

Group Report: Module 4, Simple Calculator
Group C: Toby Hansen, Dowon Yang, Rakin Bhuiyan, John Nagasawa
Group Leader and Reporter: Toby Hansen
March 2, 2025



This week Dowon and myself managed to give each other feedback on our individual programs, which helped both of us refine our projects into a final version.

Re: Module 4 Project

by Toby Hansen - Sunday, March 2, 2025, 9:31 AM

Attached is my pseudocode and solution to the simple calculator assignment. Let me know your Feedback and critiques.

Thank you,
Toby

 [ModuleFourGroupProjectPseudoCode_SimpleCalculator_TobyHansen.txt](#)
 [ModuleFourGroupProject_SimpleCalculator_TobyHansen.cpp](#)

Re: Module 4 Project

by Dowon Yang - Sunday, March 2, 2025, 5:53 PM

I just tested the code out and it worked well! Here is my version and my pseudocode. My pseudocode is a lot more barebones.

pseudocode:
Math tutor group project

Display Project title
Loop until user decides to stop program:


Prompt for first number
Prompt for second number
Prompt for +, -, *, /, %
Display error message if user inputs invalid character
End loop

 [Module4Group3IndividualProject_DowonYang.cpp](#)

Re: Module 4 Project

by Dowon Yang - Sunday, March 2, 2025, 10:03 PM

Ahh I see. I implemented the changes and it is working as intended now. the looping issue is fixed and the input validation seems to be working as planned. You can do another check but I think this will be sufficient for submission. Feel free to make any adjustments to headers or any // comments. Thank you the explanation on how to validate the inputs! I will attach my revision:

 [Module4Group3Project_DowonYang.cpp](#)

[Permalink](#) [Show parent](#) [Reply](#)

Re: Module 4 Project

by Toby Hansen - Sunday, March 2, 2025, 10:55 PM

Looks great! So what do you think, should we use the modulo formula and three doubles? Or should we restrict modulo operations to whole numbers and use two doubles?

[Permalink](#) [Show parent](#) [Edit](#) [Delete](#)

[Reply](#)

Re: Module 4 Project

by Toby Hansen - Sunday, March 2, 2025, 8:11 PM

Reading through and testing your program has been insightful.

I like that you declared your program variables outside of the main program loop, and that when restarting the loop both lowercase 'y' and uppercase 'Y' work.

I see now why the instructions say:

- "Declare ONLY three numeric variables, two of them double and one int."
- "Do NOT use fmod - use the arithmetic operators ONLY for the actions."

Professor Makhene was likely intending for us perform modulo by static casting the doubles to ints, using the % operator, and only allowing the computation if both operands are whole numbers. Personally, I feel like such a solution is incomplete, since modulo is a simple formula that can be used with decimals just as easily as whole numbers. This is a calculator, so I don't think it should throw an error for attempting an operation that is fully possible. When it comes down to it, my solution complies with the no fmod rule as it uses arithmetic operators only (-, *, and /), but it does not comply with the only two doubles rule. I will let you and the other group members decide which method we should use in our final program.

Intended method:

```
case '%':
    if (operand2 == 0) {
        cout << "Error: Cannot divide by 0" << endl;
    } else if (operand1 != static_cast<int>(operand1) || operand2 != static_cast<int>(operand2)) {
        cout << "Requires integer" << endl;
    } else {
        remainderResult = static_cast<int>(operand1) % static_cast<int>(operand2);
        cout << "Result: " << remainderResult << endl;
    }
    break;
```

My method:

```
case '%':
    // Mathematical formula of modulo.
    solution = operandOne - operandTwo * (int)(operandOne/operandTwo);
    break;
```

Unfortunately, the input handling in your program has a few problems.

For example, giving the program text input sends it into an endless loop looking for numbers and operators.

```
Enter first operand: Enter second operand: Enter operator (+, -, *, /, %): Error: Invalid operator. Please try
Enter first operand: Enter second operand: Enter operator (+, -, *, /, %): Error: Invalid operator. Please try
Enter first operand: Enter second operand: Enter operator (+, -, *, /, %): Error: Invalid operator. Please try
Enter first operand: Enter second operand: Enter operator (+, -, *, /, %): Error: Invalid operator. Please try
Enter first operand: Enter second operand: Enter operator (+, -, *, /, %): Error: Invalid operator. Please try
```

The problem with asking for all three inputs *before* doing any validations, is that it ends up attempting to assign characters still in the IO buffer to subsequent cin statements. This also means you can have weird inputs like 1.1.1+n which will successfully run the program as 1.1 + 0.1 = 1.2.

```
Enter first operand: 1.222.2
Enter second operand: Enter operator (+, -, *, /, %):
Enter first operand: 1.1.1+n
Enter second operand: Enter operator (+, -, *, /, %): Result: 1.2
Do you want to do another calculation? (y/n): End of program
```

In order to fix these issues, you need to use the IO operations we learned in Module 3.

This is how I handled edge cases:

```
while (! (cin >> operandOne)) { // Loops until valid number input.
    cin.clear();
    cin.ignore(1000, '\n');
    cout << "Invalid input, not a number.\nEnter first operand: ";
}
```

The while loop triggers if the input cannot be assigned to operandOne. cin.clear() clears error flags from the buffer.

cin.ignore(1000, '\n') discards the remaining characters in the buffer.

So if the user inputs abc, there won't be an endless loop attempting to assign abc as a double.

Instead, it will clear any error messages and the invalid input from the buffer, so that the input can be cleanly prompted for again.

```
Enter first operand: abc
Invalid input, not a number.
Enter first operand: 
```

The biggest revision Dowon needed was that he only had error handling for invalid operations, not invalid input. So I showed him how I did it in my program, which actually made me realize that there was a big problem with my own program that I had missed!



Re: Module 4 Project

by [Toby Hansen](#) - Sunday, March 2, 2025, 8:35 PM

After reading through Dowon's program, I realized I forgot to include an ignore operation after the while loops in my program to catch any double number inputs like 123.123.123.

This means that the weird input bypass bug I mentioned in his feedback applies to my program as well:

```
Enter first operand: 123.123.123
Enter second operand: []
```

After adding the ignore operation:

```
// This section gets input from the user for the first operand.
cout << "\nEnter first operand: ";
while (!(cin >> operandOne)) { // Loops until valid number input.
    cin.clear();
    cin.ignore(1000, '\n');
    cout << "Invalid input, not a number.\nEnter first operand: ";
}
cin.ignore(1000, '\n');
```

The weird input bypass bug is fixed:

```
Enter first operand: 123.123.123
Enter second operand:
Enter valid operator: []
```

(it assigned 123.123 to o1, and discarded the

remaining characters in the buffer)

Dowon quickly corrected the error handling, so we stitched together the best parts from both of our programs. I contributed my operator menu, operand input methods, and solution calculation. Dowon contributed his program variable scope and subsequent calculation method. We both contributed to the operand input method.

The biggest difference between our programs was that Dowon used one int and two double variables while I used no int and three double variables to allow for more complex modulo operations using non whole numbers. The assignment instructions were unclear with how exactly we were intended to conduct modulo operations, although they specifically specified:

- "Declare ONLY three numeric variables, two of them double and one int."
- "Do NOT use fmod() - use the arithmetic operators ONLY for the actions."

Since the official operator can only conduct on whole numbers, I used the actual modulo formula:

$$x \bmod y = y - x \lfloor \frac{x}{y} \rfloor$$

While my method complied with the no fmod() rule, it did not comply with the only two doubles rule. The downside of not using this method however, is that modulo operations could not be performed on non whole numbers, so $5 \% 2$ could be performed, but not $7.3 \% 1.6$. In the end I was not able to get input from my group members on which method they preferred, so I submitted our version that uses the mathematical modulo formula and three doubles. If Professor Makhene prefers our other version that uses static casting on two doubles with an int remainder variable, I can submit that version instead.