Technical University of Moldova

CIM Faculty

FAF

Report

Integrated Development Environments

Laboratory work #2

Done by: student, gr. FAF-151

Bîzdîga Stanislav

Verified by: Patrașco Alex

Chișinău - 2017

Topic: GUI development

Prerequisites:

- IDEs: Visual Studio, **QTCreator**, Xcode, Code::Blocks
- Languages: **C/C++**, C#, Objective C, Python or others (depends on IDE)
- Technologies and Frameworks: Forms, wxWidgets, Win32 API, WinRT API, PyQt or others (depends on IDE, chosen language and target GUI), ex: **QtWidgets**

Objectives:

- Make a simple GUI Calculator
- Default operations are: +,-,*,/,power,sqrt, Sign Inversion(+/-),operation with decimal floating point.
- Splitting code into two modules: User Interface and Business Logic.

Tasks:

Mandatory tasks:

Basic (mark = 5):

- Make a simple GUI calculator with basic operations such as + , - , / , *

Optional tasks:

Intermediate(each task is worth .5 points):

- Provide support for the following operations in your application: power, sqrt, Sign-Inversion(+/-).
- Provide support for operations with decimal floating point numbers.
- Your program must contain two modules, the GUI module and Business Logic module.

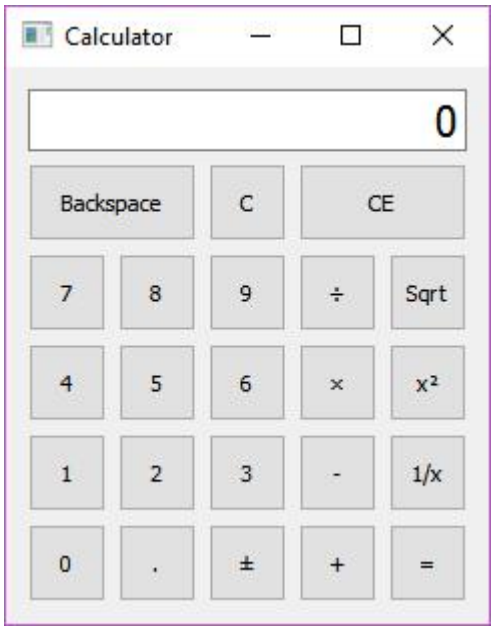
Advanced (each task is worth 1.5 point):

- Work in groups of two(Business Logic & GUI) on a common repo using merge requests.
- Extract Business Logic module into a library, that can be later used separately, either by another GUI module or from console.
- Provide a possibility for user to switch between several GUI modules(2+).
- Plot a graph if user input is a function.

Work processing

To create a GUI calculator I chose an IDE that is appropriate for this task: QTCreator.
Even though I didn't ever use it before, I could get the hang of it within a couple of hours. It is a comfortable, user-friendly IDE that allows easy project management and fast code editing, due features like auto-complete and intuitive icons and tools.

Product description:



This is my simple GUI calculator.
These are all the features it has:

It may calculate simple operations using plus, minus, multiply and divide operators.

It may operate with decimal floating point numbers.

It may perform operations such as sign inversion, square root , power and reciprocal.

It can also handle the error exception of zero division, and the negative number square root exception.

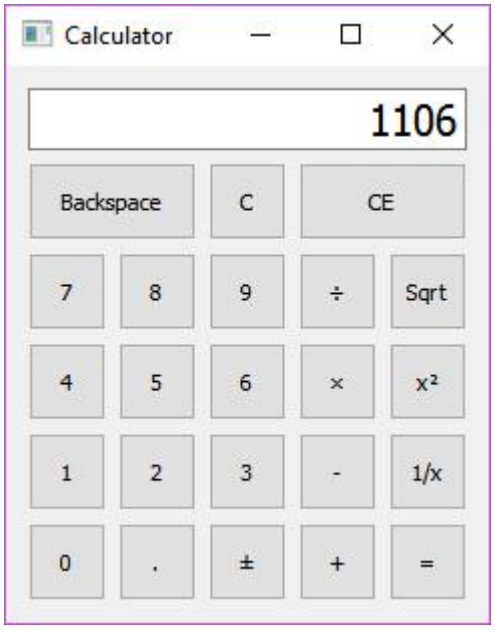
It may also delete the last input digit until cleared (backspace), clear all number(C), reset calculator (CE).

Mandatory tasks:

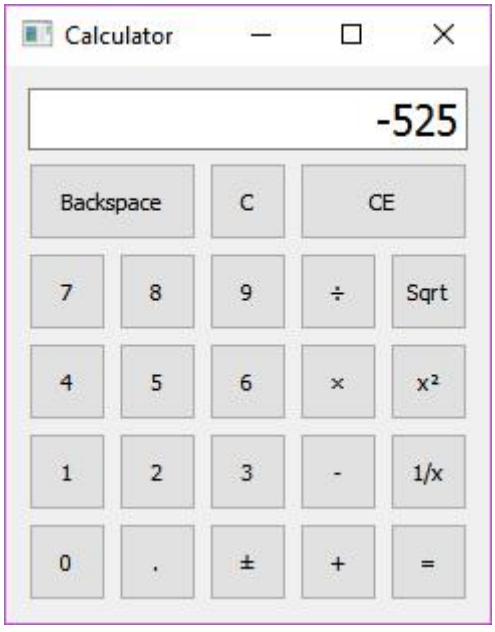
Basic:

1) Make a simple GUI calculator with basic operations such as + , - , / , *

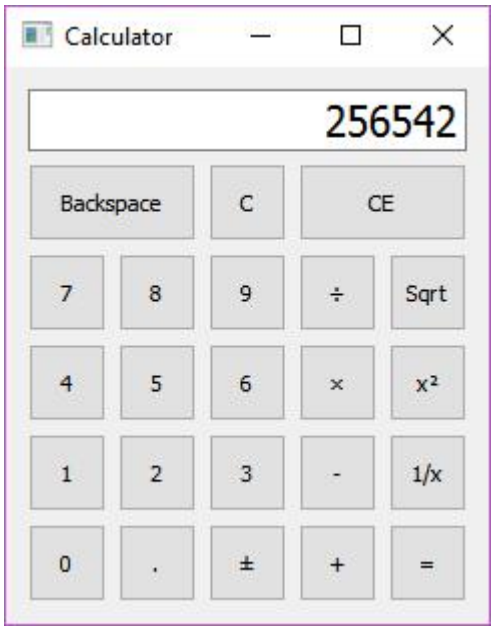
Input: 2 4 3 + 8 6 3 =



Input: 9 6 1 - 1 4 8 6 =

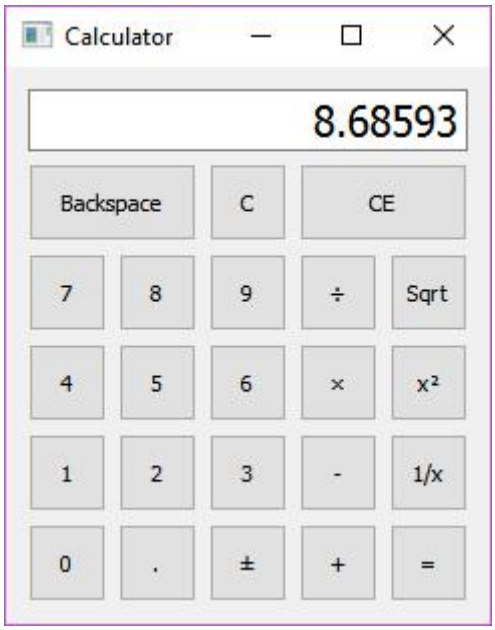


Input: 3 3 * 7 7 7 4 =



Note that it is optimized to show only 6 digits(for both before and after decimal point), a bigger number would be shown in a scientific notation (i.e. num e+n)

Input: 3 4 5 7 / 3 9 8 =



Note that windows calculator gives the following result: 8.685929648241206, my calculator approximates at the 6th digit.

Notes regarding the task:

In order to implement the above calculations I've taken the following approach:

First, I need the digit input handling. For that I needed a boolean variable to keep the information regarding if it is waiting for an operand. It is useful to have because initially it is true, so I input my first digit clearing the output, and by doing that it turns false, thus I can keep adding digits to my number. Then, once I click an operator, it turns true, thus I can input my next number.

Once I had it set up, then I could proceed to the calculation function. I decided to have it reading what kind of operator(+, -, *, /) it receives and output different results. (Actually modifying a static variable that holds the result.) It is a simple 4-case switch statement that uses += , -=, *=, /= (with operand) to change the result.

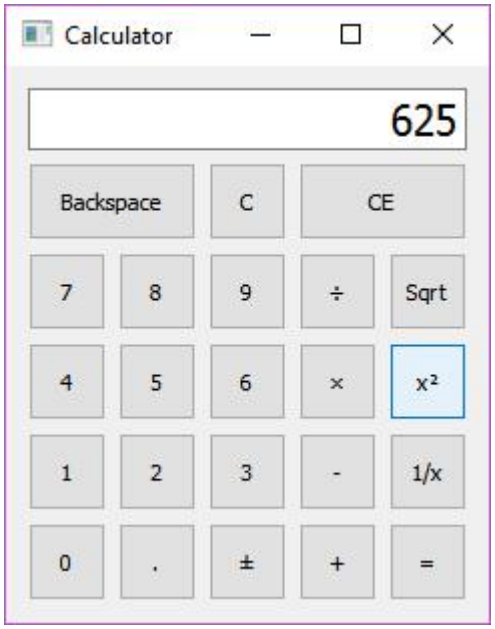
Before that, however, I needed to handle the operator clicking, so I made functions that receive the operand and the clicked button type. These functions also include the error handling.

Optional tasks

Intermediate (0.5 each):

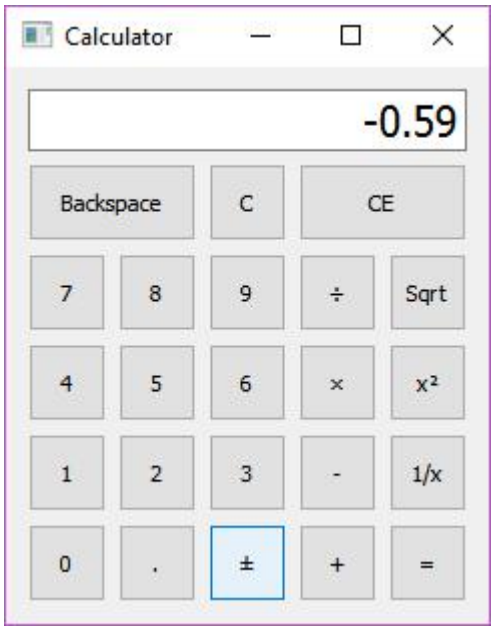
1) Provide support for the following operations in your application: power, sqrt, Sign-Inversion(+/-).
2) Provide support for operations with decimal floating point numbers.

Input: 2 5 pow



Input: 2 5 7 sqrt

Input: 0 . 5 9 Inv



Input: 0 . 9 9 6 + 0 . 0 0 0 1 = * 2 =

Commentary:

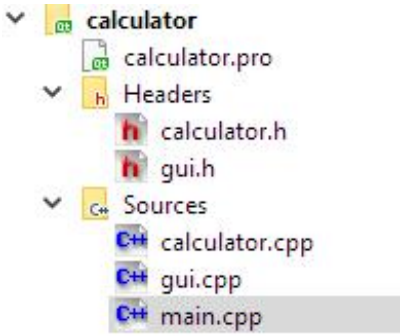
1) These features have been implemented with the help of <cmath> library.
I wrote a function to handle these features as unary operators. (using only 1 operand)
Thus by having a number on output screen, the function checks which button was clicked and performs the required case.

2) For floating point arithmetic to work, I simply always use “double” variables. However, to input a decimal point number things are more interesting.
I created a function to handle the point button click, which first of all checks if the number on screen has a point, and if it doesn’t, it adds one, and sets the wait for operand to false, thus enabling the user to input a decimal part.

3) Your program must contain two modules, the GUI module and Business Logic module.

This was the trickiest part.

My project tree:



The Business Logic is contained within calculator.cpp file.
It is composed by a handful of functions that are easy to access externally, thus making the logic usable by multiple GUI types if necessary.
The GUI module is found in gui.cpp file.
It takes care of all the GUI, the button creation, the button clicks, the function calls(from business logic) and output.

Advanced (1.5 each):

1) Extract Business Logic module into a library, that can be later used separately, either by another GUI module or from console.

The fact that I made the calculating functions apart of the gui allows its use with different gui or console. The only downside is that it might be tricky to use, since one must know the functions.

It is not very intuitive. one must call the functions using a number and an operator. It is like using a real calculator in hand, one should have that in mind.

So, for example to calculate a simple addition of x and y, I’ll need to call:

```
#include<calculator.cpp>

double result, x, y;                          // symbolic:
additiveOperatorClicked(x,”+”);               // buffer = x +
result = additiveOperatorClicked(y,”+”);      // buffer = x + y +
clearAll();                                   // buffer = 0
```

Conclusion

In the current laboratory work, I've studied a lot about GUI programming. It provided me a better understanding of what is and how should one approach GUI.

Also, I learned about modularization and how to do it, i.e. split code into modules that are independent of one another and re-usable by other code modules if necessary.

It is a great concept but also very challenging to perform without the proper experience level.

My repository: https://github.com/StasBizdiga/MIDPS

Table of contents

Introduction 1

Prerequisites 1

Objectives 1

Tasks 1

Work Processing 2

Product description 2

Mandatory tasks 2

Notes regarding the task 3

Optional tasks 4

Advanced tasks 5

Conclusion 6