Visual Calculator Documentation

Contents

1 Brief

This documentation follows the design and development of a c++ calculator, created using the visual studio programming environment. This calculator must use functions of a common scientific calculator would use and as such the interface which would also have to reflect this.

Research was conducted to find what interface would best fit this style of calculator, one found was calculators on mobile devices however not all phones use scientific interfaces. So, the research found that windows calculators would fit the purpose for this calculator and as such the UI was created with the windows calculator in mind. The calculator was built with the purpose of being very low maintenance meaning that it would use as little memory as possible, decreasing load times and improving input lag.

Lastly the calculator was developed with business logic in mind making sure that calculations were separate from the UI functions to properly use the object orientated environment. This means the 2 header files will need to constantly talk, the UI will be mainly to output results and the functions will store calculations needed by the calculator.

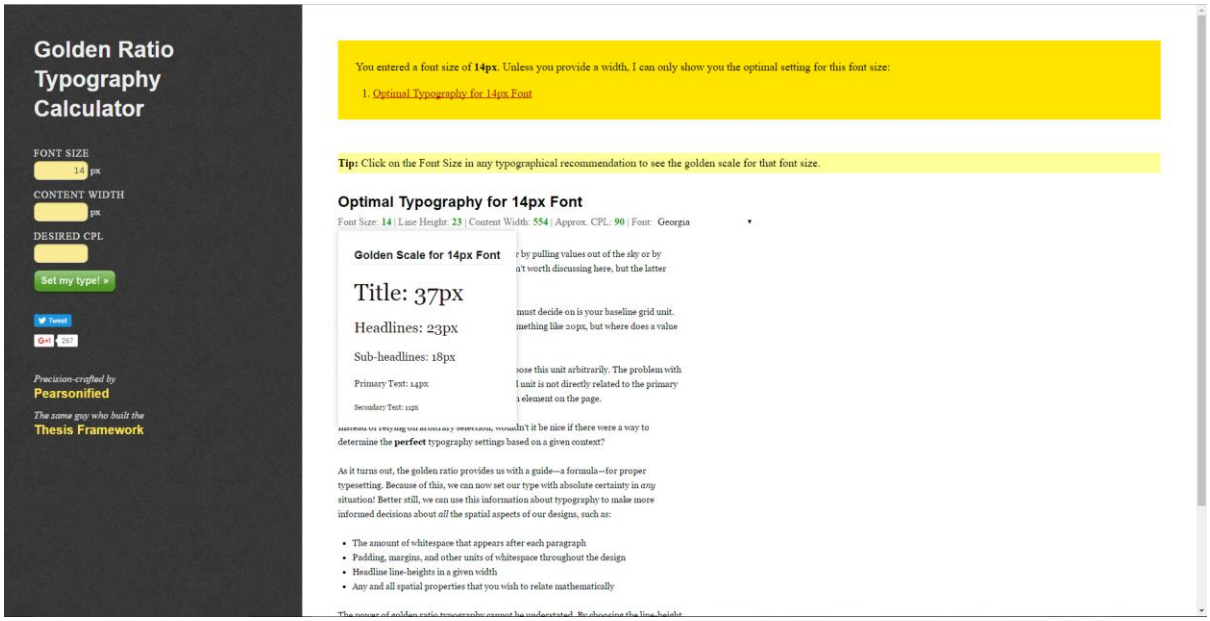
2 Development Process

Before starting the development, process research would need to be undertaken to ensure that structure, terminology is correct and that its understood. 2 key resources inspired how the calculator works the first being the biggest inspiration from (McGrath, 2011) this book went through how to work with a IDE environment, object orientated programming and setting up windows form for UI integration. The second piece of inspiration came from online videos, tutorials in how to set up a visual studio project correctly and how to output text to a textbox (Youtube, 2013).

2.1 UI Development

The UI was developed with user feedback in mind meaning that each click will inform the user of what that function has done. To implement this the calculator will consist of two text boxes, one to show the calculations output and the second to show the operation character such as the equals character. The feature works similarly to that of the windows function where the whole calculation string will be shown above the output but for this calculator it was decided just to show the operation character to save memory for the system.

The rest of the UI buttons were design to be big, this is to make the buttons easier to see and click on for the user, it also allows the calculator to be ported to other devices such as a mobile phone that uses touch input. The text used throughout the calculator was design using the font golden ratio, ensuring that text sizes are the optimal size for people with poor eyesight.



2.2 Business Logic

```
class Functions {
public:
        void Equals()
        {
            switch (op)
            {
            case '+':
                result = num1 + num2; //Work
                break;
            case '-':
                result = num1 - num2; //works
                break;
            case'/':
                result = num1 / num2; //works
                break;
            case'x':
                result = num1 * num2; //Works
                break;
            case '%':
                result = num1 / 100;
                break;
            case '&':
                result = (int)num1 % (int)num2;
                break;
            case 'p':
                result = System::Math::Pow(num1, num2);
                break;
            default:
                break;

            };
        }
```

Originally the calculator was not built with the business logic system implemented. This created problem with the calculator and made the code look messy as the calculations and outputs were coded in large functions. This was then rectified by creating a separate header file named functions, this functions file would then contain a case system that will look for a variable that contained a character, if the character matched that of the class systems character the case will then execute that function then break out of the loop.

This was a much cleaning and more efficient way of letting the UI talk to the calculations. After this test was successful all calculations were moved over into the functions header and any outputs made to the text box were left inside the UI header. Some calculations are not made within the case statement but instead have their own functions that are then called in the UI header, this allows the code to stay neat and if more code needs to be added it can be done via the function inside the functions header. A diagram has been made bellow to show how the business logic works.

```
#pragma once
#include "Functions.h"
```

Finds the function header file.

The op variable then takes in a char as soon as the subtract button is pressed, in this case a minus symbol.

*UI Header File

```
rivate: System::Void ButtonSubtract_Click(System::
    if (num1 != 0)
    {
        num2 = (float)System::Convert::ToDouble(Out
        func.Equals();
        op = '-';
        this->Output->Text = Convert::ToString(resu
        OperatorShow->Text = "-";
        num1 = (float)Convert::ToDouble(Output->Tex
        check = true;

    }
```

```
    void Equals()
    {
        switch (op)
        {
        case '+':
            result = num1 + num2;
            break;
        case '-':
            result = num1 - num2;
            break;
```

The Equals case will then be called from the UI header file and provide a char variable once a button is pressed. The case will then keep looping through until the op variable equals one of the op chars and then breaks out and provides the UI with the calculation.

*Functions Header File

calculator could only manage 2 inputs at once and not a large string like on the windows calculator, the calculator could only accept 2 + 3 and not 2 + 3 + 5, as it would still give 5. To work around this the calculator used a similar technique as the windows calculator which was to take variable 1 and 2 and equal them once a third button was pressed and from then onward one value would take the equalled amount and another would hold the new input value from the user.

```
private: System::Void ButtonSubtract_Click(System::O
    if (num1 != 0)
    {
        num2 = (float)System::Convert::ToDouble(Outp
        func.Equals();
        op = '-';
        this->Output->Text = Convert::ToString(resul
        OperatorShow->Text = "-";
        num1 = (float)Convert::ToDouble(Output->Text
        check = true;

    }
    else
    {
        num1 = (float)System::Convert::ToDouble(Outp
        this->Output->Text = "";
        OperatorShow->Text = "+";
        op = '-';
    }
}
```

Another issue that was encountered later in the calculator's development was that if any of the arithmetic buttons were pressed more than once this would cause the calculator to break. This wasn't fixed in the test build but to fix this issue the calculator would need to always equal 0 this is to ensure that the variables hold a value as the buttons can only calculate with variables stored to them.

Another smaller issue is that it was originally planned to have all the calculations on one header file, however some of the function will be created in the UI as they only need to output a number to the textbox, this could have been managed using the functions file it shows that this will decrease optimization in the program. For examples like this one they are being created in the UI header file and the heavy calculations are managed by the functions file.

3 Evaluation

The program is now complete, Ui has been constructed similarly to that of the windows version and developed using business logic. Buttons 0 to 9 allows the user to implement larges numbers such as 9999 and can also take in decimal point values such as 2.1, no errors are given when using these functions. Cin, Cos, Tan, Log, LogIn all work allowing the user to enter a number and then once equals is pressed the answer is outputted. Exp has an issue in that it can only work with one any other numbers will not output the correct number. String numbers now work and can take an infinite amount of numbers and still output the correct output e.g. 5 + 4 + 3 + 2 will now equal 14.

3.1 Black Box Testing

Test ID	Description	Expected Result	Calculators Result	Did it work?	Any fixed errors?

A1	Numbers Test Press 2 Press 3 Press 1 Press 0	2310	2310	Y	N
A2	Input 5 Add 3 Add 2	10	8	N	Y, Calculator now equals then adds to the first variable.
A3	Cos Test Press 9 Press 0 Press Cos	-0.44	-0.44	Y	N
A4	Sin Test Press 9 Press 0 Press Sin	0.89	0.89	Y	N
A5	Log Test Press 9 Press 0 Press Log	1.95	1.95	Y	N
A6	In Test Press 9 Press 0 Press In	4.49	4.49	Y	N
A7	Exp Test Press 1 Press Exp	2.71	2.71	Y	N
A8	Tan Test Press 9 Press 0 Press Tan	-1.95	-1.95	Y	N
A9	Square root Press 5 Press Sqrt	2.23	2.23	Y	N
A10	Press Pi	3.14	3.14	Y	N
A11	Press 5 Press times Press 3	15	15	Y	N
A12	Press 5 Press Divide Press 3	1.6	1.6	Y	N

A13	Press 5 Press Subtract Press 3	2	2	Y	N
A14	Press 5 Press Add Press 3	8	8	Y	N
A15	Press 5 Press X^Y Press 3	125	125	Y	N
A16	Press 2 Press . Press 1	2.1	2 Then error	N	Y, needed to convert to double rather then int.
A17	Press 5 Press X!	120	120	Y	N
A18	Press 5 Press %	0.05	0.05	Y	N

Most of the buttons worked when running through the black box test as some of the buttons used the C++ Math function that would take a variable and convert into said Math output like Cin. Other buttons just use a given equation to then output the right answer like percent var1 / 100, this allowed most of the buttons to work instantly.

The calculator works similarly to any digital one such as windows and googles, some tweaks need to be made such as the EXP button issue but over 95% of the calculator works well without bugs and receives the right answer. Overall if these minor issues were fixed this calculator could be released to the mass market to use on a day to day basis.

4 References

McGrath, M., 2011. *C++ Programming.* 4th ed. Southham: In Easy Steps Limited.

Youtube, 2013. *Visual C++ Calculator Tutorial.* [Online]
Available at: https://www.youtube.com/watch?v=zBBe9oKU7jk
[Accessed 2 September 2016].