程序设计教程

——用 C++语言编程

（习题解答）

目 录

第 1 章 概述

1、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。

2、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。

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、简述软件生存周期。

答：一个软件从无到有，一直到最后的消亡（报废），通常要经历一个过程，这个过程称为软件生存周期。它分成若干阶段：软件需求分析、软件设计、编程实现、测试以及运行与维护。软件需求分析的主要任务是明确待实现的软件要解决什么问题，即做什么，给出软件的需求说明。软件设计是根据软件的需求说明给出抽象的解决方案（设计说明），它包括概要设计和详细设计。概要设计是指软件的整体结构设计；详细设计是指抽象的数据结构和算法描述。编程实现是指根据软件设计说明，采用某种程序设计语言编写程序。测试是对编写好的程序进行测试，确认其是否满足所规定的需要。运行与维护是指使用软件并在使用过程中发现和改正程序中的错误。

6、机器语言、汇编语言以及高级语言的不同之处是什么？

答：机器语言采用指令编码和数据的存储位置来表示操作以及操作数。机器语言写的程序可以直接在计算机上执行。

汇编语言是用符合名来表示操作和操作数位置，以增加程序的易读性，它与机器语言一一对应。用汇编语言写的程序必须通过汇编程序翻译成机器语言程序才能执行。

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，一条语句可能会对应多条机器指令。用高级语言书写的程序需要通过编译程序或解释程序翻译成机器语言程序才能在计算机上运行。

7、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。

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、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

10、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++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。

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。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。由于十进制小数转成double型编码的问题导致精确度不够，通过查看结果内存内的内容，最终结果比 3.0 略小（为 3.0-4.44089e-016），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;
```

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 a 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;
        exit(-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;
```

```
        exit(-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()
{  for (char 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 a 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，检查圆括号配对情况。输出：配对、多左括号或多右括号。

解：假设输入的算术表达式以‘ # ’结束。

```
#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，对其中的“ >= ”进行计数。

解：假设输入的字符串以‘ # ’结束。

```
#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）
75 及以上	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 << 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;
}
```

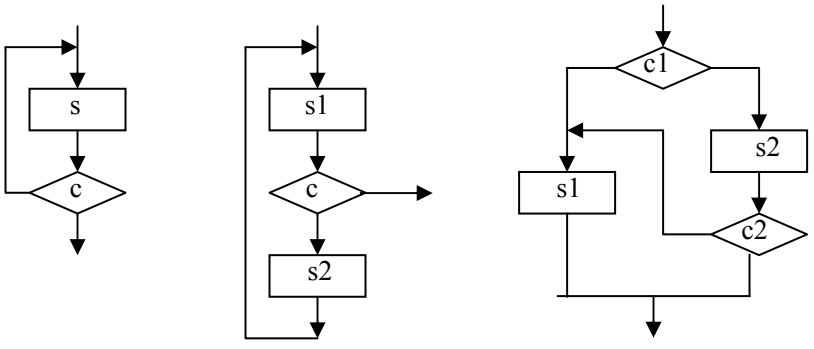
12、将下面的 for 循环重写为等价的 while 循环。

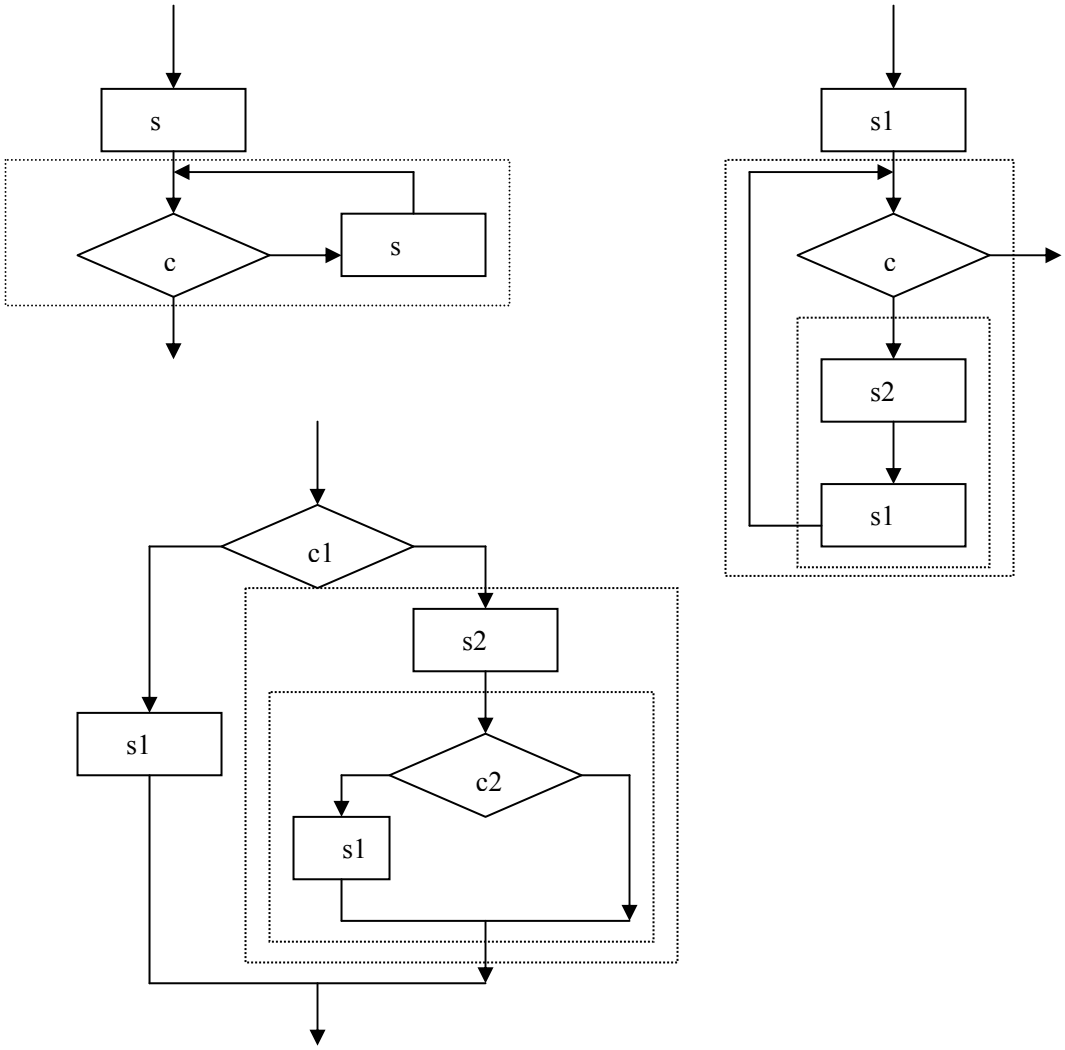
```
for (i=0; i<max_length;i++)
  if (input_line[i]== '?')  quest_count++;
```

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

14、 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功能抽象，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。

15、 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。

16、 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。

17、 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。

18、 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**);

19、 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！)

20、 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;
}
```

21、 分别用函数实现习题 0 中的第 1、7、8 和 10 题的程序功能。

答：**第 1 题：**

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

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 8 题：

```
#include <iostream>
#include <iomanip>
#include <math.h>
using namespace std;

double PI()
{ double pi=0,double x=1;
```

```
  int k=1,sign=1;
  while(fabs(x)>1e-8)
  { pi += x;
    k += 2;
    sign *= -1;
    x = sign/double(k);
  }
  return pi*4;
}

void main()
{ cout<<"The Pi is: "
    <<setiosflags(ios::fixed)
    <<setprecision(8)
    <<PI() <<endl;
}
```

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;
}
```

22、 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

23、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);
}
```

24、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));
}
```

25、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
```

26、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。

27、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。

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、 引用类型与指针类型相比，其优势在哪里？

答：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，把一个 int 型数转换成一个字符串。

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;
}
```

12、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++;
  }
}
```

13、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。

14、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;
}
```

15、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;
}
```

16、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：

```
bool insert(Node *&h,int a,int pos)
{ Node *q=new Node(a);
  if (pos==0)
  { 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->next=p->next;
      p->next=q;
      return true;
    }
    else
```

```
        return false;
    }
  }
```

17、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：

```
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->id;
    if (q!=NULL)
      q->next=p->next;
    else
      h = p->next;
    delete p;
    return true;
  }
  else
    return false;
}
```

18、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：

A	B

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：

```
void insert_sort(Node *&h)
{ if (h == NULL) return;
  Node *q=h->next;
  h->next = NULL;
  while (q != NULL)
  { Node *p=h,*p1=NULL,*q1=q;
    q = q->next;
    while (p != NULL && q1->id > p->id)
    { p1 = p;
```

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

19、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）。

20、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;
}
```

第 6 章 数据抽象——类

1、 从概念上讲，抽象与封装有什么不同？

答：处理大而复杂问题的重要手段是抽象：强调事物本质的东西。对程序抽象而言，一个语言结构的抽象强调的是该结构外部可观察到的行为，与该结构的内部实现无关。抽象包括过程抽象(Procedural Abstraction)和数据抽象(Data Abstraction)。

封装是把一个语言结构的具体实现细节作为一个黑匣子对该结构的使用者隐藏起来的一种机制，从而符合信息隐藏(Information Hiding)原则。封装包括过程封装和数据封装。

封装考虑的是内部实现，抽象考虑的是外部行为。

2、 对于一个类定义，哪些部分应放在头文件（.h）中，哪些部分放在实现文件（.cpp）中？

答：一般情况下，类定义放在头文件（.h）中，类外定义的成员函数放在实现文件（.cpp）中。

3、 对类成员的访问，C++提供了哪些访问控制？

答：在 C++的类定义中，可以用访问控制修饰符 public,private 和 protected 对类成员的访问进行限制。public：访问不受限制。private：只能在本类和友元中访问。protected：只能在本类、派生类和友

元中访问。

4、 在 C++中，this 指针的作用是什么？

答：C++中类定义中声明的非静态数据成员对该类的每个对象都有一个拷贝，而成员函数是被该类的每个对象共享的，为了确定成员函数中用到的数据成员是属于哪一个对象的，C++采用了一个 this 指针解决这个问题：每个成员函数都有一个缺省的形式参数 this，其类型为指向该类对象的指针；调用一个对象的成员函数时，实现系统会把该对象的地址传给成员函数。在成员函数中访问类中定义的数据成员时，实际访问的是 this 所指向的对象的数据成员。

5、 构造函数成员初始化表可以包含哪些内容？

答：在定义构造函数时，函数头和函数体之间可以加入一个对数据成员进行初始化的表，用于对数据成员进行初始化，特别是对常量和引用数据成员进行初始化。如需要调用成员对象和基类的非默认构造函数，也需要在对象类构造函数的成员初始化表中指定。

6、 拷贝构造函数的作用是什么？何时会调用拷贝构造函数？

答：拷贝构造函数用于在创建对象时用另一个同类的对象对其初始化。一般来说，有三种情况将调拷贝构造函数：

1） 定义对象时

2） 把对象作为值参数传给函数时

3） 把对象作为返回值时

7、 对于一个类 A，为什么下面的构造函数不合法？

```
A(A x);
```

答：因为：对于下面的程序将会递归调用 A(A x)：

A a;

A b(a); //调用 b.A(a)时，将会调用 x.A(a)，a.A(x)，…，从而形成递归！

8、 在创建包含成员对象的类的对象时，为什么要首先初始化成员对象？

答：因为在包含成员对象的构造函数中可能用到成员对象，如果这时成员对象未初始化，则会用到未初始化的对象！例如：

```
class A
{ …
 void f();
};
```

```
class B
{  A a;
  public:
   B()
   { a.f(); //如果这时a未初始化，则a.f()没有意义。
   }
};
B b; //调用b的构造函数。
```

9、 何时需要定义析构函数？

答：如果对象在创建后申请了一些资源并且没有归还这些资源，则应定义析构函数来在对象消亡时归还对象申请的资源。

10、静态数据成员的作用是什么？静态数据成员如何初始化？

答：在 C++中采用静态成员来解决同一个类的对象共享数据的问题。类定义中的静态数据成员对该类的所有对象只有一个拷贝，即它们被该类的所有对象所共享。另外，虽然全局变量也能起到共享的效果，但静态数据成员较为安全，因为静态数据成员还可以受到类的访问控制的保护。

静态数据成员必须要在类的外部给出它们的定义，定义时可以进行初始化。

11、类作用域与局部作用域有什么不同？

答：当标识符的声明出现在由一对花括号所括起来的一段程序内（块，block）时，该（局部）标识符的作用于从声明点开始，到块结束处为止，该作用域的范围具有局部性。而类作用域是指类定义和相应的成员函数定义范围。在该范围内，一个类的成员函数对同一类的数据成员具有无限制访问权。

12、在定义一个包含成员对象的类时，表示成员对象的数据成员何时定义为成员对象指针和成员对象本身？

答：当一个对象 b（类为 B）作为另一个对象 a（类为 A）的组成部分，并且组成关系不会发生变化（一个学生与他的姓名，出生日期之间的关系），则应把 b 定义为 A 类的成员对象。当对象 b 作为另一个对象 a 的组成部分，但组成关系会发生变化（如项目组和项目成员的组成关系），或者对象 b 不是 a 的一部分，但它们之间有关联（部门经理和成员之间的管理与被管理关系），这时应把 b 定义为 A 类的成员对象指针。

13、定义一个描述二维坐标系中点对象的类 Point，它具有下述成员函数：

（1）double r(); //计算极坐标的极半径

（2）double theta(); //计算极坐标的极角

（3）double distance(const Point& p); //计算与点 p 的距离

（4）Point relative(const Point& p); //计算相对于 p 的相对坐标

（5）bool is_above_left(const Point& p); //判断是否在点 p 的左上方

答：

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

class Point
{
public:
  Point(){x=y=0;}
  Point(double a,double b){x=a;y=b;}
  double r() //计算极坐标的极半径
  {
    return sqrt(x*x+y*y);
  }
  double theta() //计算极坐标的极角
  {
    return atan(y/x);
  }
  double distance(const Point& p) //计算与点p的距离
  {
    return sqrt((x-p.x)*(x-p.x)+(y-p.y)*(y-p.y));
  }
  Point relative(const Point& p) //计算相对于p的相对坐标
  {
    double xTemp,yTemp;
    xTemp=x-p.x;
    yTemp=y-p.y;
    return Point(xTemp,yTemp);
  }
  bool is_above_left(const Point& p) //判断是否在点p的左上方
  {
    if((x==p.x)&&(y==p.y))
      return false;
    else if((x<p.x)&&(y>p.y))
      return true;
    else
      return false;
  }
private:
  double x;
  double y;
};
```

14、定义一个时间类 Time，它能表示：时、分、秒，并提供以下操作：

（1）Time(int h, int m, int s); //构造函数

（2）set(int h, int m, int s); //调整时间

（3）increment(); //时间增加一秒。

（4）display(); //显示时间值。

（5）equal(Time &other_time); //比较是否与某个时间相等。

（6）less_than(Time &other_time); //比较是否早于某个时间。

答：

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

class Time
{
public:
  Time()
  { hour=minute=second=0;
  }
  Time(int h,int m,int s)
  { hour=h;minute=m;second=s;
  }
  void set(int h, int m, int s)
  { hour=h;
    minute=m;
    second=s;
  }
  void increment()
  { if (second==59)
    { if (minute==59)
      { if (hour==23)
        { hour=0;
          minute=0;
          second=0;
        }
        else
        { hour++;
          minute=0;
          second=0;
        }
     }
      else
      { minute++;
        second=0;
      }
   }
    else
      second++;
  }
  void display() const
  { cout<<hour<<':'<<minute<<':'<<second;
  }
  bool equal(const Time &time2) const
```

```
  { if (hour == time2.hour && minute == time2.minute && second == time2.second)
      return true;
    else
      return false;
  }
  bool less_than(const Time &time2) const
  { if (hour < time2.hour ||
       hour == time2.hour && minute < time2.minute ||
      hour == time2.hour && minute == time2.minute && second < time2.second)
      return true;
    else
      return false;
  }
protected:
  int hour;
  int minute;
  int second;
};
```

15、定义一个日期类 Date，它能表示：年、月、日。为其设计一个成员函数 increment，它能把某个日期增加一天。

答：

```
#include <iostream>
using namespace std;
class Date
{ public:
    Date(int y, int m, int d)
    { year = y; month = m; day = d;
    }
    void increment();
  protected:
    int year,month,day;
};

void Date::increment()
{ int d;
  switch (month)
  { case 1: case 3: case 5: case 7: case 8: case 10: case 12:
      d = 31;
      break;
    case 4: case 6: case 9: case 11:
      d = 30;
      break;
    case 2:
      if (year%400 == 0 || year%4 == 0 && year%100 != 0) //闰年
        d = 29;
      else
        d = 28;
  }
```

```
  if (day < d)
    day++;
  else if (month != 12)
  { day = 1;
    month++;
  }
  else
  { day = 1;
    month = 1;
    year++;
  }
}
```

或：

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

class Date
{
public:
  Date(){year=month=day=0;}
  Date(int a,int b,int c){year=a;month=b;day=c;}
  void increment();
protected:
  int year,month,day;
  bool Legal(int y, int m, int d);
  bool IsLeapYear(int y);
};

void Date::increment()
{
  if(Legal(year,month,day+1))
    day++;
  else if(Legal(year,month+1,1))
  { month++,day=1; }
  else if(Legal(year+1,1,1))
  { day=1,month=1,year++; }
}

bool Date::Legal(int y, int m, int d)
{
  if(y>9999||y<1||d<1||m<1||m>12)
    return false;

  int dayLimit=31;
  switch(m)
  { case 4: case 6: case 9: case 11:
      dayLimit--;
  }
  if(m==2) dayLimit = IsLeapYear(y) ? 29 : 28;
```

```
  return (d>dayLimit)? false : true;
}

bool Date::IsLeapYear(int y)
{
  return !(y%4)&&(y%100)||!(y%400);
}
```

16、为错误！未找到引用源。**中的字符串类 String 增加下面的成员函数：**

（1）bool is_substring(const char *str); //判断 str 是否为当前字符串的子串。

（2）bool is_substring(const String& str); //判断 str 是否为当前字符串的子串。

（3）String substring(int start, int length); //取从位置 start 开始、长度为 length 的子串。

（4）int find_replace_str(const char *find_str, const char *replace_str);

//查找所有子串find_str并替换成replace_str，返回替换的次数。

（5）void remove_spaces(); //删除字符串中的空格。

（6）int to_int(); //把由数字构成的字符串转成 int 类型的值。

（7）void to_lower_case(); //把字符串中的大写字母转成小写字母。

答：

```
bool String::is_substring(const char * p)
{ int len_sub=strlen(p);
  int count=strlen(str)-len_sub;
  for (int i=0;i<=count;i++)
    if(strncmp(p,str+i,len_sub)==0)
      return true;
  return false;
}

bool String::is_substring(const String & s)
{ return is_substring(s.str);
}

String String::substring(int start, int length)
{ char *temp=new char[length+1];
  strncpy(temp,str+start,length);
  temp[length] = '\0';
  String sstr(temp);
  delete []temp;
  return sstr;
}

int String::find_replace_str(const char *find_str, const char *replace_str)
{ int count=0,find_len=strlen(find_str),replace_len=strlen(replace_str);
  char *p=str;
  while (*p != '\0')
  { if (strncmp(p,find_str,find_len) == 0)
    { count++;
```

```
        p += find_len;
      }
      else
        p++;
    }
    if (count == 0) return 0;
    char *str1=new char[strlen(str)-count*find_len+count*replace_len+1];
    char *q=str1;
    p = str;
    while (*p != '\0')
    { if (strncmp(p,find_str,find_len) == 0)
      { strcpy(q,replace_str);
        q += replace_len;
        p += find_len;
      }
      else
      { *q = *p;
        q++;
        p++;
      }
    }

    delete []str;
    str = str1;
    return count;
}

void String::remove_spaces()
{ find_replace_str(" ","");
}

int String::to_int()
{ int str_int=0;
  for (int i=0; str[i]!='\0'; i++)
    str_int = str_int*10+(str[i]-'0');
  return str_int;
}

void String::to_lower_case()
{ for (int i=0; str[i]!='\0'; i++)
  { if (str[i] >= 'A' && str[i] <= 'Z')
      str[i] = str[i]-'A'+'a';
  }
}
```

17、定义一个元素类型为 int、元素个数不受限制的集合类 Set，该类具有下面的接口：

```
class Set
{  ...
  public:
    Set();
    Set(const Set& s);
```

```
    ~Set();
    bool is_empty() const; //判断是否为空集。
    int size() const; //获取元素个数。
    bool is_element(int e) const; //判断e是否属于集合。
    bool is_subset(const Set& s) const; //判断s是否包含于集合。
    bool is_equal(const Set& s) const; //判断集合是否相等。
    void display() const; //显示集合中的所有元素。
    Set& insert(int e); //将e加入到集合中。
    Set& remove(int e); //把e从集合中删除。
    Set union2(const Set& s) const; //计算集合的并集。
    Set intersection(const Set& s) const; //计算集合的交集。
    Set difference(const Set& s) const; //计算集合的差。
};
```

答：

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

struct Node
{ int value;
  Node * next;
};

class Set
{
public:
  Set();
  Set(const Set& s);
  ~Set();
  bool is_empty() const; //判断是否为空集。
  int size() const; //获取元素个数。
  bool is_element(int e) const; //判断e是否属于集合。
  bool is_subset(const Set& s) const; //判断s是否包含于集合。
  bool is_equal(const Set& s) const; //判断集合是否相等。
  void display() const; //显示集合中的所有元素。
  Set& insert(int e); //将e加入到集合中。
  Set& remove(int e); //把e从集合中删除。
  Set union(const Set& s) const; //计算集合的并集。
  Set intersection(const Set& s) const; //计算集合的交集。
  Set difference(const Set& s) const; //计算集合的差。
private:
  int count;
  Node *head;
};

Set::Set()
{ count=0;
  head=NULL;
}
```

```
Set::Set(const Set &s)
{ head = NULL;
  count = 0;
  Node *p=s.head;
  for (int i=0; i<s.count; i++)
  { insert(p->value);
    p = p->next;
  }
}

Set::~Set()
{ Node *p;
  while (head!=NULL)
  { p = head;
    head = head->next;
    delete p;
  }
  count=0;
}

bool Set::is_empty() const
{ return count==0?true:false;
}

int Set::size() const
{ return count;
}

bool Set::is_element(int e) const
{ for (Node *p=head;p!=NULL;p=p->next)
    if(p->value == e) return true;
  return false;
}

bool Set::is_subset(const Set& s) const
{ for (Node *p=s.head;p!=NULL;p=p->next)
    if(!is_element(p->value)) return false;
  return true;
}

bool Set::is_equal(const Set& s) const
{ if(count!=s.count)
    return false;
  else if(s.is_subset(*this) && is_subset(s))
    return true;
  else
    return false;
}

void Set::display() const
{ for (Node *p=head;p!=NULL;p=p->next)
    cout << p->value << '\t';
  cout << endl;
```

```
}

Set& Set::insert(int e)
{ if(!is_element(e))
  { Node *p=new Node;
    p->value = e;
    p->next = head;
    head = p;
    count++;
  }
  return *this;
}

Set& Set::remove(int e)
{ if(is_element(e))
  { if (head->value==e)
    { Node *p=head;
      head = head->next;
      delete p;
    }
    else
    { for (Node *p=head; p->next!=NULL; p=p->next)
      { if (p->next->value==e)
        { Node *temp=p->next;
          p->next = temp->next;
          delete temp;
          break;
        }
      }
    }
    count--;
  }
  return *this;
}

Set Set::union(const Set& s) const
{ Set set(s);
  Node *p=head;
  while (p!=NULL)
  { if (!set.is_element(p->value))
      set.insert(p->value);
    p=p->next;
  }
  return set;
}

Set Set::intersection(const Set& s) const
{ Set set;
  Node *p=head;
  while (p!=NULL)
  { if (s.is_element(p->value))
      set.insert(p->value);
    p = p->next;
```

```
  }
  return set;
}

Set Set::difference(const Set& s) const
{ Set set;
  Node *p=head;
  while (p!=NULL)
  { if (!s.is_element(p->value))
      set.insert(p->value);
    p = p->next;
  }
  return set;
}
```

18、定义一个由 int 型元素所构成的线性表类 LinearList，它有下面的成员函数：

```
bool insert(int x, int pos); //在位置pos之后插入一个元素x。
                             //pos为 0 时，在第一个元素之前插入。
                             //操作成功时返回true，否则返回false。
bool remove(int &x, int pos); //删除位置pos处的元素。
                              //操作成功时返回true，否则返回false。
int element(int pos) const; //返回位置pos处的元素。
int search(int x) const;
    //查找值为x的元素，返回元素的位置（第一个元素的位置为 1）。未找到时返回 0。
int length() const; //返回元素个数。
```

答：

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

class LinearList
{
public:
  LinearList(){count=0;head=NULL;}
  ~LinearList()
  { while (head != NULL)
    { Node *p=head;
      head = head->next;
      delete p;
    }
    count = 0;
  }
  bool insert(int x, int pos); //在位置pos之后插入一个元素x。
                //pos为 0 时，在第一个元素之前插入。
                //操作成功时返回true，否则返回false。
  bool remove(int &x, int pos); //删除位置pos处的元素。
                 //操作成功时返回true，否则返回false。
  int element(int pos) const; //返回位置pos处的元素。
  int search(int x) const; //查找值为x的元素，返回元素的位置（第一个元素的位置为 1）。
                //未找到时返回 0。
```

```
  int length() const; //返回元素个数。
private:
  struct Node
  { int value;
    Node * next;
  };
  int count;
  Node *head;
};

bool LinearList::insert(int x, int pos)
{ if (pos>count || pos<0)
    return false;
  Node *q=new Node;
  q->value=x;
  q->next=NULL;
  if (pos==0)
  { q->next=head;
    head=q;
  }
  else
  { Node *p=head;
    for (int i=1;i<pos;i++)
      p=p->next;
    q->next=p->next;
    p->next=q;
  }
  count++;
  return true;
}

bool LinearList::remove(int &x, int pos)
{ if (pos>count || pos<=0)
    return false;
  Node *p=head;
  if (pos==1)
  { head=head->next;
    x=p->value;
    delete p;
  }
  else
  { for (int i=2;i<pos;i++)
      p=p->next;
    Node *temp=p->next;
    p->next=temp->next;
    x=temp->value;
    delete temp;
  }
  count--;
  return true;
}

int LinearList::element(int pos) const
```

```
{ if (pos>0 && pos<=count)
  { Node *p=head;
    for (int i=1;i<pos;i++)
      p=p->next;
    return p->value;
  }
  return -1;
}

int LinearList::search(int x) const
{ int pos=0;
  Node *p=head;
  while (p != NULL)
  { pos++;
    if(p->value == x) return pos;
    p=p->next;
  }
  return 0;
}

int LinearList::length() const
{ return count;
}
```

19、定义一个类 A，使得在程序中只能创建一个该类的对象，当试图创建该类的第二个对象时，返回第一个对象的指针。

答：

```
//Design of class Singleton:
class Singleton
{ public:
    static Singleton* Instance();
  protected:
    Singleton();
  private:
    static Singleton* _instance;
}

Singleton* Singleton::_instance = 0;

Singleton* Singleton::Instance()
{ if (_instance == 0)
  { _instance = new Singleton;
  }
  return _instance;
};


//How to Use Singleton:
Singleton* singleton = Singleton::Instance();
```

20、对于下面的类 A，如何用 C++的非面向对象语言成分来实现它？

```
class A
{ public:
    A() { x = y = 0; }
    A(int i, int j) { x = i; y = j; }
    void f() { h(); ..... }
    void g(int i) { x = i; ..... }
  private:
    int x,y;
    void h() { ..... }
};
```

答：(下面的解决方案没考虑 inline 函数)

```
//A.h
struct A
{ void *p_data;
};
void A_A(struct A *p_a);
void A_A(struct A *p_a,int i, int j);
void A_f(struct A *p_a);
void A_g(struct A *p_a, int i);

//A.cpp
#include <malloc.h>
#include "A.h"
struct A_Data
{ int x,y;
};
void A_A(struct A *p_a)
{ struct A_Data *p;
  p = (struct A_Data *)malloc(sizeof(struct A_Data));
  p->x = p->y = 0;
  p_a->p_data = p;
}
void A_A(struct A *p_a,int i, int j)
{ struct A_Data *p;
  p = (struct A_Data *)malloc(sizeof(struct A_Data));
  p->x = i;
  p->y = j;
  p_a->p_data = p;
}
extern void A_h(struct A *p_a);
void A_f(A *p_a)
{ A_h(p_a);
  ......
}
void A_g(struct A *p_a, int i)
{ struct A_Data *p=(struct A_Data *)p_a->p_data;
  p->x = i;
  ......
}
static void A_h(struct A *p_a)
```

```
{ ......
}
```

第 7 章 操作符重载

1、 为什么要对操作符进行重载？是否所有的操作符都可以重载？

答：通过对 C++操作符进行重载，我们可以实现用 C++的操作符按照通常的习惯来对某些类（特别是一些数学类）的对象进行操作，从而使得程序更容易理解。除此之外，操作符重载机制也提高了 C++语言的灵活性和可扩充性，它使得 C++操作符除了能对基本数据类型和构造数据类型进行操作外，也能用它们来对类的对象进行操作。

不是所有的操作符都可以重载，因为“ . ”， “ .* ”， “ :: ”， “ ?: ”，sizeof 这五个操作符不能重载。

2、 操作符重载的形式有哪两种形式？这两种形式有什么区别？

答：一种就是作为成员函数重载操作符；另一种就是作为全局（友元）函数重载操作符。

当操作符作为类的非静态成员函数来重载时，由于成员函数已经有一个隐藏的参数 this，因此对于双目操作符重载函数只需要提供一个参数，对于单目操作符重载函数则不需提供参数。

当操作符作为全局函数来重载时，操作符重载函数的参数类型至少有一个为类、结构、枚举或它们的引用类型。而且如果要访问参数类的私有成员，还需要把该函数说明成相应类的友元。对于双目操作符重载函数需要两个参数，对于单目操作符重载函数则需要给出一个参数。操作符=、 ( )、[ ]以及->不能作为全局函数来重载。

另外，作为类成员函数来重载时，操作符的第一个操作数必须是类的对象，全局函数重载则否。

3、 定义一个时间类 Time，通过操作符重载实现：时间的比较（==、!=、>、>=、<、<=）、时间增加/减少若干秒（+=、-=）、时间增加/减少一秒（++、--）以及两个时间相差的秒数（-）。

解：

```
class Time
{
private :
  int hour, minute, second;

public :
  Time()
  { hour=minute=second=0;
  }

  Time(int h)
  { hour=h;
    minute=second=0;
  }
```

```
Time(int h, int m)
{ hour=h;
  minute=m;
  second=0;
}

Time(int h, int m, int s)
{ hour=h;
  minute=m;
  second=s;
}

Time(const Time &t)
{ hour=t.hour;
  minute=t.minute;
  second=t.second;
}

bool operator ==(Time &t)
{ if (hour==t.hour && minute==t.minute && second==t.second)
    return true;
  return false;
}

bool operator !=(Time &t)
{ return !(*this == t);
}

bool operator > (Time &t)
{ if (hour>t.hour)
    return true;
  else if (hour==t.hour && minute>t.minute)
    return true;
  else if (hour==t.hour && minute==t.minute && second>t.second)
    return true;
  else
    return false;
}

bool operator >=(Time &t)
{ return *this > t || *this == t;
}

bool operator <(Time &t)
{ return !(*this >= t);
}

bool operator <=(Time &t)
{ return !(*this > t);
}

Time &operator +=(int s)
```

```
{ second+=s;
  while (second>=60)
  { second-=60;
    minute++;
  }
  while (minute>=60)
  { minute-=60;
    hour++;
  }
  while (hour>=24)
    hour-=24;
  return *this;
}

Time &operator -=(int s)
{ second-=s;
  while (second<0)
  { second+=60;
    minute--;
  }
  while (minute<0)
  { minute+=60;
    hour--;
  }
  while (hour<0)
    hour+=24;
  return *this;
}

Time &operator ++()    //对Time t，操作为：++t
{ *this += 1;
  return *this;
}

Time operator ++(int)    //对Time t，操作为：t++
{ Time t=*this;
  *this += 1;
  return t;
}

Time &operator --()
{ *this -= 1;
  return *this;
}

Time operator --(int)
{ Time t=*this;
  *this -= 1;
  return t;
}

int operator -(Time &t)
```

```
  {//把时间直接换算成秒数计算
    int sec1=hour*3600+minute*60+second;
    int sec2=t.hour*3600+t.minute*60+t.second;
    return sec2-sec1;
  }
};
```

4、利用操作符重载给出一个完整的复数类的定义。

解：

```
class Complex
{
private :
  double real, imag;
public :
  Complex()
  { real = imag = 0;
  }
  Complex(double r)
  { real = r;
    imag = 0;
  }
  Complex(double r, double i)
  { real = r;
    imag = i;
  }
  Complex(const Complex &c)
  { real=c.real;
    imag=c.imag;
  }
  double modulus() const
  { return real*real+imag*imag;
  }
  Complex operator -() const
  { Complex temp;
    temp .real = -real;
    temp.imag = -imag;
    return temp;
  }
  friend bool operator ==(const Complex &c1,const Complex &c2);
  friend bool operator !=(const Complex &c1,const Complex &c2);
  friend bool operator > (const Complex &c1,const Complex &c2);
  friend bool operator >=(const Complex &c1,const Complex &c2);
  friend bool operator < (const Complex &c1,const Complex &c2);
  friend bool operator <=(const Complex &c1,const Complex &c2);
  friend Complex operator+(const Complex &c1,const Complex &c2);
  friend Complex operator-(const Complex &c1,const Complex &c2);
  friend Complex operator*(const Complex &c1,const Complex &c2);
  friend Complex operator/(const Complex &c1,const Complex &c2);
};

bool operator ==(const Complex &c1,const Complex &c2)
{ return (c1.real == c2.real) && (c1.imag == c2.imag);
```

```
}

bool operator !=(const Complex &c1,const Complex &c2)
{ return !(c1 == c2);
}

bool operator >(const Complex &c1,const Complex &c2)
{ return c1.modulus() > c2.modulus();
}

bool operator >=(const Complex &c1,const Complex &c2)
{ return c1.modulus() >= c2.modulus();
}

bool operator <(const Complex &c1,const Complex &c2)
{ return c1.modulus() < c2.modulus();
}

bool operator <=(const Complex &c1,const Complex &c2)
{ return c1.modulus() <= c2.modulus();
}

Complex operator+(const Complex &c1,const Complex &c2)
{ Complex temp;
  temp.real=c1.real+c2.real;
  temp.imag=c1.imag+c2.imag;
  return temp;
}

Complex operator-(const Complex &c1,const Complex &c2)
{ Complex temp;
  temp.real=c1.real-c2.real;
  temp.imag=c1.imag-c2.imag;
  return temp;
}

Complex operator*(const Complex &c1,const Complex &c2)
{ Complex temp;
  temp.real=c1.real*c2.real-c1.imag*c2.imag;
  temp.imag=c1.real*c2.imag+c1.imag*c2.real;
  return temp;
}

Complex operator/(const Complex &c1,const Complex &c2)
{ double d=c2.modulus();
  if (d != 0)
  { Complex temp;
    temp.real=(c1.real*c2.real+c1.imag*c2.imag)/d;
    temp.imag=(c1.imag*c2.real-c1.real*c2.imag)/d;
    return temp;
  }
  else
  { cout << "Error in operation / of Complex" << endl;
```

```
    exit(-1);
  }
}
```

5、 定义一个多项式类 Polynomial，其实例为多项式：$a_0+a_1x+a_2x^2+...+a_nx^n$，该类具有如下的接口：

```
class Polynomial
{   ......
public:
  Polynomial();
  Polynomial(double coefs[], int exps[], int size);
                                    //系数数组、指数数组和项数
  Polynomial(const Polynomial&);
  ~ Polynomial();
  Polynomial& operator=(const Polynomial&);
  int degree() const; //最高幂指数
  double evaluate(double x) const; //计算多项式的值
  bool operator==(const Polynomial&) const;
  bool operator!=(const Polynomial&) const;
  Polynomial operator+(const Polynomial&) const;
  Polynomial operator-(const Polynomial&) const;
  Polynomial operator*(const Polynomial&) const;
  Polynomial& operator+=(const Polynomial&);
  Polynomial& operator-=(const Polynomial&);
  Polynomial& operator*=(const Polynomial&);
};
```

解：

```
class Polynomial
{ double *pcoefs;
  int *pexps;
  int num_of_items;
  int add(const Polynomial &p, double *coefs, int *exps) const;
  int subtract(const Polynomial &p, double *coefs, int *exps) const;
public :
  Polynomial();
  Polynomial(double coefs[], int exps[], int size);
  Polynomial(const Polynomial &p);
  ~Polynomial();
  Polynomial &operator=(const Polynomial &p);
  int degree() const;
  double evaluate(double x) const;
  bool operator==(const Polynomial &p) const;
  bool operator!=(const Polynomial &p) const;
  Polynomial operator+(const Polynomial &p) const;
  Polynomial operator-(const Polynomial &p) const;
  Polynomial operator*(const Polynomial &p) const;
  Polynomial &operator+=(const Polynomial &p);
  Polynomial &operator-=(const Polynomial &p);
  Polynomial &operator*=(const Polynomial &p);
};

int Polynomial::add(const Polynomial &p, double *coefs, int *exps) const
```

```
{ int count=0,i=0,j=0;
  while (i<num_of_items && j<p.num_of_items)
  { if (pexps[i] == p.pexps[j])
    { if (pcoefs[i] != -p.pcoefs[j])
      { coefs[count] = pcoefs[i] + p.pcoefs[j];
        exps[count] = pexps[i];
        count++;
      }
      i++; j++;
    }
    else if (pexps[i] < p.pexps[j])
    { coefs[count] = pcoefs[i];
      exps[count] = pexps[i];
      count++; i++;
    }
    else
    { coefs[count] = p.pcoefs[j];
      exps[count] = p.pexps[j];
      count++; j++;
    }
  }
  if (i<num_of_items)
    while (i<num_of_items)
    { coefs[count] = pcoefs[i];
      exps[count] = pexps[i];
      count++; i++;
    }
  else
    while (j<p.num_of_items)
    { coefs[count] = p.pcoefs[j];
      exps[count] = p.pexps[j];
      count++; j++;
    }

  return count;
}

int Polynomial::subtract(const Polynomial &p, double *coefs, int *exps) const
{ int count=0,i=0,j=0;
  while (i<num_of_items && j<p.num_of_items)
  { if (pexps[i] == p.pexps[j])
    { if (pcoefs[i] != p.pcoefs[j])
      { coefs[count] = pcoefs[i] - p.pcoefs[j];
        exps[count] = pexps[i];
        count++;
      }
      i++; j++;
    }
    else if (pexps[i] < p.pexps[j])
    { coefs[count] = pcoefs[i];
      exps[count] = pexps[i];
      count++; i++;
    }
```

```
    else
    { coefs[count] = -p.pcoefs[j];
      exps[count] = p.pexps[j];
      count++; j++;
    }
  }
  if (i<num_of_items)
    while (i<num_of_items)
    { coefs[count] = pcoefs[i];
      exps[count] = pexps[i];
      count++; i++;
    }
  else
    while (j<p.num_of_items)
    { coefs[count] = -p.pcoefs[j];
      exps[count] = p.pexps[j];
      count++; j++;
    }

  return count;
}

Polynomial::Polynomial()
{ pcoefs=NULL;
  pexps=NULL;
  num_of_items=0;
}

Polynomial::Polynomial(double coefs[], int exps[], int size)
{ num_of_items=size;
  pcoefs=new double[num_of_items];
  pexps=new int[num_of_items];
  int i;
  for (i=0; i<num_of_items; i++)
  { pcoefs[i]=coefs[i];
    pexps[i]=exps[i];
  }
  //按指数排序（冒泡排序）
  for (i=num_of_items; i>1; i--)
  { bool exchange=false;
    for (int j=1; j<i; j++)
    { if (pexps[j] < pexps[j-1])
      { //交换pexps[j]和pexps[j-1]
        int temp1=pexps[j];
        pexps[j] = pexps[j-1];
        pexps[j-1] = temp1;

        //交换pcoefs[j]和pcoefs[j-1]
        double temp2 = pcoefs[j];
        pcoefs[j] = pcoefs[j-1];
        pcoefs[j-1] = temp2;
        exchange = true;
```

```
      }
    }
    if (!exchange) break;
  }
}

Polynomial::Polynomial(const Polynomial &p)
{ num_of_items=p.num_of_items;
  pcoefs=new double[num_of_items];
  pexps=new int[num_of_items];
  for (int i=0; i<num_of_items; i++)
  { pcoefs[i]=p.pcoefs[i];
    pexps[i]=p.pexps[i];
  }
}

Polynomial::~Polynomial()
{ delete []pcoefs;
  delete []pexps;
  pcoefs=NULL;
  pexps=NULL;
  num_of_items=0;
}

Polynomial& Polynomial::operator=(const Polynomial &p)
{ delete []pcoefs;
  delete []pexps;
  num_of_items=p.num_of_items;
  pcoefs=new double[num_of_items];
  pexps=new int[num_of_items];
  for (int i=0; i<num_of_items; i++)
  { pcoefs[i]=p.pcoefs[i];
    pexps[i]=p.pexps[i];
  }
  return *this;
}

int Polynomial::degree() const
{ if (num_of_items == 0)
    return 0;
  else
    return pexps[num_of_items-1];
}

double Polynomial::evaluate(double x) const
{ double sum=0;
  for (int i=0; i<num_of_items; i++)
  { double temp=pcoefs[i];
    for (int j=0; j<pexps[i]; j++)
      temp *= x;
    sum += temp;
  }
  return sum;
```

```
}

bool Polynomial::operator==(const Polynomial &p) const
{ if (num_of_items != p.num_of_items) return false;
  for (int i=0; i<num_of_items; i++)
    if (pcoefs[i]!=p.pcoefs[i] || pexps[i]!=p.pexps[i])
      return false;
  return true;
}

bool Polynomial::operator!=(const Polynomial &p) const
{ return !(*this == p);
}

Polynomial Polynomial::operator+(const Polynomial &p) const
{ double *coefs=new double[num_of_items+p.num_of_items];
  int *exps=new int[num_of_items+p.num_of_items];
  int count=add(p,coefs,exps);

  Polynomial temp(coefs, exps, count);
  delete []coefs;
  delete []exps;
  return temp;
}

Polynomial Polynomial::operator-(const Polynomial &p) const
{ double *coefs=new double[num_of_items+p.num_of_items];
  int *exps=new int[num_of_items+p.num_of_items];
  int count=subtract(p,coefs,exps);

  Polynomial temp(coefs, exps, count);
  delete []coefs;
  delete []exps;
  return temp;
}

Polynomial Polynomial::operator*(const Polynomial &p) const
{ Polynomial sum,temp=*this;

  for (int i=0; i<p.num_of_items; i++)
  { for (int j=0; j<num_of_items; j++)
    { temp.pcoefs[j] = pcoefs[j] * p.pcoefs[i];
      temp.pexps[j] = pexps[j] + p.pexps[i];
    }
    sum += temp;
  }
  return sum;
}

Polynomial &Polynomial::operator+=(const Polynomial &p)
{ double *coefs=new double[num_of_items+p.num_of_items];
  int *exps=new int[num_of_items+p.num_of_items];
  int count=add(p,coefs,exps);
```

```
  delete []pcoefs;
  delete []pexps;

  pcoefs = coefs; //有多余的空间，实际的项数由count决定。
  pexps = exps; //同上。
  num_of_items = count;

  return *this;
}

Polynomial &Polynomial::operator-=(const Polynomial &p)
{ double *coefs=new double[num_of_items+p.num_of_items];
  int *exps=new int[num_of_items+p.num_of_items];
  int count=subtract(p,coefs,exps);

  delete []pcoefs;
  delete []pexps;

  pcoefs = coefs; //有多余的空间，实际的项数由count决定。
  pexps = exps; //同上。
  num_of_items = count;

  return *this;
}

Polynomial &Polynomial::operator*=(const Polynomial &p)
{ Polynomial sum,temp=*this;

  for (int i=0; i<p.num_of_items; i++)
  { for (int j=0; j<num_of_items; j++)
    { temp.pcoefs[j] = pcoefs[j] * p.pcoefs[i];
      temp.pexps[j] = pexps[j] + p.pexps[i];
    }
    sum += temp;
  }
  *this = sum;
  return *this;
}
```

6、 用操作符重载实现集合类的一些操作：<=（包含于）、==（相等）、!=（不等）、 |（并集）、 &（交集）、-（差集）、+=（增加元素）、-=（删除元素）等。

解：

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

struct Node
{ int value;
  Node * next;
};
```

```
class Set
{
public:
  Set();
  Set(const Set& s);
  ~Set();
  bool is_empty() const; //判断是否为空集。
  int size() const; //获取元素个数。
  bool is_element(int e) const; //判断e是否属于集合。
  void display() const; //显示集合中的所有元素。
  bool operator <=(const Set& s) const; //判断s是否包含于集合。
  bool operator ==(const Set& s) const; //判断集合是否相等。
  bool operator !=(const Set& s) const; //判断集合是否不相等。
  Set& operator +=(int e); //将e加入到集合中。
  Set& operator -=(int e); //把e从集合中删除。
  Set operator |(const Set& s) const; //计算集合的并集。
  Set operator &(const Set& s) const; //计算集合的交集。
  Set operator -(const Set& s) const; //计算集合的差。
 Set &operator = (const Set& s); //集合赋值
private:
  int count;
  Node *head;
};

Set::Set()
{ count=0;
  head=NULL;
}

Set::Set(const Set &s)
{ head = NULL;
  count = 0;
  Node *p=s.head;
  for (int i=0; i<s.count; i++)
  { *this += (p->value);
    p = p->next;
  }
}

Set::~Set()
{ Node *p;
  while (head!=NULL)
  { p = head;
    head = head->next;
    delete p;
  }
  count=0;
}

bool Set::is_empty() const
{ return count==0?true:false;
```

```
}

int Set::size() const
{ return count;
}

bool Set::is_element(int e) const
{ for (Node *p=head;p!=NULL;p=p->next)
    if (p->value == e) return true;
  return false;
}

bool Set::operator <=(const Set& s) const
{ for (Node *p=s.head;p!=NULL;p=p->next)
    if(!is_element(p->value)) return false;
  return true;
}

bool Set::operator ==(const Set& s) const
{ if (count!=s.count)
    return false;
  else if (*this <= s && s <= *this)
    return true;
  else
    return false;
}

bool Set::operator !=(const Set& s) const
{ return !(*this == s);
}

void Set::display() const
{ for (Node *p=head;p!=NULL;p=p->next)
    cout << p->value << '\t';
  cout << endl;
}

Set& Set::operator +=(int e)
{ if(!is_element(e))
  { Node *p=new Node;
    p->value = e;
    p->next = head;
    head = p;
    count++;
  }
  return *this;
}

Set& Set::operator -=(int e)
{ if(is_element(e))
  { if (head->value==e)
    { Node *p=head;
      head = head->next;
```

```
      delete p;
    }
    else
    { for (Node *p=head; p->next!=NULL; p=p->next)
      { if (p->next->value==e)
        { Node *temp=p->next;
          p->next = temp->next;
          delete temp;
          break;
        }
      }
    }
    count--;
  }
  return *this;
}

Set Set::operator |(const Set& s) const
{ Set set(s);
  Node *p=head;
  while (p!=NULL)
  { if (!set.is_element(p->value))
      set += p->value;
    p=p->next;
  }
  return set;
}

Set Set::operator &(const Set& s) const
{ Set set;
  Node *p=head;
  while (p!=NULL)
  { if (s.is_element(p->value))
      set += p->value;
    p = p->next;
  }
  return set;
}

Set Set::operator -(const Set& s) const
{ Set set;
  Node *p=head;
  while (p!=NULL)
  { if (!s.is_element(p->value))
      set += p->value;
    p = p->next;
  }
  return set;
}

Set &Set::operator = (const Set& s)
{ Node *p;
  while (head!=NULL)
```

```
  { p = head;
    head = head->next;
    delete p;
  }
  count=0;
  p=s.head;
  for (int i=0; i<s.count; i++)
  { *this += (p->value);
    p = p->next;
  }
  return *this;
}
```

7、 定义一个不受计算机字长限制的整数类 INT，要求 INT 与 INT 以及 INT 与 C++基本数据类型 int 之间能进行＋、－、×、÷和＝运算，并且能通过 cout 输出 INT 类型的值。

解：

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

class INT
{ char *p_buf; //从低位到高位存储整型数，
  int buf_len, //p_buf所占空间大小（为 10 的倍数）
      sign; //整型数的符号（1 为正，-1 为负）
  void Add(const INT &i); //把i加到*this中
  void Minus(const INT &i); //从*this中减去i
  INT(char *p, int len, int s=1); //新对象的p_buf在已有的空间p上（不再另外分配空间）。

public :
  INT();
  INT(int i);
  INT(char *num);
  INT(const INT &i);
  ~INT();
  INT &operator=(const INT &i);
  INT& INT::operator+=(const INT &i);
  INT& INT::operator-=(const INT &i);
  friend INT operator+(const INT &i1, const INT &i2);
  friend INT operator-(const INT &i1, const INT &i2);
  friend INT operator*(const INT &i1, const INT &i2);
  friend INT operator/(const INT &i1, const INT &i2);
  friend ostream &operator<<(ostream &out, const INT &i);
};

INT::INT()
{ sign=1;
  buf_len = 10;
  p_buf=new char[buf_len+1];
  p_buf[0]='0';
  p_buf[1]='\0';
}
```

```
INT::INT(int i)
{ if (i>=0)
    sign=1;
  else
  { sign=-1;
    i = -i;
  }
  int j=i, k=0;
  do
  { k++;
    j/=10;
  } while (j!=0);
  buf_len = (k/10+1)*10;
  p_buf = new char[buf_len+1];
  j=0;
  do
  { k=i%10;
    p_buf[j++]=k+'0';
    i/=10;
  }while (i!=0);
  p_buf[j]='\0';
}

INT::INT(char *num)
{ if (*num=='-')
  { sign = -1;
    num++;
  }
  else if (*num=='+')
  { sign = 1;
    num++;
  }
  else
    sign = 1;
  buf_len = (strlen(num)/10+1)*10;
  p_buf = new char[buf_len+1];
  for (int i=0,j=strlen(num)-1; j>=0; i++,j--)
    p_buf[i] = num[j];
  p_buf[i] = '\0';
}

INT::INT(const INT &i)
{ sign = i.sign;
  buf_len = i.buf_len;
  p_buf = new char[buf_len+1];
  strcpy(p_buf, i.p_buf);
}

INT::INT(char *p, int len, int s)
{ p_buf = p;
  buf_len = len;
  sign = s;
```

```
}

INT::~INT()
{ delete []p_buf;
  p_buf = NULL;
  buf_len = 0;
}

INT& INT::operator =(const INT &i)
{ sign = i.sign;
  if (buf_len < i.buf_len)
  { delete []p_buf;
    buf_len = i.buf_len;
    p_buf = new char[buf_len+1];
  }
  strcpy(p_buf, i.p_buf);
  return *this;
}

void INT::Add(const INT &i) //绝对值加
{ int len1=strlen(p_buf);
  int len2=strlen(i.p_buf);
  int buf_len1=(((len1>len2?len1:len2)+1)/10+1)*10;
  if (buf_len1 > buf_len)
  { char *p_buf1=new char[buf_len1+1];
    strcpy(p_buf1,p_buf);
    delete []p_buf;
    p_buf = p_buf1;
    buf_len = buf_len1;
  }

  char *p1,*p2; //p1 指向长的数，p2 指向短的数
  if (len1 >= len2)
  { p1 = p_buf;
    p2 = i.p_buf;
  }
  else
  { p1 = i.p_buf;
    p2 = p_buf;
  }

  int carry=0, sum;
  char *p=p_buf;
  //处理公共长度部分
  while (*p2 != '\0')
  { sum = (*p1-'0')+(*p2-'0')+carry;
    if (sum >= 10)
    { carry = 1;
      *p = (sum-10)+'0';
    }
    else
    { carry = 0;
```

```
        *p = sum+'0';
      }
     p1++; p2++; p++;
    }
    //处理较大整数的剩余部分
    while (*p1 != '\0')
    { if (carry == 0)
       *p = *p1;
     else
     { sum = (*p1-'0')+carry;
       if (sum >= 10)
       { carry = 1;
         *p = (sum-10)+'0';
       }
       else
       { carry = 0;
         *p = sum+'0';
       }
     }
      p1++; p++;
    }
    if (carry != 0)      //最后检查是否还有进位，若有就放入和的最高位
    { *p = '1';
     p++;
    }
    *p = '\0';
}

void INT::Minus(const INT &i)  //绝对值减，*this-i
{ int len1=strlen(p_buf);
  int len2=strlen(i.p_buf);
  int buf_len1=((len1>len2?len1:len2)/10+1)*10;
  if (buf_len1 > buf_len)
  { char *p_buf1=new char[buf_len1+1];
    strcpy(p_buf1,p_buf);
    delete []p_buf;
    p_buf = p_buf1;
    buf_len = buf_len1;
  }

  char *p1,*p2;

  if (len1 > len2)
    p1 = p_buf;
  else if (len1 < len2)
     p1 = i.p_buf;
  else
  { int j;
    for (j=len1-1; j>=0; j--)
     { if (p_buf[j] > i.p_buf[j])
       { p1 = p_buf;
         break;
```

```
    }
    else if (p_buf[j] < i.p_buf[j])
    { p1 = i.p_buf;
      break;
    }
  }
   if (j < 0)  //如果两个数完全相同
   { sign = 1;
     strcpy(p_buf,"0");
    return;
   }
}

if (p1 == p_buf)
{ p2 = i.p_buf;
}
else
{ p2 = p_buf;
  sign = i.sign;
}

int carry=0, diff;
char *p=p_buf;
//公有长度部分相减
while (*p2 != '\0')
{ diff = *p1-*p2-carry;
  if (diff < 0)
  { *p = (diff+10)+'0';
    carry = 1;
  }
  else
  { *p = diff+'0';
    carry = 0;
  }
  p1++; p2++; p++;
}
//将被减数中未参与相减的高位也计算到差中
while (*p1 != '\0')
{ if (carry == 0)
   *p = *p1;
  else
  { diff = (*p1-'0')-carry;
    if (diff < 0)
    { *p = (diff+10)+'0';
      carry = 1;
    }
    else
    { *p = diff+'0';
      carry = 0;
    }
  }
   p1++; p++;
```

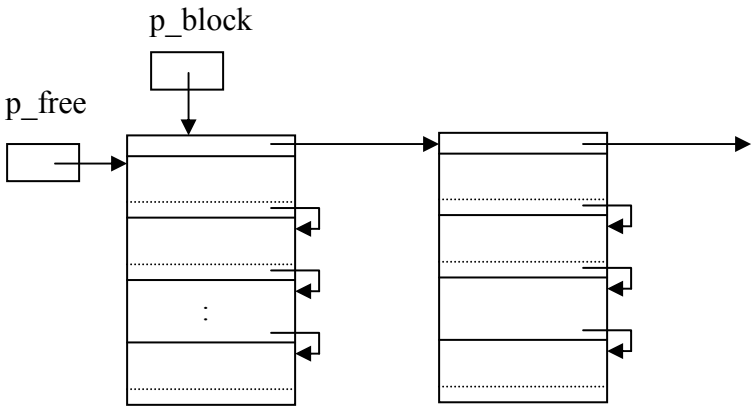
```
  }
  p--;
  while (p != p_buf && *p == '0') p--; //去掉高位的0
  *(p+1) = '\0';
}

INT& INT::operator+=(const INT &i)
{ if (sign == i.sign)  //如果两个数符号相同，绝对值相加
    Add(i);
  else        //如果两个数符号相反，绝对值相减
    Minus(i);
  return *this;
}

INT operator+(const INT &i1, const INT &i2)
{ INT temp(i1);
  temp += i2;
  return temp;
}

INT& INT::operator-=(const INT &i)
{ INT *p=(INT *)&i;
  p->sign = -p->sign; //把i变成-i
  *this += *p; // *this += -i
  p->sign = -p->sign; //把i还原
  return *this;
}

INT operator-(const INT &i1, const INT &i2)
{ INT temp(i1);
  temp -= i2;
  return temp;
}

INT operator*(const INT &i1, const INT &i2)
{ int len1=strlen(i1.p_buf);
  int len2=strlen(i2.p_buf);
  int buf_len=((len1+len2)/10+1)*10;
  char *p_buf=new char[buf_len+1];
  INT product,temp(p_buf,buf_len,1);
  //用一个乘数(i1)的每一位与另一个乘数(i2)相乘，结果(temp)向左移相应的位数后加到乘积（product）中去
  for (int j=0; j<len1; j++)
  { int carry=0, mul, n;
    for (n=0; n<j; n++)  //把temp的位置左移若干位
      p_buf[n] = '0';
    for (int k=0; k<len2; k++)   //i1[i]与i2 的每一位相乘，结果放在temp中
    { mul = (i1.p_buf[j]-'0')*(i2.p_buf[k]-'0')+carry;
      carry = mul/10;
      p_buf[n++] = mul%10+'0';
    }
    if (carry != 0) p_buf[n++] = carry+'0';
```

```
    p_buf[n]='\0';
     product += temp;//将每一位与另一个乘数相乘的结果累加就可以得到两数之积
   }
   if (i1.sign == i2.sign)      //最后统一处理积的符号
     product.sign=1;
   else
     product.sign=-1;
   return product;
}

INT operator/(const INT &i1, const INT &i2)
{ INT div1(i1), div2(i2), quotient;
   div1.sign=1;
   div2.sign=1;

   //不断地从“被除数”中减去“除数”，直到“被除数”小于0，这时，减去的“除数”的个数即为“商”。
   div1 -= div2;  //用被除数减去除数
   while (div1.sign>0)      //直到被除数小于零
   { quotient += 1;     //每减一次除数，计数器加1
     div1 -= div2;    //用被除数减去除数
   }
   if (i1.sign == i2.sign)     //最后统一处理商的符号
     quotient.sign = 1;
   else
     quotient.sign = -1;
   return quotient;
}

ostream& operator<<(ostream &out, const INT &i)
{ if (i.sign<0) out<<'-';
   for (int j=strlen(i.p_buf)-1; j>=0; j--)
     out<<i.p_buf[j];
   return out;
}

int main()
{ INT i1("1234567890"),i2("10000"),i3,i4,i5,i6;

   i3 = i1+i2;
   i4 = i1-i2;
   i5 = i1*i2;
   i6 = i1/i2;
   cout << i1 << endl
        << i2 << endl
        << i3 << endl
        << i4 << endl
        << i5 << endl
        << i6 << endl;
   return 0;
}
```

8、 设计一种解决例 7-6 中把存储块归还到堆空间的方法。(提示：可以在每次申请存储块时多申请一

个能存储一个指针的空间，用该指针把各个存储块链接起来。）

解：申请存储块时多申请一个指针空间把存储块链起来，用静态数据成员 p_block 指向存储块链表。程序结束时，调用一下静态成员函数 free_blocks()把所有存储块归还掉。




```
#include <cstring>
//using namespace std; //在visual c++ 6.0 中不要这一行！
const int NUM=2;
class A
{   //...... //类A的已有成员说明。
  public:
    static void *operator new(size_t size);
    static void operator delete(void *p);
   static void free_blocks(); //归还申请的所有存储块。
  private:
    static A *p_free; //用于指向A类对象的自由空间链表。
    A *next; //用于实现自由空间结点的链接。
   static char *p_block; //用于指向由NUM个A类对象所组成的存储块所构成的链表。
};
A *A::p_free=NULL;
char *A::p_block=NULL;
void *A::operator new(size_t size)
{ A *p;
  if (p_free == NULL)
  { char *q=new char[sizeof(char *)+size*NUM]; //申请一个块指针和NUM个A类对象所构成的堆空
间。
    *(char **)q = p_block; //建立存储块链表。
    p_block = q;

    p_free = (A *)(q+sizeof(char *));  //跳过块指针。
    for (p=p_free; p!=p_free+NUM-1; p++)  //建立自由结点链表。
      p->next = p+1;
    p->next = NULL;
  }
  p = p_free; //为当前创建的对象分配空间。
  p_free = p_free->next;
  memset(p,0,size);
```

```
  return p;
}
void A::operator delete(void *p)
{ ((A *)p)->next = p_free;
  p_free = (A *)p;
}

void A::free_blocks() //归还申请的所有存储块。
{ while (p_block != NULL)
  { char *p=p_block;
    p_block = *(char **)p_block;
    delete []p;
  }
}
```

第 8 章 继承——派生类

1、在 C++中，protected 类成员访问控制有什么作用？

答：C++中引进 protected 成员保护控制，缓解了数据封装与继承的矛盾。在基类中声明为 protected 的成员可以被派生类使用，但不能被基类的实例用户使用，这样能够对修改基类的内部实现所造成的影响范围（只影响子类）进行控制。protected 成员保护控制的引进使得类有两种接口：与实例用户的接口和与派生类用户的接口。

2、在 C++中，三种继承方式各有什么作用？

答：类的继承方式决定了派生类的对象和派生类的派生类对基类成员的访问限制。public 继承方式使得基类的 public 成员可以被派生类的对象访问，它可以实现类之间的子类型关系；protected 继承使得基类的 public 成员不能被派生类的对象访问，但可以被派生类的派生类访问；private 继承使得基类的 public 成员既不能被派生类的对象访问，也不能被派生类的派生类访问。protected 和 private 继承主要用于实现上的继承，即纯粹为了代码复用。

3、在多继承中，什么情况下会出现二义性？怎样消除二义性？

答：在多继承中会出现两个问题：名冲突和重复继承。在多继承中，当多个基类中包含同名的成员时，它们在派生类中就会出现名冲突问题；在多继承中，如果直接基类有公共的基类，就会出现重复继承，这样，公共基类中的数据成员在多继承的派生类中就有多个拷贝。在 C++中，解决名冲突的方法是用基类名受限；解决重复继承问题的手段是采用虚基类。

4、写出下面程序的运行结果：

```
#include <iostream>
using namespace std;
class A
{   int m;
  public:
    A() { cout << "in A's default constructor\n"; }
    A(const A&) { cout << "in A's copy constructor\n"; }
    ~A() { cout << "in A's destructor\n"; }
};
class B
{   int x,y;
  public:
    B() { cout << "in B's default constructor\n"; }
    B(const B&) { cout << "in B's copy constructor\n"; }
    ~B() { cout << "in B's destructor\n"; }
};
class C: public B
{   int z;
    A a;
```

```
  public:
    C() { cout << "in C's default constructor\n"; }
    C(const C&) { cout << "in C's copy constructor\n"; }
    ~C() { cout << "in C's destructor\n"; }
};
void func1(C x)
{ cout << "In func1\n";
}
void func2(C &x)
{ cout << "In func2\n";
}
int main()
{ cout << "------Section 1------\n";
  C c;
  cout << "------Section 2------\n";
  func1(c);
  cout << "------Section 3------\n";
  func2(c);
  cout << "------Section 4------\n";
  return 0;
}
```

答：------Section 1------

in B's default constructor

in A's default constructor

in C's default constructor

------Section 2------

in B's default constructor

in A's default constructor

in C's copy constructor

In func1

in C's destructor

in A's destructor

in B's destructor

------Section 3------

In func2

------Section 4------

in C's destructor

in A's destructor

in B's destructor

5、 写出下面程序的运行结果：

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

```
{   int x,y;
  public:
    A() { cout << "in A's default constructor\n"; f(); }
    A(const A&) { cout << "in A's copy constructor\n"; f(); }
    ~A() { cout << "in A's destructor\n"; }
    virtual void f() { cout << "in A's f\n"; }
    void g() { cout << "in A's g\n"; }
    void h() { f(); g(); }
};
class B: public A
{   int z;
  public:
    B() { cout << "in B's default constructor\n"; }
    B(const B&) { cout << "in B's copy constructor\n"; }
    ~B() { cout << "in B's destructor\n"; }
    void f() { cout << "in B's f\n"; }
    void g() { cout << "in B's g\n"; }
};
void func1(A x)
{ x.f();
  x.g();
  x.h();
}
void func2(A &x)
{ x.f();
  x.g();
  x.h();
}
int main()
{ cout << "------Section 1------\n";
  A a;
  A *p=new B;
  cout << "------Section 2------\n";
  func1(a);
  cout << "------Section 3------\n";
  func1(*p);
  cout << "------Section 4------\n";
  func2(a);
  cout << "------Section 5------\n";
  func2(*p);
  cout << "------Section 6------\n";
  delete p;
  cout << "------Section 7------\n";
  return 0;
}
```

答：------Section 1------

in A's default constructor

in A's f

in A's default constructor

in A's f

in B's default constructor
------Section 2------
in A's copy constructor
in A's f
in A's f
in A's g
in A's f
in A's g
in A's destructor
------Section 3------
in A's copy constructor
in A's f
in A's f
in A's g
in A's f
in A's g
in A's destructor
------Section 4------
in A's f
in A's g
in A's f
in A's g
------Section 5------
in B's f
in A's g
in B's f
in A's g
------Section 6------
in A's destructor
------Section 7------
in A's destructor

6、 利用习题 6.9 的第 14 题中的时间类 Time，定义一个带时区的时间类 ExtTime。除了构造函数和时间调整函数外，ExtTime 的其它功能与 Time 类似。

答：

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

```
class Time
{
public:
  Time();
  Time(int h,int m,int s);
  void set(int h, int m, int s);
  void increment();
  void display() const;
  bool equal(Time &other_time) const;
  bool less_than(Time &other_time) const;
protected:
  int hour;
  int minute;
  int second;
};

enum TimeZone { W12=-12,W11,W10,W9,W8,W7,W6,W5,W4,W3,W2,W1,
                GMT,
                E1,E2,E3,E4,E5,E6,E7,E8,E9,E10,E11,E12};

class ExtTime: public Time
{
public:
  ExtTime() { tz = GMT; }
  ExtTime(int h,int m,int s,TimeZone t):Time(h,m,s) { tz = t; }
  void set(int h, int m, int s,TimeZone t)
  { Time::set(h,m,s);
    tz = t;
  }
  void display() const
  { if (tz<0)
      cout<<"西"<<-int(tz)<<"区\t";
    else if(tz==0)
      cout<<"格林威治(子午线)标准时间  (GMT)\t";
    else
      cout<<"东"<<int(tz)<<"区\t";
    Time::display();
  }
  bool equal(const ExtTime &other_time) const
  { Time t((hour+other_time.tz-tz+24)%24,minute,second);
   return t.equal(other_time);
  }
  bool less_than(const ExtTime &other_time) const
  { Time t((hour+other_time.tz-tz+24)%24,minute,second);
   return t.less_than(other_time);
  }
private:
  TimeZone tz;
};
```

7、 利用习题 6.9 的第 18 题中的 LinearList 类定义一个栈类。

答：

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

class LinearList
{
public:
  LinearList();
  ~LinearList();
  bool insert(int x, int pos);
  bool remove(int &x, int pos);
  int element(int pos) const;
  int search(int x) const;
  int length() const;
private:
  struct Node
  { int value;
    Node * next;
  };
  int count;
  Node *head;
};

////////////////////////////////////////////////////////////////////
class Stack: LinearList
{
public:
  bool push(int x);
  bool pop(int &x);
  LinearList::length;
};

bool Stack::push(int x)
{ return insert(x,0);
}

bool Stack::pop(int &x)
{ return remove(x,1);
}
```

8、 利用习题 6.9 的第 14、15 题中的时间类 Time 和日期类 Date，定义一个带日期的时间类 TimeWithDate。对该类对象能进行比较、增加（增加值为秒数）、相减（结果为秒数）等操作。

答：

```
#include <iostream>
using namespace std;
class Time
{
public:
  Time()
  { hour=minute=second=0;
```

```
  }
  Time(int h,int m,int s)
  { hour=h;minute=m;second=s;
  }
  void set(int h, int m, int s)
  { hour=h;
    minute=m;
    second=s;
  }
  void increment()
  { if (second==59)
    { if (minute==59)
      { if (hour==23)
        { hour=0;
          minute=0;
          second=0;
        }
        else
        { hour++;
          minute=0;
          second=0;
        }
     }
      else
      { minute++;
        second=0;
      }
    }
    else
      second++;
  }
  void display() const
  { cout<<hour<<':'<<minute<<':'<<second;
  }
  bool equal(const Time &time2) const
  { if (hour == time2.hour && minute == time2.minute && second == time2.second)
      return true;
    else
      return false;
  }
  bool less_than(const Time &time2) const
  { if (hour < time2.hour ||
       hour == time2.hour && minute < time2.minute ||
      hour == time2.hour && minute == time2.minute && second < time2.second)
      return true;
    else
      return false;
  }
protected:
  int hour;
  int minute;
  int second;
};
```

```
class Date
{ public:
    Date(int y, int m, int d)
   { year = y; month = m; day = d;
   }
    void increment()
   { int d;
     switch (month)
     { case 1: case 3: case 5: case 7: case 8: case 10: case 12:
         d = 31;
         break;
       case 4: case 6: case 9: case 11:
         d = 30;
         break;
       case 2:
         if (year%400 == 0 || year%4 == 0 && year%100 != 0)
           d = 29;
         else
           d = 28;
     }
     if (day < d)
       day++;
     else if (month != 12)
     { day = 1;
       month++;
     }
     else
     { day = 1;
       month = 1;
       year++;
     }
   }
   bool equal(const Date &date2) const
   { return year == date2.year && month == date2.month && day == date2.day;
   }
   bool less_than(const Date &date2) const
   { if (year < date2.year ||
         year == date2.year && month < date2.month ||
         year == date2.year && month == date2.month && day < date2.day)
       return true;
     else
       return false;
   }
    void display() const
   { cout << year << '-' << month << '-' << day;
   }
  protected:
    int year,month,day;
};

class TimeWithDate: public Time, public Date
{
```

```
public:
  TimeWithDate(int y,int mo,int d,int h,int mi,int s):Date(y,mo,d),Time(h,mi,s) {}
  bool equal(const TimeWithDate &td2) const;
  bool less_than(const TimeWithDate &td2) const;
  void increment();
  int  difference(const TimeWithDate & td2) const;
  void display() const;
};

bool TimeWithDate::equal(const TimeWithDate &td2) const
{ return Time::equal(td2) && Date::equal(td2);
}
bool TimeWithDate::less_than(const TimeWithDate &td2) const
{ if (Date::less_than(td2))
    return true;
  else if (Date::equal(td2) && Time::less_than(td2))
    return true;
  else
    return false;
}
void TimeWithDate::increment()
{ Time::increment();
  if (((Time *)this)->equal(Time(0,0,0)))
  //或 if (hour == 0 && minute == 0 && second == 0)
    Date::increment();
}
int TimeWithDate::difference(const TimeWithDate &td2) const
{ int days=0;
  if (less_than(td2))
    for (TimeWithDate td=*this; td.less_than(td2); td.increment()) days--;
  else
    for (TimeWithDate td=td2; td.less_than(*this); td.increment()) days++;
  return days;

}
void TimeWithDate::display() const
{ Date::display();
  cout << ' ';
  Time::display();
}
```

9、 如何定义两个类 A 和 B（B 是 A 的派生类），使得在程序中能够创建一个与指针变量 p（类型为 A *）所指向的对象是同类的对象？

答：

```
class A
{ ...
public:
  virtual A *create()
  { return new A;
  }
```

```
};

class B:public A
{ ...
public:
  A *create()
  { return new B;
  }
};

int main()
{ A *p;

  if (...)
    p = new A;
  else
    p = new B;
  ...
  A *q;
  q = p->create(); //创建一个与p所指向的对象同类的对象。
  ...
}
```

10、下面的设计有什么问题？如何解决？

```
class Rectangle //矩形类
{ public:
    Rectangle(double w, double h): width(w), height(h) {}
    void set_width(double w) { width = w; }
    void set_height(double h) { height = h; }
    double get_width() const { return width; }
    double get_height() const { return height; }
    double area() const { return width*height; }
    void print() const { cout << width << " " << height << endl; }
  private:
    double width;   //宽
    double height;  //高
};
class Square: public Rectangle //正方形类
{ public:
    Square(double s): Rectangle(s,s) {}
    void set_side(double s) //设置边长。
    { set_width(s);
      set_height(s);
    }
    double get_side() const //获取边长。
    { return get_width();
    }
};
```

答：Square 不能以 public 继承方式继承 Rectangle 类，否则，Rectangle 的所有 public 成员函数就能被

Square 类的对象访问，特别地，当用 set_width 和 set_height 分别对 Square 类的对象进行操作时，就可能破坏 Square 类对象的长、宽相等的特性。

解决办法是： 把 Square 定义成以 protected 或 private 方式从 Rectangle 继承。为了能对 Square 类的对象访问 Rectangle 中定义的 area 和 print，可在 Square 类中加上对 Rectangle 类成员的访问控制调整声明：

```
class Square: Rectangle //正方形类
{ public:
    ...
    Rectangle::area;
    Rectangle::print;
};
```