Portfolio Assessment 1.7: Documentation

https://github.com/p466939/calculator

N Ringham [P466939]
SOUTH METRO TAFE | MURDOCH CAMPUS

Table of Contents

Introduction

In this technical document we shall describe the functioning of the Calculator application's code through pseudocode and the data used, and will make recommendations for upgrades and enhancements, and opportunities for more thorough testing before a full release.

Though the previous version, for assessment 1.6, required separate Visual Studio solutions be used to demonstrate separation between a project producing a DLL as output, and another project consuming it; in this release the projects have been combined into a single solution (still as separate projects) to use the built-in features that accelerate such development, such as building a dependency which has uncompiled source changes before and to be used by the program which depends on that library code.

I have also updated naming of variables and methods according to the code standards provided, however in some cases I feel trying to adhere to this has made the code less clear unfortunately.

Data Structures

Class CalculatorForm

Name	**Type**	**Purpose**
storedBinaryOpcode	enum(Opcodes)	Stores an opcode representing a binary operation (ie. operation taking two operands as arguments) for later evaluation/calculation
storedUnaryOpcode	enum(Opcodes)	Stores an opcode representing a unary operation (ie. taking a single operand as an argument) for later evaluation/calculation
storedNumber	decimal	Stores a value entered earlier or the intermediate result of a calculation which more operations may use
numericInput	NumericInput	Intermediate representation for a number being entered by the user digit by digit
editState	bool	Whether the calculator is in a state where it expects digits to be entered and appended to the numeric input (or if false, whether a result or error should be displayed)
errorState	bool	Whether the calculator's display should show an error message

Class NumericInput

Name	Type	Purpose
digitMaximum	int	The maximum digits that can be entered by the user (not including a decimal point or sign)
digitString	string	A string representing the digits (and decimal point if present) entered by the user
negative	bool	Whether the value stored is negative (not stored in the string)
nonInteger	bool	Whether a decimal point is present/has been entered
clear	bool	Whether the state has been cleared, and no entry has taken place following this

Algorithms

CalculatorForm methods

```
Constructor

    Perform WinForms initialisation as required

    register any additional event handlers we need (like for keyboard events)

    initialise our numeric entry with a clear operation


Method UpdateDisplay

    set the form to show either numeric entry,

    or the stored number (representing a result of a calculation),

    or an error message - based upon the state flags set


Method AddDigitDecimalPointOrToggleSign

    if not in the edit state:

        if the digit entered is a sign change apply it to the stored value displayed

        otherwise change to the edit state and accpt the digit for entry as below...

    call the relevant method of numericinput with the digit (0-9) or the sign change or
    the decimal point

    call UpdateDiplay to show the change


EventHandlerMethod for button press on a digit, decimal point or sign change

    call AddDigitDecimalPointOrToggleSign with the digit/character


Method PreformClear

    if the numeric input is diaplying an it's already been cleared, clear the entire state

        (ie. calling this twice is like "CE" functionality on calulators)


    clear the numeric input and set state flags accordingly

    call UpdateDiplay to show the change


EventHandlerMethod for Clear button press

    call PreformClear


Method QueryIfBinaryOperation

    return true if the opcode passed is for a boolean operation otherwise false


Method QueryIfBinaryOrEqualsOperation

    return true if the opcode passed is for a boolean operation or the equals operation,

    otherwise false
```

```
Method PerformUnaryOperation

    Call the unary operation method indicated by the opcode passed with an argument also
    passed

    return the result or the opcode argument if the operation was not unary/supported


Method PerformBinaryOperation

    Call the binary operation method indicated by the opcode passed with an arguments also
    passed

    return the result or the first opcode argument if the operation was not binary/supported


Method ProcessPendingOperations

    is called when the user clicked/presses an operation/opcode

    get the latest value entered from the numeric diaply or use the stored number if not used

        (as "input")

    clear the numeric input

    if the operation just selected is binary or equals:

        perform any unary operation pending with the "input"

        perform any binary operation with the above if present, otherwise the "input"

    otherwise (operation selected is unary):

        if we're not in the edit state ("input" is the stored value) then store the unary
        operation

            (it needs a number as input, note we potentially overwrite a previous unary
             operand)

        otherwise (we have a numer as input):

            perform the pending binary and unary operations stored as well as the new
            operation selected

            the operation is as  above, unless all three are present,
            in which case the order is:

                newUnary ( StoredBinary( stored number, storedUnary("input") ) )

    catch any exceptions raised, and:

        clear the calculator state

        set the error state


    set the input state to false (to show any result)

    call UpdateDiplay to show the change


EventHandlerMethod for any operation button press

    call ProcessPendingOperations
```

```
EventHandlerMethod for any operation keyboard
    mirror button press handlers by making the same calls to
        ProcessPendingOperations, PreformClear and AddDigitDecimalPointOrToggleSign
    based upon the key pressed

EventHandlerMethod for lowlevel keyboard event
    incept the enter key (otherwise it triggers button presses) and
    call ProcessPendingOperations treating the enter key as an equals operation
```

NumericInput methods

```
Method ToString

    override the builtin toString method combining any negative sign with the string stored


Method ToDecimal

    convert the string and negative state flags into a decimal value and return it


Method PerformClear

    clear the state - the digit string stored, and the flags on whether there is a

    decimal point, whether the value is negative, and set the clear flag as we have cleared


Method QueryIfClear

    return the clear flag value


Method AddDigit

    return false (failure indication) if the digit character passed wasn't recognised
    or if the digit string has the maximum number of digits already

    reset the clear flag

    append the digit to the string, unless it's just the digit "0" in which case replace it


Method AddDecimalPoint

    Check if there already is a decimal point (flag check),
    in which case return false to indicate failure

    reset the clear flag

    add a decimal point to the string, set a decimal point flag so we don't have to do a
    string search to determine this


Method ToggleSign

    reset the clear flag

    toggle the negative flag state
```

ArithmeticFunctions methods

```
Method CalculateAddition

    Add two numbers and throw an exception on error


Method CalculateSubtraction

    Subtract the second number from the first and throw an exception on error


Method CalculateMultiplication

    Multiply two numbers and throw an exception on error


Method CalculateDivision

    Divide the first number by the second and throw an exception on error
    (e.g. divide by zero)
```

AlgebraicFunctions methods

```
Method CalculateSquareRoot

    Preform a squareroot and throw an exception on error (ie. square root of negative value)


Method CalculateCubeRoot

    Preform a squareroot by taking the absolute value and using the Power function to raise
    the result to the power of a third

    Finally we set the sign of the result based upon the sign of the input and return it
    we throw an exception on error


Method CalculateInverse

    Calculate the multiplicative inverse by doing 1 divided by the number passed

    Throw an exception on error (ie. values at or close to 0)
```

TrigonometricFunctions methods

```
Method ConvertDegreesToRadians

    Convert an angle in degrees to radians by diving it by 180 and multiplying it by pi


Method CalculateSine

    Convert the input angle in degrees to radians using ConvertDegreesToRadians

    Use the builtin Sin() function to calculate the sine of the number

    Throw an exception on error


Method CalculateCosine

    Convert the input angle in degrees to radians using ConvertDegreesToRadians

    Use the builtin Cos() function to calculate the cosine of the number

    Throw an exception on error


Method CalculateTangent

    Convert the input angle in degrees to radians using ConvertDegreesToRadians

    Use the builtin Tan() function to calculate the tangent of the number

    Throw an exception on error, or if the magnitude of the result is above some
    threshold value
```

Recommended testing procedure

I feel the test tables the software has already been tested against was too simplistic, and I would wish to add tests with sequences of many operations chained together. I would also like to see unit testing added to more thoroughly test the software, and particularly prevent bug being introduced in future changes.

I would also like to procure real calculators which the software is attempting to mimic, to see how they might function slightly differently, and how lessons in there design and even shortcomings can be incorporated into the software to make it feel more like those.

Recommendations on upgrades and future enhancements

The software should format numeric results better for display so as not to fill the screen with many decimals places that do not matter (and may be slightly off what they expect after many accurate digits, as was found in testing), and code should be added to deal with large numbers that are too big to fit in the display field so they are not mistaken for numbers of smaller magnitude.

I think using decimal (instead of a binary floating point format) was a step in the right direction for developing a calculator that seems intuitively correct to humans used to working in base 10, but not enough is done to make use of this format in the calculation of trigonometric functions or roots. While it might make sense to switch to root finding using decimals, rather than floating point libraries, I think for trigonometric functions it might make sense to use modulo arithmetic on the decimal number to bring it into the +/- 180 degrees / pi radians range before conversion to floating point format for the highest precision.

I also feel if we offer sine, cosine and tangent functions, we should offer the inverses of these as a user is likely to want all or none of these, offering just sine, cosine and tangent seems incomplete.

I would also like the display to provide more user feedback, perhaps flashing briefly calculating answers, and better indicating the internal state. I can’t put my finger on it exactly, but using the calculator does not feel good from a UX perspective, and when an answer is displayed one may question or be confused as to whether this was just the value that they entered, especially in cases whether the input and output appeared the same such as when testing with the sine of 0 which gives 0 as the answer.