May 2021, Warsaw

Internet Techniques - Semester Project Report

Student:
Emirhan Caliskan (56140)

To be assessed by:
Ms. Weronika Wiszniewska

Project:
Calculator

1. Design

This paper is an introductory report of a software product developed as a part of an academic task required by the Internet Techniques course. The task is to create a simple calculator which can be used to perform certain mathematical operations between two numbers.

1.1 Operators

For starters, the program is needed to be able to take two numbers as input from the user, perform the desired operation on/between numbers and lastly display the result. A list of operators this program offers is provided with descriptions in ***Table 1*** below.

Operator	Description
+	add $secondValue$ to the $firstValue$
-	subtract $secondValue$ from the $firstValue$
*	multiply $firstValue$ by $secondValue$
/	divide $firstValue$ to $secondValue$
!	compute factorial of $firstValue$ and $secondValue$
^	take$secondValue^{th}$ power of $firstValue$
%	divide $firstValue$ to $secondValue$, return remainder
log	compute base $secondValue$ logarithm of $firstValue$
lcm	find lcm of $firstValue$ and $secondValue$
gcd	find gcd of $firstValue$ and $secondValue$

Table 1: Operators with descriptions

1.2 Buttons

In addition to operators, the program needs a set of buttons, each with their own functionalities, to perform certain actions.

Firstly, all operators are represented by a button.Once they are clicked the program will register an operator and corresponding actions will take place.

Furthermore, there are other buttons that add various features to the program. A complete list of buttons used in this program alongside their functionalities is provided below in ***Table 2***.

Button	**Action**
Calculate	Initiates calculation procedure, signals various checks, triggers displaying of either results or appropriate errors
Clear	Clears all fields and variables, including operator display, result display, input text boxes.
Exit	Quits from the program at once
Swap	Swaps values between input text boxes

Table 2: Buttons and actions they trigger

The use of a calculate button seemingly delays completing a cycle in using the program. Once input fields are filled, the role of the calculate button could have been delegated to operator buttons. This would have been simpler but also would have made using the program faster possible. The reason a calculate button was exclusively created was that, in simple programs like a calculator of this kind, it provides the user with a satisfactory feeling and a more intuitive sense when triggering the last action (displaying the output) with a final click. In more complicated GUIs where a window is so much more filled with elements, this kind of eliminations and shortcuts should be opted as far as not to make the experience frustrating. However, this is clearly not the case here.

1.3 Visuals

The developed product is a website. ***Figure 1*** below represents how the website looks like on Google Chrome.

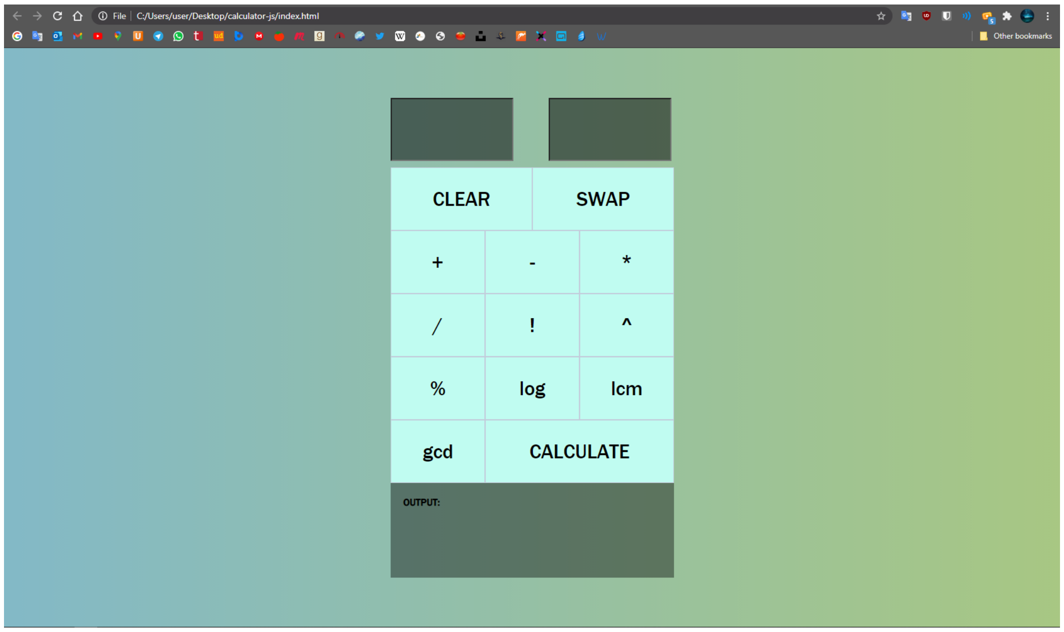


Figure 1: Website design

It is worth to note that, the label in between the input text boxes displays the operator once an operator was chosen via clicking an operator button. This particular feature does not necessarily add any functionality. Meaning that, the program works completely fine in the exact same way without it. However, it is simply nice to have it since it provides the user with a visual feedback after an operator was selected. Also, this label was placed in between the text boxes to allow the user to see the whole structure of the operation side by side much like how they would see it on paper doing math.

Labels are great for displaying certain information. But, in addition to this, action scripts were written to perform certain tasks on these labels. For example, the clear button clears all fields. Should the user want to only clear the operator display label and reset operator choice, they may do so by simply clicking on this label. As well as this, when the “OUTPUT:” label is clicked, the result label just below this text will be cleared.

2. Algorithms

There are three essential algorithms to be explained. Firstly, *calculateClick* is the main algorithm that decides what happens as the user clicks the calculate button. There are essentially two outcomes: (1) register the operator - make the calculation - and display the result (2) further investigate the situation and decide either to display an appropriate error or activate *handleFactorial*.

Normally, it is enough to decide if both inputs are valid (= double, non-empty) and either return an error or calculation results. However, not all operations in this program require two elements. For example, for division (/) to be possible requires a dividend and a divisor but for factorial to be possible only requires a single positive integer to compute its factorial. Factorial operator is in fact the single operator with this situation in this program.Therefore, it is not possible to simply return empty field error if there is only one input. The program needs to be able to work with one input in case of factorial operator. to handle this situation, *handleFactorial* procedure is created and *clickCalculate* decision making is a little more complex than what one would expect.

Figure 2 below demonstrates how *clickCalculate* action works. Then, ***Figure 3*** below shows how operations are registered and calculations happen. Lastly, ***Figure 4*** below presents how invalid input errors are handled. These flowcharts provide a good, high-level understanding of the program’s structure but for more specific information one has to refer to the source code.

2.1 Click calculate action

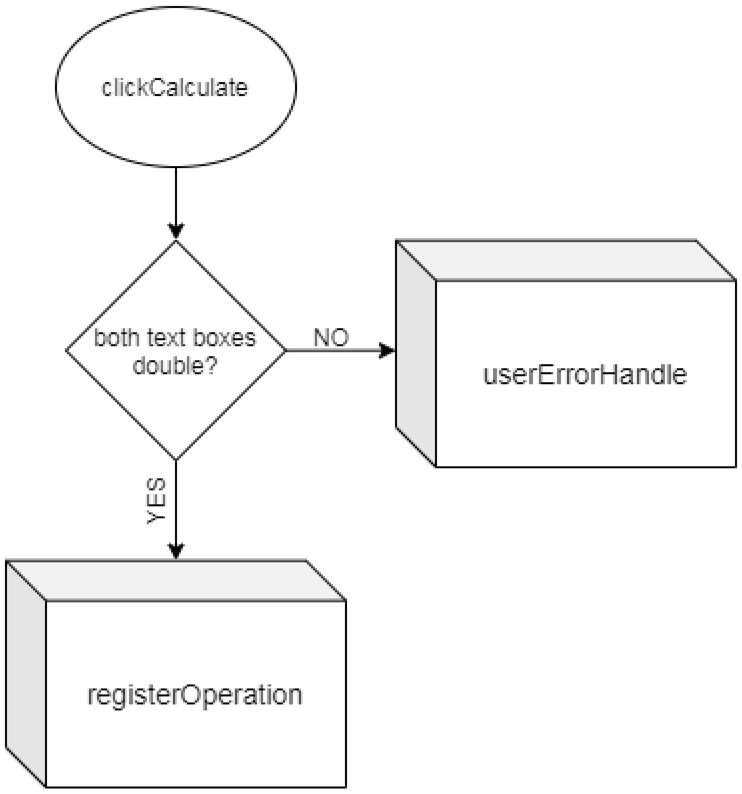


Figure 2: *clickCalculate* flowchart

2.2 Registering calculations

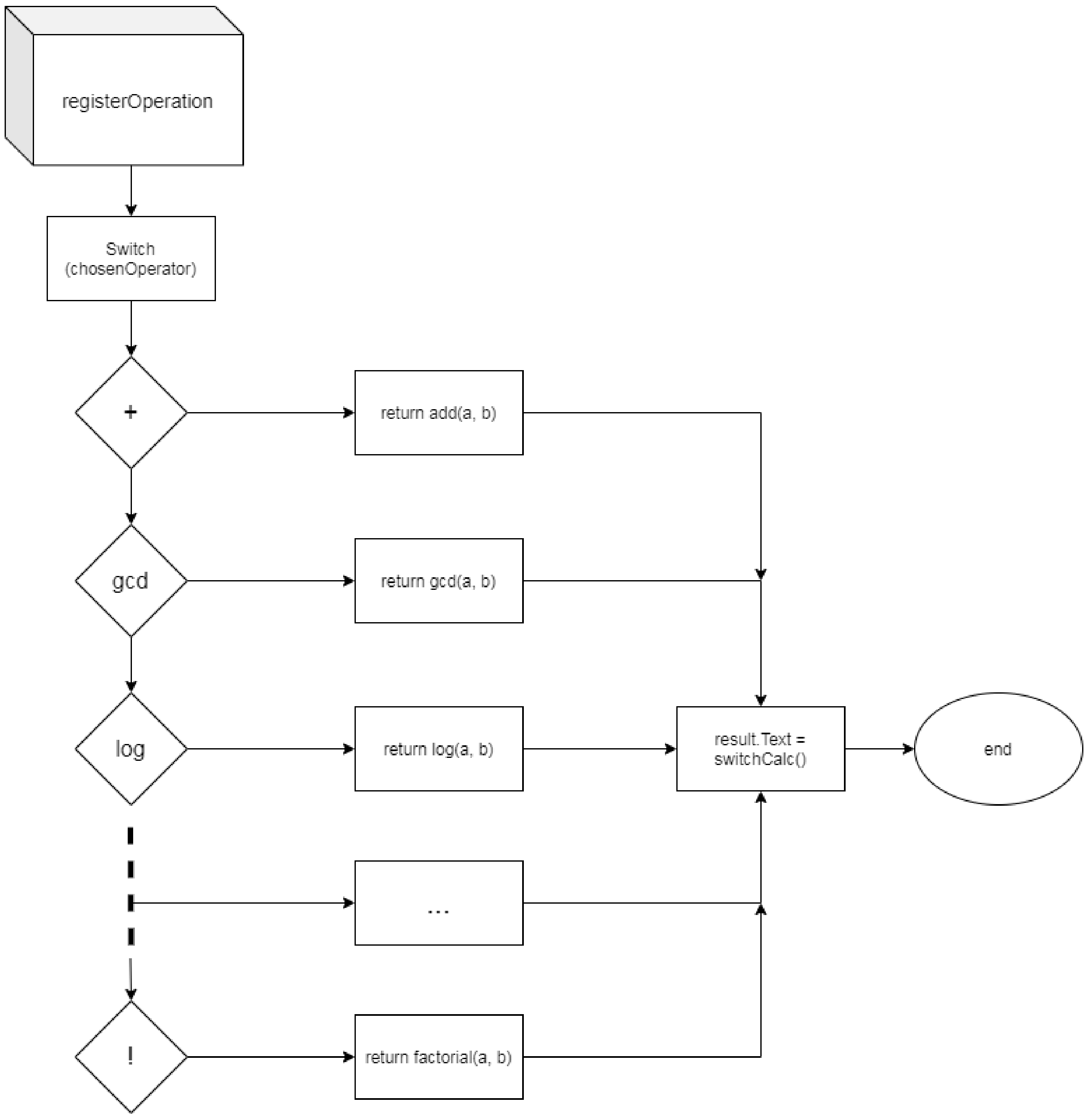
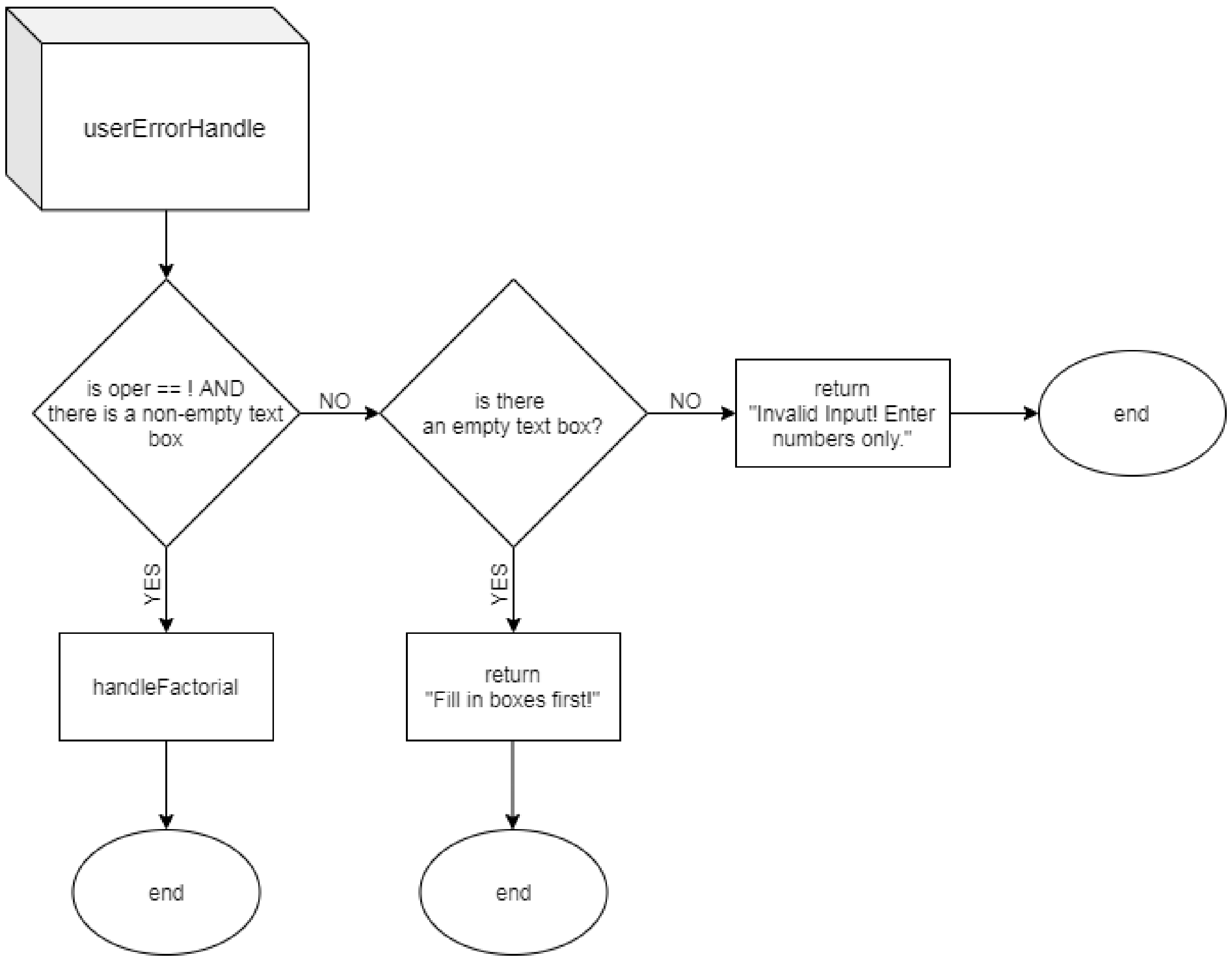


Figure 3: *registerOperation* flowchart

2.3 Handling user input errors



userErrorHandle
is oper == ! AND there is a non-empty text box
NO
is there an empty text box?
NO
return "Invalid Input! Enter numbers only."
end
YES
handleFactorial
end
YES
return "Fill in boxes first!"
end

Figure 4: *userErrorHandle* flowchart

3. Testing

3.1 Different cases

There are various cases where output behaviour may differ. These cases are generally related either to user input errors or mathematical nature of the operator selected. In ***Table 3*** below, some special notes where errors need to be displayed are shown.

Interaction	Conditions	Error
input text boxes	both inputs are empty OR a input is empty and chosen operator is not factorial	User input error
input text boxes	both inputs are not double OR an input is empty and chosen operator is not factorial	User input error
factorial calculate (factorial operation can only be applied on positive integers; there is yet no universally accepted definition in combinatorics for different cases)	chosen operator is factorial, one empty one non-int or negative input OR chosen operator is factorial, both non-int or negative input	Mathematical definition error
factorial calculate (ulong can store up to result of 170!)	chosen operator is factorial, request is bigger than 170	Technical restraint error

division calculate OR modulus calculate (division by zero is undefined)	chosen operator is division or modulus and second value (divisor) is zero	Mathematical definition error
logarithm calculate (argument can not be zero)	chosen operator is logarithm and first value (argument) is zero	Mathematical definition error
logarithm calculate (base or argument can not be negative)	chosen operator is logarithm and either or both (base and argument) is/are negative	Mathematical definition error

Table 3: Specific cases where errors need to be returned

3.2 Test results

The web app was written on Visual Studio Code 1.56.2 formatted with VS Code extension *prettier* and tested to be working bug-free on Chrome 90.0.4430.212 x64 on Windows 10. All testing was made through VS Code extension *live server* as well as by viewing locally. How the program reacts in different cases including ones from ***Table 3*** is presented in ***Table 4*** below.

INPUT	OUTPUT
firstValue: 234.45 secondValue: 453 oper: +	234.45 + 453 = 687.45
firstValue: 2456.3 secondValue: -564.3 oper: -	2456.3 - (-564.3) = 3020.6

firstValue: 34.5 secondValue: 437 oper: *	34.5 * 437 = 15076.5
firstValue: 84.3 secondValue: 95.7 oper: /	84.3 / 95.7 = 0.880877742946708
firstValue: 1345 secondValue: 0 oper: /	Mathematically undefined request: Can’t divide by zero
firstValue: 12 secondValue: 8 oper: !	12! = 479001600 8! = 40320
firstValue: secondValue: 56 oper: !	56! = 6908521828386340864
firstValue: 4 secondValue: asd oper: !	4! = 24 Mathematically undefined request: ‘input’ has to be a positive integer
firstValue: 65 secondValue: 66 oper: !	170! = 7.257415615307998e+306 Can't compute.. Too big!
firstValue: -3 secondValue: oper: !	Mathematically undefined request: ‘-3’ has to be a positive integer
firstValue: 0 secondValue: 6 oper: !	0! = 1 6! = 720
firstValue: 8 secondValue: 3.4 oper: %	8 % 3.4 = 1.2

firstValue: 0 secondValue: 10 oper: log	Mathematically undefined request: The argument of the logarithm cannot be zero
firstValue: 10 secondValue: 0 oper: log	The base 0 logarithm of 10 is: 0
firstValue: 100 secondValue: 10 oper: log	The base 10 logarithm of 100 is: 2
firstValue: -100 secondValue: 10 oper: log	Mathematically undefined request: The base of the logarithm (10) and/or the argument (-100) cannot be negative!
firstValue: 81 secondValue: 12 oper: lcm	lcm of 81 and 12 is: 324
firstValue: 81 secondValue: 12 oper: gcd	gcd of 81 and 12 is: 3
firstValue: -81 secondValue: 12 oper: lcm	lcm of -81 and 12 is: 324
firstValue: -81 secondValue: -12 oper: gcd	gcd of -81 and -12 is: 3
firstValue: 4764 secondValue: 32 oper:	No operation selected!
firstValue: 4764 secondValue: oper: +	Fill in boxes first!

firstValue: secondValue: oper: !	Fill in boxes first!
firstValue: asd secondValue: 10 oper: *	Invalid Input! Enter numbers only.

Table 4: I/O test results

4. Conclusion

As a result, there obtained a fully working calculator application that fulfils requirements of the task. There were other operators added such as gcm or log besides the ones present in a simple calculator. These operators did not only require algorithmic skills to define but also required programming skills to attentively handle all the exceptions that come along. Also, there wer features added such as swap button or negationHandle method which automatically adds parentheses on second values if they are negative. This gives better looking output displays (eg. “3 + -2 = 1” vs “3 + (-2) = 1”).