}

integer operator*(integer d1, integer d2)
{
     if ((d1.data > 0 && (d2.data < MIN / d1.data || d2.data > MAX / d1.data))
          || (d1.data < 0 && (d2.data > MIN / d1.data || d2.data < MAX / d1.data)))
        throw overflow();
     return d1.data * d2.data;
}

integer operator/(integer d1, integer d2)
{
     if (d2.data == 0) throw overflow();
     return d1.data / d2.data;
}
```

【3】修改教材第11章程序设计题中实现的string类的取子串函数。当出现所取的子串范围不合法时抛出一个异常。

【解】

```
Class indexExcept {};
String String::operator()(int start, int end)
{
    if (start > end || start < 0 || end >= len) throw indexExcept();
```

```
    String tmp;

    delete tmp.data;
    tmp.len = end - start + 1;
    tmp.data = new char[len + 1];
    for (int i = 0; i <= end - start; ++i) tmp.data[i] = data[i + start];
    tmp.data[i] = '\0';
    return tmp;
}
```

【4】修改教材第13章中的栈类，在出栈时发现栈为空时抛出一个异常。

【解】

```
class empty {};
template <class elemType>
class Stack:public linkList<elemType> {
public:
   void push(const elemType &data)
      {    Node <elemType> *p = new Node<elemType>(data);
           p->next = head->next;
           head->next = p;
      }
   bool pop(elemType &data)                //栈为空时返回false，出栈的值在data中
      {    Node <elemType> *p = head->next;

           if ( p == NULL) throw empty();
           head->next = p->next;
           data = p->data;
           delete p;

           return true;
      }
};
```

第16章 简答题

【1】什么是容器？什么是迭代器？

【解】容器是能存储和处理一组同类数据的对象。迭代器可以看成是一个指针，指向容器中的某一具体元素。通过迭代器可间接访问容器中的某一元素。

【2】为什么通常都将迭代器类设计成容器类的公有内嵌类？这样设计有什么好处？

【解】为了访问容器中的某一具体元素，容器类一般都有一个对应的迭代器类。如果程序用到多种容器，则对应也会用到多个迭代器类。如果将迭代器类设计成独立的一个类，则必须为每个迭代器类取不一样的名字,这样用户程序员必须记住多个容器类的名字和多个迭代器类的名字。但如果将迭代器类设计成容器类内嵌类，这些迭代器类就可以取同样的名字，用户程序员只要记一个类名就可以了。由于将迭代器类设置成容器类的公有内嵌类。，用户程序员可以通过“容器类名::迭代器类名”定义相应容器的迭代器对象。

【3】教材第16章中的linkList类中的结点类为什么用struct定义？这种定义方式是否安全？

【解】用struct定义类时，所有没有指定访问特性的成员都是公有的，全局函数和其他类的成员函数可以直接访问这些成员，似乎不太安全。但结点类是被定义成LinkList类的私有内嵌类，只有LinkList类的成员函数才能“看见”并访问结点类。所以结点类是安全的。

第16章 程序设计题

【1】在教材的linkList类中增加一个删除容器中一部分对象的成员函数erase，要求该函数有两个迭代器对象的参数itr1和itr2，删除（itr1，itr2] 之间的所有对象。执行该操作后，itr2指向itr1指向的对象。

【解】

```
template <class elemType>
void linkList<elemType>::erase(Itr &p1, Itr &p2)
{
    Node *p = p1.current->next;                //  被删除的第一个结点

    while (p != p2.current) {
        p1.current->next = p->next;
        delete p;
        p = p1.current->next;
    }
    p1.current->next = p2.current = p->next;
    delete p;
}
```

【2】为教材中的的linkList类重载==运算符。

【解】

```
template <class elemType>
bool operator==(const linkList<elemType> &obj1, const linkList<elemType> &obj2)
{
    linkList<elemType>::Node *p1 = obj1.head->next,  *p2 = obj2.head->next;

    while (p1 != NULL && p2 != NULL) {               // 比较对应结点的值
        if (p1->data != p2->data) return false;
        p1 = p1->next;
        p2 = p2->next;
    }
    if (p1 == NULL && p2 == NULL) return true;          // 结点数相同
    else return false;
}
```

【3】为主教材中的seqList类重载赋值运算符，使之实现两个容器的赋值。

【解】函数首先检查赋值运算符左右是否为同一对象。如果是同一对象，则不需赋值，直接返回。否则释放左边对象的数组空间，重新申请一块和右边对象的数组一样大的一块空间，并将右边对象的数组值复制到左边对象的数组中。

```
template <class T>
seqList<T> &seqList<T>::operator=(const seqList<T> &right)
{
    if (this == &right) return *this;
    delete [] storage;
    size = right.size;
    storage = new T[size];
   for (int i = 0; i < current_size; ++i) storage[i] = right.storage[i];
    return *this;
}
```

【4】为linkList和seqList类增加一个查找功能，查找某一元素在容器中是否存在。如果存在，返回指向该元素的迭代器对象。如果不存在，抛出一个异常。

【解】

```
    //linkList类中find函数的实现
```

```
Itr find(const elemType &x)
{   Node  *q = head->next;
   while (q != NULL && q->data != x) q = q->next;
    if ( q == NULL) throw notFound();
    return q;
}

//seqList类中的find函数的实现
Itr find(const T &x) {
    for (int i = 0; i < current_size; ++i)
        if (storage[i] == x) return &storage[i];
    throw notFound();
}
```