实验一

C#编码规范

**1.1.Tab**

Ö **要**使一个Tab为4个空格长。

**1.2.缩进**

Ö **要**使一个代码块内的代码都统一缩进一个Tab长度。

**1.3.空行**

Ö **建议**适当的增加空行，来增加代码的可读性。

Ö 在在类，接口以及彼此之间**要**有两行空行：

Ö 在下列情况之间**要**有一行空行：

方法之间；

局部变量和它后边的语句之间；

方法内的功能逻辑部分之间；

**1.4.函数长度**

每个函数有效代码（不包括注释和空行）长度**不要**超过100行，尽可能控制在50行。

**1.5.{”，“}”**

Ö 开括号“{”**要**放在块的所有者的下一行，单起一行；

Ö 闭括号“}”**要**单独放在代码块的最后一行，单起一行。

**1.6.行宽**

每行代码和注释**不要**超过70个字符或屏幕的宽度，如超过则应换行，换行后的代码应该缩进一个Tab。

**1.7. 空格**

´ 括号和它里面的字符之间**不要**出现空格。括号应该和它前边的关键词留有空格，如：while (true) {};

´ 但是方法名和左括号之间**不要**有空格。

Ö 参数之间的逗号后**要**加一空格。如：method1(int i1, int i2)

Ö for语句里的表达式之间**要**加一空格。如：for (expr1; expr2; expr3)

Ö 二元操作符和操作数之间**要**用空格隔开。如：i + c;

Ö 强制类型转换时，在类型和变量之间**要**加一空格。如：(int) i ;

**2.  注释**

**2.1.注释的基本约定**

Ö 注释应该增加代码的清晰度；

Ö 保持注释的简洁，不是任何代码都需要注释的，过多的注释反而会影响代码的可读性。

´ 注释**不要**包括其他的特殊字符。

Ö **建议**先写注释，后写代码，注释和代码一起完成

Ö 如果语句块（比如循环和条件分枝的代码块）代码太长，嵌套太多，则在其结束“｝”**要**加上注释，标志对应的开始语句。如果分支条件逻辑比较复杂，也**要**加上注释。

Ö 在VS2005环境中通过配置工程编译时输出XML文档文件可以检查注释的完整情况，如果注释不完整会报告编译警告；

**2.2.注释类型**

**2.2.1.块注释**

Ö 主要用来描述文件，类，方法，算法等，放在所描述对象的前边。具体格式以IDE编辑器输入“///”自动生成的格式为准，另外再附加我们自定义的格式，如下所列：

/// <Remark>作者，创建日期，修改日期</ Remark >

对类和接口的注释必须加上上述标记，对方法可以视情况考虑

**2.2.2.行注释**

Ö 主要用在方法内部，对代码，变量，流程等进行说明。整个注释占据一行。

**2.2.3.尾随注释**

Ö 与行注释功能相似，放在代码的同行，但是要与代码之间有足够的空间，便于分清。例：

int m = 4 ; /／ 注释

Ö 如果一个程序块内有多个尾随注释，每个注释的缩进**要**保持一致。

**2.3.注释哪些部分**

不是所有的成员函数都必须要注释，以下功能函数可以不加注释：

界面模块的功能按钮事件函数

override的函数

private的函数可以选择性的加入注释，如果private函数的功能十分简单，可以不加入注释，否则**建议**加入注释

Ö private的字段/属性可以不加注释

Ö 局部变量只需要用”//”说明即可

**3.  命名**

**3.1.命名的基本约定**

Ö **要**使用可以准确说明变量/字段/类的完整的英文描述符，如firstName。对一些作用显而易见的变量可以采用简单的命名，如在循环里的递增（减）变量就可以被命名为 ” i ”。

Ö **要**尽量采用项目所涉及领域的术语。

Ö **要**采用大小写混合，提高名字的可读性。为区分一个标识符中的多个单词，把标识符中的每个单词的首字母大写。不采用下划线作分隔字符的写法。有两种适合的书写方法，适应于不同类型的标识符：

PasalCasing：标识符的第一个单词的字母大写；

camelCasing：标识符的第一个单词的字母小写。

´ **避免**使用缩写，如果一定要使用，就谨慎使用。同时，应该保留一个标准缩写的列表，并且在使用时保持一致。

Ö 对常见缩略词，两个字母的缩写**要**采用统一大小写的方式（示例：ioStream，getIOStream）；多字母缩写采用首字母大写，其他字母小写的方式（示例：getHtmlTag）；

´ **避免**使用长名字（最好不超过 15 个字母）。

´ **避免**使用相似或者仅在大小写上有区别的名字

**3.2.各种标示符类型的命名约定**

**3.2.1.程序集命名**

Ö 公司域名（GTA）+ 项目名称 + 模块名称（可选），例如：

中心系统程序集：GTA.ProductionCenter；

中心系统业务逻辑程序集：GTA. ProductionCenter.Business；

**3.2.2.命名空间命名**

Ö 采用和程序集命名相同的方式：公司域名（GTA）+ 项目名称 + 模块名称。 另外，一般情况下建议命名空间和目录结构相同。例如：

中心系统：GTA.ProductionCenter；

中心系统下的用户控件：GTA.ProductionCenter.UserControl；

中心系统业务逻辑：GTA. ProductionCenter.Business；

中心系统数据访问：GTA. ProductionCenter.Data；

**3.2.3.类和接口命名**

Ö 类的名字**要**用名词；

´ 避免使用单词的缩写，除非它的缩写已经广为人知，如HTTP。

Ö 接口的名字**要**以字母I开头。保证对接口的标准实现名字只相差一个“I”前缀，例如对IComponent的标准实现为Component；

Ö 泛型类型参数的命名：命名**要**为T或者以T开头的描述性名字，例如：

public class List<T>

public class MyClass<TSession>

´ 对同一项目的不同命名空间中的类，命名**避免**重复。避免引用时的冲突和混淆；

**3.2.4.方法命名**

Ö 第一个单词一般是动词

Ö 如果方法返回一个成员变量的值，方法名一般为Get+成员变量名，如若返回的值 是bool变量，一般以Is作为前缀。另外，如果必要，考虑用属性来替代方法，具 体建议见10.1.2节;

Ö 如果方法修改一个成员变量的值，方法名一般为：Set + 成员变量名。同上，考虑 用属性来替代方法；

**3.2.5.变量命名**

Ö 按照使用范围来分，我们代码中的变量的基本上有以下几种类型，类的公有变量；类的私有变量（受保护同公有）；方法的参数变量；方法内部使用的局部变量。这些变量的命名规则基本相同，见标识符大小写对照表。区别如下：

i. 类的公有变量按通常的方式命名，无特殊要求；

ii. 类的私有变量采用两种方式均可：采用camalString，例如：workerName;采用加“m”前缀，例如mWorkerName;

iii. 方法的参数变量采用camalString，例如workerName；

iv. 方法内部的局部变量采用camalString，例如workerName；

´ **不要**用\_或&作为第一个字母；

Ö 尽量**要**使用短而且具有意义的单词；

Ö 单字符的变量名一般只用于生命期非常短暂的变量。i,j,k,m,n一般用于integer；c,d,e 一般用于characters；s用于string

Ö 如果变量是集合，则变量名**要**用复数。例如表格的行数，命名应为：RowsCount；

Ö 命名组件**要**采用匈牙利命名法，所有前缀均应遵循同一个组件名称缩写列表

**4.  声明**

Ö 每行**要**只有一个声明，如果是声明i,j,k之类的简单变量可以放在一行;

Ö 除了for循环外，声明**要**放在块的最开始部分。for循环中的变量声明可以放在for语句中。如：for(int i = 0; I < 10; i++) 。

´ **避免**块内部的变量与它外部的变量名相同。

**5.  表达式和语句**

public double fangweijiao(double[] sdr, double[] cr)

{

double sum = 0;

for (int i = 1; i < sdr.Length; i++)//从第二行开始循环计算 坐标方位角、观测角度累加值

{

cr[i] = cr[i - 1] + sdr[i] - Math.PI;//计算坐标方位角/左角

if (cr[i] >= Math.PI \* 2)//判断坐标方位角是否在0到2PI之间

cr[i] -= Math.PI \* 2;

else if (cr[i] < 0.0)

cr[i] += Math.PI \* 2;

sum += sdr[i];

}

return sum;

}

private void excel文件ToolStripMenuItem\_Click(object sender, EventArgs e)

{

dataGridView1.DataSource = null; //清除数据源

dataGridView1.Rows.Clear(); //清空数据表格的行列

dataGridView1.Columns.Clear();

OpenFileDialog file = new OpenFileDialog(); //声明 打开文件对话框 file

file.Filter = "Excel文件|\*.xls|Excel文件|\*.xlsx"; //文件过滤器，只显示Excel文件

if (file.ShowDialog() == DialogResult.OK) //如果文件正常打开

{

string fname = file.FileName; //获取打开的文件名称

string strSource = "provider=Microsoft.ACE.OLEDB.12.0;" + "Data Source=" + fname + ";Extended Properties='Excel 8.0; HDR=Yes;IMEX=1'"; //准备文件来源信息

OleDbConnection conn = new OleDbConnection(strSource); //Excel文件源放到conn中

string sqlstring = "SELECT \* FROM [Sheet1$]"; //准备选择表中的Sheet1

OleDbDataAdapter adapter = new OleDbDataAdapter(sqlstring, conn); //声明数据适配器adapter

DataSet da = new DataSet(); //声明数据集da

adapter.Fill(da); //使用adapter填充方法

dataGridView1.DataSource = da.Tables[0]; //将da.Tables[0]作为dataGridView1的数据源

}

else

return;

}

private void button1\_Click(object sender, EventArgs e)

{

string[] sd = new string[dataGridView1.RowCount - 5]; //新建一个数组存放观测角度的 原始值

double[] sdr = new double[sd.Length]; //新建一个数组存放观测角度的弧度值

double[] cr = new double[sd.Length]; //新建一个数组存放计算的坐标方位角

double sum = 0;

cr[0] = dmstorad(Convert.ToString(dataGridView1.Rows[0].Cells[4].Value)); //获取第一个坐标方位角，并将其转换成弧度，放入cr[]数组第一个元素中

double acd = dmstorad(Convert.ToString(dataGridView1.Rows[dataGridView1.RowCount - 6].Cells[4].Value)); //获取终边坐标方位角，并将其转换成弧度，放入放入acd中用于计算和检核

for (int i = 1; i < sd.Length; i++) //从第二行开始循环，将观测角度的原始值放入 sd[]数组中,并转换成弧度值存放在sdr数组中

{

sd[i] = Convert.ToString(dataGridView1.Rows[i].Cells[1].Value);

sdr[i] = dmstorad(sd[i]);

}

sum = fangweijiao(sdr, cr); //计算改正前坐标方位角和观测角度总和，分别存储在 cr数组和sum中

dataGridView1.Rows[dataGridView1.RowCount - 4].Cells[1].Value = radtodms(sum); //将观测角度总和放入表格中

double fd, fdx;

fd = cr[cr.Length - 1] - acd;//计算角度闭合差，单位弧度

fdx = 60 \* Math.Sqrt(sd.Length - 1);//计算角度闭合差限差，单位秒

dataGridView1.Rows[dataGridView1.RowCount - 3].Cells[1].Value = Convert.ToString(Math.Round(fd \* 180 / Math.PI \* 3600, 2)) + "″"; //将角度闭合差存入表格中

dataGridView1.Rows[dataGridView1.RowCount - 2].Cells[1].Value = Convert.ToString(Math.Round(fdx, 2)) + "″";//将角度闭合差限差存入表格中

if (Math.Abs(fd \* 180 / Math.PI \* 3600) > fdx)//检查角度闭合差是否满足要求

MessageBox.Show("角度闭合差超限！");

else

{

double vd = -fd / (sd.Length - 1);//分配角度闭合差（观测左角）

double sumvd = 0;

for (int i = 1; i < sdr.Length; i++)

{

sdr[i] += vd;//计算改正后的观测角度，并存入sdr数组中

sumvd += vd;

dataGridView1.Rows[i].Cells[2].Value = Convert.ToString(Math.Round(vd \* 180 / Math.PI \* 3600, 2)) + "″"; //将角度改正数存入表格中

dataGridView1.Rows[i].Cells[3].Value = radtodms(sdr[i]);

}

if (Math.Round(sumvd, 8) != Math.Round(-fd, 8)) //秒保留2位对应弧度是8位

MessageBox.Show("角度改正数分配有误！");

else

dataGridView1.Rows[dataGridView1.RowCount - 4].Cells[2].Value = Convert.ToString(Math.Round(sumvd \* 180 / Math.PI \* 3600, 2)) + "″"; //将角度改正数总和存入表格中

sum = fangweijiao(sdr, cr);//推算改正后的坐标方位角

if (Math.Round(cr[cr.Length - 1], 8) != Math.Round(acd, 8)) MessageBox.Show("坐标方位角推算有误！");

else

{

dataGridView1.Rows[dataGridView1.RowCount - 4].Cells[3].Value = radtodms(sum); //将改正后观测角度总和放入表格中

for (int i = 1; i < cr.Length - 1; i++)//将改正后坐标方位角存入表格

dataGridView1.Rows[i].Cells[4].Value = radtodms(cr[i]);

}

//坐标增量调整与计算

double[] lg = new double[cr.Length - 1];//新建一个数组存放观测长度的原始值

double[] lgx = new double[cr.Length - 1];//新建一个数组存放△x

double[] lgy = new double[cr.Length - 1];//新建一个数组存放△y

double sum1 = 0;

double sum2 = 0;

double sum3 = 0;

for (int i = 1; i < lg.Length; i++)

{

lg[1] = Convert.ToDouble(dataGridView1.Rows[1].Cells[5].Value);//将获取的距离导入数组

lg[2] = Convert.ToDouble(dataGridView1.Rows[2].Cells[5].Value);

lg[3] = Convert.ToDouble(dataGridView1.Rows[3].Cells[5].Value);

lg[4] = Convert.ToDouble(dataGridView1.Rows[4].Cells[5].Value);

lg[5] = Convert.ToDouble(dataGridView1.Rows[5].Cells[5].Value);

lgx[i] = lg[i] \* Math.Cos(cr[i]);//计算坐标增量

lgy[i] = lg[i] \* Math.Sin(cr[i]);

lgx[i] = Math.Round(lgx[i], 4);

lgy[i] = Math.Round(lgy[i], 4);

dataGridView1.Rows[i].Cells[6].Value = lgx[i];//

dataGridView1.Rows[i].Cells[7].Value = lgy[i];

sum1 = lg[1]+lg[2]+lg[3]+lg[4]+lg[5];//计算距离总和

sum2 += Math.Round(lgx[i], 4);//计算△x总和

sum3 += Math.Round(lgy[i], 4);//计算△y总和

dataGridView1.Rows[dataGridView1.RowCount - 4].Cells[5].Value = sum1;

dataGridView1.Rows[dataGridView1.RowCount - 4].Cells[6].Value = sum2;

dataGridView1.Rows[dataGridView1.RowCount - 4].Cells[7].Value = sum3;

}

double zuobiaoyA, zuobiaoyC;//新建一个数组存放x坐标值

double zbzlx, zbzly, qcbhc, xdbhc;

double[] x = new double[cr.Length - 1];

double[] y = new double[cr.Length - 1];

x[1] = Convert.ToDouble(dataGridView1.Rows[1].Cells[12].Value);//获取x坐标

zuobiaoyA = Convert.ToDouble(dataGridView1.Rows[6].Cells[12].Value);//获取y坐标

y[1] = Convert.ToDouble(dataGridView1.Rows[1].Cells[13].Value);

zuobiaoyC = Convert.ToDouble(dataGridView1.Rows[6].Cells[13].Value);

zbzlx = sum2 - (zuobiaoyA - x[1]);//计算坐标增量闭合差

zbzly = sum3 - (zuobiaoyC - y[1]);

qcbhc = Math.Sqrt(zbzlx \* zbzlx + zbzly \* zbzly);//计算导线全长闭合差

xdbhc = sum1 / qcbhc;

dataGridView1.Rows[9].Cells[7].Value = Math.Round(zbzlx, 4);//坐标增量放入表格

dataGridView1.Rows[10].Cells[7].Value = Math.Round(zbzly, 4);

dataGridView1.Rows[9].Cells[10].Value = Math.Round(qcbhc, 4);

dataGridView1.Rows[10].Cells[11].Value = Math.Round(xdbhc, 4);

double[] gzsx = new double[cr.Length - 1];//定义数组存放改正数

double[] gzsy = new double[cr.Length - 1];

double sum4 = 0;

double sum5 = 0;

double[] gzhx = new double[cr.Length - 1];//定义改正后数组

double[] gzhy = new double[cr.Length - 1];

double sum6 = 0;

double sum7 = 0;

for (int k = 1; k < lg.Length; k++)

{

gzsx[k] = -zbzlx \* lg[k] / sum1;//计算改正数

gzsy[k] = -zbzly \* lg[k] / sum1;

sum4 += gzsx[k];

sum5 += gzsy[k];

dataGridView1.Rows[k].Cells[8].Value = Math.Round(gzsx[k], 4);

dataGridView1.Rows[k].Cells[9].Value = Math.Round(gzsy[k], 4);

dataGridView1.Rows[dataGridView1.RowCount - 4].Cells[8].Value = Math.Round(sum4, 4);

dataGridView1.Rows[dataGridView1.RowCount - 4].Cells[9].Value = Math.Round(sum5, 4);

gzhx[k] = lgx[k] + gzsx[k];//计算改正后坐标增量

gzhy[k] = lgy[k] + gzsy[k];

sum6 += gzhx[k];

sum7 += gzhy[k];

dataGridView1.Rows[k].Cells[10].Value = Math.Round(gzhx[k], 4);

dataGridView1.Rows[k].Cells[11].Value = Math.Round(gzhy[k], 4);

dataGridView1.Rows[dataGridView1.RowCount - 4].Cells[10].Value = Math.Round(sum6, 4);

dataGridView1.Rows[dataGridView1.RowCount - 4].Cells[11].Value = Math.Round(sum7, 4);

}

for (int j = 2; j < lg.Length; j++)

{

x[j] = x[j - 1] + gzhx[j];

y[j] = y[j - 1] + gzhy[j];

dataGridView1.Rows[j].Cells[12].Value = Math.Round(x[j], 4);

dataGridView1.Rows[j].Cells[13].Value = Math.Round(y[j], 4);

}

}

}

private void button2\_Click(object sender, EventArgs e)

{

Application.Exit();

}

}

}

**6.  类型设计规范**

Ö **要**确保每个类型由一组定义明确，相互关联的成员组成，而不仅仅是一些无关功能的随 机集合；

Ö **不要**随意使用public去定义一个类的范围

**7.属性和方法**

Ö 基本原则是方法表示操作，属性表示数据。如果其他各方面都一样，优先使用属性而不 是方法。

Ö **要**使用属性，如果该成员表示类型的逻辑attribue

Ö 如果属性的值存储在内存中，而提供属性的目的仅仅是为了访问该值，**要**使用属性而不 要使用方法

Ö 如果该操作每次返回的结果不同，那么**要**使用方法。