C#WPF 测绘程序设计简明教程

小 y 同学

2024 年 5 月

Contents

1 致读者

1.1 简介

此教程主要是对标测绘学科创新创智能大赛——测绘程序设计组而编写，采用 C# WPF 技术结合 MVVM(Model-View-ViewModel) 设计模式进行.NET 桌面应用程序设计，适合具有一些 C# 或者 C/C++ 等编程经验的读者阅读，要求读者需要具有一些面向程序设计（类和对象）的概念。

全篇首先介绍了 WPF 界面常用控件，其目的是让读者快速了解 WPF 的界面设计模式；然后介绍了 C# 的基本操作，旨在让读者深入了解 C# 程序设计；随后结合笔者了解，简单介绍了测绘程序设计流程框架，让读者从无到有全面感受一个桌面应用程序的搭建；最后结合 C# 语言以及测绘数据的一些特点，给出一些编程注意事项，减少出错的机会。需要注意的是：由于本篇为简明教程，C# 基本语法与 C/C++ 等主流编程语言类似，XAML 基本语法与 HTML 类似，故对 C#、XAML 简单的语法基础并未做描述。

由于笔者经验水平有限，难免出现一些错误和不足，还望读者批评指正，给出建议。

1.2 为什么选择 **WPF**？

为什么选择 WPF？这需要从测绘程序设计竞赛的角度出发：竞赛明文要求“编程语言限制为 Basic、C/C++、C#，不允许使用二次开发平台（如 Matlab、Python）”，就笔者所了解的而言，可以做界面的组合有：Basic、C++ 的 MFC 框架、C# 的 WinForm 以及 C# 的 WPF。

Basic 语言是一门比较古老的编程语言，目前学习 Basic 的人少之又少，更别提测绘专业的学生了；C/C++ 基本是工科大学生的入门编程语言，其中 C++ 的 MFC 框架可以做界面设计，但就笔者从《计算机图形学》课程中接触的 MFC 学习感受来说，难度很高。

C# 作为微软公司发布的编程语言，其具有 WinForm 和 WPF 两种界面编程方式，笔者初次接触 WinForm 是做 PIE 二次开发，初次接触 WPF 是《测量程序设计》课程，直观的感受来说 WinForm 确实比 WPF 简单，最终选择 WPF 的原因如下：

1. WPF 使用 XAML 描述界面，布局的难度只是相对于初学者而言较困难，熟练掌握仍能在较短的事件内搭建出简洁美观的界面。
2. WPF 结合 MVVM 设计模式，通过数据绑定的方式驱动界面，省去了从界面取值和将结果展示到界面的环节，某种程度上减轻了 C# 代码编写的负担。
3. WPF+MVVM 是时代的潮流，可以做到前后端分离，程序架构更加清晰；XAML 描述界面在 Web、微信小程序开发等场景有着相似的方式；作为当代大学生更应不畏困难、了解 WPF。

2 WPF 界面常用控件

2.1 公共控件

2.1.1 Button（按钮）

> 描述

- 最基本的控件之一，可以结合 StackPanel 控件或者 WrapPanel 控件创建带图片的按钮。

> 重要属性

- Click（点击触发事件）：针对事件驱动的点击响应事件。
- **Command**（命令）：针对 **MVVM** 模式的绑定命令。

> 其他补充

- 结合 **StackPanel** 控件创建带图标的按钮
  在 Button 的子元素中放置 StackPanel 控件，在 StackPanel 控件中添加 Image 控件（存放图片）和 TextBlock 控件（存放按钮文字描述）即可实现带图标的按钮。

```
<Button>
    <StackPanel Orientation="Horizontal">
        <Image Source="../Resources/OpenFile2.bmp" Height="15"
           Width="15"/>
        <TextBlock Text="打开"/>
    </StackPanel>
</Button>
```

  其中，`Orientation`(朝向)属性可以设置图片在按钮左边 (Horizontal) 还是按钮上方 (Vertical)。

- 结合 WrapPanel 控件创建带图标的按钮
  与 StackPanel 类似，此处不多赘述。

```
<Button Command="{Binding CalCommand}">
    <WrapPanel>
        <Image Source="../Resources/Cal.bmp" Height="15" Width="15
           "/>
        <TextBlock Text="计算"/>
    </WrapPanel>
</Button>
```

- **Command** 属性绑定
  View-Model 类可以通过 Command 与 Button 进行绑定，从而实现交互。
  以绑定一个关闭窗口的 Command 为例，首先需要有一个继承自 ICommand 的 MyCommands 类(见 MyCommands 类)，在 View-Model 类中实现相应的 Command 属性如下：

```
public void Close()
{
    Application.Current.MainWindow.Close();
}
// 实现Command属性
public ICommand CloseCommand =>
    new MyCommands((object p) => Close(), (object p) => true);
```

到 XAML 界面进行 Command 绑定：

```
<ToolBar DockPanel.Dock="Top" Height="30">
    <Button Content="退出" Command="{Binding CloseCommand}" />
</ToolBar>
```

- Button 去边框去底色变成透明

```
<Button Background="Transparent" BorderBrush="Transparent" Margin=
    "100" Content="一个按钮"></Button>
```

2.1.2 Label（标签）和 TextBlock（文本块）

> 描述

Label 和 TextBlock 都可以用来显示文本，相对而言，Label 使用频率会搞一些。

> 重要属性

- Content(Label)：用于绑定 Label 的文字
- Text(TextBlock)：用于绑定 TextBlock 的文字
- Height
- Margin
- FontSize

> 其他补充

- **Content** 属性绑定
  View-Model 类可以将属性变量绑定到对应的控件，从而实现属性变量发生改变时，控件的属性也会随之改变。
  以 Label 作为状态栏文字显示控件绑定到 View-Model 类为例，首先需要一个继承自 INotifyPropertyChanged 的 NotifyPropertyObject 类(见 NotifyPropertyObject 类)，在 View-Model 类中实现相应的属性如下：

```
private string ltext = "准备就绪";

public string Ltext
{
    get { return ltext; }
    set { ltext = value; RaisePropertyChanged(); }
}
```

将属性绑定至界面如下

```
<StatusBar DockPanel.Dock="Bottom" Height="30">
    <StatusBarItem>
        <Label Content="{Binding Ltext}" Margin="10,0,0,0"/>
    </StatusBarItem>
</StatusBar>
```

2.1.3 TextBox（文本框）

> 描述

TextBox 用来实现用户文本输入，以及文本展示，可以用来做报告显示。

> 重要属性

- Text：显示的文本
- **TextWrapping**：默认为NoWrap无法自动换行，设置成Wrap可以按照字符自动换行（会拆散单词），设置成WrapWithOverflow则是按照空格自动换行（不会拆散单词）。
- **AcceptsReturn**：默认为False不接受用户输入回车换行符，设置成True接受输入回车换行符。

> 其他补充

- 利用 TextBox+Group 设置报告界面

```
<GroupBox Header="报告预览" Grid.Column="2">
    <TextBox Text="{Binding Report}" AcceptsReturn="True"
       TextWrapping="Wrap"/>
</GroupBox>
```

2.1.4 DataGrid（数据表格）

> 描述

DataGrid 用以表格形式展示属于，可以用来做输入数据显示。

> 重要属性

- ItemSource：用来绑定 ObservableCollecion 数据类型的变量
- CanUserAddRows：设置成`False`可以去除最后一行。
- AutoGenerateColumns：默认为`True`根据绑定的 ItemSource 自动生成列，设置成`False`可以添加子元素 DataGrid.Columns 自定义列。

> 其他补充

- View-Model 类设计属性同时绑定到 DataGrid 界面。
  在 View-Model 类中设计 OvservableCollection 属性：

```
private ObservableCollection<MyPoints> data = new
    ObservableCollection<MyPoints>();

public ObservableCollection<MyPoints> Data
{
    get { return data; }
    set { data = value; RaisePropertyChanged(); }
}
```

  其中，MyPoints 需要自己定义一个类：

```
public class MyPoints
{
    public string Name { get; set; }
    public double X { get; set; }
    public double Y { get; set; }
    public double Z { get; set; }

    public MyPoints() { }
    public MyPoints(string name, double x, double y, double z)
    {
        this.Name = name;
        this.X = x;
        this.Y = y;
        this.Z = z;
    }
}
```

  在 XAML 界面中绑定数据到 DataGrid，将子元素 **DataGridTextColumn** 的 **Width** 设置为 **1*** 可以去除最后的空白列：

```
<GroupBox Header="原始数据" Grid.Column="0">
    <DataGrid ItemsSource="{Binding Data}" Margin="2"
        AutoGenerateColumns="False" CanUserAddRows="False">
        <DataGrid.Columns>
```

```
4               <DataGridTextColumn Header="点名" Binding="{Binding
                    Name}" Width="1*"/>
5               <DataGridTextColumn Header="X(m)" Binding="{Binding X,
                    StringFormat={}{0:F3}}" Width="1*"/>
6               <DataGridTextColumn Header="Y(m)" Binding="{Binding Y,
                    StringFormat={}{0:F3}}" Width="1*"/>
7               <DataGridTextColumn Header="Z(m)" Binding="{Binding Z,
                    StringFormat={}{0:F3}}" Width="1*"/>
8           </DataGrid.Columns>
9       </DataGrid>
10  </GroupBox>
```

2.1.5 ComboBox（下拉框）

> 描述

下拉框，用户可以通过下拉框来调整预选的参数。

> 重要属性

- ItemsSource：下拉选择的项，可以绑定椭球列表，需要在椭球类中实现 ToString 函数的重写以展示正常。
- SelectedItem：选择的项，一般用作绑定当前椭球

> 其他补充

- ComboBox 绑定椭球
  椭球类的编写(见椭球类的简单实现)

  View-Model 类相关属性的设计

```
1   //椭球列表
2   private List<MyEllipsoid> ellipsoidList = EllipsoidFactory.
        EllipsoidList;
3
4   public List<MyEllipsoid> EllipsoidList
5   {
6       get { return ellipsoidList = EllipsoidFactory.EllipsoidList; }
7   }
8
9   //当前椭球
10  private MyEllipsoid currentEllipsoid = EllipsoidFactory.
        EllipsoidList[0];
11
12  public MyEllipsoid CurrentEllipsoid
13  {
14      get { return currentEllipsoid; }
15      set
```

```
    {
        currentEllipsoid = value;
        RaiseProperytChanged();
        RaiseProperytChanged(nameof(Rtext));
    }
}
```

XAML 绑定属性

```
<ComboBox ItemsSource="{Binding EllipsoidList}" SelectedItem="{
    Binding CurrentEllipsoid}"/>
```

2.2 菜单、工具栏和状态栏

2.2.1 Menu（菜单）

> 描述

配合 DockPanel 和 MenuItem 来制作菜单栏。

> 重要属性

- Header：设置标题
- DockPanel.Dock：设置锚定位置（需要是 DockPanel 的子元素）

> 其他补充

- 设计一组简单的菜单栏

```
<Menu DockPanel.Dock="Top">
    <MenuItem Header="文件(F)">
        <MenuItem Header="打开" Command="{Binding OpenFileCommand}
            "
            <MenuItem.Icon>
                <Image Source="../Resources/OpenFile2.bmp"/>
            </MenuItem.Icon>
        </MenuItem>
        <MenuItem Header="保存" Command="{Binding SaveFileCommand}
            "
            <MenuItem.Icon>
                <Image Source="../Resources/Save.bmp"/>
            </MenuItem.Icon>
        </MenuItem>
    </MenuItem>
    <MenuItem Header="计算(G)">
        <MenuItem Header="一键计算"/>
        <Separator/>
```

```
17          <MenuItem Header="计算步骤1"/>
18          <MenuItem Header="计算步骤2"/>
19          <MenuItem Header="计算步骤..."/>
20      </MenuItem>
21      <MenuItem Header="关于(H)">
22          <MenuItem Header="帮助">
23              <MenuItem.Icon>
24                  <Image Source="../Resources/Help.bmp"/>
25              </MenuItem.Icon>
26          </MenuItem>
27          <MenuItem Header="退出">
28              <MenuItem.Icon>
29                  <Image Source="../Resources/Close.bmp"/>
30              </MenuItem.Icon>
31          </MenuItem>
32      </MenuItem>
33  </Menu>
```

2.2.2 ToolBar（工具栏）

> 描述

工具栏，可以用于存放 Button、ComboBox 等小控件。

> 其他补充

- 工具栏设置带图标的 Button(见 Button)

2.2.3 StatusBar（状态栏）

> 描述

状态栏，用于展示程序的运行状态信息、一些参数信息。

> 重要属性

- DockPanel.Dock
- Height
- 子元素 StatusBarItem 的 HorizontalAlignment 属性

> 其他补充

- 状态栏设置左右文字

> 核心思想：设置 StatusBarItem 的 HorizontalAlignment 属性分别为 Left 和 Right。

```
<StatusBar DockPanel.Dock="Bottom" Height="30">
    <StatusBarItem HorizontalAlignment="Left" Margin="5,0,0,0">
        <WrapPanel>
            <Label Content="当前状态:"/>
            <Label Content="准备就绪"/>
        </WrapPanel>
    </StatusBarItem>
    <StatusBarItem HorizontalAlignment="Right" Margin="0,0,5,0">
        <WrapPanel>
            <Label Content="当前椭球: "/>
            <Label Content="CSCG2000"/>
        </WrapPanel>
    </StatusBarItem>
</StatusBar>
```

2.2.4 使用 **DockPanel** 组装一个简单界面

一个简单界面的完整性从上到下依次包括菜单栏、工具栏、功能展示区域、状态栏，此处给出一个 DockPanel 简单组装的界面。

```
<DockPanel LastChildFill="True">
    <Menu DockPanel.Dock="Top">
        <MenuItem Header="文件(F)">
            <MenuItem Header="打开数据"/>
            <MenuItem Header="保存报告"/>
            <Separator/>
            <MenuItem Header="关于"/>
            <MenuItem Header="退出"/>
        </MenuItem>
        <MenuItem Header="计算(C)">
            <MenuItem Header="一键计算"/>
            <Separator/>
            <MenuItem Header="计算步骤1"/>
            <MenuItem Header="计算步骤2"/>
            <MenuItem Header="计算步骤3"/>
        </MenuItem>
    </Menu>

    <ToolBar DockPanel.Dock="Top" Height="30">
        <Button Content="打开"/>
        <Button Content="计算"/>
        <Button Content="保存"/>
        <Separator/>
        <Button Content="退出"/>
    </ToolBar>

```

```
27      <StatusBar DockPanel.Dock="Bottom" Height="30">
28          <StatusBarItem HorizontalAlignment="Left">
29              <WrapPanel>
30                  <Label Content="状态:"/>
31                  <Label Content="准备就绪"/>
32              </WrapPanel>
33          </StatusBarItem>
34      </StatusBar>
35
36      <Border BorderBrush="Red" BorderThickness="2">
37          <Grid Background="AntiqueWhite">
38              <TextBox Margin="5"/>
39          </Grid>
40      </Border>
41  </DockPanel>
```

2.3 容器

2.3.1 Border（边框）

> 描述

用于给某个元素绘制边框或者提供背景，是最简单的美化方式之一。

> 重要属性

- BorderBrush：边框颜色
- BorderThickness：边框的宽度
- Margin：外边距
- Padding：内边距
- CornerRadius：圆角

> 其他补充

2.3.2 Grid（网格）

> 描述

Grid 的存在是 WPF 与 WinForm 布局的区别之一：通过 Grid 控件将整个窗体进行网格划分，再将对应的控件放置在对应的网格中，整个界面设计不仅更具条理，最重要的是界面大小发生变化时仍能保持合理的布局。

> 重要属性

- Grid.Row、Grid.Column：指定子元素所在的行列，索引从 0 开始。
- GridRowSpan、Grid.ColumnSapn：子元素跨多行多列。
- BackGround

> 其他补充

2.3.3 GroupBox（组合框）

> 描述

自带边框和文字的一个容器，可用与数据展示控件例如 DataGrid 和 TextBox 组合。

> 重要属性

- Header：文字描述
- BorderBrush：边框颜色
- BorderThickness：边框宽度

2.3.4 TabControl（选项卡）

> 描述

用于实现两页、三页的页面效果，配合 TabItem 进行分页设计。

> 重要属性

- TabStripPlacement：选择框的位置，一般设置 Bottom，放在下方。

> 其他补充

- TabControl 实现报告显示、图形显示

```
1  <TabControl Grid.Column="2" TabStripPlacement="Bottom" Background=
       "AntiqueWhite">
2      <TabItem Header="报 告 预 览">
3          <GroupBox Header="报 告 预 览" Grid.Column="2">
4              <TextBox Text="{Binding Report}" AcceptsReturn="True"
                   TextWrapping="Wrap"/>
5          </GroupBox>
6      </TabItem>
7      <TabItem Header="图 表 显 示">
```

```
8           <GroupBox Header="图表显示" Grid.Column="2">
9               <Canvas Background="White"></Canvas>
10          </GroupBox>
11      </TabItem>
12  </TabControl>
```

效果图如下：

2.3.5 WrapPanel（环绕面板）和 StackPanel（堆栈面板）

> 描述

WrapPanel 环绕面板，其子元素可以根据方向属性水平或垂直排列，当长度或高度不够时会自动调整进行换行。

StackPanel 堆栈面板，其子元素可以根据方向属性水平或垂直排列在一行中。

> 重要属性

- Orientation：方向属性，StackPanel 中默认是 Vertical(垂直排列)，设置成 Horizontal 可以实现水平排列。而在 WrapPenel 中默认为水平排列。

> 其他补充

- 使用 StackPanel 排列带图标的工具栏按钮(见 Button（按钮）)

2.3.6 DockPanel（停靠面板）

> 描述

定义一个区域用来对子元素设置 Left、Top、Right、Bottom 锚定。

> 重要属性

- LastChildFill：最后一个元素是否填满整个 Dock，默认为 False，需要设置成 True（最后一个元素默认填满整个 Dock）。
- DockPanel.Dock：为子元素设置停靠的方向 (Left、Top、Right、Bottom)。

> 其他补充

- 使用 DockPanel 组装一个简单界面使用 DockPanel 组装一个简单界面

2.4 其他

2.4.1 Separator（分隔符）

> 描述

菜单栏、工具栏中的分隔符，用于简单美化。

2.4.2 GridSplitter

> 描述

Grid 中的分隔符，用户可以通过拖动该分隔符调整 Grid 区域大小。

> 重要属性

- HorizontalAligment：竖向的分隔符需设置该属性为 Stretch（拉伸）
- Background

2.4.3 OpenFileDialog 和 SaveFileDialog

> 描述

OpenFileDialog 和 SaveFileDialog 用来弹出选择文件路径对话框，是桌面程序设计中与用户交互的重要组成部分。

> 用法

- OpenFileDialog

  > 核心：从左上角到右下加思考对话框的内容：Title（标题）、Multiselect（多选）、Filter（过滤条件）；同时考虑用户不选择文件直接关闭对话框的情况。

```
public void OpenFile()
{
    OpenFileDialog ofd = new OpenFileDialog();
    ofd.Title = "选择数据文件"; //设置对话框标题
    ofd.Multiselect = false; //默认可以多选，设置不可多选
    ofd.Filter = "文本文件|*.txt|所有文件|*.*";//设置过滤条件
    if (ofd.ShowDialog() != true) { return; } //如果用户没有选择文件而关闭了对话框，则return

    using (StreamReader sr = new StreamReader(ofd.FileName))
    {
        MessageBox.Show(sr.ReadToEnd());
    }
}

public ICommand OpenFileCommand => new MyCommand(
    (object p) => OpenFile(), (object p) => true);
```

- SaveFileDialog

  > 核心：与 OpenFileDialog 仅仅差一个 AddExtension(自动添加拓展名)。

```
public void SaveFile()
{
    SaveFileDialog sfd = new SaveFileDialog();
    sfd.Title = "请选择文件保存路径";
    sfd.AddExtension = true;//自动添加文件拓展名
    sfd.Filter = "文本文件|*.txt";
    if (sfd.ShowDialog() != true) { return; }

    using (StreamWriter sw = new StreamWriter(sfd.FileName))
    {
        sw.WriteLine("保存的文本");
    }
}

public ICommand SaveFileCommand => new MyCommand(
    (object p) => SaveFile(), (object p) => true);
```

2.4.4 MessageBox

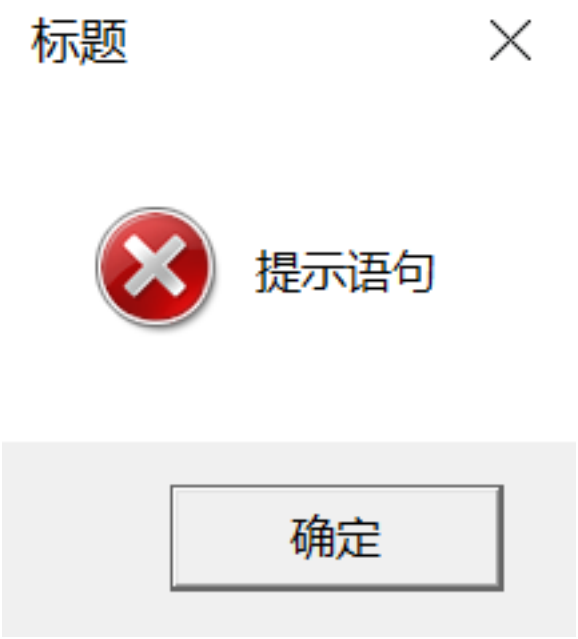
> 描述

MessageBox 用来弹出消息提示框，同时也可以用来做调试输出。在桌面程序设计中主要是给出一些必要的弹窗提示，例如错误提示和关键步骤正确提示。

> 用法

```
private void Button_Click(object sender, RoutedEventArgs e)
{
    MessageBox.Show("提示语句", "标题", MessageBoxButton.OK,
        MessageBoxImage.Error);
}
```

效果图片：



3 C# 基本操作

此章节主要是介绍 C# 的基本操作，包括读写文本文件、数据类型的相互转换、字符串处理等等，同时给出了一些测绘常见的函数代码：

3.1 读写文本文件

3.1.1 使用 StreamReader 类读取文本文件

> 核心语句

- `StreamReader sr = new StreamReader(FilePath);`
- `string AllText = sr.ReadToEnd();`
- `string Line = sr.ReadLine();`

```
1 string FilePath = @"D:\ADesktop\Southern Survey and Mapping\2024\
    MyWpfApp\LuoGu\测试数据.txt";
2 using (StreamReader sr = new StreamReader(FilePath))
3 {
4     string AllText = sr.ReadToEnd();//一次读完，同时文件指针指向文末，
          即再次用该对象读取时为null
5     Console.Write(AllText);
6 }
```

```
using (StreamReader sr = new StreamReader(FilePath))
{
    string Line = sr.ReadLine(); //一次读取一行，同时文件指针指向行末
    while (Line != null)
    {
        Console.Write(Line);
        Line = sr.ReadLine();
    }
}
```

3.1.2 使用 StreamWriter 类写入文本文件

> 核心语句

- sw.WriteLine(string.Join(",", myData[i].ConvertAll(t => t.ToString("f4"))));

```
string SavePath = @"D:\ADesktop\Southern Survey and Mapping\2024\MyWpfApp\LuoGu\SaveFile.txt";
using (StreamWriter sw = new StreamWriter(SavePath))
{
    // 为了模拟真实表格情况，用循环生成数据写入文件
    // 生成数据
    List<List<double>> myData = new List<List<double>>();
    for (int i = 0; i <= 360; i += 1)
    {
        double i_rad = i*1.0 / 180.0 * Math.PI;
        List<double> tempList = new List<double>();
        tempList.Add(i);
        tempList.Add(Math.Sin(i_rad));
        tempList.Add(Math.Cos(i_rad));
        tempList.Add(Math.Tan(i_rad));
        myData.Add(tempList);
    }
    // 写入文件
    sw.WriteLine(string.Join(",", new string[] { "角度（°）", "正弦", "余弦", "正切" }));
    for (int i = 0; i < myData.Count; i++)
    {
        sw.WriteLine(string.Join(",", myData[i].ConvertAll(t => t.ToString("f4"))));
    }
}
```

3.2 数据类型的相互转换

string 类型要转为 int 或者 double 类型才可以做数值运算。笔者推荐能用 double 就不要用 int，因为在 C# 中 int 在与 int 运算的结果仍为 int，会影响计算精度。

> 字符串转数字

1. 使用 Parse 方法

```
string myString = "1314.0";
double myDouble = double.Parse(myString);
int myInt = int.Parse(myString);
long myLongInt = Int64.Parse(myString);//长整型
```

2. 使用 Convert 类

```
string myString = "1314.0";
double myDouble = Convert.ToDouble(myString);
int myInt = Convert.ToInt32(myString);
long myLongInt = Convert.ToInt64(myString);//长整型
```

> 数字转字符串

1. 使用 $ 符号

```
int myInt = 1314;
string myString = $"{myInt}";
```

2. 使用 ToString 方法

```
int myInt = 1314;
string myString = myInt.ToString();
```

3. 使用 Convert 类

```
int myInt = 1314;
string myString = Convert.ToString(myInt);
```

3.3 字符串的处理

3.3.1 字符串分割

字符串分割是指对一个字符串例如

```
string s = "Python,Matlab,C/C++,C#,R";
string[] fruits = s.Split(','); // 按照逗号分割字符串
foreach (string fruit in fruits)
```

```
4  {
5      Console.WriteLine(fruit);
6  }
```

3.3.2 字符串取子串

> 核心语句

- `string s1 = s.Substring(0,6); //从位置0开始向后取6位，包括位置0`
- `string s2=s.Substring(6);//从索引6(绘)位置开始到结束，包括索引6`

```
1  string s = "测神州经纬，绘祖国蓝图!";
2  string s1 = s.Substring(0,6); //从位置0开始向后取6位，包括位置0
3  string s2=s.Substring(6);//从索引6(绘)位置开始到结束，包括索引6
4  Console.WriteLine(s1);
5  Console.WriteLine(s2);
6  Console.WriteLine(string.Join("", new string[] { s1, s2 }));
```

3.3.3 字符串拼接

> 使用加法运算符

```
1  string a = "Hello";
2  string b = "C#";
3  string c = "I'm little y";
4  string result = a + " " + b + "\n" + c;
5  Console.WriteLine(result);
```

> 使用 $ 符号

```
1  string a = "Hello";
2  string b = "C#";
3  string c = "I'm little y";
4  string result = $"{a} {b}\n{c}";
5  Console.WriteLine(result);
```

3.3.4 格式化字符串

格式化字符串是指对字符串保留一定的格式，例如保留两位小数、保留两位有效数字等等，此处仅说明常用的保留两位小数的方法。https://www.cnblogs.com/perfect-long/p/12799309.html。

> 使用 string.Format() 函数

```
//使用零占位符0和数字占位符#
Console.WriteLine(string.Format("{0:00.###}", a));//03.14
Console.WriteLine(string.Format("{0:00.000}", a));//03.140
```

> 使用 $ 符号

```
Console.WriteLine($"{a:00.###}");//03.14
Console.WriteLine($"{a:00.000}");//03.140
```

3.4 数组

3.4.1 固定数组 Array

需要注意的是：此处的 Array 与 ArrayList 不同，Array 是固定数组，ArrayList 是非泛型的动态数组。

> 定义和初始化

```
int[] num1 = new int[10];
int[] num2 = new int[] { 1, 2, 3 };
```

> 常用方法

```
int len = num2.Length;
double sum = num2.Sum();//数组求和
double average = num2.Average();//数组平均值
double numMax = num2.Max();//最大值
double numMin = num2.Min();//最小值
bool isContain=num2.Contains(2.0);//是否包含某个值
```

> 数组排序

- Array.Sort()：用于数组升序排序。
- Array.Reverse()：用于数组颠倒，例如 [1,2,4,3] 变成 [3,4,2,1]。结合 Array.Sort() 可以对数组降序排序。

```
double[] num1 = new double[] { 1, 2, 3, 4, 5, 10, 9, 3, 2 };
Array.Sort(num1);
foreach (double i in num1)
{
    Console.WriteLine($"{i}");
}
```

```
Array.Reverse(num1);

Console.Write(string.Join(" ", num1.ToList().ConvertAll(s => s.ToString
    ())));//一行打印数组元素
```

3.4.2 泛型数组 List

> 定义和初始化

```
//先定义，后赋值
List<double> num1 = new List<double>();
num1.Add(1); num1.Add(2);
num1.AddRange(new double[] { 1, 3, 1, 4 }); //添加一组Array
num1.AddRange(num1); //添加自己

List<double> num2 = new List<double> { 1, 2, 3, 2, 3, 5 };
```

> 常用方法

```
List<double> myList = new List<double> { 1, 2, 3, 5, 7, 12, 3, 21 };
myList.Add(1314);//在末尾添加一个元素
myList.AddRange(new double[] { 1, 2, 3, 4, 5 });//在末尾添加多个元素
bool IsExist = myList.Contains(1314); //一个元素是否在数组内
myList.Insert(0, 12);//在0号索引插入元素12
myList.Remove(1314);//移除匹配的第一个元素
myList.Sort();//升序排序
myList.RemoveAt(myList.Count - 1);//移除最后一个索引元素

myList.Clear(); //移除所有元素
```

> 泛型数组 List 与固定数组 Array 互转

```
//List->Array
List<double> myList1 = new List<double> { 1.0, 2.0, 3.0, 1.0 };
double[] myArray1 = myList.ToArray();

//Array->List
double[] myArray2 = new double[] { 1.0, 2.0, 3.0, 5.0 };
List<double> myList2 = myArray.ToList();
```

> List<int> 与 List<string> 互转

通过 ConvertAll 方法结合 Lambda 表达式，可以批量转换 List 中元素的数据类型，避免使用 for 循环。

```
//List<int> -> List<string>
List<int> myListInt1 = new List<int> { 1, 2, 3, 4, 2, 3, 5 };
List<string> myListString1 = myListInt1.ConvertAll(t => t.ToString());

//List<string> -> List<int>
List<string> myListSrting2 = new List<string> { "1", "2", "3" };
List<int> myListInt2 = myListSrting2.ConvertAll(t => int.Parse(t));
```

> List 转字符串

通过 string.Join() 方法，可以将 List 数组元素连接成字符串。如此可以避免循环，在生成文字报告时很有用。

1. List<string> 转字符串

```
List<string> myList = new List<string> { "一生一世", "我爱你", "!" };
string myListToString = string.Join("", myList);
Console.WriteLine(myListToString);
```

2. List<int> 转字符串

   C# 中，List<int> 可以直接使用 string.Join() 方法转为字符串，不需要转为 List<string>，这一点与 Python 略有区别。

```
List<int> myListInt = new List<int> { 1, 3, 1, 4, 5, 2, 0 };
string myListIntToString = string.Join("", myListInt);
Console.WriteLine(myListIntToString);
```

3. 将一个二维数据表转为 CSV 格式的文本

```
List<List<double>> myData = new List<List<double>>();
//制造数据
for (int i = 0; i < 10; i++)
{
    myData.Add(new List<double> { i, Math.Sqrt(i), Math.Pow(i, 2) });
}
//转为CSV
string myCSV = string.Join("\n", myData.ConvertAll(t => string.Join(",", t)));
Console.WriteLine(myCSV);
```

很容易联想到：如果是 Array 数组，可以先使用 ToList() 方法转为 List，再使用 string.Join() 方法转为字符串即可，读者可以自行尝试。

3.5 一些测绘代码

3.5.1 角度转化

> 从浮点数 dd.mmsss 中无损精度提取度、分、秒

无损精度提取的核心是将 dd.mmsss 乘 10000 后用其整数部分进行提取。

```
/// <summary>
/// 无损精度的从浮点数dd.mmsss->(d,m,s)
/// </summary>
/// <param name="ddmmss"></param>
/// <returns></returns>
public static (int d, int m, double s) DDMMSSToDMS(double dmsAngle)
{
    //13.14521
    dmsAngle = dmsAngle * 10000.0;//131452.1
    int myAngle = (int)dmsAngle;//131452
    int d = myAngle / 10000; //int(13.1452)
    int m = (myAngle - d * 10000) / 100; //int(14.52)
    double s = dmsAngle - d * 10000 - m * 100;
    return (d, m, s);
}
```

> 度、分、秒与度的互化

1. 度、分、秒转为度

```
/// <summary>
/// 度、分、秒->度
/// </summary>
/// <param name="d"></param>
/// <param name="m"></param>
/// <param name="s"></param>
/// <returns></returns>
public static double DMSTODEG(double d, double m, double s)
{
    return (d + m / 60.0 + m / 60.0 + s / 3600.0);
}
```

2. 度转为度、分、秒

```
/// <summary>
/// 度->度、分、秒
/// </summary>
/// <param name="deg"></param>
/// <returns></returns>
public static (int d, int m, double s) DEGTODMS(double deg)
{
    deg = deg * 3600;//转为秒
```

```
9       int angle = (int)deg;
10      int d = angle / 3600;//提取度
11      angle -= d * 3600;
12      int m = angle / 60;//提取分
13      double s = deg - d * 3600.0 - m * 60.0;//提取秒
14      return (d, m, s);
15  }
```

> 度与弧度的互化

1. 度转化为弧度

```
1  public static double DEGToRAD(double DEG)
2  {
3      return DEG / 180 * Math.PI;
4  }
```

2. 弧度转化为度

```
1  public static double RADToDEG(double RAD)
2  {
3      return RAD / Math.PI * 180;
4  }
```

3.5.2 椭球类的简单实现

```
1  public class MyEllipsoid
2  {
3      public string name;
4      public string id;
5      public double a;     //长半轴
6      public double b;     //短半轴
7      public double e_2;
8      public double e1_2;
9
10     private MyEllipsoid(string name, string id, double a, double b,
           double e_2, double e1_2)
11     {
12         this.name = name;
13         this.id = id;
14         this.a = a;
15         this.b = b;
16         this.e_2 = e_2;
17         this.e1_2 = e1_2;
18     }
19     public override string ToString()
20     {
21         return this.name;
```

```
22      }
23
24      public static MyEllipsoid CreateK()
25      {
26          return new MyEllipsoid("克拉索夫斯基椭球", "K", 6378245,
                6378245.0, 0.00669342162297, 0.00673852541468);
27      }
28
29      public static MyEllipsoid CreateIUGG1975()
30      {
31          return new MyEllipsoid("IUGG1975椭球", "IUGG1975", 6378140,
                6378140.0, 0.00669438499959, 0.00673950181947);
32      }
33
34      public static MyEllipsoid CreateCGCS2000()
35      {
36          return new MyEllipsoid("CGCS2000椭球", "CGCS2000", 6378137,
                6378137.0, 0.00669438002290, 0.00673949677548);
37      }
38  }
```

4 测绘程序设计流程框架

4.1 新建项目

- WPF 应用程序
- 框架.NET Framework 4.7.2（可以默认选）

4.2 整理创建必要的文件夹和文件

- Resources 文件夹，存放后期设计图标文件。
- Commands 文件夹，存放 MyCommand.cs 类文件。
- ViewModels 文件夹，存放 ViewModel 类、NotifyPropertyObject 类，创建 MainWindowVM 类，改 public，添加空的构造函数。
- Views 文件夹，存放 xml 文件，更改 App.config 中的路径。
- Models 文件夹，存放必要的算法类、数据结构类。
- 经上述操作后，生成一下项目，确保此时可以正常运行。

4.3 编写 MyCommands 类和 NotifyPropertyObject 类

4.3.1 MyCommands 类

改 public 权限，继承自 ICommand，根据提示引用 System.Windows.Input 命名空间、实现接口。增加 readonly 权限的_execute和_canExecute 成员，实现构造函数。

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Input;

namespace MyWpfApp.Commands
{
    public class MyCommands : ICommand
    {
        public event EventHandler CanExecuteChanged;
        public readonly Action<object> _execute;
        public readonly Predicate<object> _canExexute;

        public MyCommands(Action<object> execute, Predicate<object>
            canExecute)
        {
            this._execute = execute;
            this._canExexute = canExecute;
        }

        public bool CanExecute(object parameter)
        {
            return _canExexute(parameter);
        }

        public void Execute(object parameter)
        {
            _execute(parameter);
        }
    }
}
```

4.3.2 NotifyPropertyObject 类

改 public 权限，继承自 INotifyPropertyChanged，根据提示引用 System.ComponentModel 命名空间。编写 RaisePropertyChanged 函数，根据提示引用 System.Runtime.CompilerServices 命名空间

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

```
using System.ComponentModel;
using System.Linq;
using System.Runtime.CompilerServices;
using System.Text;
using System.Threading.Tasks;

namespace MyWpfApp.Models
{
    public class NotifyPropertyObject : INotifyPropertyChanged
    {
        public event PropertyChangedEventHandler PropertyChanged;

        public void RaisePropertyChanged([CallerMemberName] string
            propertyName = "")
        {
            PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(
                propertyName));
        }
    }
}
```

4.4 View-Model 类的编写

- 设置 public，继承 NotifyPropertyObject 类，设置构造函数
- 使用 #region 创建界面属性、私用变量、Commands 区域
- 利用 propfull，依次设置 FileName，Title，LStatus，Report，ObservableCollection(命名空间 System.Collections.ObjectModel) 等通用的属性

4.5 必要类的编写

- 对于数据含点的，应设计一个点类，包含无参构造函数和带参构造函数，甚至是点与点之间的距离与方位角的计算。
- 对于需要角度计算的，应设计 SurMath 类，编写静态的计算函数。
- 对于椭球相关的计算，需要设计椭球类，用以存放椭球的相关信息（ID，Name，两轴，两偏心率），必要时应设置一些自定义椭球的构造函数。

4.6 WPF 窗口设置

4.6.1 添加 ViewModels 所在的命名空间

```
xmlns:vm="clr-namespace:MyWpfApp.ViewModels"
```

4.6.2 设置 DataContext 环境

```
<Window.DataContext>
    <vm:MainWIndowVM/>
</Window.DataContext>
```

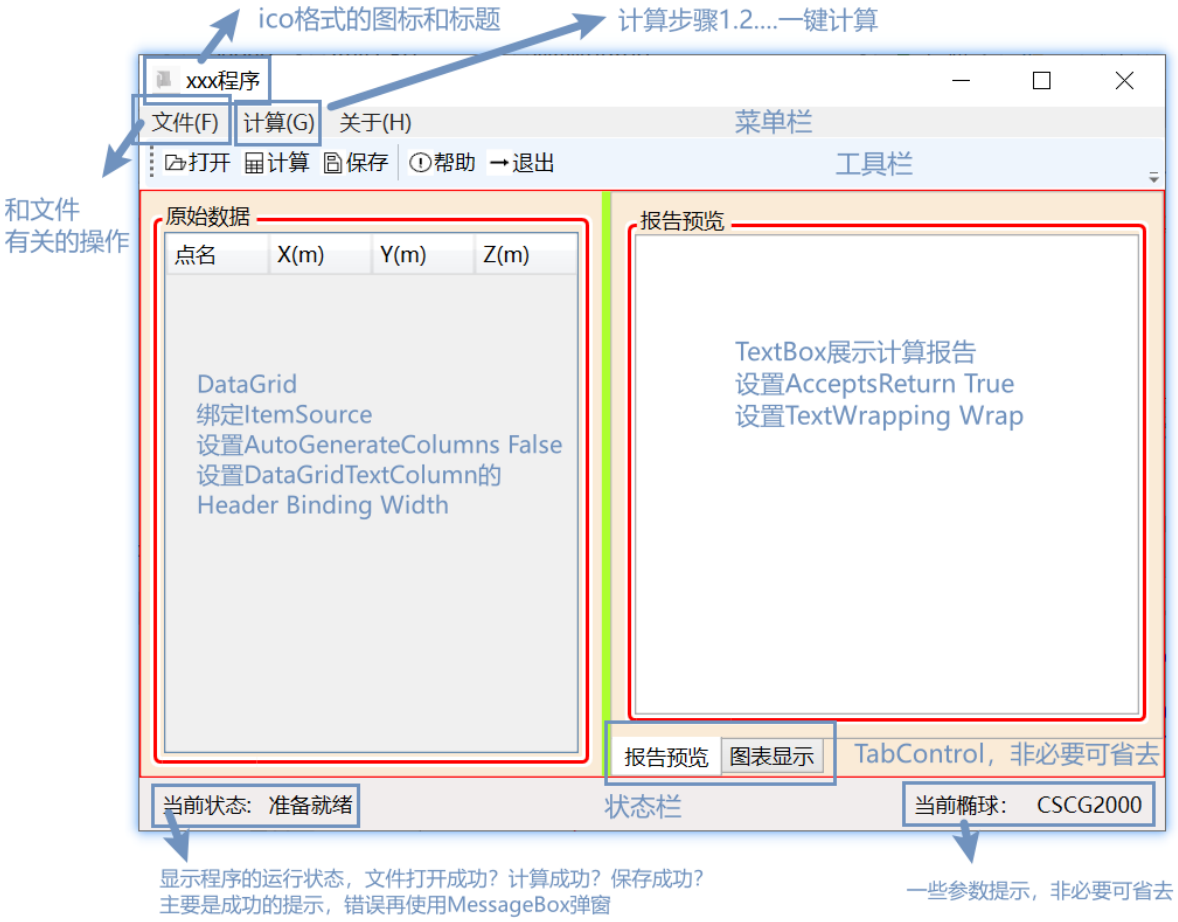
4.6.3 编写 XAML 界面

> 颜色推荐：AntiqueWhite（背景）、GreenYellow（边框 or 背景）、Red（边框）三种

> Window 标签的设置

- 设置窗体图标，Window 的 Icon 资源属性，注意相对路径从 xaml 文件开始计算，图标只能为 Icon 格式。
- 可以设置 WindowStartupLocation=”CenterScreen” 属性使得程序开始时位于屏幕中央
- 可以设置 WindowState=”Maximized” 属性默认最大化启动窗口

> 界面设计思路



> XAML 模板代码 (仅供参考)

```
<Window x:Class="MyWpfApp.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/
            presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-
            compatibility/2006"
        xmlns:local="clr-namespace:MyWpfApp"
        xmlns:vm="clr-namespace:MyWpfApp.ViewModels"
        mc:Ignorable="d"
        Title="{Binding Title}" Height="450" Width="600"  Icon="../
            Resources/Icon1.ico" WindowStartupLocation="CenterScreen"
        WindowState="Maximized">

    <Window.DataContext>
        <vm:MainWIndowVM/>
    </Window.DataContext>
```

```
15
16      <Window.Resources>
17          <Style TargetType="GroupBox">
18              <Setter Property="BorderThickness" Value="2"/>
19              <Setter Property="BorderBrush" Value="Red"/>
20              <Setter Property="Margin" Value="5"/>
21          </Style>
22
23      </Window.Resources>
24
25      <DockPanel LastChildFill="True">
26          <Menu DockPanel.Dock="Top">
27              <MenuItem Header="文件(F)">
28                  <MenuItem Header="打开" Command="{Binding
                        OpenFileCommand}">
29                      <MenuItem.Icon>
30                          <Image Source="../Resources/OpenFile2.bmp"/>
31                      </MenuItem.Icon>
32                  </MenuItem>
33                  <MenuItem Header="保存" Command="{Binding
                        SaveFileCommand}">
34                      <MenuItem.Icon>
35                          <Image Source="../Resources/Save.bmp"/>
36                      </MenuItem.Icon>
37                  </MenuItem>
38              </MenuItem>
39              <MenuItem Header="计算(G)">
40                  <MenuItem Header="一键计算"/>
41                  <Separator/>
42                  <MenuItem Header="计算步骤1"/>
43                  <MenuItem Header="计算步骤2"/>
44                  <MenuItem Header="计算步骤..."/>
45              </MenuItem>
46              <MenuItem Header="关于(H)">
47                  <MenuItem Header="帮助">
48                      <MenuItem.Icon>
49                          <Image Source="../Resources/Help.bmp"/>
50                      </MenuItem.Icon>
51                  </MenuItem>
52                  <MenuItem Header="退出">
53                      <MenuItem.Icon>
54                          <Image Source="../Resources/Close.bmp"/>
55                      </MenuItem.Icon>
56                  </MenuItem>
57              </MenuItem>
58          </Menu>
59          <ToolBar DockPanel.Dock="Top" Height="30">
60              <Button Command="{Binding OpenFileCommand}">
61                  <StackPanel Orientation="Horizontal">
62                      <Image Source="../Resources/OpenFile2.bmp" Height="
                            15" Width="15"/>
```

```
63                      <TextBlock Text="打开"/>
64                  </StackPanel>
65              </Button>
66
67              <Button Command="{Binding CalCommand}">
68                  <WrapPanel>
69                      <Image Source="../Resources/Cal.bmp" Height="15"
                            Width="15"/>
70                      <TextBlock Text="计算"/>
71                      <TextBlock/>
72                  </WrapPanel>
73              </Button>
74
75              <Button Command="{Binding SaveFileCommand}">
76                  <WrapPanel>
77                      <Image Source="../Resources/Save.bmp" Width="15"
                            Height="15"/>
78                      <TextBlock Text="保存"/>
79                  </WrapPanel>
80              </Button>
81              <Separator/>
82              <Button>
83                  <WrapPanel>
84                      <Image Source="../Resources/Help.bmp" Width="15"
                            Height="15"/>
85                      <TextBlock Text="帮助"/>
86                  </WrapPanel>
87              </Button>
88              <Button>
89                  <WrapPanel>
90                      <Image Source="../Resources/Close.bmp" Width="15"
                            Height="15"/>
91                      <TextBlock Text="退出"/>
92                  </WrapPanel>
93              </Button>
94          </ToolBar>
95
96          <StatusBar DockPanel.Dock="Bottom" Height="30">
97              <StatusBarItem HorizontalAlignment="Left" Margin="5,0,0,0">
98                  <WrapPanel>
99                      <Label Content="当前状态:"/>
100                     <Label Content="准备就绪"/>
101                 </WrapPanel>
102             </StatusBarItem>
103             <StatusBarItem HorizontalAlignment="Right" Margin="0,0,5,0"
                    >
104                 <WrapPanel>
105                     <Label Content="当前椭球: "/>
106                     <Label Content="CSCG2000"/>
107                 </WrapPanel>
108             </StatusBarItem>
```

```
109         </StatusBar>
110
111
112         <Border Background="AntiqueWhite" BorderBrush="Red"
                BorderThickness="1">
113             <Grid>
114                 <Grid.ColumnDefinitions>
115                     <ColumnDefinition Width="1*"/>
116                     <ColumnDefinition Width="5"/>
117                     <ColumnDefinition Width="1.2*"/>
118                 </Grid.ColumnDefinitions>
119
120                 <GroupBox Header="原始数据" Grid.Column="0">
121                     <DataGrid ItemsSource="{Binding Data}" Margin="2"
                            AutoGenerateColumns="False" CanUserAddRows="
                            False">
122                         <DataGrid.Columns>
123                             <DataGridTextColumn Header="点名" Binding="
                                    {Binding Name}" Width="1*"/>
124                             <DataGridTextColumn Header="X(m)" Binding="
                                    {Binding X,StringFormat={}{0:F3}}" Width
                                    ="1*"/>
125                             <DataGridTextColumn Header="Y(m)" Binding="
                                    {Binding Y,StringFormat={}{0:F3}}" Width
                                    ="1*"/>
126                             <DataGridTextColumn Header="Z(m)" Binding="
                                    {Binding Z,StringFormat={}{0:F3}}" Width
                                    ="1*"/>
127                         </DataGrid.Columns>
128                     </DataGrid>
129                 </GroupBox>
130
131                 <GridSplitter Grid.Column="1" Background="GreenYellow"
                        HorizontalAlignment="Stretch"/>
132                 <TabControl Grid.Column="2" TabStripPlacement="Bottom"
                        Background="AntiqueWhite">
133                     <TabItem Header="报告预览">
134                         <GroupBox Header="报告预览" Grid.Column="2">
135                             <TextBox Text="{Binding Report}"
                                    AcceptsReturn="True" TextWrapping="Wrap"
                                    />
136                         </GroupBox>
137                     </TabItem>
138                     <TabItem Header="图表显示">
139                         <GroupBox Header="图表显示" Grid.Column="2">
140                             <Canvas Background="White"></Canvas>
141                         </GroupBox>
142                     </TabItem>
143                 </TabControl>
144             </Grid>
145         </Border>
```

```
146         </DockPanel>
147     </Window>
```

5 一些编程注意事项

5.1 弧度（Rad）与角度（Deg）的区分

测绘的数据处理经常以角度和长度运算为主，在 C#、C/C++、Python 乃至近乎所有的编程语言中，对于正弦、余弦等三角函数的运算，都采用的是弧度制，而测绘的数据大多以浮点数 dd.mmsss 的形式进行录入，在运算时，要涉及到浮点数 dd.mmsss 无损精度提取度、分、秒，再将其转为十进制度，最后转为弧度参与计算。

在每次键盘敲下 Math 时，应下意识注意到参与运算的变量是弧度还是角度；也可以通过为角度变量设置_deg的标志来提醒自己。

5.2 int 类型与 int 类型运算结果为 int 类型

在 C#、C/C++ 编程语言中，整数与整数之间的运算仍为整数，这就会造成5/2等于 2 的情况，要想避免此类错误，最好的方式“非必要不 int，慎用 int”，实在无法避免可以采取5*1.0/2或5/2.0来避免，可以养成输入一个整数后面带小数点的习惯。

值得注意的是：由于语言自身的特性，这个问题在 Python 和 Matlab 中并不存在，这也对 Python 粉和 Matlab 粉敲响警钟。

5.3 条件语句 a<x<b 的书写

在 C#、C/C++ 编程语言中，对于条件语句a<x<b的计算会根据运算符的优先级先运算a<x得到一个 0 或者 1 的值再与 b 参与运算，实际编程中应该用与运算符分段连接a<x && x<b。

值得注意的是：a<x<b的写法是人们根据经验最容易犯错的写法，并且实际中编译也可以通过但运算结果不会百分百正确；而 Python 语法却支持该写法并且可以运算正确，习惯 Python 编程的读者应注意此问题。

6 参考

1. 马骏. C# 程序设计及应用教程 (第三版) [M]. 北京: 人民邮电出版社, 2014.
2. 李英冰. 测绘程序设计 [M]. 武汉: 武汉大学出版社, 2019.

3. GitHub 仓库：https://github.com/LiZhengXiao99/Geomatics-Program
4. 微软官方帮助文档：https://learn.microsoft.com/zh-cn/dotnet/csharp/

7 附录

7.1 相关资料

- 教育部高等学校测绘类专业教学指导委员会通知官网：比赛通知、结果发布地址；
- 智绘未来 B 站账号：赛前培训直播和录播；
- 测绘程序设计教材/例程/配套视频：测绘程序设计比赛的官方参考书、参考例程；
- 测绘程序设计比赛讲解-回放：23 年李英冰老师做的赛题讲解；
- 罗宏昆的 23 测绘程序设计仓库 CeHuiProgramDesign)/24 赛题讲解视频/界面设计视频；

7.2 2024 比赛日程


SOUTH
2024年全国大学生
测绘学科创新创业智能大赛
主办单位：中国测绘学会教育工作委员会
承办单位：滇西应用技术大学
赛前日程安排
4月
1日 下发2024年版竞赛软件及练习账号
23日 数字测图赛项全国线上培训
24日 无人机航测赛项全国线上培训
25日 机载激光雷达赛项全国线上培训
5月
1日 所有单独校赛、省赛可预约
7、15、22、29日 无人机航测赛项全国线上培训+答疑
8、14、21、28日 机载激光雷达赛项全国线上培训+答疑
9、16、23、30日 数字测图赛项全国线上培训+答疑
6月
5日 专家讲解测绘程序设计
12日 专家讲解开发设计竞赛
19日 专家讲解科技论文竞赛
20日 全国校赛、省赛报名截止
7月
3日 全国第一场模拟赛（2日赛前说明会、4日赛后总结会）
10日 全国第二场模拟赛（9日赛前说明会、11日赛后总结会）
17日 全国第三场模拟赛（16日赛前说明会、18日赛后总结会）
24日 赛前流程测试
请各参赛院校
及时关注南方直播培训具体时间
如有需求，请与南方测绘各地分公司联系

7.3 2024 比赛赛题

附件 3：

全国大学生测绘学科创新创业智能大赛——测绘技能竞赛

测绘程序设计比赛选题及说明

一、比赛选题

1. 空间数据探索性分析：计算标准误差椭圆、空间权重矩阵、常用空间自相关指数。

2. 遥感图像空间前方交会计算：计算立体像对的投影系数、像空间辅助坐标系坐标及地面摄影测量坐标系坐标。

3. GNSS 空间大气改正计算：常用电离层改正模型、对流层改正模型计算。

4. 纵横断面计算：道路纵断面、横断面的相关点位计算，以及断面面积计算。

5. 大地线长度计算：根据地球椭球参数和椭球面上的大地经纬度坐标，计算两点之间的大地线长度。

6. 曲线拟合：利用五点光滑法进行曲线拟合。

二、比赛说明

1. 比赛形式：选手单人参赛，比赛时间 4 小时。从 6 道候选题中选择 1 道题目作为比赛题目。

2. 开发环境与编程语言：编程环境为 Visual Studio2017；编程语言限制为 Basic、C/C++、C#，不允许使用二次开发平台（如 Matlab、Python 等）。

3. 输入数据说明：数据文件为文本文件(.txt)。

4. 计算成果要求：计算成果包括中间过程数据和成果数据等内容，根据要求进行输出，并根据试题册说明，将计算成果录入考试系统。

5. 用户界面要求：界面风格采用标准 Window 应用程序，包括菜单、工具条、主窗体、状态栏等要素构成。其中菜单包含文件、算法、显示等内容。