程序设计教程

——用 C++语言编程

（第二版习题解答）

目 录

第 1 章 概述

1、简述冯•诺依曼计算机的工作模型。

答：冯•诺依曼计算机的工作模型是：待执行的程序从外存装入到内存中，CPU 从内存中逐条地取程序中的指令执行；程序执行中所需要的数据从内存或从外设中获得，程序执行中产生的中间结果保存在内存中，程序的执行结果通过外设输出。

2、简述寄存器、内存以及外存的区别。

答：寄存器主要用于记录下一条指令的内存地址、当前指令的执行状态以及暂时保存指令的计算结果供下一（几）条指令使用，其作用主要是减少访问内存的次数，提高指令的执行效率。

内存用于存储计算机程序（指令和数据），内存由许多存储单元构成，每个存储单元都有一个地址，对存储单元的访问是通过其地址来进行的，与寄存器相比，内存的容量要大得多，但指令访问内存单元所花费的时间比访问寄存器要多得多。

外存是大容量的低速存储部件，用于永久性地存储程序、数据以及各种文档等信息，存储在外存中的信息通常以文件形式进行组织和访问，外存储了在容量和速度上与内存不同，另一个区别在于内存中存储的是正在运行的程序和正在使用的数据，外存中存储的则是大量的、并非正在使用的程序和数据。

3、CPU 能执行哪些指令？

答：CPU 所能执行的指令通常有：

算术指令：实现加、减、乘、除等运算。

比较指令：比较两个操作数的大小。

数据传输指令：实现 CPU 的寄存器、内存以及外设之间的数据传输。

执行流程控制指令：用于确定下一条指令的内存地址，包括转移、循环以及子程序调用/返回等指令。

4、什么是软件？软件是如何分类的？

答：计算机软件是计算机系统中的程序以及有关的文档。程序是对计算任务的处理对象（数据）与处理规则（算法）的描述；文档是为了便于人理解程序所需的资料说明，供程序开发与维护使用。

软件通常可以分为系统软件、支撑软件和应用软件。系统软件居于计算机系统中最靠近硬件的一级，它与具体的应用领域无关，其他软件一般要通过系统软件发挥作用，如操作系统属于系统软件。支撑软件是指支持软件开发与维护的软件，一般由软件开发人员使用，如软件开发环境就是典型的支撑软件。应用软件是指用于特定领域的专用软件，如人口普查软件、财务

软件等。

5、什么是虚拟机？

答：在由硬件构成的计算机（称为“**裸机**”）之上，加上一些软件就得到了一个比它功能更强的计算机，称为“**虚拟机**”。

6、十进制数 0.1 的二进制表示是什么？

答：$(0.1)_{10} = (0.00011\underline{0011}...)_2$，它是无限循环小数。也就是说，十进制数 0.1 无法精确用二进制表示！

7、简述程序设计范型。

答：基于不同的计算模型来对计算进行描述就形成了不同的**程序设计范型**。典型的程序设计范型有：过程式、对象式、函数式以及逻辑式等。

过程式程序设计是一种以功能为中心、基于功能分解和过程抽象的程序设计范型。一个过程式程序由一些子程序构成，每个子程序对应一个子功能，它实现了功能抽象。

对象式程序设计是一种以数据为中心、基于数据抽象的程序设计范型。一个面向对象程序由一些对象构成，对象是由一些数据及可施于这些数据上的操作所组成的封装体。

函数式程序设计是围绕函数来进行的，计算过程体现为一系列的函数应用。

逻辑程序设计是把程序组织成一组事实和一组推理规则，在事实基础上运用推理规则来实施计算。

8、简述程序设计的步骤。

答：程序设计一般遵循以下步骤：

明确问题；　　系统设计；　　用某种语言进行编程；　　测试与调试；　　运行与维护

9、低级语言与高级语言的不同之处是什么？

答：**低级语言**是指与特定计算机体系结构密切相关的程序语言，它是特定计算机能够直接理解的语言（或与之直接对应的语言），包括机器语言和汇编语言。低级语言的优点在于：写出的程序效率比较高，包括执行速度快和占用空间少。其缺点是：程序难以设计、理解与维护，难以保证程序的正确性。

高级语言是指人容易理解和有利于人对解题过程进行描述的程序语言。高级语言的优点在于：程序容易设计、理解与维护，容易保证程序正确性。高级语言的缺点是：用其编写的程序相对于用低级语言编写的程序效率要低，翻译成的目标代码量较大。

10、简述编译与解释的区别。

答：编译是指把高级语言程序首先翻译成功能上等价的机器语言程序或汇编语言程序，然后执行目标代码程序，在目标代码程序的执行中不再需要源程序。

解释则是指对源程序中的语句进行逐条翻译并执行，翻译完了程序也就执行完了，这种翻译方式不产生目标程序。一般来说，编译执行比解释执行效率要高。

11、简述 C＋＋程序的编译执行过程。在你的 C++开发环境中运行 1.3.2 节中给出的简单 C++程序。

答：首先可以利用某个编辑程序把 C++源程序输入到计算机中，并作为文件保存到外存中，文件名为“*.cpp”和“*.h”。然后利用某个 C++编译程序对保存在外存中的 C++源程序进行编译，编译结果作为目标文件保存到外存，文件名为“*.obj”。然后再通过一个联接程序把由源文件产生的目标文件以及程序中用到的一些系统功能所在的目标文件联接起来，作为一个可执行文件保存到外存，文件名为“*.exe”。最后通过操作系统提供的应用程序运行机制，把可执行文件装入内存，运行其中的可执行程序。

在 Visual C++ 6.0 环境中，首先要建立一个 project（项目）；其次往该 project 中添加、编辑程序模块（源文件）；然后选择菜单 Build 中的 Build ... 或 Rebuild All；最后选择菜单 Build 中的 Execute ...运行程序。

12、C++的单词分成哪些种类？

答：构成 C++的单词有：标识符、关键词、字面常量、操作符以及标点符号等。

13、下面哪一些是合法的 C++标识符？

```
extern, _book, Car, car_1, ca1r, 1car, friend, car1_Car, Car_Type, No.1, 123
```

答：合法的 C++标识符：_book, Car, car_1, ca1r, car1_Car, Car_Type

第 2 章 基本数据类型和表达式

1、 C++提供了哪些基本数据类型？检查你的计算机上各种类型数据所占内存空间的大小（字节数）。

答：C++提供了以下 5 种基本数据类型：整数类型、实数类型、字符类型、逻辑类型以及空值类型。一台计算机上各种数据类型的数据所占用的内存大小（字节数）可以通过“sizeof(类型名)”来计算。

2、 下面哪一些是合法的 C++字面常量，它们的类型是什么？

```
-5.23,  1e+50,   -25,    10^5,      20
.20,    e5,      1e-5,   -0.0e5,   '\n'
-000,   'A',     '5',    '3.14',    false
red,    '\r',    '\f'    "Today is Monday.",  "\""
```

答：字面常量是指在程序中直接写出常量值的常量。-5.23, 1e+50,-25, 20 ,.20, 1e-5, -0.0e5, '\n', -000, 'A', '5', '\r', '\f' , "Today is Monday.", "\""都是字面常量。其中：

整数类型常量：-25, 20, -000

实数类型常量：-5.23, 1e+50 , .20, 1e-5, -0.0e5

字符常量：'\n', 'A', '5', '\r', '\f'

字符串常量："Today is Monday.", "\""

3、 什么是符号常量？符号常量的优点是什么？

答：符号常量是指有名字的常量，在程序中通过常量的名字来使用这些常量。程序中使用符号常量有以下优点：

1） 增加程序易读性

2） 提高程序对常量使用的一致性

3） 增强程序的易维护性

4、 如何理解变量？变量定义和声明的作用是什么？

答：在程序中，其值可以改变的量称为变量。变量可以用来表示可变的数据。

程序中使用到的每个变量都要有定义。变量定义指出变量的类型和变量名，另外还可以为变量提供一个初值。

C++中使用变量之前，必须对使用的变量进行声明（变量定义属于一种声明，即：定义性声明），变量声明指出了一个变量的类型，使得编译程序能对变量的操作进行类型检查并做相应的类型转换。

整个程序中，某变量的定义只能由一个，但它的声明可以有多个。

5、 什么是表达式？其作用是什么？

答：表达式是由操作符、操作数以及圆括号所组成的运算式。在程序设计语言中，对数据操作的具体实施是通过表达式来描述的。

6、 操作符的优先级和结合性分别是指的什么？

答：运算符的优先级和结合性决定表达式中各个运算符的运算次序。操作符的优先级规定了相

邻的两个操作符谁先运算：优先级高的先计算；如果相邻的两个操作符具有相同的优先级，则需根据操作符的结合性来决定先计算谁，操作符的结合性通常分为左结合和右结合：左结合表示从左到右计算，右结合表示从右到左计算。

7、 表达式中的类型转换规则是什么？下面的表达式计算时如何进行操作数类型转换？

（1）3/5*12.3

（2）'a'+10*5.2

（3）12U+3.0F*24L

答：表达式中类型转换规则是：基于单个操作符依次进行转换。

1）3 与 5 同类型，不转换，结果为 0，转换成double型后与 12.3 做乘法。

2）10 转换成double型与 5.2 做乘法，'a'转换成double型后与前者结果做加法。

3）3.0F与 24L均转换成double型后做乘法，12U转换成double型后与前者结果做加法。

8、 将下列公式表示成 C++的表达式：

（1） $\frac{-b+\sqrt{b^2-4ac}}{2a}$ **（可利用 C++标准库中的求平方根的函数：sqrt(x)）**

（2） $\sqrt{s(s-a)(s-b)(s-c)}$

（3） $\frac{a\cdot b}{c\cdot d}\cdot\frac{3}{1+\frac{b}{2.5+c}}+\frac{4\cdot\pi\cdot r^3}{3}$

答：1）`(-1*b+sqrt(b*b-4*a*c))/(2*a)`

2）`sqrt(s*(s-a)*(s-b)*(s-c))`

3）`((a*b)/(c*d))*(3/(1+(b/(2.5+c))))+(4*pi*r*r*r/3)`

9、写出下列条件的 C++表达式

（1）i 能被 j 整除。

（2）ch 为字母字符。

（3）m 为偶数。

（4）n 是小于 100 的奇数。

（5）a、b、c 构成三角形的三条边。

答：1）`i%j==0`

2）`((ch>='a')&&(ch<='z'))||((ch>='A')&&(ch<='Z'))`

3）`m%2==0`

4）`(n<100)&&(n%2!=0)`

5）`(a>0) && (b>0) && (c>0) && (a+b>c) && (b+c>a) && (c+a>b)`

或

`((a+b)>c)&&(abs(a-b)<c)` `//(a>0)&&(b>0)&&(c>0)`可以不用判断

10、在你的计算机上运行下面的程序：

```
#include <iostream>
using namespace std;
int main()
{ double a=3.3, b=1.1;
  int i=a/b;
  cout << i << endl;
  return 0;
}
```

结果与你预期的是否相符？如果不符，请解释它的原因。

答：运行结果为 2。由于十进制小数 3.3 和 1.1 无法用double型精确表示。通过查看结果内存内的内容，最终结果比 3.0 略小，所以强制转换成int型后结果为 2。

11、不引进第三个变量，如何交换两个整型变量的值？

答：方法一：

```
a=b^a;
b=a^b;
a=b^a;
```

方法二：

```
a=a+b;
b=a-b;
a=a-b;
```

12、举例说明把 int 类型转成 float 类型可能会丢失精度。

答：如果 int 型与 float 型都是 4 个字节，由于在 float 型的数据表示中，有若干位用来表示指数，因此，尾数的位数不到 4 个字节（根据 IEEE 标准，只有 23 个二进制位）。如果一个 int 型的数大于 23 位（二进制），则无法用 float 型精确表示。例如：

int x=0x01000001;

float y=x; //x 的最后一位"1"不是被截掉就是被舍入！

cout << x << endl << setprecision(30) << y << endl;

第 3 章 程序的流程控制－－语句

1、 编写一个程序，将华氏温度转换为摄氏温度。转换公式为：

c = $\frac{5}{9}$(f-32)，其中，c 为摄氏温度，f 为华氏温度

解：

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

```
{  double c, f;
   cout << "Please input an F-temperature : " << endl;
   cin >> f;
   c = (f - 32) * 5 / 9;
   cout << "The C-temperature is : " << c << endl;
   return 0;
}
```

2、 编写一个程序，将用 24 小时制表示的时间转换为 12 小时制表示的时间。例如，输入 20 和 16（20 点 16 分），输出 8:16pm；输入 8 和 16（8 点 16 分），输出 8:16am。

解：

```
#include <iostream>
using namespace std;
int main()
{  int hour, minute;
   char noon;
   cout << "Please input a time in 24-hour format: " << endl;
   cout << "hour: ";    cin >> hour;
   if (hour<0 || hour>23)
   {    cout << "The input hour is wrong!" << endl;
        return -1;
   }
   if (hour>12)
   {    hour =hour - 12;
        noon = 'p';
   }
   else
        noon = 'a';
   cout << "minute: "; cin >> minute;
   if (minute<0 || minute>59)
   {    cout << "The input minute is wrong!" << endl;
        return -1;
   }
   cout << endl << "The time in 12-hour format is : " << hour << ":" << minute;
   if (noon == 'p')
        cout << "pm" << endl;
   else
        cout << "am" << endl;
   return 0;
}
```

3、 编写一个程序，分别按正向和逆向输出小写字母 a～z。

解：

```
#include <iostream>
using namespace std;
int main()
{  char c;
   for (c='a'; c<='z'; c++)
       cout << c << "    ";
   cout << endl;
   for (c='z'; c>='a'; c--)
       cout << c << "    ";
   cout << endl;
   return 0;
}
```

4、 编写一个程序，从键盘输入一个正整数，判断该正整数为几位数，并输出其位数。

解：

```
#include <iostream>
using namespace std;
int main()
{  unsigned int gzint;
   int count = 0;
   while (1)
   {   cout << "Please input an integer(greater than zero) : " << endl;
       cin >> gzint;
       if (gzint<=0)
         cout << "Your input is wrong! Please input again..." << endl;
       else
         break;
   }
   while (gzint!=0)
   {   gzint = gzint / 10;
       count++;
   }
   cout << "The number of digits in the interger is : " << count << endl;
   return 0;
}
```

5、 编写一个程序，对输入的一个算术表达式（以字符#结束），检查圆括号配对情况。输出：配对、多左括号或多右括号。

解：

```
#include <iostream>
using namespace std;
int main()
{  int count=0;
   char ch;
   cout << "Please input an expression : " << endl;
   for (cin >> ch; ch != '#'; cin >> ch)
   {    if (ch == '(')
            count++;
        else if (ch == ')')
          count--;
   }
   if (count == 0)
        cout << "配对!" << endl;
   else if (count > 0)
        cout << "多左括号!" << endl;
   else
        cout << "多右括号!" << endl;
   return 0;
}
```

6、 编写一个程序，输入一个字符串（以字符#结束），对其中的“>=”进行计数。

解：

```
#include <iostream>
using namespace std;
int main()
{  int count=0;
   char ch1='\0',ch2;
   cout << "Please input a string(terminated with #): " << endl;

   for (cin>>ch2; ch2 != '#' ; cin>>ch2)
   {    if (ch2 == '=' && ch1 == '>') count++;
      ch1 = ch2;
   }
   cout << "Number of >=: " << count << endl;
   return 0;
}
```

7、 假定邮寄包裹的计费标准如下（重量在档次之间时往上一挡靠）：

重量（克）	收费（元）
15	5
30	9
45	12
60	14（每满 1000 公里加收 1 元）
60 以上	15（每满 1000 公里加收 2 元）

编写一个程序，输入包裹重量和邮寄距离，计算并输出收费数额。

解：

```
#include <iostream>
using namespace std;
int main()
{  int charge;
   double weight;
   cout << "Please input the weight of the package : " << endl;
   cin >> weight;
   if (weight <= 0)
       cout << "The input weight is wrong!" << endl;
   else if (weight <= 15)
       charge = 5;
   else if (weight <= 30)
       charge = 9;
   else if (weight <= 45)
       charge = 12;
   else
   {   double distance;
       cout << "Please input the distance : " << endl;
       cin >> distance;
       if (distance <= 0)
         cout << "The inputed distance is wrong!" << endl;
       else
       {distance /= 1000;
        if (weight <= 60)
           charge = 14 + (int)distance;
        else
           charge = 15 + (int)distance * 2;
       }
   }
   cout << charge << endl;
```

```
    return 0;
}
```

8、 编写一个程序，计算圆周率。可利用公式：

$$\frac{\pi}{4}=1-\frac{1}{3}+\frac{1}{5}-\frac{1}{7}+\cdots$$

直到最后一项的绝对值小于 10^{-8}。

解：

```
#include <iostream>
#include <cmath>
using namespace std;
int main()
{  double item=1.0,sum=0.0;
   int i=1, sign=1;
   while (fabs(item) >= 1e-8)
   { sum += item;
      sign *= -1;
      i += 2;
       item = sign/(double)i;
   }
   cout << setprecision(8) << sum*4 << endl;
   return 0;
}
```

9、 编写一个程序，求所有这样的三位数，它们等于它们的各位数字的立方和。例如：

$153 = 1^3 + 3^3 + 5^3$

解：

```
#include <iostream>
using namespace std;
int main()
{  for (int n = 100; n <= 999; n++)
   {    int i,j, k;
        i = n/100; //百位数字
        j = n%100/10; //十位数字
        k = n%10; //个位数字
        if (n == i*i*i+j*j*j+k*k*k)
          cout << n << endl;
   }
   return 0;
}
```

或

```
#include <iostream>
using namespace std;
int main()
{  for (int i=1; i<=9; i++)
   { int n=i*100,m=i*i*i;
      for (int j=0; j<=9; j++)
      { int n1=n+j*10,m1=m+j*j*j;
         for (int k=0; k<=9; k++)
         { if (n1+k == m1+k*k*k)
               cout << n1+k << endl;
         }
      }
   }
   return 0;
}
```

10、编写一个程序，求 a 和 b 的最大公约数。

解：

```
#include <iostream>
using namespace std;
int main()
{  int a, b;
   cout << "Please input a, b : " << endl;
   cin >> a >> b;
   int c=(a>b)?b:a;
   while (c > 0)
   {    if (a%c == 0 && b%c == 0) break;
        c--;
   }
   cout << c << endl;
   return 0;
}
```

或

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

```
    cout << "Please input a, b : " << endl;
    cin >> a >> b;
    int c;
    do
    {   c = a-b*(a/b);
        a = b;
        b = c;
    } while (c != 0);
    cout << a << endl;
    return 0;
}
```

11、编写一个程序，输出十进制乘法表。

	1	2	3	...	9
1	1	2	3	...	9
2	2	4	6	...	18
3	3	6	9	...	27
:	:	:	:	...	:
9	9	18	27	...	81

解：

```
#include <iostream>
using namespace std;
int main()
{  for (int i = 0; i < 10; i++)
   {   if (i != 0)
         cout << i;
       cout << "\t";
       for (int j = 1; j < 10; j++)
        if (i * j != 0)
           cout << i * j << "\t";
        else
           cout << j << "\t";
       cout << endl;
   }
   return 0;
}
```

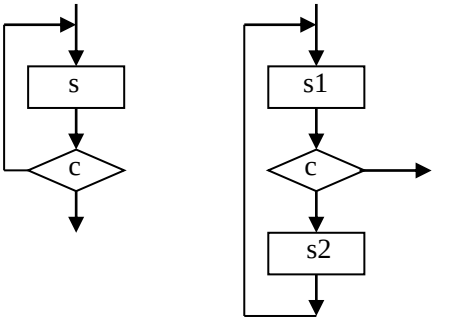
12、将下面的 for 循环重写为等价的 while 循环。

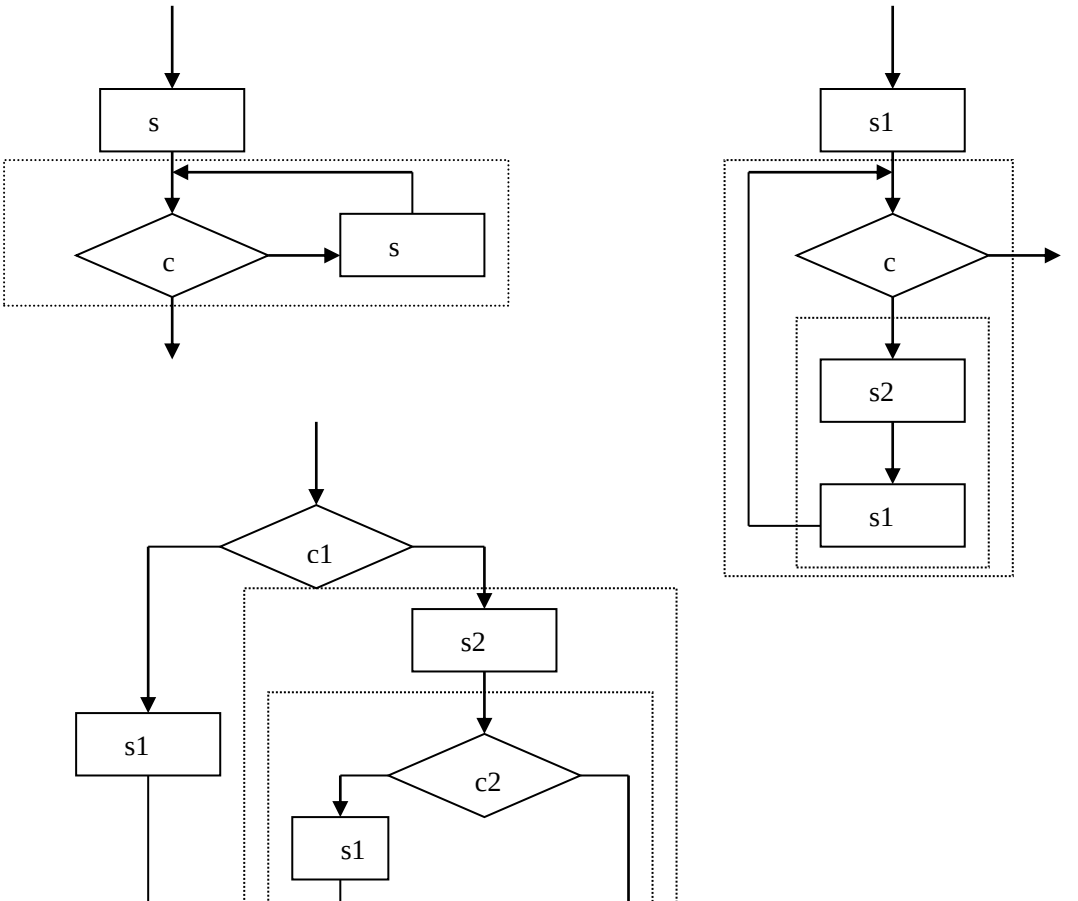
```
for (i=0; i<max_length;i++)
  if (input_line[i]== '?')  quest_count++;
```

解：

```
i = 0;
while (i < max_length)
{  if (input_line[i] == '?')  quest_count++;
   i++;
}
```

13、说明下面的三个程序可以用图 3-6 中的三种控制结构来表示。





第 4 章 过程抽象－－函数

1、 简述子程序的作用。

答：子程序是有名字的一段程序代码，它通常完成一个独立的（子）功能。在程序的其他地方通过子程序的名字来使用它们。除了能减少程序代码外，采用子程序的主要作用是实现过程抽象，使用者只需知道子程序的功能，而不需要知道它是如何实现的，这有利于大型、复杂程序的设计和理解。

2、 简述局部变量的作用。

答：1、实现信息隐藏，使得函数外无法访问该函数内部使用的数据。

2、减少名冲突，一个函数可以为局部变量定义任何合法名字，而不用担心与其他函数的局部变量同名。

2、局部变量的内存空间在栈中分配，函数调用完之后释放，因此，使用局部变量能节省程序的内存空间。

3、 简述变量的生存期和标识符的作用域。

答：变量的生存期指程序运行时一个变量占有内存空间的时间段。C++把变量的生存期分为静态、自动和动态三种。标识符的作用域是指：一个定义了的标识符的有效范围，即该标识符所标识的程序实体能被访问的程序段。在 C++中，根据标识符的性质和定义位置规定了标识符的作用域。作用域分为：全局作用域、文件作用域、局部作用域、函数作用域、函数原型作用域、类作用域、名空间作用域。

4、 全局标识符与局部标识符在哪些方面存在不同？

答：1、作用域不同

2、生存期不同

3、用途不同，全局标识符用于标识共享的实体，而局部标识符用于标识专用的实体。

5、 下面的声明中哪一些是定义性声明？这些定义性声明的非定义性声明是什么？

（1）const int i=1;

（2）static double square(double dbl) { return dbl*dbl; }

（3）char *str;

（4）struct Point;

（5）char* (*pFn)(int(*)(char*,int),char);**

答：1）是。非定义性声明：extern const int i;

2）是。非定义性声明：extern double square(double);

3）是。非定义性声明：extern char *str;

4）不是。

5）是。非定义性声明：extern char* (*pFn)(int(*)(char*,int),char**);

6、 下面的宏 cube1 和函数 cube2 相比，各有什么优缺点？

```
#define cube1(x) ((x)*(x)*(x))
double cube2(double x) { return x*x*x; }
```

答：小型函数的频繁调用会带来程序执行效率的严重下降，宏的出现解决了函数调用效率不高的问题，但宏本身也存在很多问题：（1）宏会出现重复计算，（2）不进行参数类型检查和转换，（3）不利于一些工具对程序的处理。而函数可以很好的处理这些问题。

另外，对于：int a; 当 a 的值很大时，cube1(a)得不到正确结果！（因为结果类型为 int，而如果 a*a*a 的结果超出了 int 型的范围，则结果将会截断！）

7、 编写一个函数 digit(n,k)，它计算整数 n 的从右向左的第 k 个数字。例如：

```
digit(123456,3) = 4
digit(1234,5) = 0
```

答：

```
int digit (int n,int k)
{   for ( int i=1; i<k ; i++)
        n = n/10;
    return n%10;
}
```

8、 分别用函数实现习题 3.8 中的第 1、4、7 和 10 题的程序功能。

答：第 1 题：

```
double Fahrenheit_To_Celsius(double x)
{
  return (x-32)*5/9;
```

```
}
```

第 4 题：

```
int num_of_digits(int gzint)
{  int count = 0;
   if (gzint<0) gzint = - gzint;
   while (gzint!=0)
   {    gzint = gzint / 10;
        count++;
   }
   return count;
}
```

第 7 题：

```
double charge(double weight, double distance)
{
  double money=0;
  if(weight<=15)    money=5;
  else if(weight<=30)   money=9;
  else if(weight<=45)   money=12;
  else if(weight<=60)   money=14+(int)(distance/1000);
  else money=15+(int)(distance/1000)*2;
  return money;
}
```

第 10 题：

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

9、 写出下面程序的执行结果：

```
#include <iostream>
using namespace std;
int count=0;
int fib(int n)
{ count++;
  if (n==1 || n==2)
    return 1;
  else
    return fib(n-1)+fib(n-2);
}
int main()
```

```
{ cout << fib(8);
  cout << ',' << count << endl;
  return 0;
}
```

答：21,41

10、分别写出计算 Hermit 多项式 $H_n(x)$值的迭代和递归函数。$H_n(x)$定义如下：

$$H_0(x) = 1$$
$$H_1(x) = 2x$$
$$H_n(x) = 2x\ H_{n-1}(x) - 2(n-1)\ H_{n-2}(x) \quad (n>1)$$

答：

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

double Hermit_Iterative(int,double);      //迭代方法
double Hermit_Recursion(int,double);     //递归方法

void main()
{ const int n=3;                          //n与x可自行指定
  double x=3.14;
  cout<<Hermit_Iterative(n,x)<<endl
    <<Hermit_Recursion(n,x)<<endl;
}

double Hermit_Iterative(int n,double x)
{ if(n==0)
    return 1;
  else if(n==1)
    return 2*x;
  else
  { double res1=1,res2=2*x;
    double Result=0;
    for (int i=2;i<=n;i++)
    { Result=2*x*res2-2*(i-1)*res1;
      res1=res2;
      res2=Result;
    }
    return Result;
  }
}

double Hermit_Recursion(int n,double x)
{ if(n==0)
    return 1;
  else if(n==1)
    return 2*x;
  else
    return 2*x*Hermit_Recursion(n-1,x)-2*(n-1)*Hermit_Recursion(n-2,x);
}
```

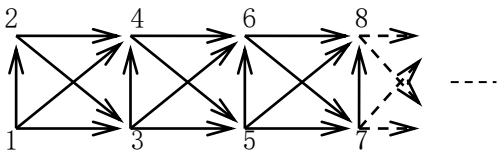
11、写出计算 Ackermann 函数 Ack(m,n)值的递归函数。Ack(m,n)定义如下(m≥0,n≥0)：

```
Ack(0,n) = n+1
Ack(m,0) = Ack(m-1,1)
Ack(m,n) = Ack(m-1,Ack(m,n-1))   (m>0, n>0)
```

答：

```
int Ack(int m,int n)
{ if(m==0)
    return n+1;
  else if(n==0)
    return Ack(m-1,1);
  else
    return Ack(m-1,Ack(m,n-1));
}
```

12、根据下图写一个函数：int path(int n); 用于计算从结点 1 到结点 n（n 大于 1）共有多少条不同的路径。



答：

```
int path(int n)
{ if (n==1) return 1;
  if (n==2) return 1;
  if (n==3) return 2;
  if (n%2 == 0)
    return  path(n-1)+path(n-2)+path(n-3);
  else
    return path(n-1)+path(n-2);
}
```

13、编程解决下面的问题：若一头小母牛，从出生起第四个年头开始每年生一头母牛，按此规律，第 n 年有多少头母牛？

答：除了第一年到第三年外，每一年的母牛数应该是上一年的母牛数加上三年前的母牛数（现在它们是第四年了，要生小牛了！）

```
int f(int n)
{ if (n==1 || n==2 || n==3) return 1;
  return f(n-3)+f(n-1);
}
```

14、假设有三个重载的函数：

```
void func(int,double);
void func(long,double);
```

```
void func(int,char);
```

对下面的函数调用，指出它们分别调用了哪一个重载函数；如果有歧义，指出导致歧义的重载函数定义。

```
func('c',3.0);
func(3L,3);
func("three",3.0);
func(3L,'c');
func(true,3);
```

答：

```
func('c',3.0); 与 void func(int,double); 匹配
func(3L,3); 与 void func(long,double); 匹配
func("three",3.0); 没有与之匹配的函数
func(3L,'c'); 与 void func(long,double); 和 void func(int,char); 均能匹配
func(true,3); 与 void func(int,double); 和 void func(int,char); 均能匹配
```

15、下面的函数定义为什么是正确的？在函数 f 中如何区分（使用）它们？

```
void f()
{ int f;
  .......
}
```

答：两个 f 的作用域不一样，void f()中的 f 为全局作用域，int f; 中的 f 为局部作用域。在函数 f 中如果使用局部变量，则用 f；如果使用函数 f，则用::f。

16、为什么一般把内联函数的定义放在个头文件中？

答：为了防止同一个内联函数的各个定义之间的不一致，往往把内联函数的定义放在某个头文件中，在需要使用该内联函数的源文件中用文件包含命令#include 把该头文件包含进来。由于内联函数名具有文件作用域，因此，不会出现重复定义问题。

17、用循环实现**错误!未找到引用源。**中的辗转相除法计算最大公约数。

答：

```
int gcd(int x, int y)
{ while (y!=0)
  { int t=y;
    y = x%y;
    x = t;
  }
  return x;
}
```

第 5 章 构造数据类型

1、 枚举类型有什么好处？C++对枚举类型的操作有何规定？

答：使用枚举类型有利于提高程序的易读性；使用枚举类型也有利于保证程序的正确性。

首先，可以对枚举类型实施赋值操作，但不同枚举类型之间不能相互赋值，而且不能把一个整型数直接赋值给枚举类型的变量。还可以对枚举类型实施比较运算。还可以对枚举类型实施算术运算，对枚举类型的运算前要转换成对应的整型值，且运算结果类型为算术类型，而且不能对枚举类型的值直接进行输入/输出。

2、 指针类型主要用于什么场合？引用类型与指针类型相比，其优势在哪里？

答：指针类型主要用于参数传递和对动态变量的访问。在 C++中，指针类型还用于访问数组元素，以提高访问效率。

引用类型与指针类型都可以实现通过一个变量访问另一个变量，但访问的语法形式不同：引用是采用直接访问形式，指针则采用间接访问形式。在作为函数参数类型时，引用类型参数的实参是一个变量，而指针类型参数的实参是一个变量的地址。

除了在定义时指定的被引用变量外，引用类型变量不能再引用其他变量；而指针变量定义后可以指向其他同类型的变量。因此，引用类型比指针类型要安全。

引用类型的间接访问对使用者而言是透明的。

3、 写出下面程序的运行结果：

```
#include <iostream>
using namespace std;
void f(int &x,int y)
{ y = x + y;
  x = y % 3;
  cout << x << '\t' << y << endl;
}
int main()
{ int x=10, y=19;
  f(y,x);
```

```
    cout << x << '\t' << y << endl;
    f(x,x);
    cout << x << '\t' << y << endl;
    return 0;
  }
```

答：
```
 2       29
10        2
 2       20
 2        2
```

4、从键盘输入某个星期每一天的最高和最低温度,然后计算该星期的平均最低和平均最高温度并输出之。

解：

```
#include <iostream>
using namespace std;
enum Day {SUN,MON,TUE,WED,THU,FRI,SAT};
int main()
{ double max, min, maxsum=0, minsum=0;
  for (Day d = SUN; d <= SAT; d=(Day)(d+1))
  { cout << "Please input ";
    switch(d)
    { case SUN : {cout << "Sunday"; break;}
      case MON : {cout << "Monday"; break;}
      case TUE : {cout << "Tuesday"; break;}
      case WED : {cout << "Wednesday"; break;}
      case THU : {cout << "Thursday"; break;}
      case FRI : {cout << "Friday"; break;}
      case SAT : {cout << "Saturday"; break;}
    }
    cout << "'s temperature(max  min) : " << endl;
    cin >> max >> min;
    maxsum += max;
    minsum += min;
  }
  cout << "The average temperature of maxism is : " << maxsum/7.0 << endl;
  cout << "The average temperature of minism is : " << minsum/7.0 << endl;
  return 0;
}
```

5、 编写一个函数，判断其 int 型参数值是否是回文数。回文数是指从正向和反向两个方向读数字都一样，例如，9783879 就是一个回文数。

解：

```
bool is_huiwen(int num)
{ char wei[100], i=0;
  while (num != 0)
  { wei[i] = num % 10;
    num /= 10;
```

```
    i++;
  }
  for (int j = 0; j <= i/2; j++)
  { if (wei[j] != wei[i-j-1])
      return false;
  }
  return true;
}
```

6、 编写一个函数 int_to_str(int n, char str[])，把一个 int 型数（由参数 n 表示）转换成一个字符串（放在 str 中）。

解：

```
void int_to_str(int num, char *str)
{ char c;
  int i=0;
  while (num != 0)
  { str[i] = num%10 + '0';
    num /= 10;
    i++;
  }
  str[i] = '\0';
  for (int j = 0; j < i/2; j++)
  { c = str[j];
    str[j] = str[i-j-1];
    str[i-j-1] = c;
  }
}
```

7、 编写一个函数计算一元二次方程的根。要求：方程的系数和根均用参数传递机制来传递。

解：

```
int qiugen(double a, double b, double c, double &x1, double &x2)
{ int i = b*b-4*a*c;
  if (i>=0)
  { x1 = (sqrt(i)-b)/(2*a);
    x2 = (0-sqrt(i)-b)/(2*a);
    return 1;
  }
  else
  { x1 = (0-b)/(2*a);
    x2 = sqrt(0-i)/(2*a);
    return 0;
  }
}
```

8、 编写一个程序，从键盘输入一个字符串，分别统计其中的大写字母、小写字母以及数字的个数。

解:

```
#include <iostream>
using namespace std;
int main()
{ char str[100];
  cout << "Please input a string:\n";
  cin >> str;
  int count_lower=0, count_upper=0, count_num=0;
  for (int i=0; str[i] != '\0'; i++)
  { if (str[i] >= 'A' && str[i] <= 'Z')
      count_upper++;
    else if (str[i] >= 'a' && str[i] <= 'z')
      count_lower++;
    else if (str[i] >= '0' && str[i] <= '9')
      count_num++;
  }
  cout << count_upper << '\t' << count_lower << '\t' << count_num << endl;
  return 0;
}
```

9、 设有一个矩阵：$\begin{bmatrix}0 & 2 & 1\\ 1 & 0 & 2\\ 1 & 2 & 0\end{bmatrix}$，现把它放在一个二维数组 a 中。写出执行下面的语句之后 a 的值：

```
for (int i=0; i<=2; i++)
  for (int j=0; j<=2; j++)
    a[i][j] = a[a[i][j]][a[j][i]];
```

解：0 2 0

2 0 0

2 2 0

10、实现下面的数组元素交换位置函数：

```
void swap(int a[], int m, int n);
```

该函数能够把数组 a 的前 m 个元素与后 n 个元素交换位置，即，

交换前：$a_1,a_2,...,a_M,\ a_{M+1},a_{M+2},...,a_{M+N}$

交换后：$a_{M+1},a_{M+2},...,a_{M+N},\ a_1,a_2,...,a_M$

要求：除数组 a 外，不得引入其它数组。

解：

```
void swap(int a[], int m, int n);
{ for (int i=0; i<m; i++)
  { int t = a[0];
    for (int j=1; j<m+n; j++)
```

```
      a[j-1] = a[j];
    A[m+n-1] = t;
  }
}
```

11、编写一个程序，计算一个矩阵的鞍点。矩阵的鞍点是指矩阵中的一个位置，该位置上的元素在其所在的行上最大、列上最小。（一个矩阵也可能没有鞍点。）

```
#include <iostream>
using namespace std;
#define M 3
#define N 3
int main()
{ int a[M][N]; //存放矩阵
  int i,j;
  //输入矩阵元素
  for (i=0; i<M; i++)
    for (j=0; j<N; j++)
      cin >> a[i][j];

  int lin_max[M], //存放各行的最大元素
      col_min[N]; //存放各列的最小元素
  //计算每行的最大元素
  for (i=0; i<M; i++)
  { int max=a[i][0];
    for (j=1; j<N; j++)
      if (a[i][j]>max) max=a[i][j];
    lin_max[i] = max;
  }
  //计算每列的最小元素
  for (j=0; j<N; j++)
  { int min=a[0][j];
    for (i=1; i<M; i++)
      if (a[i][j]<min) min=a[i][j];
    col_min[j] = min;
  }

  //求鞍点
  for (i=0; i<M; i++)
  { for (j=0; j<N; j++)
    { if (a[i][j] == lin_max[i] && a[i][j] == col_min[j])
      { cout << "鞍点是：" << i << ',' << j << endl;
        return 0;
      }
    }
  }
  cout << "没有鞍点\n";
  return 0;
}
```

12、编程实现：在一个由 N×N（N 为大于 1 的奇数）个方格组成的方阵中，填入 1、2、3、...、N^2

各个数，使得每一行、每一列以及两个对角线上数的和均相等（奇数幻方问题）。例如，下面是一个 3×3 的幻方：

8	1	6
3	5	7
4	9	2

（提示：把 1 填在第一行最中间的格子中，然后按下面的方法依次来填其它的数：如果当前格子是方阵中最右上角的格子，则把下一个数填在下一行的同一列格子中；否则，如果当前格子在第一行上，则把下一个数填在下一列的最后一行格子中；否则，如果当前格子在最后一列上，则把下一个数填在上一行的第一列格子中；否则，如果当前格子的右上角格子里没有数，则在其中填入下一个数，否则把下一个数填在下一行的同一列格子中。）

答：

```
#include <iostream>
using namespace std;
#define N 3
int a[N][N];
int main()
{ int i,j;
  for (i=0; i<N; i++)
    for (j=0; j<N; j++)
      a[i][j] = 0;

  i = 0; j = N/2;
  a[i][j] = 1;
  for (int k=2; k<=N*N; k++)
  { if (i==0 && j==N-1)
      i++;
    else if (i==0)
    { i = N-1;
      j = j++;
    }
    else if (j==N-1)
    { i--;
      j = 0;
    }
    else if (a[i-1][j+1] == 0)
    { i--;
      j++;
    }
    else
      i++;
    a[i][j] = k;
  }

  for (i=0; i<N; i++)
  { for (j=0; j<N; j++)
```

```
      cout << a[i][j] << '\t';
    cout << endl;
  }
  return 0;
}
```

13、实现 strlen、strcpy、strncpy、strcat、strncat、strcmp 以及 strncmp 函数。

答：

```
int strlen2(const char s[])
{ int n=0;
  for (const char *p=s; *p != '\0'; p++) n++;
  return n;
}
char *strcpy2(char dst[],const char src[])
{ char *p1;
  const char *p2;
  for (p1=dst,p2=src; *p2 != '\0'; p1++,p2++) *p1 = *p2;
  *p1 = '\0';
  return dst;
}
char *strncpy2(char dst[],const char src[],int n)
{ char *p1;
  const char *p2;
  for (p1=dst,p2=src; n != 0 && *p2 != '\0'; p1++,p2++,n--) *p1 = *p2;
  if (n != 0) *p1 = '\0';
  return dst;
}
char *strcat2(char dst[],const char src[])
{ char *p1;
  const char *p2;
  for (p1=dst; *p1 != '\0'; p1++) ;
  for (p2=src; *p2 != '\0'; p1++,p2++) *p1 = *p2;
  *p1 = '\0';
  return dst;
}
char *strncat2(char dst[],const char src[],int n)
{ char *p1;
  const char *p2;
  for (p1=dst; *p1 != '\0'; p1++) ;
  for (p2=src; n != 0 && *p2 != '\0'; p1++,p2++,n--) *p1 = *p2;
  if (n != 0) *p1 = '\0';
  return dst;
}
int  strcmp2(const char s1[],const char s2[])
{ for (const char *p1=s1,*p2=s2; *p1 != '\0' && *p2 != '\0'; p1++,p2++)
  { if (*p1 > *p2)
      return 1;
    else if (*p1 < *p2)
      return -1;
  }
  if (*p1 == '\0' && *p2 == '\0')
```

```
    return 0;
  else if (*p1 == '\0')
    return -1;
  else
    return 1;
}
int  strncmp2(const char s1[],const char s2[],int n)
{ for (const char *p1=s1,*p2=s2; n != 0 && *p1 != '\0' && *p2 != '\0'; p1++,p2++,n--)
  { if (*p1 > *p2)
      return 1;
    else if (*p1 < *p2)
      return -1;
  }
  if (n == 0 || *p1 == '\0' && *p2 == '\0')
    return 0;
  else if (*p1 == '\0')
    return -1;
  else
    return 1;
}
```

14、编写一个函数 int squeeze(char s1[], const char s2[])，它从字符串 s1 中删除所有在 s2 里出现的字符，函数返回删除的字符个数。

解：

```
int squeeze(char s1[], const char s2[])
{ int count=0,i=0;
  while (s1[i]!='\0')
  { for (int j=0; s2[j]!='\0' && s1[i]!=s2[j]; j++) ;
    if (s2[j] == '\0')
      i++;
    else
    { for (int k=i+1; s1[k]!='\0'; k++)
        s1[k-1] = s1[k];
      s1[k-1]='\0';
      count++;
    }
  }
  return count;
}
```

15、编写一个函数 find_replace_str，其原型如下：

```
int find_replace_str(char str[],
                     const char find_str[],
                     const char replace_str[]);
```

要求：该函数能够完成把字符串 str 中的所有子串 find_str 都替换成字符串 replace_str，返回值为替换的次数。

解：

```
void find_replace_str(char str[],const char find_str[],const char replace_str[])
```

```
{ int index=0, //str中的当前处理位置
      find_len=strlen(find_str),
      replace_len=strlen(replace_str),
      offset=find_len-replace_len;

  while (strlen(str+index) >= find_len)
  { if (strncmp(str+index,find_str,find_len) == 0)
    { if (offset < 0) //把字符串剩余部分往后移-offset个位置
      { int n=strlen(str+index)-find_len+1; //剩余部分的字符个数+1（'\0'）
        for (int i=strlen(str); n>0; i--,n--)
         str[i+(-offset)] = str[i];
      }
      else if (offset > 0) //把字符串剩余部分往前移offset个位置
      { int n=strlen(str+index)-find_len+1;//剩余部分的字符个数+1（'\0'）
        for (int i=index+find_len; n>0; i++,n--)
        str[i-offset] = str[i];
      }
      for (int i=0; i<replace_len; i++) //复制被替换成的串到str
        str[index+i] = replace_str[i];
      index += replace_len;
    }
    else
      index++;
  }
}
```

16、编写一个程序，从键盘输入一批学生的成绩信息，每个学生的成绩信息包括：学号、姓名以及 8 门课的成绩。然后按照平均成绩由高到低顺序输出学生的学号、姓名以及平均成绩。

解：

```
#include <iostream>
using namespace std;
struct Student
{ char id[11];
  char name[9];
  double scores[9];
};

int main()
{ int n;

  cout << "请输入学生人数：";
  cin >> n;

  Student *students=new Student[n]; //创建动态数组以存放学生信息。

  //输入每个学生的信息。
  int i,j;
  for (i=0; i<n; i++)
  { cout << "学号：";
    cin >> students[i].id;
    cout << "姓名：";
    cin >> students[i].name;
```

```
    cout << "8 门课成绩：";
    students[i].scores[8] = 0.0;
    for (j=0; j<8; j++)
    { cin >> students[i].scores[j];
      students[i].scores[8] += students[i].scores[j];
    }
    students[i].scores[8] /= 8; //平均成绩
  }

  //根据平均成绩对学生信息进行排序。
  for (i=n; i>1; i--)
  { bool exchange=false;
    for (j=1; j<i; j++)
    { if (students[j].scores[8] > students[j-1].scores[8])
      { Student temp=students[j];
        students[j] = students[j-1];
        students[j-1] = temp;
        exchange = true;
      }
    }
    if (!exchange) break;
  }

  //输出排序后的学生信息
  for (i=0; i<n; i++)
  { cout << students[i].id << ','
         << students[i].name << ','
         << students[i].scores[8] << endl;
  }
  return 0;
}
```

17、下面的交换函数正确吗？

```
void swap_ints(int &x, int &y)
{ int &tmp=x;
  x = y;
  y = tmp;
}
```

答：不正确，因为 temp 为引用类型，它与 x 占有相同的空间，当执行“x=y;”操作之后，temp 的值已不是 x 原来的值了！按照这个函数，x 和 y 的值会相等并且等于 y 的值，不能实现将 x 和 y 交换的目的。

18、写一个函数 map，它有三个参数。第一个参数是一个一维 double 型数组，第二个参数为数组元素个数，第三个参数是一个函数指针，它指向带有一个 double 型参数、返回值类型为 double 的函数。函数 map 的功能是把数组的每个元素替换成：用它原来的值（作为参数）调用第三个参数所指向的函数得到的值。

解：

```
void map(double d[], int n, double (*fp)(double d))
{ for (int i=0; i<n; i++)
    d[i]=(*fp)(d[i]);
  return;
}
```

19、把在链表中插入一个新结点的操作写成一个函数：

```
bool insert(Node *&h,int a,int pos);
```

其中，h 为表头指针，a 为要插入的结点的值，pos（≥0）表示插入位置。当 pos 为 0 时表示在表头插入；否则，表示在第 pos 个结点的后面插入。操作成功返回 true，否则返回 false。

解：

```
struct Node
{ int value;
   Node *next;
};
bool insert(Node *&h,int a,int pos)
{ Node *q;
  if (pos==0)
  { q=new Node;
    q->value = a;
    q->next=h;
    h=q;
    return true;
  }
  else
  { Node *p=h;
    int i=1;
    while (p!=NULL && i<pos)
    { p=p->next;
      i++;
    }
    if (p!=NULL)
    { q=new Node;
      q->value = a;
      q->next=p->next;
      p->next=q;
      return true;
    }
    else
      return false;
  }
}
```

20、把在链表中删除一个结点的操作写成一个函数：

```
bool remove(Node *&h,int &a, int pos);
```

其中，h 为表头指针，a 用于存放删除的结点的值，pos（>0）表示删除结点的位置。操作成功返回 true，否则返回 false。

解：

```
struct Node
{ int value;
   Node *next;
};
bool remove(Node *&h,int &a, int pos)
{ Node *p=h, *q=NULL;
  int i=1;
  while (p!=NULL && i<pos)
  { q=p;
    p=p->next;
    i++;
  }
  if (p!=NULL)
  { a=p->value;
    if (q!=NULL)
      q->next=p->next;
    else
      h = p->next;
    delete p;
    return true;
  }
  else
    return false;
}
```

21、编写一个程序，首先建立两个集合（从键盘输入集合的元素），然后计算这两个集合的交集、并集以及差集，最后输出计算结果。要求用链表实现集合的表示。

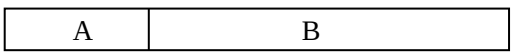
解：

```
#include <iostream>
using namespace std;
struct Node
{ int value;
   Node *next;
};
bool find(Node *h,int x) //在h中查找x
{ for (Node *p=h; p!=NULL; p=p->next)
    if (p->value == x) return true;
  return false;
}
void insert(Node *&h, int x) //在h中增加一个元素
{ Node *p=new Node;
  p->value = x;
  p->next = h;
  h = p;
}
Node *input() //建立集合
{ Node *h=NULL;
```

```
  int x;
  for (cin >> x; x != -1; cin >> x)
  { if (find(h,x)) continue;
    insert(h,x);
  }
  return h;
}
Node *set_union(Node *h1, Node *h2) //集合“并”
{ Node *h=NULL,*p;
  //生成一个与h1 一样的集合h
  for (p=h1; p!=NULL; p=p->next)
    insert(h,p->value);
  //把h2 加入到h中（去重）
  for (p=h2; p!=NULL; p=p->next)
  { if (!find(h1,p->value))
      insert(h,p->value);
  }
  return h;
}
Node *set_intersection(Node *h1, Node *h2) //集合“交”
{ Node *h=NULL;
  for (Node *p=h1; p!=NULL; p=p->next)
  { if (find(h2,p->value))
      insert(h,p->value);
  }
  for (Node *q=h2; q!=NULL; q=q->next)
  {   if (!find(h1,q->value))
       insert(h,q->value);
  }
  return h;
}
Node *set_difference(Node *h1, Node *h2) //集合“差”
{ Node *h=NULL;
  for (Node *p=h1; p!=NULL; p=p->next)
  { if (!find(h2,p->value))
      insert(h,p->value);
  }
  return h;
}
void output(Node *h) //输出集合的所有元素
{ for (Node *p=h; p!=NULL; p=p->next)
    cout << p->value << ' ';
   cout << endl;
}
void remove(Node *&h) //删除集合
{ while (h != NULL)
  { Node *p=h;
    h = h->next;
    delete p;
  }
}
int main(int argc, char* argv[])
```

```
{ Node *set1,*set2,*set3,*set4,*set5;

  set1 = input();
  set2 = input();
  set3 = set_union(set1,set2);
  set4 = set_intersection(set1,set2);
  set5 = set_difference(set1,set2);

  output(set1);
  output(set2);
  output(set3);
  output(set4);
  output(set5);

  remove(set1);
  remove(set2);
  remove(set3);
  remove(set4);
  remove(set5);

  return 0;
}
```

22、在排序算法中，有一种排序算法（插入排序）是：把待排序的数分成两个部分：



其中，A 为已排好序的数，B 为未排好序的数，初始状态下，A 中只有一个元素。该算法依次从 B 中取数插入到 A 中的相应位置，直到 B 中的数取完为止。请在链表表示上实现上述的插入排序算法。

解：

```
struct Node
{ int content;
   Node *next;
};
void insert_sort(Node *&h)
{ if (h == NULL) return;
  Node *q=h->next; //q指向B部分的第一个元素
  h->next = NULL; //h指向A部分的第一个元素（初始状态下，A部分只有一个元素）
  while (q != NULL)//对第二部分的元素进行循环
  { //从B部分取一个元素
    Node *q1=q; //q1 指向取到的元素
    q = q->next; //从B中去掉一个元素

    //循环在A部分中从头开始找一个元素（p指向它），使得（q1->content）小于（p->content）
    Node *p=h,*p1=NULL; //p指向A部分第一个元素；p1 指向p指向的前一个结点，初始化为空
    while (p != NULL && q1->content > p->content)
    { p1 = p;
      p = p->next;
    }
```

```
      if (p1 != NULL)
      { q1->next = p;
        p1->next = q1;
      }
      else
      { q1->next = h;
        h = q1;
      }
    }
  }
```

23、下面的求 n!的函数有什么问题？

```
int factorial(int &n)
{ int f=1;
  while (n > 1)
  { f *= n;
    n--;
  }
  return f;
}
```

答：有函数副作用的问题。函数执行结束后，调用该函数的实参值被改变了（通过形参，变为 1）。

24、写出例 5-11 名表查找中折半查找的递归函数。

解：

```
#include <cstring>
using namespace std;
int b_search(char key[], TableItem t[], int first, int last)
{ if (first > last) return -1;
  int index=(first+last)/2;
  int r=strcmp(key,t[index]);
  if (r == 0)  // key等于t[index]
    return index;
  else if (r > 0)  // key大于t[index]
    return b_search(key,t,index+1,last);
  else  //key小于t[index]
    return b_search(key,t,first,index-1);
}
int binary_search(char key[], TableItem t[], int num_of_items)
{ return b_search(key,t,0,num_of_items-1);
}
```

25、编写一个程序解八皇后问题。八皇后问题是：设法在国际象棋的棋盘上放置八个皇后，使得其中任何一个皇后所处的“行”、“列”以及“对角线”上都不能有其它的皇后。

解：

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

```
bool a[8];  //a[i]表示第i行是否可以放皇后
bool b[15]; //b[k]表示“从左下往右上”('/')的第k个对角线是否可以放皇后
bool c[15]; //c[k]表示“从左上往右下”('\')的第k个对角线是否可以放皇后
//与棋盘第i行、第j列的格子所对应的行和对角线为：a[i],b[i+j],c[j-i+7]

int  x[8];  //x[j]表示第j列上皇后的位置（所在的行）。

bool try_by_col(int j)
{ for (int i=0; i<8; i++) //选择第j列中可放皇后的行，然后递归选择第j+1 列可放皇后的行...。
  { if (a[i] && b[i+j] && c[j-i+7]) //第j列第i行的位置所在的行以及两个对角线上无皇后。
    { x[j] = i; //设置第j列皇后的位置。
      a[i] = b[i+j] = c[j-i+7] = false; //把第j列皇后所在的行以及两个对角线设为已占用。
      if (j == 7 || try_by_col(j+1)) //是最后一列或第j+1 列皇后位置选择成功。
        return true;
      else //第j+1 列皇后位置选择失败。
        a[i] = b[i+j] = c[j-i+7] = true; //取消第j列皇后位置，准备选择下一个位置。
    }
  }
  return false;
}

int main()
{ int i,j,k;
  //初始化：所有行以及对角线可放皇后。
  for (i=0; i<8; i++)
    a[i]=true;
  for (k=0; k<15; k++)
    b[k]=true;
  for (k=0; k<15; k++)
    c[k]=true;

  if (try_by_col(0)) //从第 0 列开始尝试放皇后的位置。
  { //x[0],x[1],...,x[7]分别为第一列、第二列、...、第七列上皇后的位置（所在的行）
    for (j=0; j<8; j++) //按列输出皇后
    { for (i=0; i<x[j]; i++) cout << "|_";
      cout << "|Q|";
      for (i=x[j]+1; i<8; i++) cout << "_|";
      cout << endl;
    }
  }
  cout<<endl;
  return 0;
}
```