实验 1 熟悉 C#语言：数组和类的定义与项目管理

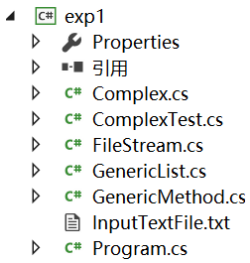
实验目的：

理解 C#的基本概念及其基本操作，认识和使用类库中若干常用的类型，重点是数组的处理和新的类型的定义。

题意：

1. 熟练掌握 Visual Studio 进行项目和类的建立和管理。新建空白解决方案（如称作 DSAGL-EXP）；在新的解决方案中新建一个控制台应用程序类型的项目（如称作 explapp）。本次实验的各项内容各自以单独的类实现，但都置于 explapp 项目中，通过设置项目属性选择不同的可启动（运行）类，以及设置所需的命令行参数。

本次实验完成后，项目及类源代码文件结构如下所示：



2. 定义一个含 Main 方法的类（ArrayTest.cs），在其中定义和随机初始化一个（具有 20 个元素/值域在-99 到 99 的）整数数组，在数组中查找特定数据，对数组中的数据进行排序。重新随机初始化数组，对数组中的数据按绝对值大小进行排序。认识和使用类库中 Array、Random、Console 等常用的类型。

3. 定义一个含 Main 方法的类（GenericList.cs），在其中利用 List<T>类定义和初始化一个 int 类型的线性表，在表中添加（Add）和插入（Insert）新的元素；定义一个自定义 Student 类，定义和初始化一个 Student 类型的线性表，在表中添加（Add）和插入（Insert）新的元素。认识和使用 List<T>泛型类。

4. 定义一个含 Main 方法的类（GenericMethod.cs），在其中利设计一个泛型方法 swap，能交换不同类型的两个变量的值。认识和使用泛型方法。

5. 设计定义一个复数类（Complex.cs），实现复数的基本操作。要认识到，在科学与工程数值计算中，复数运算是基本运算之一。当需要频繁操作复数时，需要自定义复数类。定义一个含 Main 方法、测试复数类的类（ComplexTest.cs），测试复数的基本操作。

6. 定义一个含 Main 方法的类（FileStreamTest.cs），在其中打开、读入文本文件，将其内容逐行输出到一个新文件，实现文件的拷贝，记录和显示拷贝过程的时间。认识和使用 File、StreamReader、StreamWriter、StopWatch 等类。

实验要点解释和参考实现

2. 定义一个含 Main 方法的类（ArrayTest.cs），在其中定义和随机初始化一个（具有 20 个元素/值域在-99 到 99 的）整数数组，在数组中查找特定数据，对数组中的数据进行排序。重新随机初始化数组，对数组中的数据按绝对值大小进行排序。认识和使用类库中 Array、Random、Console 等常用的类型。

源文件 ArrayTest.cs

```
class ArrayTest {
    static void Main(string[] args) {
        int[] a = new int[20];

        RandomizeData(a);  // 自定义方法：相对独立的一段功能代码，便于多处调用
        Show(a);           // 自定义方法：相对独立的一段功能代码，便于多处调用

        int i = Array.IndexOf<int>(a, 10);
        Console.WriteLine("10's index is: {0}", i);
        Console.WriteLine("Min of the array is {0}", a.Min());
        Console.WriteLine("Sorted Array: ");
        Array.Sort(a);
        Show(a);

        RandomizeData(a, -99, 100);
        Show(a);
        Console.WriteLine("Sorted by Absolute Value: ");
        Array.Sort(a, new AbsComparer());
        Show(a);
    }

    private static void Show(int[] a) {
        for (int i = 0; i < a.Length; i++) { //数组的Length属性告知数组元素个数
            Console.Write(a[i] + " ");
        }
        Console.WriteLine();

    }

    private static void RandomizeData(int[] a, int minValue, int maxValue) {
        Random rd = new Random();       //面向对象，需要随机数就找Random对象

        for (int i = 0; i < a.Length; i++) {
            a[i] = rd.Next(minValue, maxValue);
        }
    }
class AbsComparer : IComparer<int> {
    public int Compare(int x, int y) {
        return (Math.Abs(x)).CompareTo(Math.Abs(y));
```

```
    }
  }
}
```

3/4. 泛型方法与泛型类

泛型通常与集合类以及作用于集合的方法一起使用。C#语言中泛型的优越性在下面的一段例子中应能较好的显示出来。对于同样的运算逻辑（例子中是交换两个变量的内容），但仅是数据的类型不一样，可能就需要定义一堆相似的方法；而应用泛型特性则可仅定义一个泛型方法（例子中是 swap<T>）。

源文件 GenericMethod.cs

```
static void Main(string[] args) {
     int a = 3;  int b = 7;

    swapint(ref a, ref b);
    double ad = 3.5; double bd = 7.5;
    swapdouble(ref ad, ref bd);

    swap<int>(ref a, ref b);
}

static void swapint(ref int a, ref int b) {
    int x = a;
    a = b;
    b = x;
}

static void swapdouble(ref double a, ref double b) {
    double x = a;
    a = b;
    b = x;
}

static void swap<T>(ref T a, ref T b) {
    T x = a;
    a = b;
    b = x;
}
```

泛型类 List<T>的使用。参考代码

声明并构造整型数的列表：

```
List<int>  a = new List<int>(); // 声明并构造整型数的列表
a.Add(86); a.Add(100);          // 向列表中添加整型元素
```

也可以声明并构造自定义类型的列表：

```
        List< Student>  st = new List< Student>(); // 声明并构造学生列表
        st.Add(new Student (200518001, “王兵”, 92)); //向列表中添加学生类型元素
```

源文件 GenericList.cs

```
class GenericList {
    static void Main(string[] args) {
        List<Student> stuList = new List<Student>() { new Student(3016, "张超", 89),
        new Student(3053, "马飞", 80),new Student(3041, "刘羽", 96),
        new Student(3025, "赵备", 79), new Student(3039, "关云", 85)};

        stuList.Insert(2, new Student(3000, "马超", 95));
        foreach (var item in stuList) {
            Console.WriteLine(item.ToString());
        }
    }
}

class Student {
    int studentID;
    string name;
    double mark;

    public Student(int id, string name, double mark) {
        this.studentID = id;
        this.name = name;
        this.mark = mark;
    }

    public int StudentID {
        get { return studentID; }
        set { studentID = value; }
    }
    public string Name {
        get { return name; }
        set { name = value; }
    }
    public double Mark {
        get { return mark; }
        set { mark = value; }
    }

    public override string ToString() {
        return studentID.ToString() + "-" + name;
    }
}
```

5. 设计定义一个复数类（Complex.cs），实现复数的基本操作。要认识到，在科学与工程数值计算中，复数运算是基本运算之一。当需要频繁操作复数时，需要自定义复数类。定义一个含 Main 方法、测试复数类的类（ComplexTest.cs），测试复数的基本操作。

源文件 ComplexTest.cs

```
class ComplexTest {
    static void Main(string[] args) {
        Complex[] ca = new Complex[10];
        RandomizeData(ca,-10,10);
        Show(ca);

        //int i = Array.IndexOf<Complex>(ca, new Complex());
        int i = Array.IndexOf<Complex>(ca, ca[5]);
        Console.WriteLine("{0}'s index is: {1}", ca[5], i);
        Console.WriteLine("Sorted Array: ");
        Array.Sort(ca);
        Show(ca);
    }

    private static void Show(Complex[] a) {
        for (int i = 0; i < a.Length; i++) {
            Console.Write(a[i] + " ; ");
        }
        Console.WriteLine();

    }

    private static void RandomizeData(Complex[] a, int minValue, int maxValue) {
        Random rd = new Random();
        int k = 0;
        for (int i = 0; i < a.Length; i++) {
            k = rd.Next(minValue, maxValue+1);
            a[i] = new Complex(k * rd.NextDouble(), k * rd.NextDouble());
        }
    }
}
```

源文件 Complex.cs

```
/*
 * 操作复数的类Complex
 *
 * wwwang编制
 */
```

```
using System;
using System.Collections.Generic;

namespace introduction {
    /**
     * 操作复数的类Complex

     * @author wwwang
     * @version 1.0
     */
    public class Complex: IComparable {
        private double rp = 0.0;            // 复数的实部
        private double ip = 0.0;            // 复数的虚部
        private static double eps = 0.0;           // 缺省精度

        /**
         * 属性: 实部
         */
        public double RealPart {
            get {
                return rp;
            }
            set {
                rp = value;
            }
        }

        /**
         * 属性: 虚部
         */
        public double ImaginaryPart {
            get {
                return ip;
            }
            set {
                ip = value;
            }
        }

        /**
         * 属性: Eps
         */
        public static double Eps {
            get {
```

```
        return eps;
    }
    set {
        eps = value;
    }
}

/**
 * 基本构造函数
 */
public Complex() {
}

/**
 * 指定值构造函数
 *
 * @param r - 指定的实部
 * @param i - 指定的虚部
 */
public Complex(double r, double i) {
    rp = r;
    ip = i;
}

/**
 * 拷贝构造函数
 *
 * @param sc - 源复数
 */
public Complex(Complex sc) {
    rp = sc.rp;
    ip = sc.ip;
}

/**
 * 根据"a,b"形式的字符串来构造复数，以a为复数的实部，b为复数的虚部
 *
 * @param s - "a,b"形式的字符串，a为复数的实部，b为复数的虚部
 * @param sDelim - a, b之间的分隔符
 */
public Complex(string s, string sDelim) {
    SetValue(s, sDelim);
}
```

```
/**
 * 将"a,b"形式的字符串转化为复数，以a为复数的实部，b为复数的虚部
 *
 * @param s - "a,b"形式的字符串，a为复数的实部，b为复数的虚部
 * @param sDelim - a, b之间的分隔符
 */
public void SetValue(string s, string sDelim) {
    int nPos = s.IndexOf(sDelim);
    if (nPos == -1) {
        s = s.Trim();
        rp = Double.Parse(s);
        ip = 0;
    }
    else {
        int nLen = s.Length;
        string sLeft = s.Substring(0, nPos);
        string sRight = s.Substring(nPos + 1, nLen - nPos - 1);
        sLeft = sLeft.Trim();
        sRight = sRight.Trim();
        rp = Double.Parse(sLeft);
        ip = Double.Parse(sRight);
    }
}

/**
 * 重载 + 运算符
 *
 * @return Complex对象
 */
public static Complex operator +(Complex c1, Complex c2) {
    Complex rs = new Complex(c1);
    rs.rp += c2.rp;
    rs.ip += c2.ip;
    return rs;
}

/**
 * 重载 - 运算符
 *
 * @return Complex对象
 */
public static Complex operator -(Complex c1, Complex c2) {
    Complex rs = new Complex(c1);
```

```
        rs.rp -= c2.rp;
        rs.ip -= c2.ip;
        return rs;
    }

    /**
     * 重载 * 运算符
     *
     * @return Complex对象
     */
    public static Complex operator *(Complex c1, Complex c2) {
        Complex rs = new Complex(c1);
        rs.rp = rs.rp * c2.rp - rs.ip * c2.ip;
        rs.ip = rs.rp * c2.ip + rs.ip * c2.rp;
        return rs;
    }

    /**
     * 重载 / 运算符
     *
     * @return Complex对象
     */
    public static Complex operator /(Complex c1, Complex c2) {
        Complex rs = new Complex(c1);
        rs.Divide(c2);
        return rs;
    }

    /**
     * 重载 double 运算符
     *
     * @return double值
     */
    public static implicit operator double(Complex c) {
        return c.Abs();
    }

    /**
     * 将复数转化为"a+bj"形式的字符串
     *
     * @return string 型，"a+bj"形式的字符串
     */
    public override string ToString() {
        string s;
```

```
    if (rp != 0.0) {
        if (ip > 0)
            s = rp.ToString("F") + "+" + ip.ToString("F") + "j";
        else if (ip < 0) {
            double absImag = -1 * ip;
            s = rp.ToString("F") + "-" + absImag.ToString("F") + "j";
        }
        else
            s = rp.ToString("F");
    }
    else {
        if (ip > 0)
            s = ip.ToString("F") + "j";
        else if (ip < 0) {
            double absImag = -1 * ip;
            s = absImag.ToString("F") + "j";
        }
        else
            s = rp.ToString("F");
    }

    return s;
}

/**
 * 比较两个复数是否相等
 *
 * @param other - 用于比较的复数
 * @return bool型，相等则为true，否则为false
 */
public override bool Equals(object other) {
    Complex c = other as Complex;
    if (c == null)
        return false;
    return Math.Abs(rp - c.rp) <= eps &&
        Math.Abs(ip - c.ip) <= eps;
}

/**
 * 因为重写了Equals，因此必须重写GetHashCode
 *
 * @return int型，返回复数对象散列码
 */
public override int GetHashCode() {
```

```
    return (int)this.Abs();
}

/**
 * 给复数赋值
 *
 * @param x - 用于给复数赋值的源复数
 * @return Complex型，与x相等的复数
 */
public Complex SetValue(Complex x) {
    rp = x.rp;
    ip = x.ip;
    return this;
}

/**
 * 实现复数的加法
 *
 * @param c - 与指定复数相加的复数
 * @return Complex型，指定复数与c相加之和
 */
public Complex Add(Complex c) {
    this.rp += c.rp;
    this.ip += c.ip;
    return this;
}

/**
 * 实现复数的减法
 *
 * @param c - 与指定复数相减的复数
 * @return Complex型，指定复数减去c之差
 */
public Complex Subtract(Complex c) {
    this.rp -= c.rp;
    this.ip -= c.ip;
    return this;
}

/**
 * 实现复数的乘法
 *
 * @param c - 与指定复数相乘的复数
 * @return Complex型，指定复数与c相乘之积
```

```
 */
public Complex Multiply(Complex c) {
    this.rp = this.rp * c.rp - this.ip * c.ip;
    this.ip = this.rp * c.ip + this.ip * c.rp;
    return this;
}

/**
 * 实现复数的除法
 *
 * @param c - 与指定复数相除的复数
 * @return Complex型，指定复数除与c之商
 */
public Complex Divide(Complex c) {
    double e, f, x, y;

    if (Math.Abs(c.rp) >= Math.Abs(c.ip)) {
        e = c.ip / c.rp;
        f = c.rp + e * c.ip;

        x = (rp + ip * e) / f;
        y = (ip - rp * e) / f;
    }
    else {
        e = c.rp / c.ip;
        f = c.ip + e * c.rp;

        rp = (rp * e + ip) / f;
        ip = (ip * e - rp) / f;
    }

    return this;
}

/**
 * 计算复数的模
 *
 * @return double型，指定复数的模
 */
public double Abs() {
    return (Math.Sqrt(rp*rp +ip*ip));
}
```

```
/**
 * 计算复数的实幂指数
 *
 * @param x - 待求实幂指数的幂次
 * @return Complex型，复数的实幂指数值
 */
public Complex Pow(double x) {
    // 常量
    const double PI = 3.14159265358979;

    // 局部变量
    double r, t;

    // 特殊值处理
    if ((rp == 0) && (ip == 0))
        return this;

    // 幂运算公式中的三角函数运算
    if (rp == 0) {
        if (ip > 0)
            t = 1.5707963268;
        else
            t = -1.5707963268;
    }
    else {
        if (rp > 0)
            t = Math.Atan2(ip, rp);
        else {
            if (ip >= 0)
                t = Math.Atan2(ip, rp) + PI;
            else
                t = Math.Atan2(ip, rp) - PI;
        }
    }

    // 模的幂
    r = Math.Exp(x * Math.Log(Math.Sqrt(rp * rp + ip * ip)));

    // 复数的实幂指数
    rp = r * Math.Cos(x * t);
    ip = r * Math.Sin(x * t);
    return this;
}
```

```
/**
 * 计算复数的自然对数
 *
 * @return Complex型，复数的自然对数值
 */
public Complex Log() {
    double p = Math.Log(Math.Sqrt(rp * rp + ip * ip));
    rp = p; ip = Math.Atan2(ip, rp);
    return this;
}

/**
 * 计算复数的正弦
 *
 * @return Complex型，复数的正弦值
 */
public Complex Sin() {
    int i;
    double x, y, y1, br, b1, b2;
    double[] c = new double[6];

    // 切比雪夫公式的常数系数
    c[0] = 1.13031820798497;
    c[1] = 0.04433684984866;
    c[2] = 0.00054292631191;
    c[3] = 0.00000319843646;
    c[4] = 0.00000001103607;
    c[5] = 0.00000000002498;

    y1 = Math.Exp(ip);
    x = 0.5 * (y1 + 1 / y1);
    br = 0;
    if (Math.Abs(ip) >= 1)
        y = 0.5 * (y1 - 1 / y1);
    else {
        b1 = 0;
        b2 = 0;
        y1 = 2 * (2 * ip * ip - 1);
        for (i = 5; i >= 0; --i) {
            br = y1 * b1 - b2 - c[i];
            if (i != 0) {
                b2 = b1;
                b1 = br;
```

```
            }
        }

        y = ip * (br - b1);
    }

    // 组合计算结果
    rp = x * Math.Sin(rp);
    ip = y * Math.Cos(rp);

    return this;
}

/**
 * 计算复数的余弦
 *
 * @return Complex型，复数的余弦值
 */
public Complex Cos() {
    int i;
    double x, y, y1, br, b1, b2;
    double[] c = new double[6];

    // 切比雪夫公式的常数系数
    c[0] = 1.13031820798497;
    c[1] = 0.04433684984866;
    c[2] = 0.00054292631191;
    c[3] = 0.00000319843646;
    c[4] = 0.00000001103607;
    c[5] = 0.00000000002498;

    y1 = Math.Exp(ip);
    x = 0.5 * (y1 + 1 / y1);
    br = 0;
    if (Math.Abs(ip) >= 1)
        y = 0.5 * (y1 - 1 / y1);
    else {
        b1 = 0;
        b2 = 0;
        y1 = 2 * (2 * ip * ip - 1);
        for (i = 5; i >= 0; --i) {
            br = y1 * b1 - b2 - c[i];
            if (i != 0) {
                b2 = b1;
```

```
                    b1 = br;
                }
            }

            y = ip * (br - b1);
        }

        // 组合计算结果
        rp = x * Math.Cos(rp);
        ip = -y * Math.Sin(rp);

        return this;
    }

    public int CompareTo(object obj) {
        Complex c = obj as Complex;
        if (c == null)
            throw new ArgumentException("CompareTo（Complex）");
        if (this.Equals(c))
            return 0;
        double d1, d2;
        d1 = this.Abs();
        d2 = c.Abs();
        if (d1 < d2)
            return -1;
        else
            return 1;
    }
}
}
```

6. 定义一个含 Main 方法的类（FileStreamTest.cs），在其中打开、读入文本文件，将其内容逐行输出到一个新文件，实现文件的拷贝，记录和显示拷贝过程的时间。认识和使用 File、StreamReader、StreamWriter、StopWatch 等类。

源文件 FileStreamTest.cs

```
class FileStreamTest {
    static void Main(string[] args) {
        try {
            StreamReader infile = File.OpenText(@"InputTextFile.txt");
            StreamWriter outfile = File.CreateText(@"OutputTextFile.txt");
            string tmpstr = null;
            Stopwatch timer = new Stopwatch();

            timer.Start();
```

```
            while ((tmpstr = infile.ReadLine()) != null) {
                outfile.WriteLine(tmpstr);
            }
            timer.Stop();
            //Console.WriteLine("Elapsed time = {0} ms", timer.ElapsedMilliseconds);
            Console.WriteLine("Elapsed time = {0} Ticks", timer.ElapsedTicks);
            infile.Close();
            outfile.Close();

            Console.WriteLine();
            Console.WriteLine(File.ReadAllText(@"OutputTextFile.txt"));
        }
        catch(Exception ex) {
            Console.WriteLine(ex.Message);
        }

    }
}
```