武汉大学计算机学院

本科生实验报告

简易科学计算器的总体设计与实现

专 业 名 称 ：软件工程

课 程 名 称 ：Windows编程实验

指 导 教 师 一：胡继承 教授

学 生 学 号 ：2017302580170

学 生 姓 名 ：周潜

二○一九年十二月

郑 重 声 明

本人呈交的实验报告，是在指导老师的指导下，独立进行实验工作所取得的成果，所有数据、图片资料真实可靠。尽我所知，除文中已经注明引用的内容外，本实验报告不包含他人享有著作权的内容。对本实验报告做出贡献的其他个人和集体，均已在文中以明确的方式标明。本实验报告的知识产权归属于培养单位。

本人签名： 周潜 日期： 2019.12.20

摘 要

本实验的实验目的是加强学生对于Windows编程中动态链接库DLL的理解，熟悉DLL的创建和调用过程，对在Windows环境下编写GUI程序的相关技术，例如MFC、WPF、Winform要熟练掌握。

实验设计主要采用逆波兰表达式解析算法，将用户输入的中序表达式转为后缀表达式，然后进行计算，界面采用Winform技术。

实验内容主要包括：界面设计、逆波兰表达式解析、错误提示设计。

实验结论为实现计算器的基本功能，能够正确输出结果，有错误提示。

关键词**：**科学计算器；C#；Winform

**目 录**

[1 实验目的和意义 2](#_Toc21908717)

[1.1 实验目的 2](#_Toc21908718)

[1.1.1 目的一：检验XXX（黑体小4号加粗） 2](#_Toc21908719)

[2 实验目的和意义 2](#_Toc21908720)

[2.1 实验目的 2](#_Toc21908721)

[2.1.1 目的一：检验XXX（黑体小4号加粗） 2](#_Toc21908722)

[3 实验目的和意义 2](#_Toc21908723)

[3.1 实验目的 2](#_Toc21908724)

[3.1.1 目的一：检验XXX（黑体小4号加粗） 2](#_Toc21908725)

[3.1.2 2](#_Toc21908726)

1. 实验目的和意义
   1. 实验**目的**

本实验的主要目的是设计一个计算器，要求通过DLL实现加减乘除的基本四则运算，并且兼容浮点型和整型的输出。同时对于非法数据或无效数据要有相应提示，对输入错误的数据可以进行删除操作。

* 1. 实验**意义**

本实验可加强学生对于动态链接库创建和使用的理解，熟练掌握Windows编程下的界面设计的技术，并对编程能力有所提升。

1. 实验设计
   1. 概述

本实验主要需求如下：

（1）实现加、减、乘、除四种基本运算功能

（2）上述四种运算均要求使用dll实现，其中dll使用c++或c#编程语言来创建

（3）上述四种运算输入、操作和运算结果均要求直接在界面中进行显示，其中界面可以选用MFC、Winform、WPF技术实现

（4）上述四种运算的实现均要求输入数据兼容数据类型int、double，输出结果也兼容数据类型int、double

（5）\*对于输入的非法数据或无效数据要进行相关提示；

（6）\*对于输入的数据有回退删除功能

针对上述需求，做出如下设计和准备：

1. 界面设计采用Winform，用户在输入区输入相应表达式，结果在输出区显示。
2. 字符串表达式解析采用逆波兰解析算法，将中序表达式转化为后缀表达式，然后计算结果。
3. 错误提示，可以先存储字符串表达式每个单元的类型，然后遍历进行判断。
4. 编程语言采用C#，集成开发环境是Visual Studio 2019 Community版本，使用C#创建托管DLL。
   1. 实验原理

本实验的主要难点在于逆波兰表达式算法（又称后缀表达式算法），其中心思想在于，将中序表达式转化为后序表达式，然后求值。表达式中的基本单元主要分为操作数和运算符。其中操作数可分为布尔型、整型、浮点型，运算符可分为一元运算符（例如平方运算“X^2”）、二元运算符（例如加法运算“+”），而一元运算符又可细分为前置一元运算符（在数字之前的运算符，例如正弦函数“Sin”）和后置一元运算符（在数字之后的运算符，例如平方运算“^2”）。得到这些基本运算单元比较简单，用户每输入一个，就添加一个。难点在于表达式的转换和求值，下面讲述这两个算法。

* + 1. 算法一：将中缀表达式转换成后缀表达式算法

中缀表达式符合人们的书写方式，但用计算机去实现比较困难，主要在于运算符的优先级问题，在解析表达式时，不能做到边解析边计算，因为无法得知后续运算符是否比当前运算符优先级高，同时括号也会影响运算顺序。而后缀表达式当中不存在括号，运算符存放的顺序就是实际的运算次序，可以做到一边扫描即可得出结果。将中缀表达式转化为后缀表达式的算法如下：先声明两个堆栈，一个是运算符堆栈，一个是操作数堆栈。

1、从左至右扫描中缀表达式，读取一个表达式单元。

2、若读取的是操作数，将该操作数存入操作数堆栈。

3、若读取的是运算符，则进行如下判断：

(1) 若该运算符为左括号"("，则直接存入运算符堆栈。

(2) 若该运算符为右括号")"，则输出运算符堆栈中的运算符到操作数堆栈，直到遇到左括号为止。

(3) 若该运算符为非括号运算符：

(a) 若运算符堆栈栈顶的运算符为括号，则直接存入运算符堆栈。

(b) 若比运算符堆栈栈顶的运算符优先级高，则直接存入运算符堆栈。

(c) 若比运算符堆栈栈顶的运算符优先级低或者相等，则输出栈顶运算符到操作数堆栈，并将当前运算符压入运算符堆栈。

4、重复1-3直到表达式读取完成。若读取完成后，运算符堆栈中还有运算符，则依次取出运算符到操作数堆栈，直到运算符堆栈为空。

上述算法涉及到运算符的优先级判断，基于此，在定义运算符时要包含优先级，并可进行优先级比较。考虑到要将运算符存放在操作数堆栈，实际编码时可以将操作数作为父类，运算符作为子类继承操作数类。

* + 1. 算法二：逆波兰表达式求值算法

      在后缀表达式中不存在括号，也不需要考虑运算符的优先级，运算符存放的顺序就是实际运算的顺序，因此求值算法比较简单，具体如下：

1. 初始化一个空堆栈
2. 从左到右读入后缀表达式
3. 如果读取到的是操作数，把它压入堆栈。
4. 如果读取到的是运算符，则进行如下判断：
   1. 如果是一元运算符，从堆栈弹出一个操作数，执行相关运算，然后把结果作为操作数压入堆栈。
   2. 如果是二元运算符，从堆栈弹出两个操作数，执行相关运算，然后把结果作为操作数压入堆栈。
5. 重复2-4，直到后缀表达式读取完毕，从堆栈中弹出结果，此时堆栈应为空。

上述算法可以对后缀表达式进行计算，得出最后结果。以上各步如若出现任何问题，都说明后缀表达式格式不正确。例如，读取到的是一元运算符，但从堆栈弹出的是运算符不是操作数。

* 1. 实验设计
     1. 方案设计

本实验对表达式求值的设计经历了三个方案，分别如下：

* + - 1. 方案一：对表达式直接求值

用户每按下一个运算符就进行一次求值，类似于小商店使用的语音计算器（如图2.1），主要用于进行加减乘除四则运算，不能带括号。例如，用户依次按下2，+，5，-，6，/，7，=，当按下2+5后，输入区变成7，用户按下-6后，输入去变成1，用户按下/7后，输入区变成0.142857，按下=后，输出区显示0.142857。此方案实现起来较为简单，但缺点在于忽略了运算符优先级，不能带有括号，功能过于简单。



图2.1 小商店使用的语音计算器

* + - 1. 方案二：构建表达式树

此方案借助于编译原理的思想，分为三个主要部分，词法分析、语法分析和语义分析。词法分析就是得到一个个的表达式基本单元，语法分析就是依据这些基本单元构建表达式树，语义分析就是依据表达式树来进行计算。在表达式树中每个叶子节点都只能是操作数，每个双亲结点都是运算符。例如如果用户输入2+5^2/sin3-6\*(8+2)，那么表达式树如图2.2所示。本方案的难点在于构建表达式树，由表达式树可以看出，表达式树中最重要的部分就是双亲节点—运算符。因此表达式树的构建算法如下：

1. 依次扫描中序表达式，只读取运算符，并记录当前读取运算符的位置，将第一个运算符作为根节点。
2. 每读取一个运算符就新建一个节点，并将其和根节点运算符的优先级进行比较。如果优先级高于根节点，就作为根节点的孩子节点。否则，将根节点作为自己的孩子节点。
3. 如果读取的是前置一元运算符，则将当前运算符位置的后一个单元作为孩子节点。
4. 如果读取的是后置一元运算符，则将当前运算符的前一个单元作为孩子节点。
5. 如果读取的是二元运算符，则将当前运算符的前一个单元和后一个单元作为孩子节点。
6. 如果读取的是左括号，就新建一个表达式树，直到遇到右括号时，再将新建的语法树和之前的语法树链接起来。

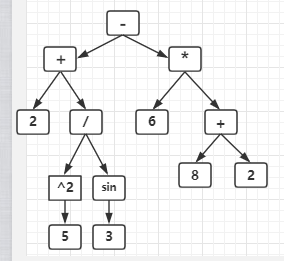


图2.2 表达式语法树

此方案的优点在于可拓展性好，一元运算符、二元运算符、三元运算符甚至更复杂的运算符都可以实现。但是构建表达式树的过程实在太过麻烦，上述构建表达式树的算法有些地方描述的也不太清楚。而且构建好表达式树之后，计算最终结果也需要后序遍历表达式树，这和逆波兰表达式算法有点类似。

* + - 1. 方案三：逆波兰表达式算法

逆波兰表达式算法，主要分为两个部分：表达式转换和表达式求值。相关算法在2.2节已有详细介绍，此处就不赘述。此方案的优点在于，实现相对方案二比较简单，功能也比较完善，可以实现很多复杂的运算。

* + 1. 界面设计

界面采用Winform技术实现，整体布局采用TableLayoutPanel实现，示意图如图2.3所示。最外层是一个三行一列的tableLayouPanel，第三行嵌套了一个一行三列的tableLayoutPanel，每个Group里面又嵌套了一个tableLaoutPanel。最后界面设计如图2.4所示。

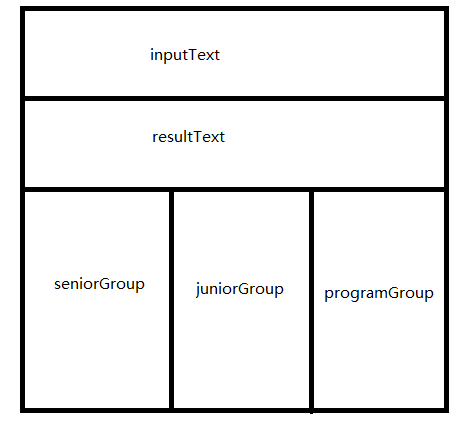


图2.3 界面布局

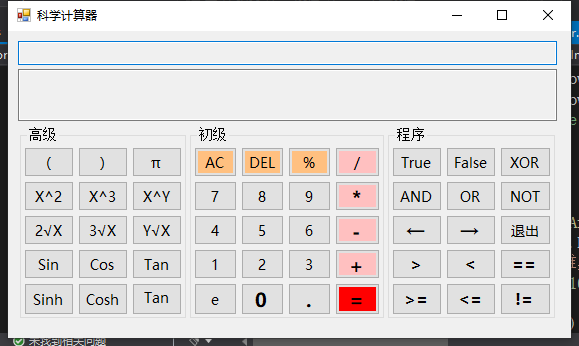


图2.4 最终界面设计

1. 代码实现和分析
   1. 界面设计代码

界面设计部分，采用Winform技术，可以可视化拖动相应控件并修改相应属性，所以这一部分的代码是VS2019自动生成的，就不展示具体代码了。

* 1. 按钮响应代码

按钮分为三大部分，一个是运算符按钮，一个是控制按钮，一个是数字按钮。这三大部分对应不同的相应事件。

* + 1. 数字按钮

数字按钮主要是0-9和小数点，由于用户输入一个数字后，对应的并不是一个完整的表达式单元，用户还可以继续输入。所以要建立一个数字缓冲区，当用户按下一个数字按钮，就将数字写入缓冲区。当用户按下一个非数字按钮时，则判断数字缓冲区是否为空，如果不为空，判断数字格式是否正确，如果正确就将数字添加到表达式中。因此，当用户按下数字按钮时，并不会在输入区立即显示数字，而是将数字临时显示在结果区。判断数字格式是否正确，主要是小数点不能在第一位并且小数点的个数至多1个。如果格式不正确，就会清空缓冲区。数值类型按钮还有常量按钮和布尔值按钮，这两类和其它数字按钮不同，不需要数字缓冲区，每次按下只需要直接写入表达式字符串即可。

常量按钮点击事件如下：showText()函数是为了刷新输入区。

//常量按钮点击事件

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

{

if (!checkDigit()) //检查数字缓冲区

return;

if (curIndex >= 0)

curSelect += expr[curIndex].Length;

curIndex++;

expr.Insert(curIndex, "e");

types.Insert(curIndex, type.digit);

showText();

}

数字按钮响应事件如下：curIndex是当前表达式单元的索引，curSelect是当前光标所在位置。

private StringBuilder digit = new StringBuilder();//数字缓冲区

private void dight\_Click(object sender, EventArgs e) //数字按钮响应事件

{

if (sender is Button)

{

Button btn = (Button)sender;

string text = btn.Text;

digit.Append(text);

resultTxtBox.Text = digit.ToString();

}

}

private bool checkDigit() //检查数字格式是否正确

{

int pointCount = 0; //小数点的数量

resultTxtBox.Text = "";

if (digit.Length != 0)

{

if (digit[0] == '.')

{

MessageBox.Show("小数点不能在第一位");

digit.Clear();

return false;

}

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

{

if (digit[i] == '.')

pointCount++;

}

if (pointCount > 1)

{

MessageBox.Show("一个数不能有多个小数点");

digit.Clear();

return false;

}

if (curIndex >= 0)

curSelect += expr[curIndex].Length;//光标位置

curIndex++;//当前索引

expr.Insert(curIndex, digit.ToString());//写入表达式字符串

types.Insert(curIndex, type.digit);

showText();//刷新输入区

digit.Clear();//清空缓冲区

}

return true;

}

* + 1. 控制按钮

控制按钮主要包括删除按钮（DEL）、清空按钮（AC）、左移（←）、右移（→）、退出按钮（exit）、等于按钮（=）。清空按钮直接将所有数据清空即可、退出按钮直接调用Close()函数。主要是删除按钮和移动按钮，对于删除按钮来说，只需按照当前索引，删除表达式中相关单元即可。对于移动按钮来说，需要对光标位置和当前索引进行修改。等于按钮点击之后，先对各运算符单元的类型（前置一元运算符、后置一元运算符、二元运算符、左括号、右括号、数字）进行类型检查，如果检查通过再调用DLL进行计算。

删除按钮响应事件代码：

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

{

if (!checkDigit())

return;

if (curIndex >= 0)

{

expr.RemoveAt(curIndex);

types.RemoveAt(curIndex);

curIndex--;

if (curIndex >= 0)

curSelect -= expr[curIndex].Length;

}

showText();

}

表达式单元类型定义：

enum Type

{

beforeUnOp, afterUnOp, binOp, leftBracket, rightBracket, digit,

}

等于按钮响应事件：

先进行类型判断，再调用DLL计算。checkType()函数很长，这里只截取了判断括号是否匹配的部分。

private bool checkType()

{

int leftBracketCount = 0;

for (int i = 0; i < types.Count; i++)

{

if (types[i] == Type.leftBracket)

{

leftBracketCount++;

}

else if (types[i] == Type.rightBracket)

{

leftBracketCount--;

if (leftBracketCount < 0)

{

MessageBox.Show("表达式括号不匹配");

return false;

}

}

}

if (leftBracketCount != 0)

{

MessageBox.Show("表达式括号不匹配");

return false;

}

}

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

{

if(!checkDigit() || !checkType())

{

return;

}

else

{

Calculate.getExpr(expr);

Object result = Calculate.getValue();

resultTxtBox.Text = result.ToString();

}

}

移动按钮响应事件：

左移需要判断是否在最左端，右移需要判断是否在最右端。同时左移和右移都有考虑curIndex == -1的情况。

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

{

if (!checkDigit())

return;

if (curIndex > 0) //判断是否在最左端

{

curIndex--;

curSelect -= expr[curIndex].Length;

}

else if (curIndex == 0)

{

curIndex--;

curSelect = 0;

}

showText();

}

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

{

if (!checkDigit())

return;

if (curIndex < expr.Count - 1 && curIndex >= 0)//判断是否在最右端

{

curSelect += expr[curIndex].Length;

curIndex++;

}

else if (curIndex == -1 && expr.Count > 0)

{

curIndex++;

}

showText();

}

* + 1. 运算符按钮

运算符按钮点击之后，要先进行数字缓冲区判断，任何将相应的字符串加入表达式。另外对于加入文本和按钮显示文本不一致的需要单独写响应事件，例如平方按钮（“X^2”）加入的文本为“^2”而非“X^2”。

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

{

if (!checkDigit())

return;

if (sender is Button)

{

Button btn = (Button)sender;

if (curIndex >= 0)

curSelect += expr[curIndex].Length;

curIndex++;

string text = btn.Text;

expr.Insert(curIndex, text);

if (text == "Sin" || text == "Cos" || text == "Tan"

|| text == "Sinh" || text == "Cosh" || text == "Tanh"

|| text == "NOT")

types.Insert(curIndex, type.beforeUnOp);

else

types.Insert(curIndex, type.binOp);

showText();

}

}

* 1. 字符串表达式解析代码
     1. 类型定义

类型包括数值类型和运算符类型，数值类型包括整型、浮点型、布尔型，运算符类型包括一元运算符和二元运算符。布尔型参与运算时，True表示1，False表示0。对于运算符类型来说，要包含优先级。代码如下：

public class Number //自定义操作数类

{

public object value;//数值

public bool isDouble = true;//是否为double类型

public bool isBool = false;//是否为bool类型

public bool boolValue = false;//bool值

public void copy(Number number)//拷贝函数

{

value = number.value;

isDouble = number.isDouble;

isBool = number.isBool;

boolValue = number.boolValue;

}

}

public class Operator : Number//自定义运算符类

{

public string operatorType = null;//运算符

public int priority = 0;//优先级,越小越高

public bool isUnary = false;//是否为一元运算符

}

* + 1. 中缀转后缀以及求值计算

这一部分算法已经给出，具体代码比较冗长，需要注意的是，对于每一个给定的字符串，需要将字符串转化为对应的表达式单元。在参与运算时，先将每个数值转化为double类型，然后依据操作数和运算符来决定返回值的类型。为了提供更大的范围，数值类型包括double类型和long型。例如，加法运算对应的代码如下：

result = leftValue + rightValue;

@operator.isDouble = left.isDouble || right.isDouble;

return @operator.isDouble ? result : (long)result;

对于运算符，一元运算符和二元运算符的操作数个数不一样，故需要分成两个函数来进行计算。对于逻辑运算，例如OR，如果两个操作数都是布尔类型，那么OR为逻辑或，否则为按位或。

1. 实验结论与分析
   1. 实验结果

最后成功完成本次实验，能够成功解析字符串表达式并输出响应结果，对于表达式中的错误能够有所提示，同时也可以进行回退删除等操作。部分测试案例如下：

1. 输入(3+5)\*(6-7)^3+SinΠ，输出-8
2. 输入Sin(Π/2)\* Sinh(Π),输出11.5487393572577
3. 输入Sin(),提示表达式不能以前置一元运算符结尾
4. 输入Sin(SinΠ),输出1.22460635382238E-16
5. 输入SinSinΠ，提示解析出错

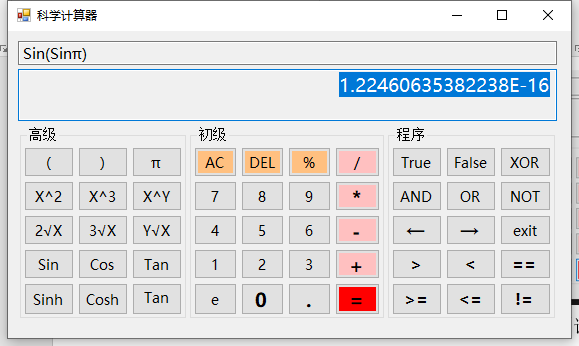


图4.1 正确测试案例

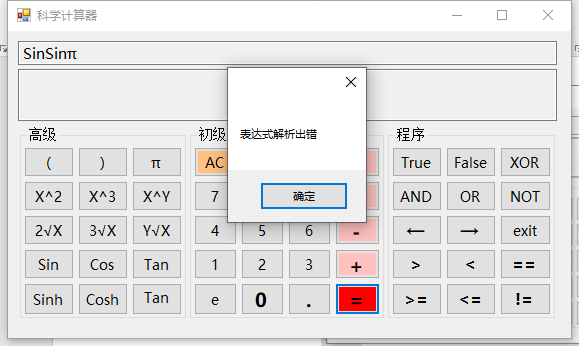


图4.2 错误测试案例

* 1. 个人收获

通过本次实验,熟练掌握了DLL的创建和使用，并对Winform技术的布局有了了解，学会了使用逆波兰表达式算法来解析字符串表达式进行计算。此外，在写代码的过程中，也经过了几次迭代，最终才确定使用逆波兰表达式算法。这对以后的开发起到了一个警示作用，在开发伊始，就要设计良好的开发方案，否则，在开发过程中需要全部重头来过是一件很耗时耗力的事情。

* 1. 不足与反思

功能不足：不能直接进行对输入区进行编辑，只能通过按钮进行输入，操作起来多有不便，日后可以改进为直接对输入区进行编辑和按钮输入相结合的方式，可以提高程序可用性。另外，有些表达式的支持度还不够，例如SinSinΠ这样的表达式，用户实际书写的时候是不会带括号的，但是本计算器不带括号无法正常进行解析，这也是有待改进的一个方面。还有一点，支持的运算符还不够多，目前只能进行数值计算，不能进行矩阵运算。

改进方向：对字符串表达式的解析可以参考编译原理的词法分析，这样就可以实现对输入区直接进行编辑的功能。对SinSinΠ的支持度不够，主要是逆波兰表达式的缺陷，需要加以改进。对于运算符的支持，很容易实现只需要添加相应按钮即可。对于矩阵运算，可以考虑使用Matlab和C#混合编程，利用Matlab编写相应矩阵运算函数，然后编译为dll文件，在C#中调用。

教师评语评分

评语：

评分：

评阅人：

年 月 日

（备注：对该实验报告给予优点和不足的评价，并给出百分之评分。）