C#程序设计结课报告

计算器

班级：2013155

学号：201315505

姓名：王 宁 波

使用WinForm实现简单计算器程序：

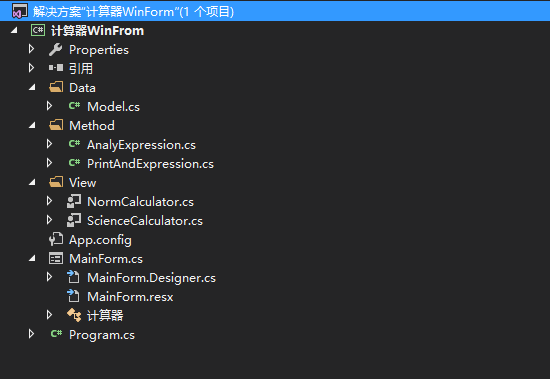
程序实现过程：

* 首先设计主窗体，包括两个TextBox用来显示数据，使用数据绑定实现，在Model中实现INotifyPropertyChanged接口，以起到数据更改通知的功能。一个Button和Panel作为菜单来切换计算器模式。最后一个Panel用来显示不同模式的计算器输入面板
* 其次是标准计算器输入面板。包括常用的数字和运算符以及Delete，CE和清除按钮。科学计算模式也只是多了几个运算符而已。在后置代码中获取被单击的按钮的名称，调用Model中的Print方法在TextBox中显示数据。
* Model中声明两个字符串变量作为两个TextBox中的数据，还包括Print方法，一方面得到按钮的名称，另一方面调用此程序中最重要的逻辑处理方法PrintText方法。传进去两个值，即当前TextBox中显示的数据，输出两个值作为TextBox在一系列处理之后应显示的数据。
* PrintAndExpression类主要负责处理TextBox的显示和生成算术表达式。GetValue方法根据名字返回应显示的字符串。IsOpreation方法判断字符串是数字，一元操作符还是二元操作符及其他符号。PrintText处理显示格式。大致为当输入数字的时候在textBox1中，每按下一个操作符即将textBox1中的数字连接上操作符显示在textBox2中，同时处理简单的一元运算符的运算，生成表达式。最终将表达式传递给AnalyExpressions。将解析表达式运算之后的结果返回。
* AnalyExpressions解析表达式，利用堆栈来实现，将中缀表达式分解之后保存在字符串数组中，即所谓的“后缀表达式”。大致规则为：
* 将中缀表达式翻译成后缀表达式
* 输入中缀表达式： A+B\*(C+D)-E/F
* 中缀表达式翻译成后缀表达式的方法如下：
* （1）从左向右依次取得数据ch
* （2）如果ch是操作数，直接输出
* （3）如果ch是运算符（含左右括号），则：
* a：如果ch = "("，放入堆栈
* b：如果ch = ")"，依次输出堆栈中的运算符， 直到遇到"("为止
* c：如果ch不是")"或者"("，那么就和堆栈顶点 位置的运算符top做优先级比较
* 1：如果ch优先级比top高，那么 将ch放入堆栈
* 2：如果ch优先级低于或者等于 top，那么输出top，然后将ch放入堆栈
* （4）如果表达式已经读取完成，而堆栈中还有运算符时，依次由顶端输出

**结语：**

我就是那个先用WPF做了一遍又用WinForm做了一遍的人。好在是用MVVM模式做的（也全是为了练手，徒增很多了麻烦），但是好处也就立刻体会到了。换成WinForm需要重新做三个界面，把命令和绑定换一下就可以了。其他的代码就写的很烂啦，可见我离合格的软件开发者还差的很远，期间也遇到很多问题，直至完成也有一些无法实现，比如负数只对数字有用。

**目录结构：**



* Data文件夹
* Model.cs为程序执行所操作的数据
* Method文件夹
* PrintAndExpression.cs为事件处理程序所调用的方法，主要用于处理显示数据
* AnalyExpression.cs用于将算术表达式解析为“后缀表。达式数组”并计算返回计算结果
* View文件
* NormCalculator.cs为用户控件显示标准计算器视图
* ScienceCalculator.cs用户控件显示科学计算器视图
* MainForm.cs为程序主窗体
* Program.cs为程序入口

**Data.cs**代码：

Method.cs代码：

using System.ComponentModel;

namespace 计算器WinForm.Data

{

public class Model:INotifyPropertyChanged

{

public event PropertyChangedEventHandler PropertyChanged;

public string num1;

public string num2;

public string Num1

{

get { return num1; }

set

{

num1 = value;

this.RaisePropertyChanged("Num1");

}

}

public string Num2

{

get { return num2; }

set

{

num2 = value;

this.RaisePropertyChanged("Num2");

}

}

private void RaisePropertyChanged(string propertyName)

{

var handler = this.PropertyChanged;

if(handler!=null)

{

handler(this, new PropertyChangedEventArgs(propertyName));

}

}

//声明一个实例，供全局调用

public static Model model = new Model();

//打印方法，每按下一个按钮在TextBox中显示Value

public void Print(string name)

{

Method.PrintAndExpression method = new Method.PrintAndExpression();

string outnum1, outnum2;//获取输出参数

//将当前Numb1和Num2的值传递过去

method.PrintText(name,Num1,Num2,out outnum1,out outnum2);

Num1 = outnum1;

Num2 = outnum2;

}

}

}

PrintAndExpression.cs部分代码：

using System;

namespace 计算器WinForm.Method

{

public class Method

{

//保存表达式

public static string expression;

public void PrintText(string name, string GetText1, string GetTxet2, out string SetText1, out string SetText2)

{

//每按下一个按钮，将按钮所表示的含义打印到TextBlock中

//如果为数字（包括多位数），暂时保存在SetText1中。即下方那一栏文本框

//如果为一元运算符，将GetText1中的数字连同一元运算符一并保存到SetText2中，表达式直接获得一元运算符计算结果

//如果为二元运算符，将GetText1中的数字连同二元运算符一并保存到SetText2中。

//判断str1是否含有等号。和算术表达式异常提示信息。则先清空。

if (GetText1 != null && GetText1.Contains("="))

{

GetTxet2 = null;

GetText1 = null;

}

else if (GetText1 == "你的算式无法计算" || GetText1 == "除数不能为零" || GetText1 == "输入无效")

{

GetTxet2 = null;

GetText1 = null;

}

//是否为操作符，数字

int isOperation = Is0peration(name);

if (isOperation == 0) //数字

{

if(GetText1=="0")

{

GetText1 = "";

}

SetText2 = GetTxet2;

SetText1 = GetText1 + GetValue(name); //显示

expression = GetTxet2;

}

else if (isOperation == 1) //一元运算符

{

SetText1 = GetText1;

//对于一元运算符直接调用Math的方法计算出结果

try

{

switch (name)

{

case "square":

SetText2 = GetTxet2 + "(" + GetText1 + ")²";

expression = expression + float.Parse(GetText1) \* float.Parse(GetText1) + "";

break;

case "sin":

SetText2 = GetTxet2 + GetValue(name) + "(" + GetText1 + ")";

expression = expression + Math.Sin(float.Parse(GetText1)) + "";

break;

case "cos":

SetText2 = GetTxet2 + GetValue(name) + "(" + GetText1 + ")";

expression = expression + Math.Cos(float.Parse(GetText1)) + "";

break;

case "tan":

SetText2 = GetTxet2 + GetValue(name) + "(" + GetText1 + ")";

expression = expression + Math.Tan(float.Parse(GetText1)) + "";

break;

case "radication":

SetText2 = GetTxet2 + GetValue(name) + "(" + GetText1 + ")";

expression = expression + Math.Sqrt(float.Parse(GetText1)) + "";

break;

case "shimi":

SetText2 = GetTxet2 + "10^" + GetText1;

expression = expression + Math.Pow(10, float.Parse(GetText1)) + "";

break;

case "fact":

SetText2 = GetTxet2 + GetText1 + "!";

expression = expression + Fact(GetText1) + "";

break;

case "minus": //负号

SetText2 = GetTxet2 + "(" + "-" + GetText1 + ")";

expression = expression + "(0" + "-" + GetText1 + ")";

break;

case "daoshu":

SetText2 = GetTxet2 + "1⁄" + GetText1;

if (0 == int.Parse(GetText1))

{

SetText1 = "除数不能为零";

break;

}

else

{

expression = expression + 1 / float.Parse(GetText1) + "";

}

break;

case "exp":

SetText2 = GetTxet2 + "exp(" + GetText1 + ")";

expression = expression + Math.Exp(float.Parse(GetText1)) + "";

break;

case "log":

SetText2 = GetTxet2 + "log(" + GetText1 + ")";

if (0 == int.Parse(GetText1))

{

SetText1 = "输入无效";

break;

}

else

{

expression = expression + Math.Log(float.Parse(GetText1)) + "";

}

break;

default:

SetText2 = GetTxet2;

SetText1 = "计算错误";

break;

}

SetText1 = "";

GetText1 = "";

}

catch (Exception e)

{

SetText1 = "输入无效";

SetText2 = "";

}

}

else if (isOperation == 2) //二元运算符

{

//如果运算符为左括号

if (name == "left")

{

SetText2 = GetTxet2 + GetValue(name);

SetText1 = "";

expression = expression + GetValue(name);

}

//如果运算符为右括号

else if (name == "right")

{

SetText2 = GetTxet2 + GetText1 + GetValue(name);

SetText1 = "";

expression = expression + GetText1 + GetValue(name);

}

else

{

SetText2 = GetTxet2 + GetText1 + GetValue(name); //显示

SetText1 = "";

expression = expression + GetText1 + GetValue(name);

}

}

else if (GetValue(name) == "delete")

{

//将Text1中的数依次删除

SetText2 = GetTxet2;

if(GetText1!="0"&&GetText1!=null&&GetText1!="")

{

SetText1 = GetText1.Remove(GetText1.Length - 1, 1);

}

else if(GetText1==null)

{

SetText1 = "0";

}

else

{

SetText1 = GetText1;

}

expression = GetTxet2;

}

else if (GetValue(name) == "CE")

{

//将Text1中的数清空

SetText2 = GetTxet2;

SetText1 = "0";

expression = GetTxet2;

}

else if (GetValue(name) == "C")

{

//清空所有内容

SetText2 = "";

SetText1 = "0";

expression = null;

}

else if (GetValue(name) == "=")

{

SetText2 = GetTxet2 + GetText1;

expression = expression + GetText1;

if (expression == null)

{

SetText1 = "";

}

else

{

//添加一个等号，标记为计算结果。再输入数字的时候应该先清空。

//SetText1 = "=" + Caculate();

string result = Caculate();

SetText1 = result == "error" ? "你的算式无法计算" : "=" + result;

}

}

else

{

SetText1 = "error";

SetText2 = "error";

}

}

public string Caculate()

{

string str = expression;

//调用算术表达式解析算法

double result = 0;

//捕获异常

try

{

AnalyExpression.AnalyExpressions(str, out result);

}

catch (Exception e)

{

result = 0;

return "error"; //出现异常返回error

}

return result + "";

}}}

AnalyExpression.cs代码：

using System;

using System.Collections;

namespace 计算器WinForm.Method

{

public class PosfixParser

{

//声明表达式并赋初值

public static string expression;

//声明堆栈

private static Stack myStack = new Stack();

//作为主函数，供外界调用，传入参数为表达式，输出参数为解析并计算表达式之后的结果

public static void Analysisexpressions(string exp, out double res)

{

//将表达式赋值为传进来的参数exp

expression = exp;

//对B初始化，每一个索引全部赋值为空

for (int i = 0; i < B.Length; i++)

{

B[i] = null;

}

//调用解析方法，将中缀表达式解析为类似的“后缀表达式”

Parse();

//调用计算方法，对“后缀表达式”进行计算，返回计算结果

res = Calculate();

}

//声明数组，将中缀表达式解析之后放入此数组中

static string[] B = new string[100];

//标记一个多位数字是否为小数（带小数点）

static bool haspoint;

//标记小数位的位数，从1开始计数

static int figure = 1;

//将中缀表达式解析成后缀表达式

public static void Parse()

{

int i, j = 0;

string ch, ch1;

char[] A = expression.ToCharArray(); //将字符串转成字符数组

int length = A.Length;

int index = 1; //记录当前数字的索引

for (i = 0; i < length; i++)

{

ch = A[i] + ""; //对字符之后添加一个空引号，以隐式转换为字符串

//强制转换字符char为字符串string会转换为对应的ASCII码

if (IsOperand(ch)) //如果是操作数，直接放入B中

{

//中缀表达式被分解为字符数组，因此在支持两位以上的数字时

//首先记录一个数字的索引，假如第二个数字的索引与第一个数字的索引之差为1

//则说明两个数字应组成为一个多位数

if (index == i - 1)

{

index = i;

//如果当前字符为小数点，则在数组B上一个索引后追加“.0”，并标记当前数据为小数

if (ch == ".")

{

B[--j] = B[j] + ".0";

haspoint = true;

}

//如果当前数据为小数，则之后的字符追加到此数据之后，按照以下规则

else if (haspoint)

{

B[--j] = (double.Parse(B[j])) + Convert.ToDouble(ch.ToString()) \* (1 / (Math.Pow(10, figure))) + "";

figure++;

}

//当前数据不是小数，追加到此数据之后，按照以下规则

else

{

B[--j] = (double.Parse(B[j]) \* 10 + Convert.ToDouble(ch.ToString())) + "";

}

++j;

}

else

{

index = i; //记录当前数字的索引

B[j++] = ch + "";

}

}

else

{

if (ch == "(") //如果是“(”，将它放入堆栈中

myStack.Push(ch);

else if (ch == ")") //如果是“)”

{

while (!IsEmpty(myStack)) //不停地弹出堆栈中的内容，直到遇到“(”

{

ch = (string)myStack.Pop();

if (ch == "(")

break;

else

B[j++] = ch + ""; //将堆栈中弹出的内容放入B中

}

}

else //既不是“(”，也不是“)”，是其它操作符，比如 +, -, \*, / 之类的

{

if (!IsEmpty(myStack))

{

do

{

ch1 = (string)myStack.Pop();//弹出栈顶元素

if (Priority(ch) > Priority(ch1)) //如果栈顶元素的优先级小于读取到的操作符

{

myStack.Push(ch1);//将栈顶元素放回堆栈

myStack.Push(ch);//将读取到的操作符放回堆栈

break;

}

else//如果栈顶元素的优先级比较高或者两者相等时

{

B[j++] = ch1 + ""; //将栈顶元素弹出，放入B中

if (IsEmpty(myStack))

{

myStack.Push(ch); //将读取到的操作符压入堆栈中

break;

}

}

} while (!IsEmpty(myStack));

}

else //如果堆栈为空，就把操作符放入堆栈中

{

myStack.Push(ch);

}

}

}

}

while (!IsEmpty(myStack))

{

B[j++] = myStack.Pop() + "";//将堆栈中剩下的操作符输出到B中

}

}

//计算“后缀表达式”的值

public static double Calculate()

{

int i;

double no1, no2, ret;

string ch;

//把B中的null值去掉

int n = 0;

for (; n < B.Length; n++)

{

if (B[n] == null)

{

break;

}

}

string[] A = new string[n];

//将B的非null数据复制到A中

for (n = 0; n < A.Length; n++)

{

A[n] = B[n];

}

myStack.Clear();

for (i = 0; i < A.Length; i++)

{

ch = A[i];

if (IsOperand(ch))//如果是操作数，直接 压入栈

{

myStack.Push(double.Parse(ch));

}

else //如果是操作符，就弹出两个数字来进行运算

{

no1 = (double)myStack.Pop();

no2 = (double)myStack.Pop();

ret = GetValue(ch, no1, no2);

myStack.Push(ret);//将结果压入栈

}

}

return (double)myStack.Pop();//弹出最后的运算结果

}

//对两个值利用运算符计算结果

private static double GetValue(string op, double ch1, double ch2)

{

switch (op)

{

case "+":

return ch2 + ch1;

case "-":

return ch2 - ch1;

case "\*":

return ch2 \* ch1;

case "/":

return ch2 / ch1;

default:

return 0;

}

}

//判断堆栈是否为空

private static bool IsEmpty(Stack st)

{

return st.Count == 0 ? true : false;

}

//判断是否是操作数

private static bool IsOperand(string ch)

{

string[] operators = { "+", "-", "\*", "/", "(", ")" };

for (int i = 0; i < operators.Length; i++)

if (ch == operators[i])

return false;

return true;

}

//返回运算符的优先级

private static int Priority(string ch)

{

int priority;

switch (ch)

{

case "+":

priority = 1;

break;

case "-":

priority = 1;

break;

case "\*":

priority = 2;

break;

case "/":

priority = 2;

break;

default:

priority = 0;

break;

}

return priority;

}

}

}

**NormCaalculator视图后置代码：**

using System;

using System.Windows.Forms;

using 计算器WinForm.Data;

namespace 计算器WinForm.View

{

public partial class NormCalculator : UserControl

{

public NormCalculator()

{

InitializeComponent();

}

private void GetButtonClickedName(object sender, EventArgs e)

{

//获取被单击控件的名字

string name = ((System.Windows.Forms.Control)sender).Name;

//MessageBox.Show(name);

//调用打印方法

Model.model.Print(name);

}

}

}

**SccienceCalculator视图后台代码同上：**

**MainForm.cs代码：**

using System;

using System.Windows.Forms;

using 计算器WinForm.Data;

using 计算器WinForm.View;

namespace 计算器WinForm

{

public partial class 计算器 : Form

{

public 计算器()

{

InitializeComponent();

//初始化计算器为标准模式

panel1.Controls.Clear();

NormCalculator normCalculator = new NormCalculator();

normCalculator.Show();

panel1.Controls.Add(normCalculator);

//初始化Model

Model.model.Num2 = null;

Model.model.Num1 = "0";

//textBox的数据绑定，参数;控件的属性，数据源，数据成员

textBox2.DataBindings.Add("Text", Model.model, "Num2");

textBox1.DataBindings.Add("Text", Model.model, "Num1");

}

//打开菜单

private void button\_menu\_Click(object sender, EventArgs e)

{

if(menulist.Visible==true)

{

menulist.Visible = false;

}

else

{

menulist.Visible = true;

}

}

//切换到标准计算器模式

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

{

//初始化Model

Model.model.Num2 = null;

Model.model.Num1 = "0";

//先清空panel1中的控件，在添加新的控件

panel1.Controls.Clear();

NormCalculator normCalculator = new NormCalculator();

normCalculator.Show();

panel1.Controls.Add(normCalculator);

menulist.Visible = false;

label1.Text = "标准";

}

//切换到科学计算器模式

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

{

//初始化Model

Model.model.Num2 = null;

Model.model.Num1 = "0";

//先清空panel1中的控件，在添加新的控件

panel1.Controls.Clear();

ScienceCalculator scienceCalculator = new ScienceCalculator();

scienceCalculator.Show();

panel1.Controls.Add(scienceCalculator);

menulist.Visible = false;

label1.Text = "科学";

}

private void textBox2\_TextChanged(object sender, EventArgs e)

{

if(textBox2.TextLength>=24)

{

MessageBox.Show("算式过长，不可再继续输入！");

}

}

}

**功能展示：**



