**信息科学与工程学院**

**2020－2021学年第一学期**

实 验 报 告

课程名称： 高级编程技术

实验名称： 实验二

专 业 班 级 通信工程 二班

学 生 学 号 201800121050

学 生 姓 名 孟麟芝

实 验 时 间 2020年10月11日

实验报告

## 【实验目的】

1.掌握理解C#这一面向对象编程语言中类、对象、接口、继承的概念和使用方法

2.掌握方法的重载、虚方法等。

3.学会将C#运用到实际场景中。

## 【实验要求】

1.求三角形的面积，要求用“方法的参数数组”、“接口”、“构造函数”、“类继承”、至少四种方法实现三角形的面积的方法。

2. 设计一个耗材管理类，用来保存和记录实验室耗材分类和记录。这个类的成员包括耗材名、使用者、实验室存量。至少提供两个方法：store 耗材的入库处理，show 显示耗材信息。程序运行时，可以从控制台上输入需入库耗材总数，根据这个总数创建耗材类对象数组，然后输入数据，最后可按照耗材名、使用者或实验室存量排序。

## 【第一个实验】

1.实验源代码

using System;

namespace AreaCalculator

{

    interface IShape

    {

        //理论上多边形的面积都是可以这样计算的

        public double getArea(double bottom, double height);

        //周长使用参数数组，可实现任意多边形周长的求解

        public double getPerimeter(params double[] sides);

    }

    class Triangle : IShape

    {

        public double getArea(double bottom, double height)

        {

            //这是使用底乘高计算面积的方法

            return height \* bottom / 2;

        }

        public double getArea(double[] sides)

        {

            //这是使用海伦公式计算面积的方法

            double halfPerimeter = getPerimeter(sides) / 2;

            return Math.Sqrt(halfPerimeter \* (halfPerimeter - sides[0]) \* (halfPerimeter - sides[1]) \* (halfPerimeter - sides[2]));

        }

        public double getArea(double a, double b, double angle)

        {

            //这是使用两边和其夹角计算面积的方法

            return a \* b \* Math.Sin(angle) / 2;

        }

        public double getPerimeter(params double[] sides)

        {

            double perimeter = 0;

            foreach (double i in sides)

            {

                perimeter += i;

            }

            return perimeter;

        }

    }

    class IsoscelesTriangle : Triangle

    {

        public double getArea(double waist, double bottom, int judge)

        {

            //这是等腰三角形利用腰和底计算面积

            double halfPerimeter = getPerimeter(waist, bottom) / 2;

            return Math.Sqrt(halfPerimeter \* (halfPerimeter - bottom)) \* (halfPerimeter - waist);

        }

        public double getPerimeter(double waist, double bottom)

        {

            return 2 \* waist + bottom;

        }

    }

    class EquilateralTriangle : IsoscelesTriangle

    {

        public double getArea(double side)

        {

            //这是等边三角形利用边长求面积

            return Math.Pow(side, 2) \* Math.Sqrt(3) / 4;

        }

        public double getPerimeter(double side)

        {

            return 3 \* side;

        }

    }

    class Program

    {

        static void Main(string[] args)

        {

            Triangle tri = new Triangle();

            IsoscelesTriangle isoTri = new IsoscelesTriangle();

            EquilateralTriangle eqtri = new EquilateralTriangle();

            double[] sides = new double[] { 3, 4, 5 };

            Console.WriteLine("首先是比较普遍的面积求法\n如用底乘高求，设底为3，高为4：则面积为：{0}\n还可以海伦公式求面积\n如设三边长分别为3，4，5，则面积为：{1}\n还可用两边与其夹角求面积\n如设两边长3，4，夹角为π/2，则面积为：{2}", tri.getArea(3, 4), tri.getArea(sides), tri.getArea(3, 4, Math.PI / 2));

            Console.WriteLine("下面是等腰三角形\n除去继承来的前面的三个方法，还可以利用腰长和底长求面积\n如设腰长为根号2，底长为2，则面积为：{0}", isoTri.getArea(Math.Sqrt(2), 2, 1));

            Console.WriteLine("下面是等边三角形\n除去从等腰三角形和基本三角形继承来的所有方法，即底乘高、海伦公式、腰长与底长求、两边与夹角求，还可以用一个边长求面积\n如设边长为2，则面积为：{0}", eqtri.getArea(2));

            Console.Read();

        }

    }

}

2.实验记录与说明分析

所有的多边形的求周长都是各边长度之和，而求面积方式各异，最常用的是底乘高的做法，故可以定义一个接口为类架构一个大框架。

1. **interface** IShape
2. {
3. //理论上多边形的面积都是可以这样计算的
4. **public** **double** getArea(**double** bottom, **double** height);
5. //周长使用参数数组，可实现任意多边形周长的求解
6. **public** **double** getPerimeter(**params** **double**[] sides);
7. }

对于普通的三角形，大致可以有三种计算面积的方法：底乘高法，两边与夹角求面积，海伦公式，故可以有下面的Triangle类，它的getArea()方法有三个重载，且都不是static的，需要用实例化对象来调用。

1. **class** Triangle : IShape
2. {
3. **public** **double** getArea(**double** bottom, **double** height)
4. {
5. //这是使用底乘高计算面积的方法
6. **return** height \* bottom / 2;
7. }
8. **public** **double** getArea(**double**[] sides)
9. {
10. //这是使用海伦公式计算面积的方法
11. **double** halfPerimeter = getPerimeter(sides) / 2;
12. **return** Math.Sqrt(halfPerimeter \* (halfPerimeter - sides[0]) \* (halfPerimeter - sides[1]) \* (halfPerimeter - sides[2]));
13. }
14. **public** **double** getArea(**double** a, **double** b, **double** angle)
15. {
16. //这是使用两边和其夹角计算面积的方法
17. **return** a \* b \* Math.Sin(angle) / 2;
18. }
19. **public** **double** getPerimeter(**params** **double**[] sides)
20. {
21. **double** perimeter = 0;
22. **foreach** (**double** i **in** sides)
23. {
24. perimeter += i;
25. }
26. **return** perimeter;
27. }
28. }

更特殊一些的三角形是等腰三角形，可以由三角形类派生一个等腰三角形类，继承父类的所有方法，同时还可添加或重写一些方法，要注意的是，如果重写（覆盖）方法，需要在父类中使用virtual，在子类中加override，本次实验中并没有涉及到重写。

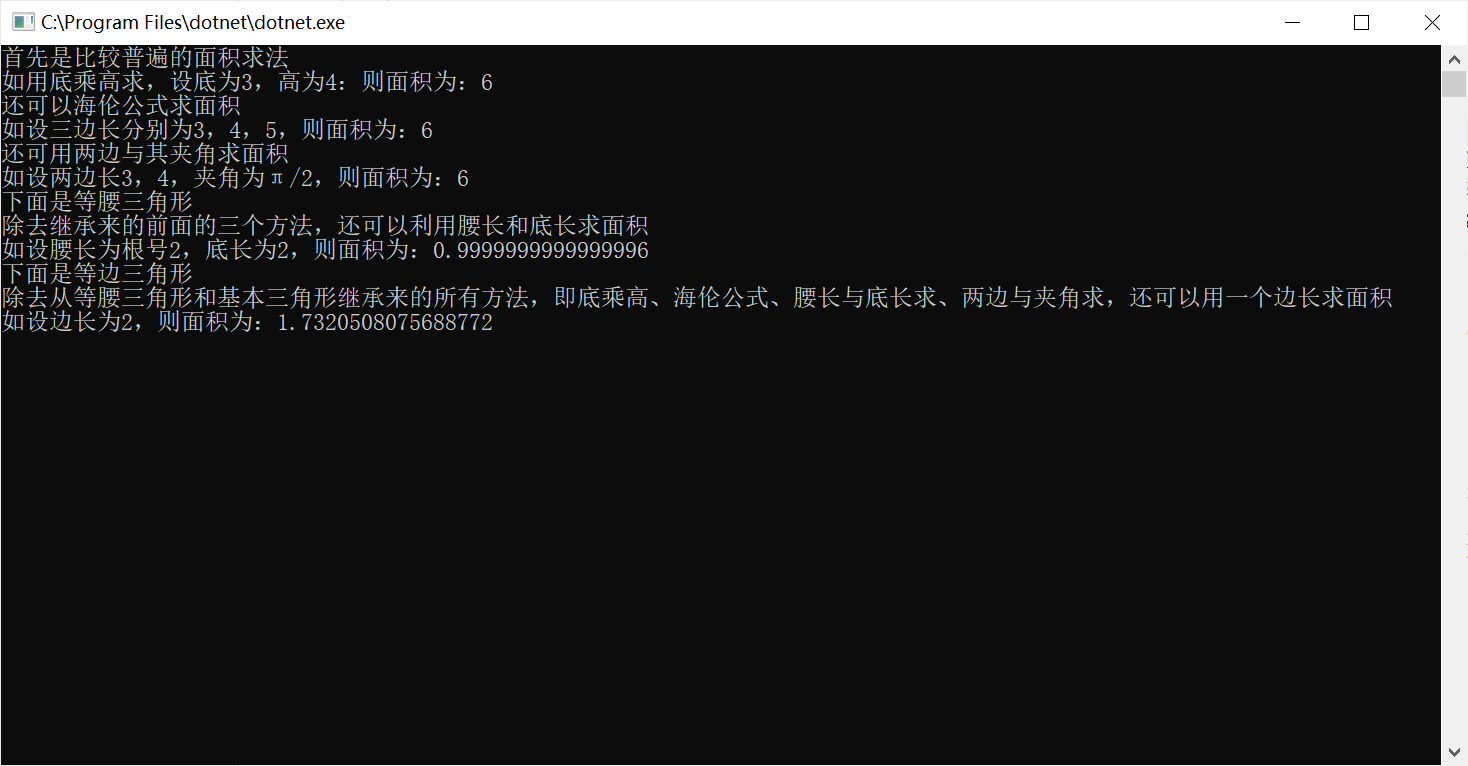
下面的等腰三角形类继承了普通三角形所有面积的计算方法，同时添加了getArea()的一个重载，即使用腰和底计算面积。还添加了getPerimeter()的一个重载，可以使用腰和底计算周长。

1. **class** IsoscelesTriangle : Triangle
2. {
3. **public** **double** getArea(**double** waist, **double** bottom, **int** judge)
4. {
5. //这是等腰三角形利用腰和底计算面积
6. **double** halfPerimeter = getPerimeter(waist, bottom) / 2;
7. **return** Math.Sqrt(halfPerimeter \* (halfPerimeter - bottom)) \* (halfPerimeter - waist);
8. }
9. **public** **double** getPerimeter(**double** waist, **double** bottom)
10. {
11. **return** 2 \* waist + bottom;
12. }
13. }

比等腰三角形更加特殊的是等边三角形，只需要一条边即可得到它的所有信息，故可以从等腰三角形类派生出等边三角形类，有：

1. **class** EquilateralTriangle : IsoscelesTriangle
2. {
3. **public** **double** getArea(**double** side)
4. {
5. //这是等边三角形利用边长求面积
6. **return** Math.Pow(side, 2) \* Math.Sqrt(3) / 4;
7. }
8. **public** **double** getPerimeter(**double** side)
9. {
10. **return** 3 \* side;
11. }
12. }

下面是程序运行的结果，这些结果在误差范围内是正确的，另外可见涉及到开根等运算的方法精度都是有限的：



## 【第二个实验】

1.实验源代码

using System;

namespace LaboratoryManagement

{

    public class Materials

    {

        private string name, user;

        private int total;

        //这里定义了三个属性，每个属性都是可读可写的

        public string Name

        {

            get { return name; }

            set { name = value; }

        }

        public string User

        {

            get { return user; }

            set { user = value; }

        }

        public int Total

        {

            get { return total; }

            set { total = value; }

        }

        //构造函数

        public Materials()

        {

            name = "";

            user = "";

            total = 0;

        }

        public Materials(string name, string user, int total)

        {

            this.name = name;

            this.user = user;

            this.total = total;

        }

        //若不加ref，传入的是一个引用值的副本，本实验中加与不加ref效果是一样的

        public void store(ref Materials material)

        {

            name = material.name;

            user = material.user;

            total = material.total;

        }

        public void show()

        {

            Console.WriteLine("耗材名：{0}，使用者：{1}，用量：{2}", name, user, total);

        }

    }

    class Program

    {

        static void Main(string[] args)

        {

            //这里是一个中转的对象，用于将数据存入类数组

            Materials card = new Materials();

            Console.Write("请问总共要记录多少种耗材？");

            string strCounts = Console.ReadLine();

            int counts = int.Parse(strCounts);

            //这里是一个类类型的数组

            Materials[] material = new Materials[counts];

            //建立一个“目录”方便后面排序

            int[] index = new int[counts];

            //类数组内的元素需要手动进行实例化

            for (int i = 0; i < counts; i++)

            {

                material[i] = new Materials();

            }

            for (int i = 0; i < counts; i++)

            {

                Console.Write("请输入耗材名： ");

                card.Name = Console.ReadLine();

                Console.Write("请输入耗材使用者： ");

                card.User = Console.ReadLine();

                Console.Write("请输入这种耗材的用量： ");

                strCounts = Console.ReadLine();

                card.Total = int.Parse(strCounts);

                material[i].store(ref card);

                index[i] = i;

            }

            Console.Write("请选择按什么关键字进行排序(1.按耗材名，2.按使用者，3.按用量)");

            strCounts = Console.ReadLine();

            int choice = int.Parse(strCounts);

            sortMaterials(choice, material, index);

            for (int i = 0; i < counts; i++)

            {

                material[index[i]].show();

            }

            Console.Read();

        }

        static void sortMaterials(int choice, Materials[] material, int[] index)

        {

            switch (choice)

            {

                case 1:

                    //对耗材名进行排序，使用的是冒泡法

                    for (int i = 0; i < index.Length - 1; i++)

                    {

                        for (int m = 0; m < index.Length - i - 1; m++)

                        {

                            if (string.Compare(material[index[m]].Name, material[index[m + 1]].Name) > 0)

                            {

                                int temp = index[m];

                                index[m] = index[m + 1];

                                index[m + 1] = temp;

                            }

                        }

                    }

                    break;

                case 2:

                    //对使用者进行排序，使用的是冒泡法

                    for (int i = 0; i < index.Length - 1; i++)

                    {

                        for (int m = 0; m < index.Length - i - 1; m++)

                        {

                            if (string.Compare(material[index[m]].User, material[index[m + 1]].User, false) > 0)

                            {

                                int temp = index[m];

                                index[m] = index[m + 1];

                                index[m + 1] = temp;

                            }

                        }

                    }

                    break;

                case 3:

                    //对用量进行排序，使用的是冒泡法

                    for (int i = 0; i < index.Length - 1; i++)

                    {

                        for (int m = 0; m < index.Length - i - 1; m++)

                        {

                            if (material[index[m]].Total > material[index[m + 1]].Total)

                            {

                                int temp = index[m];

                                index[m] = index[m + 1];

                                index[m + 1] = temp;

                            }

                        }

                    }

                    break;

                default: break;

            }

        }

    }

}

2.实验记录与说明分析

该代码借鉴了课本中图书管理系统的实现方式，其核心为一“耗材类”，它拥有三个属性：名称、使用者、数量，这三个属性都设置成可读可写的，含有两个方法，即store()和show()方法，分别用来存入一种耗材的信息和显示出信息。

下面是这个类的代码，它实现了上述功能。

1. **public** **class** Materials
2. {
3. **private** string name, user;
4. **private** **int** total;
5. //这里定义了三个属性，每个属性都是可读可写的
6. **public** string Name
7. {
8. get { **return** name; }
9. set { name = value; }
10. }
11. **public** string User
12. {
13. get { **return** user; }
14. set { user = value; }
15. }
16. **public** **int** Total
17. {
18. get { **return** total; }
19. set { total = value; }
20. }
21. //构造函数
22. **public** Materials()
23. {
24. name = "";
25. user = "";
26. total = 0;
27. }
28. **public** Materials(string name, string user, **int** total)
29. {
30. **this**.name = name;
31. **this**.user = user;
32. **this**.total = total;
33. }
34. //本实验中这里若不加ref效果也是一样的
35. **public** **void** store(ref Materials material)
36. {
37. name = material.name;
38. user = material.user;
39. total = material.total;
40. }
41. **public** **void** show()
42. {
43. Console.WriteLine("耗材名：{0}，使用者：{1}，用量：{2}", name, user, total);
44. }
45. }

目前我们遇到的问题就是，如何存储多种耗材？其实这也非常简单，只需要使用一个类类型的数组即可，其长度可以根据用户的输入而确定，这一点显得十分灵活，但要注意的是必须手动对每个元素进行实例化才可对这个数组内的对象进行操作。

1. **class** Program
2. {
3. **static** **void** Main(**string**[] args)
4. {
5. //这里是一个中转的对象，用于将数据存入类数组
6. Materials card = **new** Materials();
7. Console.Write("请问总共要记录多少种耗材？");
8. **string** strCounts = Console.ReadLine();
9. **int** counts = **int**.Parse(strCounts);
10. //这里是一个类类型的数组
11. Materials[] material = **new** Materials[counts];
12. //建立一个“目录”方便后面排序
13. **int**[] index = **new** **int**[counts];
14. //类数组内的元素需要手动进行实例化
15. **for** (**int** i = 0; i < counts; i++)
16. {
17. material[i] = **new** Materials();
18. }
19. }

对用户输入数据的存储经过了一个card对象进行中转，先将数据存入card，再把card这一对象的引用使用ref关键字传递给要存入的对象的store()方法，这一过程与将数组作为函数参数的原理是一致的，因为数组也是一个对象，其数组名是对这个对象的引用。

最后，需要再将card中的数据转移到要存入的对象内。这一操作类似C语言中将指针作为函数参数。

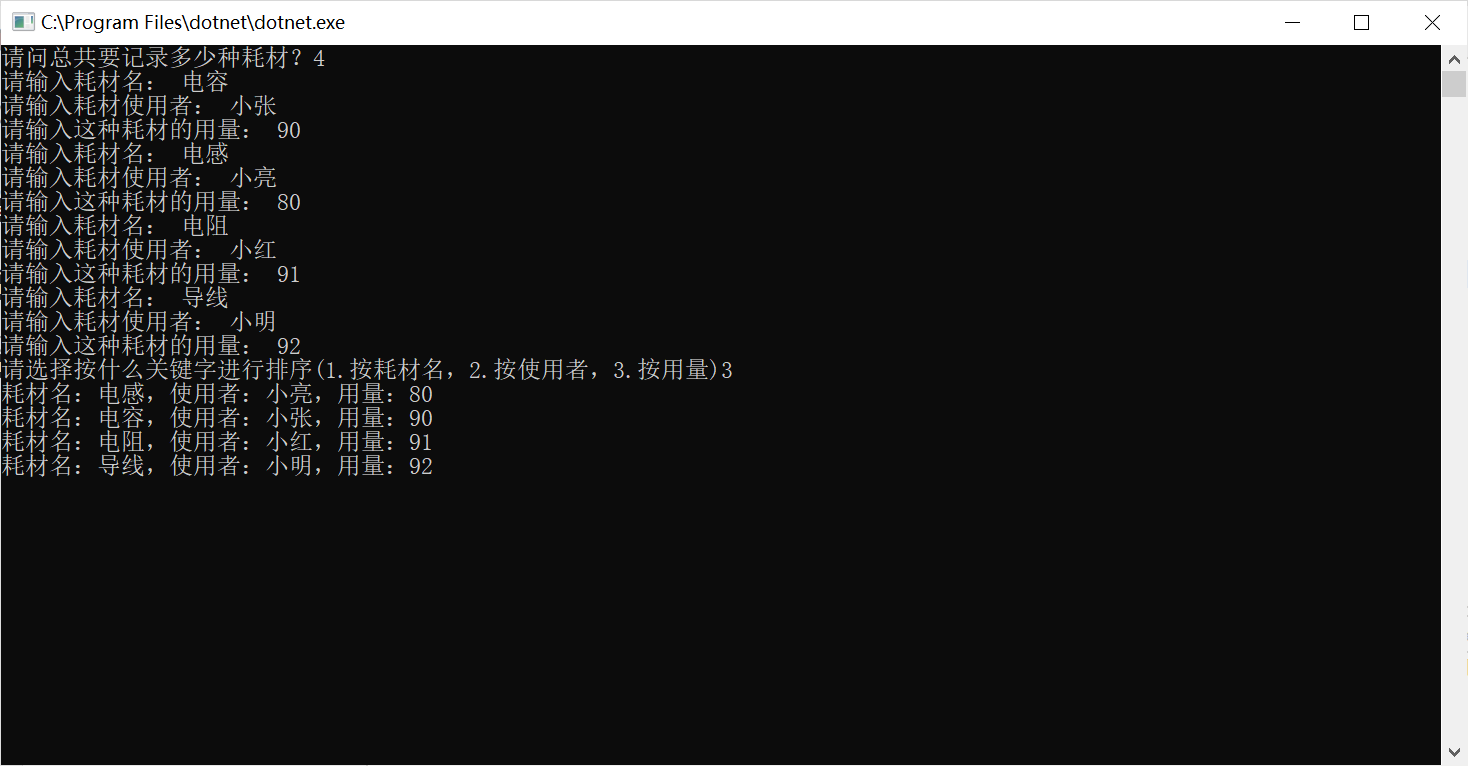
实际上，这里不加ref关键字也是可以的，只是这时传给方法的是引用值的一个副本。但改变其引用的对象时，仍会改变实参所引用的对象。

1. **for** (**int** i = 0; i < counts; i++)
2. {
3. Console.Write("请输入耗材名： ");
4. card.Name = Console.ReadLine();
5. Console.Write("请输入耗材使用者： ");
6. card.User = Console.ReadLine();
7. Console.Write("请输入这种耗材的用量： ");
8. strCounts = Console.ReadLine();
9. card.Total = **int**.Parse(strCounts);
10. material[i].store(**ref** card);
11. index[i] = i;
12. }

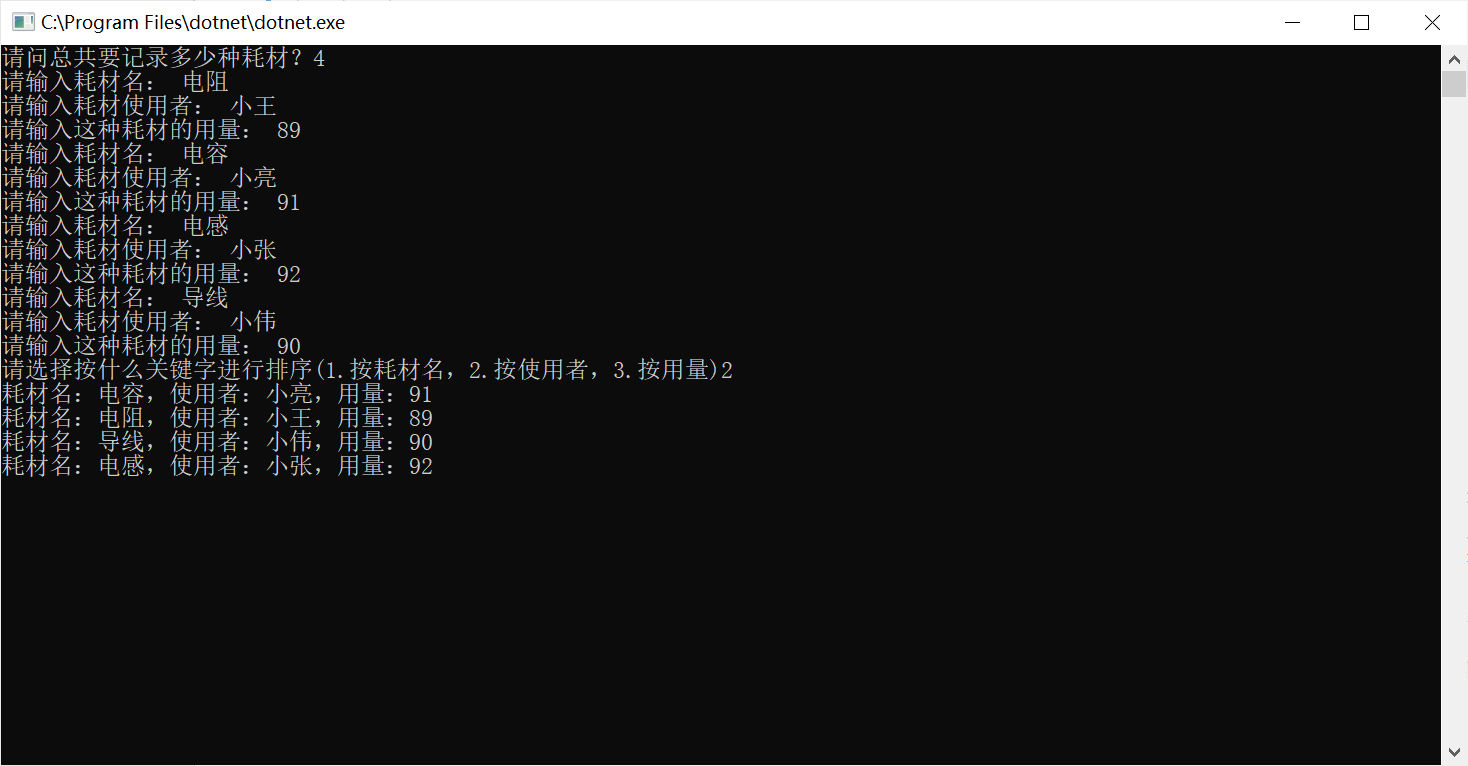
然后我们需要对存入的数据进行排序，这里使用了冒泡排序

1. **static** **void** sortMaterials(**int** choice, Materials[] material, **int**[] index)
2. {
3. //对耗材名进行排序，使用的是冒泡法
4. **for** (**int** i = 0; i < index.Length - 1; i++)
5. {
6. **for** (**int** m = 0; m < index.Length - i - 1; m++)
7. {
8. **if** (**string**.Compare(material[index[m]].Name, material[index[m+1]].Name) > 0)
9. {
10. **int** temp = index[m];
11. index[m] = index[m + 1];
12. index[m + 1] = temp;
13. }
14. }
16. }
17. //对使用者进行排序，使用的是冒泡法
18. **for** (**int** i = 0; i < index.Length - 1; i++)
19. {
20. **for** (**int** m = 0; m < index.Length - i - 1; m++)
21. {
22. **if** (**string**.Compare(material[index[m]].User, material[index[m+1]].User) > 0)
23. {
24. **int** temp = index[m];
25. index[m] = index[m + 1];
26. index[m + 1] = temp;
27. }
28. }
30. }
31. //对用量进行排序，使用的是冒泡法
32. **for** (**int** i = 0; i < index.Length - 1; i++)
33. {
34. **for** (**int** m = 0; m < index.Length - i - 1; m++)
35. {
36. **if** (material[index[m]].Total > material[index[m+1]].Total)
37. {
38. **int** temp = index[m];
39. index[m] = index[m + 1];
40. index[m + 1] = temp;
41. }
42. }
44. }
45. }

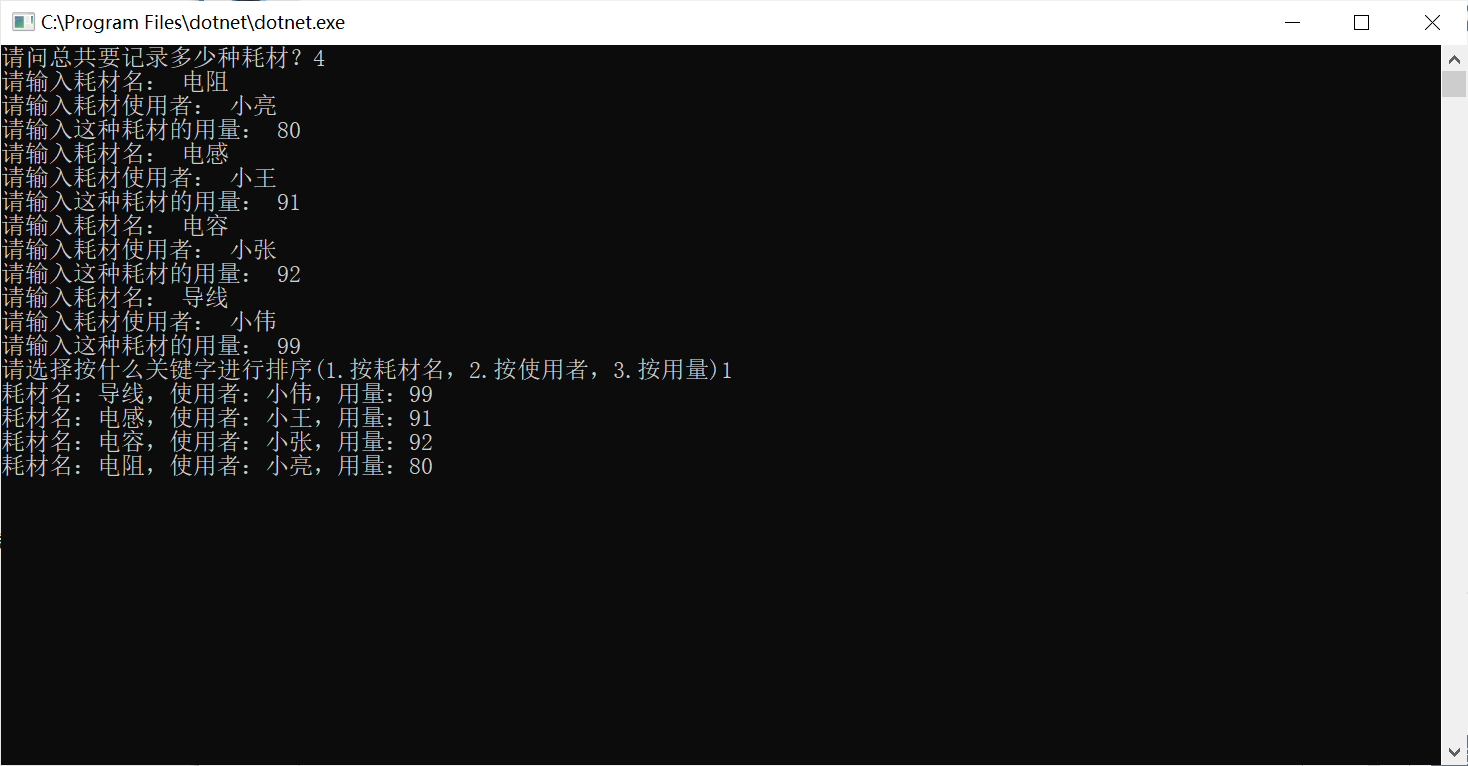
最后将存好的数据显示输出即可，先按使用量排序，得到运行结果如下：



按照使用者排序：



按照元件名排序：



## 【实验心得】

经过本次实验，对C#的类、方法、对象、接口有了更深的认识，逐渐意识到了面向对象的含义，还直观的感受了了重写、继承等操作的过程与要求，如实验中三角形面积计算的问题实现。

在实验过程中，还学习了一些IDE中快捷方便的操作，最重要的操作就是如何快速查看声明与引用，这一点对库函数的理解帮助巨大。虽然库函数完全可以被当做一个“黑匣子”去调用，不过这样永远触及不到它的本质，ctrl+鼠标左键，直接跳转到其声明与实现，则可以直观的看出它的来龙去脉，对学习的帮助巨大。

另外，学会看官方文档也是一个重要的能力，作为微软自己开发的语言，VS中C#的文档可谓是十分详尽了。