מטלה מונחה עצמים: 204901417_206713141

Monom:

right input("5X","3.22X^2)
wrong input("3X2","2x","3X^3.2")

Monom(string)-
The function obtains a string that represents a monom, our function checks whether the string actually is a monom format and, if so, builds a monom by properties that represent monom. In our function, we have defined monom by the following features:
- X is only represented by an X letter(capital letter)
-The coefficient of X can only be a decimal number (can also be without a coefficient)
- It possession should be an integer
-The power represent :^

add
- The function receives a monum and if its power equal to our holding activated monum (this) we can connect between the monumes.

derivative-A function that calculates the monom derivative

f-
Calculates the function value at a point

Multiply-
Gets a monum and multiplies the monom coefficient and holds it in the activated monom (this)
Equals- A function that receives a monum and compares it with the activated monum

Polynom:
right input("3X^2+5X+8","3.22X+5","2X^4+8+2")
wrong input("(3X+5)","2x","3X^3.2")
Defined by a collection of monomes in a linked list, in our function we defined the Polynom by the following features:
- The polynomial does not work with the '/ ' operation
-The polynomial does not contain the characters: ")", "("

Polynom(String)-
The function gets a string representing a polynomial and works by the recursion method: The function will look for a central + or - action and when you find one, send it to itself until the +/- and from then on.
For example, our function receives a string, the function runs until you find a + or - action, save all the word until the action and send it to built Monom, and the rest of the word to the end, call itself recursively (for that recursion method we built helpPolynom)

f - Calculates the function value at a point.

add(Monom)- The function goes over the linked list and checks each monom whether it is equivalent to the monum that we want to add, if so we connect the monum, if not , we add the monum to the end of the list

and send it to the sort function we built.

add-
Connecting Polynomials: Using an iterator, we go over the polynom able and each time we send the monom to the add function with monom.
multiply(Monom m1)-Each monom in the polynomial we sent to the multiply function of monom.

-multiply(Polynom_able p1)
Multiplication of Polynomials: We took the polynomial on which the action was called and we copied by the copy function, so that the polynomial does not change at runtime.
We used the itinerary on a polynom able to multiplied each time a Monom with the copied polynom.
And so we avoided the problem that multiplication would change the monom and not reach the real result.
We saved all the duplicates we made in a new polynomial, which we added to the original polynomial and then subtracted what was in the polynomial before multiplication.
substract(Polynom_able p1)-
Substract Polynomials: Sends the function to multiply by -1 And the answer is sents to the add function.

equals(Polynom_able p1)- Comparison of Polynomials:
With an iterator we go over the polynomable and at the same time using "place"
on the activated polynomial,
In each iteration, we will check whether the monomers are equal and do place++
If we have finished and the polynomial sizes are not equal, we will return false,
Otherwise we will return true.

root(double x0, double x1, double eps)-
We built a function (roothelp) that works in recursion, the function will only send to the roothelp function if the x0 x1 marks are opposite symbols(-,+).
roothelp - we created a middle variable that represents the middle of the segment between x0 and x1, if the value of the middle in the f function is 0, we will return the middle, otherwise you will recursively recalculate the middle with the x with the opposite symbols. Our stopping conditions - if one value of x0 x1 is equal to zero we will return them otherwise we will continue to split until we will get to x whose function value is close to 0 – close by epsilon and return it.

derivative()-
We will go over with an iterator on the activated polynomial and each monom is sent to the monom derivative function.

area-
According to "", the calculation of the integral area is by division into small sections up to Epsilon And connecting all areas together. We split the section from x0 to x1 into small sections and returned the area connection.

sort()- We sorted by the compare function in the Comparator class to sort by power.

toString()-
In this function we had to handle some problems such as:
- If the prefix is non-minus then before adding the monom to the string representing the answer, we will add a plus before, if the prefix is minus we will add the monom as is.
If the coefficient of the monom is 0, the function will not add it to the answer stirng.
gather ()-

first we will do the sort function and then we move on monom monum and check if its power is equal to the the monum afterwards, if so,we use the add function and reset the monum.

test3()-We tested the multiplication and connection function:
We took a polynom and connected it to itself, and on the other hand we took a polynom and multiplied by 2 and expected to get the same result.

test4()-A function that finds the function's extreme point

Test5()-
We tested the parabola symmetry relative to the vertical axis

Test6()-We tested the copy and subtraction functions