**Assignment 2**

**Name**: Longqing Chen **SID**: 22010057

## Part 1. Description

This calculator uses infix to postfix algorithms and stack to implement input calculation with correct operator precedence, parenthesis priorities and special evaluation, such as sqrt(), sin(), cos(), factorial calculation and pi. It also tolerates input without proper parenthesis. For instance, no error would pop when user inputs sqrt2. Current program does NOT support arbitrary precision and variable definition, in that sense, it satisfies requirement 1, 2, and 4 of the assignment. There are also error alerts and other user-friendly instructions in the beginning. The source code, which is extensively commented, can be found in the GitHub repo attached here. All credits are listed in the code.

## Part 2. Result & Verification

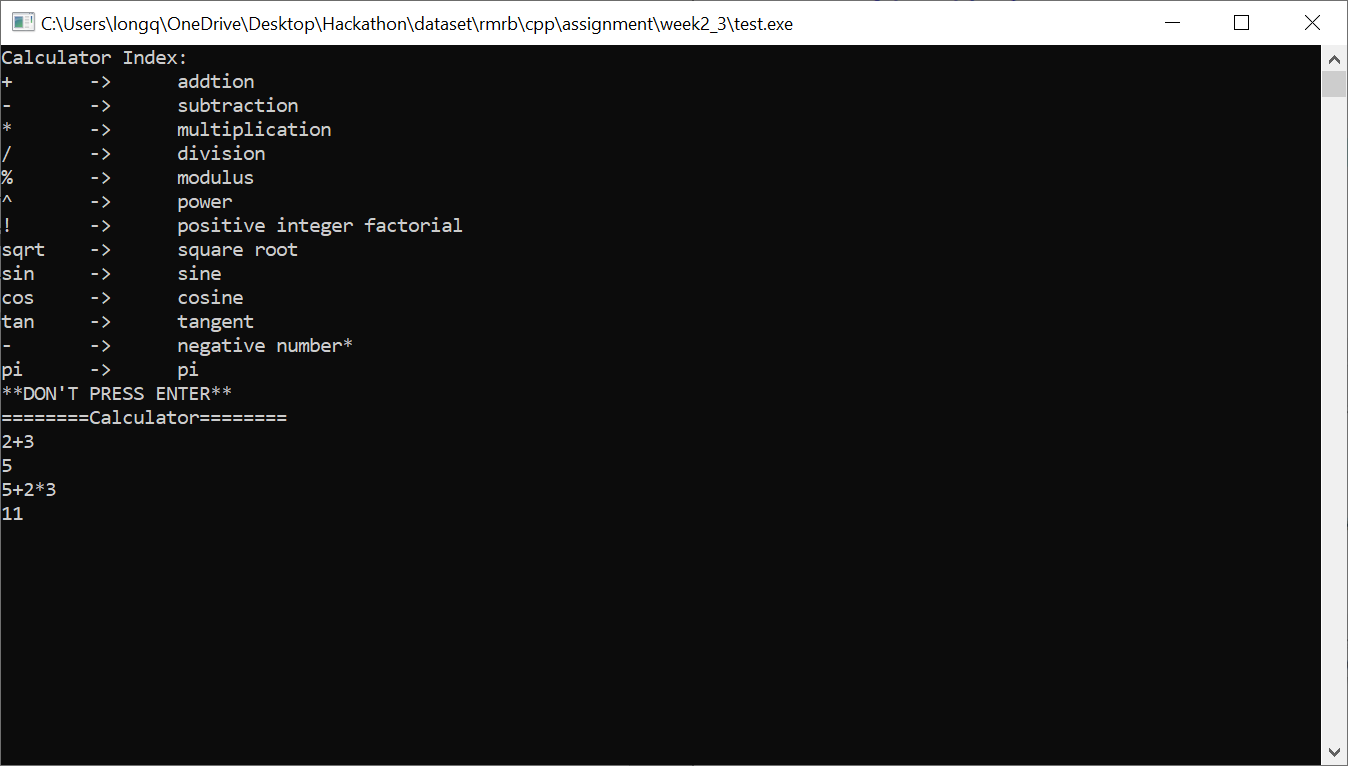
Test case #1: Input and operator precedence check

2+3

5+2\*3

2--3

Screen-short for case #1:



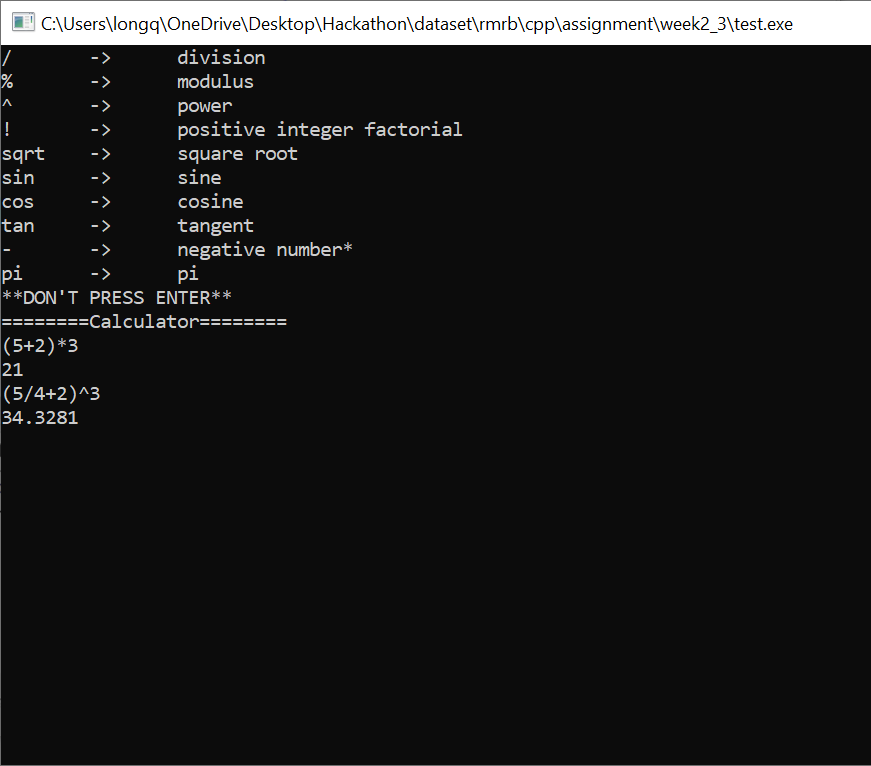


Test case #2: Parenthesis priorities

(5+2)\*3

(5/4+2)^3

Screen-short for case #2:

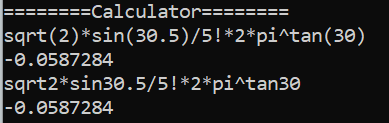


Test case #3: Special Evaluation and Error Tolerance

sqrt(2)\*sin(30.5)/5!\*2\*pi^tan(30)

sqrt2\*sin30.5/5!\*2\*pi^tan30

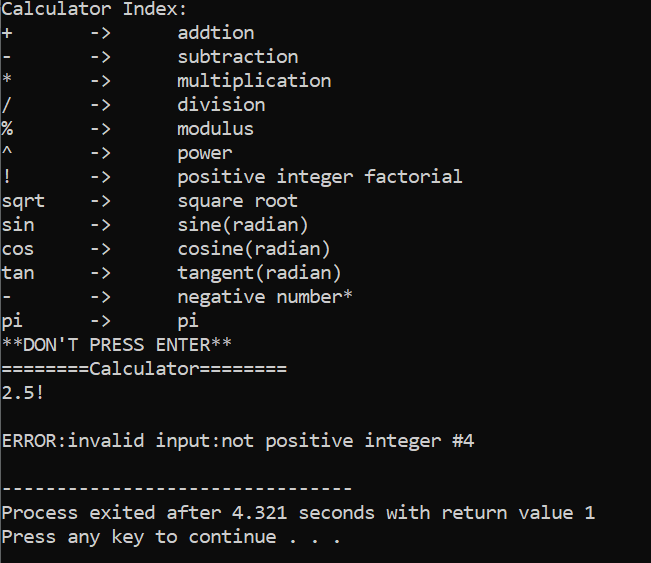
Screen-short for case #3:



Test case #4: User friendly features

Calculator index and error alerts

Screen-short for case #3:



## Part 3. Difficulties & Solutions, or others

This project is by far the most challenging one I have ever had since I have never actually coded before and only learnt MATLAB at school and python by myself. So yes, lots of difficulties.

There are many trivial problems, such as how to turn the string “sqrt” to an operator that the calculation process would easily pick up and how to make sure the negative numbers

* Infix to Postfix – keep it straight!

I do not know how to parse input and calculate the parsed input in the correct order. Through extensive research for a couple of days, burying my head in tutorial videos, stack overflow blogs and coding tips websites, I learnt about stack as a data structure tool to flexibly store and return strings that comes in hand. Another big discovery is the postfix expression/notation that automatically places operands and operators in the ordered precedence. For instance (2+3)/5\*6, an infix expression would be transformed into 2 3+ 5 /6\* as a postfix expression.

* FSM and Enumerator – keep it clean!

The next big obstacle is to tokenize the parsed string inputs into doubles that are operatable. How to determine whether the users’ input is legit for calculation? What if there are multiple decimal points? Negative marks in the middle of digits? Chinese characters?

Again, I run into a tutorial about the finite state machine that helps determine what types of instructions can/cannot be accepted according to the enumerator specified previously, that indicates whether the token is negative, integer, has decimal point or unknown.

* User friendly alerts and instructions – keep it nice!

This should be trivial and not about coding skills, but it really improves user experience in real life! Calculators with instructions and alerts can help users adjust their inputs more easily.

Further breakdowns of why I did what I did and how I did it can be found in the code in the GitHub repo! Thanks a lot!!