**Deliverable 3**

CRC Model

Calculator is main class with functions for:

* Choose Function (User chooses initial function to use)
* Parse Additional Function (After choosing first and entering values, can choose to parse additional function or get answer)
* Display History (Displays calculator history of the current user since they started using calculator)
* Addition (Simple addition function)
* Subtraction (Simple subtraction function)
* Multiplication (Simple multiplication function)
* Division (Simple division function)
* History (History is an array of Functions, which are separate classes for special ones and strings for simple arithmetic)

Exponential Function is a class

* Calculate Answer (calculates value of exponential function after user enter values for a, b and x – currently x can only be an integer)
* Get Superscript (takes in string and outputs it as an exponent)
* Multiplier Number (the value of a for the function)
* Base Number (the value of b for the function)
* Exponent (the value of x for the function)
* Answer (the final answer of the function after values entered)

Pseudo Code for Exponential Function

function calculate\_answer(mult\_num, base\_num, exp\_num, answer)

answer = 1

if exp\_num is a fraction then

num = numerator of exp\_num

den = denominator of exp\_num

if num is an integer then

if den is an integer then

// denominator of fraction represents root value for base\_num

base\_num = get\_root(den, base\_num)

exp\_num = num

return calculate\_answer(mult\_num, base\_num, exp\_num, answer)

else

i = 2

tempDen = den

d = 0.0

while tempDen is not an integer

d = den

d \*= i

i += 1

tempDen = d

base\_num = get\_root(d, base\_num)

exp\_num = num \* (i- 1)

return calculate\_answer(mult\_num, base\_num, exp\_num, answer)

else

i = 2

tempNum = num

n = 0.0

while tempNum is not an integer

n = num

n \*= i

i += 1

tempNum = n

if den is an integer then

den \*= (i – 1)

base\_num = get\_root(den, base\_num)

exp\_num = n

return calculate\_answer(mult\_num, base\_num, exp\_num, answer)

else

j = 2

den \*= (i – 1)

tempDen = den

d = 0.0

while tempDen is not an integer

d = den

d \*= j

j += 1

tempDen = d

base\_num = get\_root(d, base\_num)

exp\_num = num \* (j – 1)

return calculate\_answer(mult\_num, base\_num, exp\_num, answer)

else if exp\_num is an integer then

if exp\_num >= 0 then

for i = 0 to exp\_num

answer = answer \* base\_num

answer = answer \* mult\_num

return answer

else if exp\_num < 0 then

for i = 0 to (-1 \* exp\_num)

answer = answer \* base\_num

answer = (1/answer) \* mult\_num

return answer

else

i = 2

tempExp = exp\_num

e = 0.0

while tempExp is not an integer

e = exp\_num

e \*= i

i += 1

tempExp = e

base\_num = get\_root(e, base\_num)

exp\_num \*= i - 1

return calculate\_answer(mult\_num, base\_num, exp\_num, answer)

function get\_root (root, num)

i = 1

result = False

while result == False

n = i

for index = 0 to root – 1

n \*= i

if n == num then

return i

else if n > num then

i = Root(root, num, i – 1, i)

return i

i += 1

function Root (root, num, i, j)

mid = (i + j)/2

n = mid

for index = 0 to root – 1

n \*= mid

if (n == num) OR (abs(num – n) < 0. 000000001) then

return mid

else if n > num then

return Root(root, num, i , mid)

else

return Root(root, num, mid, j)

Task

Compute the sum of the square root of 2, the cube root of 3 and the fourth root of 4 using the Exponential Function (√2 + 3√3 + 4√4). Hint: use fractions in the exponent.

Persona

Marco is a 2nd year Concordia Student in the Pure and Applied Math Undergraduate program. As a student, he uses a calculator almost every day, mostly for trigonometric functions or simple arithmetic. He prefers the practical aspect of math, specifically algebra and statistics, and enjoys solving problems. While not experienced with computers, he has used an online scientific calculator before and has some experience with command line. A hard worker who values simplicity and ease of use, he hopes to become a TA for an algebra class in the future.

Use Case

This use case is a simple use of our Eternity System. The user is a student, and the purpose of their use is to retrieve an answer to a problem. The Choose Function use case is a generalized use case, which branches into Choose Special Functions or Choose Simple Arithmetic. The initial use case also has the preconditions that the student has analyzed the problem and identified the desired function and needed values. The user makes a choice, and is then asked to Enter Values, for their chosen functions. Once values have been entered, the system gives the student the chance to Parse Secondary Function. If they want to, then Parse Secondary Function becomes an extension of Choose Function as the student must now choose a second function to parse. Once the secondary function is parsed, or if none was chosen, then student moves on to Perform Calculation, which is a generalization of the Receive Final Answer use case and displays the answer to their function. The student can then view the calculator history, as at least one function must be performed for the history to not be empty.

Micro Architecture Design

The micro architecture design for the Eternity System is broken down into classes. One Calculator class, which acts as the main class and how the user interacts with the Calculator, and the Function classes, a class for each of the special functions that includes any subordinate functions or variables they may need. This way, each function can be modified on its own without changing the Calculator class. The Calculator class includes the functions for simple arithmetic and allows the user to parse functions as well. This was done since both of those are functions of the calculator system, and not special functions designed by others. Each function class also has a \_\_str\_\_ function which is used to store the function information in history, which can be viewed later. The Calculator has only 1 data member which is the History, represented as an array, which stores the information of all calculations done from the latest start up of the system. The History can also be displayed through the Calculator class.

The Exponential Function class has several functions, such as the set methods to set the values for the function: multiplier, base, and exponent. The set\_values function is used to retrieve the values from the user and allows the user to enter fractions as well. The representing function is used to determine if the passed string s, is an integer or not, since the algorithm used requires an integer exponent. However, to deal with decimal or fractional exponents, the square root function is used. The get super function is just used to display the exponent as a subscript to the screen, instead of using the ^ sign. And finally, the \_\_str\_\_ method allows the information from the function to be displayed in an easy-to-understand way.

Algorithm

The algorithm chosen was used to deal with decimal or fractional exponents. When the exponent is an integer, calculating the answer becomes trivial, the base is multiplied by itself the exponent number of times, and then multiplied by the multiplier to get a final answer. When the exponent is a fraction or decimal, it was decided the easiest way to deal with it, was changing the base so that the exponent becomes an integer. The algorithm works by first turning the decimal into a fraction if it is not already a fraction. This is done by multiplying the exponent by integer values starting from 2 and increasing by 1 each time, until the exponent becomes an integer. Increasing by 1 was decided since it is unclear how many digits the decimal will have and to avoid missing smaller integer values to multiply by. For instance, if the fraction is 4/100, while multiplying by 100 seems like the easiest choice, we can multiply by 25 and get 1 instead, making it even simpler.

To turn the decimal into a fraction, we multiply it by the integer value found over itself (essentially, multiplying it by 1). With a fraction, the denominator can be used to change the base, since #1/a is equivalent to a√#. The get\_root and Root functions are used to determine the specified root, since the denominator can be any value, for passed argument. This generates a new base, and then the numerator is used as the new exponent value for the new base. Since the exponent is an integer again, the new base is multiplied by itself, the passed number of times. The calculate\_answer function deals with all 3 of the mentioned cases, when the exponent is already a fraction, when it is an integer, or when it is a decimal, and calls the get\_root function if need be.

The root function was used since it still allows the original algorithm for calculating the exponential function to be used. It is also easier to deal with the exponents when they are integers, while for the base, it does not matter since it is just being used for simple multiplication. The get\_root function works by first taking an integer i and assigning it the value of 1. It is then multiplied the desired number of times for the root (twice for square, thrice for cube, etc.), to determine if it is the root value. The starting position was chosen to be 1, since we do not know what root we are looking for util the function is called, meaning an educated guess cannot be made for the value, like with simple square root. If the value of i is the root value, the function returns i, else it increments i by 1 until it is the root, or until the multiplication of i to itself the root number of times is greater than the value we are trying to find the root for. Meaning that the root of the passed value is between i and i – 1. When that is the case, the root is not an integer, so Root is called. It is initially called with a range of i and i – 1, and calculates the middle value between the two, mid, to see if it is the root, or a reasonable approximation. If it is, it returns the middle value, if it is greater than the root, it calls Root again, with a range from i – 1 to mid, and if it is less, it calls Root with a range from mid to i. This cycle repeats, as the function calls itself, changing the interval of search each time, until it finds the root, or the root within a reasonable difference of the value |value - midn| < 0.000000001. 0.000000001 was chosen to offer a little more precision then ordinary calculator, and to ensure no infinite loops.

Test Result

Calculate the sum of the square root of 2, the cube root of 3 and the fourth root of 4

Using percent difference, we will see the error percentage from the cube root value calculated using the Eternity System. For the purpose of simplicity, we will use up to 11 decimal points each, for both values.

Percent Difference = [|V1 – V2|/([V1+V2]/2)] x 100

V1 = 1.44224957031

V2 = 1.44224957027

Percent Difference = 2.77345 x10-9 %

This margin of error is negligible, so the value was calculated as close to reality as possible.