Ideagen – Coding Test

Time Taken – About 1 Day cumulatively (Split across 2 days 14th and 15th Dec 2022)

Given the conditions:

1. String parameter
2. Numbers and operators are separated by spaces
3. Brackets are also separated by spaces
4. Not using built-in methods or third-party libraries

This Code considers and can handle:

1. Simple equations
2. Equations with brackets
3. Nested brackets
4. Equations follow BODMAS where possible
5. Division by zero

Some constraints:

1. If the operators and brackets are not separated by spaces, it will not compute properly.
2. If the equation has more than 2 operators without brackets it is not able to calculate completely.

For the solution, I implemented a method Calculate which will be able to calculate the string parameter that is keyed in by the user.

When the user keys in the value, the program looks at the string, if there is any brackets in the string parameter, we will call method RemoveBrackets.

For this method it will check the string to get the values between the brackets, then solve it by calling ResolveBrackets (which in turn will call Calculate method to carry out the calculations) and then replace the values in the bracket with the computed value.

From there, it will check if there are any other brackets or values to compute first, before resolving to call Calculate method in the end to get the final results.

Example:

10 - ( 2 + 3 \* ( 7 - 5 ) )

Calls, RemoveBrackets which calls ResolveBrackets ( to solve equation in the bracket)

ResolveBrackets which calls Calculate (for the values in the brackets only that is passed from ResolveBrackets method)

Results:

10 – ( 2 + 3 \* 2 )

In this method, it is able to calculate equations, as it splits the values into the first value, operator and then the second value. If the equation is longer than two values, it can compute up to 3 values.

If there are 3 values, we apply BODMAS to make sure the equation is computed properly.

Example:

For the second part of the equation, since it is in brackets, this will be done first.

( 2 + 3 \* 2 )

Will give the result of

( 2 + 6 )

= 8

Then we call Calculate again at the end of ResolveBrackets to get the final result:

Example:

10 – 8 (value which was calculated from ResolveBrackets)

= 2

This gives us our final answer.

If the equation only has one set of brackets, it will be similar to the nested brackets but it will compute the brackets, and once that has been solved it will call the Calculate method and return us the final result.

Example:

( 11.5 + 15.4 ) + 10.1

The first part (11.5 + 15.4 ) will trigger RemoveBrackets to be called similar to the first scenario, then once it has been solved, it will call the Calculate function to solve the remaining equation

26.9 + 10.1

= 37

For normal equations, when the user keys in the values, it will straight call the Calculate function, as it checks the string and sees there is no brackets.

Example:

11.1 + 23

The Calculate method gets the string and then splits it so 11.1 will be the first value , + is the operator and 23 is the second value.

From there it will compute 11.1 + 23 and give us the result of 11.1 + 23 = 34.1

This will then be returned and displayed to the user.

Below is the code that would be able to calculate all the test equations provided:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace IdeagenAssessment

{

class Calculator

{

public static double Calculate(string sum)

{

//Your code starts here

double result = double.NaN;

string numInput1 = "";

string numInput2 = "";

string numInput3 = "";

string op = "";

string op2 = "";

sum = sum.Trim();

string[] phrases = sum.Split(' ');

List<String> list = new List<String>();

int i = 0;

foreach (var phrase in phrases)

{

list.Add(phrase);

i++;

}

//first number

numInput1 = list[0].ToString();

double num1 = 0;

double.TryParse(numInput1, out num1);

//second number

numInput2 = list[2].ToString();

double num2 = 0;

double.TryParse(numInput2, out num2);

op = list[1].ToString();

switch (op)

{

case "+":

result = num1 + num2;

break;

case "-":

result = num1 - num2;

break;

case "\*":

result = num1 \* num2;

break;

case "/":

// If user enter non-zero divisor. - If not break and return error

if (num2 != 0)

{

result = num1 / num2;

}

break;

default:

break;

}

if (list.Count > 3)

{

//third number number

numInput3 = list[4].ToString();

double num3 = 0;

double.TryParse(numInput3, out num3);

op2 = list[3].ToString();

if (op2 != "")

{

switch (op2)

{

case "+":

result = result + num3;

break;

case "-":

result = result - num3;

break;

case "\*":

result = result \* num3;

break;

case "/":

// If user enter non-zero divisor. - If not break and return error

if (num3 != 0)

{

result = result / num3;

}

break;

default:

break;

}

}

//special case with more than 1 value in equation - following BODMAS

if (op == "+" && op2 == "\*")

{

result = num1 + (num2 \* num3);

}

else if (op == "-" && op2 == "\*")

{

result = num1 - (num2 \* num3);

}

else if (op == "+" && op2 == "/")

{

result = num1 + (num2 / num3);

}

else if (op == "-" && op2 == "/")

{

result = num1 - (num2 / num3);

}

}

return result;

}

//If equation has brackets - solve first

public static string RemoveBrackets(string text)

{

while (text.Contains('(') && text.Contains(')'))

{

int openIndex = 0;

int closeIndex = 0;

int j = 0;

for (int i = 0; i < text.Length; i++)

{

if (text[i] == '(')

{

openIndex = i;

}

if (text[i] == ')')

{

closeIndex = i;

text = text.Remove(openIndex, closeIndex - openIndex + 1).Insert(openIndex, ResolveBrackets(openIndex, closeIndex, text).ToString());

break;

}

}

}

return Calculate(text).ToString();

}

public static double ResolveBrackets(int openIndex, int closeIndex, string text)

{

double bracketAnswer = 0;

bracketAnswer = Calculate(text.Substring(openIndex + 2, closeIndex - openIndex - 3));

return bracketAnswer;

}

}

class Program

{

static void Main(string[] args)

{

bool endApp = false;

while (!endApp)

{

double result = 0;

string sum = "";

Console.WriteLine("Please enter an equation here:");

sum = Console.ReadLine();

try

{

if (sum.Contains('(') && sum.Contains(')'))

{

double.TryParse(Calculator.RemoveBrackets(sum), out result);

}

else

{

result = Calculator.Calculate(sum);

}

if (double.IsNaN(result))

{

Console.WriteLine("This operation will result in a mathematical error.\n");

}

else Console.WriteLine("Your result: {0:0.##}\n", result);

}

catch (Exception e)

{

Console.WriteLine("An exception occurred trying to do the math.\n - Details: " + e.Message);

}

// Wait for the user to respond before closing - or continuing

Console.Write("Press 'n' and Enter to close the app, or just press Enter to continue: ");

if (Console.ReadLine() == "n") endApp = true;

Console.WriteLine("\n");

}

return;

}

}

}