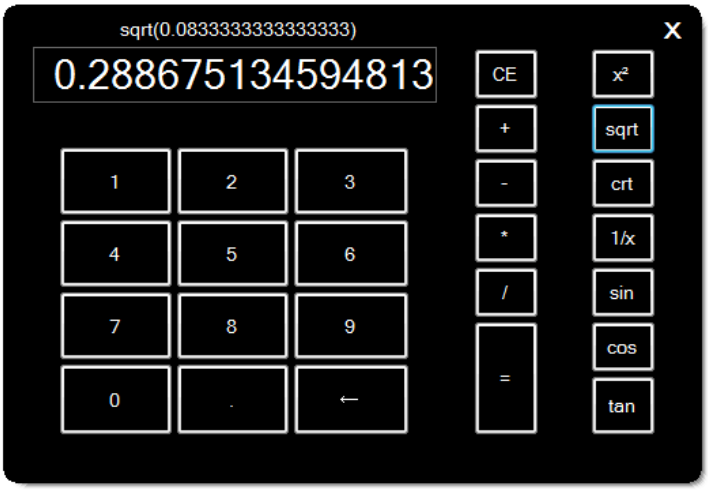
Desktop calculator with advanced functions

Version 1.0 – 27 December 2016

Development report



Data structures

Name	Data Type	Purpose
keyDict	Dictionary<string, Keys>	links key names to keys
keysToButtons	Dictionary<Keys, CalcButton>	links keys to buttons
tempString	string	temporary storage of the value entered by user as string
previousValue	double	previous value entered by user as double
currentValue	double	current value entered by user as double
currentFunc	Func<double, double, double>	arithmetical operation being currently performed
arOps	Dictionary<String, Func<double, double, double>>	links button text to the corresponding arithmetical operation
advOps	Dictionary<String, Func<double, double>>	links button text to the corresponding advanced operation
arOpsList	List<string>	list of arithmetical buttons
advOpList	List<string>	list of advanced operation buttons
newCalcStarted	Bool	has the user started a new calculation?

Algorithms

Populate numeric key panel

```
counter = 1
for each row in panel.rows
{
    For each column in panel.columns
    {
        button_text = counter % 10
        button_key = key_dictionary[button_text]
        create button and assign NumPad key to it
        add button to panel
        counter++
        if (counter > 10)
            break
    }
}
```

Add callback to each numeric key

```
For each button on the numeric key panel
{
    add onClick callback
    {
        if new_calculation == true
         // previous calculation has been completed
        {
            Erase calculation history
            Set output textbox text to key text
            new_calculation = false
        }
        else
        {
            if (output_textbox_text == "0")
                // the user has just pressed an arithmetical operation key
                // such as “+” or “*”
            {
                output_textbox_text = key_text
            }
            else
         // concatenate key text with textbox text and check if it’s a
         // valid double. If yes, update textbox text, otherwise ignore
            {
                tempString = output_textbox_text + key_text
                if isValidDouble(tempString)
                {
                    output_textbox_text = tempString
                }
            }

        }

    };
```

Add buttons for arithmetical operations and create callbacks

```
// List of arithmetical operations: add, subtract, multiply, divide

arOpsList = [“+”,”-“,”*”,”/”]

            for each operation in arOpsList
            {
                Create new button and assign function and NumPad key to it
                Add button to arithmetic_operations_panel
                button.Text = operation
                 // Add callback
                // The minus button is to be handled differently from the others to
                // allow entry of negative numbers
                if (operation == "-")
                {
                    Add callback
                    {
                        // user entering negative number
                        if (output_textbox.Text == "0" || new_calculation)
                        {
                            if new_calculation: erase calculation history
                            output_textbox.Text = "-"
                            new_calculation = false;
                        }
                        else if (isValidDouble(output_textbox.Text))
                        {
                            current_function = subtract
                        }

                    };
                }
                else
                {
                    Add callback
{                    {
                        if (isValidDouble(output_textbox.Text))
                        {
                            current_function = button.function
                        }
                    };
                }
```

There are no error-handling techniques used within these algorithms as they do not involve user interaction and the program flow is controlled with if/else statements.

Other algorithms within the Calculator project are either very similar or self-explanatory.

MyMathLib library:

Sin

```
Sin(input_value)

// since the input_value is expected to be in degrees, it needs to be converted to radians
// first to make use of the built-in Math.Sin function:

       radians = input_value * pi / 180
```

```
// the following operations are required to prevent floating-point imprecision so
// that e.g. sin(0) is equal 0
output = math.sin(radians)
round output to 8 decimals
inverse output
inverse output again and round to 8 decimals
return output
```

Other trigonometric functions are very similar to the sin functions and other arithmetical functions are self-explanatory.

Recommended testing procedures

Prior to the commercial release, it is recommended to test all functions on the calculator to make sure that they produce valid outputs. Both clicking on the calculator buttons with the mouse and using the corresponding NumPad keys needs to be tested (note, however, that the advanced function buttons are not linked to NumPad keys). In particular, the following values and processes will need to be tested:

- Division by zero – should produce infinity (Inf) as the answer. The same value should be produced as output of tangent of 90, 270 etc. degrees
- Floating-point arithmetic: 1 divided by 3 and then multiplied by 3 should produce 1
- Negative numbers need to be handled correctly
- Decimal values
- Square and cubic roots of negative numbers should produce NaN (not a number) as the calculator does not handle complex numbers
- Very small and very large numbers should be displayed using scientific notation
- Invalid user input such as attempting to enter more than one decimal point

Suggested future enhancements

- Adding further advanced operations such as exponentiation and logarithms
- Adding functionality to perform calculations involving complex numbers
- Providing an option to store longer operation history in memory – currently the calculator stores and displays only the most recent operation