Index

Contents

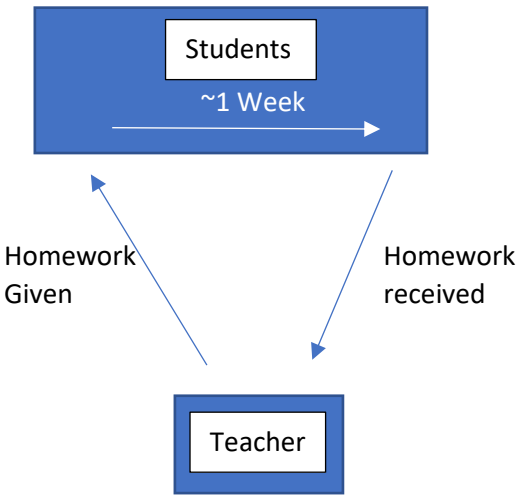
ANALYSIS

Introduction.

The subjects that I have chosen to undertake in A level are primarily maths related. This has also shown the importance of homework in these subjects. There is a strong correlation between homework completion and grade improvement; it is not a hard thing to understand, sure, you will get better at something you practice.

Therefore, this is relied to conjecture your grades, and your likelihood of success in exams. Homework has great importance, both for determining and improving. It is in this heavy reliance that a flaw lies.

Original System.



The system does not use any form of computing in this stage. I will call it Stage 1, and in this stage, students receive homework and complete it within a timeframe. The context of what the work is is important, though. Typically, for A level, maths homework requires a great period of time to mark. **Marking is where the flaw lies**, or more so, where it can be **greatly improved**.

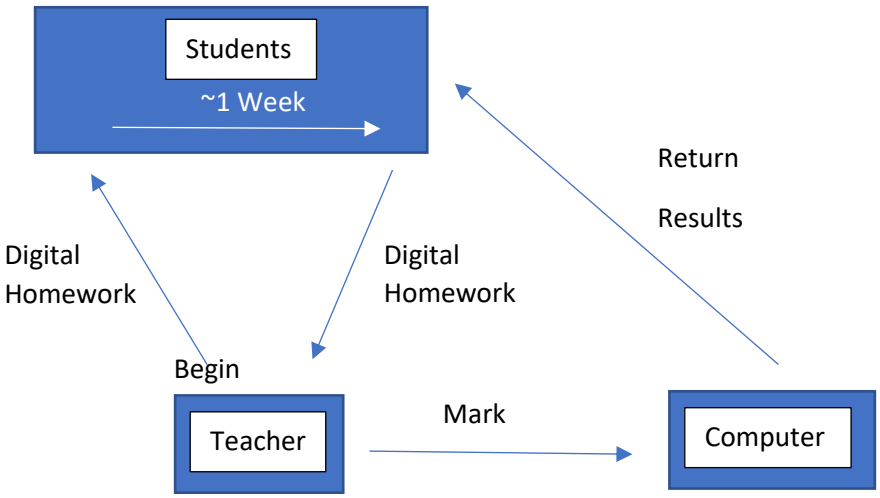
It is typical to mark one question at a time, for all papers. Situations may arise where (unfortunately) students' hand in their work late. The consequence is, when of merciful mind, homework is delayed in return. Alas, there is not much one can do about this situation. The grave danger in such delays are (of human nature) when inefficiencies are procured, which can degrade teaching time. This can be from a loss of motivation, time limits (stemming from having to focus on homework) and portions of a lesson spent on gathering lost homework.

Nonetheless computation is a great tool in automation. It would be theoretically possible to mark homework without a human's intervention, yes? Therefore, it would not be needless to say that the solution is as such: a **math's questioning program**.

Top end Solution.

Of course, there are certain restrictions (both due to technical and intellectual inability) that need to be examined. One is that the method of input for the homework has to be within the system; you cannot complete the homework on a piece of paper (at least for the answers).

Students must receive homework digitally.



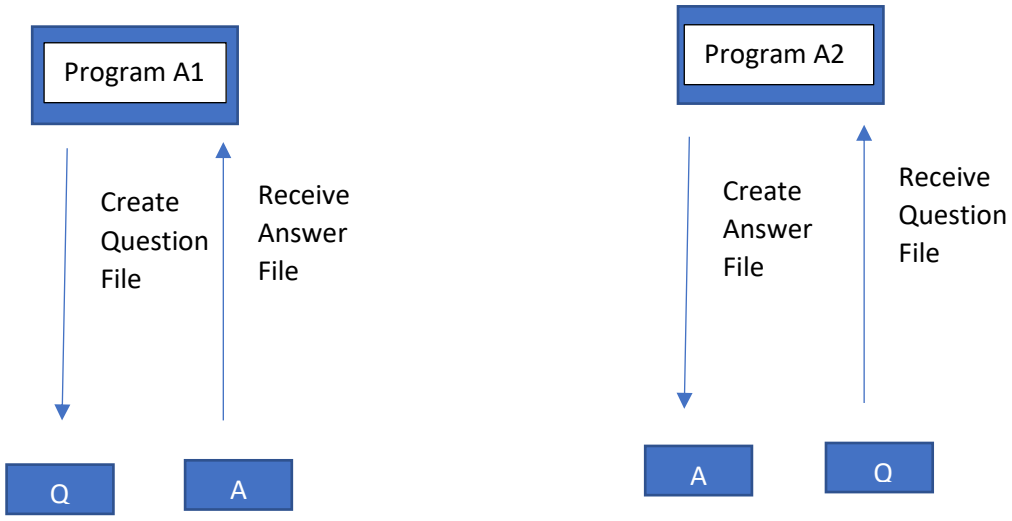
There are a two main differences in this new model.

- Due to it being digital, homework can be returned whenever they want (within the deadline)
- Homework can be immediately marked on a per student basis.

Now, this is no do it all method. In the front end of A level, it gets tough. I don't mean naturally; I mean in implementation. I must understand the mathematical theory before hard coding a system that solves it for me.

For a method like this to work there requires a strict system. It is such that the homework generated must be created by pre-existing sets of questions; though the specifics of the question can be altered depending on its type (such as 2x +3 =?).

A program cannot solve something if it has not been told how to.



Teachers will have Module A1; this will allow them to create a set of questions and send it off to their students. Module A2 then (for students) can receive this question file and complete it. Once completed this program module can create an answer file to hand back. This system is designed so that a central server is not required.

The answer file will be marked by Module A1 only.

It is imperative that Students cannot access software that can mark their questions. The consequence of such a thing is quite self-explanatory.

Therefore, there will be a simple login interface before being able to use the program. Teachers and Students will each have their own 'code' that lets them login. There won't be separate logins for every student, or teacher.

Due to the advanced nature of these files they will probably require a form of JSON; its either that or a very arduous syntax creation for text saving.

Constraints and Limitations.

There are a lot of nuances that will be apparent when programming such a thing. A program like this will have two challenges –

- Coding a coherent method of solving multiple different areas of mathematics
- Making a user interface that is usable.

For point one the challenge is clear. It is not easy making a computer do math. This sounds odd, but really, having a computer evaluate "2x +3x = ?" isn't as easy as it may seem. Yet this is but the trivial part of coding a 'coherent' method. Things like simple derivatives require a great amount of foundation to attempt. D(f(x))/dx means nothing to a computer, and this is greatly simplified. If you begin going abstract, such as partial derivatives, multivariable functions, solving confusing fractions with many weird operators. It blazes into a nightmare.

It is the type of thing that (just as you would do normally) you climb up gradually (and maybe gracefully). There are things I mentioned that really don't need creating. Even though I am focusing on A level, a contingency will be said that it really depends on the difficulty of making a computer solve problems for you.

Onto that topic, the only real non messy method of doing this is through Object Orientated Programming. Unfortunately, I am not highly skilled in Visual Basic's implementation, instead used to a more novel approach to it (LUA). This specific problem suits it well, have a math class handle evaluating with Polish Notation, conversion with Shunting Yard – and then have sub classes that inherit, like a simultaneous equation solver.

On that regard there are many wonderful algorithms that can help me achieve some of the computational requirements. Reverse Polish Notation, being what lets one know what 51-26/2 is, is great. This cannot use infix (what I just mentioned) so I need the Shunting Yard algorithm for conversion (operator-precedence parsing) to post-fix. The calculator industry is a great tool for me in this endeavour.

You should get the idea, general functions in a general class, and specific functions sub classes of that, or maybe sub-sub classes in some cases.

The data aspect of this program will also require implementation. Reading JSON is one, and also accounting for things like –

What happens if a student only does half of their homework?

How will they input their answer? Will it use a math format standard? What technical level would they have to be?

These things account for a lot of time spending on things normally overlooked. Problems typically are either noticed, or not…

End Users.

This system will be built for teachers and students. I intend for an ease of use where no servers are needed. It is simply a one to one relationship; email is a preferable method of communication, or google classroom.

Meeting 9/07/2020.

In this meeting I and my client discussed the outline of the program. One thing stated was that I should make the interface as simple as possible; he said many of the maths teachers are not well versed in technology.

He also listed that, if feasible, there should be a method to upload the workings out for questions. Maths in the higher level typically gets marked by method more than answer. I noted that this can be possible, but it may make the exporting method much more advanced.

He agreed that I shouldn't make multiple logins for teachers; there are systems already in place for student mark storage and sorting. Though he noted that it may be useful to have a method to export the percentage and each question's marks into a spreadsheet. I said that this can be made, but I would have to leave it to a late prototype. I have little experience with spreadsheet's in vb.net so I wouldn't want to place a striking unknown in the middle of development.

We went onto the actual system for creating questions. I explained that there will be a number of question templates that teachers can choose from. Each template can be modified in specific ways that I have allowed (this is to make sure some questions can be made to have less customisability in case of complications). I followed by saying that I plan on having two options on question creation. The other being a pseudo-random selection where the program creates a question file for you.

He said that it would be useful to have some parameters that you can edit with the pseudo-random question creator. I agreed, and on specifics he wanted categories of questions. An example could be 'Algebra 1' where it creates simplifying questions.

He also questioned whether there would be a difficulty parameter. I noted that the implementation will have to be decided when I program it. But there were two ways that I thought of. Either have different templates for different difficulties, or have a single template with a difficulty slider. For now, we agreed on the first method as I guessed that it would be easier to implement; no disagreement was found from the client.

We shifted focus onto the answering of question files. I said that the general idea was that you select an answer file and the program would show an outline of how well the student did. He asked whether it would be possible to look at each question. I followed that I plan to give teachers access to every answer in case the program errored (though I noted that this should be rare). I also said that this would allow the teachers to view the working out.

He expressed concern on how the students would find the questions compared to that of paper. Specifically, he was unsure how they would input the answers. I pondered and found that the most universal method would be for each template to have its own method of answering. Some may just have a text box; others may have a very constricted set of boxes the user merely needs to put numbers into.

For the workings out he found that the options to upload images of written work would be useful. I agreed.

We ended the meeting there. In retrospect I found that I covered most (if not everything) of what I needed to know. The client seemed happy with the arrangement and was fine with the constraints that I needed to put in place.

Possible Backend Solutions.

To approach a problem such as solving abstract forms of data, like $3x + 4x$, there are many methodologies to approaching it. Some are more suitable than others, and have their own merits and drawbacks.

Currently, I can either rely heavily on REGEX, or use expression trees. REGEX is a language used to match patterns of numbers and letters. This is great if I, say, want to detect all the coefficients of a quadratic equation.

However the flexibility of such a system is limiting. If I were to want to develop some form of simplifier using this, it would be nigh impossible. One must understand that although things like brackets are intuitive to humans, computers have a much harder time sorting out what to do.

Thus, if I were to approach the problem by just reading from a string, and figuring it out through REGEX, it will always have an inherent limitation.

Say the expression:

$$\text{“}(3x\ + 9x^2) * \big(9x - (2y + z)\big)\text{”}$$

I want to simplify it. I can attempt to create regex expression to match what I want from here, and eventually I will most likely solve it. But this attempt can be circumvented by a simple variation like –

$$\text{“}(3x\ + 9x^2) * \Big(9x - \big(2y + (z - 2x * 9x)\big)\Big)\text{”}$$

This is an inherent flaw.

If it is not clear already there is a certain recursive nature of expressions. Brackets, like a bunch of items in a stack, are made to evaluate in turn.

Thus comes the second method. **Expression trees.**

This takes the form (at first) as a binary tree that takes a simple concept. The node is either an operator (of binary descent) or a value like 3x. I have already mentioned of the ideas such as ‘Post Fix’ and ‘Shunting Yard’, which are of great use to remove brackets.

$$\text{“}3x\ 9x\ 2\ \hat{}\ +\ 9x\ 2y\ z\ 2x\ 9x\ *\ -\ +\ -\ *\text{”}$$

The method of converting to post fix is simple and efficient. It allows the computer to better understand the expression. However, I need to parse this expression. It requires simplification.

With the removal of brackets, there is a design in this post fix expression, one that allows me to easily convert it into an expression tree.

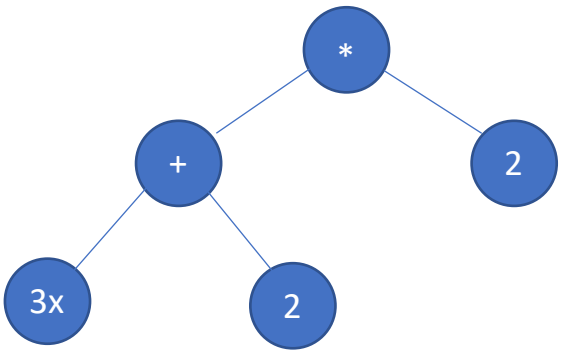
Say for a simpler example –

$$\text{“}(3x + 2) * 2\text{”}$$

This in post fix is –

$$\text{“}3x\ 2 + 2 *\text{”}$$

The details of this will be in **design**. But I can parse this into an expression tree with ease.



A tree is something a computer understand well. This form of data is what ‘computer algebra systems’ typically use. Manipulating and collecting like terms, etc, are done in the tree and through traversals.

There are flaws to its general form. Expressions can be represented in multiple ways, and computers hate the ideas of division and negation. Nonetheless this way of representing data has the most

potential to encompass problems. As it no longer wholly relies on REGEX I can input anything and it should be able to simplify it.

This of course entails a much harder system to procure, which heavily involves recursive thinking. But it will in the long run help me greatly.

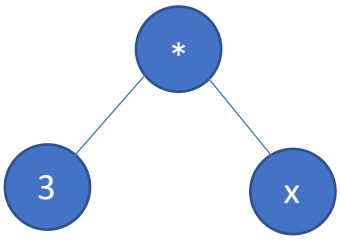
Another idea is that things like -

$$\text{“}3x\text{”}$$

For the general tree to work I must represent this as

$$\text{“}3 * x\text{”}$$

The difference lies in the fact that I will parse it as a tree instead of a term.



Thus there will be no occurrence where I will have odd unpredictable behaviour occur.

Evaluating Expression Trees.

This is an important process for planning my evaluation. I have already stated my preferred method of expressing expressions in tree, but its use will only be great if I can learn how to traverse such a tree.

There are two ways I can approach this issue. One will rely on REGEX to perform comparisons, but will be in a controlled fashion compared to using only REGEX, and the other will solely use the tree to do every simplification.

Mentioned before is the term ‘traversal’ and there are three main ones which I am interested in.

Traversing Name	**Traversing Method**
Post Order	Left Right Root
Pre Order	Root Left Right
Inorder	Left Root Right

This traversing method tells what you recursively traverse in order. The pseudocode for Inorder is -

```
Function IN_ORDER(NODE As TREE_NODE, FIRST As Boolean)
    OUTPUT = ""
    If NODE.LEFT Exists Then
        For Each NODE_ELEMENT As TREE_NODE In NODE.LEFT
            OUTPUT = OUTPUT & IN_ORDER(NODE_ELEMENT, False)
        Next
    End If
    OUTPUT = OUTPUT & NODE.VALUE
    If NODE.RIGHT Exists Then
```

```
            For Each NODE_ELEMENT As TREE_NODE In NODE.RIGHT
                OUTPUT = OUTPUT & IN_ORDER(NODE_ELEMENT, False)
            Next
        End If
        Return OUTPUT
    End Function
```

This takes into account a limitless tree form, where a tree can have any number of children. At the beginning this will of course be binary, as the algorithm assumes each operator can only have two children, left and right.

This can be changed; the operators **+ and *** are able to be applied to many children without the need of order. This is not of great important right now, and will be greatly expanded upon in the design stage.

Nonetheless the reason I showed Inorder instead of any other traversal is due to its inherent property to take in a tree and spit out an infix string. This will allow me to transform an expression tree into a form that humans can understand.

Testing Possible Solutions and Determining their Validity.

The reason I have stated all these different methods is that they all have their own merits. Although I have heavily lied on the idea of trees, I did not come to this conclusion lightly. I designed this program such that it works upon templates. This means I can tailor systems to solve specific equations well, instead of creating a general all purpose 'spit out the answer' type function.

The scale of this program would require me to endeavour in each, REGEX, and TREES. I took the general ideas of these and saw what I could conjure within my means, by **developing a framework to see how plausible my solution would be to develop**. The ideas stated above are the fruit of this, and I have not come without failures.

The original system I proposed to solve equations (or at least simplify them) was a hybrid of these two systems. Say I have the expression –

$$\text{“}(3x\ + 9x^2) * \big(9x - (2y + z)\big)\text{”}$$

What can I do about approaching it? Of course the answer would be a tree.

So, I converted this expression into a tree. Take note that I took each expression as its own object, ie $9x^2$ was kept as a single element in the tree, and not decomposed. This was the first mistake I made in the system.

To continue, with a tree, it would be quite obvious that I could take advantage of another traversal method. **Post Order.**

Post order procures a Postfix expression. This form allows the removal of brackets, and was the method I converted it into to create the expression tree in the first place.

Ie, simply

Infix -> Postfix -> Expression Tree -> Postorder Traversal.

This may see pointless, but to me the value of this traversal was that I could calculate expressions in an order that wouldn't cause any mishaps. I can evaluate an operator, one step at a time.

Such a task requires REGEX. This is partly due to my nodes being able to contain ax^b and because of the nature of this traversal. It is essentially a growing stack. I start off with some simple operation, such as $3x + 9x$ which I can evaluate through regex quite simply. It grows more and more as I go up the tree, evaluating more things.

Say $(12x) + (9z - 2y)$ would be the next evaluation. I will need a system to decompose these elements, as each will be stored as a string. Then I will loop through each addition, removing if added, etc.

The main problem with this design was its inability to scale well. I was able to formulate some pseudo addition simplification. But this was still unable to handle certain scenarios.

Say, what if I had $x^{x\text{^}2}$? My program would not know how to evaluate this if I assume numeric powers. My aspiration was a fully purposed simplifier, as without it many computations, such as rearranging equations, would become very cumbersome.

I am ignoring multiplication and division completely.

The nature of regular expressions made it so that strange bugs would emerge with certain expressions. Over and over again, regex expressions would have to be remade and expanded upon, creating rather unslightly things like

$$(\backslash+|-) * (([1-9] + [0] * [a-z] * ([a-z]|[0-9]))|([1-9]))$$

Its job was to remove zero variables, such as $0x^2$

Eventually I came to the conclusion that this was an uphill battle. I required a much more predictable and non-chaotic solution. What led from this was an entirely Tree dependent system to be set out.

My decision on this has not, however, removed the entirety of REGEX. I understand that I still need to solve questions. Doing this is difficult in any form, but I have templates I can use. I can specialise functions to solve specific forms of problems, and only these problems.

This may be inefficient, but for my goal it is the easiest one to go about. General solutions are difficult, and this isn't a calculator.

Prototype 0.

From my research I developed a prototype to test my theories. This has been documented above, and from it I have obtained a partial framework for algebraic simplifications. I will base my data dictionaries and class diagram on this prototype as it will remain an integral part of my program coming into the fully developed stage.

I also made an outline of my login system, which seemed to be a plausibly solution to develop.

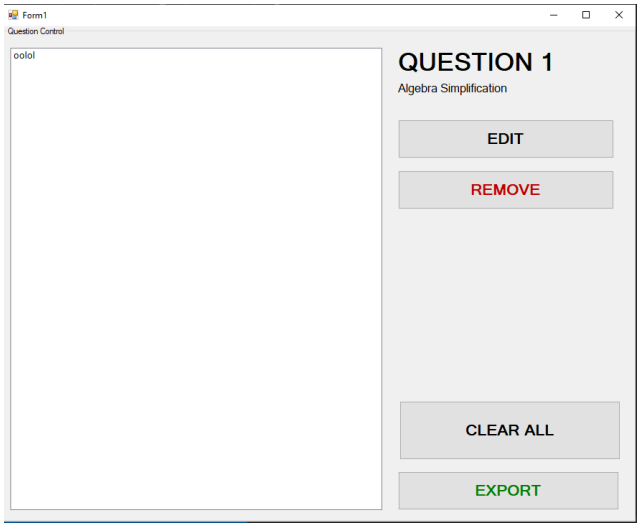
Meeting 27/07/2020.

We held another meeting to discuss the progress I made. On this instance I had decided to design the general layout of the UI, and how the program would function because of it.

I first explained my understanding from my testing of this idea. I said that it would be possible to implement a form of homework solver, however I expressed my concern on my ability to develop more advanced mathematical solvers.

I stated that I will be constraining the templates to a level that would allow me to develop template specific solvers. This came from the difficulty in solving general equations.

I said that for things like simplification, I would most likely be able to develop and all-purpose method. Also, I continued that there may be a possibility that equations, such as $2x - 3 = 9x + 1$ could be possible with a general purpose form. Though I noted that I still haven't delved too deeply in this aspect due to my attention being on expression simplification.



The UI sample I showed.

He said that the UI was simple enough to understand, so I didn't make any large changes.

Business Requirements.

Index	Business Requirements	System Requirements
1	A login system to differentiate students and teachers.	Two read only variables which the program uses to check against the input.
2	A set of question templates that increase in difficulty. They should be made from the start of A level to the hardest areas (calculus is a focus).	A class system where each question template inherits a base question class. Templates are then customised depending on their requirements. • Regular expressions are of great use in more complicated implementations.

		• Although they will be in tree form, sometimes this form may not be the most efficient. • There will also be a likelihood of a clean-up using regex. The tree form will produce some idiosyncrasies that came from the simplification.
3	A basic method of solving algebraic questions. This includes simplification.	A class that contains central functions many other classes may want to inherit. For example, converting text inputs into post-fix; simplifying expressions. • This requires an expression tree • It will be populated with nodes with the operator. • It will start binary but will be simplified extensively to properly use.
4	Design a functional user interface.	I will create illustrator layouts of every window that my program will have. For example, the login page, a question page (and how the customisation is formatted). This will allow me to create an intuitive flow in the entire program.
5 FT*	Create a question file creator area.	Have a system that is able to display all the current Question Templates I have. A dynamic area where you can add questions and delete them is required too. A question editor window will be needed to create/edit questions.
6 FS*	Create a question answer area.	Have a logical system where the students go through each question sequentially. Have the ability to save progress midway. Therefore, when opening question files give the option to also open answer files so they can resume where they left off.
7 FT*	Create a question marking area.	Prompt the user to input an answer file. The program will then display the total percentage correct and each questions' mark. The user should be able to open any question and view what the student inputted and their workings (if there are any). They should also be allowed to edit the marks given.
8 FT*	Allow teachers to export results to a spreadsheet.	Teachers can either export to a properly formatted spreadsheet or create a new one. The results are then placed in with the student name.
9	The program must be easy to use.	Use terms that are intuitive and have a layout that requires no steep learning

		curve. Make bugs and crashes a rarity so as to not elicit frustration from the user.

FS dictates that this BR is for the student module. **FT** dictates the teacher module.

IPSO CHART.

IPSO	Information	Evidence
Input	**Login Information** - Password/Code	**Meeting**
Input	**Answer Information** - Question Answers - Workings Out	**Meeting**
Output	**Question Information** - Questions - Teacher Name - Deadline	**Meeting**
Process	Decode JSON string	**Meeting**
Process	Encode data to JSON	**Meeting**
Process	Convert clean Infix strings into Postfix using Shunting Yard	**Constraints and Limitations**
Process	Evaluate simple expression like $(2x + 5y + 3z + 9x)$	**Analysis**
Process	Clean up infix input strings. This involves changes $3x$ to $3 * x$.	**Possible Backend Solutions.**
Process	Parse postfix expression into an expression tree.	**Possible Backend Solutions.**
Storage	**Store Question Data** Store the data in JSON in a file. Can be encrypted.	
Storage	**Store Answer Data** Store the data in JSON.	

Data Dictionary.

This is a preliminary representation of my data types in my program. It may be modified later on if I need to add new systems.

Backend Math System:

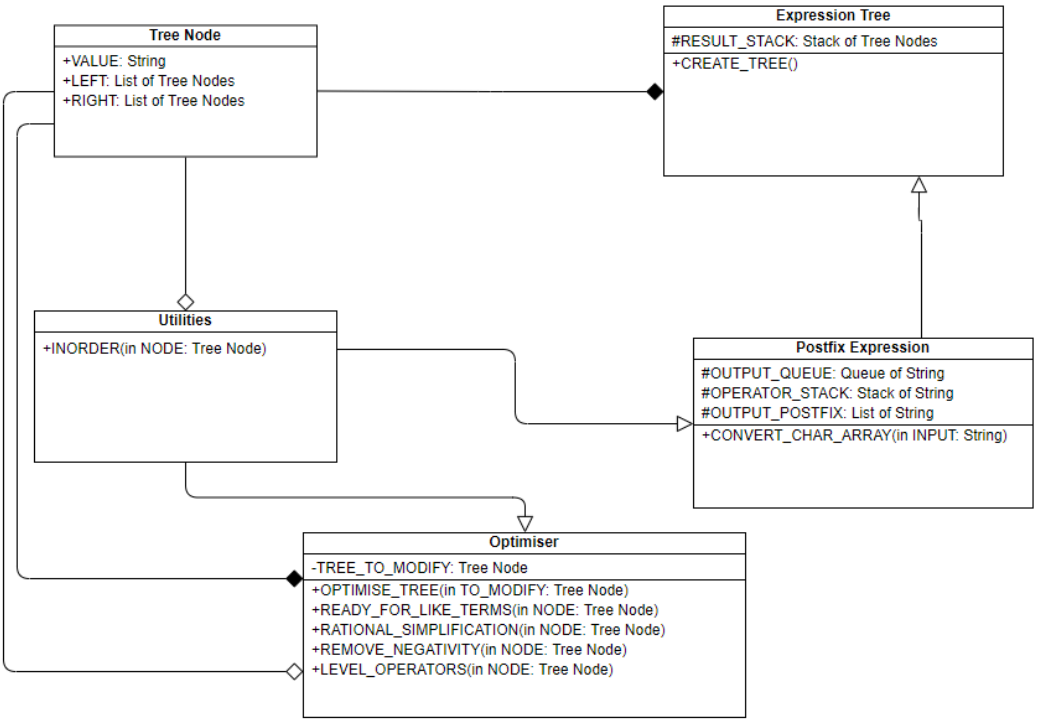
This is for the math system that has been developed pre-emptively to test the validity of my project. It will be integrated with the login system later on in prototyping.

Data Item	Data Type	Validation	Sample Data
*** Expression Tree**	**Class**	**Input must be a postfix string**	**Input: "3 x –"**
+ Result Stack	Stack Of **Tree Node**	Items must be **Tree Nodes**	
*** Tree Node**	**Class**		**Node.Value = "*" Node. Left = {Tree Nodes} Node.Right = {Tree Nodes}**
* Tree Root	**Tree Node**		
* **Postfix Expression**	**Class**	**Input must be a infix string**	**Input: "3 – x"**
+ Output Queue	Queue of String		
+ Operator Stack	Stack of String		
+ Output Postfix	List of String		
* **Optimiser**	**Class**		
+ Tree To Modify	**Tree Node**		

A **'+'** below any **'*'** indicates it is part of the * data type. This is used for **Classes**.

Class Diagram:

This comes from the preliminary testing. This will be a representation of a general system that will simplify tree expressions. This diagram will be expanded upon in the design stage



I have discarded some classes, and one of them used the old method of simplification, where it traversed via postfix and used REGEX. Explained in **Testing Possible Solutions and Determining their Validity.**

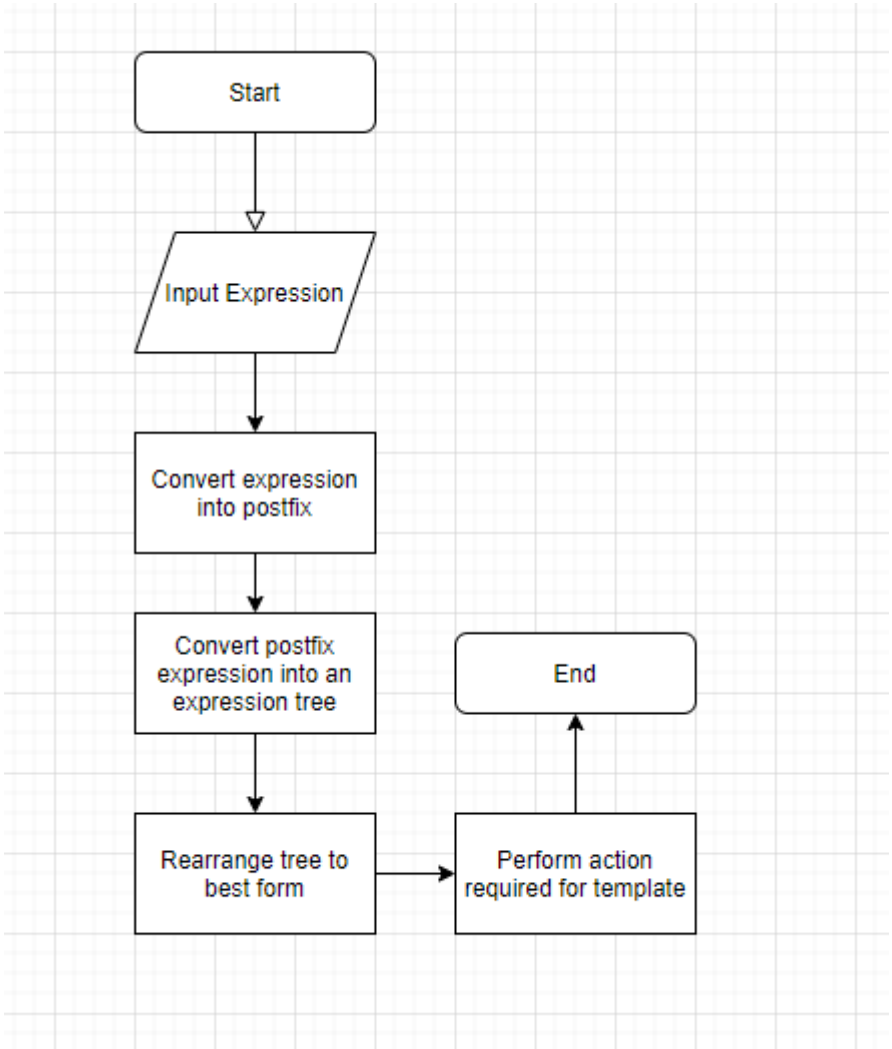
Simple Simplify
+RESULT: String
-REMOVE_ZERO_CLEANUP(in INPUT: String) -RECURSIVE_SOLVER(in NODE: Tree Node)

A discarded Class.

Flow Chart.

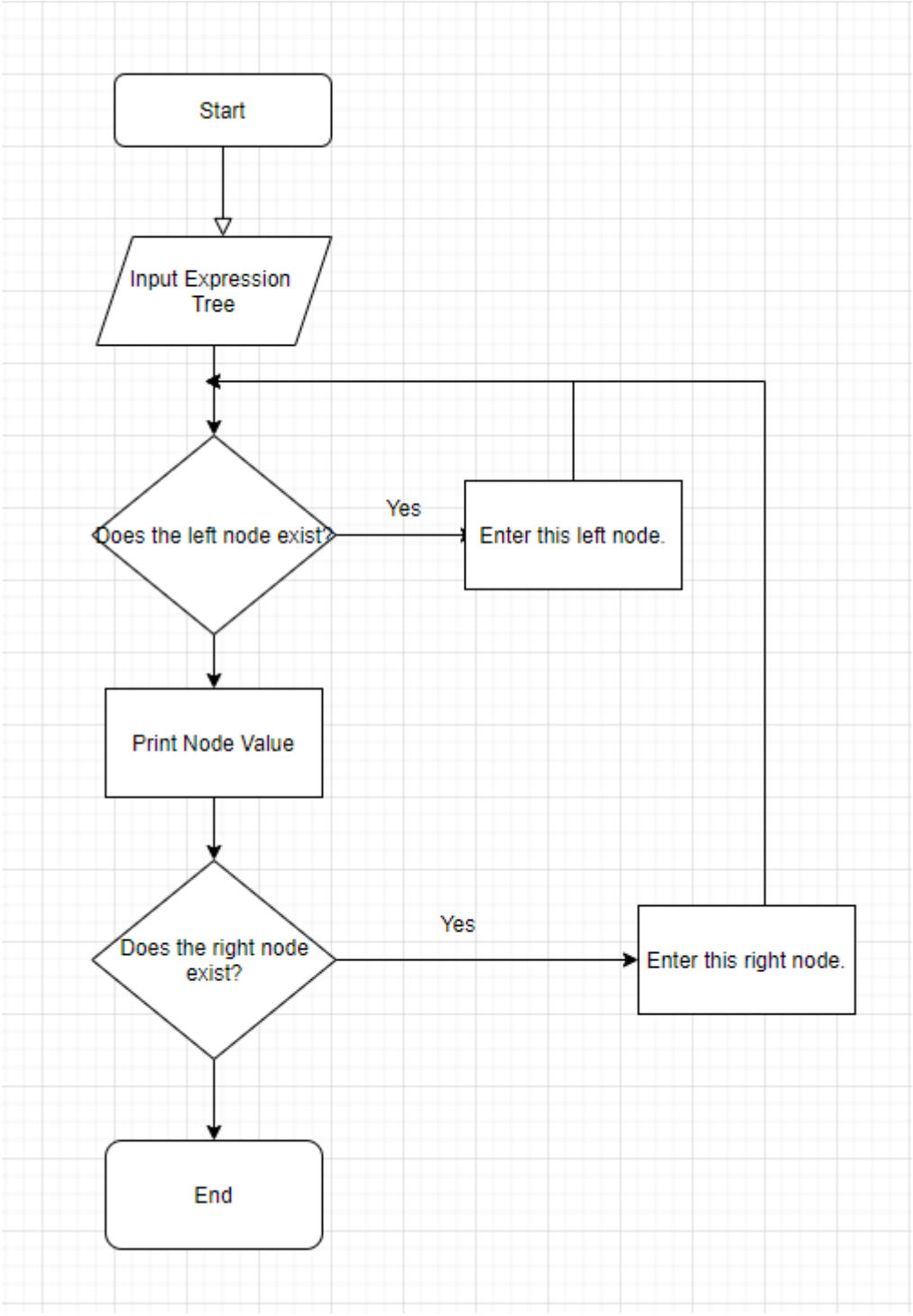
Expression Tree Creation:

This flow chart will represent the high level method in creating an Expression Tree to be used in solving and simplification.



Expression Tree Traversal, Inorder.

This shows the general function for inorder. The other traversals can be simulated if you just moved where the node value is printed.



Stated Language.

I will be creating this program using visual studio and coding in VB.Net. I have moderate experience with VB.Net so I shouldn't have too many issues about implementing my designs—the other options such as C# just seem to have more complicated syntax that I do not wish

to bother with. The IDE is the go-to option for creating windows applications, with world leading debugging and tools.

Prototyping.

I will be setting out prototypes of my project, where I develop part of my program each turn. Due to this my **Design** page will have sub headings determining which prototype they are describing. My Critical Path will determine the order at which I develop for my prototypes, and will be stated as to what objectives are being developed in said prototype.

There may also be amendments to the analysis for feedback, and thus will be noted whenever these are made. Largely this will be for my **BR's so I will name them depending on their prototype.**

DESIGN

Critical Path.

Due to the nature of this project it is vital that I develop the integral objectives before anything else.

This table will list the objectives I will develop first.

Objectives	Comment
3 (Prototype 1,2)	This is the integral system for my project, and the hardest. I will need to design the majority of my project for this part.
2 (Prototype 3)	This is an expansion on the idea where I specialise the systems I devised in the previous objective; this will allow for more specialised areas to be developed.
1 (Prototype 3)	
4 (Prototype 3)	
5 FT* (Prototype 3)	**Although these are integral to my project, they should only be made when there is an underlying system first.**
6 FS* (Prototype 3)	
7 FT* (Prototype 3)	
8 FT* (Not Completed)	**This was not completed due to time constraints.**
9 (Prototype 3)	

The objectives will be marked at what prototype they were developed.

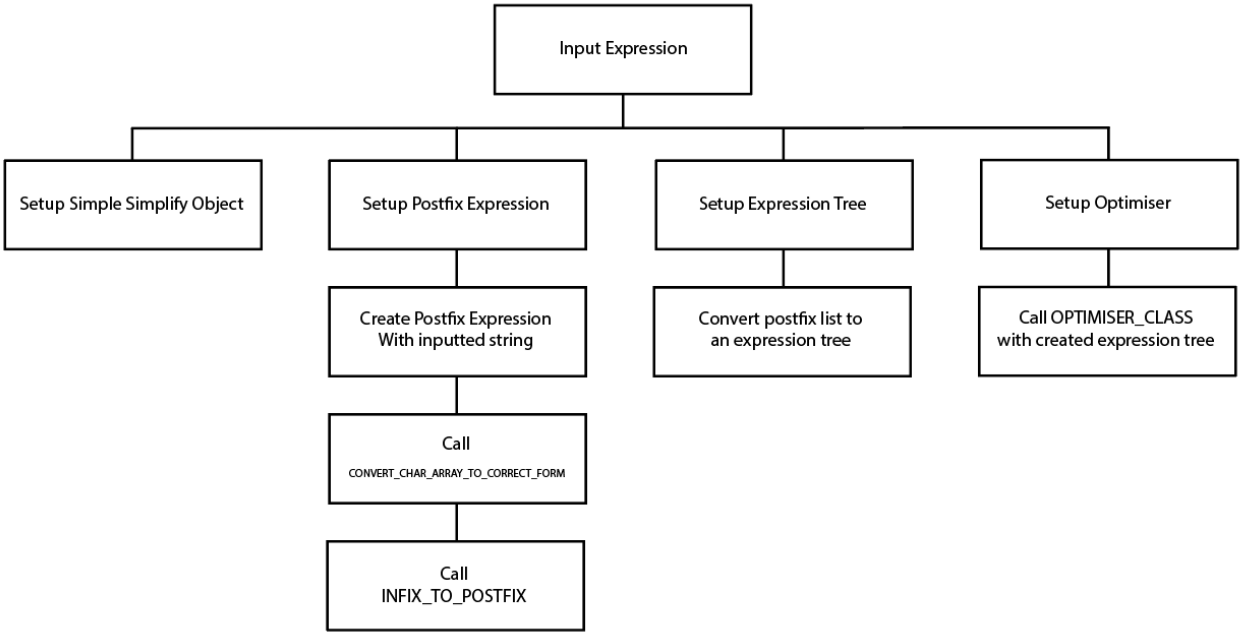
Topdown Design.

There a many classes involved in this program, due to its large scope, so I will do a top down design for each class. I will also name what prototype this design is for, so that it is easy to discern the progress I will have made.

Prototype 1 & 2:

Backend Maths Solver:

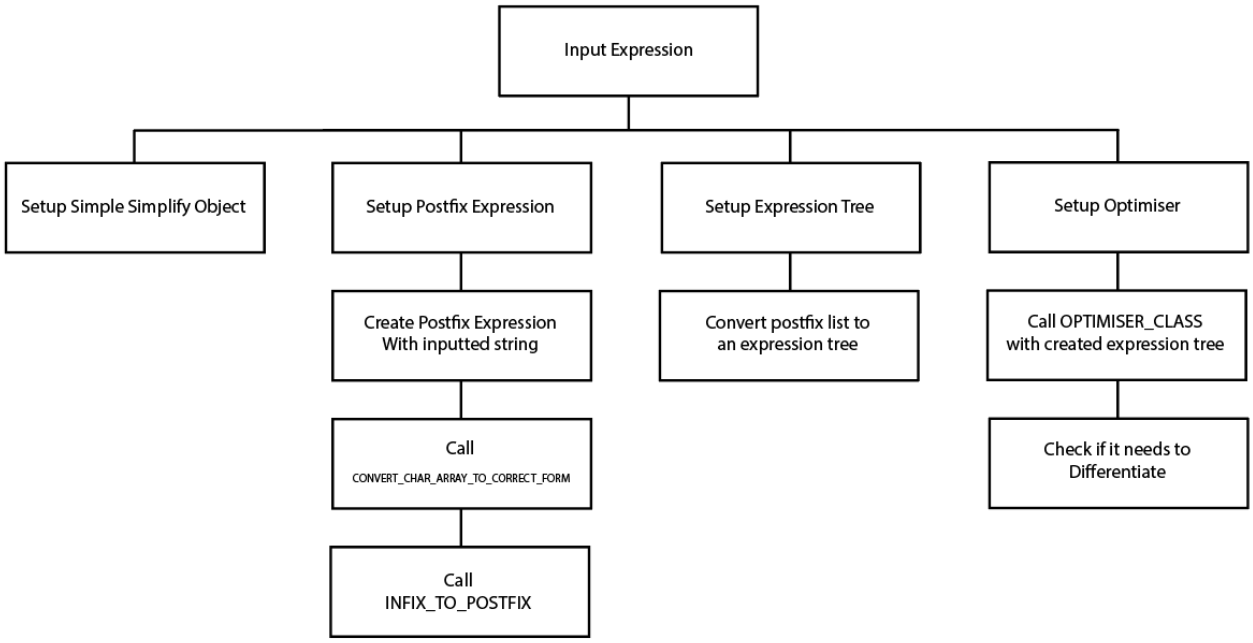
This is what will handle the expressions that will be inputted throughout the program. So whenever you want to simplify a string expression, this is the process created in the first prototype. Although this looks simple, I have abstracted all the functions called in Optimiser.



Prototype 3:

Backend Maths Solver:

This was updated to support differentiation.



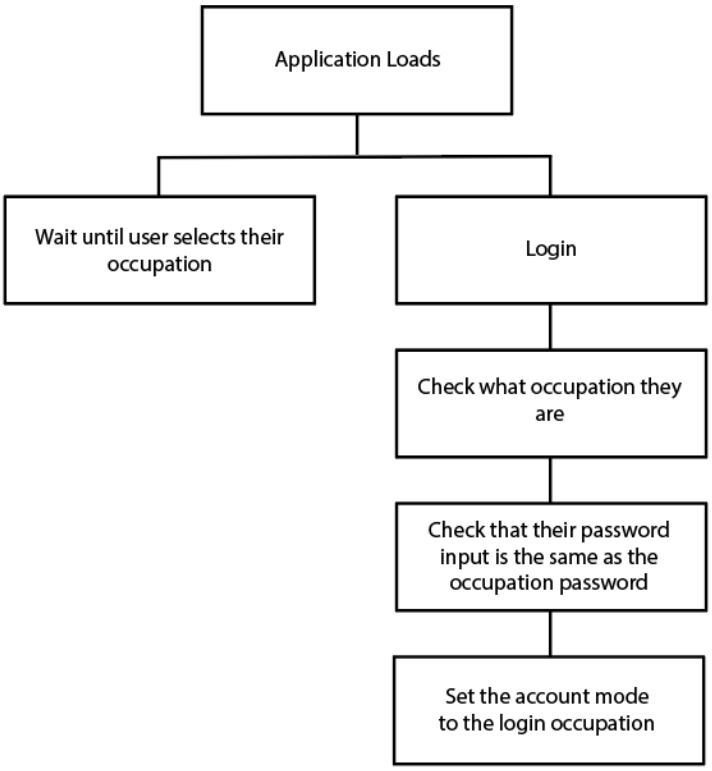
Frontend System:

If you look at the critical path, I have created the entire frontend in prototype 3. This was a moderately easy task, as all it required was me stitching together my backend, and making it function behind many buttons.

The frontend involves a number of things, like JSON export of questions and submissions (3 types to be exact, but use the same system). It also includes many systems for adding questions, login in, the notification system, marking student submissions – etc.

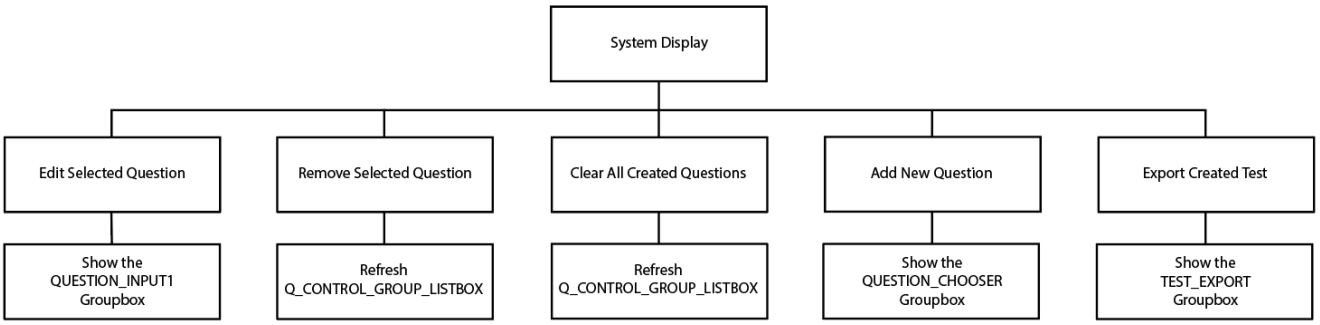
I will go in depth on every single system in the **interface design,** but I'll do a top down for some important systems.

Login System:



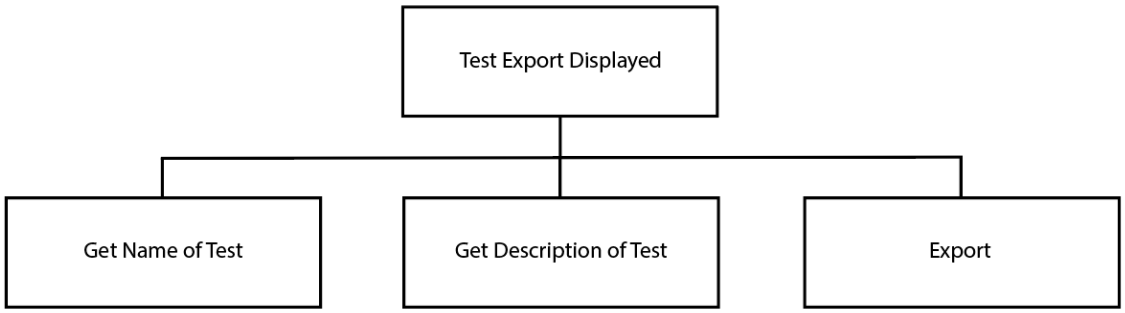
Teacher Question Creator System:

This is the area where the teacher creates the tests for the student to use.



Each of these groupboxes has their own system.

Test Export System:



OOP Class Design.

Backend Maths Solver

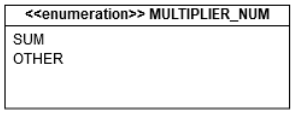
Prototype 1&2:

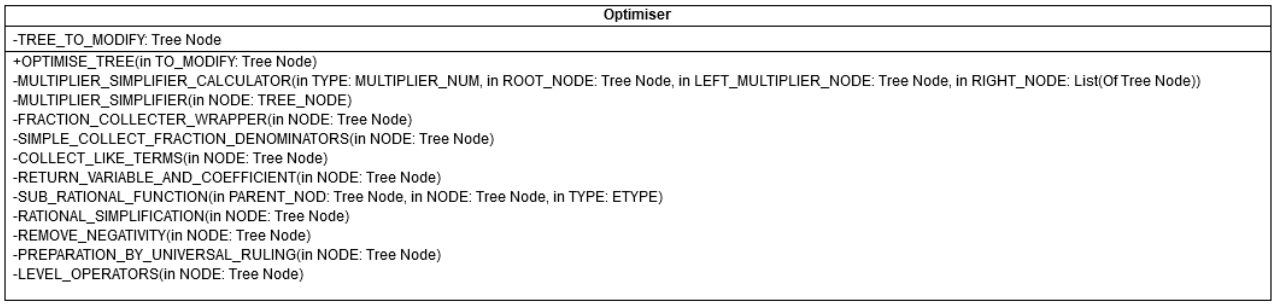
From my analysis I have conjured mostly complete picture on the design of parts of my system. Most predominantly I have ventured on testing the feasibility of **Tree Implementation** and **Dynamic Simplification**. Through testing I found that it was a possibility, and thus conjured a pre-design phase OOP diagram of this distinct part of the program.

The system has not seen a massive overhaul to its overall structure, so I will merely update on a more succinct part of it.

The function ‘Optimiser’.

It is the essential part of the system, where I store numerous functions that look for patterns in the trees.



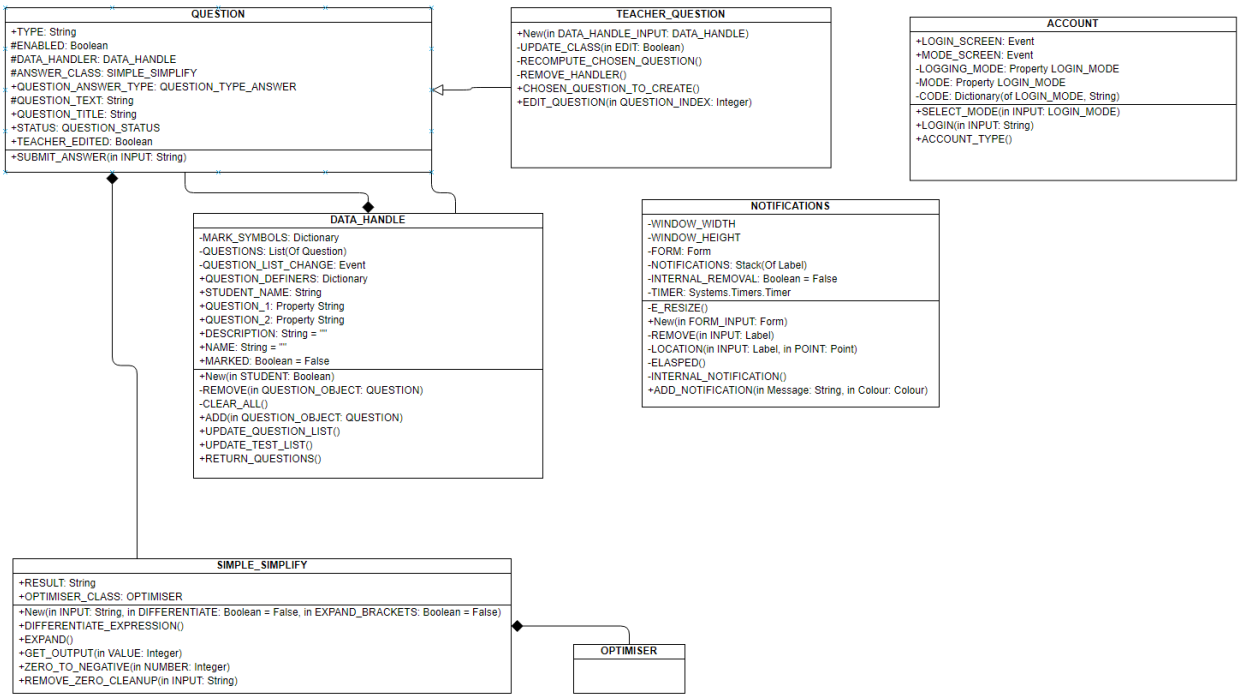


Each function solves a different problem for an expression tree. I merely pass it my newly formed binary tree, and it spits out a simplified version. As this is the design phase there are some parts that are not in complete working order, of which include clean-up.

I have also re-added the **SIMPLE_SIMPLIFY** class on this prototype, as it acts as a wrapper for the other classes.

Prototype 3:

I have added some new classes on this prototype. Namely those for the frontend systems.



IPSO Chart

Prototype 1 & 2:

This will be a general process of the optimiser part of the system.

IPSO	Program Section	Information
Input	Getting User input	TO_BE_SIMPLFIED
Process	Transform User Input into array of expressions.	TRUE_EXPRESSION_LIST
Process	Convert the list from INFIX into POSTFIX	OUTPUT_POSTFIX OUTPUT_QUEUE
Process	Create a tree from the postfix list.	TREE_ROOT
Process	Optimise the tree and simplify.	RESULT OPTIMISER_CLASS
Process	Check if it needs to expand brackets.	OPTIMISER_CLASS
Process	Check if it needs to differentiate.	OPTIMISER_CLASS
Output	The user has inputted the expression.	Output the simplified expression.

Prototype 3:

This has a lot more content in it as it covers the front-end system that just utilises the backend maths solver I created in the first two prototypes. This will mostly be button events, classes that composite the backend maths classes, and special functions like **export** that uses JSON to create files of data to be sent to students/teachers.

However, the content itself is trivial, so I will just show the important functions. Note that a lot of the functions are already shown in the class diagram, so I will mostly only list those in the FORM.vb file, as this is treated as a different type of class (it being the form itself).

IPSO	Program Section	Information
Store	Exporting data from user input.	DATA_HANDLER JsonSerializer Class QUESTION_DATA **SERIALIZED DATA:** **META DATA:** NAME DESCRIPTION STUDENT NAME MARKED **EACH QUESTION:** QUESTION QUESTION TITLE QUESTION TYPE ANSWER TYPE TEACHER EDITED STATUS

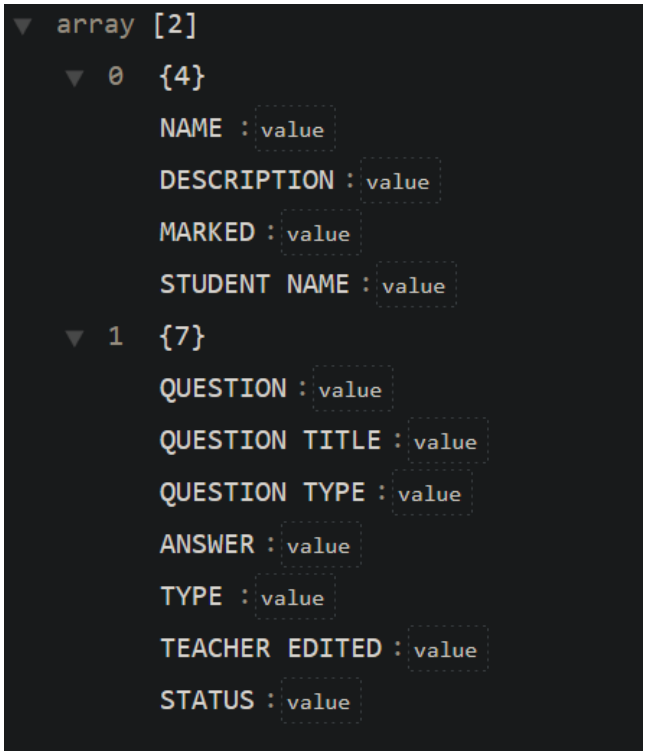
Store	Importing data from user input.	TESTS JsonSerializer Class **DE-SERIALIZED DATA** **META DATA:** NAME DESCRIPTION STUDENT NAME MARKED **EACH QUESTION:** QUESTION QUESTION TITLE QUESTION TYPE ANSWER TYPE TEACHER EDITED STATUS
Process	Add Submission from student.	Add specified submission through the teacher selecting a JSON text file, and then check whether the de-serialized data contains any tests. Loop through each question in the added test, and then mark it as either 'CORRECT', 'INDETERMINED' or 'INCORRECT'.
Process	Create a Question.	If the teacher has selected a question template, change their screen to the question creation groupbox QUESTION_INPUT1.
Output	Add a notification.	Whenever the function is called, create a notification using the class system. Check whether there are less than 6 notifications already before doing so.
Input	Edit a Question	The teacher can edit a group box's text boxes to change the answer and question text. This also includes re-computing the answer via the program's backend system.
Input	Submit Answer	ANSWER ANSWER_CLASS The question class object will have its answer set by the student's input, an then the true answer that it will be compared against is calculated and stored in ANSWER_CLASS.

Record Structure.

As I export some JSON files, I'll show the general files that I expect my program to create. There is no 'size' as the records can be as big, or small, as they want. The teacher can create 20 questions in a test, or 100. The only limit is computer memory.

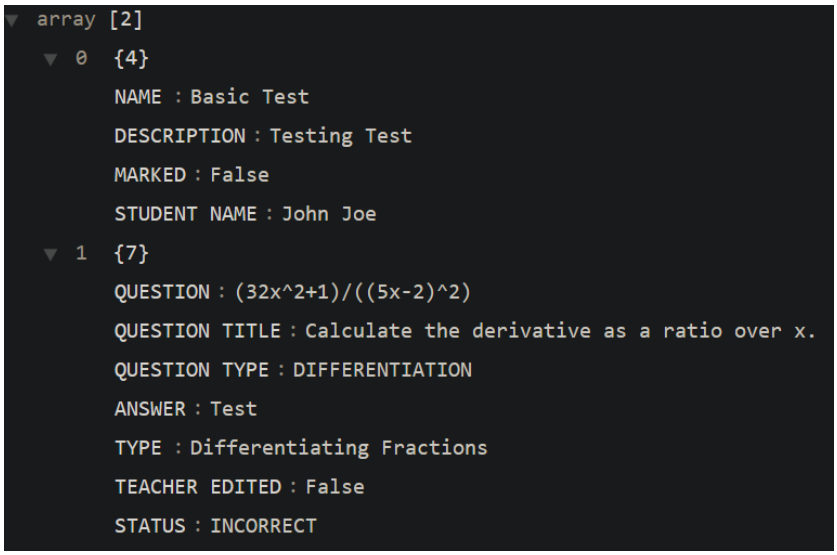
Record	Purpose
Question Export Test (JSON)	This is the test that the teacher will send to their students to complete.
ANSWERS [Student Name] (JSON)	This is the answers that the student will create after they finish doing the test they received.
MARKED ANSWERS [STUDENT NAME] (JSON)	This is the file created after the teacher received the student's submission and marks it with the computer. This can then be returned, and the student can see their score through the program.

There is a general structure to these JSON files, so I'll show the form:



Each of these records uses the same structure.

Here is an example:



Data Dictionary

Prototype 1&2:

I will show an updated evaluation of the backend maths classes created in this prototype.

Data Item	Data Type	Validation	Sample Data
TO_BE_SIMPLFIED	String		“3x-5+2”
TRUE_EXPRESSION_LIST	List(of string)		{“3”,”*”,”x”,”-“,”5”,”+”,”2”}
OUTPUT_POSTFIX	List(of string)		{“3”,”x”,”*”,”5”,”-“,”2”,”+”}
OUTPUT_QUEUE	Queue(of string)		{“+”,”2”,“-“,“5”,”*”,”x”,”3”}

TREE_ROOT	Class(of TREE_NODE)		{
			Value String,
			Left List(of TREE_NODE),
			Right List(of TREE_NODE)
			}

RESULT	String		“3x + -3”

OPTIMISER_CLASS	Class(of OPTIMISER)		{
			TREE_TO_MODIFY As TREE_NODE
			OPERATORS As Dictionary(String, Func(Of Double, Double, Double))
			MULTIPLIER_NUM As Enum
			}

Algorithms.

Prototype 1:

As shown on the **Critical Path** section, the first prototype is designated for development of the core maths simplification class. This class is tasked to apply as much dynamic ability to expression as possible, such as expanding $(2x\ +9)(3x+z)$.

This objective (3) is not clearly cut. I can essentially go as far I want to in the ability of my algorithms. I decided to stop at a point where putting any more time into it would mean the risk of not finishing other core parts of my program.

This may be amended on future protypes, and will thus be listed in the **Critical Path** section. Currently it is able to collect like terms, multiply to a 'degree', and add and rearrange divider expressions.

It may not be clear, but this is not true for but '$2x + 3x$' statements. It is true for $2x^{3x+9x} + 3x^{3x+9x}$, it is true for $2x^{3x^{\frac{9x}{7}+\frac{10x}{7}}+9x^{\frac{9x}{7}+\frac{10x}{7}}} + 3x^{3x^{\frac{9x}{7}+\frac{10x}{7}}+9x^{\frac{9x}{7}+\frac{10x}{7}}}$. And so on.

The power of trees lies therein the fact that it merely looks for patterns that do not change under any circumstance. There will always be a + node to denote the sum of those two variables in $2x^{3x+9x}$. Thus, I can achieve a superhuman feat of simplification.

It is the solution to my originally chaotic solution.

This current prototype also supports pseudo-multiplication.

For example, $(2x + 9)(3x + z)$.

However, it does not solve things like 2*3. In the prototype it will leave all integers alone. This can be seen as a goal to be done in a later prototype, as the problem should not be too remarkable to solve.

Nonetheless the current abilities are fine to implement some trivial questions in the ks4 area.

Most of the tree algorithms were made by me, so functions that were made earlier on in development will show slight differences in technique and method of approach. I intend to explain most of the vital algorithms, such as the method of Collection Like Terms, or Multiplication.

I will split the algorithms into two parts. '**Reorganisation**' and '**Simplification**'. The key difference is in complexity. The algorithms that reorganise division nodes to more standard forms, or make every integer into the form $1 * a$, or every variable x^1, are '**Reorganisation**'.

'**Simplification**' are (in relative terms) complex algorithms that make major modifications. These are things like "Like Term Collection".

Preparation Algorithms

There are also other algorithms that go before the tree stack, such as the shunting yard algorithm. I will explain this too, as much of my tree algorithms have overlapping feature set.

The Modified Shunting Yard Algorithm

Objective 3 is of the intent to create a dynamic solver. It takes an input, say "5x+3x", and spits out "8x". This is simplification.

As for solving equations, say, "$3x + 9 = 2$", the first prototype has not solved. Nonetheless, the foundation has been formed.

This algorithm takes the input and converts it into post-fix (or reverse polish notation). This is a method of maths that retains all the information, but removes the need for brackets. It makes the operators follow the operands.

So $3x + 9 = 2 \Rightarrow 3x\ 9 + = 2$

Where the operands are 3x and 9, and the operator is +.

The reason I do this is so that I can parse the post-fix into another algorithm that will then convert it into an expression tree.

The algorithm works by having an **operator stack** and an **output queue.** The operator stack is made to hold operators that are yet to have been added to the output queue. The output queue, as you can probably guess, will hold the final output in RPN.

To note, before the algorithm is made to run, I must convert my string input into an array of characters. However, this form must then be modified so that terms are properly grouped together. Like {"1","3","x"} -> {"13","*","x"}. This must be done as the algorithm assumes that numbers are properly formatted. It does not know what "13x" means, as there is not a "*" symbol.

This algorithm has some idiosyncrasies that have a tendency to output wrong lists of items. The errors are few and far between, and when it occurs its normally higher up in the stack that causes the program to crash. A note has been put in place to modify it so that it can take all cases of input and properly segregate variables. However, it has a 95% rate of not making the program crash, so for most inputs it will be fine.

The issue is mostly due to the placing of "*" between variables, like "13x", and brackets. This is due to the input not including a * between all coefficients (you input "5xyz" instead of "5*x*y*z").

Assuming this is done, we will now loop through every item in the array.

```
For Each variable in the char array
I will loop through every item in the list of variables.
 Select a token
 If the token is a number:
 Push the token to the output queue
 Else If the token is an operator:
  If the operator stack has items:
   If the operator at the top of the stack is not a "(":
    While (The precedence of the top operator in the stack has a higher precedence) or
(They have the same precedence and the token is left associative)
     Enqueue the output queue with the top operator in the stack
     Pop the operator stack
     If The operator stack is empty:
      Exit the While loop
     End If
     If The operator stack's top item is a "(":
      Exit the While loop
     End If
     End While
```

```
    Push the token into the operator stack.
 Else If the token is a “(“:
  Push it onto the stack
 Else If the token is a “)”:
  While the operator at the top of the stack is not “(“
   Pop the operator stack and add it to the output queue
  End While
 Pop the operator stack (as it has the “(“)
   End If
  End If
End If
```

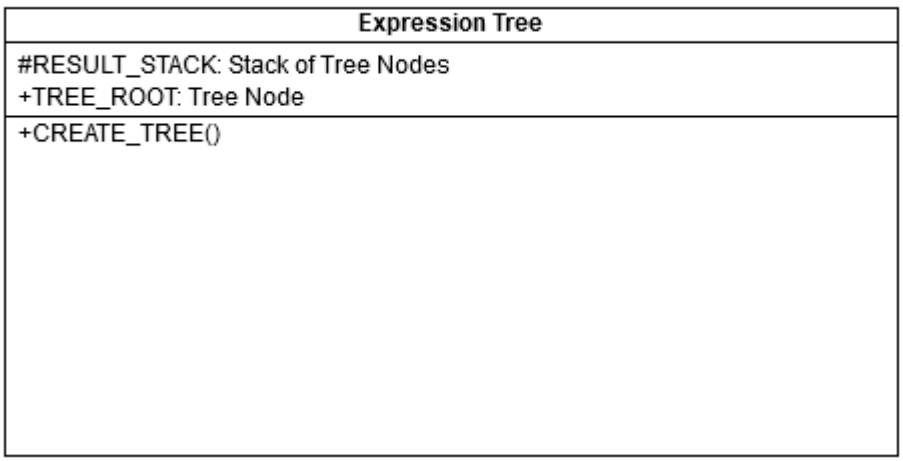
The output queue will need to be iterated over to create a list of items that I can use to create an expression tree. Because it is in postfix the process of making an expression tree is incredibly simple.

Creating an Expression Tree

A tree node is simply:

```
Class TREE_NODE
    VALUE As String
    LEFT As New List(Of TREE_NODE)
    RIGHT As New List(Of TREE_NODE)
End Class
```

I created another class called `EXPRESSION_TREE` which composites the class `TREE_NODE`. And Inherits the class `POSTFIX_EXPRESSION`. It initiates the postfix `sub new()` function that converts the input string into a postfix list. The expression tree class then uses the list of terms `OUTPUT_POSTFIX` that is created in the process within the postfix class.



The algorithm is within the `CREATE_TREE()` subroutine, using the `RESULT_STACK` to output the root of the tree into the member `TREE_ROOT`.

The algorithm’s pseudocode is:

```
For Each item in the list of variables
 If the item is an operand:
  Create a new tree node
  Make the tree node's root node the operand
  Push the tree node into the result stack
 Else If the item is an operator
  Create a new tree node
  Make the tree node's root node the operator
  Pop the result stack and add it to the tree node's right node
  Pop the result stack and add it to the tree node's left node
 End If
End For
Pop the result stack and make the TREE_ROOT variable equal to it. This is the top of the tree.
```

The TREE_ROOT variable will then have the completed expression tree. This tree is binary, and so can be optimised for certain operators.

Reorganisation Algorithms

Preparation by Universal Ruling

There are certain algorithms that work better on the assumption a tree is binary. This algorithm's job is to modify two cases in a tree.

One, $\boldsymbol{x \rightarrow x^1}$

And, $\boldsymbol{a \rightarrow 1 \times a}$

These two cases are used so that there is a universal assumption in the tree. Whenever I encounter a variable, say x, I will always know that it has a power. Whenever I encounter a number, a, I will always know it is being multiplied by 1.

The first case is for collecting powers. It is much easier to design an algorithm that knows it will always have this power, than one that has to check and make special cases for when it doesn't.

The other is more of a specific design decision. But essentially it is so that my later function COLLECT_LIKE_TERMS can assume the form a*b in everything. You see, I designed this function for terms in mind, like $\boldsymbol{2x + 3x}$, so it was built on a system of 'coefficient' and 'variable' (2,3 and x).

For numbers no such thing exists. But if I were to modify them so that they are a*b, then it would be such that the coefficient is the number, and the variable is 1. This makes my life a lot easier, as I do not need to make too many modifications to accept this 'variable'.

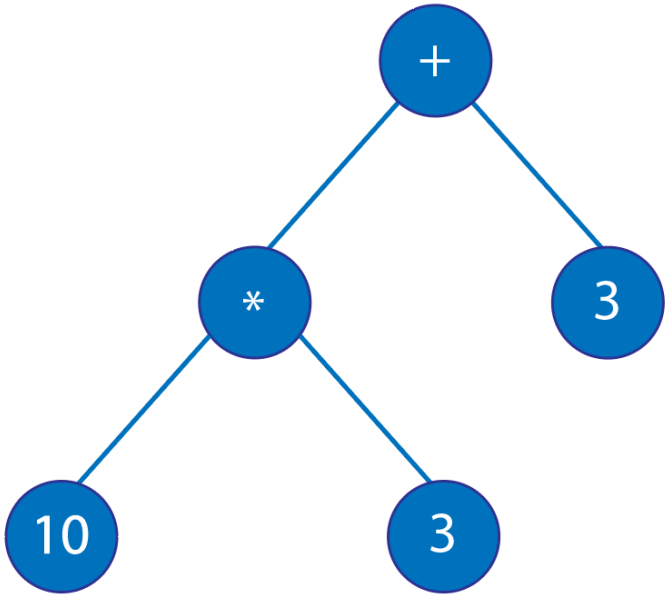
Another note is that these functions/subroutines are run sequentially depending on their use case. This function is run at the start, as other functions will make the tree non-binary.

That out of the way, this algorithm is based on the ‘post-order traversal’. Although it seems confusing, all it really is doing is taking advantage of the execution stack.

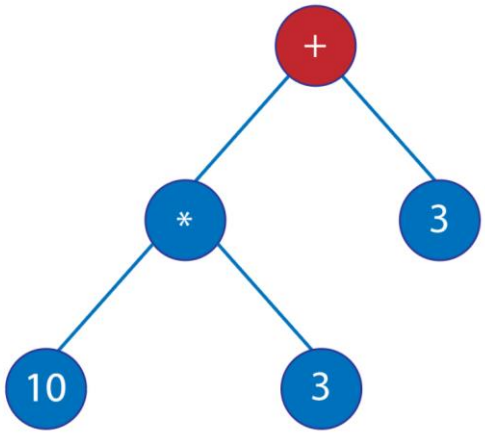
```
Sub PREPERATION_BY_UNIVERSAL_RULING(NODE)
If the left node is not empty:
 For each node element in the left node
  Call the subroutine PREPERATION_BY_UNIVERSAL_RULING with the selected node element
 End For
End If
If the right node is not empty:
 For each node element in the right node
  Call the subroutine PREPERATION_BY_UNIVERSAL_RULING with the selected node element
 End For
End If
```

The first part of the subroutine is essentially the traversal part. As most of my algorithm’s base themselves off this concept, I will explain its use case here and then assume it is understood elsewhere.

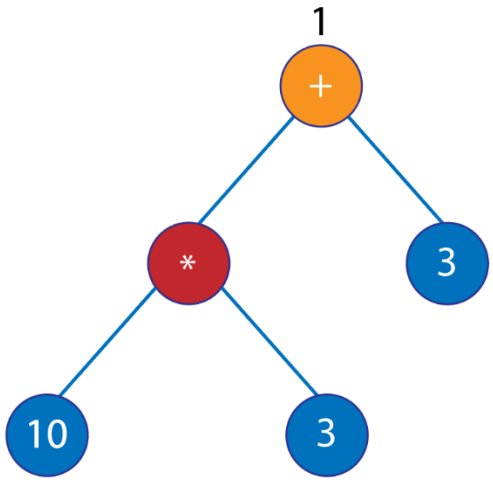
Say you have this tree ->



I start with executing the subroutine with the root node “+”.

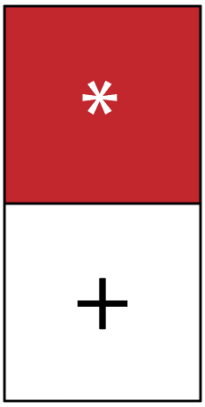


As shown in the pseudocode, it will then loop through the elements in its left node. There is only one, and it’s the “*”. Now, it will now execute the same subroutine, but this time with the “*” node as the argument.



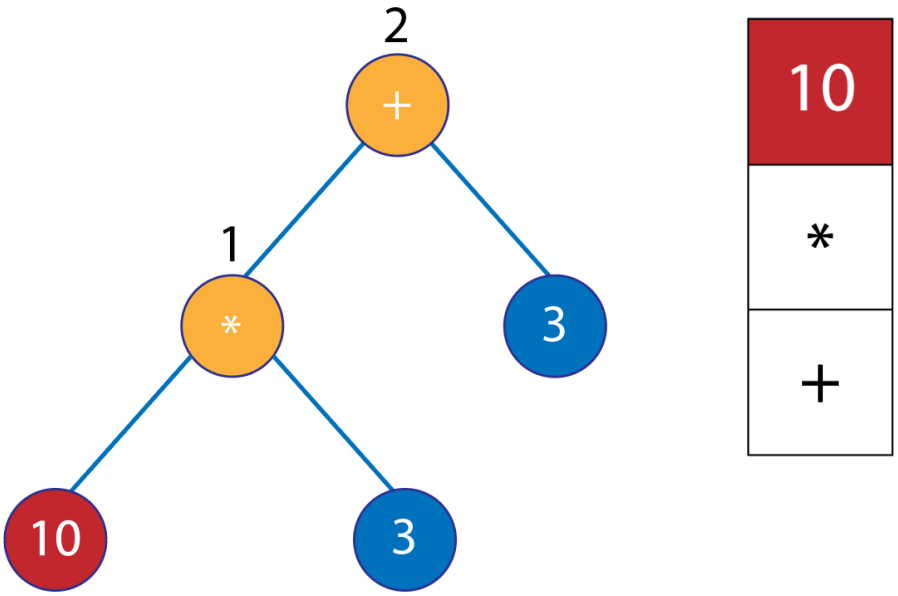
There is now what we call a stack of states. The “*” node is at the top of stack, and thus its state is running. The yellow is the “+” node, with a 1 at the top. This means it 1 below the top of the stack, and must wait for the stack state above it to finish and pop itself.

It can be imagined as

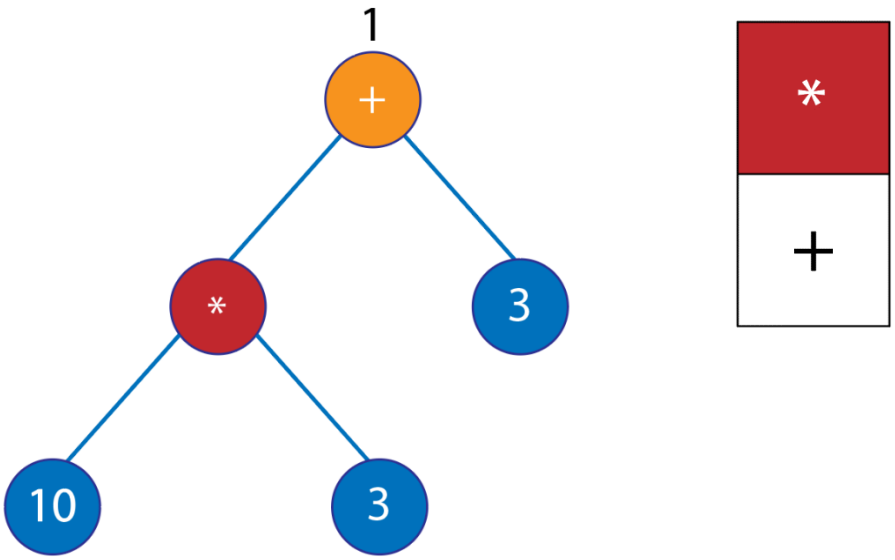


The top of the stack is always the current subroutine running. The reason it is called a stack of states is that when the top item is popped, the next item returns to where it left off. It was merely paused and waiting for itself to become at the top of the stack.

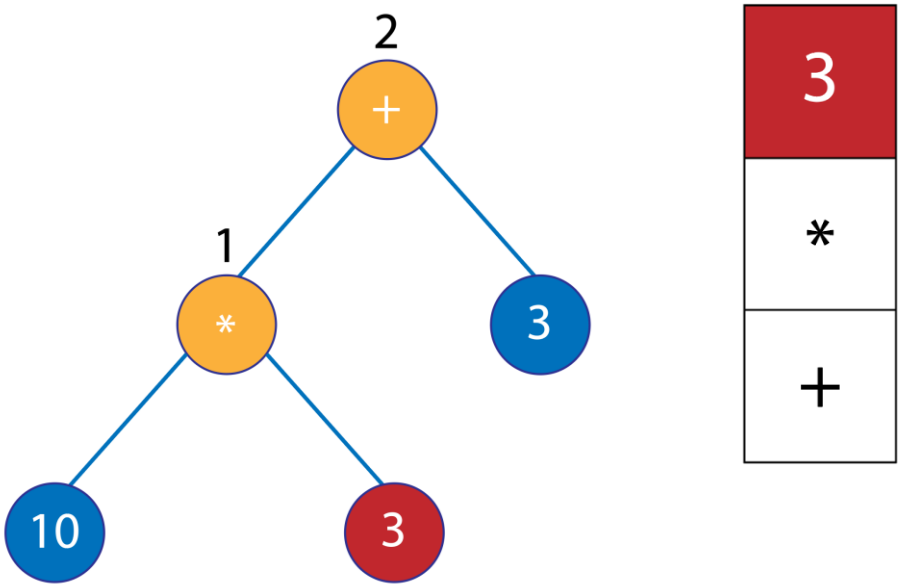
So, in the current state we now have ourselves looping through the "*" nodes left nodes. The subroutine finds a new node "10" and thus executes the `PREPERATION_BY_UNIVERSAL_RULING` subroutine and adds a new level to the state stack.



On this occasion, looking through the left nodes of "10" will yield no results, and so will be looking through the right. Therefore, the subroutine will eventually end and have done nothing. The top of the stack will be popped, and the state below it will resume its operation.



Now that it has resumed its state, it continues to loop through the left node, but finds nothing else. So, dithering on, it now beings looking through the right node. It finds the “3” node. Once again, it executes the `PREPERATION_BY_UNIVERSAL_RULING` subroutine, but with the “3” node as the argument.



The subroutine evaluating the node “3” will find nothing, so will end and the stack will pop. Thus, returning back to the “*” node.

This traversal will visit every node, left, right, and then root. This algorithm seems rather useless, but the use case comes in the “root”. After I have evaluated both the left and right nodes in the same state, I can then look at the root node.

Patterns, it is patterns that you wish to find. If you haven’t noticed it by now, the traversal will only stop adding to the state stack once it finds a node with no left and right nodes in it. Ie, it works from the bottom to the top.

In the example above, I have now returned to the “*” node. It will now try to find more right nodes, but finds none. So, it leaves the for loop.

It would normally stop executing, but I could add an if statement that checks that the root node is a “*”. And then I can maybe try evaluating the children of this root node, 10*3, and replace it with the number 30.

This will work, and apply this rule to every single node in the tree.

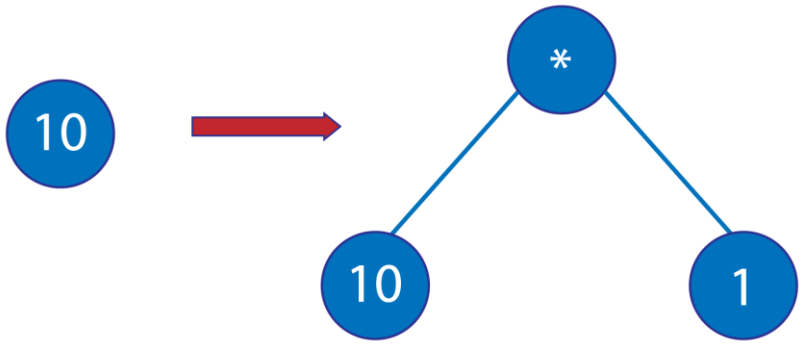
For certain cases there a more efficient ways to approach recursion, especially if it involves “+” nodes, but this is the basic premise to it that is fundamental to any approach. Post order is not a requirement either, as different traversals are sometimes more suitable; if I wanted to evaluate the tree top to bottom, then I could try root left right. It depends on what I want to check, but most of the time any can work.

Back to the function at hand. Remember that I am assuming the tree to be binary, so each list of tree nodes for left and right will only contain 1 or 0 children.

The implemented pseudocode after the traversing is (for the left node but assume it does it for the right also)

```
If the left node is numeric:
  Create a new node and a one node
  Make "1" one node's root
  Make "*" new node's root
  Place the one node into new node's right list
  Add the left node into new node's left list
  Remove the left node.
  Add new node in its place
 End If
```

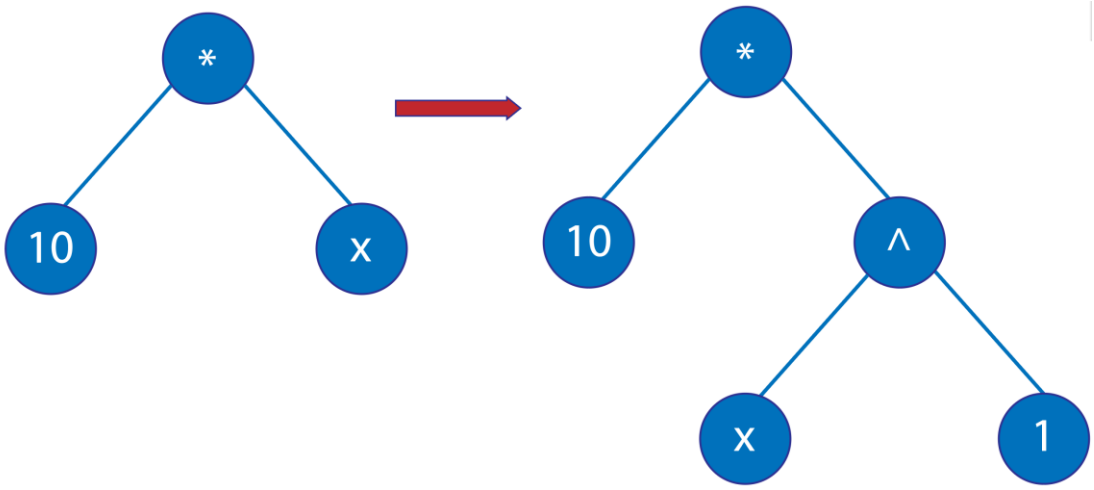
This check can be surmised with the example:




```
 If the root node is a "*":
   If the left node is a variable:
    Create a new node and a one node
    Make "^" new node's root
    Make "1" one node's root
    Add the left node to the left of the new node
    Add the one node to the right of the new node
    Remove the left node
    Add new node in its place.
   End If
End If
```

All variables (13**x**) are in a * node. So, I check for a * node.

The check can be surmised as:



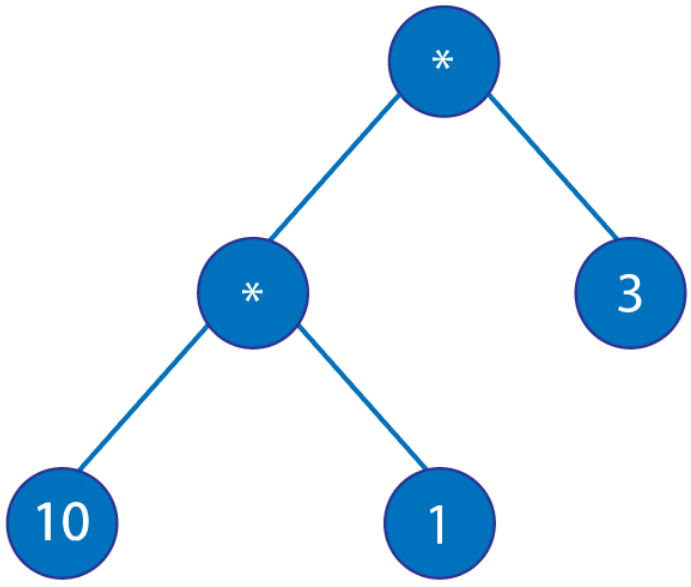
Level Operators

Parsed expressions are always binary, due to the assumption that all operations can only take two parameters. However, this is not true, as the operations "+" and "*" can be used on multiple operands. These are commutative, ie a*b=b*c, so this algorithms job is to level the situations where "*"/"+" nodes are children of each other.

Please note that the "-" node can technically be made into a "+" node, and is an algorithm called **REMOVE_NEGATIVITY** in my technical solution, but I wont go into it as all it does is:

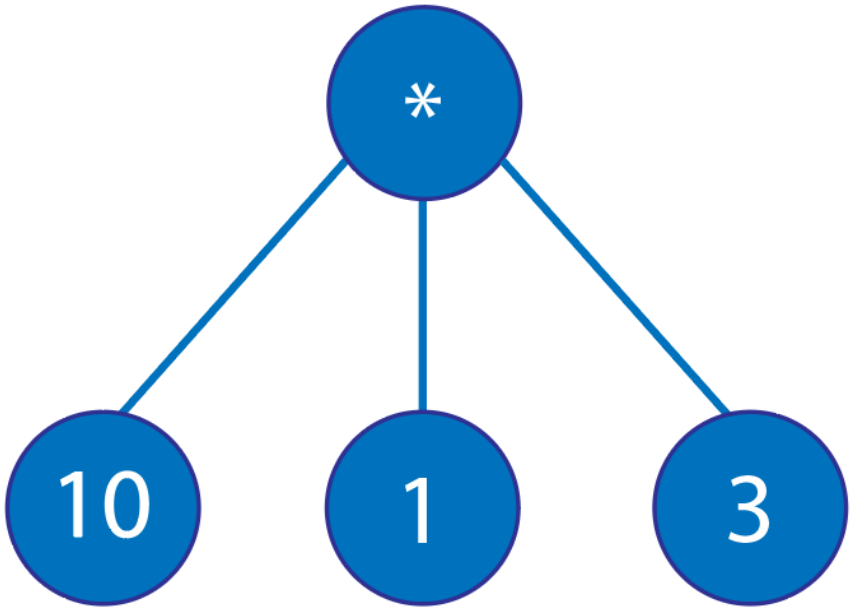
$$1 - 2 \rightarrow 1 + (-1) * 2$$

So, as both terms (+ or -) will essentially work the same way, I'll show an example for "*" nodes.



This situation, currently, says that we must first calculate $10 * 1$, and then $(10 * 1) * 3$. This is not required, as anything with $(a * b) * c$ can be simplified to $a * b * c$, or $b * c * a$, or any permutation of that.

So I can rewrite this as:



This application is useful because, when I need to explore algorithms relating to * nodes, sometimes it is much more useful to have the assumption that they are all under one node.

Anyway, to approach this I will be using a post order traversal, ie left right root. The method at which I approach it is a bit novel, but it isn't too bad.

For the left node –

```
Function LEVEL_OPERATORS(Node)
 If the left list of nodes contains nodes:
    For each left node in the current node:
     Create a new node that is pointing to the left node I'm on
     Call LEVEL_OPERATORS with new node as its argument.
     Make the variable type return the value returned by LEVEL_OPERATORS
     If type return is not nothing and type return is the Node.Value:
      Create a new list of nodes called to_move_element
      Add the left nodes of new node to to_move_element
      Add the right nodes of new node to to_move_element
      Remove the left node I'm on in the loop
      Add the nodes in to_move_element to the left list of nodes.
     End If
```

```
    End If
 End If
```

The next if statement is basically the same but looping through the right list of nodes.

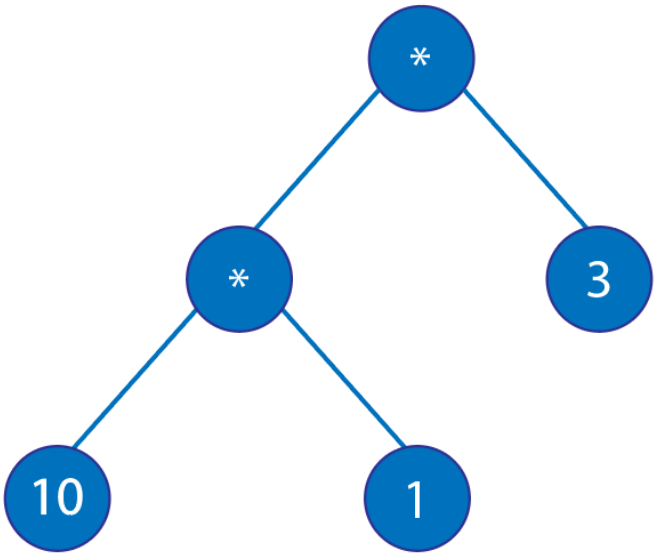
```
If the right list of nodes contains nodes:
    For each right node in the current node:
     Create a new node that is pointing to the right node I'm on
     Call LEVEL_OPERATORS with new node as its argument.
     Make the variable type return the value returned by LEVEL_OPERATORS
     If type return is not nothing and type return is the Node.Value:
      Create a new list of nodes called to_move_element
      Add the left nodes of new node to to_move_element
      Add the right nodes of new node to to_move_element
      Remove the right node I'm on in the loop
      Add the nodes in to_move_element to the right list of nodes.
     End If
    End If
 End If
```

After this is the important bit.

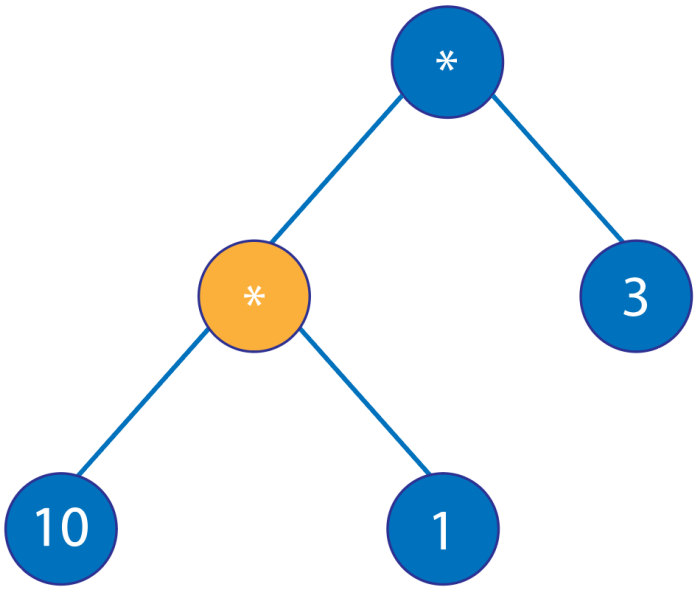
```
 If Node.Value is a "+" Or Node.Value is a "*":
  Return Node.Value
 End If
 Return Nothing
End Function
```

The basic premise of this algorithm is that it loops through all the nodes in the left and right list of nodes, calls the function itself on each of these nodes (thus becoming recursive) and then utilising the way it returns to work out if it has a * node in its own * node.

The “important bit” is key to this, as say we have

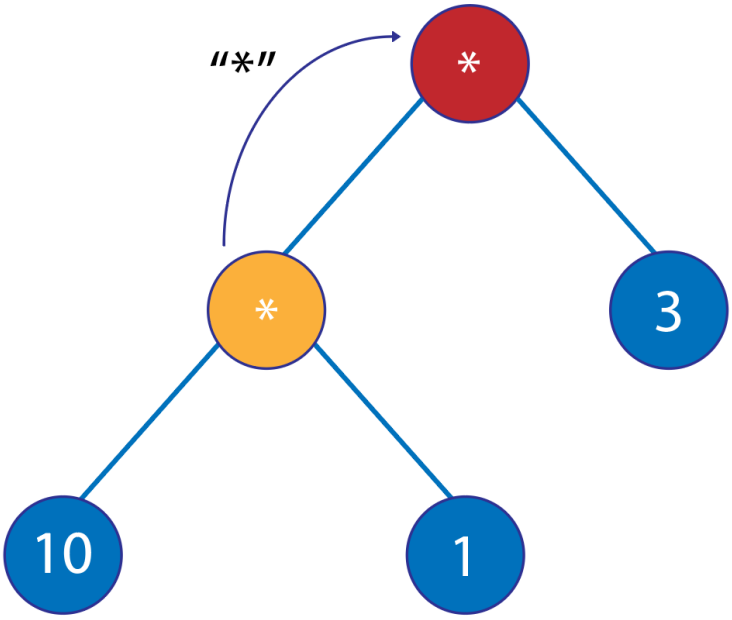


And we are at:



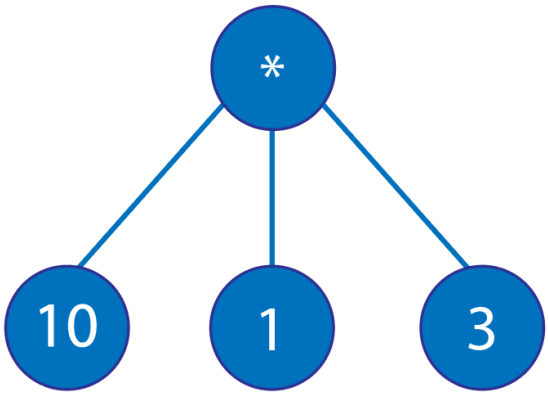
In this situation where the stack is on the child of the root “*” node, this child will return “*” … will it not? The if statement will find that it has “*” node as its value.

I.e., the situation is



The recursive self-call, the variable **type return**, will become the value "*", and this will be the same as the node "*", so the algorithm will say that it can combine these two nodes. It will do this for the right nodes as well, ultimately landing on all the nodes being levelled.

Like this:



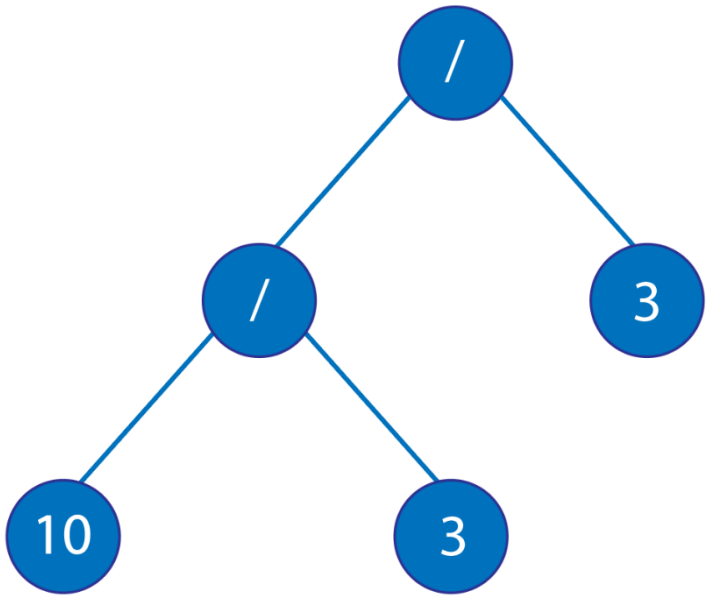
Rational Simplification

This will be the last algorithm I cover for this section, and it is ultimately the most expansive for these methods. Its aim is to cover 3 'checks' in the tree, and simplify them to their proper form.

I will list out the checks for this algorithm:

1. When the division nodes numerator is a division node
2. When the division nodes denominator is a division node.
3. When the child of a multiplication node is a division node.

For the first check, a simple way to surmise it is to imagine this tree:



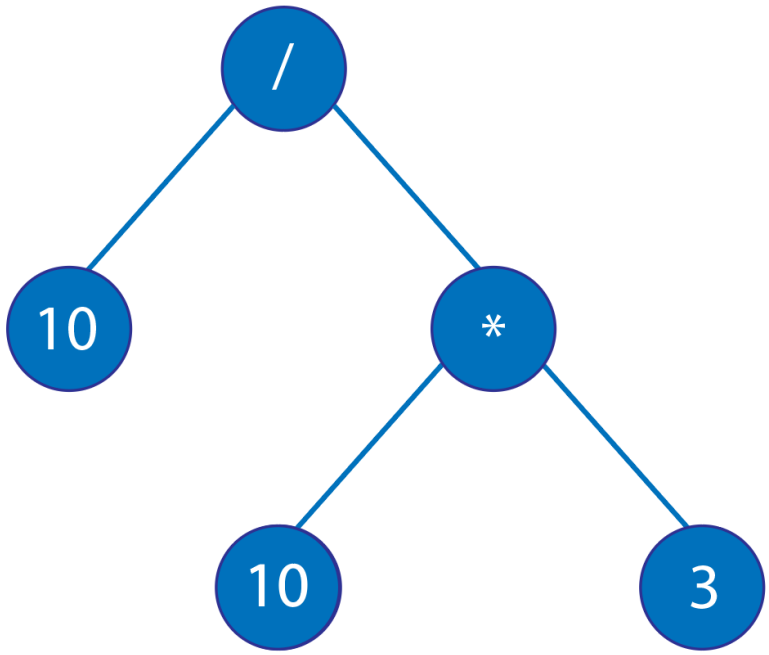
This tree can be written as:

$$\frac{\frac{10}{3}}{3}$$

However this is not of proper form. A divisor divided by something is merely equal to the numerator divided by the product of their two denominators. It would be written as:

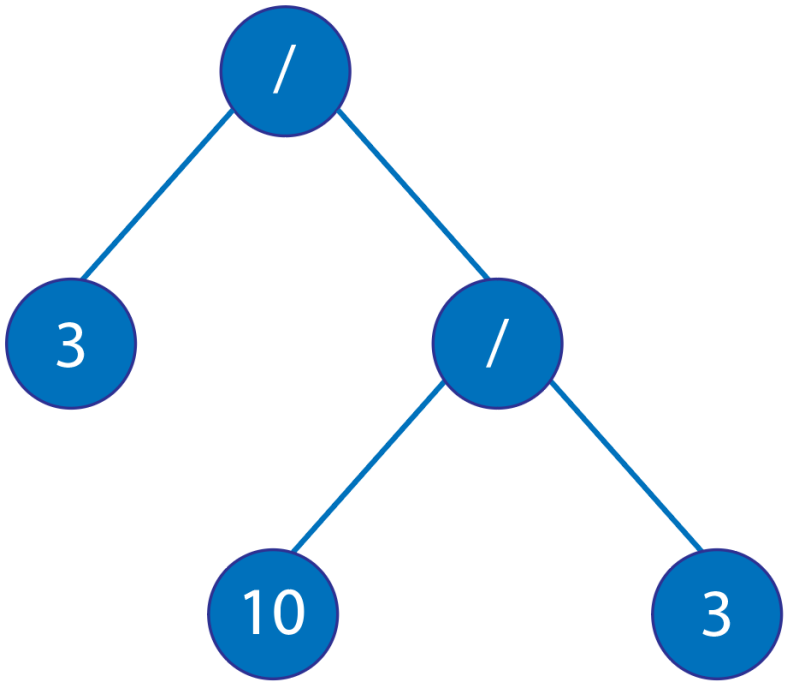
$$\frac{10}{3} \times \frac{1}{3} = \frac{10}{3 \times 3}$$

This in tree form would be:



The first check aims to achieve this.

For the second check, it would be:



You would write this as:

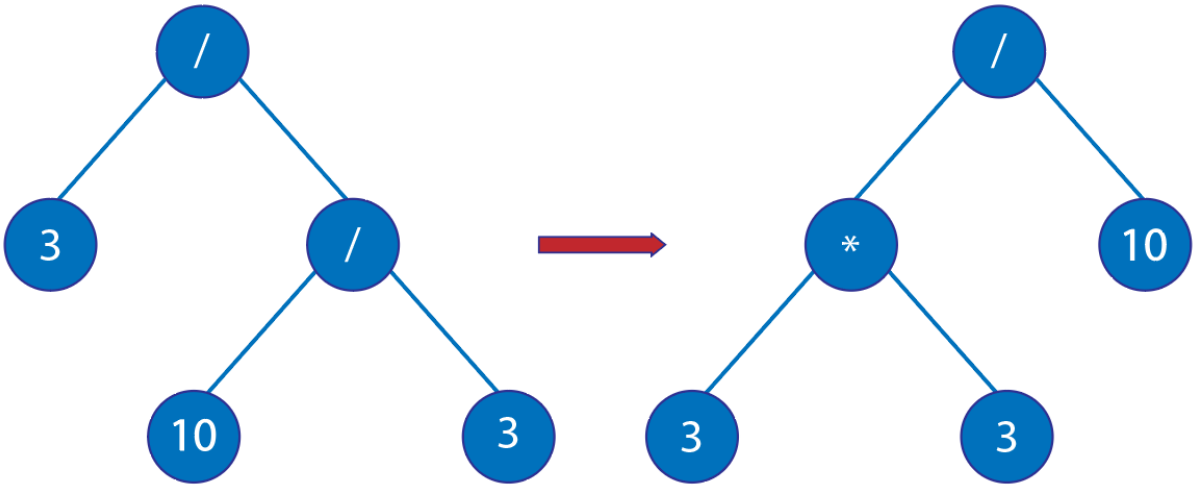
$$\frac{3}{\frac{10}{3}}$$

This is not the proper form. The rule for this is that you flip the division node that is in the denominator, and then times it by the numerator. So:

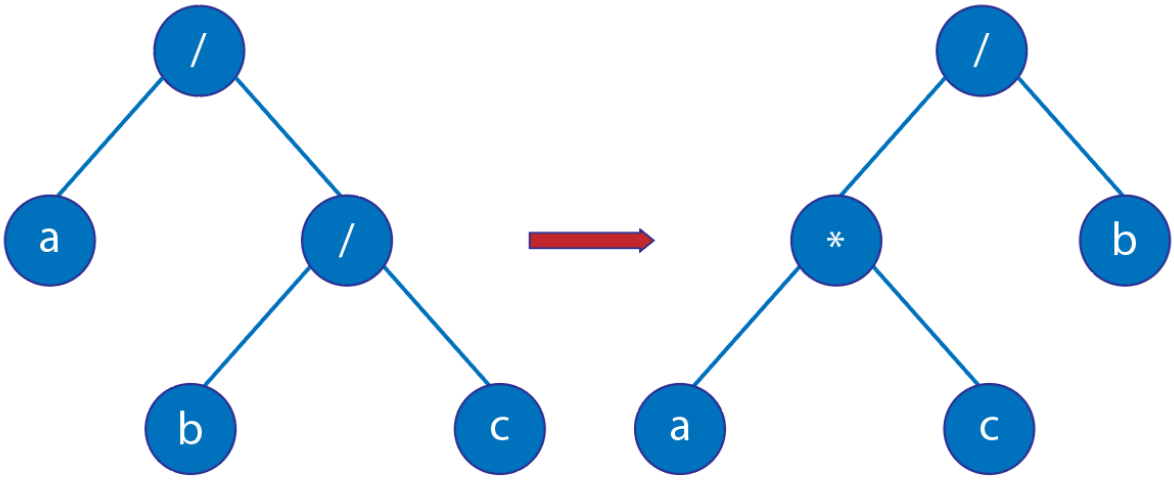
$$\frac{3}{\frac{10}{3}} = 3 \times \frac{3}{10} = \frac{3 \times 3}{10}$$

The result means that you should times the numerator and denominator of the denominator divisor node, and have this over the numerator of the denominator divisor node.

So, the example would be:



Or generalised:



For the third case it can be seen as the easiest to unravel. In cases where you have a multiplication node with a divisor in it, you can put the multiplication node into the divisor's numerator.

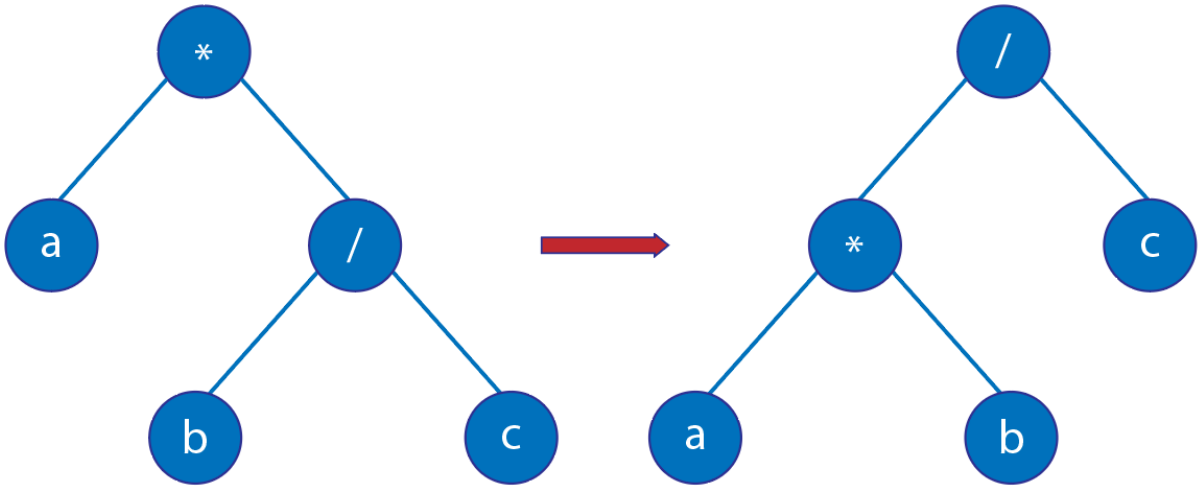
Say we have:

$$3 \times \frac{10}{3}$$

This can be turned into:

$$\frac{3 \times 10}{10}$$

So, the algorithm is:



This form is easier to work with when designing later algorithms, thus the case. I won't write the pseudocode as it is rather long winded and just reiterates what I said. The technical solution will contain all the algorithm's code with comments.

Prototype 2:

Preparation Algorithms

Collect Like Terms

This is an algorithm that executes a simple concept. Say I have $a \times b + c \times b$, then the result can be expressed as $(a + c) \times b$. To properly execute this in tree form, I required the reorganisation algorithm `Level_Operators`. This is because this made all "+" nodes have no other "+" nodes within them.

So, I merely need to loop through the items within the "+" node, and compare them to each other. Another small preparation was to make every non-variable a (like 5) into $1 \times a$ of that form. This is because this function assumes that every child of a plus node has a form a*b, and numbers inherently are not * nodes, so I must give them multipliers for the algorithm to remain happy (and me not needing special cases).

The algorithm is broken down into 2 functions, one that returns the variable and coefficient of a given node, and the other that use this function to compare nodes in a given plus node.

First of all, lets break down the `RETURN_VARIABLE_AND_COEFFICIENT` function. This is what will return the variable part, and the coefficient part (Like 50x, and it returning {50,x}).

The function uses a structure called SEPERATED_TERMS which has the form:

```
Structure Separated Terms
 Coefficient as Integer
 Variable as Tree Node
End Structure
```

This will be used in the function:

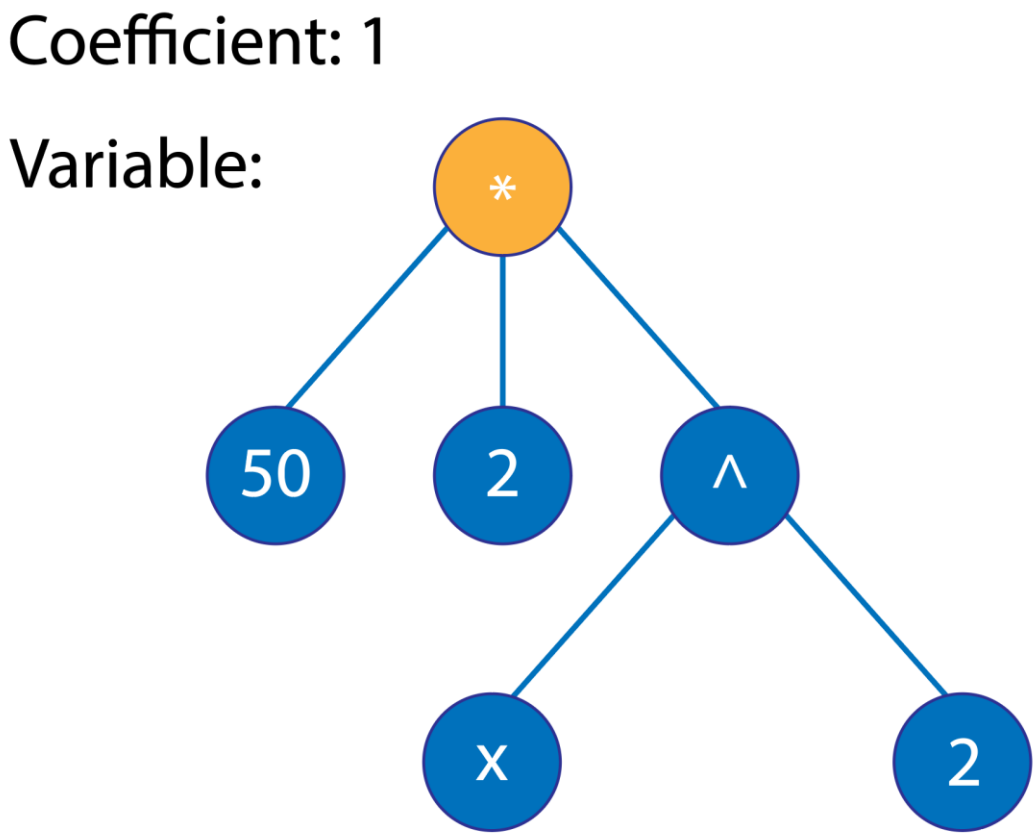
```
Function RETURN VARIABLE AND COEFFICIENT(Node)
 Create a new variable Coefficient and make it 1
 Create a new variable Return Structure and make it a Separated Term

 If the Node.Value isn't a "+":

   For each Tree Item in the Left node do:
    If the Tree Item isn't a "^" node:
     Create a new variable Node Element_S and make it a Separated Term
     Call function RETURN VARIABLE AND COEFFICIENT with parameter Tree Item
     Make Node Element_S equal to what this function returned.
     Create a new variable Node Element and make it a Tree Node
     Make the variable Node Element equal to the Variable part of Node Element_S
     Times the Coefficient variable by the Coefficient in Node Element_S
     If the value of Node Element is a number:
      Remove the Tree Item from the left node
      Times the Coefficient by the value of the Node Element
     End If
     If the node is a negative times node:
      Remove the Tree Item from the left node
      Times the Coefficient by the negative number in the times node
     End If
    End If
   End For
```

```
For each Tree Item in the Right node do:
  If the Tree Item isn't a "^" node:
   Create a new variable Node Element_S and make it a Separated Term
   Call function RETURN VARIABLE AND COEFFICIENT with parameter Tree Item
   Make Node Element_S equal to what this function returned.
   Create a new variable Node Element and make it a Tree Node
   Make the variable Node Element equal to the Variable part of Node Element_S
   Times the Coefficient variable by the Coefficient in Node Element_S
   If the value of Node Element is a number:
    Remove the Tree Item from the Right node
    Times the Coefficient by the value of the Node Element
   End If
   If the node is a negative times node:
    Remove the Tree Item from the Right node
    Times the Coefficient by the negative number in the times node
   End If
  End If
 End For

End If

 Make the Return Structure's Coefficient equal to the variable Coefficient
 Make the Return Structure's Variable equal to Node
End Function
```

This algorithm has a pretty simple method in solving this. Say we have a "*" node as the input for this function. The Coefficient and Variable will represent the values for the selected node. I.e., for the certain function that is looping through the yellow node.

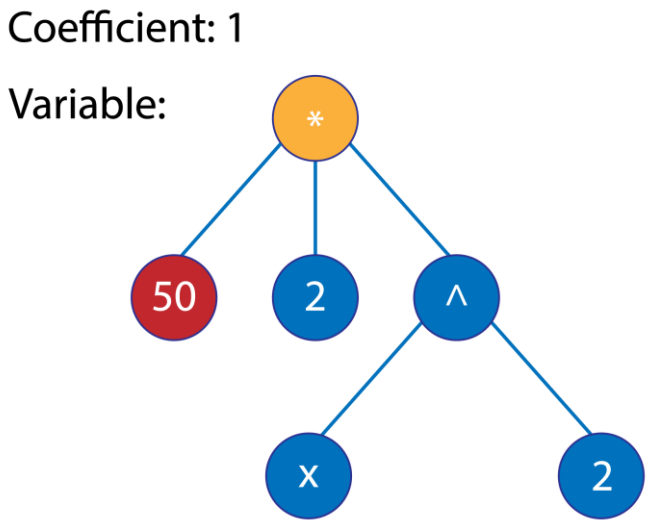
It is good to mention that the modifications made to the node is permanent. The function intends to remove all the coefficients from the node, and then return that with a number that is the coefficient. This means that those that call the function must use a clone if they need the node for other things.

So, let's input $50 \times 2 \times x^2$:



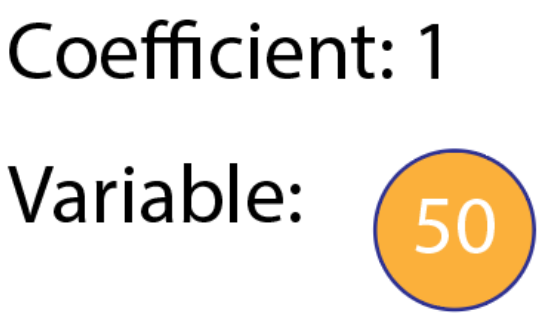
(Note that I am emitting the * node that would normally represent 1*x as it is not relevant to this node design)

We are starting at the "*" node, so the first node it loops to will be "50".



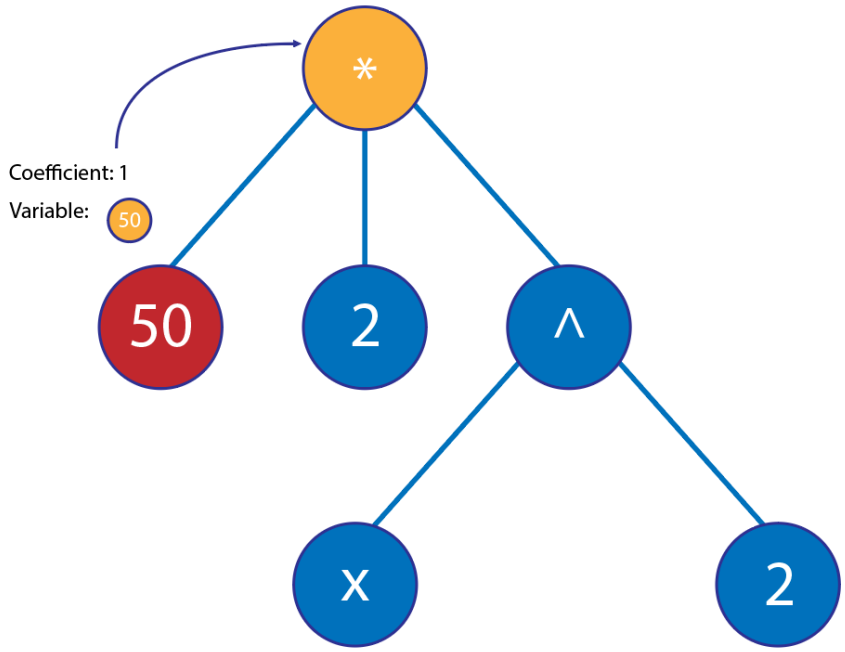
As shown in the pseudocode, we will now call the function itself and start anew from this number.

This is the situation we end up in:



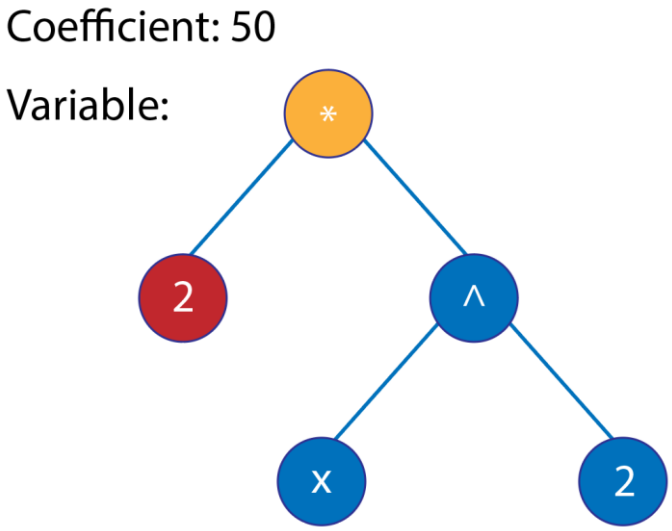
As the tree doesn't have any children the for loops end up doing nothing. So, what is returned is a structure with a coefficient of value 1, and the variable as the node that has a value of 50.

We end back up to the top:

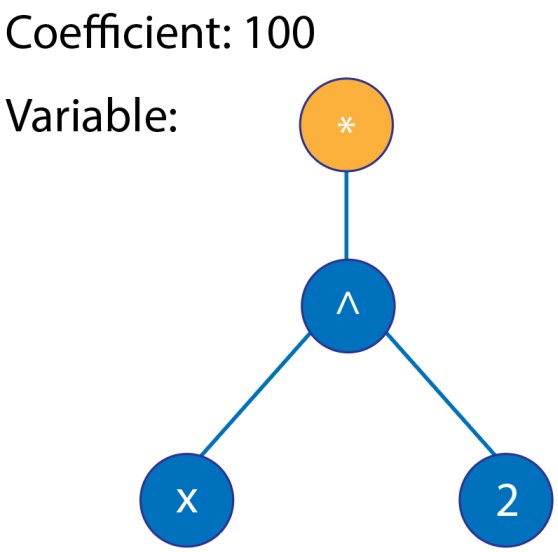


And this time, the returned values are checked. As the returned coefficient is 1, the existing coefficient will remain 1. However, in the pseudocode, it checks whether the variable's value is a number. The value of the tree node is 50, which is a number, so the coefficient is multiplied by 50; making it 50. Also, the tree node will be deleted.

After this we are left, with the algorithm moving to 2, the node:



The algorithm will do much of the same, returning 1 and the node. So, continuing, we end up with this:



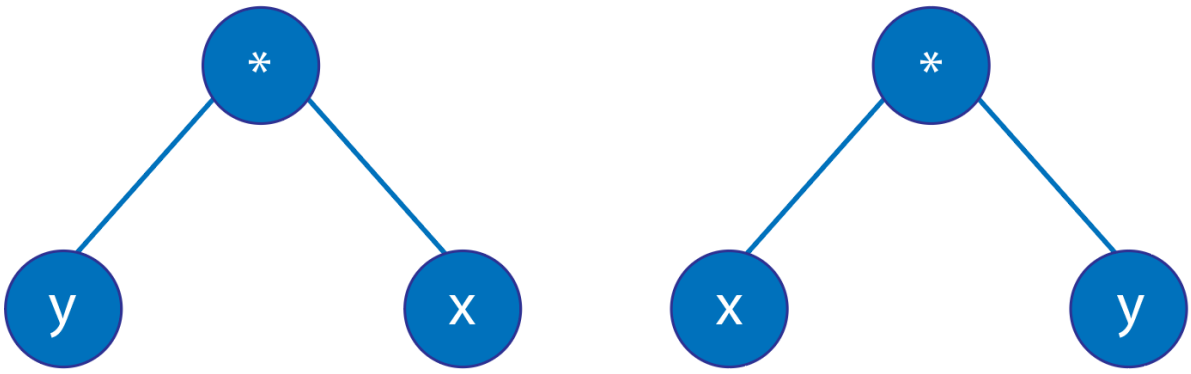
We will not self-call this time, as the for-loop checks for "^" nodes and does nothing with them. As this is so, the for-loop will terminate with nothing else to check. Afterwards the function itself will return a structure, with the coefficient having a value of 100, and the variable being just the "*" node with the single child. (x^2).

The use of this can be seen to be pretty obvious. As I can call this function on multiple different nodes, if it is found that the same variable is returned for two, I can sum their coefficients and make them one. This is the crux of the other algorithm, that essentially is a for loop that looks through a "+" node, comparing every node with each other, finding matches and combining the nodes into one.

I won't bore you with this, but the only real challenge is when I have differently arranged variables. To compare variables, I can easily just reconvert them into infix strings, but this will also include their order. Say $5xy + 10yx$.

So, to approach this problem lets store each element of the variable in a list. In this case the variable for $5xy$ is xy and for $10yx$ its yx.

We can create two new lists of tree nodes, and we can then add all of the elements of the variable into each. Say, `List1` and `List2`. These lists will be of tree nodes, as the variable part of the return is just a returned tree. More so, these two trees:



I will add the children to these two lists. And then, using `LINQ`, group these into a new list of their string counterparts.

What we end up with is two lists {“x”,”y”} and {“y”,”x”}. With this list we can now do a for loop through one of them (if the list’s counts are the same).

```
If List1.Count is the same as List2.Count:
 For each Item in List1 do:
   List2.Remove(Item)
 End For
End If
```

This simple loop will show that they are the same when List2’s count is 0. Remember that I only do this when they both contain the same number of elements. If this check is true, then I can continue by creating a new tree node with the summed coefficient, and variable “xy”.

I will remove the two nodes I am comparing from the main plus node, and then replace it with this new tree. In this case it would be $15xy$.

Prototype 3:

Preparation Algorithms

Limited Differentiation

Differentiation is a topic that is hard to approach, as one must first understand what it *actually* means to differentiate. Some people like to confuse this, especially its notation, so I'll try my best to make it as clear as possible.

The basic premise is also its most titular feature, and it allows quite natural recursive behaviour to emerge (by nature of itself). I will just do a general rundown of what the algorithm is doing, and why it does it, but it shouldn't be too much of a task because calculus can be disambiguated quite easily.

So, say I have the function $f(x) = x\text{^}2 + 3$.

To differentiate is to merely apply a 'tiny' change to this function and see what it results.

D(f(x)) is the syntax for this. The d(whatever) is a rather largely misunderstood idea, but really it always boils down to: "Apply a tiny change onto whatever the selected function is".

To change a function, you must change its parameters. The only thing you can change is this.

In the $f(x) = x\text{^}2 + 3$ example, the parameter that I can change is so dutifully called 'x'. I won't be pompous about the thing, so the general rule is:

For a function $f(x, y, z \dots)$, the derivative is the sum of the tiny changes resulting from each variable. A two-variable function $f(x, y)$ would be the change from changing x plus the change from changing y. This idea calls itself, as the changes of x, and y, are themselves derivatives. Ie, d(f(x,y) is defined as a tiny change, much like dx and dy.

This idea is literally all there is to a derivative, and the reason it is recursive (and why chain rule can exist). It is mostly a game of disambiguation, so I'll generalise the rule that I will be using for calculation of limited derivatives (no functions like sin).

$$d\big(f(x, y, z \dots)\big) = \frac{\partial f(x, y, z, \dots)}{\partial x} dx + \frac{\partial f(x, y, z, \dots)}{\partial y} dy + \frac{\partial f(x, y, z, \dots)}{\partial z} dz + \cdots$$

Recursion is inherent through the natural occurrence of the multipliers next to the partial derivative. The partial derivative just tells you to derivative with respect to one variable, ie making one tiny change to a variable.

$$\frac{\partial f(x, y, z, \dots)}{\partial x} \; means\ to\ apply\ a\ tiny\ change, dx, to\ the\ function.\ This\ is\ a\ ratio.$$

**One common misconception is that $\partial x \neq dx$. Do not think about it this way. The divider is required and wrote differently because it states that I must only change x. In fact, this would be another way to write it:*

$$\frac{\partial f(x, y, z, \dots)}{\partial x} = \frac{f(x + dx, y, z, \dots) - f(x, y, z, \dots)}{dx}$$

Do not confuse notation with difference. Partial derivatives apply the same rules, but just focus on one variable. The notation shows this by making the symbol different. There is no other place for this different symbol, so you never write ∂x by itself, as it holds no meaning. ∂x by itself doesn't tell you what your differentiating with respect to!

This is why $\frac{\partial y}{\partial x} = \frac{dy}{dx}$ when y is only in terms of x. 'differentiate y with respect to x and divide by dx' is the same as 'take the total derivative of y and divide by dx' as the only variable in y is x, so you can only change the variable x.

dx itself means to take the total derivative of the function x. There is no function x, so I merely write 'dx' because I am not a magician. Anyway, due to this, I can define chain rule. This allows me to differentiate functions that would otherwise take immense computational power, with ease.

Say I have,

$$f(x) = (x^2 + 3)^{56}$$

I want to differentiate this function. I know the general rule for power differentiation, that is,

$$f(x) = x^n \ \rightarrow d\big(f(x)\big) = nx^{n-1} \times dx$$

This can be proved easily by applying the definition of a derivative. I.e.,

$$d\big(f(x)\big) = f(x + dx) - f(x)$$

This is all I need to know for my algorithm. It may seem like $(x^2 + 3)^{56}$ is not in the form of the rule, but you need to look at things in a more general sense. Derivatives work on disambiguation.

I have this function, instead.

$$f(u) = u^{56}$$

This is a totally fine function, indeed, so let's take the derivative. The total derivative, mind you, and of the appearance that its most disambiguated form is of 'u'.

$$d\big(f(u)\big) = 56u^{55} \times du$$

This sort of thing is where the utter beauty of derivatives emerges. As every derivative is merely something multiplied by the derivative of the variable it is differentiating with respect to (in this case it is '$56u^{55}$' multiplied by 'du') I have the ability to apply the general rule on anything.

For the example, as du just refers to the derivative of u, I can let $u = x^2 + 3$. Even if this wasn't stated with just du, I could infer this direction through some notation. Normally, you would write:

$$d\big(f(u)\big) = 56u^{55} \times \frac{du}{dx} \times dx$$

But you must remember that just because I put $\frac{du}{dx}$, it doesn't mean that I am differentiating with respect to x. It is the 'total derivative of u' over x. If u was in terms of c, then this would merely become something unusable. A partial derivative means what most people think of a derivative, but as this is a one variable function, there is no difference.

So, if I have the function, u, in terms of x, then I can calculate this quite easily.

$$d\big(f(u)\big) = 56u^{55} \times \frac{d(x^2 + 3)}{dx} \times dx$$

Apply the general rule,

$$\Rightarrow 56u^{55} \times \frac{2x \times dx}{dx} \times dx$$

Remember that every derivative (if it is in terms of only x) will be in the form, $A \times dx$. I can simplify this, and replace u with the form in terms of x.

$$\Rightarrow 56(x^2 + 3)^{55} \times 2x \times dx$$

Now, I am back where I was at the start. But this time it's in terms of x. If I now happened to say that $x = 9t^3 - t$ then I can redo this whole thing. Note that x doesn't have to be in terms of one variable, but that is beyond the general scope of this algorithm (the user cannot input this, and thus are limited to one round of disambiguation, namely $u \to x$).

Typically, the form that derivatives are in is $\frac{d(f(x))}{dx}$, so I will amend this to become:

$$\frac{d(f(x))}{dx} = 56(x^2 + 3)^{55} \times 2x$$

I will also like to focus on some other items of interest within my algorithm. Namely, multivariable form and how this presents some beautiful results.

I can write x^3 as $x \times x \times x$, and this is not in the form of the power rule I stated. Let's say that I didn't bother to simplify this, and just left the input as three terms in a multiplier node. This is still very much a possible scenario for my algorithm.

This is because the way I designed it. I.e., I followed the principle of partial derivatives – as explained in the introduction.

Let's create a function,

$$f(u, t, c) = u \times t \times c$$

Let's take the total derivative of this function,

$$d(f(u, t, c)) = \frac{\partial f(u, t, c)}{\partial u} du + \frac{\partial f(u, t, c)}{\partial t} dt + \frac{\partial f(u, t, c)}{\partial c} dc$$

This is just applying the definition, or the sum of each change I can make (the small change of u, t, and c.).

Let's apply the definition of a partial derivative.

$$\frac{\partial f(u, t, c)}{\partial u} du = \frac{du \times t \times c}{du} \times du$$

I treat the variable t and c as constants because they are not changing in this scenario. A partial derivative is always a ratio, due to the way it is written, so the du's will form 1. This is why I multiply by du to put it in the proper form, and not a ratio. Think of it as a targeted total derivative.

I can also write it using this definition:

$$\frac{\partial f(u, t, c)}{\partial u} du = [f(u + du, t, c) - f(u, t, c)]du = \frac{(u + du)tc - utc}{du} du = du \times t \times c$$

So, applying this to the three partial derivatives, it will form:

$$\Rightarrow du \times t \times c + u \times dt \times c + u \times c \times dc$$

Now, let's make u,t and c equal to x. This will procure the expression:

$$\Rightarrow dx \times x + x \times dx \times x + x \times x \times dx$$

Let's factorise the dx,

$$\Rightarrow dx(xx + xx + xx)$$

Collecting like terms, this will form:

$$\Rightarrow 3xx \times dx$$

Adding powers, the simplified form is:

$$\Rightarrow 3x^2 \times dx$$

This is exactly the rule that you would normally apply. This can be seen as a proof, but in the end, it is just a beautiful result derived from something you wouldn't expect. It relates multivariable calculus to the one variable functions one may be used to, and shows that they are just as similar, and not some arcane gibberish.

My algorithm follows this rule, and thus can work with both forms. Now that I've gone over the general theory of derivatives, I'll explain how I implement it. And also why they're very compatible with tree functions, and recursion.

This is the name of the algorithm. As I have mentioned, it cannot handle functions, like sin, as I have not implemented them into the tree system. It also cannot handle non numeric powers, because this requires the introduction of the logarithm function.

Like,

$$d(2^x) = d(e^{ln2^x}) = d\left(e^{xln2}\right) = e^{xln2} \times d(xln2) = 2^x \times \ln(2) \times dx$$

This system cannot be implemented without the introduction of e and ln, which are just inverse functions of each other. This will also require the implementation of their respective rules, like the logarithm rules. I did not have the time to write another 15 functions to accommodate this.

Meeting 13/01/2021.

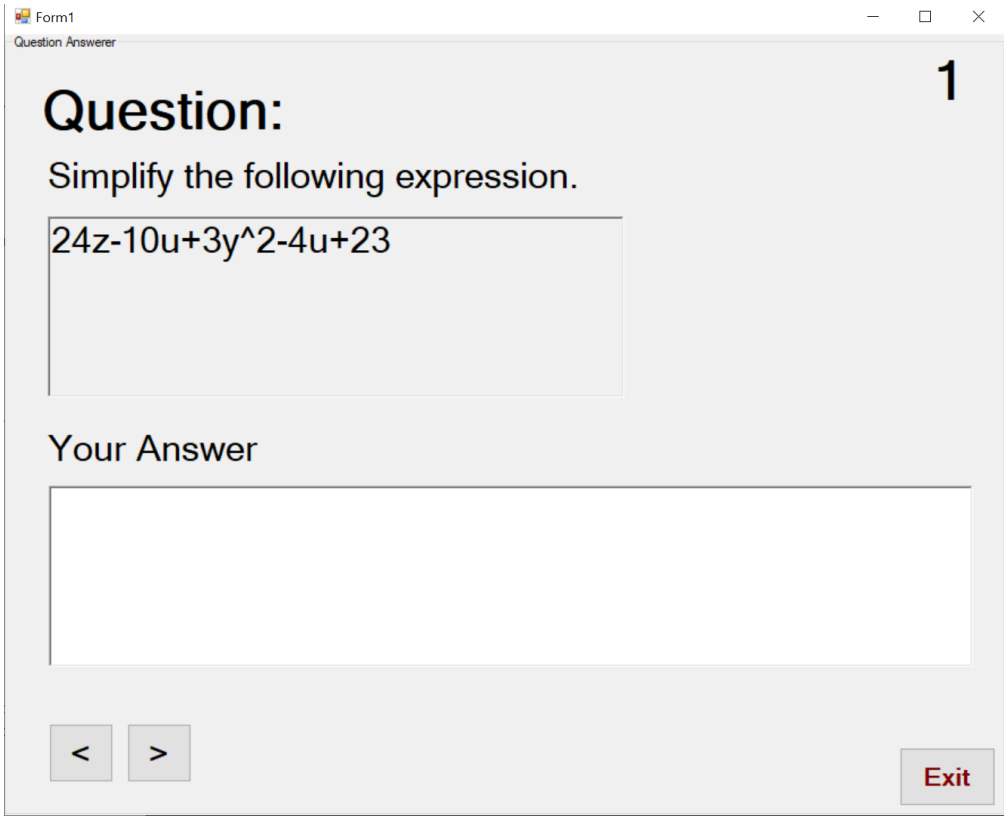
I held a meeting to discuss this. Because I had not talked about implementing derivatives as one of the question formats, I began to explain their use case. In my current system, I had nearly implemented two major areas, which are simplification and derivatives.

For the derivatives I spoke about the difficulties of implementing a far-reaching solution, such that it will not be able to cover some area of A level calculus. Though, I added that it can compute multivariable functions, and thus surpasses A level and enters University calculus.

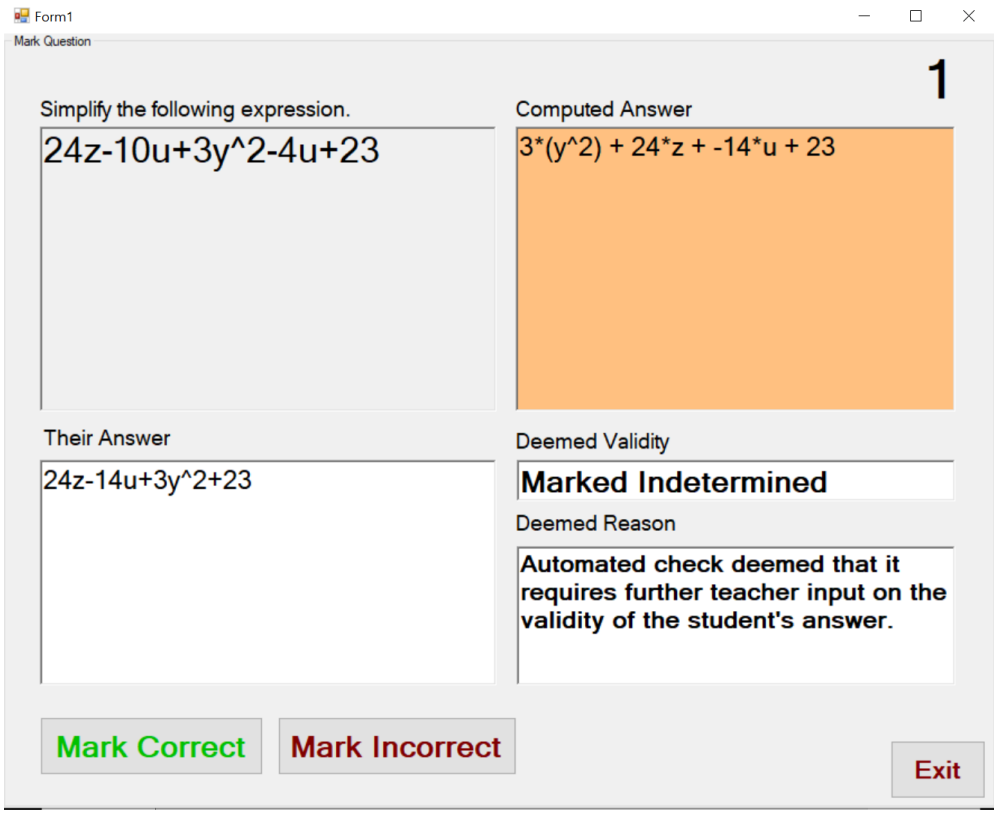
He said that that would be useful for preparing student who want to do maths in uni, and seemed pleased with this fact. He said that the functions are not needed, as my method still does product rule, chain rule, and handles fraction derivatives well.

He said that the simplification side of the system would be more useful for lower year students, as it can teach them how to add powers, multiply out brackets, and collect like terms.

I also showed him some of my prototype designs.



and the teacher answer menu.



He was happy with the overall design and how it adhered to a standard that was logical and easy to understand. I explained that on the simplification area, the automated marking wouldn't be able to know for sure if an answer was correct (though it would know if it was wrong). I said this was due to the complexity of checking, and that it would require a lot of time to properly implement a fully checked solution. Derivatives, however, will be able to be ably checked and won't have this issue.

Overall, he was pleased with the way it was heading. So, I ended the meeting and finished my derivatives algorithm.

Limited Differentiation

In my technical solution there will be a function with every part explained in vb. It merely does a traversal, and looks for situations where it can differentiate, applying the rules I have shown above.

One thing would be that it doesn't do $d(f(x,y,z,\dots))$, but $\frac{d(f(x,y,z,\dots))}{dx}$. As shown, this will merely make everything over dx, and in occasions where it cannot be computed, it will just leave it in the notation.

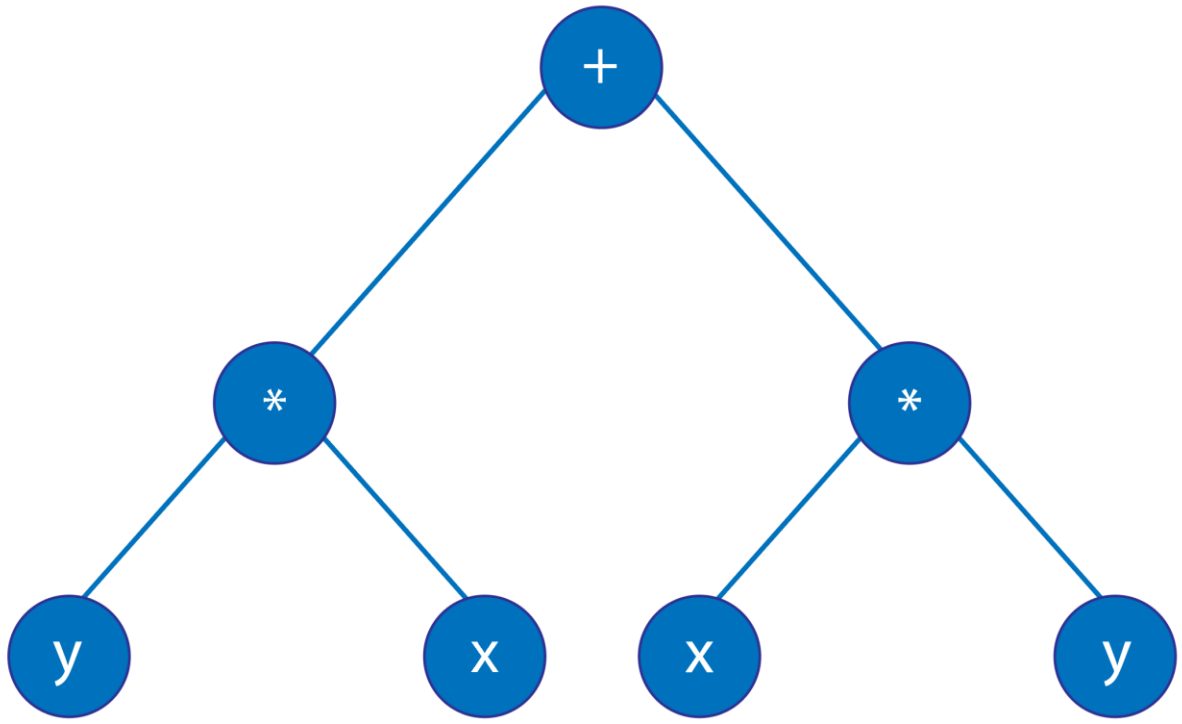
For example:

$$\frac{d(x^2+y^2)}{dx} = \frac{\partial(x^2+y^2)}{\partial x}\frac{dx}{dx} + \frac{\partial(x^2+y^2)}{\partial y}\frac{dy}{dx} = 2x \times \frac{dx}{dx} + 2y \times \frac{dy}{dx} = 2x + 2y \times \frac{dy}{dx}$$

This is a normal way to apply this.

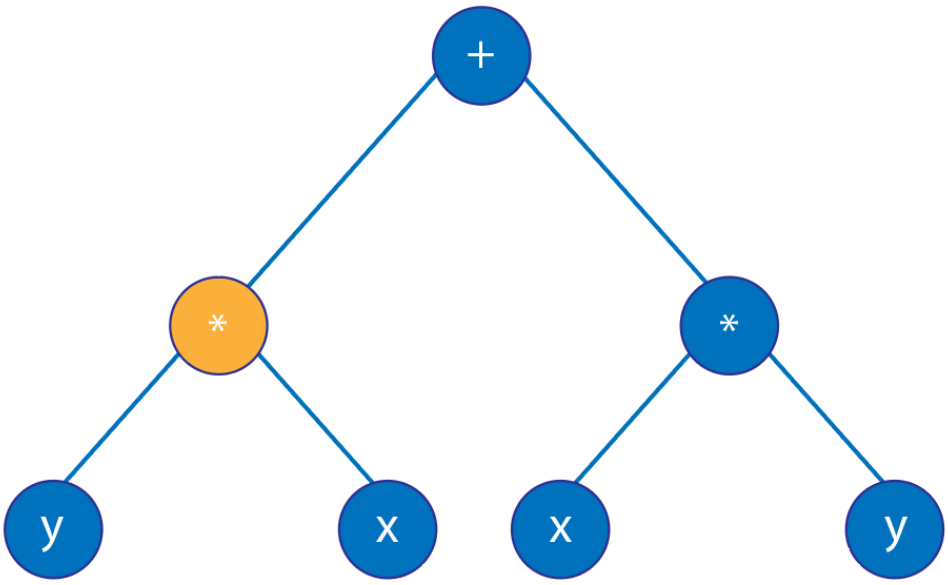
I will show a scenario where the algorithm works. Pseudocode will not be needed as it will be too long to help understanding (the algorithm is separated into 3 functions and around 200 lines long).

Say we start with this tree:



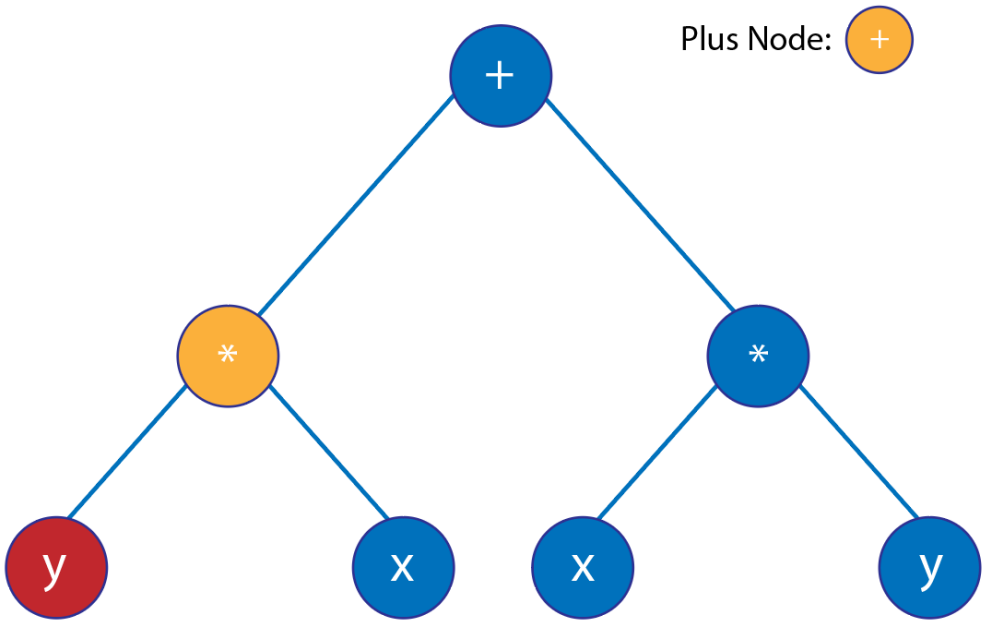
My differentiation algorithm starts with the function `DIFFERENTIATION_WRAPPER` and it will look at the root node, checking to see if it is $+$ or something else. A $+$ node will make the algorithm loop through each child, calling the function `LIMITED_DIFFERENTIATE` for each. It will then replace each child with the output of the function.

In this case we have a $+$ node, so we will start looping through.



The function has now started with the $*$ node as its input. The algorithm will first check whether it is a * node, and it is. This is done because any variable will be in a * node (my preparation algorithm will make everything $1 * a$).

It will make a $+$ node, and then start to loop through each child of the input node, so we end up with this situation:



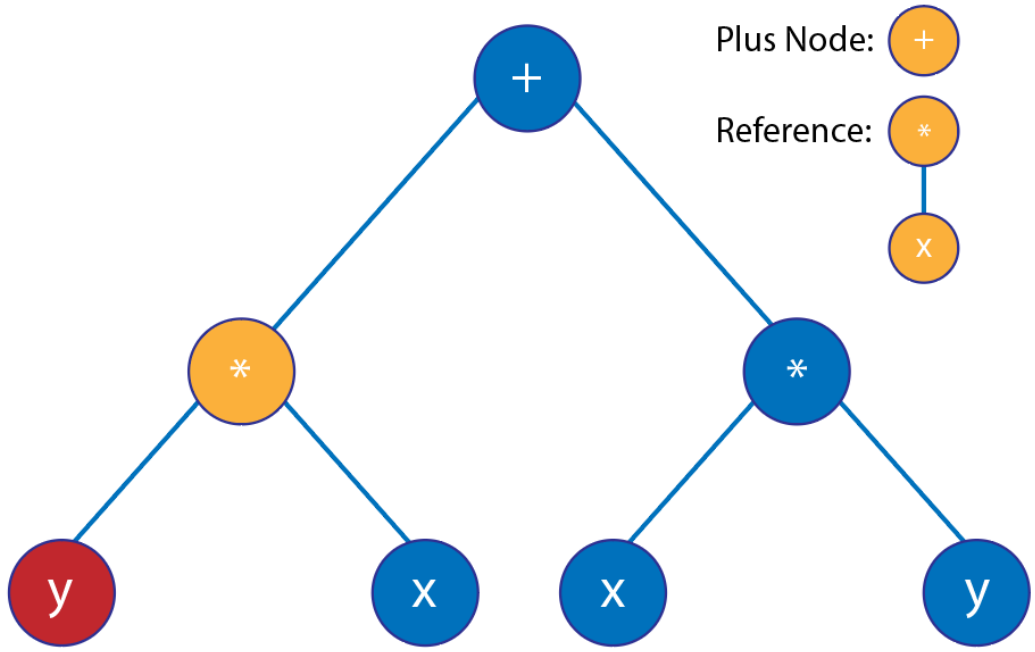
If you remember the rules, then the reason for the plus node is that this is a product derivative. Thus, it becomes the sum:

$$\frac{\partial(xy)}{\partial x} + \frac{\partial(xy)}{\partial y}\frac{dy}{dx}$$

With y selected, I will now create a 'reference node'. This node will contain every other node in the $*$ node I have selected. In this case it will just x. This is because, remember, a partial derivative treats every other variable as a constant. In this case I am computing $\frac{\partial(xy)}{\partial y}\frac{dy}{dx}$, so with respect to y.

This 'reference node' will be used when I need to put my derivative together.

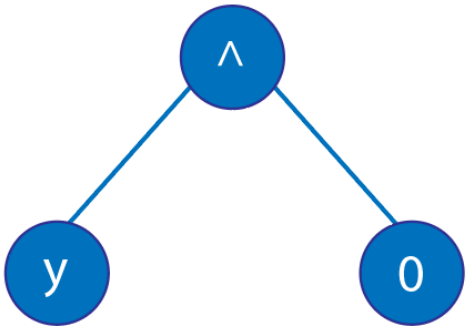
So, this is the situation:



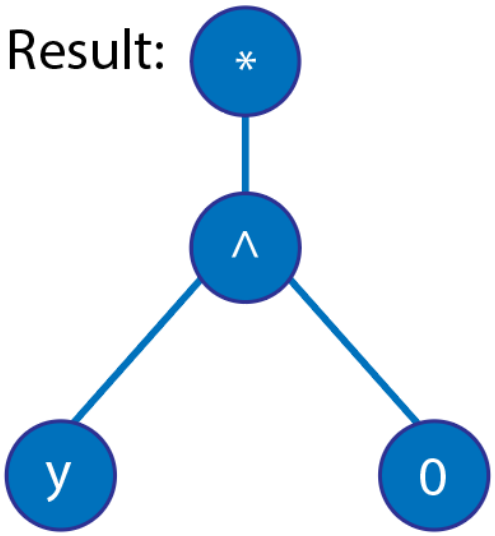
Now, to create the derivative. I will first check that it is a ^ node. It is. To save space I have omitted this, but normally every single variable in this would be in the form a^1, so that it is easy to deal with a 1 power. I also have it as a * node, so $1 \times a^1$ is the general form everything follows. I will ignore this sometimes, as it just adds complexity for no reason.

With this assumption, I will see if it is a variable or a number. It is a variable. So, I begin to create my derivative. First, I create a new multiplication node to hold the whole result. I first clone the selected node, which is y^1, and take away 1 from its power.

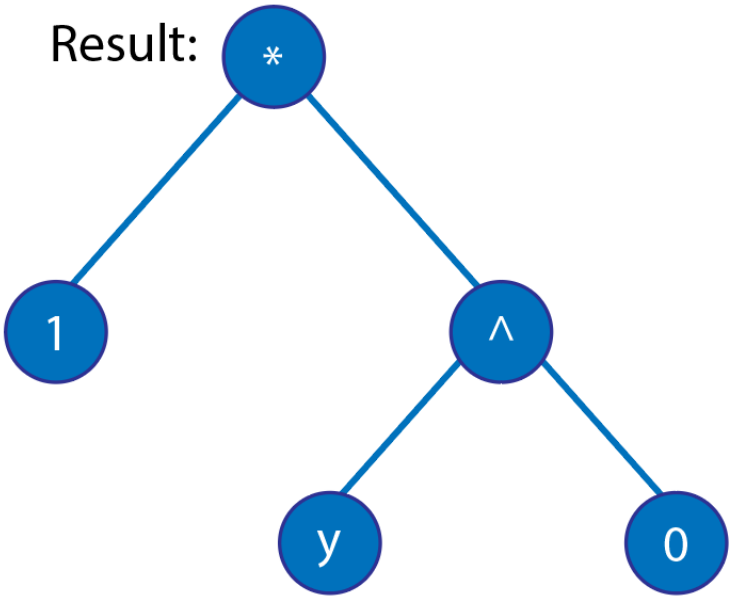
So, I end up with the node:



This node is then added to the multiplication node that will hold the result. I'll call it the result node.

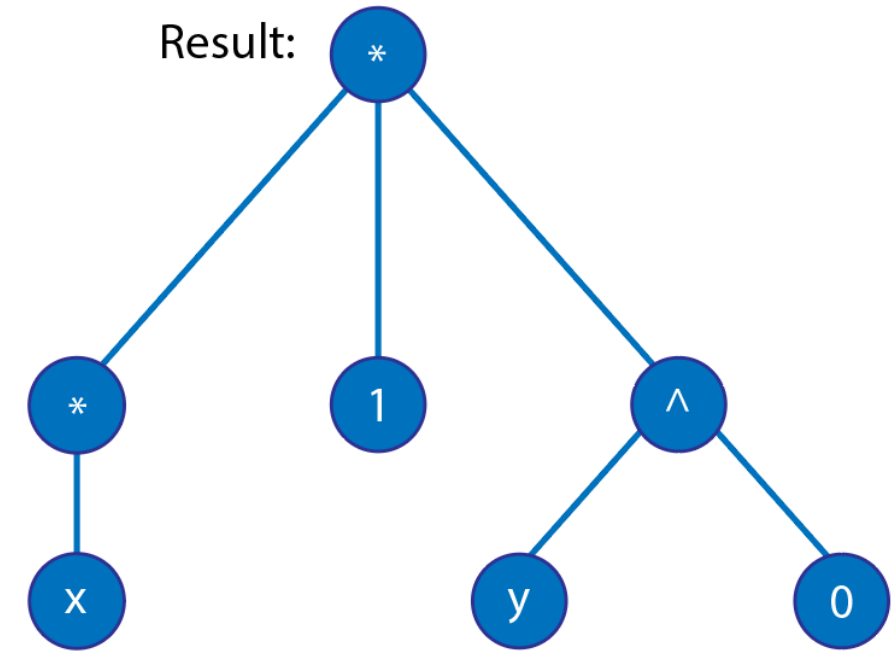


After this I will add the power of the original selected node. The y^1 one. This would be 1.



As you can see, this is implementing the power rule. However, the reference node's children still need to be added.

So,



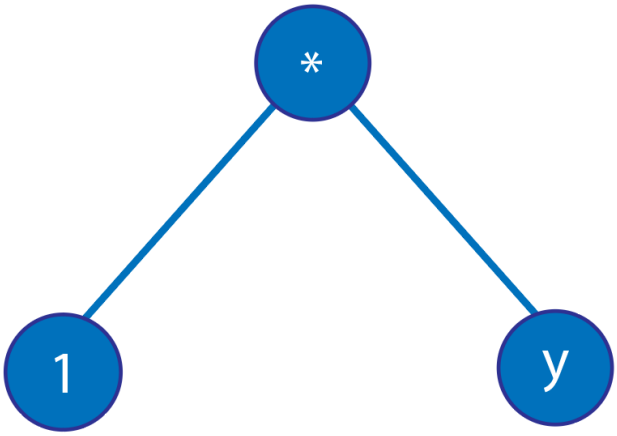
I now need to add the derivative node. I will first create in the form $1 * y$, or more so the selected node multiplied by 1. If this wasn't y, but $3x^2 + 3$, then I would instead send $1 * (3x^2 + 3)$. As I would be differentiating with respect to u in that case.

The reason I do not put it in the form $1 * y^1$ is that it would cause it to run forever. The if statement would find a ^ node, and then recursively call itself again, and again. Without a power node it will stop at this point.

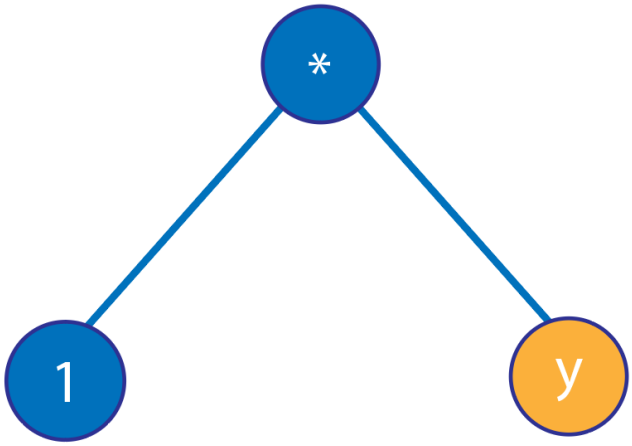
It would do something like,

$$x \times 1 \times (y^0 \times y^0 \times y^0 \dots)$$

Anyway, it will now recursively call the function with this function as the input,



This input has a $*$, so it begins to loop through. A new plus node is created, however the '1' node will not cause anything, as it doesn't satisfy any of the if statements in the for loop. So, it moves to the power node.

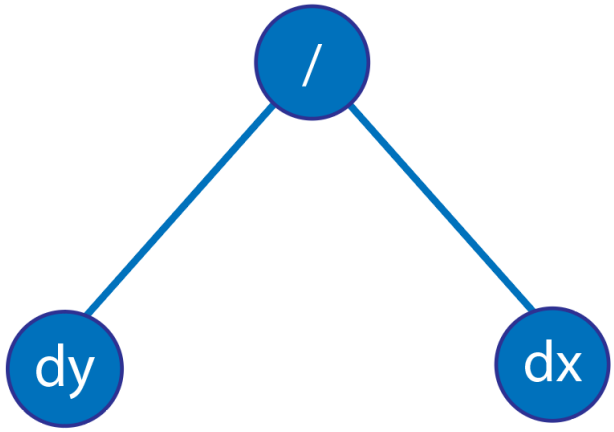


It will go through the same process. But this time the if statement will execute a different part of the code. It will execute on the `Else`, where the two before checked whether it was a ^ node, and then a + node.

On this `Else` it will see if it contains a variable other than x. It uses the regex:

$$(?=[a-z])[\hat{}x]$$

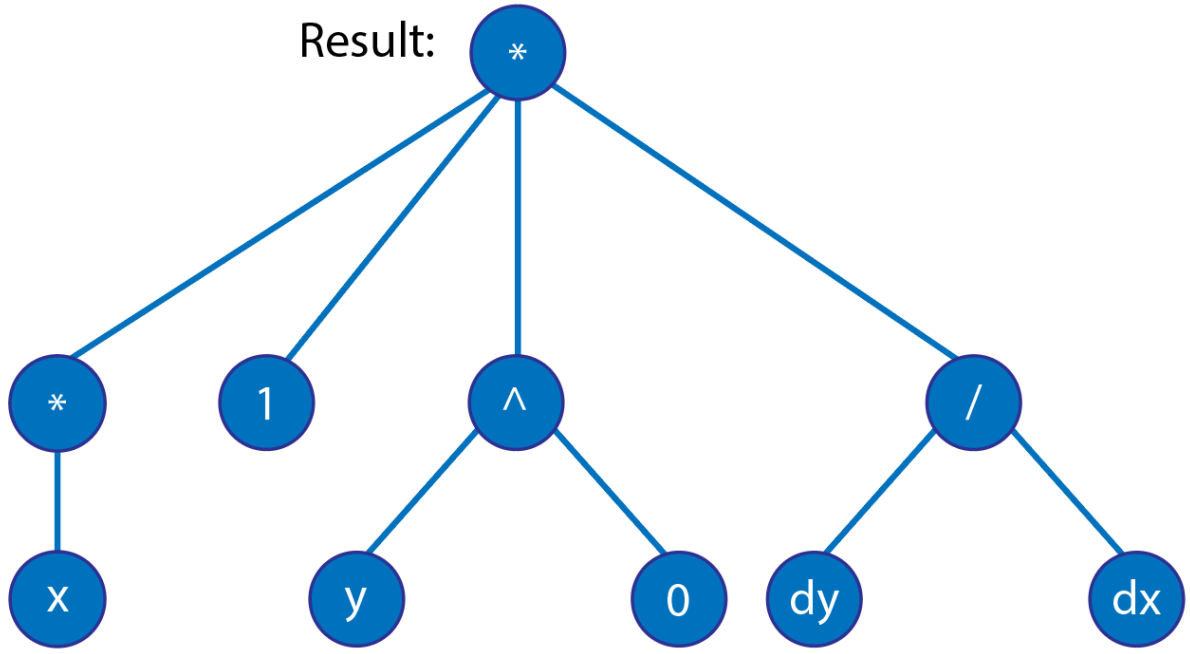
As it is a 'y', the regex will return a count of more than 0. So, I create a new node,



It would be d(whatever the variable was) over dx. Notice that this (dy) is not in the form of a * node, so these nodes will be treated separate from others. Even if you input dx, it will transform into $d \times x$, which will not simplify with just a single node dx. This means I do not need to add special classes to accommodate this.

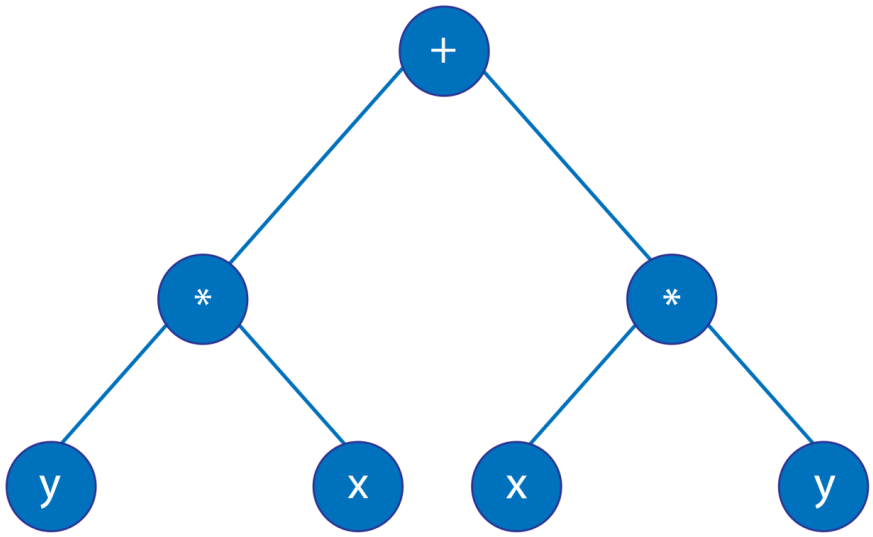
This node is added to a new 'plus node' and returned. This returned value is what will be added to the 'Result node'.

So, I end up with this node:



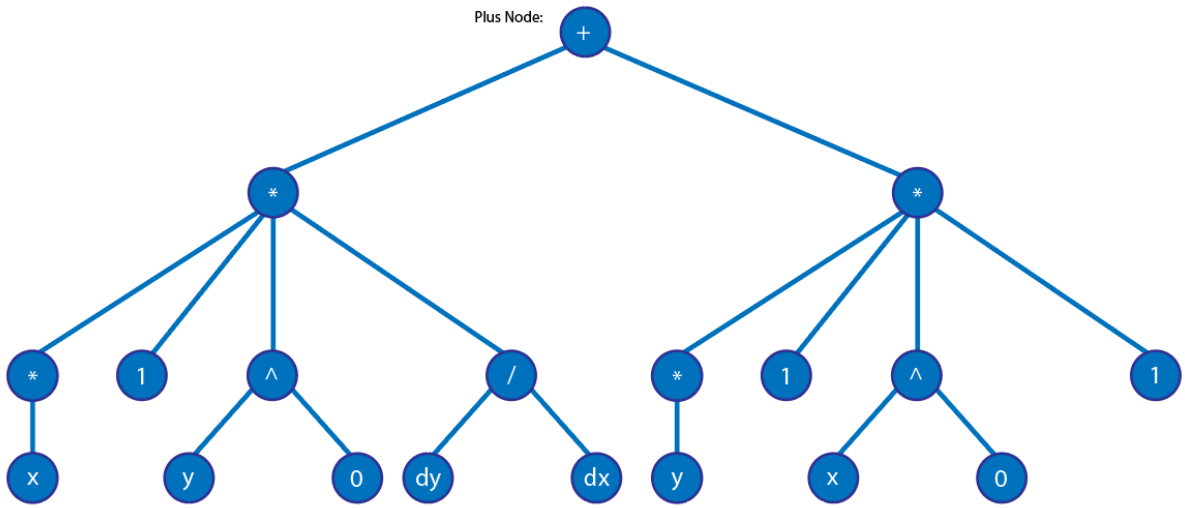
Afterwards this result node is added to the plus node. This will continue through each term, so 4 times. However, this will make two plus nodes with two terms in each. For each * node.

These two plus nodes are then added to the original + node that I begun with. Ie,



In this case it will remove each * node, replacing them with the new plus node that contains two result nodes. I will then reapply my algorithms, which will level these + nodes to one whole.

I won't display the whole result as it will be too big, but the first plus node will be this:



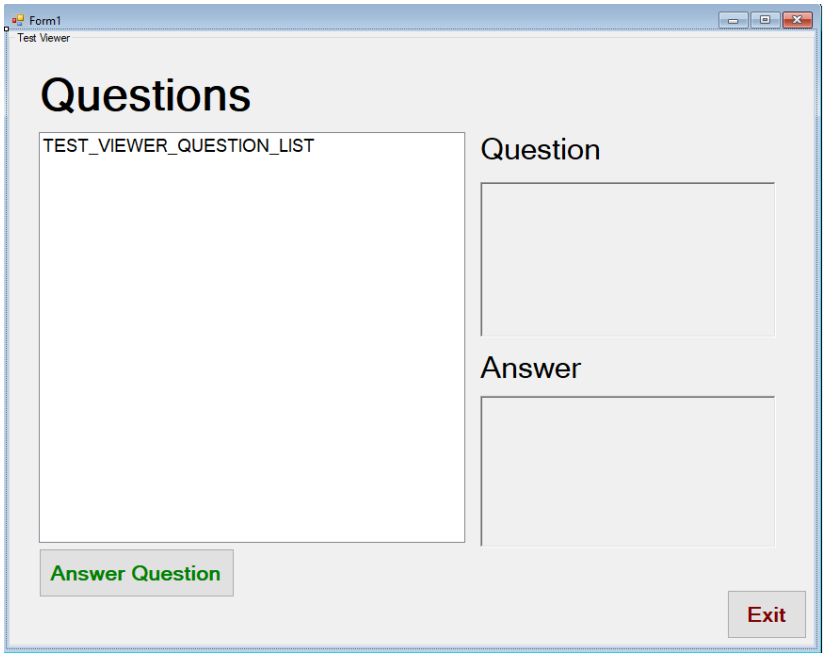
This is how the algorithm works in general, but there are some systems I haven't covered, like how it handles + nodes. The technical solution will contain the code so you can see exactly how it handles those situations.

Interface Design

Prototype 3

If you look at the critical path, the maths backend was created in prototype 1 and 2, and everything else in prototype 3 (including the specialisation of differentiating). I designed and created the interface, and the form code that interacts with my backend solution in this prototype. This happens to include most of my other BR's, however this is because the enormity of the first two required a lot more effort.

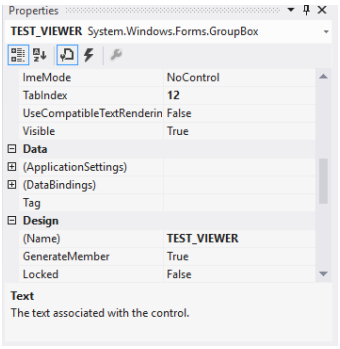
My interface is segragated into groups. Take this example:



At the top of the form it says ‘Test Viewer’. This is a **groupbox** control and it is used to contain a ‘screen’ for a part of my system. Due to the enormity of my requirements, there are numerous of these groups. Their usefulness comes from the fact that, when I create a control as a child of this group, everything in the group dissapears when I set its ‘Visible’ property to **false**.

This means that I can make designs and easily clear and display new screens whenever needed. Without this it would have been a pain to manage the numerous buttons I have. So this can be seen as something that groups the relevants buttons/controls together.

A logical system follows on how I name each button. For the example I showed, the member name is this:



Every control within this group will be called ‘**TEST_VIEWER_**X’. So, for example, the ‘Answer Question’ button is called,

The code in **FORM.vb will use this naming scheme.** So any button, list item, or whatever will be easily identifiable and found where it emerges. For the design, there are a couple template groups that I used a couple times for different areas of the program.

As illustrated in my **BRs**, the areas in my program are:

Teacher Area:

1. **Question Creation Area**
2. **Question Marking Area**

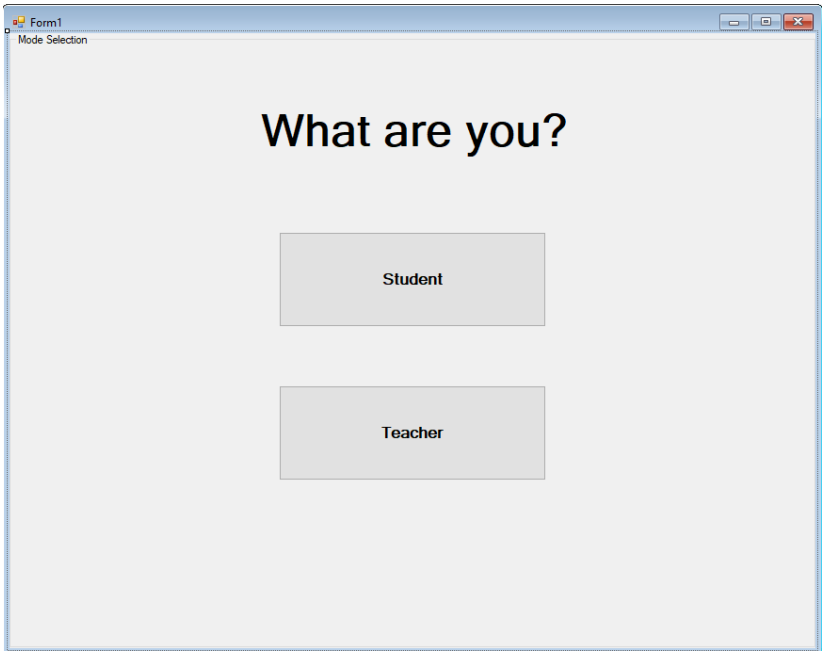
Student Area:

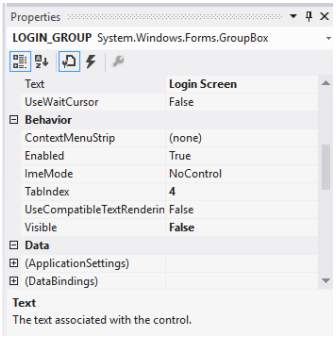
1. **Question Answerer Area**

These areas normally house a couple of groupbox’s, each with a different purpose. I will go through each of these 3 areas.

Login Area

This is the first thing I designed, and only has 2 group boxes.

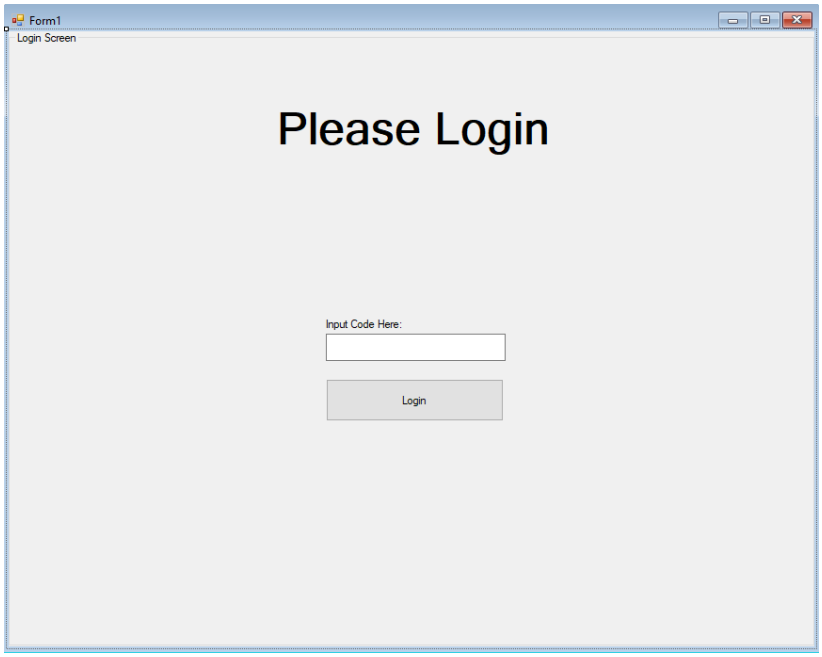


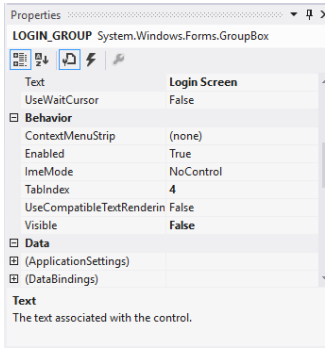


This is the first thing that the user is greeted. Simply, you click the button for what you are. These both lead to the same groupbox, but the class will be set to accept only the login details of the selected type.

For student, the password is '**student'** as default, and for teacher, the password is '**teacher'**.

After clicking either button, you are then shown this group box. Note that I clear all the groupboxes, and show the selected one as the system.

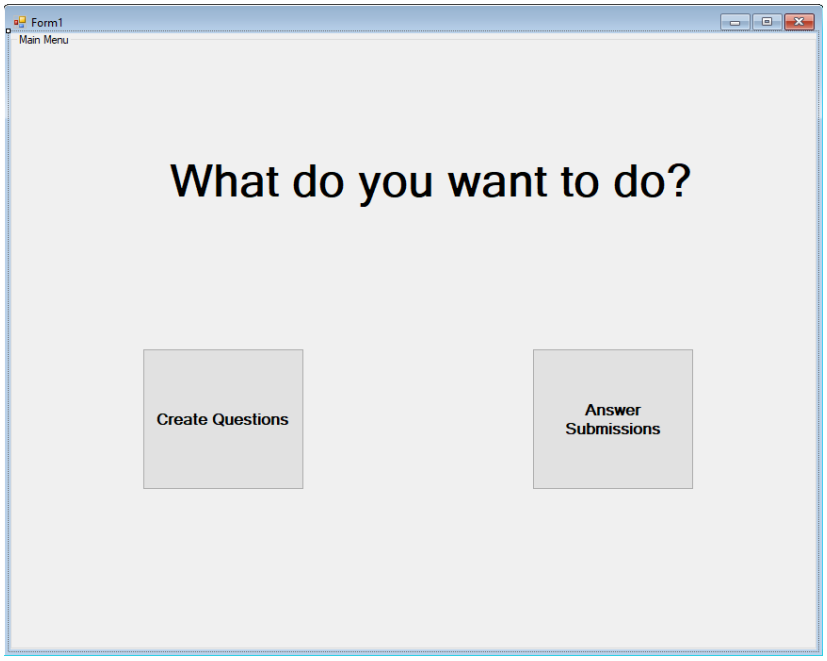


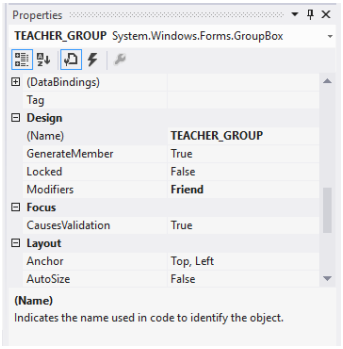


The user inputs their password into the LOGIN_INPUT box, and then clicks the LOGIN_BUTTON**. Note that in this group there is a minor difference where the group is called LOGIN_GROUP. This may sometimes emerge, but in general everything follows some form of easy identification to which group they are in. For this case the first item 'LOGIN' is used as the identifier, and the groupbox is also identified using this.**

The next group displayed is dependent on what the user chose. I'll start with the **teacher area**.

Teacher Area

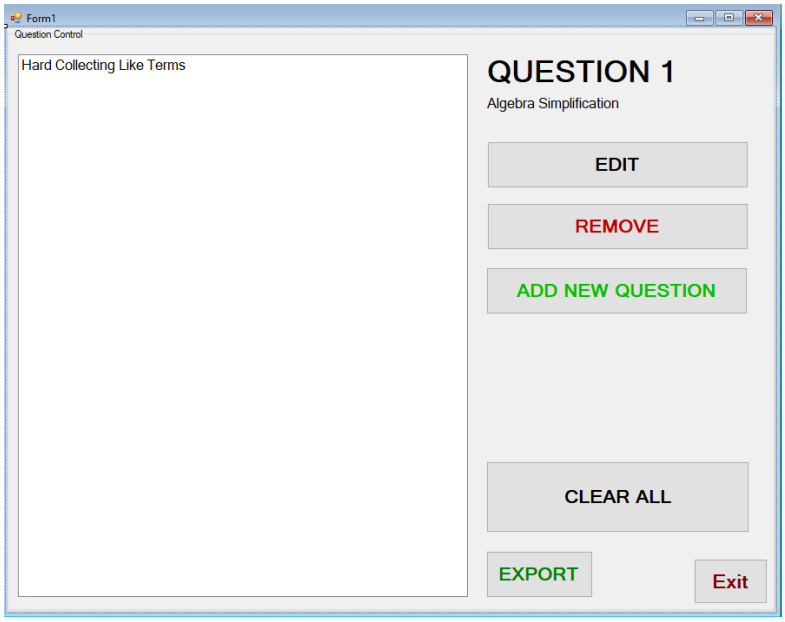


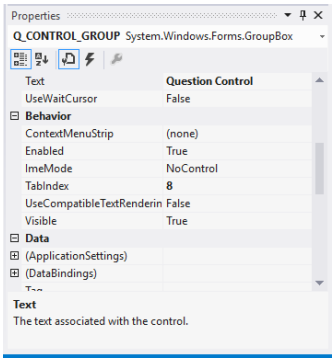


This is the first thing you are greeted with when you login as a teacher. The buttons you click determine which are you end up in; either the **Question Creation Area,** or the **Question Marking area.**

When you click the TEACHER_CREATE_QUESTIONS button you will be moved to the Question Creation Area, and when you click the TEACHER_MARK_SUBMISSIONS you will be moved to the Question Marking Area.

Question Creation Area

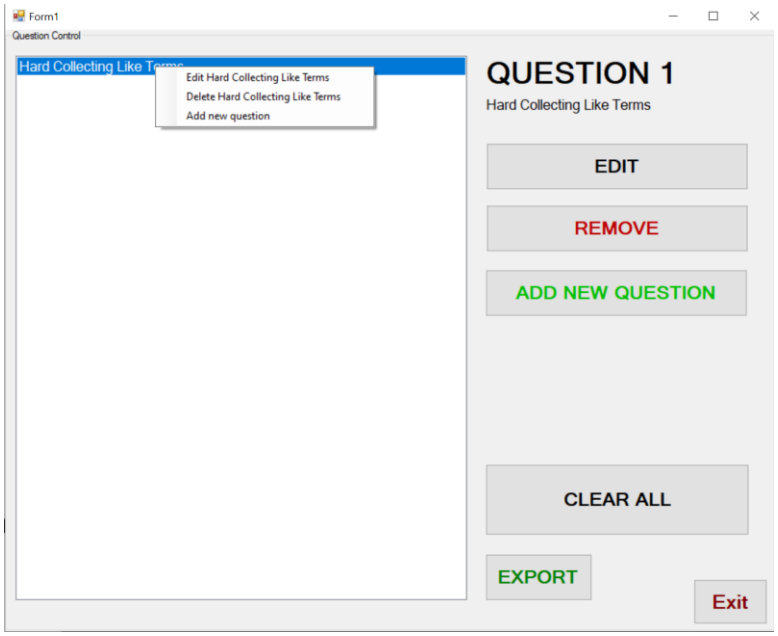




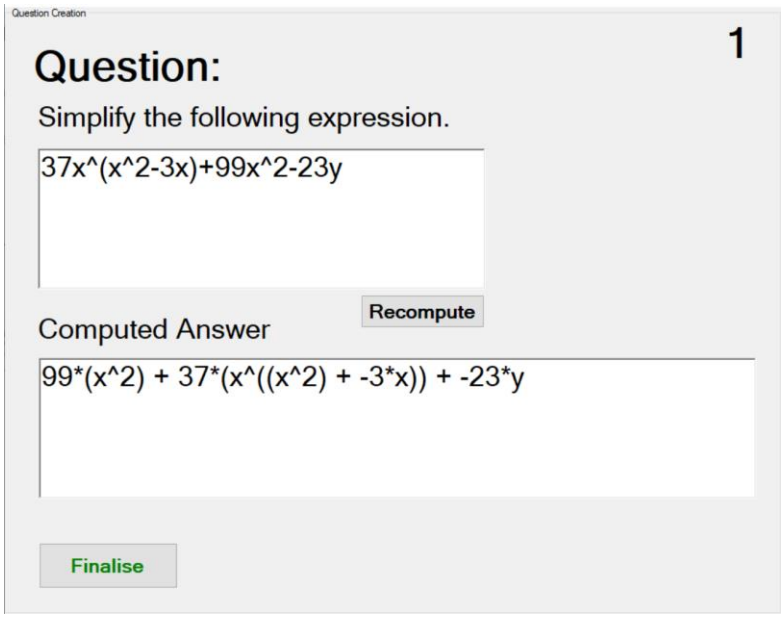
This type of interface will be seen three times in the program. It is essentially a listbox that contains questions. The listbox is called Q_CONTROL_GROUP_LISTBOX and you can use the various functions shown in the buttons. Each called after what they do.

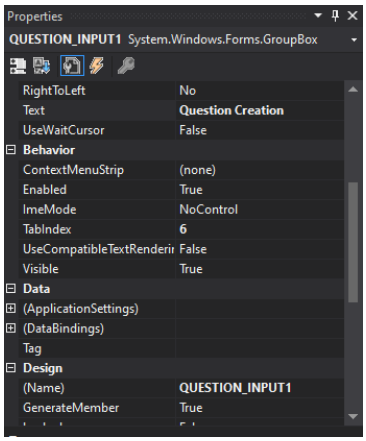
To use this, you select a question in the listbox, the 'QUESTION 1' will change to the number, with its description also; you will then be able to either right-click the listbox, or click the buttons **Q_CONTROL_GROUP_EDIT**, **Q_CONTROL_GROUP_REMOVE**, and **Q_CONTROL_GROUP_ADD_QUESTION**.

So, say I have it selected.



Clicking 'Edit Hard Collecting Like Terms', or the button **Q_CONTROL_GROUP_EDIT,** I move to this box.

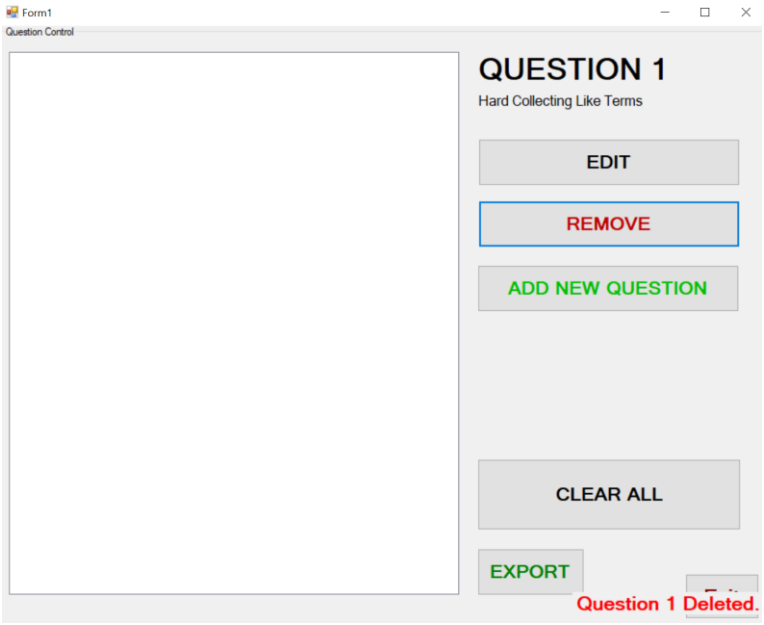




This box is where I can edit any questions I have created for the test. The box **QUESTION_INPUT** contains the question text, in this case what to simplify.

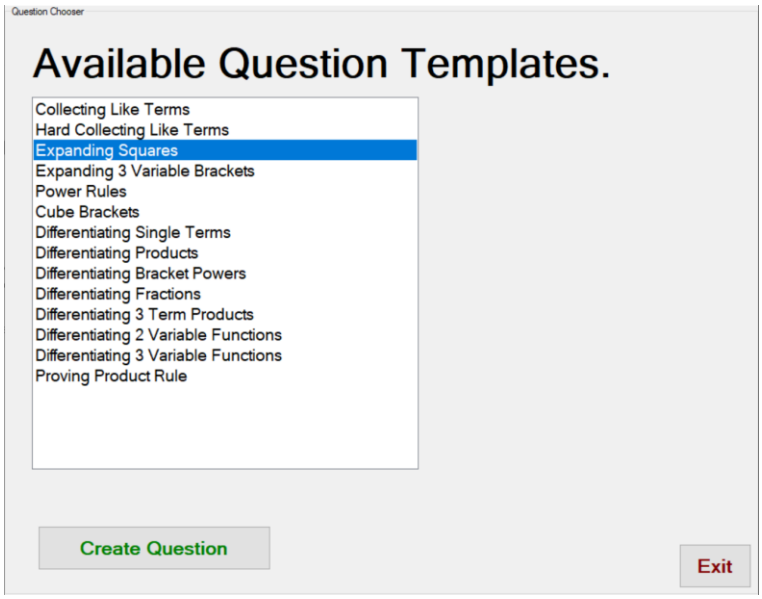
The box **QUESTION_CREATION_ANSWER** contains the computed answer given by the program. Both these boxes can be edited by the teacher. The button **QUESTION_RECOMPUTE_ANSWER** is used to recompute the answer if the teacher has edited the question and wants to see the new answer. Clicking finalise will save any changes made to the two text boxes.

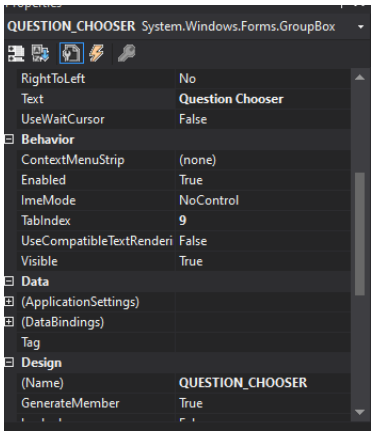
Going back to the main screen, I will now click **Q_CONTROL_GROUP_REMOVE** or 'Delete Hard Collecting Like Terms'.



Now, lets create a new question. This can be done either by right clicking the listbox and clicking 'Add new question' or clicking the button **Q_CONTROL_GROUP_ADD_QUESTION.**

This is what will be shown.

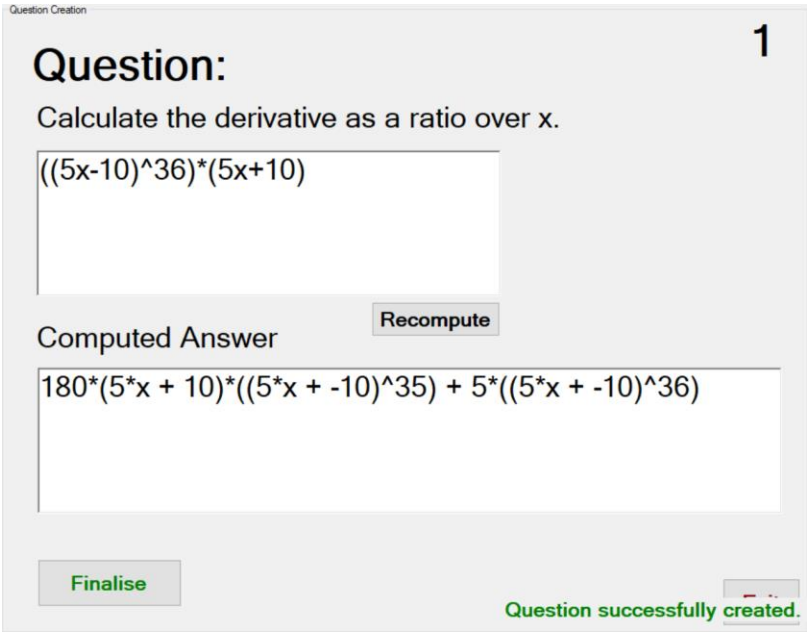




The listbox **QUESTION_CHOOSER_LIST** will contain every available template. These templates are what will be displayed for the student for each question. Each template contains 3 questions and the teacher can edit these are he likes. The plates with differentiating in them will differentiate the question instead of merely simplifying, so although the teacher can edit each question as they like,

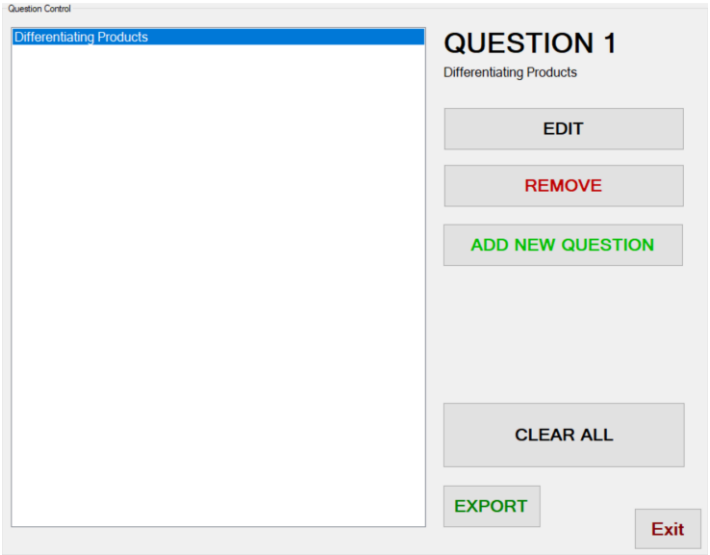
they should choose a template that will follow what they want to do; this is so that it functions as intended.

Let's say I choose the template 'Differentiating Products'. I merely need to click it in the list box, thus selecting it, and then click the button **QUESTION_CHOOSER_CREATE**. Doing this, I go to this groupbox:

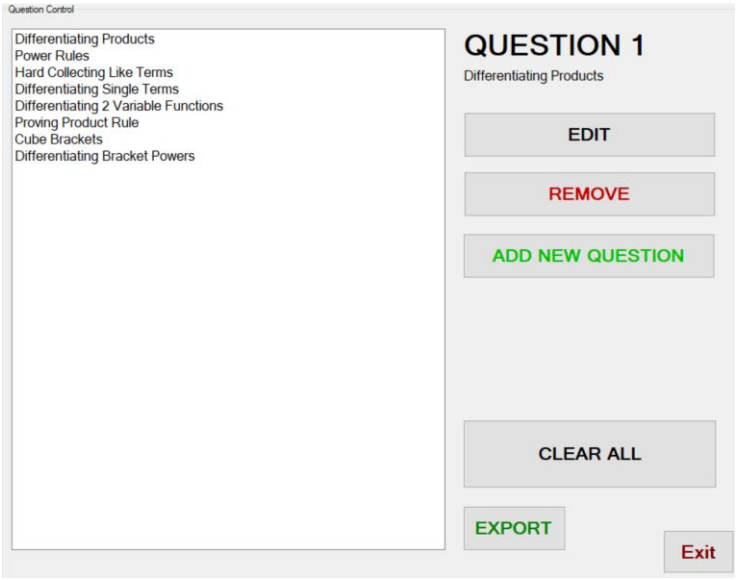


This is the same groupbox as the one used in the edit section. The question was chosen at random from the dictionary in the **DATA_HANDLE** class, so can be added onto at later dates. I do not need to expand on this as I have already explained its function on the edit section.

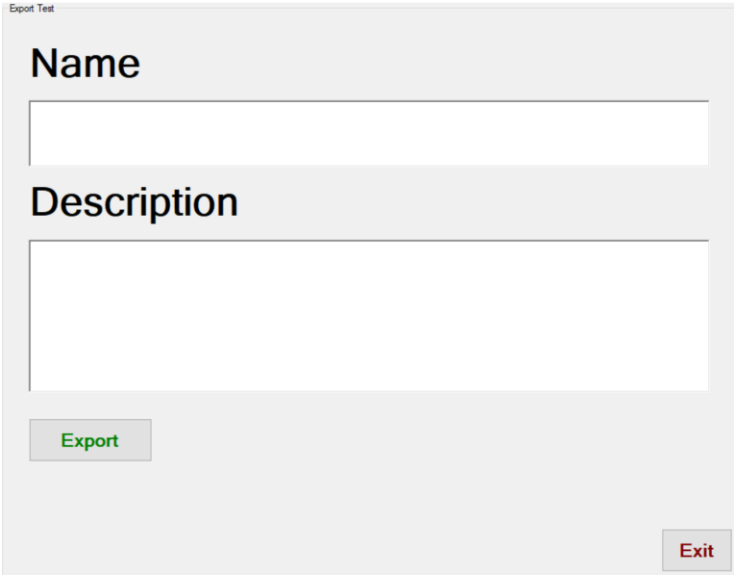
So, clicking 'Finalise', I have this screen.

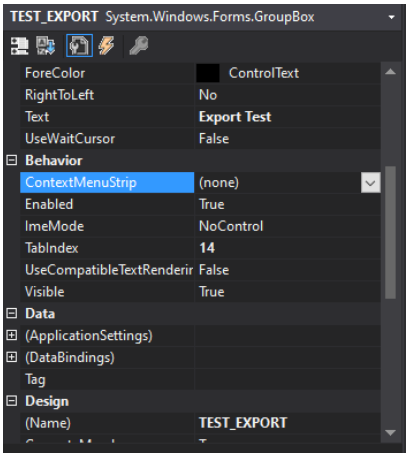


I will create a couple more questions:



Now, I will display the function of the 'Export' button. It is called **Q_CONTROL_GROUP_EXPORT,** and when clicked will send you to this groupbox.





This is where you name the test, and give it a small description. The name text box is called **TEST_EXPORT_NAME,** and the description **TEST_EXPORT_DESC**. I will enter a basic name and description.

Now, I will click the export button, **TEST_EXPORT_EXPORT.** It will create a file in your document file area.

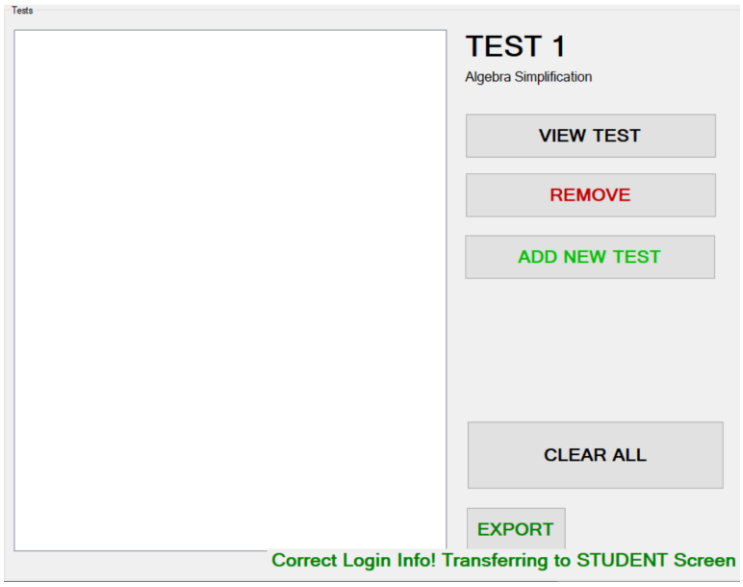
This will contain the json equivalent of an array of the tests serialized data. I am not serializing the class as that will waste data. Instead it just a string array with the important parts, such as the question, the computed answer. The details are described in the **IPSO chart**.

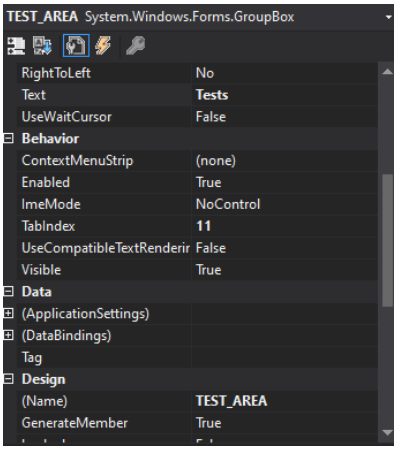
Now, following this, I will close the program and act as the student receiving this 'Test'.

Student Area

Question Answerer Area

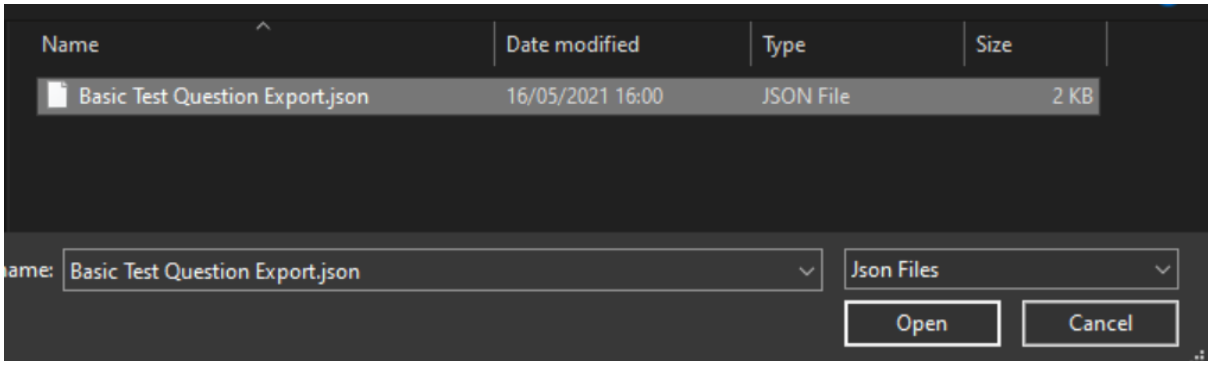
Logining as a student, I am greeted with this area.



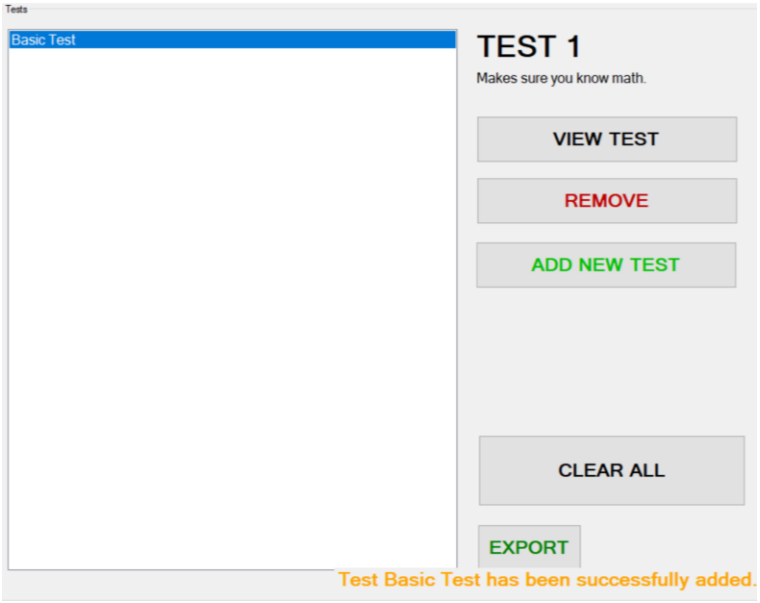


This is essentially the same groupbox as the one used by the teacher, but just modified to conform to the students uses. The button on the right, up to down, are called **TESTS_AREA_VIEW_TEST**, **TESTS_AREA_REMOVE_TEST**, **TESTS_AREA_ADD_NEW_TEST**, **TESTS_AREA_CLEAR_ALL** and **TESTS_AREA_EXPORT_TEST**.

Starting out, I will add a new test. So, I will click the **TESTS_AREA_ADD_NEW_TEST** button. This will bring up a file explorer window where I need to select my file.

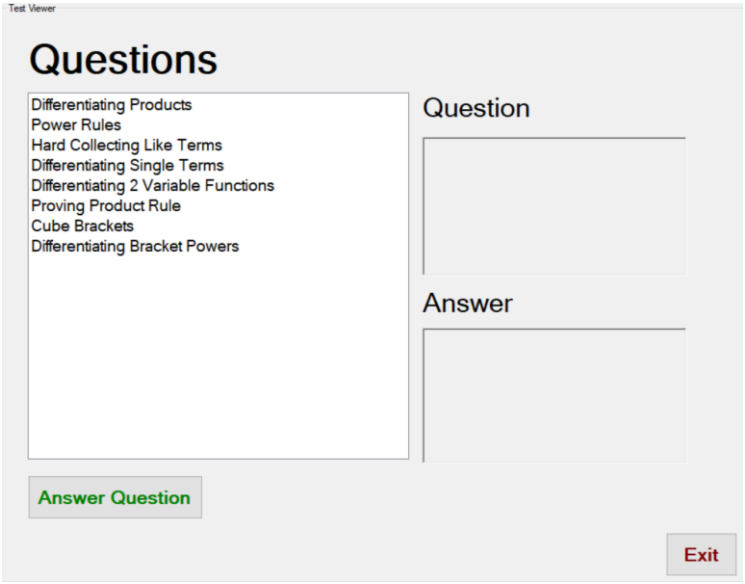


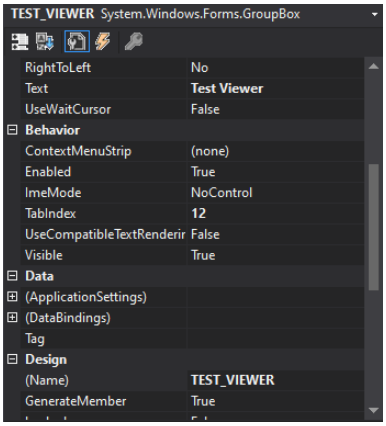
Doing so, my groupboxes **TEST_AREA_LIST** updates.



As you can see the name and description are displayed here. The name being on the **TESTS_AREA_TEST_DESCRIPTION** text.

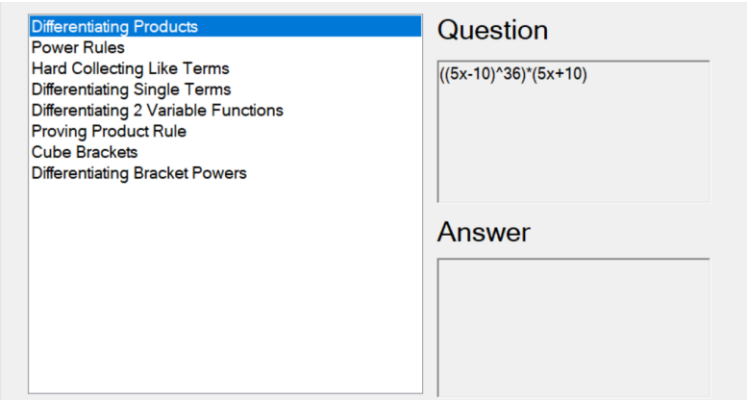
With this, I will now click the **TESTS_AREA_VIEW_TEST** button.



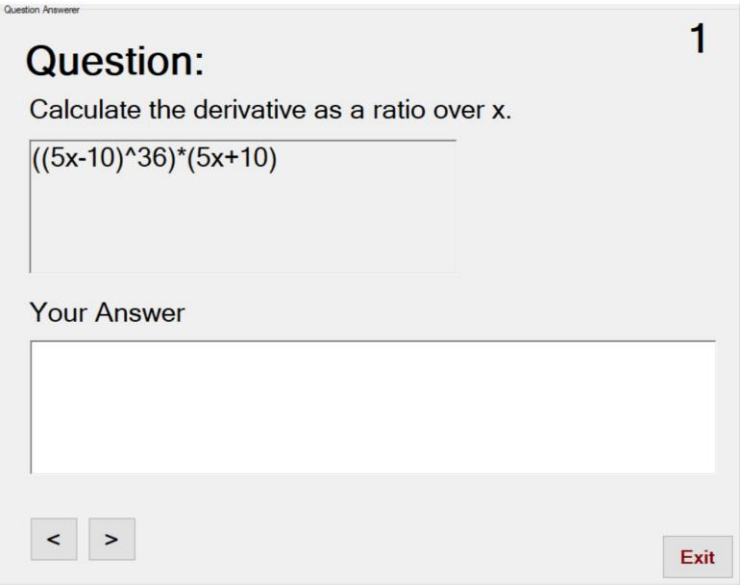


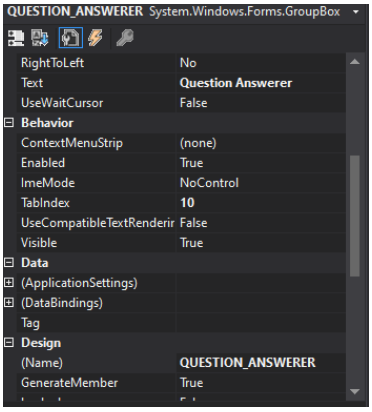
This is a new groupbox that displays every question I create, and is used by the student to answer them. The listbox **TEST_VIEWER_QUESTION_LIST** displays these. When I select a question, it updates the two boxes beside it. The question text box is called **TEST_VIEWER_PREVIEW_QUESTION**, and the answer called **TEST_VIEWER_PREVIEW_ANSWER.**

For example, selecting the first one changes it to this:



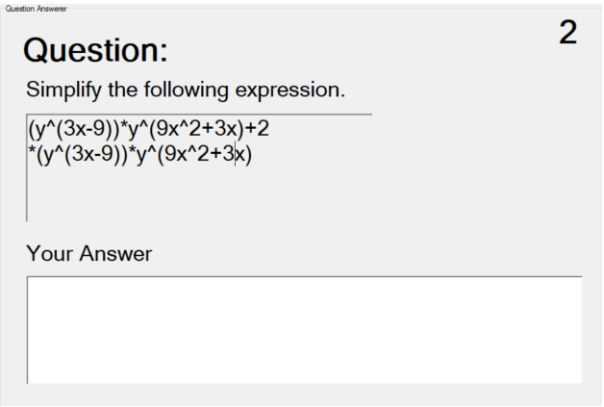
Now, I will click the button **TEST_VIEWER_ANSWER_SELECTED_QUESTION** to answer the question I have selected.



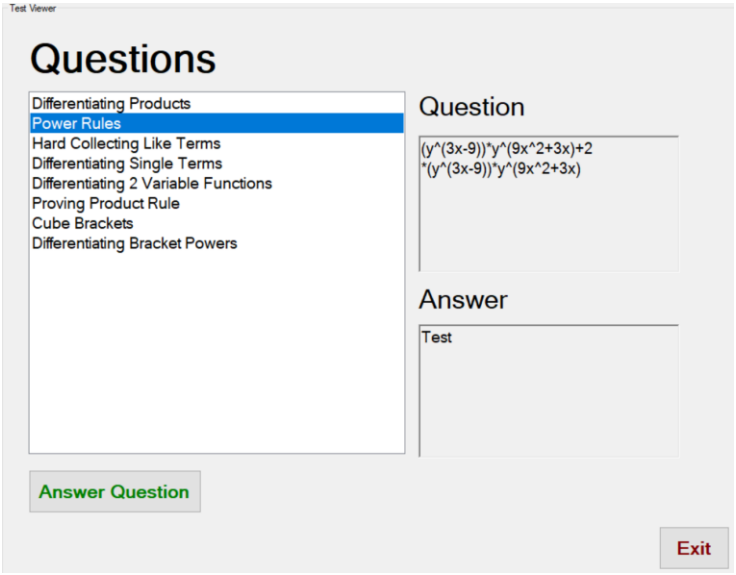


This is similar to the groupbox for adding new questions, but it is suited to the student. The text box **QUESTION_ANSWERER_ANSWER** is where the student will put their answer. The two left and right arrow buttons (**QUESTION_ANSWERER_LEFT** and **QUESTION_ANSWERER_RIGHT** respectively) are used to traverse the test with ease.

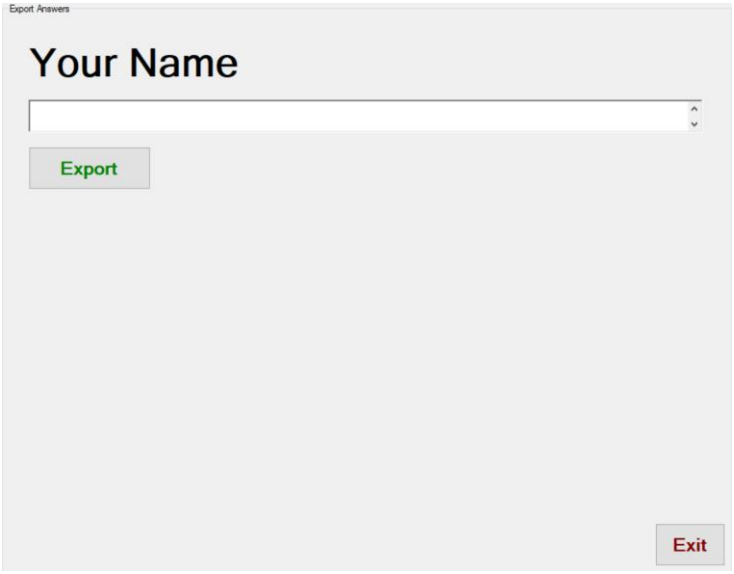
Clicking the right arrow once will move my question number up one:

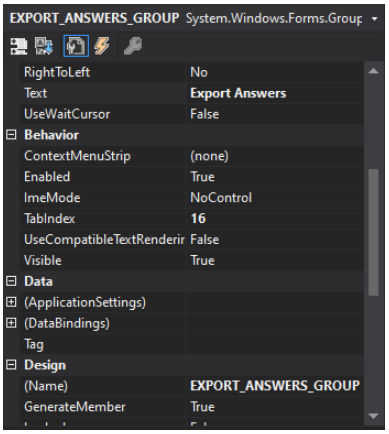


If I add 'Test' to the answer box, and then exit, the preview will change to this:



This is the basic function of answering a test. Clicking exit once more I will be once again greeted with the test screen. This time I wish to export my 'answered' test, so that I can send it back to the teacher. To do this I need to click the 'Export' button (**TESTS_AREA_EXPORT_TEST**).





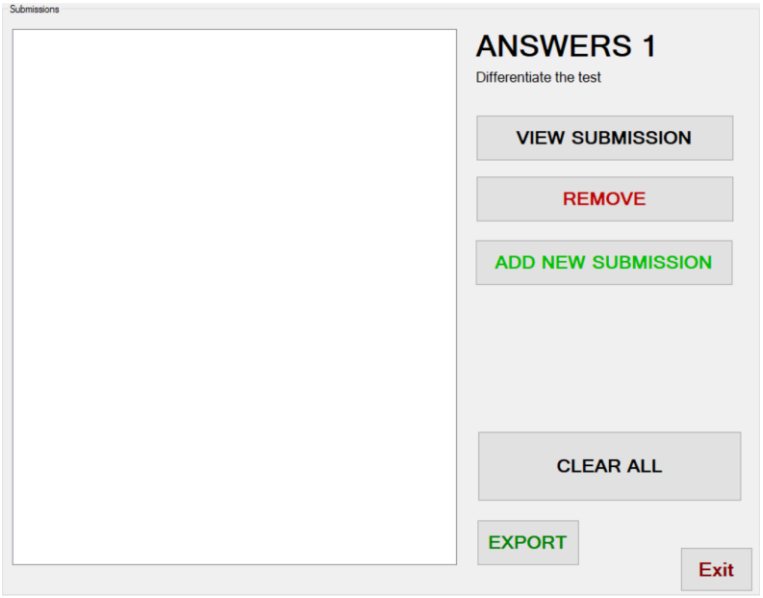
This only has a name input (**EXPORT_ANSWERS_NAME**), and this is for the student. So, I click the export button, **EXPORT_ANSWERS_EXPORT.**

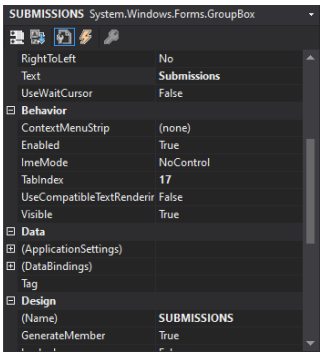
It is good to note that these files are able to be reopened by the student. I can re-add this test file at a later date and it will still display the answers I made. So, this can be seen as a save function, in case I want to finish it in increments instead of all at once.

But, saying I completed this test and sent it back to the teacher, I will now re-open the program and login as a teacher.

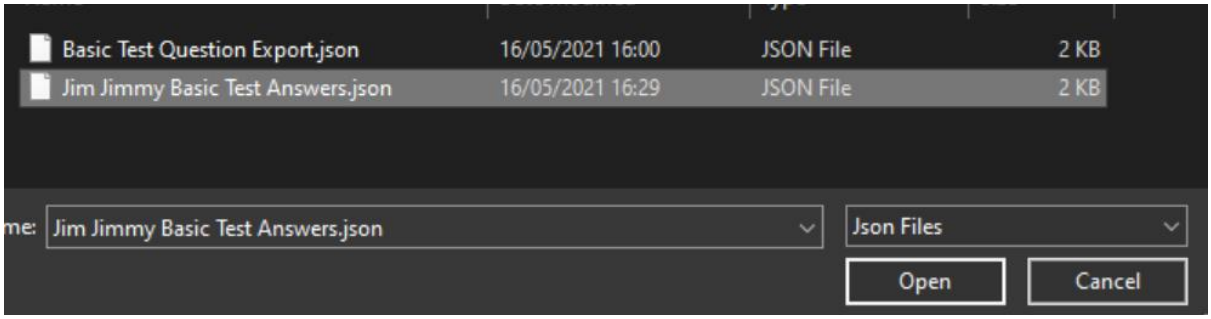
Teacher Area

Question Marking Area

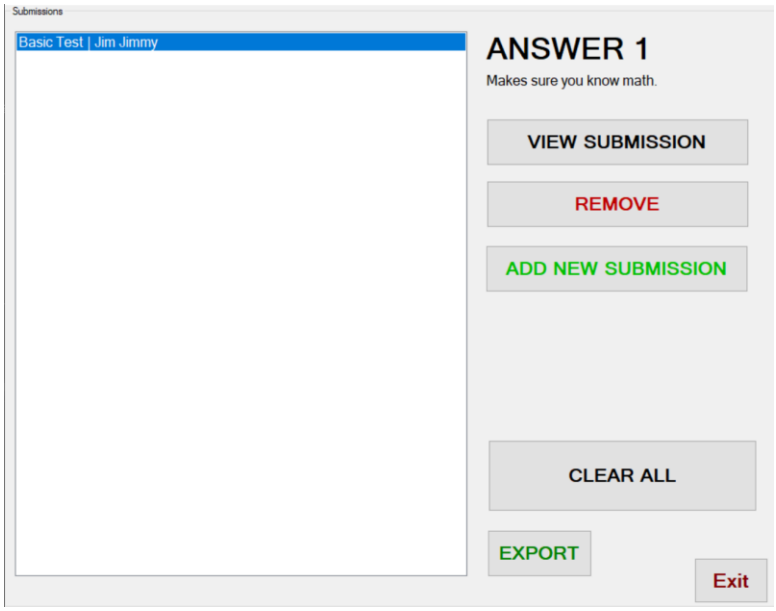




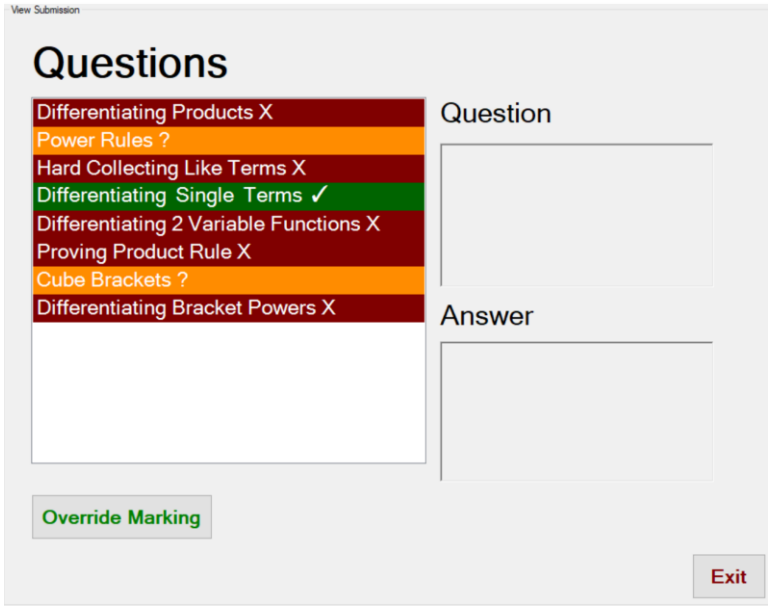
This is the third time I've used this design, and this time its for viewing submissions of students. Clicking **SUBMISSIONS_ADD,** I open a file explorer.

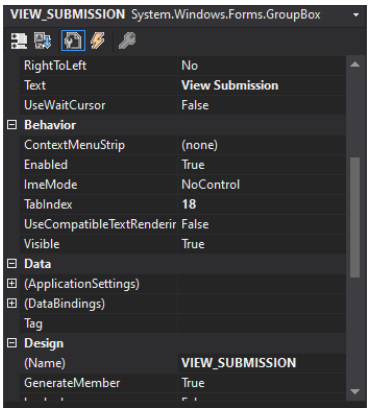


I select the answers file (though both will work).

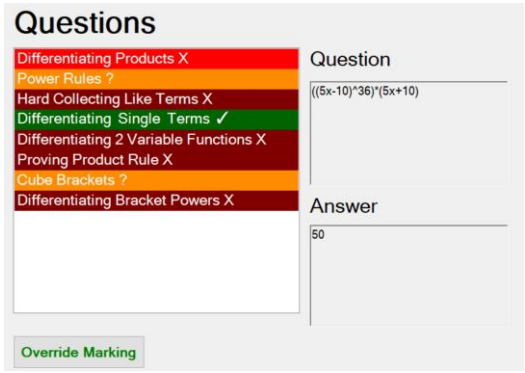


Now, clicking the **SUBMISSIONS_VIEW** button, I am greeted with this.

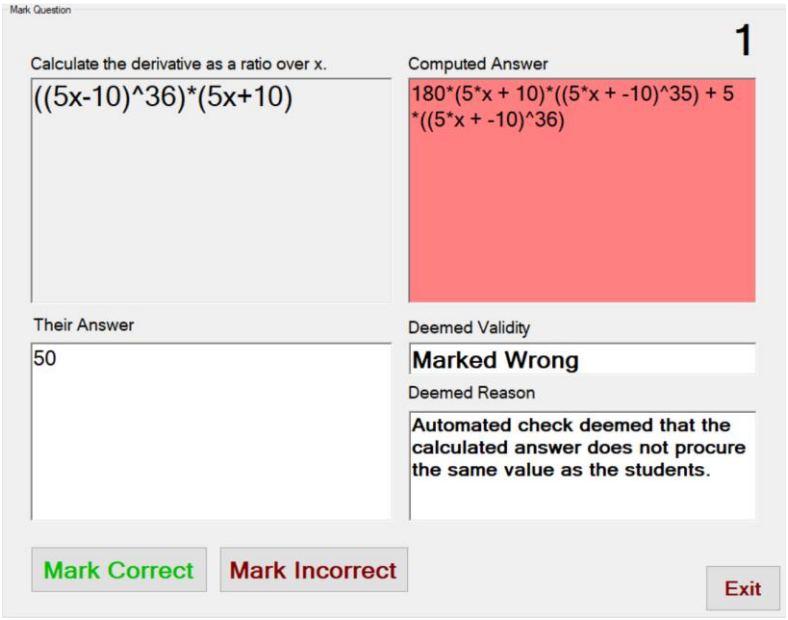


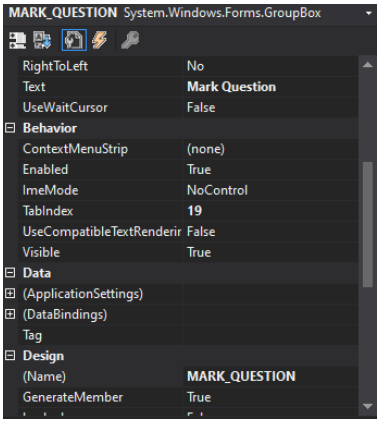


The program has automatically marked my test. I have answered all the questions to showcase how each situation function, so let's first go over when its wrong.



I will click the button **VIEW_SUBMISSION_OVERRIDE** to view the automated marking done by the computer.



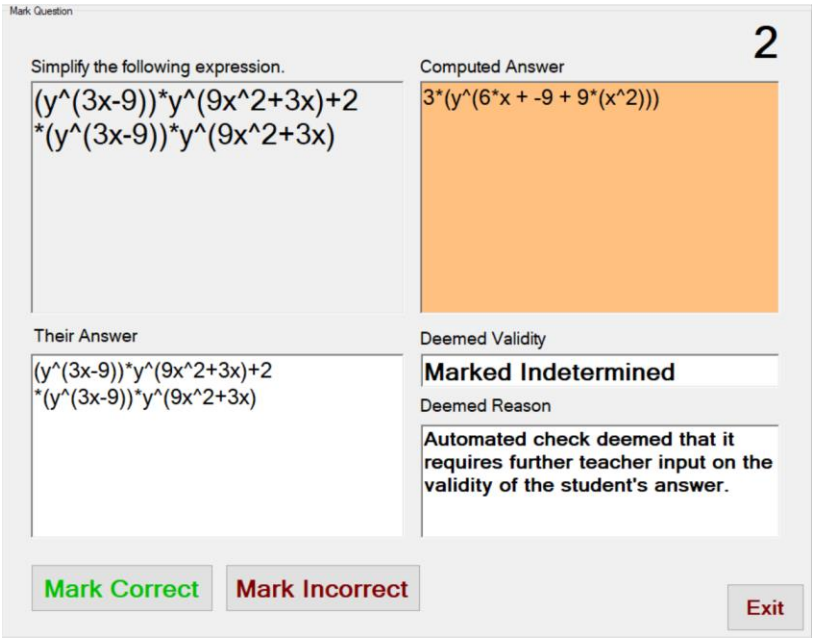


The first box **MARK_QUESTION_QUESTION** displays the question text, the next box **MARK_QUESTION_COMPUTED_ANSWER** displays what the computer thinks the answer should be.

The student's answer is displayed in the **MARK_QUESTION_STUDENTS_ANSWER** box; the validity being **MARK_QUESTION_DEEMED_VALIDITY**, and the reason **MARK_QUESTION_DEEMED_REASON**.

The teacher can override this, but it is obviously right, so I will move on and exit this question.

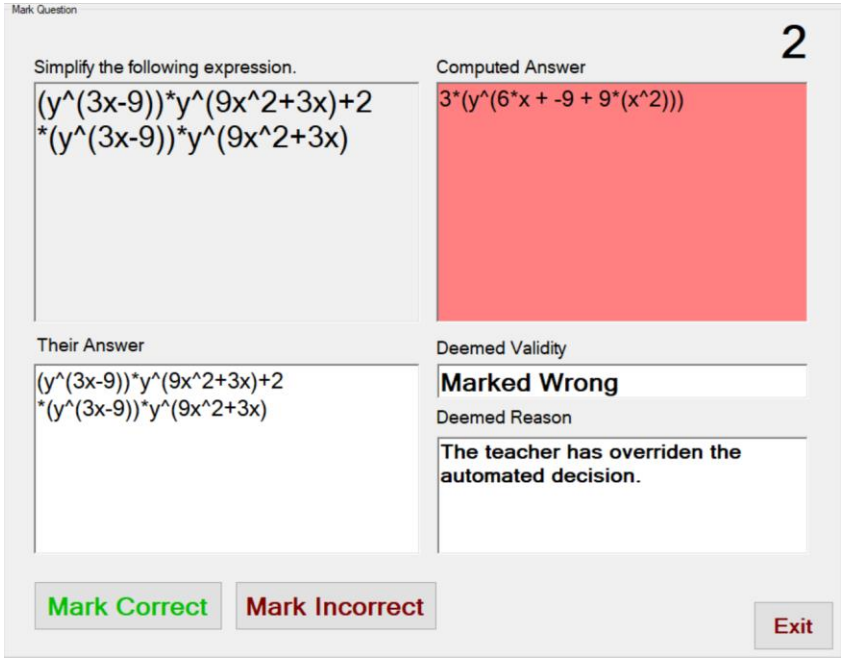
Instead, I will select the second answer:



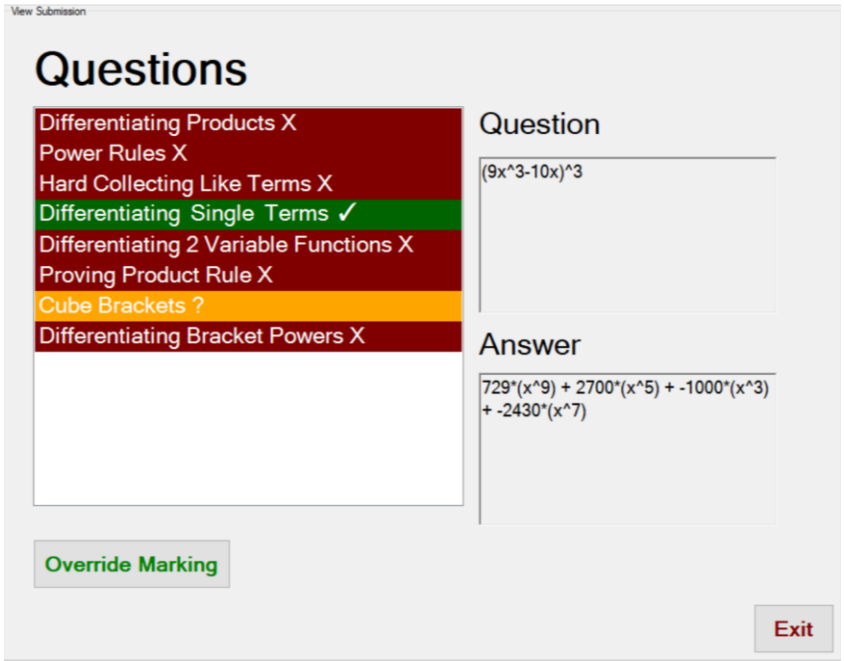
Why was it marked indetermined? This is because both answers procure the same **value**. I check the answers by substituting a value for x, and then seeing whether the students and computers answer matches. In this case, they do. But I asked the student to simplify, not copy and paste.

So, the computer marks it as indetermined so that the teacher can check whether it is actually correct. This is not an issue for differentiation, as you cannot just copy and paste the question. The derivative procures a different value.

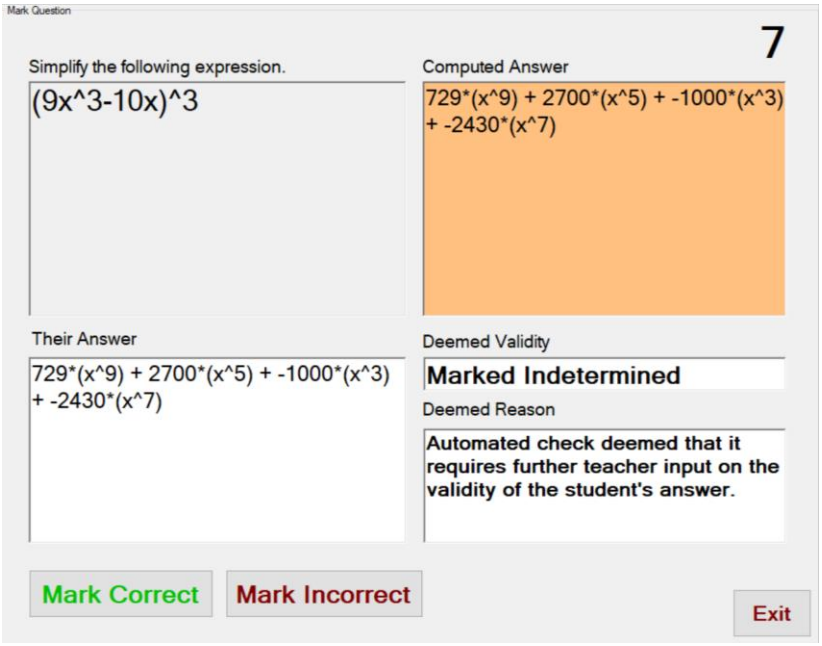
Anyway, I can see that they are incorrect, so I will click the **MARK_QUESTION_INCORRECT** button.



Going back, this is now the scenario.

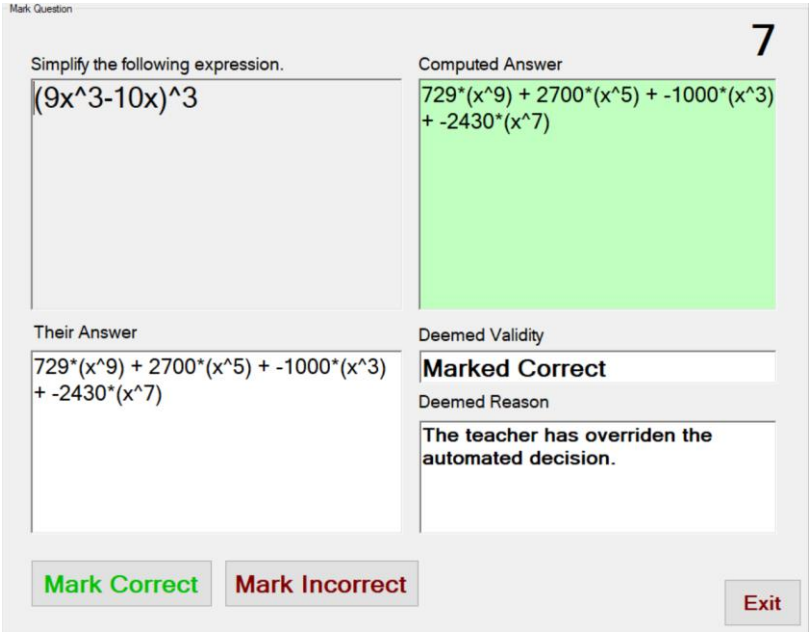


I will showcase the other possible scenarios. First, when the simplification is actually correct. I done this for the 'Cube Brackets' question.

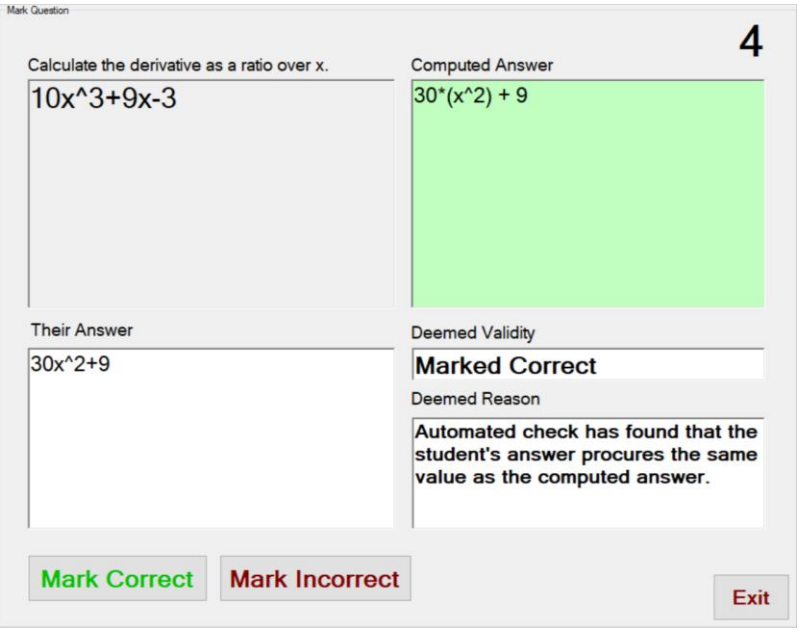


As you can see this is actually correct. I cannot have the computer check this without designing a rather elaborate system (although a string check would do for this situation, but remember that students won't write the answer in the exact order of the computed answer… unless they were cheating).

So, I will mark this as correct using the **MARK_QUESTION_CORRECT** button.

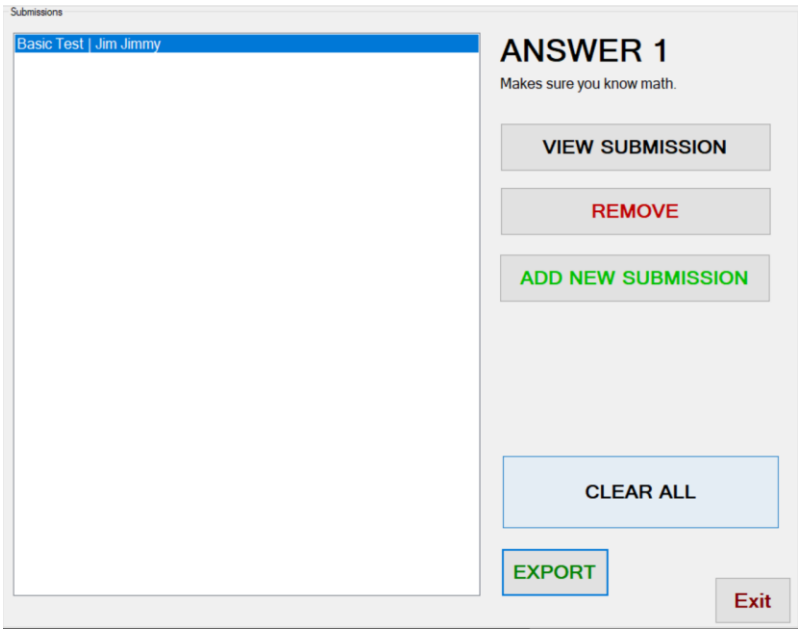


The last scenario is when the computer has marked a question correct. This can only happen for differentiation questions.



These are all the possible scenarios for the teacher. Now, confirming that the marking is correct, I will go back to the main screen. I want to export this so that I can send it back to student, and they can see their marking.

I merely need to click the **SUBMISSIONS_EXPORT** button.

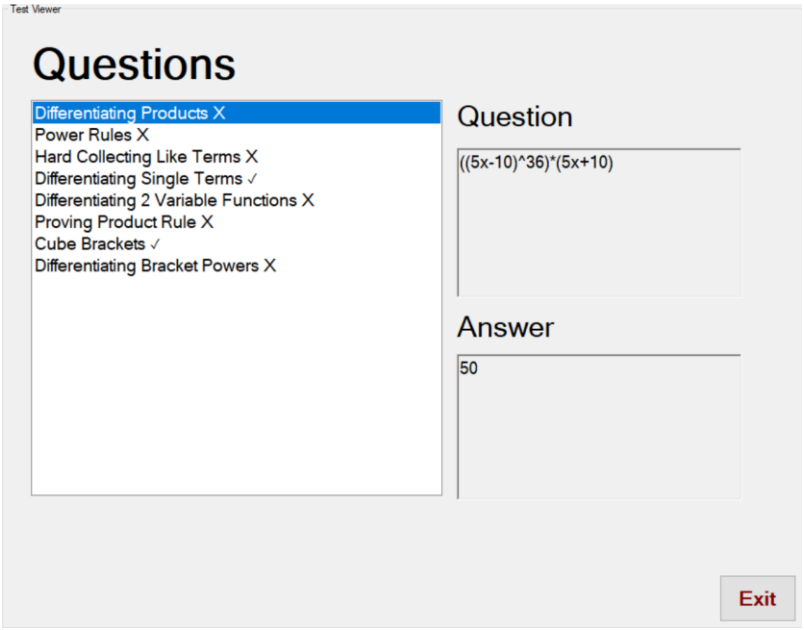


It will create this file:

Student Area:

There is no special area for looking at marked answers, as the student only needs to open the json file and the program will correct itself.

Adding the test file, and then viewing the test in the student area, it will display the questions like this:



Each will either have a tick or a cross next to them. As you can see, it follows what the teacher marked.

Technical Solution

Code

This will be the code of the final prototype, and will be split into the different classes used.

DATA_HANDLE.vb

This is the area that is used to contain the list of **QUESTIONS**. This is used to create 'Tests' that contains these questions. This is not called 'Tests' as it has multiple use cases for its general form.

```
' This handles the exporting/importing of a set of questions.

Public Enum QUESTION_TYPE
    COLLECTING_LIKE_TERMS
    HARD_COLLECTING_LIKE_TERMS
    EXPANDING_SQUARES
    EXPANDING_3_VARIABLE_BRACKETS
    POWER_RULES
    CUBE_BRACKETS
    DIFFERENTIATING_SINGLE_TERMS
    DIFFERENTIATING_PRODUCTS
```

```
    DIFFERENTIATING_BRACKET_POWERS
    DIFFERENTIATING_FRACTIONS
    DIFFERENTIATING_3_TERM_PRODUCTS
    DIFFERENTIATING_2_VARIABLE_FUNCTIONS
    DIFFERENTIATING_3_VARIABLE_FUNCTIONS
    PROVING_PRODUCT_RULE
End Enum



Class DATA_HANDLE_TEACHER : Inherits DATA_HANDLE ' For Teachers.
    Public QUESTIONS As New List(Of TEACHER_QUESTION)
End Class
Class DATA_HANDLE

    Private MARK_SYMBOLS As Dictionary(Of QUESTION_STATUS, String) = New Dictionary(Of QUES
TION_STATUS, String) From
    {{QUESTION_STATUS.CORRECT, "✓
"}, {QUESTION_STATUS.WRONG, "X"}, {QUESTION_STATUS.INDETERMINED, "?"}}

    Shadows QUESTIONS As New List(Of QUESTION)
    Protected Event QUESTION_LIST_CHANGE()
    Protected Form As Form
    Public QUESTION_DEFINERS As Dictionary(Of String, List(Of String))

    Public STUDENT_NAME As String = ""
    Public QUESTION_1 = "Simplify the following expression."
    Public QUESTION_2 = "Calculate the derivative as a ratio over x."

    Public DESCRIPTION As String = ""
    Public NAME As String = ""
    Public MARKED As Boolean = False

    Sub New(Optional STUDENT As Boolean = False)
        If Not STUDENT Then
            AddHandler QUESTION_LIST_CHANGE, AddressOf UPDATE_QUESTION_LIST
        End If
        ' A bit of data for the templates.Some of it is pretty dumb, like the question iden
tifiers. But I have 2 hours before this must be done :).
        QUESTION_DEFINERS = New Dictionary(Of String, List(Of String)) From {{"Collecting L
ike Terms", New List(Of String)({QUESTION_1, "3x+9z-10x^2+9y+10x-2y", "24z-10u+3y^2-
4u+23", "99x^2-6*9x+33y"})},
            {"Hard Collecting Like Terms", New List(Of String)({QUESTION_1, "33yx^2+99y^(x-
3)+2yx^2", "27zx^(x^2-3x)+99zx^2-23y", "37x^(x^2-3x)+99x^2-23y"})},
            {"Expanding Squares", New List(Of String)({QUESTION_1, "(x-10)^2", "(2x^(3x-
2)+-3y)^2", "(9x^(10x+3)+3x)^2"})},
            {"Expanding 3 Variable Brackets", New List(Of String)({QUESTION_1, "(9x^3-
10x+9y)^2", "(6x^2-y+10z)^2", "(9x^10-23x+2y)^2"})},
```

```
            {"Cube Brackets", New List(Of String)({QUESTION_1, "(9x^3-
10x)^3", "(6x^2+10z)^3", "(9x^10+2y)^3"})},
            {"Power Rules", New List(Of String)({QUESTION_1, "((x^(3x-
9))*x^(9x^2+3x))^3", "(y^(3x-9))*y^(9x^2+3x)+2*(y^(3x-
9))*y^(9x^2+3x)", "(x^(6x+18))^2+(x^(3x+9))^4"})},
            {"Differentiating Single Terms", New List(Of String)({QUESTION_2, "10x^3+9x-
3", "32x^3-2x^2+55x", "9x^2-10x"})},
            {"Differentiating Products", New List(Of String)({QUESTION_2, "((3x^2-
10x)^10)*33x", "((5x-10)^36)*(5x+10)", "((2x^3+10x)^5)*(9x^3+3)"})},
            {"Differentiating Bracket Powers", New List(Of String)({QUESTION_2, "((3x^2-
10x+99)^10)", "((5x-10-10x)^36)", "((2x^3+10x+3x^2)^5)"})},
            {"Differentiating Fractions", New List(Of String)({QUESTION_2, "(9x+3)/(10x-
3)", "((x-2)^2)/(10x+3)", "(32x^2+1)/((5x-2)^2)"})},
            {"Differentiating 3 Term Products", New List(Of String)({QUESTION_2, "(9x+3)*(1
0x-3)*((10x+3)^10)", "(99x^2+3)*((3x-2)^5)*((3x+2)^5)", "((5x-6x^2)^5)*((3x-
2)^5)*((3x+2)^5)"})},
            {"Differentiating 2 Variable Functions", New List(Of String)({QUESTION_2, "(2y+
3x)^2", "(5y-7x)*((13y^2+3)^3)", "(5y+3y^2)/(3x-10)"})},
            {"Differentiating 3 Variable Functions", New List(Of String)({QUESTION_2, "(2y+
3x+5z)^2", "(5y-7x-3yu)*((13y^2+3)^3)", "(5yu+3y^2)/(3xy-10)"})},
            {"Proving Product Rule", New List(Of String)({QUESTION_2, "uv"})}}
    End Sub

    Function REMOVE(QUESTION_OBJECT As QUESTION)
        QUESTIONS.Remove(QUESTION_OBJECT)
        RaiseEvent QUESTION_LIST_CHANGE()
        Return True
    End Function

    Function CLEAR_ALL()
        QUESTIONS = New List(Of QUESTION)
        RaiseEvent QUESTION_LIST_CHANGE()
    End Function

    Function ADD(QUESTION_OBJECT As QUESTION)
        QUESTIONS.Add(QUESTION_OBJECT)
        RaiseEvent QUESTION_LIST_CHANGE()
        Return True
    End Function

    Sub UPDATE_QUESTION_LIST()
        FORM1.Q_CONTROL_GROUP_LISTBOX.Items.Clear()
        For Each QUESTION As QUESTION In QUESTIONS
            FORM1.Q_CONTROL_GROUP_LISTBOX.Items.Add(QUESTION.TYPE)
        Next
    End Sub

    Sub UPDATE_TEST_LIST()
```

```
        FORM1.TEST_VIEWER_QUESTION_LIST.Items.Clear()
        For Each QUESTION As QUESTION In QUESTIONS
            Dim POSSIBLE_ADD As String = ""
            If MARKED Then
                Select Case QUESTION.STATUS
                    Case QUESTION_STATUS.WRONG
                        POSSIBLE_ADD = " " & MARK_SYMBOLS(QUESTION_STATUS.WRONG)
                    Case QUESTION_STATUS.CORRECT
                        POSSIBLE_ADD = " " & MARK_SYMBOLS(QUESTION_STATUS.CORRECT)
                    Case QUESTION_STATUS.INDETERMINED
                        POSSIBLE_ADD = " " & MARK_SYMBOLS(QUESTION_STATUS.INDETERMINED)
                End Select
            End If
            FORM1.TEST_VIEWER_QUESTION_LIST.Items.Add(QUESTION.TYPE & POSSIBLE_ADD)
        Next
    End Sub

    Function RETURN_QUESTIONS() As List(Of QUESTION)
        Return QUESTIONS
    End Function
End Class
```

EXPRESSION_TREE.vb

This class builds upon the formulated list created by **POSTFIX_EXPRESSION** and turns it into an expression tree.

```
Class EXPRESSION_TREE : Inherits POSTFIX_EXPRESSION ' This class builds upon the POSTFIX_EX
PRESSION and enables the creation of an expression tree. This class is to be used instead o
f POSTFIX_EXPRESSION when that feature is required.
    Dim RESULT_STACK As Stack(Of TREE_NODE) = New Stack(Of TREE_NODE)
    Public TREE_ROOT As TREE_NODE

    Public Sub CREATE_TREE()
        Dim ROOT As New TREE_NODE
        For Each LETTER As String In OUTPUT_POSTFIX ' Loops through the converted expressio
n list.
            'Console.WriteLine("ITEMN" & LETTER)
            If Not OPERATORS.ContainsKey(LETTER) Then ' This means it is an operand, such a
s 30.
                ROOT = New TREE_NODE ' Creates a node for the operand
                ROOT.VALUE = LETTER
                RESULT_STACK.Push(ROOT) ' Pushes it onto the stack
            Else ' It is an operator, like +
                ROOT = New TREE_NODE ' New tree node
                ROOT.VALUE = LETTER ' Makes the operator the value
                ROOT.RIGHT.Add(RESULT_STACK.Pop) ' Remember that the right and left are lis
ts of nodes. This is important later on.
                ROOT.LEFT.Add(RESULT_STACK.Pop) ' Pops the stack twice for the correct oper
ators.
                RESULT_STACK.Push(ROOT) ' Pushes the resultant node into the stack.
            End If
        Next
```

```
        TREE_ROOT = RESULT_STACK.Pop() ' This will be the root of the tree created.
    End Sub

    Sub New(Input As String)
        MyBase.New(Input)
    End Sub
End Class
```

FORM1.Vb

This is the code for the windows form. This holds the code that manipulates the groupboxes, thus a lot of events are involved. Nearly all of this stuff just utilises pre-existing classes, but functions like 'EXPORT' use JSON to create files.

```
' The Account Class contains the methods for authentication.
' Events 'Login' and 'Logout' are to be connected to the respective buttons.

' Items are named first as their page, in form page_x.

' GLOBAL ENUMS

' Login enum for the login_mode of the account.
' It is also used to state what mode the account is in (once logged in).
Imports System.Text.RegularExpressions
Imports System.IO
Imports System.Text.Json
Imports System.Text.Json.Serialization
Public Enum LOGIN_MODE
    TEACHER
    STUDENT
End Enum



' This enum contains all the available questions

' The class for the form.
Public Class FORM1

    ' Main Sub
    Private Sub Main(sender As Object, e As EventArgs) Handles Me.Shown
        SHOW_MODE_SCREEN()
    End Sub

    ' EVENTS

    Dim WithEvents ACCOUNT_OBJECT As New ACCOUNT
    Dim WithEvents NOTIFICATIONS_OBJECT As New NOTIFICATIONS(Me)
    Dim DATA_HANDLER
    Dim TESTS As New List(Of DATA_HANDLE)
    Dim QUESTION_POINTER As Integer = 0

    '////////////////////////////
    ' General Form Subroutines
    '////////////////////////////

    Public Function CLEAR_ALL()
        For Each cControl As Control In Controls  'Loops through all 'controls' in the form
```

```
            If TypeOf cControl Is GroupBox Then
                cControl.Visible = False
            End If
        Next
        Return True
    End Function

    Private Sub SHOW_LOGIN_SCREEN() Handles ACCOUNT_OBJECT.LOGIN_SCREEN
        CLEAR_ALL()
        LOGIN_GROUP.Visible = True
    End Sub

    Private Sub SHOW_MODE_SCREEN() Handles ACCOUNT_OBJECT.MODE_SCREEN
        CLEAR_ALL()
        MODE_GROUP.Visible = True
    End Sub

    Private Function TOGGLE_CERTAIN_SCREEN(SCREEN, SETTING)
        CLEAR_ALL()
        SCREEN.Visible = SETTING
        Return True
    End Function


    ' Account Login Subroutines

    Public Function UPPER_CASE(ByVal Match As Match) As String
        ' Used for .Replace regex to set a match upper case
        Return Match.ToString.ToUpper()
    End Function

    Public Sub ATTEMPT_LOGIN() Handles LOGIN_BUTTON.MouseClick ' This fires when the user c
licks the login buttonm
        Dim result As Boolean = ACCOUNT_OBJECT.LOGIN(LOGIN_INPUT.Text) ' Calls login method
 in the account object.
        If result Then 'if it was correct
            NOTIFICATIONS_OBJECT.ADD_NOTIFICATION("Correct Login Info! Transferring to " &
ACCOUNT_OBJECT.ACCOUNT_TYPE.ToString & " Screen", Color.Green)

            If ACCOUNT_OBJECT.ACCOUNT_TYPE = LOGIN_MODE.STUDENT Then
                TOGGLE_CERTAIN_SCREEN(TEST_AREA, True) ' Shows the default area for doing t
he tests given by teachers.

                ' Setup common events for the student.
                AddHandler TEST_VIEWER_EXIT.Click, Function(sender, e) TOGGLE_CERTAIN_SCREE
N(TEST_AREA, True)
                AddHandler TESTS_AREA_VIEW_TEST.Click, Function(sender, e) VIEW_TEST()
                AddHandler TESTS_AREA_REMOVE_TEST.Click, Function(sender, e) TEST_DELETE()
                AddHandler TEST_VIEWER_ANSWER_SELECTED_QUESTION.Click, Function(sender, e)
DISPLAY_ANSWER_QUESTION()
                AddHandler QUESTION_ANSWERER_EXIT.Click, Function(sender, e) TOGGLE_CERTAIN
_SCREEN(TEST_VIEWER, True)
                AddHandler QUESTION_ANSWERER_EXIT.Click, Function(sender, e) ANSWER_QUESTIO
N() ' This is easier trust me :L
                AddHandler QUESTION_ANSWERER_EXIT.Click, Function(sender, e) TEST_QUESTION_
LIST_MOUSE_UP()
```

```
                AddHandler EXPORT_ANSWERS_EXPORT.Click, Function(sender, e) EXPORT(sender,
e, TESTS(TEST_AREA_LIST.SelectedIndex), "Answers.json", EXPORT_ANSWERS_NAME.Text) ' The exp
ort function that I modified to accomodate for custom data_handle object inputs.
                AddHandler EXIT_EXPORT_ANSWERS.Click, Function(sender, e) TOGGLE_CERTAIN_SC
REEN(TEST_AREA, True)
            Else
                TOGGLE_CERTAIN_SCREEN(TEACHER_GROUP, True)
                AddHandler TEACHER_CREATE_QUESTIONS.Click, Function(sender, e) SETUP_QUESTI
ON_CREATOR()
                DATA_HANDLER = New DATA_HANDLE_TEACHER()

                ' Update the question chooser with the current available enums.
                Dim ENUMS As New List(Of QUESTION_TYPE)(System.Enum.GetValues(GetType(QUEST
ION_TYPE)))
                ' Setup common events for the teacher.
                AddHandler Q_CONTROL_GROUP_ADD_QUESTION.Click, Function(sender, e) SETUP_QU
ESTION_CHOOSER()
                AddHandler QUESTION_CHOOSER_BACK.Click, Function(sender, e) TOGGLE_CERTAIN_
SCREEN(Q_CONTROL_GROUP, True)
                AddHandler QUESTION_CHOOSER_CREATE.Click, Function(sender, e) CREATE_QUESTI
ON()
                AddHandler QUESTION_CREATION_EXIT.Click, Function(sender, e) TOGGLE_CERTAIN
_SCREEN(QUESTION_CHOOSER, True)
                AddHandler QUESTION_CREATE.Click, Function(sender, e) TOGGLE_CERTAIN_SCREEN
(Q_CONTROL_GROUP, True)
                AddHandler Q_CONTROL_GROUP_EXIT.Click, Function(sender, e) TOGGLE_CERTAIN_S
CREEN(TEACHER_GROUP, True)
                AddHandler Q_CONTROL_GROUP_EXIT.Click, Function(sender, e) CLEAR_ALL_DATA()
                AddHandler Q_CONTROL_GROUP_EDIT.Click, Function(SENDER, E) EDIT()
                AddHandler Q_CONTROL_GROUP_REMOVE.Click, Function(sender, e) DELETE(Q_CONTR
OL_GROUP_LISTBOX.SelectedIndex)
                AddHandler Q_CONTROL_GROUP_CLEAR_ALL.Click, Function(sender, e) DATA_HANDLE
R.CLEAR_ALL()
                AddHandler Q_CONTROL_GROUP_EXPORT.Click, Function(sender, e) TOGGLE_CERTAIN
_SCREEN(TEST_EXPORT, True)
                AddHandler TEST_EXPORT_EXIT.Click, Function(sender, e) TOGGLE_CERTAIN_SCREE
N(Q_CONTROL_GROUP, True)

                ' Answer Submissions section
                AddHandler SUBMISSIONS_REMOVE.Click, Function(sender, e) TEST_DELETE(SUBMIS
SIONS_LIST.SelectedIndex) ' As there are a number of simularities between this and the stud
ent area I will recycle some functions.
                AddHandler MARK_QUESTION_CORRECT.Click, Function(sender, e) OVERRIDE_SUBMIS
SION_QUESTION(sender, e, QUESTION_STATUS.CORRECT)
                AddHandler MARK_QUESTION_INCORRECT.Click, Function(sender, e) OVERRIDE_SUBM
ISSION_QUESTION(sender, e, QUESTION_STATUS.WRONG)
                AddHandler MARK_QUESTION_EXIT.Click, Function(sender, e) TOGGLE_CERTAIN_SCR
EEN(VIEW_SUBMISSION, True)
                AddHandler MARK_QUESTION_EXIT.Click, Function(sender, e) UPDATE_LIST_OF_QUE
STIONS_ON_SELECTED_SUBMISSION()
                AddHandler VIEW_SUBMISSION_EXIT.Click, Function(sender, e) TOGGLE_CERTAIN_S
CREEN(SUBMISSIONS, True)
                AddHandler SUBMISSIONS_EXIT.Click, Function(sender, e) TOGGLE_CERTAIN_SCREE
N(TEACHER_GROUP, True)
                AddHandler SUBMISSIONS_EXIT.Click, Function(sender, e) CLEAR_ALL_DATA()
                AddHandler SUBMISSIONS_EXPORT.Click, Function(sender, e) EXPORT(sender, e,
TESTS(SUBMISSIONS_LIST.SelectedIndex), "Marked Answers.json", TESTS(SUBMISSIONS_LIST.Select
edIndex).STUDENT_NAME)
```

```
                ' Update the question chooser listview.
                For Each ENUM_ITEM As QUESTION_TYPE In ENUMS
                    Dim MODIFIABLE As String = ENUM_ITEM.ToString.ToLower()
                    Dim SPACE As New Regex("_")
                    Dim CAPITALISE As New Regex("[ ]\w|(^\w)") ' Gets every first letter.
                    MODIFIABLE = SPACE.Replace(MODIFIABLE, " ")
                    MODIFIABLE = CAPITALISE.Replace(MODIFIABLE, New MatchEvaluator(AddressO
f UPPER_CASE))
                    QUESTION_CHOOSER_LIST.Items.Add(MODIFIABLE)
                Next

            End If

        Else
            NOTIFICATIONS_OBJECT.ADD_NOTIFICATION("Wrong Login Info!", Color.Red)
        End If
    End Sub

    Public Sub MODE_CLICK(sender As Object, e As EventArgs) Handles MODE_STUDENT.MouseClick
, MODE_TEACHER.MouseClick
        If sender Is MODE_STUDENT Then ' This means they chose to be a student
            ACCOUNT_OBJECT.SELECT_MODE(LOGIN_MODE.STUDENT)
        Else ' They chose teacher
            ACCOUNT_OBJECT.SELECT_MODE(LOGIN_MODE.TEACHER)
        End If
    End Sub


    '////////////////////////////
    ' END
    '////////////////////////////

    Private Function CLEAR_ALL_DATA()
        TESTS = New List(Of DATA_HANDLE)
        DATA_HANDLER.CLEAR_ALL()
        Return True
    End Function

    '////////////////////////////
    ' Student UI Subroutines.
    '////////////////////////////

    Dim SELECTED_TEST As Integer = 0
    Private Sub EXPORT_ANSWERS(sender As Object, e As EventArgs) Handles TESTS_AREA_EXPORT_
TEST.Click

        ' I will be using much of the same method for exporting the answers of the student.
        If TEST_AREA_LIST.SelectedItems.Count = 1 Then '
            TOGGLE_CERTAIN_SCREEN(EXPORT_ANSWERS_GROUP, True)
        End If
    End Sub

    Private Sub CLEAR_ALL(sender As Object, e As EventArgs) Handles TESTS_AREA_CLEAR_ALL.Cl
ick
        TESTS.RemoveRange(0, TESTS.Count)
        TEST_LIST_UPDATE()
    End Sub
```

```
    Private Sub TEST_LIST_UPDATE() ' Updates the test list.
        TEST_AREA_LIST.Items.Clear()
        For Each TEST As DATA_HANDLE In TESTS
            TEST_AREA_LIST.Items.Add(TEST.NAME)
        Next
    End Sub
    Private Function VIEW_TEST()
        If TEST_AREA_LIST.SelectedItems.Count = 1 Then '
            Dim INDEX_OF_ITEM = TEST_AREA_LIST.SelectedIndex
            SELECTED_TEST = INDEX_OF_ITEM
            TESTS(INDEX_OF_ITEM).UPDATE_TEST_LIST() ' This updates the list of questions in
 the selected test.
            TOGGLE_CERTAIN_SCREEN(TEST_VIEWER, True)
            If TESTS(INDEX_OF_ITEM).MARKED Then ' If the test is marked then they cannot an
swer it anymore.
                TEST_VIEWER_ANSWER_SELECTED_QUESTION.Visible = False
            Else
                TEST_VIEWER_ANSWER_SELECTED_QUESTION.Visible = True
            End If
            Return True
        End If
        Return False
    End Function

    Private Function ADD_TEST(ByVal SENDER As Object, ByVal E As EventArgs, Optional REVOKE
_EXTRAS As Boolean = False) Handles TESTS_AREA_ADD_NEW_TEST.Click
        '   Dim TO_BE_CONVERTED As String = TEST_INPUT_DATA_TEXT.Text
        '  Debug.WriteLine(TEST_INPUT_DATA_TEXT.Text)

        Dim DIALOG As New OpenFileDialog()
        DIALOG.Filter = "Json Files|*.json"
        If DialogResult.OK = DIALOG.ShowDialog Then
            Dim JSON_STRING As String = File.ReadAllText(DIALOG.FileName)
            Dim QUESTION_LIST As List(Of Dictionary(Of String, String)) = JsonSerializer.De
serialize(Of List(Of Dictionary(Of String, String)))(JSON_STRING)  ' Decserialiszes the dat
a yeahhhh!!!
            Dim NEW_TEST
            If Not REVOKE_EXTRAS Then
                NEW_TEST = New DATA_HANDLE(True)
            Else
                NEW_TEST = New DATA_HANDLE_TEACHER()
            End If

            NEW_TEST.NAME = QUESTION_LIST(0).Item("NAME") ' Adds the metadata for the test.
            NEW_TEST.DESCRIPTION = QUESTION_LIST(0).Item("DESCRIPTION")

            If QUESTION_LIST(0).Item("MARKED") = "False" Then ' This deems whether it has b
een marked by a teacher.
                NEW_TEST.MARKED = False
            ElseIf QUESTION_LIST(0).Item("MARKED") = "True" Then
                NEW_TEST.MARKED = True
            End If
            If QUESTION_LIST(0).ContainsKey("STUDENT NAME") Then
                NEW_TEST.STUDENT_NAME = QUESTION_LIST(0).Item("STUDENT NAME")
            End If
            QUESTION_LIST.RemoveAt(0) ' Removes the metadata
```

```
            For Each QUESTION As Dictionary(Of String, String) In QUESTION_LIST
                Dim NEW_QUESTION As New QUESTION()
                NEW_QUESTION.QUESTION_TEXT = QUESTION.Item("QUESTION")
                NEW_QUESTION.QUESTION_TITLE = QUESTION.Item("QUESTION TITLE")
                If QUESTION.Item("QUESTION TYPE") = "DIFFERENTIATION" Then
                    NEW_QUESTION.QUESTION_ANSWER_TYPE = QUESTION_TYPE_ANSWER.DIFFERENTIATIO
N
                Else
                    NEW_QUESTION.QUESTION_ANSWER_TYPE = QUESTION_TYPE_ANSWER.SIMPLIFICATION
                End If
                If Not QUESTION.ContainsKey("TEACHER EDITED") Then
                    NEW_QUESTION.TEACHER_EDITED = False
                ElseIf QUESTION.Item("TEACHER EDITED") = "False" Then ' For the submission
data.
                    NEW_QUESTION.TEACHER_EDITED = False
                ElseIf QUESTION.Item("TEACHER EDITED") = "True" Then
                    NEW_QUESTION.TEACHER_EDITED = True
                End If

                If QUESTION.ContainsKey("STATUS") Then ' Status enum serialisation, which p
robably can be done better but it works hahahahahahahahaaaaaaa
                    Select Case QUESTION.Item("STATUS")
                        Case "CORRECT"
                            NEW_QUESTION.STATUS = QUESTION_STATUS.CORRECT
                        Case "INDETERMINED"
                            NEW_QUESTION.STATUS = QUESTION_STATUS.INDETERMINED
                        Case "WRONG"
                            NEW_QUESTION.STATUS = QUESTION_STATUS.WRONG
                    End Select
                End If

                NEW_QUESTION.TYPE = QUESTION.Item("TYPE")
                If QUESTION.ContainsKey("ANSWER") Then ' If the user is adding their answer
 document back in, then they go back to where they were.
                    NEW_QUESTION.SUBMIT_ANSWER(QUESTION.Item("ANSWER"))
                End If
                NEW_TEST.ADD(NEW_QUESTION)
            Next
            TESTS.Add(NEW_TEST)
                If Not REVOKE_EXTRAS Then
                    TEST_LIST_UPDATE()
                    TOGGLE_CERTAIN_SCREEN(TEST_AREA, True)
                    NOTIFICATIONS_OBJECT.ADD_NOTIFICATION("Test " & NEW_TEST.NAME & " has b
een successfully added.", Color.Orange)
                End If
            End If
            Return True
    End Function

    Private Function MOVE_LEFT_QUESTION() Handles QUESTION_ANSWERER_LEFT.Click ' This is fo
r moving left in the test.
        ANSWER_QUESTION() 'Auto updates the current question.
        If QUESTION_POINTER > 0 Then
            QUESTION_POINTER -= 1
            DISPLAY_ANSWER_QUESTION() ' Displays the new question, even if it does some red
undant things ;0.
        End If
    End Function
```

```
    Private Function MOVE_RIGHT_QUESTION() Handles QUESTION_ANSWERER_RIGHT.Click ' This is
for moving left in the test.
        ANSWER_QUESTION() 'Auto updates the current question.
        If QUESTION_POINTER < TESTS(SELECTED_TEST).RETURN_QUESTIONS.Count() - 1 Then
            QUESTION_POINTER += 1
            DISPLAY_ANSWER_QUESTION() ' Displays the new question, even if it does some red
undant things ;0.
        End If
    End Function

    Private Function DISPLAY_ANSWER_QUESTION() ' Displays the question
        Dim SELECTED_QUESTION = TESTS(SELECTED_TEST).RETURN_QUESTIONS()(QUESTION_POINTER)
        QUESTION_ANSWERER_QUESTION.Text = SELECTED_QUESTION.RETURN_QUESTION
        QUESTION_ANSWERER_TITLE.Text = SELECTED_QUESTION.RETURN_QUESTION_TITLE
        QUESTION_ANSWERER_ANSWER.Text = SELECTED_QUESTION.RETURN_ANSWER
        QUESTION_ANSWERER_NUMBER.Text = QUESTION_POINTER + 1
        TOGGLE_CERTAIN_SCREEN(QUESTION_ANSWERER, True)
        Return True
    End Function

    Private Function ANSWER_QUESTION() ' Answers the question via the user.
        Dim SELECTED_QUESTION = TESTS(SELECTED_TEST).RETURN_QUESTIONS()(QUESTION_POINTER)
        SELECTED_QUESTION.SUBMIT_ANSWER(QUESTION_ANSWERER_ANSWER.Text)
        'TEST_QUESTION_LIST_MOUSE_UP()
        Return True
    End Function
    Private Function TEST_DELETE(Optional INDEX As Integer = -1)
        If INDEX = -1 Then
            INDEX = TEST_AREA_LIST.SelectedIndex
        End If
        If TESTS.Count >= INDEX And INDEX <> -1 Then
            Dim SELECTED As Integer = INDEX
            TESTS.RemoveAt(INDEX)
            TEST_LIST_UPDATE()
            SUBMISSIONS_LIST_UPDATE()
            NOTIFICATIONS_OBJECT.ADD_NOTIFICATION("Test " & SELECTED + 1 & " Deleted.", Col
or.Red)
            Return True
        End If
        Return False
    End Function

    Private Function TEST_QUESTION_LIST_MOUSE_UP(Optional ByVal SENDER As Object = Nothing,
 Optional ByVal E As System.Windows.Forms.MouseEventArgs = Nothing) Handles TEST_VIEWER_QUE
STION_LIST.MouseUp
        ' This shows the options when you right click on the question viewer for the teache
r.
        Dim SELECTED_TEST = TEST_AREA_LIST.SelectedIndex
        Dim SELECTED_TEST_QUESTION = TEST_VIEWER_QUESTION_LIST.SelectedIndex  ' The questio
n within the test.
        If TEST_VIEWER_QUESTION_LIST.SelectedItems.Count = 1 Then
            QUESTION_POINTER = SELECTED_TEST_QUESTION ' This is for use when the user is mo
ving through the questions in the question answer page.
            TEST_VIEWER_PREVIEW_QUESTION.Text = TESTS(SELECTED_TEST).RETURN_QUESTIONS()(SEL
ECTED_TEST_QUESTION).RETURN_QUESTION ' Sets the preview of the question and answer.
            TEST_VIEWER_PREVIEW_ANSWER.Text = TESTS(SELECTED_TEST).RETURN_QUESTIONS()(SELEC
TED_TEST_QUESTION).RETURN_ANSWER
```

```
        End If
        Return True
    End Function
    Private Sub STUDENT_LISTBOX_MOUSE_UP(ByVal SENDER As Object, ByVal E As System.Windows.
Forms.MouseEventArgs) Handles TEST_AREA_LIST.MouseUp
        ' This shows the options when you right click on the question viewer for the teache
r.
        Dim CMS = New ContextMenuStrip
        Dim SELECTED_ITEM = TEST_AREA_LIST.SelectedItem
        If E.Button = MouseButtons.Right Then
            If TEST_AREA_LIST.SelectedItems.Count = 1 Then
                Dim ITEM1 = CMS.Items.Add("View " & SELECTED_ITEM.ToString)
                ITEM1.Tag = 1
                AddHandler ITEM1.Click, AddressOf VIEW_TEST
                Dim ITEM2 = CMS.Items.Add("Delete " & SELECTED_ITEM.ToString)
                ITEM2.Tag = 2
                AddHandler ITEM2.Click, Function(s, ev) TEST_DELETE()
            End If
            Dim ITEM3 = CMS.Items.Add("Add new test")
            ITEM3.Tag = 3
            'AddHandler ITEM3.Click, ADD_TEST()
            CMS.Show(TEST_AREA_LIST, E.Location)
        ElseIf TEST_AREA_LIST.SelectedItems.Count = 1 Then
            Dim INDEX_OF_ITEM = TEST_AREA_LIST.SelectedIndex
            TESTS_AREA_TEST_TITLE.Text = "TEST " & (INDEX_OF_ITEM + 1)
            TESTS_AREA_TEST_DESCRIPTION.Text = TESTS(INDEX_OF_ITEM).DESCRIPTION
        End If
    End Sub


    '////////////////////////////
    ' END
    '////////////////////////////


    '////////////////////////////
    ' Teacher UI Subroutines.
    '////////////////////////////

    Private Function ADD_SUBMISSION(ByVal SENDER As Object, ByVal E As System.Windows.Forms
.MouseEventArgs) Handles SUBMISSIONS_ADD.Click
        ADD_TEST(SENDER, E, True) ' Adds a selected test through deserialisation.
        SUBMISSIONS_LIST_UPDATE()
        If TESTS.Count > 0 Then
            Dim ADDED_TEST As DATA_HANDLE = TESTS(TESTS.Count - 1)
            ADDED_TEST.MARKED = True ' This will allow the student to view the data and see
 the marks they got.
            For Each SELECTED_QUESTION In ADDED_TEST.RETURN_QUESTIONS
                If Not SELECTED_QUESTION.TEACHER_EDITED Then ' This means that the teacher
hasn't overridden the marking.
                    Dim BASELINE As Double = SELECTED_QUESTION.RETURN_ANSWER_FUNCTION_OUTPU
T(3)
                    Dim NEW_QUESTION As New SIMPLE_SIMPLIFY(SELECTED_QUESTION.RETURN_ANSWER
)
                    Dim TO_COMPARE As Double = NEW_QUESTION.GET_OUTPUT(3)
                    If BASELINE = TO_COMPARE And SELECTED_QUESTION.QUESTION_ANSWER_TYPE = Q
UESTION_TYPE_ANSWER.DIFFERENTIATION Then
                        ' I am only checking that the functions output the same values.
```

```
                        ' For differentiation where form doesn't matter, this is good enoug
h.
                        SELECTED_QUESTION.STATUS = QUESTION_STATUS.CORRECT
                    ElseIf BASELINE <> TO_COMPARE Then
                        ' This will always mean that it is wrong.
                        SELECTED_QUESTION.STATUS = QUESTION_STATUS.WRONG
                    ElseIf BASELINE = TO_COMPARE And SELECTED_QUESTION.QUESTION_ANSWER_TYPE
 = QUESTION_TYPE_ANSWER.SIMPLIFICATION Then
                        ' As 3x+2x = 5x, and substituting 3 for each will yield the same re
sult, I cannot say it is correct, as the user can change nothing and it will still say base
line = to_compare.
                        SELECTED_QUESTION.STATUS = QUESTION_STATUS.INDETERMINED
                        ' This must be checked by the teacher
                    End If
                End If
            Next

            Return True
        End If
        Return False
    End Function

    Private Function UPDATE_LIST_OF_QUESTIONS_ON_SELECTED_SUBMISSION()
        ' This updates the list of questions in the selected submission that the teacher ch
ose to view.
        Dim SELECTED_TEST As Integer = SUBMISSIONS_LIST.SelectedIndex
        VIEW_SUBMISSION_COLLECTION.Items.Clear()
        For Each QUESTION_O As QUESTION In TESTS.Item(SELECTED_TEST).RETURN_QUESTIONS
            Dim TO_BE_STRING As String = QUESTION_O.TYPE & " "
            Select Case QUESTION_O.STATUS
                Case QUESTION_STATUS.CORRECT
                    TO_BE_STRING = TO_BE_STRING & "✓"
                Case QUESTION_STATUS.WRONG
                    TO_BE_STRING = TO_BE_STRING & "X"
                Case QUESTION_STATUS.INDETERMINED
                    TO_BE_STRING = TO_BE_STRING & "?"
            End Select
            VIEW_SUBMISSION_COLLECTION.Items.Add(TO_BE_STRING)
        Next
    End Function

    Private Function OVERRIDE_SUBMISSION_QUESTION(SENDER As Object, E As EventArgs, TYPE As
 QUESTION_STATUS)
        Dim SELECTED_QUESTION As QUESTION = TESTS(SELECTED_TEST).RETURN_QUESTIONS()(VIEW_SU
BMISSION_COLLECTION.SelectedIndex)
        SELECTED_QUESTION.STATUS = TYPE
        SELECTED_QUESTION.TEACHER_EDITED = True
        OVERRIDE_SUBMISSION_QUESTION_PREP(SENDER, E) ' This will redisplay the page with th
e correct data.
    End Function
    Public Function OVERRIDE_SUBMISSION_QUESTION_PREP(ByVal SENDER As Object, ByVal E As Ev
entArgs) Handles VIEW_SUBMISSION_OVERRIDE.Click
        ' This is for overriding the submission's teacher selected question.

        If VIEW_SUBMISSION_COLLECTION.SelectedItems.Count > 0 Then
            Dim SELECTED_QUESTION As QUESTION = TESTS(SELECTED_TEST).RETURN_QUESTIONS()(VIE
W_SUBMISSION_COLLECTION.SelectedIndex) ' This is the question chosen to override marking.
            MARK_QUESTION_QUESTION.Text = SELECTED_QUESTION.RETURN_QUESTION ' The Question
```

```
            MARK_QUESTION_COMPUTED_ANSWER.Text = SELECTED_QUESTION.RETURN_COMPUTED_ANSWER ' the computed answer.
            MARK_QUESTION_STUDENTS_ANSWER.Text = SELECTED_QUESTION.RETURN_ANSWER ' The students answer.
            MARK_QUESTION_TITLE.Text = SELECTED_QUESTION.RETURN_QUESTION_TITLE
            MARK_QUESTION_NUMBER.Text = VIEW_SUBMISSION_COLLECTION.SelectedIndex + 1
            Select Case SELECTED_QUESTION.STATUS
                Case QUESTION_STATUS.CORRECT
                    MARK_QUESTION_COMPUTED_ANSWER.BackColor = Color.FromArgb(192, 255, 192)
                    MARK_QUESTION_DEEMED_VALIDITY.Text = "Marked Correct"
                    If Not SELECTED_QUESTION.TEACHER_EDITED Then
                        MARK_QUESTION_DEEMED_REASON.Text = "Automated check has found that the student's answer procures the same value as the computed answer."
                    Else
                        MARK_QUESTION_DEEMED_REASON.Text = "The teacher has overriden the automated decision."
                    End If
                Case QUESTION_STATUS.WRONG
                    MARK_QUESTION_COMPUTED_ANSWER.BackColor = Color.FromArgb(255, 128, 128)
                    MARK_QUESTION_DEEMED_VALIDITY.Text = "Marked Wrong"
                    Debug.WriteLine("qwrr" & SELECTED_QUESTION.TEACHER_EDITED.ToString)
                    If Not SELECTED_QUESTION.TEACHER_EDITED Then
                        MARK_QUESTION_DEEMED_REASON.Text = "Automated check deemed that the calculated answer does not procure the same value as the students."
                    Else
                        MARK_QUESTION_DEEMED_REASON.Text = "The teacher has overriden the automated decision."
                    End If
                Case QUESTION_STATUS.INDETERMINED
                    MARK_QUESTION_DEEMED_VALIDITY.Text = "Marked Indetermined"
                    MARK_QUESTION_COMPUTED_ANSWER.BackColor = Color.FromArgb(255, 192, 128)
                    MARK_QUESTION_DEEMED_REASON.Text = "Automated check deemed that it requires further teacher input on the validity of the student's answer."
            End Select
            TOGGLE_CERTAIN_SCREEN(MARK_QUESTION, True)
        End If
    End Function

    Public Function VIEW_SUBMISSION_EVENT(ByVal SENDER As Object, ByVal E As System.Windows.Forms.MouseEventArgs) Handles SUBMISSIONS_VIEW.Click
        ' This occurs when the teachers clicks the button to view the actual submitted test
        If SUBMISSIONS_LIST.SelectedItems.Count > 0 Then
            UPDATE_LIST_OF_QUESTIONS_ON_SELECTED_SUBMISSION()
            TOGGLE_CERTAIN_SCREEN(VIEW_SUBMISSION, True)
        End If
    End Function

    Private Sub VIEW_SUBMISSIONS_LIST_MOUSE_UP(ByVal SENDER As Object, ByVal E As System.Windows.Forms.MouseEventArgs) Handles VIEW_SUBMISSION_COLLECTION.MouseUp
        ' This shows the options when you right click on the question on the submission test section for the teacher.
        Dim CMS = New ContextMenuStrip
        Dim SELECTED_ITEM = VIEW_SUBMISSION_COLLECTION.SelectedItem
        If E.Button = MouseButtons.Right Then
            If VIEW_SUBMISSION_COLLECTION.SelectedItems.Count = 1 Then
                Dim ITEM1 = CMS.Items.Add("Override " & SELECTED_ITEM.ToString)
                ITEM1.Tag = 1
                AddHandler ITEM1.Click, Function(s, ev) VIEW_SUBMISSION_EVENT(s, ev)
```

```
            End If
            CMS.Show(VIEW_SUBMISSION_COLLECTION, E.Location)
        ElseIf VIEW_SUBMISSION_COLLECTION.SelectedItems.Count = 1 Then ' If they havent sel
ected any question.
            Dim INDEX_OF_ITEM = VIEW_SUBMISSION_COLLECTION.SelectedIndex
            VIEW_SUBMISSION_QUESTION.Text = TESTS(SUBMISSIONS_LIST.SelectedIndex).RETURN_QU
ESTIONS()(INDEX_OF_ITEM).RETURN_QUESTION
            VIEW_SUBMISSION_ANSWER.Text = TESTS(SUBMISSIONS_LIST.SelectedIndex).RETURN_QUES
TIONS()(INDEX_OF_ITEM).RETURN_ANSWER
        End If
    End Sub

    Private Sub SUBMISSIONS_LIST_MOUSE_UP(ByVal SENDER As Object, ByVal E As System.Windows
.Forms.MouseEventArgs) Handles SUBMISSIONS_LIST.MouseUp
        ' This shows the options when you right click on the question viewer for the teache
r.
        Dim CMS = New ContextMenuStrip
        Dim SELECTED_ITEM = SUBMISSIONS_LIST.SelectedItem
        If E.Button = MouseButtons.Right Then
            If SUBMISSIONS_LIST.SelectedItems.Count = 1 Then
                Dim ITEM1 = CMS.Items.Add("View " & SELECTED_ITEM.ToString)
                ITEM1.Tag = 1
                AddHandler ITEM1.Click, Function(s, ev) VIEW_SUBMISSION_EVENT(s, ev)
                Dim ITEM2 = CMS.Items.Add("Delete " & SELECTED_ITEM.ToString)
                ITEM2.Tag = 2
                AddHandler ITEM2.Click, Function(s, ev) TEST_DELETE(SUBMISSIONS_LIST.Select
edIndex)
            End If
            Dim ITEM3 = CMS.Items.Add("Add new submission")
            ITEM3.Tag = 3
            AddHandler ITEM3.Click, Function(s, ev) ADD_SUBMISSION(SENDER, E)
            CMS.Show(SUBMISSIONS_LIST, E.Location)
        ElseIf SUBMISSIONS_LIST.SelectedItems.Count = 1 Then ' If they havent selected any
question.
            Dim INDEX_OF_ITEM = SUBMISSIONS_LIST.SelectedIndex
            SUBMISSION_TITLE.Text = "ANSWER " & (INDEX_OF_ITEM + 1)
            SUBMISSIONS_DESCRIPTION.Text = TESTS(INDEX_OF_ITEM).DESCRIPTION
        End If
    End Sub

    Private Sub VIEW_SUBMISSION_COLLECTION_HOVER(SENDER As System.Object, E As EventArgs) H
andles VIEW_SUBMISSION_COLLECTION.SelectedIndexChanged
        ' This is required so that it looks like something is happening when you hover.

        VIEW_SUBMISSION_COLLECTION.Refresh() ' I have to refresh it every single time and i
t is appalling who made this system ahhh
        ' E.Item.BackColor = Color.Lime
    End Sub

    Private Sub VIEW_SUBMISSION_COLLECTION_DRAW(sender As System.Object, e As System.Window
s.Forms.DrawItemEventArgs) Handles VIEW_SUBMISSION_COLLECTION.DrawItem
        e.DrawBackground()
        ' This basically colour codes the questions based on their status.
        Dim TEXT_COLOUR = Brushes.White
        If VIEW_SUBMISSION_COLLECTION.SelectedIndex = e.Index Then
            If VIEW_SUBMISSION_COLLECTION.Items(e.Index).ToString().Contains("?") Then
                e.Graphics.FillRectangle(Brushes.Orange, e.Bounds)
            ElseIf VIEW_SUBMISSION_COLLECTION.Items(e.Index).ToString().Contains("✓") Then
```

```
                e.Graphics.FillRectangle(Brushes.LightGreen, e.Bounds)
            ElseIf VIEW_SUBMISSION_COLLECTION.Items(e.Index).ToString().Contains("X") Then
                e.Graphics.FillRectangle(Brushes.Red, e.Bounds)
            End If
        ElseIf VIEW_SUBMISSION_COLLECTION.Items(e.Index).ToString().Contains("?") Then
            e.Graphics.FillRectangle(Brushes.DarkOrange, e.Bounds)
        ElseIf VIEW_SUBMISSION_COLLECTION.Items(e.Index).ToString().Contains("✓") Then
            e.Graphics.FillRectangle(Brushes.DarkGreen, e.Bounds)
        ElseIf VIEW_SUBMISSION_COLLECTION.Items(e.Index).ToString().Contains("X") Then
            e.Graphics.FillRectangle(Brushes.Maroon, e.Bounds)
        End If
        e.Graphics.DrawString(VIEW_SUBMISSION_COLLECTION.Items(e.Index).ToString(), e.Font,
 TEXT_COLOUR, New System.Drawing.PointF(e.Bounds.X, e.Bounds.Y))
        e.DrawFocusRectangle()
    End Sub
    Private Function SELECTED_SUBMISSION_QUESTION_UPDATE() ' Updates the SUBMISSION list.
        SUBMISSIONS_LIST.Items.Clear()
        For Each TEST As DATA_HANDLE In TESTS
            SUBMISSIONS_LIST.Items.Add(TEST.NAME & " | " & TEST.STUDENT_NAME)
        Next
    End Function
    Private Function SUBMISSIONS_LIST_UPDATE() ' Updates the SUBMISSION list.
        SUBMISSIONS_LIST.Items.Clear()
        For Each TEST As DATA_HANDLE In TESTS
            SUBMISSIONS_LIST.Items.Add(TEST.NAME & " | " & TEST.STUDENT_NAME)
        Next
    End Function
    Private Function SETUP_QUESTION_CREATOR()
        TOGGLE_CERTAIN_SCREEN(Q_CONTROL_GROUP, True) ' Shows the question creator screen.
        Return True
    End Function

    Private Function SETUP_QUESTION_CHOOSER()
        TOGGLE_CERTAIN_SCREEN(QUESTION_CHOOSER, True) ' Shows the question creator chooser
screen.
        Return True
    End Function

    Private Function CREATE_QUESTION()
        If QUESTION_CHOOSER_LIST.SelectedItems.Count = 1 Then
            Dim NEW_QUESTION As New TEACHER_QUESTION(DATA_HANDLER)
            NEW_QUESTION.CHOSEN_QUESTION_TO_CREATE()
            TOGGLE_CERTAIN_SCREEN(QUESTION_INPUT1, True)
            QUESTION_CREATION_EXIT.Visible = True
            NOTIFICATIONS_OBJECT.ADD_NOTIFICATION("Question successfully created.", Color.G
reen)
        End If
        Return True
    End Function

    Private Function ENTER_SUBMISSION_AREA(ByVal SENDER As Object, ByVal E As System.Window
s.Forms.MouseEventArgs) Handles TEACHER_MARK_SUBMISSIONS.MouseUp
        If E.Button = MouseButtons.Left Then
            TESTS = New List(Of DATA_HANDLE)
            TOGGLE_CERTAIN_SCREEN(SUBMISSIONS, True)
        End If
    End Function
```

```
    Private Sub LISTBOX_MOUSE_UP(ByVal SENDER As Object, ByVal E As System.Windows.Forms.Mo
useEventArgs) Handles Q_CONTROL_GROUP_LISTBOX.MouseUp
        ' This shows the options when you right click on the question viewer for the teache
r.
        Dim CMS = New ContextMenuStrip
        Dim SELECTED_ITEM = Q_CONTROL_GROUP_LISTBOX.SelectedItem
        If E.Button = MouseButtons.Right Then
            If Q_CONTROL_GROUP_LISTBOX.SelectedItems.Count = 1 Then
                Dim ITEM1 = CMS.Items.Add("Edit " & SELECTED_ITEM.ToString)
                ITEM1.Tag = 1
                AddHandler ITEM1.Click, AddressOf EDIT
                Dim ITEM2 = CMS.Items.Add("Delete " & SELECTED_ITEM.ToString)
                ITEM2.Tag = 2
                AddHandler ITEM2.Click, Function(s, ev) DELETE(Q_CONTROL_GROUP_LISTBOX.Sele
ctedIndex)
            End If
            Dim ITEM3 = CMS.Items.Add("Add new question")
            ITEM3.Tag = 3
            AddHandler ITEM3.Click, AddressOf SETUP_QUESTION_CHOOSER
            CMS.Show(Q_CONTROL_GROUP_LISTBOX, E.Location)
        ElseIf Q_CONTROL_GROUP_LISTBOX.SelectedItems.Count = 1 Then ' If they havent select
ed any question.
            Dim INDEX_OF_ITEM = Q_CONTROL_GROUP_LISTBOX.SelectedIndex
            QUESTION_TITLE_NUMBER.Text = "QUESTION " & (INDEX_OF_ITEM + 1)
            QUESTION_TITLE_NAME.Text = DATA_HANDLER.RETURN_QUESTIONS()(INDEX_OF_ITEM).TYPE
        End If
    End Sub

    Private Function EDIT()
        If Q_CONTROL_GROUP_LISTBOX.SelectedItems.Count = 1 Then ' Edit the selected questio
n item.
            Dim INDEX_OF_ITEM = Q_CONTROL_GROUP_LISTBOX.SelectedIndex
            DATA_HANDLER.RETURN_QUESTIONS()(INDEX_OF_ITEM).EDIT_QUESTION(INDEX_OF_ITEM)
            TOGGLE_CERTAIN_SCREEN(QUESTION_INPUT1, True)
            QUESTION_CREATION_EXIT.Visible = False
            Return True
        End If
        Return False
    End Function

    Private Function DELETE(INDEX As Integer)
        If DATA_HANDLER.RETURN_QUESTIONS().Count >= INDEX And INDEX <> -1 Then
            DATA_HANDLER.REMOVE(DATA_HANDLER.RETURN_QUESTIONS().Item(INDEX))
            NOTIFICATIONS_OBJECT.ADD_NOTIFICATION("Question " & INDEX + 1 & " Deleted.", Co
lor.Red)
            Return True
        End If
        Return False
    End Function

    Private Function EXPORT(SENDER As Object, E As EventArgs, Optional TEST As DATA_HANDLE
= Nothing, Optional TEST_NAME As String = "Question Export.json", Optional STUDENT_NAME As
String = "") Handles TEST_EXPORT_EXPORT.Click

        Dim CHOSEN_TO_EXPORT = DATA_HANDLER
        Dim NAME = TEST_EXPORT_NAME.Text
        Dim DESCRIPTION = TEST_EXPORT_DESC.Text
```

```
        If Not TEST Is Nothing Then ' As the function is largely the same for exporting the
 student answers, I will just modify this to accomodate for it :P.
            CHOSEN_TO_EXPORT = TEST
            NAME = CHOSEN_TO_EXPORT.NAME
            DESCRIPTION = CHOSEN_TO_EXPORT.DESCRIPTION
        End If

        If CHOSEN_TO_EXPORT.RETURN_QUESTIONS().Count > 0 Then

            ' I aim to create a list of all the data I need.
            Dim STREAM As Stream = File.Open(My.Computer.FileSystem.SpecialDirectories.MyDo
cuments & "\" & STUDENT_NAME & " " & NAME & " " & TEST_NAME, FileMode.Create) ' The file I
will be exporting to.

            Dim DATA_LIST As New List(Of Dictionary(Of String, String))

            Dim META_DATA As New Dictionary(Of String, String) ' The name and description o
f the test.
            META_DATA.Add("NAME", NAME)
            META_DATA.Add("DESCRIPTION", DESCRIPTION)
            If Not STUDENT_NAME = "" Then
                META_DATA.Add("STUDENT NAME", STUDENT_NAME)
            End If
            META_DATA.Add("MARKED", CHOSEN_TO_EXPORT.MARKED.ToString)
            DATA_LIST.Add(META_DATA)

            For Each QUESTION As QUESTION In CHOSEN_TO_EXPORT.RETURN_QUESTIONS
                Dim QUESTION_DATA As New Dictionary(Of String, String)
                QUESTION_DATA.Add("QUESTION", QUESTION.RETURN_QUESTION) ' The question, lik
e 3x+3x
                QUESTION_DATA.Add("QUESTION TITLE", QUESTION.RETURN_QUESTION_TITLE) ' The t
itle, like "Differentiate with respect to x"
                QUESTION_DATA.Add("QUESTION TYPE", QUESTION.RETURN_QUESTION_TYPE) ' The typ
e, either differentiation or simplification.
                QUESTION_DATA.Add("ANSWER", QUESTION.RETURN_ANSWER) ' The answer the user s
et.
                QUESTION_DATA.Add("TYPE", QUESTION.TYPE) ' The type, either differentiation
 or simplification.
                QUESTION_DATA.Add("TEACHER EDITED", QUESTION.TEACHER_EDITED.ToString) ' The
 type, either differentiation or simplification.
                QUESTION_DATA.Add("STATUS", QUESTION.STATUS.ToString) ' The type, either di
fferentiation or simplification.
                DATA_LIST.Add(QUESTION_DATA)
            Next

            JsonSerializer.SerializeAsync(STREAM, DATA_LIST) ' This converts the list into
a string that is then written into my text file.
            Debug.WriteLine(JsonSerializer.Serialize(DATA_LIST))
            STREAM.Close()
            Shell("explorer /select," & My.Computer.FileSystem.SpecialDirectories.MyDocumen
ts & "\" & STUDENT_NAME & " " & NAME & " " & TEST_NAME, AppWinStyle.NormalFocus)
            If TEST_NAME = "\QUESTION Export.json" Then
                TOGGLE_CERTAIN_SCREEN(Q_CONTROL_GROUP, True)
            End If
            NOTIFICATIONS_OBJECT.ADD_NOTIFICATION("Successfully exported", Color.Green)
        End If
        Return False
    End Function
```

```
    '////////////////////////////
    ' END
    '////////////////////////////


End Class


' A custom class for notification creation. An optional addition that I made... because I l
ike to waste time.
Class NOTIFICATIONS ' Handles notifications, has a max of 6 notifications at one time.

    ' Possible improvement*
    '//Make the timer reset on each new notification.//

    Private WINDOW_WIDTH
    Private WINDOW_HEIGHT
    Private FORM As Form
    Private NOTIFICATIONS As New Stack(Of Label) ' Will be in the form { Label,Label }, it
is used to hold the notifications.
    Private INTERNAL_REMOVAL As Boolean = False
    Private TIMER As System.Timers.Timer

    ' Window Resize Handling

    ' Delegate for control changing (due to thread differences this is required for invokin
g)
    Private Delegate Sub PROPERTY_CONTROL(Label As Label)
    Private Delegate Sub PROPERTY_CONTROL_LOCATION(Label As Label, POINT As Point)

    Private Sub E_RESIZE() 'Changes the WINDOW_WIDTH & WINDOW_HEIGHT whenever window is res
ized
        Debug.Print(Me.FORM.Height & Me.FORM.Width)
        Me.WINDOW_HEIGHT = FORM1.LOGIN_GROUP.Height ' The group has an offset, allowing the
 placing to be pleasing to the eye.
        Me.WINDOW_WIDTH = FORM1.LOGIN_GROUP.Width
    End Sub

    Sub New(ByRef FORM_INPUT As Form) ' This is sent a reference for the form, so it can ac
cess it.
        Me.FORM = FORM_INPUT ' Sets private variable
        AddHandler Me.FORM.ResizeEnd, AddressOf E_RESIZE 'Adds the resize handler to the E_
RESIZE subroutine.
        Threading.Thread.Sleep(500)
        Me.WINDOW_HEIGHT = 690
        Me.WINDOW_WIDTH = 883 ' Initialises the variables
    End Sub

    ' Delegate methods used for thread communication.
    Private Sub REMOVE(INPUT As Label) ' Removes a control.
        INPUT.Dispose()
    End Sub

    Private Sub LOCATION(INPUT As Label, POINT As Point) ' Used to change controls position
.
        INPUT.Location = POINT
```

```
    End Sub

    ' Notification System

    Private Sub ELAPSED()
        If Me.NOTIFICATIONS.Count = 0 Then ' If there are no more notifications.
            TIMER.Dispose()
            Me.INTERNAL_REMOVAL = False ' Disables the check variable so this sub can be in
itialised again.
            Return
        End If
        Dim REMOVED As Label = Me.NOTIFICATIONS.Pop() ' Gets the popped label so we can dis
pose of it.
        Dim REMOVE_DELEGATE As PROPERTY_CONTROL = New PROPERTY_CONTROL(AddressOf REMOVE) '
Creates a delegate that points to the remove subroutines
        REMOVED.Invoke(REMOVE_DELEGATE, New Object() {REMOVED}) 'Passes the delegate (ie a
pointer to a function) with the paramter of the label. This removes the control within the
controls thread.
        Dim Count As Integer = 0 ' Used to stack the notifications.
        Dim POSITION_DELEGATE As PROPERTY_CONTROL_LOCATION = New PROPERTY_CONTROL_LOCATION(
AddressOf LOCATION) ' Creates a delegate for position changing.
        For Each ILabel As Label In Me.NOTIFICATIONS 'Rearrange them
            Count += 1
            ILabel.Invoke(POSITION_DELEGATE, New Object() {REMOVED, New Point(Me.WINDOW_WID
TH - ILabel.Width, (Me.WINDOW_HEIGHT - ILabel.Height) - ILabel.Height * (Count))}) ' Sends
both argurments with the delegate to run in the correct thread.
        Next
    End Sub

    Private Sub INTERNAL_NOTIFICATION() ' Intermediate subroutine for elapsed subroutines.
        Me.INTERNAL_REMOVAL = True ' Enables the check variable so this cannot be called ag
ain.
        TIMER = New System.Timers.Timer(3000) ' Creates a timer.
        TIMER.Enabled = True
        AddHandler TIMER.Elapsed, AddressOf ELAPSED ' Connects the event of the timer to th
e elapsed subroutines, making it run ever 3 seconds.
    End Sub


    Public Sub ADD_NOTIFICATION(Message As String, Colour As Color)
        If Me.NOTIFICATIONS.Count < 6 Then
            ' Create Label
            Dim Label As New Label
            Debug.Print(FORM1.Controls.Count)
            FORM1.Controls.Add(Label)
            Label.Text = Message
            Label.AutoSize = True
            Label.Font = New System.Drawing.Font("Microsoft Sans Serif", 16.0!, System.Draw
ing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point, CType(0, Byte))
            Label.Refresh()
            Label.Location = New System.Drawing.Point(Me.WINDOW_WIDTH - Label.Size.Width, (
Me.WINDOW_HEIGHT - Label.Size.Height / 2) - Label.Height * (Me.NOTIFICATIONS.Count + 1)) '
Positions it in the corner.
            Label.TabIndex = 4
            Label.Anchor = System.Windows.Forms.AnchorStyles.Right Or System.Windows.Forms.
AnchorStyles.Bottom ' Anchors it so that it moves when resized.
            Label.BringToFront() ' Brings it to the front so it can be seen.
            Label.ForeColor = Colour ' Sets the arguement colour.
```

```
            Me.NOTIFICATIONS.Push(Label)
            If Not Me.INTERNAL_REMOVAL Then 'Starts a thread to automatically remove notifi
cations every 3 seconds.
                Me.INTERNAL_NOTIFICATION()
            End If
        End If
    End Sub

End Class

' Handles the logging in of an account.
Class ACCOUNT

    ' INITIATOR

    Public Sub New()

    End Sub

    ' EVENTS

    ' Page Events
    ' Fire when the program should change page.

    Public Event LOGIN_SCREEN()
    Public Event MODE_SCREEN()

    ' Properties

    ' Account Properties
    Private Property LOGGING_MODE As LOGIN_MODE ' This determines whether the student is lo
gging in as a teacher or a student.
    Private Property MODE As LOGIN_MODE ' This is the property for the accounts actual mode
. This is changed when logging in.

    Private CODE As New Dictionary(Of LOGIN_MODE, String) From {{LOGIN_MODE.STUDENT, "stude
nt"}, {LOGIN_MODE.TEACHER, "teacher"}} ' Used for easy indexing of the login codes.

    ' Methods

    ' Account Methods

    ' Subroutines
    Public Sub SELECT_MODE(INPUT As LOGIN_MODE) ' The input must be in the Enum form. This
selects the mode
        Me.LOGGING_MODE = INPUT
        RaiseEvent LOGIN_SCREEN() ' Shows the login screen.
    End Sub

    ' Functions
    Public Function LOGIN(INPUT As String)
        If Me.CODE.Item(Me.LOGGING_MODE) = INPUT Then 'They entered the correct code
            Me.MODE = Me.LOGGING_MODE 'Set the current mode the program is in.
            Return True
        End If
        Return False
    End Function
```

```
    Public Function ACCOUNT_TYPE()
        Return Me.MODE
    End Function

End Class
```

OPTIMISER.vb

The class that handles the simplification and differentiation. It is the crux of this program, and thus the largest. Most functions are heavily commented and described.

```
Imports System.Text.RegularExpressions

Enum ETYPE
    LEFT
    RIGHT
End Enum


Class OPTIMISER : Inherits UTILITIES

    ' General class for optimising trees.
    ' Will be extremely long and tantamount to the pain I'm willing to endure.


    Private TREE_TO_MODIFY As TREE_NODE
    '' FIX LIKE TERM SUMMING AS IT BREAKS THE POWER ADDITION
    Public Function OPTIMISE_TREE(TO_MODIFY As TREE_NODE, Optional EXPAND_BRACKETS As Boole
an = False)
        ' This function is important in simplifying the tree expressions. The disadvantage
of trees are such that certain multiplicative expressions can be interpreted in multiple wa
ys.
        ' Division and Negative nodes are not optimal for a computer to handle. This functi
on/(collection of functions) aims to solve these issues by rearranging nodes into a form th
at is fundamentally the same.
        ' ... But much easier to interpret for a tree.

        ' Change the negative nodes to positive nodes, where the negative value is now a mu
ltiply node with -1 and the actual value.
        ' Level operators. Operators like + and * can be changed to have many children, ins
tead of the binary two. This is why removing - and simplifying / operators are important.
        Dim LAST_TREE As String

        Dim GLOBAL_LAST_TREE As String

        TREE_TO_MODIFY = TO_MODIFY


        For i = 1 To 8
            GLOBAL_LAST_TREE = IN_ORDER(TREE_TO_MODIFY, True)
            PREPARATION_BY_TIMES_RULING(TREE_TO_MODIFY)
            NEGATIVE_POWERS_TO_DIVISORS(TREE_TO_MODIFY)
            TREE_TO_MODIFY = TO_MODIFY

            While Not LAST_TREE = IN_ORDER(TREE_TO_MODIFY, True)
                LAST_TREE = IN_ORDER(TREE_TO_MODIFY, True)
```

```
                REMOVE_NEGATIVITY(TREE_TO_MODIFY) ' removes the negative roots.
                TREE_TO_MODIFY = TO_MODIFY
                LEVEL_OPERATORS(TREE_TO_MODIFY) ' levels the operators, see function for mo
re details.
                RATIONAL_SIMPLIFICATION(TREE_TO_MODIFY)
            End While

            LAST_TREE = ""

            While Not LAST_TREE = IN_ORDER(TREE_TO_MODIFY, True)
                LAST_TREE = IN_ORDER(TREE_TO_MODIFY, True)
                PREPARATION_BY_TIMES_RULING(TREE_TO_MODIFY)
                FRACTION_COLLECTER_WRAPPER(TREE_TO_MODIFY)
                COLLECT_LIKE_TERMS(TREE_TO_MODIFY)
            End While

            If EXPAND_BRACKETS Then
                MULTIPLIER_SIMPLIFIER(TREE_TO_MODIFY)
            End If

            MULTIPLIER_WRAPPER_SOLVER(TREE_TO_MODIFY)
            PREPERATION_POWER_RULING(TREE_TO_MODIFY)
            POWER_SOLVER(TREE_TO_MODIFY)
            REMOVE_LOOSE_ADDITIONS(TREE_TO_MODIFY)
            REMOVE_POWERS(TREE_TO_MODIFY)
            CLEANUP_FINALISER(TREE_TO_MODIFY)
        Next
        Return IN_ORDER(TREE_TO_MODIFY, True)
    End Function

    Public Function EXPAND_BRACKETS()
        BRACKET_POWER_EXPANDER(TREE_TO_MODIFY)
        OPTIMISE_TREE(TREE_TO_MODIFY, True)
        Return IN_ORDER(TREE_TO_MODIFY, True)
    End Function


    Private OPERATORS = New Dictionary(Of String, Func(Of Double, Double, Double))() From {
 ' An operator dictionary.
    {"+", Function(x, y) x + y},
    {"-", Function(x, y) x - y},
    {"/", Function(x, y) x / y},
    {"*", Function(x, y) x * y},
    {"^", Function(x, y) x ^ y}
}

    Public Function OUTPUT_VALUE_WRAP(VALUE As Integer)
        Return OUTPUT_VALUE(TREE_TO_MODIFY, VALUE)
    End Function
    Public Function OUTPUT_VALUE(NODE As TREE_NODE, VALUE As Integer) As Double
        ' This is a simple algorithm that will set every variable to the input parameter 'V
ALUE'.
        ' The output will then be returned.

        ' This only exists to compare functions easily, as even if two functions are equal,
 their form can be different.
        ' This will be primarially used in the homework submission area.
```

```
        ' The general idea will be to compute the lowest left node, then return this value,
 and go up to the root, and through the right.
        ' Ie I calculate the left dependent on the root, then I 'go to the root' and calcul
ate the right node with the computed left node as the starting point.
        ' Normally this would be a binary tree, so I would actually 'go to the root' by cal
culating the left and right, but as trees can have more than one in the left and right, I m
ust first calculate their stuff.

        ' The values of each node will be calulated much the same, so by virtue of this alg
orithm I go from the bottom to the top.
        Dim COMPUTED As Double
        If NODE.VALUE = "*" Or NODE.VALUE = "/" Or NODE.VALUE = "^" Then
            COMPUTED = 1 'This will be the starting integer I will go off.
        ElseIf NODE.VALUE = "+" Or NODE.VALUE = "-" Then
            COMPUTED = 0 ' Obviously + nodes will start from 0.
        ElseIf IsNumeric(NODE.VALUE) Then
            COMPUTED = NODE.VALUE
        Else
            ' This is probably a variable, like an x, so I return 1. I don't need to waste
time as it won't have anything as its children.
            Return VALUE
        End If

        If Not NODE.LEFT Is Nothing Then
            For Each NODE_ELEMENT As TREE_NODE In NODE.LEFT
                Dim OPERATION As Func(Of Double, Double, Double) = OPERATORS(NODE.VALUE)
                COMPUTED = OPERATION(COMPUTED, OUTPUT_VALUE(NODE_ELEMENT, VALUE)) ' This wi
ll calculate the resultant.
            Next
        End If

        If Not NODE.RIGHT Is Nothing Then
            For Each NODE_ELEMENT As TREE_NODE In NODE.RIGHT
                Dim OPERATION As Func(Of Double, Double, Double) = OPERATORS(NODE.VALUE) '
The same deal for the right.
                COMPUTED = OPERATION(COMPUTED, OUTPUT_VALUE(NODE_ELEMENT, VALUE))
            Next
        End If

        Return COMPUTED
    End Function

    Public Sub BRACKET_POWER_EXPANDER(NODE As TREE_NODE)
        'u^3=u*u*u

        If Not NODE.LEFT Is Nothing Then
            For Each NODE_ELEMENT As TREE_NODE In NODE.LEFT
                BRACKET_POWER_EXPANDER(NODE_ELEMENT)
            Next
        End If
        If Not NODE.RIGHT Is Nothing Then
            For Each NODE_ELEMENT As TREE_NODE In NODE.RIGHT
                BRACKET_POWER_EXPANDER(NODE_ELEMENT)
            Next
        End If

        If NODE.VALUE = "^" Then
            If IsNumeric(NODE.RIGHT(0).VALUE) Then
```

```
                NODE.VALUE = "*"
                For I As Integer = 1 To NODE.RIGHT(0).VALUE - 1
                    If NODE.RIGHT.Count = 1 Then
                        NODE.RIGHT.Add(NODE.LEFT(0).CLONE)
                    Else
                        NODE.LEFT.Add(NODE.LEFT(0).CLONE)
                    End If
                Next
                NODE.RIGHT.RemoveAt(0)
            End If
        End If


    End Sub

    Public Sub REMOVE_DIVISION_ENTIRELY(NODE As TREE_NODE)
        ' THIS ASSUMES NUMERIC POWERS!!
        If Not NODE.LEFT Is Nothing Then
            For Each NODE_ELEMENT As TREE_NODE In NODE.LEFT
                REMOVE_DIVISION_ENTIRELY(NODE_ELEMENT)
            Next
        End If
        If Not NODE.RIGHT Is Nothing Then
            For Each NODE_ELEMENT As TREE_NODE In NODE.RIGHT
                REMOVE_DIVISION_ENTIRELY(NODE_ELEMENT)
            Next
        End If

        If NODE.VALUE = "/" Then
            Dim POWER_NODE, MINUS_NODE As New TREE_NODE
            MINUS_NODE.VALUE = "-1"
            If NODE.RIGHT(0).VALUE <> "^" Then
                POWER_NODE.VALUE = "^"
                POWER_NODE.RIGHT.Add(MINUS_NODE) ' So the power is ^-
1, the same as division
                POWER_NODE.LEFT.AddRange(NODE.RIGHT) ' The actual divisor
                NODE.RIGHT.RemoveRange(0, NODE.RIGHT.Count)
                NODE.RIGHT.Add(POWER_NODE)
            Else
                NODE.RIGHT(0).RIGHT(0).VALUE = "-" & NODE.RIGHT(0).RIGHT(0).VALUE
            End If
            NODE.VALUE = "*"
        End If


    End Sub

    Public Function DIFFERENTIATE(Optional TO_MODIFY As TREE_NODE = Nothing)
        If TO_MODIFY IsNot Nothing Then
            TREE_TO_MODIFY = TO_MODIFY
        End If

        ' This is only here as when there is no + node at the top, things get iffy for some
 algorithms designed around them.
        ' I could fix it but my attempts just make things annoying.
        If TREE_TO_MODIFY.VALUE <> "+" Then
            Dim ADD_NODE, ZERO_NODE As New TREE_NODE
            ADD_NODE.VALUE = "+"
```

```
            ZERO_NODE.VALUE = "0"
            ADD_NODE.LEFT.Add(TREE_TO_MODIFY)
            ADD_NODE.RIGHT.Add(ZERO_NODE)
            TREE_TO_MODIFY = ADD_NODE
        End If
        REMOVE_DIVISION_ENTIRELY(TREE_TO_MODIFY)
        PREPARATION_BY_TIMES_RULING(TREE_TO_MODIFY)
        PREPERATION_POWER_RULING(TREE_TO_MODIFY)
        MULTIPLIER_WRAPPER_SOLVER(TREE_TO_MODIFY)
        'Console.WriteLine("to be " & IN_ORDER(TREE_TO_MODIFY, True))
        DIFFERENTATION_WRAPPER(TREE_TO_MODIFY)
        ' Simplify the differentiated
        ' NEGATIVE_POWERS_TO_DIVISORS(TREE_TO_MODIFY)
        OPTIMISE_TREE(TREE_TO_MODIFY)
        'NEGATIVE_POWERS_TO_DIVISORS(TREE_TO_MODIFY)
        Return IN_ORDER(TREE_TO_MODIFY, True)
    End Function


    ' Simplification Functions
    ' //These are made to actually modify the expression.

    Enum MULTIPLIER_NUM ' Used by 'MULTIPLIER_SIMPLIFIER'
        SUM
        OTHER
    End Enum

    Private Function LIMITED_DIFFERENTIATE(NODE As TREE_NODE)
        ' This is a recursive algorithm that differentiates specific input functions.
        ' Differentiation is inherently recursive, or more so it is a downward loop of refi
ning what the derivative actually equals.
        ' Whatever input function I have, I can change it so that it is in proper form.
        ' If it is d(f(x)*f(y)), then the resultant after partial derivatives is d(f(x))*f(
y) + f(x)*d(f(y))... or (d(f(x))/dx) *dx + f(x) * d(f(y))/dx * dy
        ' This can only work when I let all functions into some form u, and the basic princ
iples of differentiating powers, or multipliers, follows.

        ' Ie, f(x) and f(y) become f(u) and f(t), where u and t are made to put it in the f
orm (u^a)*(t^b).

        ' General cases to look for.
        ' When form u^a is found, do the total derivative of a modified function -
        ' EXAMPLE: f(x) = (9x^2+3)^4, let u = 9x^2+3, so f(u)= u^4, taking total derivative
 of apparent input -> d(f(u)) = d(f(u))/du * du.
        ' Recursively call the algorithm for "du", so that du is now evaluated, and made to
 be du/dx, because u is in terms of x and can be further decomposed.

        ' When form (u^a)*(t^b) is found, do product rule derived from partial derivatives,
        ' EXAMPLE: f(x) = ((9x^2+3)^4)*((39x^5+2x)^93), let u = 9x^2+3 and t = 39x^5+2x.  N
ew form f(u,t) = (u^4)*(t^93). So 'multivariable' function.
        ' d(f(u,t)) = sum of partial derivatives = ∂(f(u,t))/∂u *du + ∂(f(u,t))/∂t *dt= d(u
^4)*(t^93) + (u^4)*d(t^93), -
> For d(u^4), the same principle for recursive differentiation applies,
        ' ie d(u^4) -> d(u^4)/du * du. Call function again for u and differentiate.


        ' This will be an up to down traversal, as this is how differentiating works.
```

```
        ' The traversal will only continue when the derivative can be further deconstructed
. In the form of u!

        If NODE.VALUE = "*" Then
            ' I will make a general rule for products, even if they are merely 9x^2.
            ' So, it will become a sum of partial derivatives.
            ' If the product has 4 varriables, then there will be 4 sums.
            ' This will be correct as all product derivatives are in this form, but most of
 the time the partial derivatives end up turning to nothing.
            ' General form, say y= f(u,t,c,...) -
> d(f(u,t,c,...)) = ∂y/∂u *du + ∂y/∂t * dt + ∂y/∂c * dc ... to infinity. Wonderful elegance
 in a derivative :).


            Dim COLLECTIVE = {NODE.LEFT, NODE.RIGHT}
            Dim PLUS_NODE As New TREE_NODE
            PLUS_NODE.VALUE = "+"


            For Each ELEMENT In COLLECTIVE
                For A = 0 To ELEMENT.Count - 1

                    Dim SELECTED_NODE As TREE_NODE = ELEMENT(A)
                    ' Console.WriteLine("the element" & IN_ORDER(ELEMENT(A), True))
                    Dim REFERENCE_NODE As New TREE_NODE
                    REFERENCE_NODE.VALUE = "*"
                    REFERENCE_NODE.LEFT.AddRange(NODE.LEFT)
                    REFERENCE_NODE.RIGHT.AddRange(NODE.RIGHT)

                    If REFERENCE_NODE.LEFT.Contains(ELEMENT(A)) Then
                        REFERENCE_NODE.LEFT.RemoveAt(A)
                    Else
                        REFERENCE_NODE.RIGHT.RemoveAt(A)
                    End If

                    ' This is for the partial part. Ie d(x*y*z) = d(x)*y*z + x*d(y)*z... et
c. So I must remove the selected node, and then add the reference on later.

                    If SELECTED_NODE.VALUE = "^" Then
                        If Not IsNumeric(IN_ORDER(SELECTED_NODE.LEFT(0), True)) Then
                            ' Remember that I assume powers are numeric. I'll deal with log
s when I feel succinctly mad.
                            Dim MULTIPLIER As New TREE_NODE
                            MULTIPLIER.VALUE = "*" ' The decomposed derivative
                            Dim NUMERIC_NODE As New TREE_NODE ' the power
                            NUMERIC_NODE.VALUE = SELECTED_NODE.RIGHT(0).VALUE

                            Dim CLONED_SELECTED_NODE As TREE_NODE = SELECTED_NODE.CLONE
                            CLONED_SELECTED_NODE.RIGHT(0).VALUE = CLONED_SELECTED_NODE.RIGH
T(0).VALUE - 1 'd(u^3) = 3*u^2 du


                            MULTIPLIER.LEFT.Add(NUMERIC_NODE) ' The power, ie the 3 in the
3*u^2 du
                            MULTIPLIER.RIGHT.Add(CLONED_SELECTED_NODE) ' The remaining func
tion, ie the u^2

                            MULTIPLIER.RIGHT.AddRange(REFERENCE_NODE.LEFT)
```

```
                            MULTIPLIER.RIGHT.AddRange(REFERENCE_NODE.RIGHT)
                            Dim NEW_TIMES_NODE, ONE_NODE As New TREE_NODE ' As this algorit
hm only accepts (x*b), then I will make this *1. Either that or a load of redundancy.
                            ONE_NODE.VALUE = "1"
                            NEW_TIMES_NODE.VALUE = "*"
                            NEW_TIMES_NODE.LEFT.Add(SELECTED_NODE.LEFT(0))
                            NEW_TIMES_NODE.RIGHT.Add(ONE_NODE)

                            If LIMITED_DIFFERENTIATE(NEW_TIMES_NODE) IsNot Nothing Then
                                If (LIMITED_DIFFERENTIATE(NEW_TIMES_NODE).LEFT.COUNT + LIMI
TED_DIFFERENTIATE(NEW_TIMES_NODE).RIGHT.COUNT) > 0 Then
                                    MULTIPLIER.RIGHT.Add(LIMITED_DIFFERENTIATE(NEW_TIMES_NO
DE)) ' the du. or more so du/dx as everything is divided by dx.
                                End If
                            End If
                            If PLUS_NODE.LEFT.Count = 0 Then
                                PLUS_NODE.LEFT.Add(MULTIPLIER) ' Add the subsequent derivat
ive to the plus node
                            Else
                                PLUS_NODE.RIGHT.Add(MULTIPLIER) ' Add the subsequent deriva
tive to the plus node
                            End If
                        Else
                            ' This means its an integer ^ of something. so 0.
                            Dim ZERO_NODE As New TREE_NODE
                            ZERO_NODE.VALUE = "0"
                            PLUS_NODE.LEFT.Add(ZERO_NODE)
                        End If

                    ElseIf SELECTED_NODE.VALUE = "+" Then
                        ' Easier to implement.
                        ' This will just look through each item in the node and differentia
te it.
                        Dim SUB_COLLECTIVE = {SELECTED_NODE.LEFT, SELECTED_NODE.RIGHT}
                        Dim NEW_PLUS_NODE As New TREE_NODE
                        NEW_PLUS_NODE.VALUE = "+"
                        For Each SUB_ELEMENT In SUB_COLLECTIVE
                            For B = 0 To SUB_ELEMENT.Count - 1
                                Dim DIFFERENTIATED_NODE As TREE_NODE = LIMITED_DIFFERENTIAT
E(SUB_ELEMENT(B))
                                If NEW_PLUS_NODE.LEFT.Count = 0 Then
                                    NEW_PLUS_NODE.LEFT.Add(DIFFERENTIATED_NODE) ' Ie (x+y+z
) = (dx+dy+dz)
                                Else
                                    NEW_PLUS_NODE.RIGHT.Add(DIFFERENTIATED_NODE)
                                End If
                            Next
                        Next
                        ' I now have a differentiated + node. But in context this must be p
roperly formed.
                        Dim NEW_NODE As New TREE_NODE
                        NEW_NODE.VALUE = "*"
                        NEW_NODE.LEFT.Add(NEW_PLUS_NODE)
                        NEW_NODE.RIGHT.AddRange(REFERENCE_NODE.LEFT)
                        NEW_NODE.RIGHT.AddRange(REFERENCE_NODE.RIGHT)
                        PLUS_NODE.LEFT.Add(NEW_NODE)
                    Else
```

```
                        Dim CHECK = MATCH_COLLECTION_TO_DICTIONARY(Regex.Matches(SELECTED_N
ODE.VALUE, "(?=[a-z])[^x]")) ' This looks for variables other than x.
                        ' The point of this is to use the correct notiation when I am diffe
rentiating variables other than x.
                        If CHECK.Count > 0 Then
                            ' This means that the input variable is not a x.
                            ' If it was y, then this would output dy/dx.
                            Dim TOP, BOTTOM As New TREE_NODE
                            Dim CLONED_SELECTED_NODE As TREE_NODE = SELECTED_NODE.CLONE
                            TOP.VALUE = "d" & SELECTED_NODE.VALUE ' This would be the dy
                            BOTTOM.VALUE = "dx" ' The divider, dx
                            CLONED_SELECTED_NODE.VALUE = "/"
                            CLONED_SELECTED_NODE.LEFT.Add(TOP)
                            CLONED_SELECTED_NODE.RIGHT.Add(BOTTOM)
                            Return CLONED_SELECTED_NODE ' Returns this node.
                        End If
                    End If
                Next
            Next
            Return PLUS_NODE

        Else
            Dim ZERO_NODE As New TREE_NODE
            ZERO_NODE.VALUE = "0"
            Return ZERO_NODE

        End If
        Return NODE
    End Function

    Private Sub DIFFERENTATION_WRAPPER(NODE As TREE_NODE)

        ' Where u^(A), and u is a variable of some kind (differentiable, ie if u = (2x^2+3)
 then implement chain rule) and A is a numeric power.
        ' Numeric powers are a prerequisite due to logarithms being required for any other
form.
        ' This will be implemented elsewhere and added on in another prototype.

        ' The form is of a "^", and the left node either being a * or a + node, and the rig
ht node being a number.
        ' d(u^a) -> a*u^(a-
1)*du (normally derivatives are in the form d(f(x))/dx, but it is fine to revoke the ratio
definer in this case for generalisation. In a more formal definition, df(u)=(d(f(u))/du) *
du (typically u = some x, so you write du = (du/dx)*dx))
        ' This form is constant, so I will differentiate the +, or * node in any fashion.

        ' Where this must not work:
        ' Differentiation of this form MUST only occur on the foremost + node.
        ' This will not be recursive. Ie, I will not look for nodes to differentiate beyond
 the root node.
        ' Ie, I can only differentiate "u+a", where u and a are of a common variable x (mul
tivariable form is not an aim right now). I cannot differentiate "u+a" in "u^(u+a)", where
I differentiate the bracket of the power, as this is wrong..


        If NODE.VALUE = "+" Then

            Dim COLLECTIVE = {NODE.LEFT, NODE.RIGHT}
```

```
            For Each ELEMENT In COLLECTIVE
                Dim CLONED_LOOP As New List(Of TREE_NODE)
                CLONED_LOOP.AddRange(ELEMENT)
                'Console.WriteLine("THE COUNTTTT" & CLONED_LOOP.Count)
                For A = 0 To ELEMENT.Count - 1

                    Dim SELECTED_NODE As TREE_NODE = CLONED_LOOP(A)
                    'Console.WriteLine("THIS THINGGG" & IN_ORDER(CLONED_LOOP(A), True))
                    Dim DIFFERENTIABLE = LIMITED_DIFFERENTIATE(SELECTED_NODE)
                    'Console.WriteLine("THIS THINGGzzzzG" & IN_ORDER(DIFFERENTIABLE, True))
                    ELEMENT.RemoveAt(A)
                    If DIFFERENTIABLE IsNot Nothing Then
                        ELEMENT.Insert(A, DIFFERENTIABLE)
                    End If
                Next
            Next
        Else
            Dim DIFFERENTIABLE As TREE_NODE = LIMITED_DIFFERENTIATE(NODE)
            'Console.WriteLine("lol" & IN_ORDER(DIFFERENTIABLE, True) & NODE.VALUE)
            NODE.LEFT.RemoveRange(0, NODE.LEFT.Count)
            NODE.RIGHT.RemoveRange(0, NODE.RIGHT.Count)
            NODE.VALUE = DIFFERENTIABLE.VALUE
            NODE.LEFT.AddRange(DIFFERENTIABLE.LEFT)
            NODE.RIGHT.AddRange(DIFFERENTIABLE.RIGHT)
        End If

    End Sub


    Private Sub MULTIPLIER_SIMPLIFIER_CALCULATOR(TYPE As MULTIPLIER_NUM, ROOT_NODE As TREE_
NODE, LEFT_MULTIPLIER_NODE As TREE_NODE, RIGHT_NODE As List(Of TREE_NODE))
        ' Two Cases

        ' This is setup in the way such that one single element node is sent (like 5x, in t
he form of a * or / node) and the rest of the RIGHT_NODE is sent.
        ' The RIGHT_NODE is a list.

        ' In the case it isn't a + node, then it will just be, say, 5x * (a*b*c*...)
        ' Remember that it is assumed that the LEFT_MULTIPLIER_NODE comes from a + node fro
m the left.
        ' In this instance it is very easy, and I will just move the resulting node (5*x*a*
b*c*...) out of the + node and into the main LEFT NODE of the ROOT_NODE.
        ' This is because I assume the ROOT_NODE will turn into a + node by the end of it.

        ' In the case the RIGHT_NODE is a + NODE (ie it has 1 child and its a + NODE)
        ' I will loop through the RIGHT_NODE(0) (the + node) and for each item I will form
a * node of them combined.
        ' I will then add this new node to the LEFT NODE/RIGHT NODE
        ' This continues until all nodes are done.

        Select Case TYPE
            Case MULTIPLIER_NUM.SUM ' This means its A*(B+C+D...)
                Dim ITERATION_ARRAY = {RIGHT_NODE(0).LEFT, RIGHT_NODE(0).RIGHT}
                For Each ELEMENT As List(Of TREE_NODE) In ITERATION_ARRAY
                    For Each NODE As TREE_NODE In ELEMENT
                        Dim NEW_NODE As New TREE_NODE
                        NEW_NODE.VALUE = "*"
```

```
                        NEW_NODE.LEFT.Add(LEFT_MULTIPLIER_NODE)
                        NEW_NODE.RIGHT.Add(NODE)
                        ROOT_NODE.LEFT.Add(NEW_NODE) ' Adds the new node into the LEFT chil
d of the root.
                    Next
                Next
            Case MULTIPLIER_NUM.OTHER ' It is A*B
                Dim NEW_NODE As New TREE_NODE
                NEW_NODE.VALUE = "*"
                NEW_NODE.LEFT.Add(LEFT_MULTIPLIER_NODE)
                NEW_NODE.RIGHT.AddRange(RIGHT_NODE)
                ROOT_NODE.LEFT.Add(NEW_NODE)
        End Select

    End Sub

    'Private Sub

    Private Sub COLLECT_LIKE_TERMS_WRAPPER(NODE As TREE_NODE)
        LEVEL_OPERATORS(NODE)
        PREPARATION_BY_TIMES_RULING(TREE_TO_MODIFY)
        COLLECT_LIKE_TERMS(NODE)
    End Sub


    Private Sub MULTIPLIER_WRAPPER_SOLVER(NODE As TREE_NODE)
        LEVEL_OPERATORS(NODE)
        MULTIPLIER_SOLVER(NODE)
    End Sub

    ' The 'Give Up' Subroutines
    ' Madness congealed
    ' Not made for elegance or efficiency TM.


    Private Sub REMOVE_POWERS(NODE As TREE_NODE)
        If Not NODE.LEFT Is Nothing Then
            For Each NODE_ELEMENT As TREE_NODE In NODE.LEFT
                REMOVE_POWERS(NODE_ELEMENT)
            Next
        End If
        If Not NODE.RIGHT Is Nothing Then
            For Each NODE_ELEMENT As TREE_NODE In NODE.RIGHT
                REMOVE_POWERS(NODE_ELEMENT)
            Next
        End If


        If NODE.VALUE = "^" Then
            If NODE.RIGHT(0).VALUE = "1" Then
                NODE.VALUE = NODE.LEFT(0).VALUE
                If Not IsNumeric(NODE.VALUE) Then
                    Dim TOTAL_NODE As New List(Of TREE_NODE)
                    TOTAL_NODE.AddRange(NODE.LEFT(0).LEFT)
                    TOTAL_NODE.AddRange(NODE.LEFT(0).RIGHT)
                    NODE.LEFT.RemoveRange(0, NODE.LEFT.Count)
                    NODE.RIGHT.RemoveRange(0, NODE.RIGHT.Count)
                    ALTERNATING_TREE_INSERTING(TOTAL_NODE, NODE)
```

```
                End If
            ElseIf NODE.RIGHT(0).VALUE = "0" Then
                NODE.VALUE = "1"
                NODE.LEFT.RemoveRange(0, NODE.LEFT.Count)
                NODE.RIGHT.RemoveRange(0, NODE.RIGHT.Count)
            End If
        End If

    End Sub

    Private Sub REMOVE_ZERO_MULTIPLIERS(NODE As TREE_NODE)
        If Not NODE.LEFT Is Nothing Then
            For Each NODE_ELEMENT As TREE_NODE In NODE.LEFT
                REMOVE_LOOSE_ADDITIONS(NODE_ELEMENT)
            Next
        End If
        If Not NODE.RIGHT Is Nothing Then
            For Each NODE_ELEMENT As TREE_NODE In NODE.RIGHT
                REMOVE_LOOSE_ADDITIONS(NODE_ELEMENT)
            Next
        End If
        Dim ITERABLE = {NODE.LEFT, NODE.RIGHT}


        If NODE.VALUE = "+" Then
            For Each ELEMENT As List(Of TREE_NODE) In ITERABLE
                For A = 0 To ELEMENT.Count - 1
                    If ELEMENT(A).VALUE = "0" Then

                    End If
                Next
            Next
        End If
    End Sub

    Private Sub REMOVE_LOOSE_ADDITIONS(NODE As TREE_NODE)
        If Not NODE.LEFT Is Nothing Then
            For Each NODE_ELEMENT As TREE_NODE In NODE.LEFT
                REMOVE_LOOSE_ADDITIONS(NODE_ELEMENT)
            Next
        End If
        If Not NODE.RIGHT Is Nothing Then
            For Each NODE_ELEMENT As TREE_NODE In NODE.RIGHT
                REMOVE_LOOSE_ADDITIONS(NODE_ELEMENT)
            Next
        End If

        If NODE.VALUE = "+" Then
            If (NODE.LEFT.Count + NODE.RIGHT.Count) <= 1 Then
                Dim CHOSEN_NODE As TREE_NODE
                If NODE.LEFT.Count = 1 Then
                    CHOSEN_NODE = NODE.LEFT(0)
                ElseIf NODE.RIGHT.Count = 1 Then
                    CHOSEN_NODE = NODE.RIGHT(0)
                Else
                    Return
                End If
                NODE.VALUE = CHOSEN_NODE.VALUE
```

```
                If IsNumeric(CHOSEN_NODE.VALUE) Then
                    NODE.LEFT.RemoveRange(0, NODE.LEFT.Count)
                    NODE.RIGHT.RemoveRange(0, NODE.RIGHT.Count)
                Else
                    Dim NEW_READDING_LIST As New List(Of TREE_NODE)
                    NEW_READDING_LIST.AddRange(CHOSEN_NODE.LEFT)
                    NEW_READDING_LIST.AddRange(CHOSEN_NODE.RIGHT)
                    NODE.LEFT.RemoveRange(0, NODE.LEFT.Count)
                    NODE.RIGHT.RemoveRange(0, NODE.RIGHT.Count)
                    ALTERNATING_TREE_INSERTING(NEW_READDING_LIST, NODE)
                End If
            End If
        End If
    End Sub

    Private Sub CLEANUP_FINALISER(NODE As TREE_NODE)

        ' Let A*1 = A

        If Not NODE.LEFT Is Nothing Then
            For Each NODE_ELEMENT As TREE_NODE In NODE.LEFT
                CLEANUP_FINALISER(NODE_ELEMENT)
            Next
        End If
        If Not NODE.RIGHT Is Nothing Then
            For Each NODE_ELEMENT As TREE_NODE In NODE.RIGHT
                CLEANUP_FINALISER(NODE_ELEMENT)
            Next
        End If


        If NODE.VALUE = "+" Then
            Dim INTERATION_LIST = {NODE.LEFT, NODE.RIGHT}
            For Each ELEMENT_LIST As List(Of TREE_NODE) In INTERATION_LIST
                Dim BUFFER As Integer = 0
                For A = 0 To ELEMENT_LIST.Count - 1
                    If (A - BUFFER) <= ELEMENT_LIST.Count - 1 Then
                        Dim TRUE_POINTER As Integer = A - BUFFER
                        If ELEMENT_LIST(TRUE_POINTER).VALUE = "*" Then ' If its a * node, s
o something like 0*x
                            Dim TOTAL_LIST As New List(Of TREE_NODE)
                            TOTAL_LIST.AddRange(ELEMENT_LIST(TRUE_POINTER).RIGHT)
                            TOTAL_LIST.AddRange(ELEMENT_LIST(TRUE_POINTER).LEFT)
                            Dim ZEROS = From INT In TOTAL_LIST Where INT.VALUE = "0" ' Coll
ects all the 0's in the node, if there are any.
                            If ZEROS.Count > 0 Then
                                ELEMENT_LIST.RemoveAt(TRUE_POINTER)
                                If NODE.LEFT.Count + NODE.RIGHT.Count = 0 Then
                                    NODE.VALUE = ""
                                End If
                                BUFFER += 1
                            End If
                        ElseIf ELEMENT_LIST(TRUE_POINTER).VALUE = "0" Then ' Just a 0
                            ELEMENT_LIST.RemoveAt(TRUE_POINTER)
                            BUFFER += 1
                        End If
                    End If
                Next
```

```
            Next
        End If

        If NODE.VALUE = "*" Then
            If NODE.RIGHT.Count = 1 Then
                If NODE.RIGHT(0).VALUE = "1" Then
                    NODE.RIGHT.RemoveAt(0)
                    If Not NODE.LEFT.Count = 1 Then
                        Dim NEW_READDING_LIST As New List(Of TREE_NODE)
                        NEW_READDING_LIST.AddRange(NODE.LEFT)
                        NODE.LEFT.RemoveRange(0, NODE.LEFT.Count)
                        ALTERNATING_TREE_INSERTING(NEW_READDING_LIST, NODE) 'Adds it via al
ternate insertion (even number of items on left and right) so it displays correctly.
                    ElseIf NODE.LEFT.Count = 1 And IsNumeric(NODE.LEFT(0).VALUE) Then
                        NODE.VALUE = NODE.LEFT(0).VALUE
                        NODE.LEFT.RemoveAt(0)
                    Else
                        NODE.VALUE = NODE.LEFT(0).VALUE
                        Dim NEW_READDING_LIST As New List(Of TREE_NODE)
                        NEW_READDING_LIST.AddRange(NODE.LEFT(0).LEFT)
                        NEW_READDING_LIST.AddRange(NODE.LEFT(0).RIGHT)
                        NODE.LEFT.RemoveRange(0, NODE.LEFT.Count)
                        ALTERNATING_TREE_INSERTING(NEW_READDING_LIST, NODE)
                    End If
                End If
            End If
        End If

    End Sub

    Private Sub POWER_SOLVER(NODE As TREE_NODE)
        ' Let x^n*x^b = x^(n+b)

        If Not NODE.LEFT Is Nothing Then
            For Each NODE_ELEMENT As TREE_NODE In NODE.LEFT
                POWER_SOLVER(NODE_ELEMENT)
            Next
        End If
        If Not NODE.RIGHT Is Nothing Then
            For Each NODE_ELEMENT As TREE_NODE In NODE.RIGHT
                POWER_SOLVER(NODE_ELEMENT)
            Next
        End If

        Dim NODE_LIST As New List(Of TREE_NODE)

        If NODE.VALUE = "*" Then
            Dim ITERABLE = {NODE.LEFT, NODE.RIGHT}
            For Each ELEMENT As List(Of TREE_NODE) In ITERABLE
                Dim DOWN_BREAK As Integer = 0
                For A As Integer = 0 To ELEMENT.Count - 1
                    Dim LESSENED_INDEX As Integer = A - DOWN_BREAK
                    If (LESSENED_INDEX) <= ELEMENT.Count - 1 Then ' Makes sure that it is w
ithin bounds ;)
                        If ELEMENT(LESSENED_INDEX).VALUE = "^" Then
                            NODE_LIST.Add(ELEMENT(LESSENED_INDEX))
                            ELEMENT.RemoveAt(LESSENED_INDEX)
```

```
                            DOWN_BREAK += 1 ' Shifts the loop the left by 1, so that we ain
't losing any looping buddies :0
                        End If
                    End If
                Next
            Next
            ' "^" nodes are binary, so we can only have 0 indexes for left and right.
            Dim GROUPED = NODE_LIST.GroupBy(Function(x) IN_ORDER(x.LEFT(0), True)) ' Off di
d the computer go, for this program was trying to turn it into a toaster. (It groups each b
y the same variables, ie {{x},{x^3,x^2,x^1}})
            For Each GROUP In GROUPED ' We gonna loop through it all baby
                Dim MAIN_VARIABLE As TREE_NODE = GROUP.ElementAt(0).LEFT(0)
                Dim MAIN_POWER As New TREE_NODE
                MAIN_POWER.VALUE = "+" ' This is weird but I will just solve for this + nod
e like I do for an input :)

                Dim ALTERNATING_EVEN_NUMBER As Integer = 1
                Dim CHECK As New Dictionary(Of String, String)
                For Each ITEM As TREE_NODE In GROUP
                    CHECK = MATCH_COLLECTION_TO_DICTIONARY(Regex.Matches(ITEM.VALUE, "[*,+,
/,-,^]"))
                    If CHECK.Count = 0 Then
                        MAIN_POWER.LEFT.Add(ITEM.RIGHT(0))
                    Else
                        MAIN_POWER.RIGHT.Add(ITEM.RIGHT(0))
                    End If
                    ALTERNATING_EVEN_NUMBER += 1
                Next
                'LEVEL_OPERATORS(MAIN_POWER)

                ' Dim SIMPLIFIED_TREE_NODE As TREE_NODE = OPTIMISE_TREE(MAIN_POWER.CLONE) '
 Optimise the tree. Technically recursively. Gosh.
                ' Console.WriteLine("IDKFv2" & IN_ORDER(SIMPLIFIED_TREE_NODE, True))
                Dim FINAL_NODE As New TREE_NODE
                FINAL_NODE.VALUE = "^"
                FINAL_NODE.LEFT.Insert(0, MAIN_VARIABLE)
                FINAL_NODE.RIGHT.Insert(0, MAIN_POWER)
                CHECK = MATCH_COLLECTION_TO_DICTIONARY(Regex.Matches(MAIN_VARIABLE.VALUE, "
[*,+,/,-,^]"))
                If CHECK.Count = 0 Then
                    NODE.LEFT.Add(FINAL_NODE)
                Else
                    NODE.RIGHT.Add(FINAL_NODE)
                End If
            Next




        End If
    End Sub
    Private Sub MULTIPLIER_SOLVER(NODE As TREE_NODE)
        ' Let A*B = C, when A and B are integers.

        If Not NODE.LEFT Is Nothing Then
            For Each NODE_ELEMENT As TREE_NODE In NODE.LEFT
                MULTIPLIER_SOLVER(NODE_ELEMENT)
            Next
```

```
        End If
        If Not NODE.RIGHT Is Nothing Then
            For Each NODE_ELEMENT As TREE_NODE In NODE.RIGHT
                MULTIPLIER_SOLVER(NODE_ELEMENT)
            Next
        End If
        If NODE.VALUE = "*" Then
            Dim TOTAL_LIST As New List(Of TREE_NODE)
            TOTAL_LIST.AddRange(NODE.RIGHT)
            TOTAL_LIST.AddRange(NODE.LEFT)
            Dim INTEGERS = From INT In TOTAL_LIST Where IsNumeric(INT.VALUE)
            Dim FINAL_INT As Integer = 1
            For A = 0 To INTEGERS.Count - 1
                FINAL_INT *= INTEGERS(0).VALUE
                TOTAL_LIST.Remove(INTEGERS(0))
            Next
            Dim NEW_INT As New TREE_NODE
            If FINAL_INT <> "1" Or TOTAL_LIST.Count = 0 Then
                NEW_INT.VALUE = FINAL_INT
                TOTAL_LIST.Insert(0, NEW_INT)
            End If
            NODE.LEFT = New List(Of TREE_NODE)
            NODE.RIGHT = New List(Of TREE_NODE)
            Dim CHECK As Dictionary(Of String, String) = MATCH_COLLECTION_TO_DICTIONARY(Reg
ex.Matches(TOTAL_LIST(0).VALUE, "[*,+,/,-,^]"))
            If TOTAL_LIST.Count = 1 And ((IsNumeric(TOTAL_LIST(0).VALUE) Or CHECK.Count = 0
)) Then
                NODE.VALUE = TOTAL_LIST(0).VALUE
            ElseIf TOTAL_LIST.Count >= 1 Then
                For Each ITEM_IN_LIST As TREE_NODE In TOTAL_LIST
                    CHECK = MATCH_COLLECTION_TO_DICTIONARY(Regex.Matches(ITEM_IN_LIST.VALUE
, "[*,+,/,-,^]"))
                    If CHECK.Count = 0 Then ' Its a number or variable.
                        NODE.LEFT.Add(ITEM_IN_LIST)
                    Else
                        NODE.RIGHT.Add(ITEM_IN_LIST)
                    End If
                Next
            End If
        End If
    End Sub

    ' End of the 'Give Up' Subroutines

    Private Function MULTIPLIER_SIMPLIFIER(NODE As TREE_NODE)


        Dim NODE_LIST As New List(Of TREE_NODE) 'Used so that I can easily compare.
        Dim MODE As MULTIPLIER_NUM

        ' The objective of this is to remove bracket multiplication.
        ' Ie, when either a bracket is multiplied by a single value, or there are 2 bracket
s.
        ' When such a scenarior exists, it will loop through the discovered bracket.

        ' On this instance there are two occurences -
```

```
        ' <A> the other node is a * or a / node. This means I can merely create a multiplic
ation node for each element being multiplied.
        ' Ie A*(B+C) = A*B + B*C. I will transform the multiplication node into a + node to
 house the subsequent calculations.

        ' <B> the other node is a + node.
        ' This means I must loop through each individual node within the other.
        ' Ie (A+B)(C+D) = (A*C + A*D) + (B*C + B*D).


        If Not NODE.LEFT Is Nothing Then
            For Each NODE_ELEMENT As TREE_NODE In NODE.LEFT
                MULTIPLIER_SIMPLIFIER(NODE_ELEMENT)
            Next
        End If
        If Not NODE.RIGHT Is Nothing Then
            For Each NODE_ELEMENT As TREE_NODE In NODE.RIGHT
                MULTIPLIER_SIMPLIFIER(NODE_ELEMENT)
            Next
        End If
   If NODE.VALUE = "*" Then
            Dim ITERATION_LIST = {NODE.LEFT, NODE.RIGHT} ' This is a slight modification to
 allow >2 brackets in a multiplier.
            ' If I have, say, (9x+3)*(10x-
3)*8x, then I will first select (9x+3) and compare it to (10x-
3) and 8x, both are doable, but the algorithm will first evaluate (9x+3)*(10x-3).
            ' It will the return the result as one bracket.

            For Each ELEMENT_LIST As List(Of TREE_NODE) In ITERATION_LIST

                Dim BUFFER As Integer = 0
                For A = 0 To ELEMENT_LIST.Count - 1
                    If ELEMENT_LIST.Count = 0 Then
                        Exit For
                    End If
                    ' As whenever there is a successful operation, the count is reduced by
1, I must add a buffer to properly reflect it.
                    Dim SELECTED_NODE As TREE_NODE = ELEMENT_LIST(A - BUFFER)

                    ' Now I must compare this selected node with every other node in the li
st.
                    Dim EVERY_OTHER_NODE As New List(Of TREE_NODE)
                    EVERY_OTHER_NODE.AddRange(NODE.LEFT)
                    EVERY_OTHER_NODE.AddRange(NODE.RIGHT)
                    EVERY_OTHER_NODE.Remove(SELECTED_NODE) '  Removes the one I dont want.


                    For C = 0 To EVERY_OTHER_NODE.Count - 1

                        Dim TEMPORARY_NODE As New TREE_NODE
                        TEMPORARY_NODE.LEFT.Add(SELECTED_NODE)
                        TEMPORARY_NODE.RIGHT.Add(EVERY_OTHER_NODE(C))
                        TEMPORARY_NODE.VALUE = "*"
                        ' I will simulate a binary multiplier so that it will accomodate fo
r previous code I wrote (that only works with 'two' multipliers, like 9x*(10x+3) or somethi
ng).

                        If TEMPORARY_NODE.RIGHT.Count = 1 Then
```

```
                            If TEMPORARY_NODE.RIGHT(0).VALUE = "+" Then
                                MODE = MULTIPLIER_NUM.SUM
                            Else
                                MODE = MULTIPLIER_NUM.OTHER
                            End If
                        Else
                            MODE = MULTIPLIER_NUM.OTHER
                        End If

                        If TEMPORARY_NODE.LEFT.Count = 1 Then
                            If TEMPORARY_NODE.VALUE = "+" Then ' These two checks mean I ha
ve a (a+b+c+...) form for my left node

                                Dim ITERATION_ARRAY = {TEMPORARY_NODE.LEFT(0).LEFT, TEMPORA
RY_NODE.LEFT(0).RIGHT}
                                For Each ELEMENT As List(Of TREE_NODE) In ITERATION_ARRAY '
 Sum Nodes have left and right.
                                    For B As Integer = 0 To ELEMENT.Count() - 1
                                        If Not ELEMENT(B) Is Nothing Then ' Makes sure it e
xists.
                                            Dim NODE_ELEMENT As TREE_NODE = ELEMENT(B)

                                            ' I have the node I want to multiply.
                                            MULTIPLIER_SIMPLIFIER_CALCULATOR(MODE, TEMPORAR
Y_NODE, NODE_ELEMENT, TEMPORARY_NODE.RIGHT)
                                            BUFFER += 1
                                        End If
                                    Next
                                Next
                                TEMPORARY_NODE.VALUE = "+"
                                TEMPORARY_NODE.LEFT.RemoveAt(0)
                                TEMPORARY_NODE.RIGHT = New List(Of TREE_NODE)
                            End If
                        End If
                        If TEMPORARY_NODE.LEFT.Count >= 1 And TEMPORARY_NODE.RIGHT.Count =
1 Then ' This is the 1 scenario the top if statement doesn't take into account.
                            ' I have had some preperations done beforehand so that all the
* nodes have the + nodes on the right.
                            Dim BEFORE_COUNT As Integer = TEMPORARY_NODE.LEFT.Count
                            '  Console.WriteLine("DETECTED THE SHIT" & IN_ORDER(NODE, True)
)
                            If TEMPORARY_NODE.RIGHT(0).VALUE = "+" Then ' These two checks
mean I have a (a+b+c+...) form for my right node, ie A*(x+y+z)
                                ' Console.WriteLine("DETECTED THE SHIT")
                                Dim NEW_NODE As New TREE_NODE
                                NEW_NODE.VALUE = "*"
                                If TEMPORARY_NODE.LEFT.Count = 2 Then
                                    NEW_NODE.LEFT.Add(TEMPORARY_NODE.LEFT(0))
                                    NEW_NODE.RIGHT.Add(TEMPORARY_NODE.LEFT(1))
                                ElseIf TEMPORARY_NODE.LEFT.Count > 2 Then
                                    NEW_NODE.LEFT = TEMPORARY_NODE.LEFT.GetRange(0, TEMPORA
RY_NODE.LEFT.Count - 1) ' Splits the newnode so it is displayed properly.
                                    NEW_NODE.RIGHT = TEMPORARY_NODE.LEFT.GetRange(TEMPORARY
_NODE.LEFT.Count - 1, 1)
                                Else
                                    NEW_NODE.LEFT.Add(TEMPORARY_NODE.LEFT(0))
                                    Dim ONE_NODE As New TREE_NODE
                                    ONE_NODE.VALUE = "1"
```

```
                                    NEW_NODE.RIGHT.Add(ONE_NODE)
                                End If



                                MULTIPLIER_SIMPLIFIER_CALCULATOR(MODE, TEMPORARY_NODE, NEW_
NODE.CLONE(), TEMPORARY_NODE.RIGHT)
                                TEMPORARY_NODE.VALUE = "+"
                                TEMPORARY_NODE.LEFT.RemoveRange(0, BEFORE_COUNT)
                                TEMPORARY_NODE.RIGHT = New List(Of TREE_NODE)
                                TEMPORARY_NODE.RIGHT.Add(TEMPORARY_NODE.LEFT(0))
                                TEMPORARY_NODE.LEFT.RemoveAt(0)
                            End If
                        End If

                        If TEMPORARY_NODE.VALUE = "+" Then ' This means that there was a su
ccessful calculation.
                            ' We will remove the two nodes that were previously in the temp
orary node
                            ELEMENT_LIST.Remove(SELECTED_NODE)
                            NODE.LEFT.Remove(EVERY_OTHER_NODE(C))
                            NODE.RIGHT.Remove(EVERY_OTHER_NODE(C))
                            If NODE.LEFT.Count + NODE.RIGHT.Count = 0 Then ' I will replace
 this node with the formed node, as there will only be 1 element in the * node.
                                NODE.VALUE = TEMPORARY_NODE.VALUE
                                NODE.LEFT.AddRange(TEMPORARY_NODE.LEFT)
                                NODE.RIGHT.AddRange(TEMPORARY_NODE.RIGHT)
                                TEMPORARY_NODE.LEFT.RemoveRange(0, TEMPORARY_NODE.LEFT.Coun
t)
                                TEMPORARY_NODE.RIGHT.RemoveRange(0, TEMPORARY_NODE.RIGHT.Co
unt)
                                TEMPORARY_NODE = Nothing
                            Else
                                NODE.RIGHT.Add(TEMPORARY_NODE)
                            End If
                            BUFFER += 1 ' Set the buffer.
                            Exit For 'Exit the for
                        End If


                    Next
                Next

            Next
        End If


        Return NODE

    End Function

    Private Sub FRACTION_COLLECTER_WRAPPER(NODE As TREE_NODE)
        ' As the function 'SIMPLE_COLLECT_FRACTION_DENOMINATORS' does not level operators w
hen it creates Addition Nodes.
        LEVEL_OPERATORS(NODE)
        SIMPLE_COLLECT_FRACTION_DENOMINATORS(NODE)
    End Sub
```

```
    Private Sub CLEANUP_ADDITION(NODE As TREE_NODE)

    End Sub

    Private Function SIMPLE_COLLECT_FRACTION_DENOMINATORS(NODE As TREE_NODE)
        ' This is made to combine fractions with the same denominator.
        ' It should be a fairly simple scenarior to solve, and the reason for its existence
 is for the like term collector to work properly.
        ' There are a lot more nuances to this solution, though, so I named this one 'Simpl
e'.
        ' A 'common denominator' can normally be found with ease.
        ' a/b + c/d, I need to multiply by every other denominator for each term. a/b * d/d
. c/d * b/b.

        ' This function was made before any Multiplication Solver was implemented, so it is
 crippled in functionality.
        ' I do not intend to make a fully featured simplifier, so this is fine.
        ' ~I'll leave it at that before I enter a self serving tangent and realise the enor
mity of what I'm attempting to 'partially' solve.

        ' I decided to use a slightly more efficient method compared to COLLECT_LIKE_TERMS.
        ' It is a pre-order traversal.

        Dim ITERATION_ARRAY = {NODE.LEFT, NODE.RIGHT}
        Dim NODE_LIST As New List(Of TREE_NODE) 'Used so that I can easily compare.



        For Each ELEMENT As List(Of TREE_NODE) In ITERATION_ARRAY ' This will just loop thr
ough all the nodes and recursively call them.
            ' To be fair I can probably change all the functions below to this form, but it
 will become a little annoying to read.
            If NODE.VALUE = "+" Then
                ' Imagine a circumstance where there is a + node and a bunch of variables.
Even if I am performing a recursive call, eventually they will all terminate to the nodes i
n the + node.
                ' Thus I will be able to list all the nodes that need to be compared, all w
hile having it do it to all plus nodes, as it's recursive.
                ' Wunderbar!
                For A As Integer = 0 To ELEMENT.Count() - 1
                    If (A) <= (ELEMENT.Count - 1) Then
                        Dim TREE_ELEMENT As TREE_NODE = COLLECT_LIKE_TERMS(ELEMENT(A))
                        If TREE_ELEMENT.VALUE = "/" Then
                            For B As Integer = 0 To NODE_LIST.Count() - 1 ' Same method as
COLLECT_LIKE_TERMS
                                If IN_ORDER(TREE_ELEMENT.RIGHT(0), True) = IN_ORDER(NODE_LI
ST(B).RIGHT(0), True) Then ' Comparing denominators.
                                    Dim NEW_NODE, ADD_NODE As New TREE_NODE
                                    ADD_NODE.VALUE = "+"

                                    NEW_NODE.VALUE = "/"
                                    NEW_NODE.RIGHT.Add(TREE_ELEMENT.RIGHT(0)) ' I add the c
ommon denominator
                                    NEW_NODE.LEFT.Add(ADD_NODE) ' Division is binary, so I
add a sum node for the two numerators being added.

                                    NEW_NODE.LEFT(0).LEFT.Add(TREE_ELEMENT.LEFT(0)) ' I add
 the numerators to this sum node, in left and right.
```

```
                                    NEW_NODE.LEFT(0).RIGHT.Add(NODE_LIST(B).LEFT(0))
                                    ELEMENT.Remove(TREE_ELEMENT)
                                    ITERATION_ARRAY(1).Remove(NODE_LIST(B)) ' Removes the n
ode_list node from the array. Badly.
                                    ITERATION_ARRAY(0).Remove(NODE_LIST(B))

                                    NODE_LIST.RemoveAt(B)
                                    NODE_LIST.Add(NEW_NODE) ' Adds the new node to be compa
red in the node list.

                                    ELEMENT.Add(NEW_NODE) ' Adds the new node.
                                    Exit For
                                Else
                                    NODE_LIST.Add(TREE_ELEMENT)
                                End If
                            Next
                            If NODE_LIST.Count = 0 Then
                                NODE_LIST.Add(TREE_ELEMENT)
                            End If
                        End If
                    End If
                Next
            Else
                For A As Integer = 0 To ELEMENT.Count() - 1
                    COLLECT_LIKE_TERMS(ELEMENT(A)) 'Go down and down and down
                Next
            End If
        Next

        Return NODE
    End Function

    Structure SEPERATED_TERMS
        Public COEFFICIENT As Integer
        Public VARIABLE As TREE_NODE
    End Structure

    Private Function COLLECT_LIKE_TERMS(NODE As TREE_NODE)
        ' To collect like terms there are certain rules.
        ' For example, 3xy + 2xy^2 cannot be collected as they do not have the same variabl
e.
        ' For trees I must identify the variables of each 'term' and be able to compare whe
ther they are able to collected.
        ' This of course can only happen on + modifiers, and due to my simplifying beforeha
nd I do not need to worry about '-' nodes.

        ' To identify terms -
        ' I already know that a single term will can be represented as 3 f g (...) *, where
 f and g are some functions.
        ' Thus, every term will eventually terminate to a multplication node, or to a numbe
r if it is not a variable.
        ' Therefore, when I eventually encounter an execution Stack Item where my root is a
 +, and my children are either numbers or variables, I will be able to compare them.

        ' I will use a pre order search for this circumstance.

        Dim ITERATION_ARRAY = {NODE.LEFT, NODE.RIGHT}
        Dim NODE_LIST As New List(Of TREE_NODE) 'Used so that I can easily compare.
```

```
        Dim ADDED = False
        ' I consider this an 'Up to Down' Method.


        If NODE.VALUE = "+" Then ' Like terms can be collected.
            For Each ELEMENT As List(Of TREE_NODE) In ITERATION_ARRAY
                For A As Integer = 0 To ELEMENT.Count() - 1
                    If (A) <= (ELEMENT.Count - 1) Then
                        Dim ELEMENT_A As TREE_NODE = ELEMENT(A) ' The element I will compar
e against all the others within the NODE_LIST.
                        If ELEMENT_A.VALUE <> "+" Then
                            Dim ELEMENT_TOTAL, ELEMENT_B As TREE_NODE
                            ' Console.WriteLine(IN_ORDER(ELEMENT_A, True) & " juh ")
                            For Each ELEMENT_B_TEMP As TREE_NODE In NODE_LIST ' I will now
compare it to every node I HAVE looked at. (This is better than, say, comparing it to every
thing from the getgo.)
                                Dim TERMS_A, TERMS_B As SEPERATED_TERMS
                                TERMS_A = RETURN_VARIABLE_AND_COEFFICIENT(ELEMENT_A.CLONE()
)
                                TERMS_B = RETURN_VARIABLE_AND_COEFFICIENT(ELEMENT_B_TEMP.CL
ONE())
                                'Console.WriteLine(IN_ORDER(ELEMENT_A, True) & " WHAT " & I
N_ORDER(ELEMENT_B_TEMP, True))


                                Dim ALL_TERMS_A As New List(Of TREE_NODE)
                                ALL_TERMS_A.AddRange(TERMS_A.VARIABLE.LEFT)
                                ALL_TERMS_A.AddRange(TERMS_A.VARIABLE.RIGHT)
                                Dim ALL_TERMS_B As New List(Of TREE_NODE)
                                ALL_TERMS_B.AddRange(TERMS_B.VARIABLE.LEFT)
                                ALL_TERMS_B.AddRange(TERMS_B.VARIABLE.RIGHT)

                                Dim STRING_TERMS_A = (From ITEM In ALL_TERMS_A Select IN_OR
DER(ITEM, True)).ToList()

                                Dim STRING_TERMS_B = (From ITEM In ALL_TERMS_B Select IN_OR
DER(ITEM, True)).ToList() ' This allows me to compare the two lists, so that in cases where
 I have 9xy + 9yx, I can still find they have the same variables.


                                If ALL_TERMS_A.Count = ALL_TERMS_B.Count Then ' They must h
ave the same number of variables.
                                    For Each ITEM In STRING_TERMS_A
                                        STRING_TERMS_B.Remove(ITEM)
                                    Next

                                    If STRING_TERMS_B.Count = 0 Then ' If I just compared t
heir string then it wouldnt allow 9xy + 9yx as although they're the same, the string wont b
e.
                                        'Console.WriteLine(IN_ORDER(TERMS_A.VARIABLE, True)
 & " " & IN_ORDER(TERMS_B.VARIABLE, True))
                                        Dim NEW_NODE As New TREE_NODE
                                        NEW_NODE.VALUE = TERMS_A.COEFFICIENT + TERMS_B.COEF
FICIENT ' This will be the new coefficient
                                        ' Console.WriteLine("FINALISED THING" & IN_ORDER(TE
RMS_B.VARIABLE, True) & "NODE ROOT" & TERMS_B.VARIABLE.LEFT(0).VALUE)
                                        If TERMS_B.VARIABLE.VALUE = "*" Then
```

```
                                        TERMS_B.VARIABLE.LEFT.Insert(0, NEW_NODE) ' I w
ill add the coefficient to the front of the node's variable.
                                    ElseIf TERMS_B.VARIABLE.VALUE = "/" Then ' As "/" i
s binary, then I must insert it into the extra node beyond the left node. Say * or +. Ie, i
ts some addition, or multiplication, divided by something else.
                                        TERMS_B.VARIABLE.LEFT(0).LEFT.Insert(0, NEW_NOD
E) ' I'll go left in the left because who likes it right.
                                    End If
                                    ELEMENT_TOTAL = TERMS_B.VARIABLE ' This will be the
 new node.
                                    ELEMENT_B = ELEMENT_B_TEMP ' This is here so I can
remove it.
                                End If
                            End If
                        Next

                        If Not ELEMENT_B Is Nothing Then ' As it is an 'each' loop, I w
ill need to modify it outside of it.
                            NODE_LIST.Remove(ELEMENT_B)
                            If ELEMENT.Contains(ELEMENT_B) Then
                                ELEMENT.Remove(ELEMENT_B)
                            Else
                                ITERATION_ARRAY(1).Remove(ELEMENT_B) ' "It just works."
 TM
                                ITERATION_ARRAY(0).Remove(ELEMENT_B) ' If the list beco
mes massive then this will become a problem. It won't become massive.
                            End If
                            ELEMENT.Remove(ELEMENT_A)
                            If NODE.LEFT.Count > NODE.RIGHT.Count Then
                                NODE.RIGHT.Add(ELEMENT_TOTAL)
                            Else
                                NODE.LEFT.Add(ELEMENT_TOTAL)
                            End If
                        Else ' This means no like term could be found. Thus I will just
 add ELEMENT_A to the NODE_LIST to be compared to the rest.
                            NODE_LIST.Add(ELEMENT_A)
                        End If
                        ELEMENT_B = Nothing
                        ELEMENT_A = Nothing
                    End If
                End If
            Next
        Next
    End If
    If Not NODE.LEFT Is Nothing Then
        For A As Integer = 0 To NODE.LEFT.Count() - 1
            COLLECT_LIKE_TERMS(NODE.LEFT(A)) 'A simple traversal, which is nice.
        Next
    End If

    If Not NODE.RIGHT Is Nothing Then
        For A As Integer = 0 To NODE.RIGHT.Count() - 1
            COLLECT_LIKE_TERMS(NODE.RIGHT(A)) ' This goes down each path.
        Next
    End If
```

```
        Return NODE


    End Function

    Private Function RETURN_VARIABLE_AND_COEFFICIENT(NODE As TREE_NODE)
        ' This will go through a node and remove the coefficient. It returns an array of th
e cleaned node and the coefficient.
        ' Do not use this on a + node, as that is not okay and hurts me in many ways.
        ' UPDATE -
I can not be bothered to stop it from doing that, but it doesn't seem to do much except was
te resources. Probably fine, but I won't test it to make sure. That would give me a reason
to fix it.
        Dim COEFFICIENT As Integer = 1
        Dim RETURN_STRUCTURE As New SEPERATED_TERMS

        'Console.WriteLine("THIS IS WHAT WILL BE SOPRTED " & IN_ORDER(NODE, True))
        If Not NODE.VALUE = "+" Then
            If Not NODE.LEFT Is Nothing Then
                For A As Integer = 0 To NODE.LEFT.Count() - 1
                    If A <= (NODE.LEFT.Count - 1) And NODE.VALUE <> "^" Then ' Makes sure i
t exists.
                        Dim NODE_ELEMENT_S As SEPERATED_TERMS = RETURN_VARIABLE_AND_COEFFIC
IENT(NODE.LEFT(A))
                        Dim NODE_ELEMENT As TREE_NODE = NODE_ELEMENT_S.VARIABLE
                        COEFFICIENT *= NODE_ELEMENT_S.COEFFICIENT
                        If IsNumeric(NODE_ELEMENT.VALUE) Then
                            NODE.LEFT.RemoveAt(A) ' Removes the coefficient
                            COEFFICIENT *= NODE_ELEMENT.VALUE
                            ' Console.WriteLine("addd 1" & COEFFICIENT)
                        End If
                        If NODE_ELEMENT.RIGHT.Count = 1 And NODE_ELEMENT.LEFT.Count = 1 The
n
                            If NODE_ELEMENT.VALUE = "*" And NODE_ELEMENT.LEFT(0).VALUE = "-
1" Then ' This is a negative number
                                NODE.LEFT.RemoveAt(A) ' Removes the coefficient
                                COEFFICIENT *= -NODE_ELEMENT.RIGHT(0).VALUE
                                'Console.WriteLine("addd 2" & COEFFICIENT)
                            End If
                        End If
                    End If
                Next
            End If

            If Not NODE.RIGHT Is Nothing Then
                For A As Integer = 0 To NODE.RIGHT.Count() - 1
                    If NODE.RIGHT.Count = 1 And IsNumeric(NODE.RIGHT(0).VALUE) Then ' This
takes into account that a number will be in the form a*b, where a is the coefficient and b
"1"
                        If NODE.RIGHT(0).VALUE = 1 Then
                            Exit For
                        End If
                    End If
                    If A <= (NODE.RIGHT.Count - 1) And NODE.VALUE <> "^" Then  ' Makes sure
 it exists.
                        Dim NODE_ELEMENT_S As SEPERATED_TERMS = RETURN_VARIABLE_AND_COEFFIC
IENT(NODE.RIGHT(A))
                        Dim NODE_ELEMENT As TREE_NODE = NODE_ELEMENT_S.VARIABLE
```

```
                        COEFFICIENT *= NODE_ELEMENT_S.COEFFICIENT
                        If IsNumeric(NODE_ELEMENT.VALUE) Then
                            NODE.RIGHT.RemoveAt(A) ' Removes the coefficient
                            COEFFICIENT *= NODE_ELEMENT.VALUE
                        End If
                        If NODE_ELEMENT.RIGHT.Count = 1 And NODE_ELEMENT.LEFT.Count = 1 The
n
                            If NODE_ELEMENT.VALUE = "*" And NODE_ELEMENT.LEFT(0).VALUE = "-
1" Then ' This is a negative number
                                NODE.RIGHT.RemoveAt(A) ' Removes the coefficient
                                COEFFICIENT *= -NODE_ELEMENT.RIGHT(0).VALUE
                            End If
                        End If
                    End If
                Next
            End If
        End If
        'Console.WriteLine("RETURNED COF " & COEFFICIENT)
        RETURN_STRUCTURE.COEFFICIENT = COEFFICIENT
        RETURN_STRUCTURE.VARIABLE = NODE
        Return RETURN_STRUCTURE

    End Function


    ' Trivial Flattening functions

    Private Sub SUB_RATIONAL_FUNCTION(PARENT_NODE As TREE_NODE, NODE As TREE_NODE, TYPE As
ETYPE) ' TYPE is the orientation of the child, left or right.
        ' First Case
        ' Division nodes are binary!
        If TYPE = ETYPE.LEFT And NODE.VALUE = "/" And PARENT_NODE.VALUE = "/" Then ' The ch
ild is the numerator and both are division
            Dim NUMERATOR_CHILD As List(Of TREE_NODE) ' The left children. This will only b
e one element, but due to my system it is still in a list.
            NUMERATOR_CHILD = NODE.LEFT ' This is the numerator of the child.
            Dim PRODUCT_TREE As New TREE_NODE
            PRODUCT_TREE.VALUE = "*"
            PRODUCT_TREE.LEFT = NODE.RIGHT
            PRODUCT_TREE.RIGHT = PARENT_NODE.RIGHT ' This is the product node.
            PARENT_NODE.LEFT = NUMERATOR_CHILD
            Dim HOLDER As New List(Of TREE_NODE)
            HOLDER.Add(PRODUCT_TREE)
            PARENT_NODE.RIGHT = HOLDER ' Sets the same
            ' This equates to the same thing, but with only one division node.
        End If

        ' Second Case
        If TYPE = ETYPE.RIGHT And NODE.VALUE = "/" And PARENT_NODE.VALUE = "/" Then
            Dim PRODUCT_TREE As New TREE_NODE
            PRODUCT_TREE.VALUE = "*"
            PRODUCT_TREE.LEFT = PARENT_NODE.LEFT
            PRODUCT_TREE.RIGHT = NODE.RIGHT ' This is the product node.
            Dim DENOMINATOR As List(Of TREE_NODE)
            DENOMINATOR = NODE.LEFT ' The denominator of the parent is the numerator of the
 child.
            Dim HOLDER As New List(Of TREE_NODE)
            HOLDER.Add(PRODUCT_TREE)
```

```
            PARENT_NODE.LEFT = HOLDER
            PARENT_NODE.RIGHT = DENOMINATOR
        End If

        'Third Case
        If NODE.VALUE = "/" And PARENT_NODE.VALUE = "*" Then
            PARENT_NODE.VALUE = "/"
            Dim CHILDREN_OF_PARENT As New List(Of TREE_NODE) ' I must get all the children
except the child I have selected.
            For Each NODE_LOOP As TREE_NODE In PARENT_NODE.LEFT
                If Not NODE_LOOP Is NODE Then
                    CHILDREN_OF_PARENT.Add(NODE_LOOP)
                End If
            Next
            For Each NODE_LOOP As TREE_NODE In PARENT_NODE.RIGHT ' I keep right and left an
yway, eventhough it is not needed.
                If Not NODE_LOOP Is NODE Then
                    CHILDREN_OF_PARENT.Add(NODE_LOOP)
                End If
            Next
            Dim PRODUCT_NODE As New TREE_NODE
            PRODUCT_NODE.VALUE = "*"
            PRODUCT_NODE.LEFT = NODE.LEFT
            PRODUCT_NODE.RIGHT = CHILDREN_OF_PARENT
            Dim HOLDER As New List(Of TREE_NODE)
            HOLDER.Add(PRODUCT_NODE)
            PARENT_NODE.LEFT = HOLDER
            PARENT_NODE.RIGHT = NODE.RIGHT
        End If
    End Sub

    Private Function RATIONAL_SIMPLIFICATION(NODE As TREE_NODE)
        ' This is a much more complicated function than the first two REMOVE_NEGATIVITY AND
 LEVEL_OPERATORS.
        ' It contains 3 cases.
        ' The point of this is to make it impossible for a division node to have a child th
at is also a division node.
        ' This aims to make it so that there is only one division node as the root (in any
instance there is a division node..!), and its children are multplicative nodes (if needed)
.

        ' The three cases are checks to transform this tree in traversal.
        ' The first check looks for when a division node's numerator is a division node.
        ' The second check is the same, but for when the division node is in the denominato
r.
        ' The third case is when a child of a multplicative node is a division node. Only t
he first division node in the children is used.

        ' These three cases are the foundation of this function.
        ' Postorder traversal will be used for this simplification due to its inherent left
 right design.

        ' !!Please note that due to the nature of this algorithm it must be run multiple ti
mes to ensure the simplification is completed!!

        If Not NODE.LEFT Is Nothing Then
            For A As Integer = 0 To NODE.LEFT.Count() - 1 ' I am modifying the stream, so I
 have to do this loop method.
```

```
                If A <= NODE.LEFT.Count - 1 Then
                    If Not NODE.LEFT(A) Is Nothing Then ' Makes sure it exists.
                        Dim NODE_ELEMENT As TREE_NODE = NODE.LEFT(A)
                        Dim TYPE_RETURN As String = RATIONAL_SIMPLIFICATION(NODE_ELEMENT) '
 Returns either a * or a /
                        If Not TYPE_RETURN Is Nothing Then
                            SUB_RATIONAL_FUNCTION(NODE, NODE_ELEMENT, ETYPE.LEFT) ' I send
the current node and the child node that has been deemed possible to simplify.
                        End If
                    End If
                End If
            Next
        End If
        If Not NODE.RIGHT Is Nothing Then
            For A As Integer = 0 To NODE.RIGHT.Count() - 1
                If A <= NODE.LEFT.Count - 1 Then
                    If Not NODE.RIGHT(A) Is Nothing Then
                        Dim NODE_ELEMENT As TREE_NODE = NODE.RIGHT(A)
                        Dim TYPE_RETURN As String = RATIONAL_SIMPLIFICATION(NODE_ELEMENT) '
 Returns either a * or a / (currently optional data that may be used in later stages)
                        If Not TYPE_RETURN Is Nothing Then ' This will only run if it retur
ned
                            SUB_RATIONAL_FUNCTION(NODE, NODE_ELEMENT, ETYPE.RIGHT) ' As the
 required code is large I will seperate it in a new function
                        End If
                    End If
                End If
            Next
        End If

        If NODE.VALUE = "/" Or NODE.VALUE = "*" Then ' This will be used to identify when a
 child has a division or multiplication node.
            ' The general idea is that the traversal will bring us to the bottom of the lef
t sub tree, and when it does it will return nothing.
            ' The recursive stack will then be popped through subsequent 'upping'. In cases
 where a certain instance gets a returned value of a * or a /
            ' It will then do the checks for the cases.
            Return NODE.VALUE
        End If
        Return Nothing
    End Function


    Private Sub REMOVE_NEGATIVITY(NODE As TREE_NODE)

        ' This uses a post order traversal. Ie, I need to see each consitutent node in a ro
ot, right and left, and evaluate whether I can convert them.
        ' The simple requirement is such that their root is negative.
        ' In this case a new node is created such that it is the multiplication of -
1 to the negative node.
        ' The negative root is then made positive and I continue the traversal.
        If Not NODE.LEFT Is Nothing Then
            For Each NODE_ELEMENT As TREE_NODE In NODE.LEFT ' This loops through each node,
 resulting in the same for binary trees, but will also allows non binary trees in cases whe
re its needed.
                REMOVE_NEGATIVITY(NODE_ELEMENT)
            Next
        End If
```

```
        If Not NODE.RIGHT Is Nothing Then
            For Each NODE_ELEMENT As TREE_NODE In NODE.RIGHT
                REMOVE_NEGATIVITY(NODE_ELEMENT)
            Next
        End If
        ' Recursively calls this. I will do the calculations here, as both the right and le
ft have been evaluated.
        ' I assume the tree is in binary format. This algorithm must be used before any oth
er for this to be true.
        If NODE.LEFT.Count = 1 And NODE.RIGHT.Count = 1 And NODE.VALUE = "-
" Then ' There are two items and the root is a negative.
            Dim NEW_NODE, NEGATIVE_NODE As New TREE_NODE
            NEW_NODE.VALUE = "*"
            NEGATIVE_NODE.VALUE = "-1"
            NEW_NODE.LEFT.Add(NEGATIVE_NODE) ' Adds the negative multiplication
            NEW_NODE.RIGHT.Add(NODE.RIGHT.Item(0)) ' Adds the exisiting node.
            NODE.RIGHT.Remove(NODE.RIGHT.Item(0)) ' Removes the exisiting node
            NODE.RIGHT.Add(NEW_NODE)
            NODE.VALUE = "+" ' Sets the root as a plus. This should now be correctly optimi
sed.
        End If
    End Sub



    Private Sub PREPARATION_BY_UNIVERSAL_RULING(NODE As TREE_NODE)
        PREPARATION_BY_TIMES_RULING(NODE)
        LEVEL_OPERATORS(NODE)
        PREPERATION_POWER_RULING(NODE)
    End Sub

    Private Sub PREPERATION_POWER_RULING(NODE As TREE_NODE)

        ' Does not assume binary tree.


        Dim ITERATION_ARRAY = {NODE.LEFT, NODE.RIGHT}

        If Not NODE.LEFT Is Nothing Then
            For Each NODE_ELEMENT As TREE_NODE In NODE.LEFT
                PREPERATION_POWER_RULING(NODE_ELEMENT)
            Next
        End If
        If Not NODE.RIGHT Is Nothing Then
            For Each NODE_ELEMENT As TREE_NODE In NODE.RIGHT
                PREPERATION_POWER_RULING(NODE_ELEMENT)
            Next
        End If
        If NODE.VALUE = "*" Then
            For Each ELEMENT As List(Of TREE_NODE) In ITERATION_ARRAY
                ' This means it may have variables in it.
                ' I can modify it such that I loop through each of the variables, and if th
ey have no left and right children, I can modify it to a power node.
                ' This works as I assume I will traverse each node, so I will be looking at
 all the children. (He says with mocking self-assurance.)
                Dim CLONED_NODES As New List(Of TREE_NODE)
                CLONED_NODES.AddRange(ELEMENT)
                For A As Integer = 0 To ELEMENT.Count - 1
```

```
                    If CLONED_NODES(A).VALUE <> "+" And CLONED_NODES(A).VALUE <> "-
" And CLONED_NODES(A).VALUE <> "/" And CLONED_NODES(A).VALUE <> "*" And CLONED_NODES(A).VAL
UE <> "^" And Not IsNumeric(CLONED_NODES(A).VALUE) Then ' Its just a check that means its a
 variable. There are other ways.

                        Dim NEW_NODE, ONE_NODE As New TREE_NODE
                        NEW_NODE.VALUE = "^"
                        ONE_NODE.VALUE = "1"
                        NEW_NODE.LEFT.Add(CLONED_NODES(A))
                        NEW_NODE.RIGHT.Add(ONE_NODE)
                        ELEMENT.Remove(CLONED_NODES(A))
                        ELEMENT.Add(NEW_NODE)
                    End If
                Next
            Next
        End If
    End Sub
    Private Sub PREPARATION_BY_TIMES_RULING(NODE As TREE_NODE)

        ' To easily accomodate for variables and coefficients, I'll change all the numerica
ls to a*1.
        ' This is so that functions like 'COLLECT_LIKE_TERMS' can function with numbers. It
 assumes everything is in the form a*b, where a is the coefficient, and b the variable.
        ' In this case the variable will be 1, and this is fine when comparisons are made a
s it will still proceed with summations.

        ' I will also change variables to all have powers.
        ' Ie, x = x^1, but in tree form.
        ' This is useful for algorithms, as I won't need to accomodate special cases and tu
rn spaghetti into taglietti. (assume one is worse than the other)
        ' If you can think of a better name than I'm all ears. Unfortunately I don't have a
ny.


        Dim ITERATION_ARRAY = {NODE.LEFT, NODE.RIGHT}

        If Not NODE.LEFT Is Nothing Then
            For Each NODE_ELEMENT As TREE_NODE In NODE.LEFT
                PREPARATION_BY_TIMES_RULING(NODE_ELEMENT)
            Next
        End If
        If Not NODE.RIGHT Is Nothing Then
            For Each NODE_ELEMENT As TREE_NODE In NODE.RIGHT
                PREPARATION_BY_TIMES_RULING(NODE_ELEMENT)
            Next
        End If
        If NODE.VALUE = "+" Then
            For Each ELEMENT As List(Of TREE_NODE) In ITERATION_ARRAY
                For A As Integer = 0 To ELEMENT.Count - 1
                    Dim CHECK As Dictionary(Of String, String) = MATCH_COLLECTION_TO_DICTIO
NARY(Regex.Matches(ELEMENT(A).VALUE, "[*,+,/,-]")) 'Looks for operators.
                    If (IsNumeric(ELEMENT(A).VALUE) Or CHECK.Count = 0 And ELEMENT(A).VALUE
 <> "0") Then ' Every single node is checked, so I won't need to loop through.
                        Dim NEW_NODE, ONE_NODE As New TREE_NODE
                        NEW_NODE.VALUE = "*"
                        ONE_NODE.VALUE = "1"
                        NEW_NODE.RIGHT.Add(ONE_NODE) ' Adds the 1 multiplication
                        NEW_NODE.LEFT.Add(ELEMENT(A).CLONE) ' Adds the exisiting node.
```

```
                        ELEMENT.RemoveAt(A)
                        ELEMENT.Add(NEW_NODE)
                    End If
                Next
            Next
        End If
    End Sub
    Private Function LEVEL_OPERATORS(NODE As TREE_NODE)

        ' In cases where (3+3)+4 occur there will be a binary method of parsing. Ie, two ro
ot nodes to create the tree.
        ' This is not needed, as a plus can be done to many items at once. I can do 3 5 3 +
, and it will not change anything.
        ' This concept is also true for multiplication.
        ' Therefore the simplification involves changing * and + into a single node with ma
ny children - as many as possible.
        ' Negation is assumed to be removed, but for division we must be careful and ignore
 them.

        ' I will be using postorder for this. (postorder works from the bottom to the top n
aturally, which I am fanciful towards. Dunno why.)

        ' The entire children, left and right, will be examined. I will then need to go 'up
' a root, and see if that root is the same as my current one.
        ' If there are two */+ then I will edit the tree to accomodate for only one root, a
nd all of the children.
        ' This is done recursively so by the end the entire tree should be simplified.
        ' A~wala

        If Not NODE.LEFT Is Nothing Then
            For A As Integer = 0 To NODE.LEFT.Count() - 1
                If A < NODE.LEFT.Count() Then
                    Dim NODE_ELEMENT As TREE_NODE = NODE.LEFT(A)
                    Dim TYPE_RETURN As String = LEVEL_OPERATORS(NODE_ELEMENT) ' This is vit
al. It returns a +, * or nothing. #
                    'If its a + or a * that means that the child can be simplified, if this
 root node has the same root.
                    ' We have to do this as this is a recursive execution from bottom to up
.
                    ' It will only return this value when it can no longer traverse down, s
o that means I can edit from bottom to top. (duh that is how it works)
                    If TYPE_RETURN <> Nothing And TYPE_RETURN = NODE.VALUE Then
                        Dim TO_MOVE_ELEMENT_TOTAL As New List(Of TREE_NODE)
                        TO_MOVE_ELEMENT_TOTAL.AddRange(NODE_ELEMENT.RIGHT)
                        TO_MOVE_ELEMENT_TOTAL.AddRange(NODE_ELEMENT.LEFT)
                        NODE.LEFT.RemoveAt(A)
                        NODE.LEFT.AddRange(TO_MOVE_ELEMENT_TOTAL)
                    End If
                End If
            Next
        End If
        If Not NODE.RIGHT Is Nothing Then
            For a As Integer = 0 To NODE.RIGHT.Count() - 1
                If a < NODE.RIGHT.Count() Then
                    Dim NODE_ELEMENT As TREE_NODE = NODE.RIGHT(a)
                    Dim TYPE_RETURN As String = LEVEL_OPERATORS(NODE_ELEMENT) ' this is vit
al. it returns a +, * or nothing. #
                    If TYPE_RETURN <> Nothing And TYPE_RETURN = NODE.VALUE Then
```

```
                        Dim TO_MOVE_ELEMENT_TOTAL As New List(Of TREE_NODE)
                        TO_MOVE_ELEMENT_TOTAL.AddRange(NODE_ELEMENT.RIGHT)
                        TO_MOVE_ELEMENT_TOTAL.AddRange(NODE_ELEMENT.LEFT)
                        NODE.RIGHT.RemoveAt(a)
                        NODE.RIGHT.AddRange(TO_MOVE_ELEMENT_TOTAL)
                    End If
                End If
            Next
        End If

        If NODE.VALUE = "+" Or NODE.VALUE = "*" Then
            Return NODE.VALUE
        End If

        Return Nothing
    End Function

    Public Sub DIVISION_MULTIPLICATION(NODE As TREE_NODE)
        ' let (a^b)^c =a^(b*c)

        If NODE.VALUE = "^" Then
            ' "^" Nodes are binary, so I can easily check this condition.
            If NODE.LEFT(0).VALUE = "^" Then
                NODE.LEFT.Add(NODE.LEFT(0).LEFT(0)) ' I will add the "a" to the top most po
wer node.

                Dim MULTIPLY_NODE_1, MULTIPLY_NODE_2, MULTIPLY_NODE_HOLD As New TREE_NODE

                MULTIPLY_NODE_1 = NODE.RIGHT(0).CLONE ' the c
                MULTIPLY_NODE_2 = NODE.LEFT(0).RIGHT(0).CLONE ' the b
                MULTIPLY_NODE_HOLD.VALUE = "*"
                MULTIPLY_NODE_HOLD.LEFT.Add(MULTIPLY_NODE_1)
                MULTIPLY_NODE_HOLD.RIGHT.Add(MULTIPLY_NODE_2) ' This forms b*c

                NODE.RIGHT.RemoveAt(0) ' removes the original c, so it is left as ((a^b)a)^
                NODE.RIGHT.Add(MULTIPLY_NODE_HOLD) ' add the multiply node, so ((a^b)a)^(b*
c)

                NODE.LEFT.RemoveAt(0) ' Remove the original power node. ie the a^b in the l
eft, so we get a^(b*c)
            End If
        End If


        If Not NODE.LEFT Is Nothing Then
            For Each NODE_ELEMENT As TREE_NODE In NODE.LEFT
                DIVISION_MULTIPLICATION(NODE_ELEMENT)
            Next
        End If
        If Not NODE.RIGHT Is Nothing Then
            For Each NODE_ELEMENT As TREE_NODE In NODE.RIGHT
                DIVISION_MULTIPLICATION(NODE_ELEMENT)
            Next
        End If

    End Sub

    Public Sub NEGATIVE_POWERS_TO_DIVISORS(NODE As TREE_NODE)
```

```
        ' When finished with differentiation, revoke the negative powers to division nodes,
 so that it is displayed properly
        ' Let a^-b = 1/a^b


        If Not NODE.LEFT Is Nothing Then
            For Each NODE_ELEMENT As TREE_NODE In NODE.LEFT
                NEGATIVE_POWERS_TO_DIVISORS(NODE_ELEMENT)
            Next
        End If
        If Not NODE.RIGHT Is Nothing Then
            For Each NODE_ELEMENT As TREE_NODE In NODE.RIGHT
                NEGATIVE_POWERS_TO_DIVISORS(NODE_ELEMENT)
            Next
        End If

        If NODE.VALUE = "^" Then
            If IsNumeric(NODE.RIGHT(0).VALUE) Then
                If Convert.ToInt32(NODE.RIGHT(0).VALUE) < 0 Then
                    NODE.RIGHT(0).VALUE = Convert.ToString(-
Convert.ToInt32(NODE.RIGHT(0).VALUE))

                    Dim CLONED_NODE = NODE.CLONE
                    If CLONED_NODE.RIGHT(0).VALUE = "1" Then ' Remove the power node :OOOO
                        CLONED_NODE.VALUE = CLONED_NODE.LEFT(0).VALUE

                        CLONED_NODE.LEFT.RemoveAt(0) ' Remove the original stuff
                        CLONED_NODE.RIGHT.RemoveAt(0)

                        CLONED_NODE.LEFT.AddRange(NODE.LEFT(0).LEFT) ' Add the left and rig
ht lists of the left node. Ie the a of a^b
                        CLONED_NODE.RIGHT.AddRange(NODE.LEFT(0).RIGHT)
                    End If


                    NODE.VALUE = "/" ' Make this node a divisor

                    Dim ONE_NODE As New TREE_NODE
                    ONE_NODE.VALUE = "1" ' Create the One_Node

                    NODE.LEFT.RemoveAt(0) 'Clear the node
                    NODE.RIGHT.RemoveAt(0)

                    NODE.LEFT.Add(ONE_NODE) ' Add the 1
                    NODE.RIGHT.Add(CLONED_NODE) ' Add the modified node. Ie 1/(a^b).
                End If
            End If
        End If
    End Sub
End Class
```

POSTFIX_EXPRESSION.vb

This handles the input of strings. This object is never outright created, as it is inherited by the super class **SIMPLE_SIMPLIFY**. My IPSO chart shows what it does.

```
Imports System.Text.RegularExpressions
'\(([^(]+)\) FINDS ANY ITEMS WITH (..)
Class POSTFIX_EXPRESSION : Inherits UTILITIES ' defines any expression that is wished to be
 simplified to postfix.

    Protected OPERATORS As Dictionary(Of String, Array) = New Dictionary(Of String, Array)
From
    {{"+", {2, False}}, {"-
", {2, False}}, {"/", {3, False}}, {"*", {3, False}}, {"^", {4, True}}} 'defines precedence


    Protected TO_BE_SIMPLFIED As String ' original input string
    Protected STRING_ARRAY() As Char ' an array of the string for each character
    Protected INFIX_VERSION As String ' the infix version of the string, 3+3 would be 3 3 +
    Protected OUTPUT_QUEUE As Queue(Of String) = New Queue(Of String)() ' used to convert i
nfix to postfix
    Protected OPERATOR_STACK As Stack(Of String) = New Stack(Of String)()
    Protected OUTPUT_POSTFIX As List(Of String) = New List(Of String)
    Protected TRUE_EXPRESSION_LIST As List(Of String) = New List(Of String)
    'FIXED!
    Function CONVERT_CHAR_ARRAY_TO_CORRECT_FORM(INPUT As String) ' This changes something l
ike {1,3,x,+,3} into {13x,+,3}, otherwise I wouldn't be able to tell where the numbers end,
 easily.
        Dim TEMP_QUEUE As Queue(Of String) = New Queue(Of String) ' A temporary queue used
to hold the numbers, say if I have {1,3,x,+}, it will add {1,3,x} to the stack, and stop at
 the +.
        Dim TEMP_STRING_LIST As List(Of String) = New List(Of String)
        Dim CONTINUE_ADD_UNTIL_OPERATOR As Boolean = False ' This is used whenever somethin
g lke a 3x^23 occurs. When it sees a ^ it will activate, and continue until a operator is s
een.
        Dim COUNT As Integer
        For Each CHAR_TO_COMBINE As String In INPUT
            COUNT += 1
            If (CONTINUE_ADD_UNTIL_OPERATOR And (IsNumeric(CHAR_TO_COMBINE) Or Char.IsLette
r(CHAR_TO_COMBINE))) Or (IsNumeric(CHAR_TO_COMBINE) Or Char.IsLetter(CHAR_TO_COMBINE)) And
Not OPERATORS.ContainsKey(CHAR_TO_COMBINE) And (CHAR_TO_COMBINE <> "(" Or CHAR_TO_COMBINE <
> ")") Then
                ' Console.WriteLine(CHAR_TO_COMBINE & " wtf")
                If Not IsNumeric(CHAR_TO_COMBINE) Then ' Adds a multplication sign so that
it can be converted to tree form
                    If TEMP_QUEUE.Count > 0 Then ' If the queue contains anything then it w
ill append the letters together
                        TEMP_STRING_LIST.Add(QUEUE_TO_STRING(TEMP_QUEUE))
                        TEMP_QUEUE = New Queue(Of String) ' Resets the stack.
                    End If
                    If TEMP_STRING_LIST.Count >= 1 Then
                        If TEMP_STRING_LIST(TEMP_STRING_LIST.Count - 1) <> "*" And TEMP_STR
ING_LIST(TEMP_STRING_LIST.Count - 1) <> "(" And TEMP_STRING_LIST(TEMP_STRING_LIST.Count - 1
) <> "+" And TEMP_STRING_LIST(TEMP_STRING_LIST.Count - 1) <> "-
" And TEMP_STRING_LIST(TEMP_STRING_LIST.Count - 1) <> "/" And TEMP_STRING_LIST(TEMP_STRING_
LIST.Count - 1) <> "^" Then
                            TEMP_STRING_LIST.Add("*")
                            TEMP_STRING_LIST.Add(CHAR_TO_COMBINE) ' Add any numbers or vari
ables to the queue, to be    made as a single entity.
                        Else
                            TEMP_QUEUE.Enqueue(CHAR_TO_COMBINE)
                        End If
                    Else
```

```
                        TEMP_QUEUE.Enqueue(CHAR_TO_COMBINE)
                    End If
                Else
                    TEMP_QUEUE.Enqueue(CHAR_TO_COMBINE) ' Add any numbers or variables to t
he queue, to be made as a single entity.
                End If
            ElseIf CHAR_TO_COMBINE <> " " Then ' Then it is an operator.
                If TEMP_QUEUE.Count > 0 Then ' If the queue contains anything then it will
append the letters together
                    TEMP_STRING_LIST.Add(QUEUE_TO_STRING(TEMP_QUEUE)) ' Adds the created st
ring, such as 13x.
                    TEMP_QUEUE = New Queue(Of String) ' Resets the stack.
                    TEMP_STRING_LIST.Add(CHAR_TO_COMBINE)
                Else
                    If CHAR_TO_COMBINE = "-" Then
                        If TEMP_STRING_LIST.Count > 0 Then
                            Dim CHECK As Dictionary(Of String, String) = MATCH_COLLECTION_T
O_DICTIONARY(Regex.Matches(TEMP_STRING_LIST(TEMP_STRING_LIST.Count - 1), "[*,+,/,-,(,),^]")
)
                            If IsNumeric(TEMP_STRING_LIST(TEMP_STRING_LIST.Count - 1)) Or C
HECK.Count = 0 Then
                                TEMP_STRING_LIST.Add(CHAR_TO_COMBINE)
                            Else
                                TEMP_QUEUE.Enqueue(CHAR_TO_COMBINE)
                            End If
                        Else
                            TEMP_STRING_LIST.Add("*")
                            TEMP_STRING_LIST.Add(CHAR_TO_COMBINE & "1")
                        End If
                    Else
                        TEMP_STRING_LIST.Add(CHAR_TO_COMBINE)
                    End If
                End If
            End If
        Next
        If TEMP_QUEUE.Count > 0 Then
            TEMP_STRING_LIST.Add(QUEUE_TO_STRING(TEMP_QUEUE)) ' Adds the created string, su
ch as 13x.
        End If
        For Each ELEMENT As String In TEMP_STRING_LIST
            'Console.WriteLine("FINISH2" & ELEMENT)
        Next
        Return TEMP_STRING_LIST ' Sets the global variable.
    End Function

    Sub New(ByVal INPUT As String) ' intitlises systems so that it can be used by the progr
am.
        TO_BE_SIMPLFIED = INPUT ' sets the protected variable to the input specified by the
 program
        STRING_ARRAY = INPUT.ToCharArray ' splits the string into an array so that it can b
e looped through
        Dim COUNT As Integer = 0
        TRUE_EXPRESSION_LIST = CONVERT_CHAR_ARRAY_TO_CORRECT_FORM(STRING_ARRAY)
        For Each ELEMENT As String In TRUE_EXPRESSION_LIST
            'Console.WriteLine("FINISH" & ELEMENT)
        Next
        INFIX_TO_POSTFIX() ' CONVERTS POSTFIX TO INFIX
```

```
    End Sub

    ' the output queue contains the output of the conversion
    ' the stack is used for operators
    ' the algorithm used here is called the 'shunting yard algorithm',  first described by
dijkstra.
    ' the method and actual conversion will be modified to accomodate for actual variables,
 such as 5x + 2x, not just 2 + 3; otherwise it would be pointless.

    Public Sub INFIX_TO_POSTFIX() ' converts the infix input to postfix (reverse polish not
ation).
        For I = 0 To TRUE_EXPRESSION_LIST.Count() - 1
            Dim CURRENT_CHAR As String = TRUE_EXPRESSION_LIST(I)
            If CURRENT_CHAR = "(" Then ' it pushes the ( onto the operator statck to begin
parenthese
                OPERATOR_STACK.Push(CURRENT_CHAR)
            ElseIf CURRENT_CHAR = ")" Then ' this means it has closed parenthese, so it wil
l add the operators from the operator stack onto the output queue
                While OPERATOR_STACK.Peek() <> "("
                    OUTPUT_QUEUE.Enqueue(OPERATOR_STACK.Peek())
                    OPERATOR_STACK.Pop()
                End While
                OPERATOR_STACK.Pop() ' pops the )
            ElseIf OPERATORS.ContainsKey(CURRENT_CHAR) Then ' if it is an operator
                Dim OPERATOR_DATA As Array = OPERATORS.Item(CURRENT_CHAR)
                If OPERATOR_STACK.Count > 0 Then
                    If OPERATOR_STACK.Peek() <> "(" Then ' this is placed outside due to th
e operators dictionary not containing "(", thus the while loop would error out.
                        While OPERATORS.Item(OPERATOR_STACK.Peek())(0) > OPERATOR_DATA(0) O
r
                            (OPERATORS.Item(OPERATOR_STACK.Peek())(0) = OPERATOR_DATA(0) An
d OPERATOR_DATA(1) = False) ' this will pop the stack with any operations that cannot conti
nue to be held when this operator is pushed onto the stack
                            ' the workings include, if the current top stack operator has a
 higher precedence or if the operator has the same precedence and the token (the one being
added) is left (false) in associativity
                            ' the stack item must also not be a ')'
                            OUTPUT_QUEUE.Enqueue(OPERATOR_STACK.Peek())
                            OPERATOR_STACK.Pop()
                            If OPERATOR_STACK.Count = 0 Then
                                Exit While
                            End If
                            If OPERATOR_STACK.Count > 0 Then
                                If OPERATOR_STACK.Peek() = "(" Then
                                    Exit While
                                End If
                            End If
                        End While
                    End If
                End If
                If Not (CURRENT_CHAR = "-
" And TRUE_EXPRESSION_LIST(Math.Abs(I)) = "(") Then
                    OPERATOR_STACK.Push(CURRENT_CHAR) ' pushes the operator after all check
s are done
                Else
                    OUTPUT_QUEUE.Enqueue(CURRENT_CHAR)
                End If
            Else ' It is a normal number.
```

```
                OUTPUT_QUEUE.Enqueue(CURRENT_CHAR)
            End If
        Next
        While OPERATOR_STACK.Count() > 0 ' while there are items in the operator stack.
            OUTPUT_QUEUE.Enqueue(OPERATOR_STACK.Peek())
            OPERATOR_STACK.Pop()
        End While

        ' output
        Dim COUNT As Integer = 0
        Dim COMPLETED As String = ""
        Do
            COMPLETED = COMPLETED & OUTPUT_QUEUE.Peek()
            OUTPUT_POSTFIX.Add(OUTPUT_QUEUE.Dequeue.ToString())
        Loop While (OUTPUT_QUEUE.Count > 0)
    End Sub

End Class
```

QUESTION.vb

A question is just a class that contains data useful to the student and teacher. They are separated for this use case, but the normal class is normally always used.

```
' This is the class that is used to create questions.
' Handles the storage of answers and the custom interfaces it uses.

Enum QUESTION_TYPE_ANSWER
    DIFFERENTIATION
    SIMPLIFICATION
End Enum

Enum QUESTION_STATUS
    CORRECT
    INDETERMINED
    WRONG
End Enum

Class TEACHER_QUESTION : Inherits QUESTION

    Sub New(ByRef DATA_HANDLE_INPUT As DATA_HANDLE)
        DATA_HANDLER = DATA_HANDLE_INPUT
        AddHandler FORM1.QUESTION_RECOMPUTE_ANSWER.Click, RECALC_EVENT
    End Sub


    Dim RECALC_EVENT As EventHandler = Function(sender, e) RECOMPUTE_CHOSEN_QUESTION()
    Dim ADDING_EVENT As EventHandler = Function(sender, e) UPDATE_CLASS(False)
    Dim EDITING_EVENT As EventHandler = Function(sender, e) UPDATE_CLASS(True)
    Dim REVOKER_EVENT As EventHandler = Function(sender, e) REMOVE_HANDLER()

    Private Function UPDATE_CLASS(Optional EDIT As Boolean = False)
        QUESTION_TEXT = FORM1.QUESTION_INPUT.Text
        If Not EDIT Then
            DATA_HANDLER.ADD(Me)
        End If
    End Function
```

```
    Private Function RECOMPUTE_CHOSEN_QUESTION()
        ' This is for the recomputing whenever the user wants.

        Dim TO_BE_SOLVED As SIMPLE_SIMPLIFY '
        Debug.WriteLine("im doing")
        If QUESTION_ANSWER_TYPE = QUESTION_TYPE_ANSWER.SIMPLIFICATION Then
            TO_BE_SOLVED = New SIMPLE_SIMPLIFY(FORM1.QUESTION_INPUT.Text, False, True)
        ElseIf QUESTION_ANSWER_TYPE = QUESTION_TYPE_ANSWER.DIFFERENTIATION Then
            TO_BE_SOLVED = New SIMPLE_SIMPLIFY(FORM1.QUESTION_INPUT.Text, True) ' Parameter
s are Differentiate, Expand Brackets.
        End If
        FORM1.QUESTION_CREATION_ANSWER.Text = TO_BE_SOLVED.RESULT
        ANSWER_CLASS = TO_BE_SOLVED


        Return True
    End Function

    Function REMOVE_HANDLER()
        RemoveHandler FORM1.QUESTION_RECOMPUTE_ANSWER.Click, RECALC_EVENT
        RemoveHandler FORM1.QUESTION_CREATE.Click, ADDING_EVENT
        RemoveHandler FORM1.QUESTION_CREATE.Click, REVOKER_EVENT
        RemoveHandler FORM1.QUESTION_CREATION_EXIT.Click, REVOKER_EVENT
        RemoveHandler FORM1.QUESTION_CREATE.Click, EDITING_EVENT
        Return True
    End Function

    Public Function CHOSEN_QUESTION_TO_CREATE()
        ' This functions occurs when the user has selected a template and clicked create.

        Dim CHOSEN_QUESTION_TEMPLATE As String = FORM1.QUESTION_CHOOSER_LIST.SelectedItem.T
oString
        Dim TEMPLATE_ITEMS = DATA_HANDLER.QUESTION_DEFINERS.Item(CHOSEN_QUESTION_TEMPLATE)
        Type = FORM1.QUESTION_CHOOSER_LIST.SelectedItem.ToString
        ' Update the group 'Question Creation'
        FORM1.QUESTION_DISPLAY.Text = TEMPLATE_ITEMS(0) ' The first item is always the stri
ng question, like "Calculate the like terms." or whatever.
        QUESTION_TITLE = TEMPLATE_ITEMS(0)
        ' Calculate the number that this question will have, which is always one more than
the current number of created questions.
        Dim QUESTION_COUNT_NUM As Integer = DATA_HANDLER.RETURN_QUESTIONS().Count() + 1
        FORM1.QUESTION_COUNT.Text = QUESTION_COUNT_NUM
        ' Get a random question
        Dim RANDOM As New Random
        Dim QUESTION_INDEX As Integer = RANDOM.Next(1, TEMPLATE_ITEMS.Count) ' The random i
ndex for 'Template_items'.
        Dim QUESTION_STRING As String = TEMPLATE_ITEMS(QUESTION_INDEX)
        FORM1.QUESTION_INPUT.Text = QUESTION_STRING

        ' Create a 'Simple_Simplify' object that will answer the question.
        Debug.WriteLine(QUESTION_STRING)
        Dim TO_BE_SOLVED As SIMPLE_SIMPLIFY ' This will simplify the expression without exp
anding power brackets.

        If TEMPLATE_ITEMS(0) = DATA_HANDLER.QUESTION_1 Then ' This is a non differentiation
 question.
            TO_BE_SOLVED = New SIMPLE_SIMPLIFY(QUESTION_STRING, False, True)
```

```
            QUESTION_ANSWER_TYPE = QUESTION_TYPE_ANSWER.SIMPLIFICATION
        ElseIf TEMPLATE_ITEMS(0) = DATA_HANDLER.QUESTION_2 Then ' Differentiation does not
need bracket expansion.
            TO_BE_SOLVED = New SIMPLE_SIMPLIFY(QUESTION_STRING, True) ' Parameters are Diff
erentiate, Expand Brackets.
            QUESTION_ANSWER_TYPE = QUESTION_TYPE_ANSWER.DIFFERENTIATION
        End If

        FORM1.QUESTION_CREATION_ANSWER.Text = TO_BE_SOLVED.RESULT ' This displays the resul
t of the class's calculations.

        FORM1.QUESTION_REMOVE.Visible = False ' As this is a creation we can not remove it.
        FORM1.QUESTION_CREATE.Visible = True

        ANSWER_CLASS = TO_BE_SOLVED

        AddHandler FORM1.QUESTION_CREATE.Click, REVOKER_EVENT
        AddHandler FORM1.QUESTION_CREATION_EXIT.Click, REVOKER_EVENT
        AddHandler FORM1.QUESTION_CREATE.Click, ADDING_EVENT

        Return True
    End Function

    Public Function EDIT_QUESTION(QUESTION_INDEX As Integer)

        FORM1.QUESTION_INPUT.Text = QUESTION_TEXT
        FORM1.QUESTION_COUNT.Text = QUESTION_INDEX
        FORM1.QUESTION_DISPLAY.Text = QUESTION_TITLE
        RECOMPUTE_CHOSEN_QUESTION()


        AddHandler FORM1.QUESTION_RECOMPUTE_ANSWER.Click, RECALC_EVENT
        AddHandler FORM1.QUESTION_CREATE.Click, REVOKER_EVENT
        AddHandler FORM1.QUESTION_CREATE.Click, EDITING_EVENT
        Return True
    End Function
End Class


Class QUESTION

    Public TYPE As String

    Protected ENABLED As Boolean = True ' Determines if editable.
    Protected DATA_HANDLER As DATA_HANDLE
    Protected ANSWER_CLASS As SIMPLE_SIMPLIFY ' Dynamic answer that can be recomputed.
    Public QUESTION_ANSWER_TYPE As QUESTION_TYPE_ANSWER

    Public QUESTION_TEXT As String ' This is the actual question that is saved in the class
.
    Public QUESTION_TITLE As String
    Public STATUS As QUESTION_STATUS ' This is used for submissions. I won't make a seperat
e class for this as it will just complicate things for no reason... e.e
    Public TEACHER_EDITED As Boolean = False

    Public Function RETURN_QUESTION()
        Return QUESTION_TEXT
    End Function
```

```
    Public Function RETURN_COMPUTED_ANSWER()
        Return ANSWER_CLASS.RESULT
    End Function

    Public Function RETURN_QUESTION_TYPE()
        Return QUESTION_ANSWER_TYPE.ToString()
    End Function

    Public Function RETURN_QUESTION_TITLE()
        Return QUESTION_TITLE
    End Function

    Protected ANSWER As String = ""

    Public Function RETURN_ANSWER()
        Return ANSWER
    End Function

    Public Function RETURN_ANSWER_FUNCTION_OUTPUT(VALUE As Integer)
        Return ANSWER_CLASS.GET_OUTPUT(VALUE) ' Says f(x) = 9x^2, this will output the 9x^2
 for the selected value, like 1.
    End Function

    Public Overridable Sub SUBMIT_ANSWER(INPUT As String) 'Submits answer..
        Me.ANSWER = INPUT

        ' When the user answer is set it is expected that the actual answer will also exist
, so this will check for it.
        If ANSWER_CLASS Is Nothing Then
            If QUESTION_ANSWER_TYPE = QUESTION_TYPE_ANSWER.SIMPLIFICATION Then
                ANSWER_CLASS = New SIMPLE_SIMPLIFY(Me.QUESTION_TEXT, False, True)
            ElseIf QUESTION_ANSWER_TYPE = QUESTION_TYPE_ANSWER.DIFFERENTIATION Then
                ANSWER_CLASS = New SIMPLE_SIMPLIFY(Me.QUESTION_TEXT, True) ' Parameters are
 Differentiate, Expand Brackets.
            End If
        End If
    End Sub

    'Public Sub SET_ANSWER_MENU() ' Sets the answer menu's (groupbox) properties.
    '    For Each IARRAY As Array In Properties ' Loop through the properties list
    '        If ANSWER_MENU.Controls.ContainsKey(IARRAY(0)) Then
    '            Dim LOCAL_CONTROL As Array = ANSWER_MENU.Controls.Find(IARRAY(0), True)
    '            For Each ICONTROL As Control In LOCAL_CONTROL
    '                ICONTROL.Text = IARRAY(1)
    '            Next
    '        End If
    '    Next
    'End Sub



End Class
```

SIMPLE_SIMPLIFY.vb

This class is what properly solves expressions. It encompasses and encapsulates the entirety of **OPTIMISER, EXPRESSION TREE,** and **POSTFIX_EXPRESSION**. When this class is created it is expected to be used for simplify an input string expression.

```
Imports System.Text.RegularExpressions
' This is a simplification class, which works out the solutions to problems like 300x+50x+2
y-4z
' This class inherits EXPRESSION_TREE, so that it can create a tree to parse the solution e
asily.
' Redunancy may appear.

Class SIMPLE_SIMPLIFY : Inherits EXPRESSION_TREE

    Public RESULT As String
    Public OPTIMISER_CLASS As New OPTIMISER

    Sub New(INPUT As String, Optional DIFFERENTIATE As Boolean = False, Optional EXPAND_BRA
CKETS As Boolean = False)
        MyBase.New(INPUT)
        CREATE_TREE() ' Creates a tree from the input specified.
        RESULT = OPTIMISER_CLASS.OPTIMISE_TREE(TREE_ROOT)
        If DIFFERENTIATE Then
            DIFFERENTIATE_EXPRESSION()
        End If
        If EXPAND_BRACKETS Then
            EXPAND()
        End If
    End Sub

    Public Sub DIFFERENTIATE_EXPRESSION() 'Intermediary for the recursive solver.
        RESULT = OPTIMISER_CLASS.DIFFERENTIATE()
    End Sub
    Public Sub EXPAND() 'Intermediary for the recursive solver.
        RESULT = OPTIMISER_CLASS.EXPAND_BRACKETS()
    End Sub

    Public Function GET_OUTPUT(VALUE As Integer)
        Return OPTIMISER_CLASS.OUTPUT_VALUE_WRAP(VALUE)
    End Function

    Private Function ZERO_TO_NEGATIVE(NUMBER As Integer)
        If NUMBER = 0 Then
            Return -1
        End If
        Return NUMBER
    End Function

    Private Function REMOVE_ZERO_CLEANUP(INPUT As String) ' This is used to cleanup strings
 that contain 0 elements. This is required due to the nature of the solver.
        ' Console.WriteLine("TO DODD" & INPUT)
        Dim ELEMENTED_INPUT = INPUT.ToCharArray ' The array {-,6,x,^,2,+}
        Dim TO_ADD As New Queue(Of String) ' This will be used to create 'items', like 3x^2
. This is then added to a final list.
        Dim PREVIOUS_ELEMENT As String = ""
        Dim ELEMENT_TO_ADD As String = ""
        Dim FINISHED_LIST As New List(Of String)
        For Each ELEMENT As String In ELEMENTED_INPUT
```

```
            ' Console.WriteLine("Current Element" & ELEMENT)
            If PREVIOUS_ELEMENT <> "+" And PREVIOUS_ELEMENT <> "-
" And PREVIOUS_ELEMENT <> "*" And PREVIOUS_ELEMENT <> "/" Then ' The previous element isn't
 an operator.
                FINISHED_LIST.Add(ELEMENT) ' Add the operator seperately.
            Else
                If TO_ADD.Count > 0 Then
                    For I As Integer = 1 To TO_ADD.Count()
                        ELEMENT_TO_ADD = ELEMENT_TO_ADD & TO_ADD.Dequeue()
                    Next
                    FINISHED_LIST.Add(ELEMENT_TO_ADD)
                    ELEMENT_TO_ADD = ""
                End If
                TO_ADD.Enqueue(ELEMENT)
                FINISHED_LIST.Add(PREVIOUS_ELEMENT) ' Add actual data.
            End If
            PREVIOUS_ELEMENT = ELEMENT
        Next
        For I As Integer = 1 To TO_ADD.Count()
            ELEMENT_TO_ADD = ELEMENT_TO_ADD & TO_ADD.Dequeue()
        Next
        FINISHED_LIST.Add(ELEMENT_TO_ADD)
        ELEMENT_TO_ADD = ""
        For Each ELEMENT As String In FINISHED_LIST
            ' Console.WriteLine("FIISHED " & ELEMENT)
        Next
        Return True

    End Function
End Class
```

TREE_OPTIMISER.vb

This contains the general class of the tree node. It was made to be serializable, so that it is clonable.

```
Imports System.IO
Imports System.Collections
Imports System.Runtime.Serialization.Formatters.Binary
Imports System.Runtime.Serialization
<Serializable()> Class TREE_NODE ' This is a general tree with n children for each node. It
 is partitioned into left and right to allow for inorder traversing.
    Public VALUE As String
    Public LEFT As New List(Of TREE_NODE)
    Public RIGHT As New List(Of TREE_NODE)

    Function CLONE() As TREE_NODE ' A deep clone function.

        ' If the object is nil then return nothing
        If (Object.ReferenceEquals(Me, Nothing)) Then Return Nothing

        Dim FORM As New BinaryFormatter()
        Dim STREAM As New MemoryStream()

        FORM.Serialize(STREAM, Me)
        STREAM.Seek(0, SeekOrigin.Begin) ' clones the object... deep.
```

```
        Return CType(FORM.Deserialize(STREAM), TREE_NODE) ' Returns the object in a useful
type.

    End Function
End Class
```

UTILITIES.vb

A general class that contains commonly used functions.

```
Imports System.Text.RegularExpressions
' A commonly used collection of methods.
Class UTILITIES

    ' For Trees

    Public Function IN_ORDER(NODE As TREE_NODE, FIRST As Boolean) ' Returns an inorder stri
ng.
        Dim OUTPUT As String = "" ' String to be created.
        Dim Count As Integer = 0
        If FIRST = False And NODE.LEFT.Count > 0 And NODE.RIGHT.Count > 0 And NODE.VALUE <>
 "*" Then
            OUTPUT = OUTPUT & "("
        End If
        If Not NODE.LEFT Is Nothing Then
            For Each NODE_ELEMENT As TREE_NODE In NODE.LEFT ' This loops through each node,
 resulting in the same for binary trees, but will also allows non binary trees in cases whe
re its needed.
                Count += 1
                If Count > 1 Then
                    If NODE.VALUE = "+" Then
                        OUTPUT = OUTPUT & " "
                    End If
                    OUTPUT = OUTPUT & NODE.VALUE
                    If NODE.VALUE = "+" Then
                        OUTPUT = OUTPUT & " "
                    End If
                End If
                OUTPUT = OUTPUT & IN_ORDER(NODE_ELEMENT, False)
            Next
        End If
        Dim CHECK As Dictionary(Of String, String) = MATCH_COLLECTION_TO_DICTIONARY(Regex.M
atches(NODE.VALUE, "[*,+,/,-,^]"))
        If NODE.RIGHT.Count > 0 Or CHECK.Count = 0 Then ' Removes instances of 9*x*, where
there is nothing on the right node.
            If NODE.VALUE = "+" Then
                OUTPUT = OUTPUT & " "
            End If
            OUTPUT = OUTPUT & NODE.VALUE
            If NODE.VALUE = "+" Then
                OUTPUT = OUTPUT & " "
            End If
            If Not NODE.RIGHT Is Nothing Then
                Count = 0
                For Each NODE_ELEMENT As TREE_NODE In NODE.RIGHT
                    Count += 1
                    If Count > 1 Then
```

```
                        If NODE.VALUE = "+" Then
                            OUTPUT = OUTPUT & " "
                        End If
                        OUTPUT = OUTPUT & NODE.VALUE
                        If NODE.VALUE = "+" Then
                            OUTPUT = OUTPUT & " "
                        End If
                    End If
                    OUTPUT = OUTPUT & IN_ORDER(NODE_ELEMENT, False)
                Next
            End If
        End If
        If FIRST = False And NODE.LEFT.Count > 0 And NODE.RIGHT.Count > 0 And NODE.VALUE <>
 "*" Then
            OUTPUT = OUTPUT & ")"
        End If
        Return OUTPUT
    End Function

    ' For Linear Data Structures

    Public Function MATCH_COLLECTION_TO_DICTIONARY(INPUT As MatchCollection) ' Converts a m
atch collection to dictionary.
        Dim TO_OUTPUT As New Dictionary(Of String, String)
        For Each ELEMENT As Match In INPUT
            If ELEMENT.Value <> "" And ELEMENT.Value <> "0" Then ' For some reason some reg
ex gives "" in some areas.
                TO_OUTPUT.Add(ELEMENT.Value, ELEMENT.Value)
            End If
        Next
        Return TO_OUTPUT ' Returns the collected dictionary
    End Function

    Public Function MATCH_COLLECTION_TO_STRING(INPUT As MatchCollection) ' Converts a match
 collection to a string.
        Dim TO_OUTPUT As String = ""
        For Each ELEMENT As Match In INPUT
            TO_OUTPUT = TO_OUTPUT & ELEMENT.Value
        Next
        Return TO_OUTPUT ' Returns the collected string
    End Function

    Public Function QUEUE_TO_STRING(INPUT As Queue(Of String))
        Dim CREATED_STRING As String = "" ' The created string.
        For Each CHAR_LIST As String In INPUT ' Goes through each char and appends them tog
ether
            CREATED_STRING = CREATED_STRING & CHAR_LIST
        Next
        Return CREATED_STRING
    End Function

    Public Function DICTIONARY_TO_STRING(INPUT As Dictionary(Of String, String))
        Dim RETURN_STRING As String = ""
        For Each ELEMENT As KeyValuePair(Of String, String) In INPUT
            'Console.WriteLine(ELEMENT.Value)
            RETURN_STRING = RETURN_STRING & ELEMENT.Value
        Next
        Return RETURN_STRING
```

```
    End Function
    Public Function DICTIONARY_KEYS(INPUT As Dictionary(Of String, String))
        Dim RETURN_STRING As List(Of String) = New List(Of String)
        For Each ELEMENT As KeyValuePair(Of String, String) In INPUT
            RETURN_STRING.Add(ELEMENT.Key)
        Next
        Return RETURN_STRING
    End Function

    Public Function LIST_OF_TREES_TO_STRING(LIST_NODES As List(Of TREE_NODE))
        Dim NEW_NODE, ONE_NODE As TREE_NODE
        NEW_NODE.VALUE = "*"
        ONE_NODE.VALUE = "1"
        NEW_NODE.RIGHT.Add(ONE_NODE)
        NEW_NODE.LEFT.AddRange(LIST_NODES)
        Return IN_ORDER(NEW_NODE, True)
    End Function


    Public Sub ALTERNATING_TREE_INSERTING(LIST_NODE, NODE)
        ' Adds one to the left, one to the right, and so on.
        ' ~left~right~left~right

        Dim ALTERNATING_EVEN_NUMBER As Integer = 1

        For Each Item As TREE_NODE In LIST_NODE
            If ALTERNATING_EVEN_NUMBER Mod 2 = 1 Then
                NODE.LEFT.Add(Item)
            Else
                NODE.RIGHT.Add(Item)
            End If
            ALTERNATING_EVEN_NUMBER += 1
        Next
    End Sub
End Class
```

Testing

There are quite a number of systems in play for this program, so I will break down the tests for the 3 prototypes I have. For the first two I will show the results in a console application, as this was where it was developed before being incorporated into a form.

The third prototype will be shown in both a video, and a list of tests here like normal.

I will be using 'black-box' testing as the method, and will list the tests in this form:

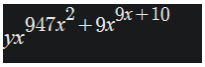
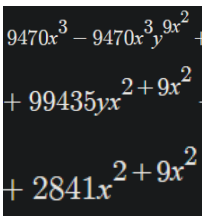
Test	Objective	Purpose of test	Test Data	Expected Result	Actual Result
The test number	*The objective it will partially/fully fulfil*	*Explained purpose of this test*	*Typical*		
			Erroneous		
			Extreme		

Each test will contain 2/3 sub-tests for each type of test data. Some of data remains unchanged, so I will have them apply to all of them. Also, the result will be colour coded to determine whether the result is correct or not.

Prototype 1&2

Tests

Test	Objective	Purpose of test	Test Data	Expected Result	Actual Result
1	3	Test whether addition of simple expressions is possible.	Typical Input: 3x+9x	10x	10*x
			Extreme Input: 3x+9x-10x+13x-132x	-117x	-117*x
2		Test whether addition of power expression is possible.	Typical Input: 13x^2+10x^2+10x	23x^2 +10x	23*(x^2) + 10*x
			Extreme Input: 13x^3-192x^3+10x-10x+32x^4+32x^3	32x^4 -147x^3	32*(x^4) + -147*(x^3)
3		Test whether multivariable expression addition is possible.	Typical Input: 3xy+3xy+9xy^2+9xy	15xy+9xy^2	9*x*(y^2) + 15*y*x
			Extreme Input: 9xy^3+10xy-10xy^2-10xy+293xy^4+13x+10x-300xy^4-300y^4x	-307xy^4 +9xy^3 -10xy^2 +23x	-10*x*(y^2) + 9*x*(y^3) + -307*x*(y^4) + 23*x
4		Test whether non-numeric power expression addition is possible.	Typical Input: 3x^(9x+10)+6x^(9x+10)	9x^(9x+10)	9*(x^(9*x + 10))
			Extreme Input: 3yx^(9x^2)+3x^(9x^2)+10x-10xy^(9x^2)+102x^(9x^2)y	10x-10xy^(9x^2) +105yx^(9x^2) +3x^(9x^2)	-10*x*(y^9*(x^2)) + 10*x + 3*(x^9*(x^2)) + 105*y*(x^9*(x^2))
5		Test whether numbers multiply.	Typical Input: 3*9x	27x	27*x
			Extreme Input: 3*10x^2+9*103x^2-10x^2	947x^2	947*(x^2)
6		Test whether power addition occurs on multiplying the same variables.	Typical Input: 3xxx	3x^3	3*(x^3)
			Extreme Input:	19x^4y^3z^2	19*(y^3)*(x^4) *(z^2)

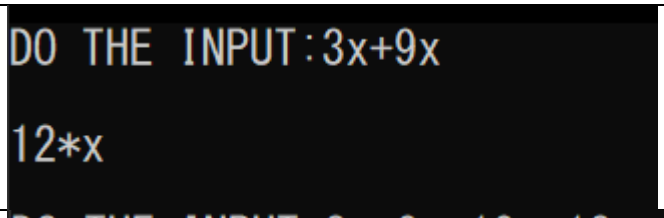
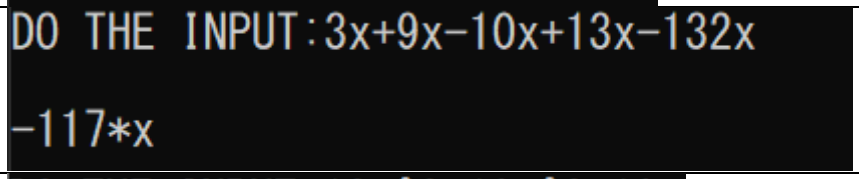
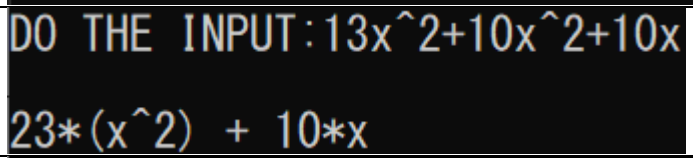
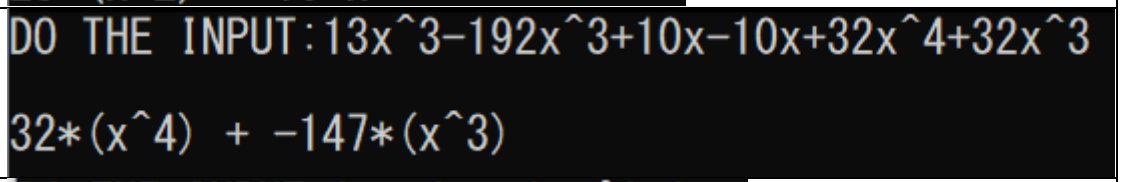
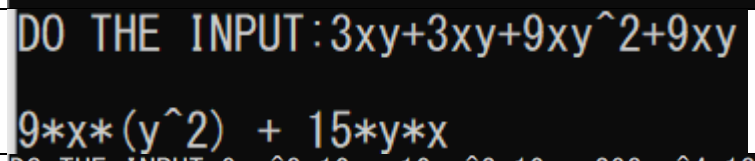
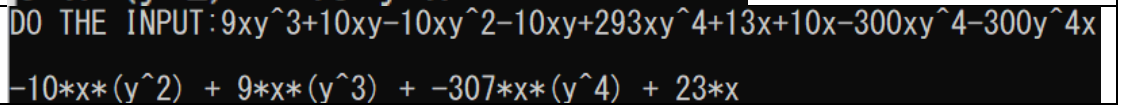
			9xyxzyxxzy+10yzxxyxyxz		
7		Test whether power addiction occurs on non-numerical powers.	Typical Input: x*x^(9x+10)	x^(9x+11)	(x^(11 + 9*x))
			Extreme Input: y*(x^(3*10x^2+9*103x^2-10x^2))*x^(3x^(9x+10)+6x^(9x+10))		y*(x^(947*(x^2) + 9*(x^(9*x + 10))))
8		Test whether single term multiplication works on brackets.	Typical Input: 3x*(10x+3)	30x^2+9x	30*(x^2) + 9*x
			Extreme Input: (3x^3y)*(99x^3+10x^2-9x+10)	297yx^6 +30yx^5 -27yx^4 +30yx^3	30*y*(x^5) + 297*y*(x^6) + 30*y*(x^3) + -27*y*(x^4)
9		Test whether 2 term bracket multiplication is possible.	Typical Input: (3x+9)*(9x+3)	27x^2+90x+27	90*x + 27 + 27*(x^2)
			Extreme Input: (9yx^2-10x)*(3x^(9x)+10)	27yx^(2+9x)+90yx^2-30x^(1+9x)-100x	-100*x + -30*(x^(9*x + 1)) + 27*y*(x^(9*x + 2)) + 90*y*(x^2)
10		Test whether ‘infinite’ term bracket multiplication is possible.	Typical Input: (3x^2+9x+10)*(9x^2-10+10x)	27x^4+111x^3+150x^2+10x-100	27*(x^4) + 111*(x^3) + -100 + 10*x + 150*(x^2)
			Extreme Input: (3*10x^2+9*103x^2-10x^2)*(3yx^(9x^2)+3x^(9x^2)+10x-10xy^(9x^2)+102x^(9x^2)y)		99435*y*(x^(2 + 9*(x^2))) - 9470*(x^3)*(y^9*(x^2)) + 9470*(x^3) + 2841*(x^(2 + 9*(x^2)))
11		Test whether multiple bracket multiplication is possible	Typical Input: (3x+9)*(9x+3)*(10x^2-19)	270x^4+900x^3-243x^2-1710x-513	270*(x^4) + -1710*x + 900*(x^3) + -243*(x^2) + -513
			Extreme Input: (3*10x^2+9*103x^2-10x^2)*(3yx^(9x^2)+3x^(9x^2)+10x-10xy^(9x^2)+102x^(9x^2)y)*(9x+3)	85230x^4+28410x^3-85230x^4y^(9x^2)-28410x^3y^(9x^2)+894915yx^(9x^2+3)+298305yx^(2+9x^2)+25569x^(9x^2+3)+8523x^(2+9x^2)	8523*(x^(2 + 9*(x^2))) + 298305*y*(x^(2 + 9*(x^2))) + -28410*(x^3)*(y^9*(x^2)) + 28410*(x^3) + 25569*(x^(3 + 9*(x^2))) + 894915*y*(x^(3 + 9*(x^2))) + -

					85230*(x^4)*(y^9*(x^2)) + 85230*(x^4)
12		Test fraction addition.	Typical Input: (3x/5)+10/5	(10 + 3x)/5	(10 + 3*x)/5
			Extreme Input: (10 + 3*x)/(9x^2-10)+(9x^2+3)/(9x^2-10)	(9x^2+3x+13)/(9x^2-10)	(9*(x^2) + 13 + 3*x)/(9*(x^2) + -10)
13		Test two term fraction multiplication.	Typical Input: 3*(9x/10)	27x/10	27*x/10
			Extreme Input: (3x/5)*(9x/10)	27x^2/50	27*(x^2)/50
14		Test bracket fraction multiplication.	Typical Input: ((3x+5)/10)*((3x+5)/10)	(9x^2+30x+25)/100	(30*x + 25 + 9*(x^2))/100
			Extreme Input: ((9xy^9-10y^2)/(20x^3-3))*((139x^2+3)/(u^3+10))	(1251x^3y^9+27xy^9-1390x^2y^2-30y^2)/(20x^3u^3+200x^3-3u^3-30)	(27*x*(y^9) + -30*(y^2) + -1390*(y^2)*(x^2) + 1251*(x^3)*(y^9))/(200*(x^3) + -30 + 20*(x^3)*(u^3) + -3*(u^3))
15		Test expanding bracket powers.	Typical Input: (x+y)^2	x^2+y^2+2xy	(y^2) + 2*x*y + (x^2)
			Extreme Input: (10y^2+3u+3x^(9x+1))^4	5400y^4x^(18x+2)-1080y^2x^(27x+3)+486u^2x^(18x+2)-3240y^2ux^(18x+2)+324ux^(27x+3)+81x^(36x+4)-12000y^6x^(9x+1)+324u^3x^(9x+1)-3240y^2u^2x^(9x+1)+10800y^4ux^(9x+1)+10000y^8+81u^4-1080y^2u^3+5400y^4u^2-12000y^6u	81*(u^4) + 81*(x^(36*x + 4)) + -270*(y^2)*(x^(3 + 27*x)) + 162*u*(x^(3 + 27*x)) + 162*u*(x^(27*x + 3)) + -1080*(u^3)*(y^2) + -12000*u*(y^6) + 486*(u^2)*(x^(2 + 18*x)) + 10800*u*(y^4)*(x^(9*x + 1)) + 10000*(y^8) + -810*(x^(27*x + 3))*(y^2) + -3240*u*(y^2)*(x^(2 + 18*x)) + 324*(u^3)*(x^(9*x + 1)) + -

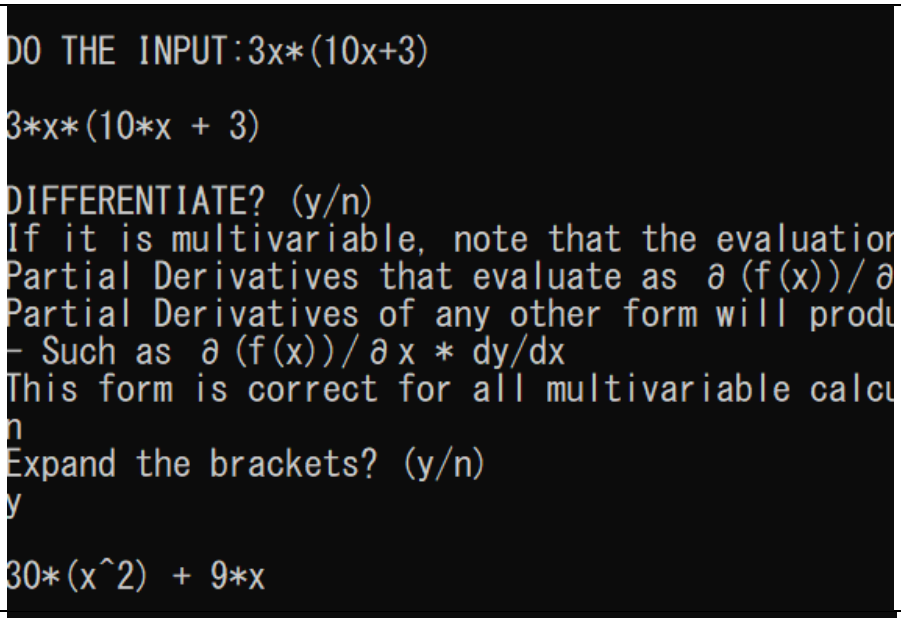
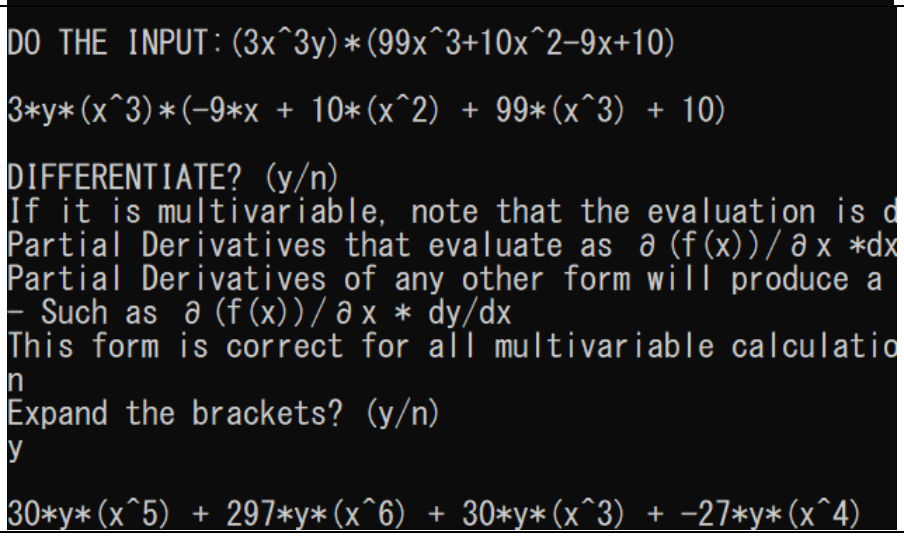
					12000*(y^6)*(x^(9*x + 1)) + 5400*(y^4)*(x^(2 + 18*x)) + 5400*(u^2)*(y^4) + -3240*(y^2)*(u^2)*(x^(9*x + 1))

Test Evidence:

Each test either has 2 or 3 different test data inputs. Here, the type of test will be stated with the letter, **a**, **b**, or **c**. These merely tell you which test data it is for, and is in the same order.

Test	Evidence
1 a	
1 b	
2 a	
2 b	
3 a	
3 b	

4 a	DO THE INPUT:3x^(9x+10)+6x^(9x+10) 9*(x^(9*x + 10))
4 b	DO THE INPUT:3yx^(9x^2)+3x^(9x^2)+10x-10xy^(9x^2)+102x^(9x^2)y -10*x*(y^9*(x^2)) + 10*x + 3*(x^9*(x^2)) + 105*y*(x^9*(x^2))
5 a	DO THE INPUT:3*9x 27*x
5 b	DO THE INPUT:3*10x^2+9*103x^2-10x^2 947*(x^2)
6 a	DO THE INPUT:3xxx 3*(x^3)
6 b	DO THE INPUT:9xyxzyxxzy+10yzxxyxyxz 19*(y^3)*(x^4)*(z^2)
7 a	DO THE INPUT:x*x^(9x+10) (x^(11 + 9*x))
7 b	DO THE INPUT:y*(x^(3*10x^2+9*103x^2-10x^2))*x^(3x^(9x+10)+6x^(9x+10)) y*(x^(947*(x^2) + 9*(x^(9*x + 10))))

8 a	 DO THE INPUT:3x*(10x+3) 3*x*(10*x + 3) DIFFERENTIATE? (y/n) If it is multivariable, note that the evaluation Partial Derivatives that evaluate as ∂(f(x))/∂ Partial Derivatives of any other form will produ - Such as ∂(f(x))/∂x * dy/dx This form is correct for all multivariable calcu n Expand the brackets? (y/n) y 30*(x^2) + 9*x
8 b	 DO THE INPUT:(3x^3y)*(99x^3+10x^2-9x+10) 3*y*(x^3)*(-9*x + 10*(x^2) + 99*(x^3) + 10) DIFFERENTIATE? (y/n) If it is multivariable, note that the evaluation is d Partial Derivatives that evaluate as ∂(f(x))/∂x *dx Partial Derivatives of any other form will produce a - Such as ∂(f(x))/∂x * dy/dx This form is correct for all multivariable calculatio n Expand the brackets? (y/n) y 30*y*(x^5) + 297*y*(x^6) + 30*y*(x^3) + -27*y*(x^4)

9 a

```
DO THE INPUT:(3x+9)*(9x+3)

*(9*x + 3)*(3*x + 9)

DIFFERENTIATE? (y/n)
If it is multivariable, no
Partial Derivatives that e
Partial Derivatives of any
- Such as ∂(f(x))/∂x * d
This form is correct for a
n
Expand the brackets? (y/n)
y

90*x + 27 + 27*(x^2)
```

9 b

```
DO THE INPUT:(9yx^2-10x)*(3x^(9x)+10)

*(3*(x^9*x) + 10)*(9*y*(x^2) + -10*x)

DIFFERENTIATE? (y/n)
If it is multivariable, note that the evaluation is d(f(x,y,z..
Partial Derivatives that evaluate as ∂(f(x))/∂x *dx will beco
Partial Derivatives of any other form will produce a ratio deri
- Such as ∂(f(x))/∂x * dy/dx
This form is correct for all multivariable calculations, no mat
n
Expand the brackets? (y/n)
y

-100*x + -30*(x^(9*x + 1)) + 27*y*(x^(9*x + 2)) + 90*y*(x^2)
```

10 a

```
DO THE INPUT:(3x^2+9x+10)*(9x^2-10+10x)

*(9*(x^2) + -10 + 10*x)*(9*x + 3*(x^2) + 10)

DIFFERENTIATE? (y/n)
If it is multivariable, note that the evaluatio
Partial Derivatives that evaluate as ∂(f(x))/∂
Partial Derivatives of any other form will prod
- Such as ∂(f(x))/∂x * dy/dx
This form is correct for all multivariable calc
n
Expand the brackets? (y/n)
y

27*(x^4) + 111*(x^3) + -100 + 10*x + 150*(x^2)
```

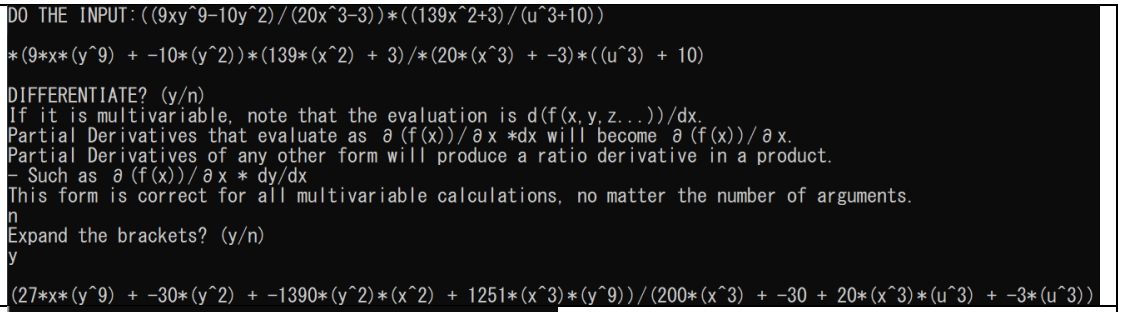
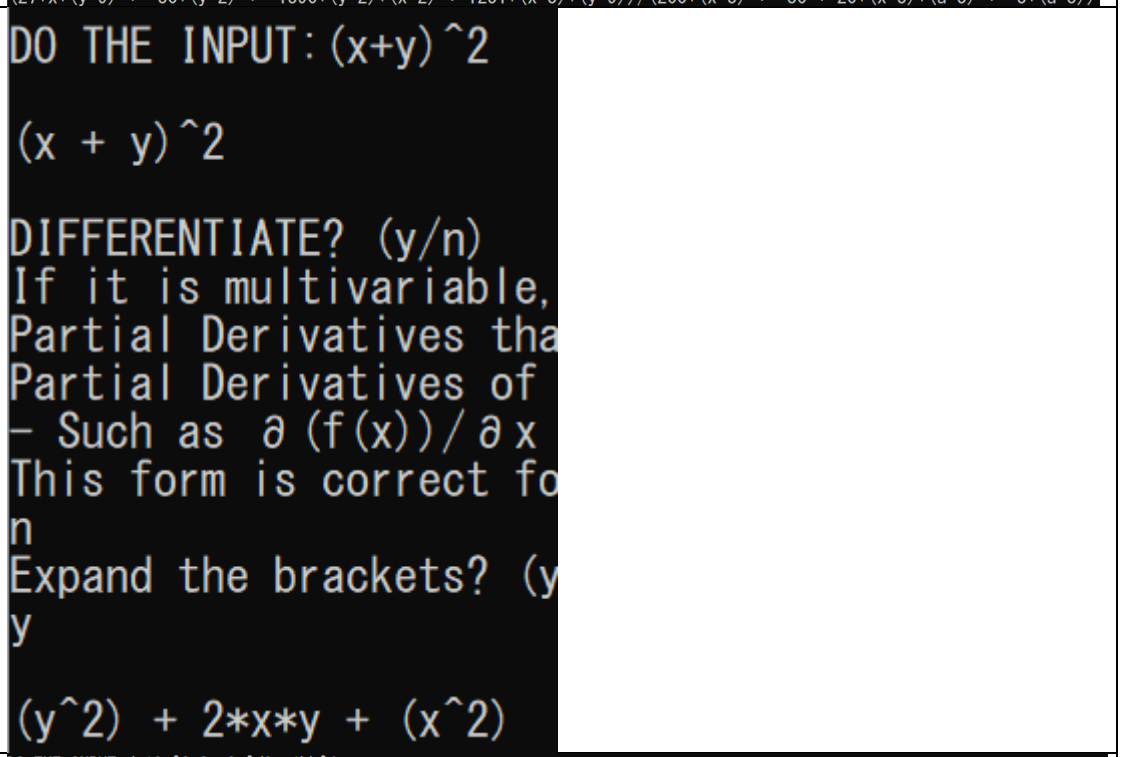
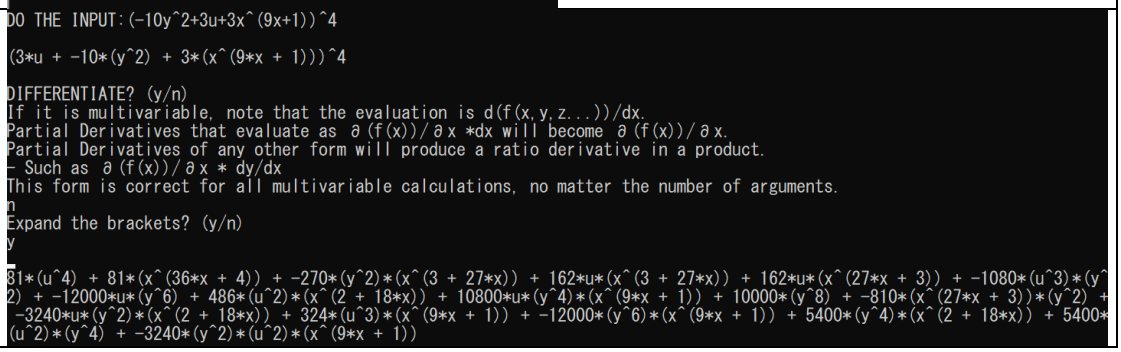
10 b

```
DO THE INPUT:(3*10x^2+9*103x^2-10x^2)*( 3yx^(9x^2)+3x^(9x^2)+10x-10xy^(9x^2)+102x^(9x^2)y)

947*(x^2)*(-10*x*(y^9*(x^2)) + 10*x + 3*(x^9*(x^2)) + 105*y*(x^9*(x^2)))

DIFFERENTIATE? (y/n)
If it is multivariable, note that the evaluation is d(f(x,y,z...))/dx.
Partial Derivatives that evaluate as ∂(f(x))/∂x *dx will become ∂(f(x))/∂x.
Partial Derivatives of any other form will produce a ratio derivative in a product.
- Such as ∂(f(x))/∂x * dy/dx
This form is correct for all multivariable calculations, no matter the number of arguments.
n
Expand the brackets? (y/n)
y

99435*y*(x^(2 + 9*(x^2))) + -9470*(x^3)*(y^9*(x^2)) + 9470*(x^3) + 2841*(x^(2 + 9*(x^2)))
```

11 a

```
DO THE INPUT:(3x+9)*(9x+3)*(10x^2-19)

*(10*(x^2) + -19)*(9*x + 3)*(3*x + 9)

DIFFERENTIATE? (y/n)
If it is multivariable, note that the evaluation is d(
Partial Derivatives that evaluate as ∂(f(x))/∂x *dx
Partial Derivatives of any other form will produce a r
- Such as ∂(f(x))/∂x * dy/dx
This form is correct for all multivariable calculation
n
Expand the brackets? (y/n)
y

270*(x^4) + -1710*x + 900*(x^3) + -243*(x^2) + -513
```

11 b	DO THE INPUT:(3*10x^2+9*103x^2-10x^2)*(3yx^(9x^2)+3x^(9x^2)+10x-10xy^(9x^2)+102x^(9x^2)y)*(9x+3) 947*(x^2)*(9*x + 3)*(-10*x*(y^9*(x^2)) + 10*x + 3*(x^9*(x^2)) + 105*y*(x^9*(x^2))) DIFFERENTIATE? (y/n) If it is multivariable, note that the evaluation is d(f(x,y,z...))/dx. Partial Derivatives that evaluate as ∂(f(x))/∂x *dx will become ∂(f(x))/∂x. Partial Derivatives of any other form will produce a ratio derivative in a product. - Such as ∂(f(x))/∂x * dy/dx This form is correct for all multivariable calculations, no matter the number of arguments. n Expand the brackets? (y/n) y 8523*(x^(2 + 9*(x^2))) + 298305*y*(x^(2 + 9*(x^2))) + -28410*(x^3)*(y^9*(x^2)) + 28410*(x^3) + 25569*(x^(3 + 9*(x^2))) + 894915*y*(x^(3 + 9*(x^2))) + -85230*(x^4)*(y^9*(x^2)) + 85230*(x^4)
12 a	DO THE INPUT:(3x/5)+10/5 (10 + 3*x)/5
12 b	DO THE INPUT:(10 + 3*x)/(9x^2-10)+(9x^2+3)/(9x^ (9*(x^2) + 13 + 3*x)/(9*(x^2) + -10)
13 a	DO THE INPUT:3*(9x/10) 27*x/10
13 b	DO THE INPUT:(3x/5)*(9x/10) 27*(x^2)/50
14 a	DO THE INPUT:((3x+5)/10)* ((3x+5)/10) *(3*x + 5)*(3*x + 5)/100 DIFFERENTIATE? (y/n) If it is multivariable, note that the Partial Derivatives that evaluate as Partial Derivatives of any other form - Such as ∂(f(x))/∂x * dy/dx This form is correct for all multivar n Expand the brackets? (y/n) y (30*x + 25 + 9*(x^2))/100

14 b	
15 a	
15 b	

Prototype 3

Tests

In the 3rd prototype I developed the front end and added the differentiation part for my backend. I will first test this area, and then move onto the front-end functions.

Test	Objective	Purpose of test	Test Data	Expected Result	Actual Result

16		Test that it can differentiate 1 variable simple terms.	Typical Input: 9x^2+3x+3	18x+3	18*x + 3
			Extreme Input: 9x^5-10x^10+123x^5	-100x^9+660x^4	-100*(x^9) + 660*(x^4)
17		Test that it can differentiate 1 variable power brackets and display chain rule.	Typical Input: (3x^2+9)^5	30x(3x^2+9)^4	30*x*((3*(x^2) + 9)^4)
			Extreme Input: (9x^5-10x^10+123x^5)^99	(99*(-10x^10+132x^5)^98)*(-100x^9+660x^4)	99*(-100*(x^9) + 660*(x^4))*((-10*(x^10) + 132*(x^5))^98)
18		Test that it can differentiate 2 term 1 variable products and display chain rule.	Typical Input: uv	du/dxv+udv/dx	(u*dv + v*du)/dx
			Extreme Input: ((x^3+9x^2)^50)*(10x^3+9x^5)^32	50(3 x^2 + 18 x) (10 x^3 + 9 x^5)^32 (x^3 + 9 x^2)^49 + 32(45 x^4 + 30 x^2) (10 x^3 + 9 x^5)^31 (x^3 + 9 x^2)^50	50*(3*(x^2) + 18*x)*(((x^3) + 9*(x^2))^49)*((10*(x^3) + 9*(x^5))^32) + 32*(30*(x^2) + 45*(x^4))*((10*(x^3) + 9*(x^5))^31)*(((x^3) + 9*(x^2))^50)
19	**3**	Test that it can differentiate 1 variable fractions. ***(Although they look different, this uses the quotient rule whilst my calculator applies product rule with negative powers)***	Typical Input: (3x^2+9)/(9x^3-10x)	-(3 (-30 + 91 x^2 + 9 x^4))/(x^2 (10 - 9 x^2)^2)	((-81*(x^4) + -213*(x^2) + 90)/(100*(x^2) + -180*(x^4) + 81*(x^6))) + (6*x/(9*(x^3) + -10*x))
			Extreme Input: ((3x^2+9)*(x^3+3x+9))/(9x^3-10x)	(3 (18 x^7 - 40 x^5 - 81 x^4 - 282 x^3 - 819 x^2 + 270))/(x^2 (10 - 9 x^2)^2)	(-1*(3*x + (x^3) + 9)*(3*(x^2) + 9)*(27*(x^2) + -10)/((9*(x^3) + -10*x)^2)) + ((6*x*(3*x + (x^3) + 9) + *(3 + 3*(x^2))*(3*(x^2) + 9))/(9*(x^3) + -10*x))
20		Test that it can differentiate 2 variable expressions.	Typical Input: 3x+3y^2+10x^2	3+6ydy/dx+20x	(6*y*dy/dx) + 3 + 20*x
			Extreme Input: 9yx^2+3xy-10y+10x^50	18xy+18x^2(dy/dx)+3y+3xdy/dx -10dy/dx +500x^49	18*x*y + 3*y + ((3*x*dy + -10*dy + 9*dy*(x^2))/dx) + 500*(x^49)
21		Test that it can differentiate more than 2	Typical Input: xyz+x+xy	yz+1+y+(xdyz+xydz+xdy)/dx	z*y + 1 + ((x*dy + z*x*dy + y*x*dz)/dx) + y

		variable expressions. ***(The calculators I were using to check could not compute more than 2 variable differentiation. It is rather surprising, so I had to do the inputs as general form)***	Extreme Input: xyz+ucv	zy+(xdyz+xydz+ducv+udcu+ucdv)/dx	z*y + ((z*x*dy + y*x*dz + v*u*dc + c*u*dv + v*c*du)/dx)

After this is the testing for the user interface. In my interface design I went through the general functions of every area, but now I will verify the validity of each system. I will supplant this with a video, so I will not show every single thing that my program can do, but I'll do the important ones.

Test	Objective	Purpose of test	Test Data	Expected Result	Actual Result
22	**1**	Test that it can check the correct login details for student (and thus teacher).	Typical Input: student	Sees that it is the correct password and move the user to the system screen.	It displayed a notification saying "Correct Login Info! Transferring to STUDENT screen" and displayed the tests groupbox.
			Erroneous Input: blah	It will display a notification telling the user they entered the wrong password.	It showed a notification saying "Wrong Login Info!"
23	**5 FT**	The teacher can enter the Question Creation area.	-	When they click the button 'Create Questions' the correct groupbox will be displayed	It displayed the 'Question Control' Groupbox.
24	**5 FT**	The teacher can click 'Add New Question' and view the question template area.	-	When they click the button 'ADD NEW QUESTION' it changes groupbox to 'Question Chooser'	It displayed the 'Question Chooser' Groupbox.

25	**2**	The teacher can select a question template.	-	When they select a template in the listbox, and then click 'Create Question' it should the question editor groupbox.	It displayed the 'Question Creation' Groupbox.
26	**2**	The teacher can add a question to their question list.	-	When the teacher clicks finalise on a question it should return them to their question creation area with the question added to the list.	It added the question to the question list, and displayed the 'Question Control' Groupbox.
27	**2**	The teacher can right click on a selected question and display some options to modify it.	-	When the teacher right clicks the listbox with a selected question, the options to Edit, Delete, and Add a new question should appear.	It displayed an option window with the options: 'Edit Collecting Like Terms' 'Delete Collecting Like Terms', and 'Add a new question'.
28	**5 FT**	The teacher can add a new question through right clicking.	-	When the teacher right clicks and clicks 'Add a new question' the groupbox 'Question Creation' should be viewed.	It displayed the 'Question Creation' Groupbox.
29	**5 FT,4**	The teacher can exit the template groupbox.	-	When the teacher clicks 'Exit' it returns them to the 'Question Control' Groupbox.	It returned the teacher to the 'Question Control' Groupbox.

30	**5 FT**	The teacher can edit the selected question by clicking the 'EDIT' button.	-	When the teacher clicks the 'EDIT' button it will display the 'Question Creation' groupbox with the question's data.	It displayed the 'Question Creation' Groupbox with the question's data.
31	**5 FT**	The teacher can edit a question when there are multiple questions in the list.	-	When the teacher clicks the 'EDIT' button it will display the 'Question Creation' groupbox with the question's data.	It displayed the 'Question Creation' Groupbox with the question's data.
32	**5 FT**	The teacher can change the question data and click recompute to create a new computed answer.	Typical Input: Question = 49x^3-5x^5	The answer should be recomputed correctly.	The correct answer was displayed in the 'Computed Answer' textbox.
33	**5 FT**	Editing questions are saved.	-	When the teacher clicks finalise, edits another question, and then returns back to the previous question, it should display the same text.	The same result was displayed.
34	**5 FT**	The teacher can delete questions.	-	When the teacher clicks 'Remove' with a selected question, it should be removed from the list.	The question were removed from the list.

35	**5 FT**	The teacher can clear all questions.	-	When the teacher clicks ‘Clear All’ all the questions in the list should be removed.	All the questions were removed.
36	**5 FT**	The teacher can view the EXPORT menu.	-	When the teacher clicks ‘Export’ the Groupbox ‘Export Test’ should be displayed.	The Groupbox ‘Export Test’ was displayed.
37	**5 FT**	The teacher can export their test.	Typical Input: Name = “Basic Test” Description = “Testing Test”	When the teacher clicks ‘Export’ the questions should be converted to a JSON file.	A JSON file was created and placed into the teacher’s document path.
38	**6 FS**	The student can add a new test.	Typical Input: Exported test created by teacher.	When the student clicks the ‘Add new test’ they are prompted with an explorer window to select the JSON file. They can then select a file, and when done so the test list will display the test from the data.	The student opened the json file and it was displayed in the test listbox.
39	**6 FS**	The student can right click the test and view its options.	-	When the student right clicks the options of 'View', ‘Delete’, and ‘Add new test’ should show.	The options ‘View Basic Test’, ‘Delete Basic Test’ And ‘Add new Test’ were displayed.

40	**6 FS**	The student can view their selected test.	-	When the student clicks 'View Test' it should display the 'Test Viewer' Groupbox with all the questions in the list.	It displayed the 'Test Viewer' Groupbox with the correct questions listed.
41	**6 FS**	The student can view a selected question.	-	When the student clicks 'Answer Question' on a selected question it will display the 'Question Answerer' Groupbox with the correct question.	It displayed the 'Question Answerer' Groupbox with the correct question.
42	**6 FS**	The student can add an answer to any question, and when they click exit it saves.	-	When the student adds an answer and exits, the question's answer will save.	It saved the question answer.
43	**6 FS**	The student can traverse the questions to the right using the right arrow button.	-	When the student clicks the ">" button, and if there are questions after the current one, it will move up one and display the next question.	It moved to the next question if there are more left.
44	**6 FS**	The student can preview the answer and question on the 'Test Viewer' groupbox.	-	When the student selects a question, the question and answer textbox will update.	They update and display the correct data.

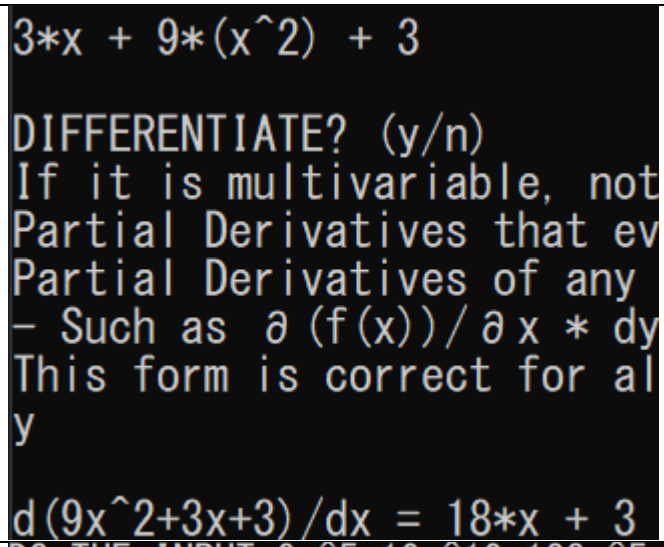
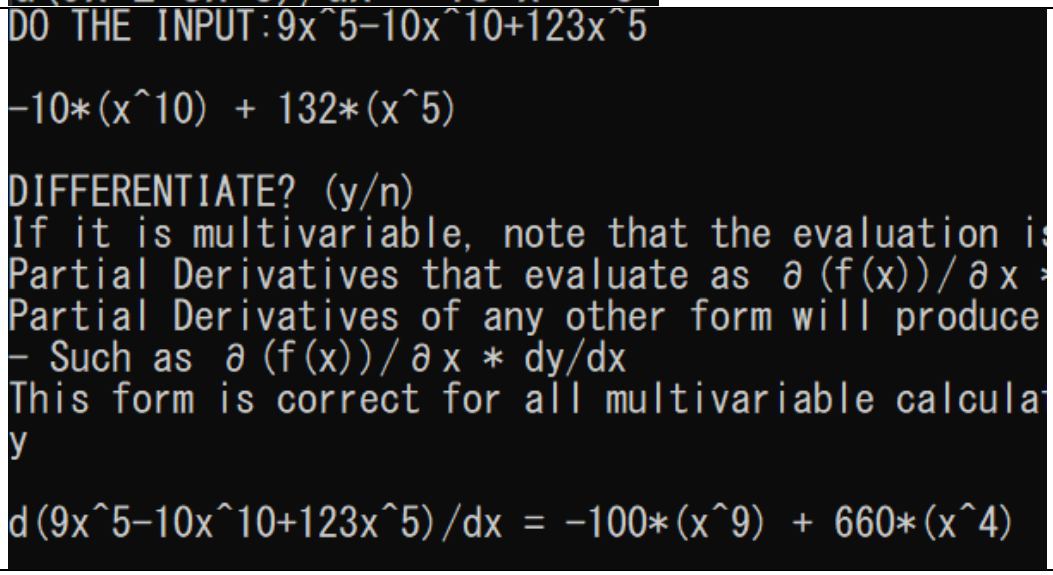
45	**6 FS**	The student can click export and give their name.	-	When the student clicks 'EXPORT' they will move to the 'Export Answers' Groupbox.	The Groupbox 'Export Answers' was displayed.
46	**6 FS**	The student can export their completed test.	Typical Input: Name = "John Joe"	When the student clicks 'Export' on the 'Export Answers' Groupbox, it will create a file in the documents.	The file was created.
47	**6 FS**	The student can remove their selected test.	-	When the student clicks 'REMOVE' the selected test in the listbox is removed.	The test was removed.
48	**6 FS**	The student can re-add their saved test so that they can work on it at different times.	-	When the student clicks 'ADD NEW TEST' and selects their exported test, it should display the saved answers correctly.	The re-added test retained all the data exported.
49	**6 FS**	The student can clear all their tests.	-	When the student clicks 'CLEAR ALL' all the tests are deleted.	All the tests were deleted.
50	**7 FT**	The teacher can view the answer questions area.	-	When the teacher clicks 'Answer Submissions' the 'Submissions' Groupbox is displayed.	The Groupbox 'Submissions' was displayed.

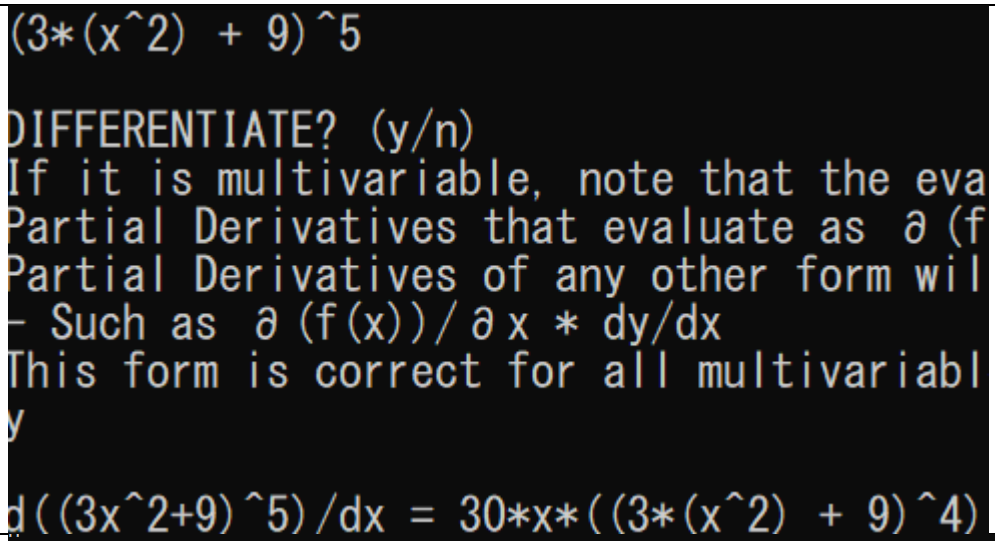
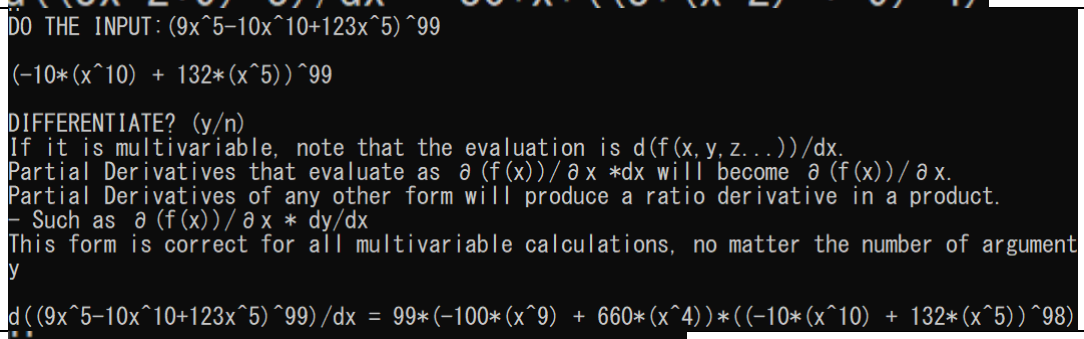
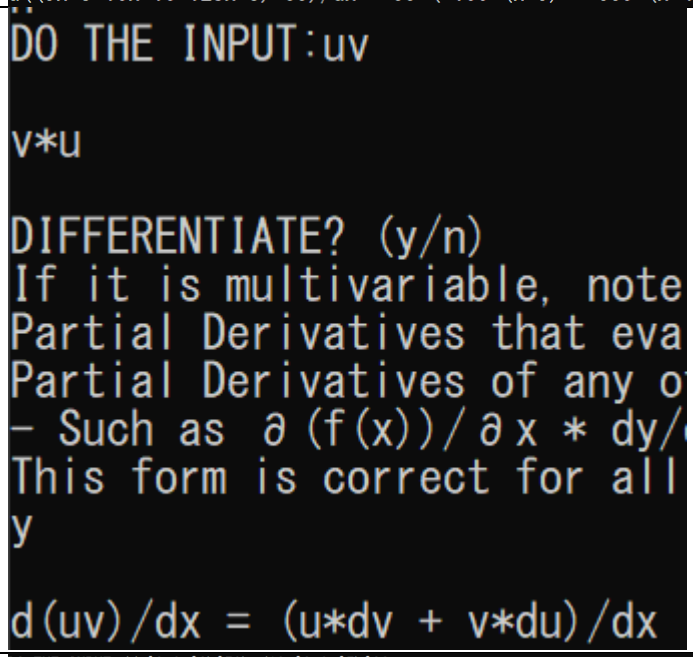
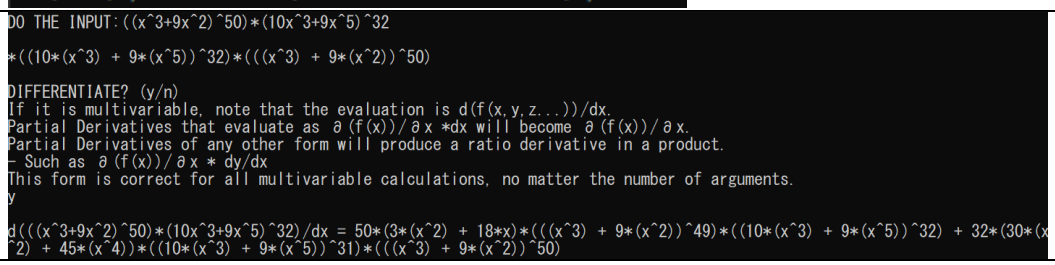
51	**7 FT**	The teacher can add a new submission.	Typical Input: Student Created submission JSON file.	When the teacher clicks 'ADD NEW SUBMISSION' it should display an explorer to select a JSON file. It will then add a new item into the listbox.	Selecting a JSON file added a new submission to the listbox.
52	**7 FT**	The teacher can right click the listbox with a submission selected to display options.	-	When the teacher right clicks with a submission selected it should display the options: 'View', 'Delete' and 'Add new Submission'	The options 'View Basic Test | John Joe', 'Delete Basic Test | John Joe' and 'Add new submission' Were displayed.
53	**7 FT**	The teacher can view their selected submission.	-	When the teacher clicks the button 'VIEW SUBMISSION' it should display the 'View Submission' groupbox and populate the question list with the correct data.	It displayed the 'View Submission' Groupbox with the correct data.
54	**7 FT**	The teacher can preview the question and answer of the selected question.	-	When the teacher selects a question, the text boxes for Question and Answer are updated.	The textboxes were updated to the correct data.
55	**7 FT**	The teacher can select to view one of the automated marked questions.	-	When the teacher clicks 'Override Marking' with a question selected it should display the 'Mark Question' GroupBox with	It displayed the 'Mark Question' Groupbox and had updated the correct data for the text boxes.

				the correct data.	
56	**7 FT**	The teacher can override the marking for a question and mark it correct.	-	When the teacher clicks ‘Mark Correct’ it should change the Deemed Validity as ‘Marked Right’ and update the reason.	It updated the reason and validity.
57	**7 FT**	The system can mark correct differentiation answers by itself.	-	A differentiation question with the correct answer should be automatically deemed correct.	Question 6 was deemed correct without the teacher needing to mark it.
58	**7 FT**	The system can mark simplification questions as ‘Indetermined’ if they procure the same value as the computed answer.	-	A simplification question with an answer that has the same value as its computed answer should be marked ‘Indetermined.	Question 2 was deemed indetermined without the teacher needing to mark it.
59	**7 FT**	The teacher can mark any question as incorrect.	-	When the teacher clicks ‘Mark Incorrect’ it should change the Deemed Validity to ‘Marked Wrong’ and update the reason.	It updated the reason and validity.
60	**7 FT**	The teacher can export the marked test.	-	When the teacher clicks ‘EXPORT’ an explorer should open with a JSON file created and selected.	A JSON was created and shown in the explorer.

61	**7 FT**	The teacher can remove a selected test.		When the teacher clicks 'REMOVE' and selected a test it should be deleted from the listbox.	The selected test was deleted from the listbox.
62	**7 FT**	The teacher is able to re-add their marked test.		When the teacher re-adds their marked test, it should be displayed and contain the correct data.	The marked test retained the data and displayed properly.
63	**8 FT(!)**	The student is able to view their markings.		When the student adds a marked JSON file, and then view the questions, the correct and wrong questions are shown.	Either a cross or a tick are shown beside each question.

Test Evidence:

Test	Evidence
16a	
16b	

17 a	
17 b	
18 a	
18 b	

19 a

```
DO THE INPUT:(3x^2+9)/(9x^3-10x)

(3*(x^2) + 9)/(9*(x^3) + -10*x)

DIFFERENTIATE? (y/n)
If it is multivariable, note that the evaluation is d(f(x,y,z...))/dx.
Partial Derivatives that evaluate as ∂(f(x))/∂x *dx will become ∂(f(x))/∂x.
Partial Derivatives of any other form will produce a ratio derivative in a product.
- Such as ∂(f(x))/∂x * dy/dx
This form is correct for all multivariable calculations, no matter the number of arguments.
y

d((3x^2+9)/(9x^3-10x))/dx = (-1*(3*(x^2) + 9)*(27*(x^2) + -10)/((9*(x^3) + -10*x)^2)) + (6*x/(9*(x^3) + -10*x))

Expand the brackets? (y/n)
y

((-81*(x^4) + -213*(x^2) + 90)/(100*(x^2) + -180*(x^4) + 81*(x^6))) + (6*x/(9*(x^3) + -10*x))
```

19 b

```
DO THE INPUT:((3x^2+9)*(x^3+3x+9))/(9x^3-10x)

*(3*x + (x^3) + 9)*(3*(x^2) + 9)/(9*(x^3) + -10*x)

DIFFERENTIATE? (y/n)
If it is multivariable, note that the evaluation is d(f(x,y,z...))/dx.
Partial Derivatives that evaluate as ∂(f(x))/∂x *dx will become ∂(f(x))/∂x.
Partial Derivatives of any other form will produce a ratio derivative in a product.
- Such as ∂(f(x))/∂x * dy/dx
This form is correct for all multivariable calculations, no matter the number of arguments.
y

d(((3x^2+9)*(x^3+3x+9))/(9x^3-10x))/dx = (-1*(3*x + (x^3) + 9)*(3*(x^2) + 9)*(27*(x^2) + -10)/((9*(x^3) + -10*x)^2)) + (
(6*x*(3*x + (x^3) + 9) + *(3 + 3*(x^2))*(3*(x^2) + 9))/(9*(x^3) + -10*x))

Expand the brackets? (y/n)
y

((-729*(x^4) + -81*(x^7) + -456*(x^5) + -549*(x^3) + 270*x + -1917*(x^2) + 810)/(100*(x^2) + -180*(x^4) + 81*(x^6))) + (
(54*x + 54*(x^2) + 15*(x^4) + 27)/(9*(x^3) + -10*x))
```

20 a

```
DO THE INPUT:3x+3y^2+10x^2

3*(y^2) + 3*x + 10*(x^2)

DIFFERENTIATE? (y/n)
If it is multivariable, note that the evaluatic
Partial Derivatives that evaluate as ∂(f(x))/
Partial Derivatives of any other form will proc
- Such as ∂(f(x))/∂x * dy/dx
This form is correct for all multivariable calc
y

d(3x+3y^2+10x^2)/dx = (6*y*dy/dx) + 3 + 20*x
```

20 b

```
DO THE INPUT:9yx^2+3xy-10y+10x^50

-10*y + 3*y*x + 9*y*(x^2) + 10*(x^50)

DIFFERENTIATE? (y/n)
If it is multivariable, note that the evaluation is d(f(x,y,z...))/dx.
Partial Derivatives that evaluate as ∂(f(x))/∂x *dx will become ∂(f(x))/∂x.
Partial Derivatives of any other form will produce a ratio derivative in a product.
- Such as ∂(f(x))/∂x * dy/dx
This form is correct for all multivariable calculations, no matter the number of arguments.
y

d(9yx^2+3xy-10y+10x^50)/dx = 18*x*y + 3*y + ((3*x*dy + -10*dy + 9*dy*(x^2))/dx) + 500*(x^49)
```

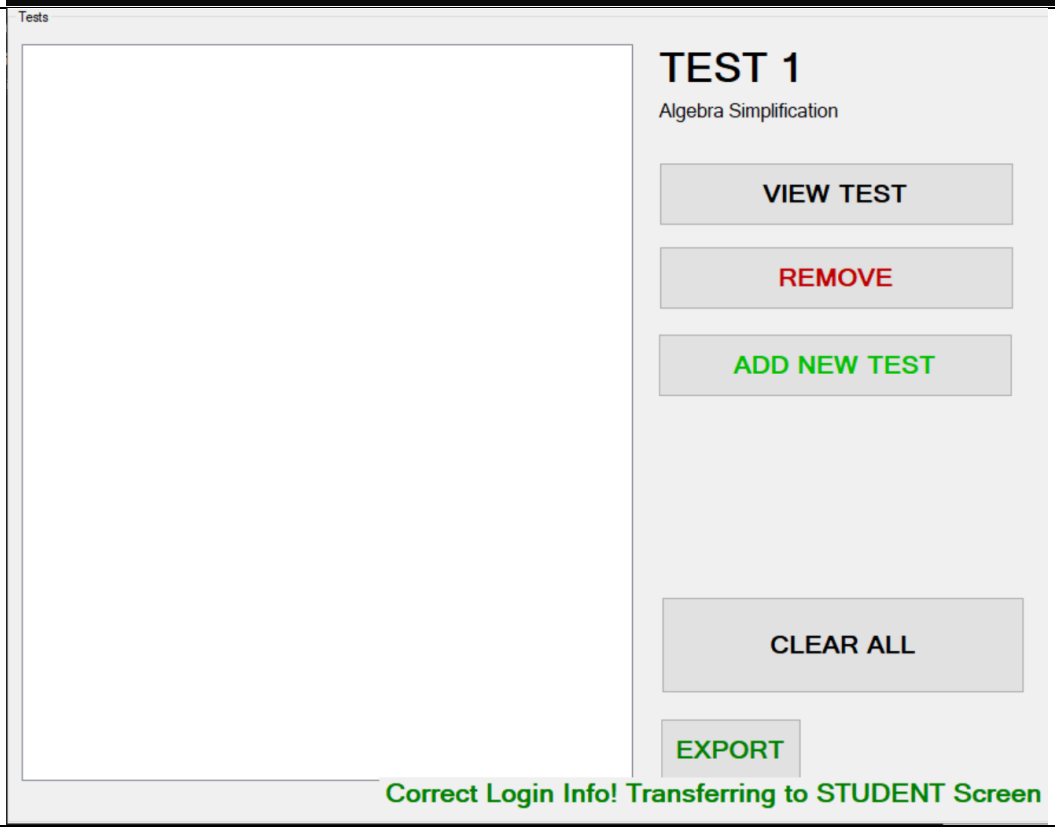
21 a

```
DO THE INPUT:xyz+x+xy

z*y*x + x + y*x

DIFFERENTIATE? (y/n)
If it is multivariable, note that the evaluation is d(f(x,y,z
Partial Derivatives that evaluate as ∂(f(x))/∂x *dx will be
Partial Derivatives of any other form will produce a ratio de
- Such as ∂(f(x))/∂x * dy/dx
This form is correct for all multivariable calculations, no m
y

d(xyz+x+xy)/dx = z*y + 1 + ((x*dy + z*x*dy + y*x*dz)/dx) + y
```

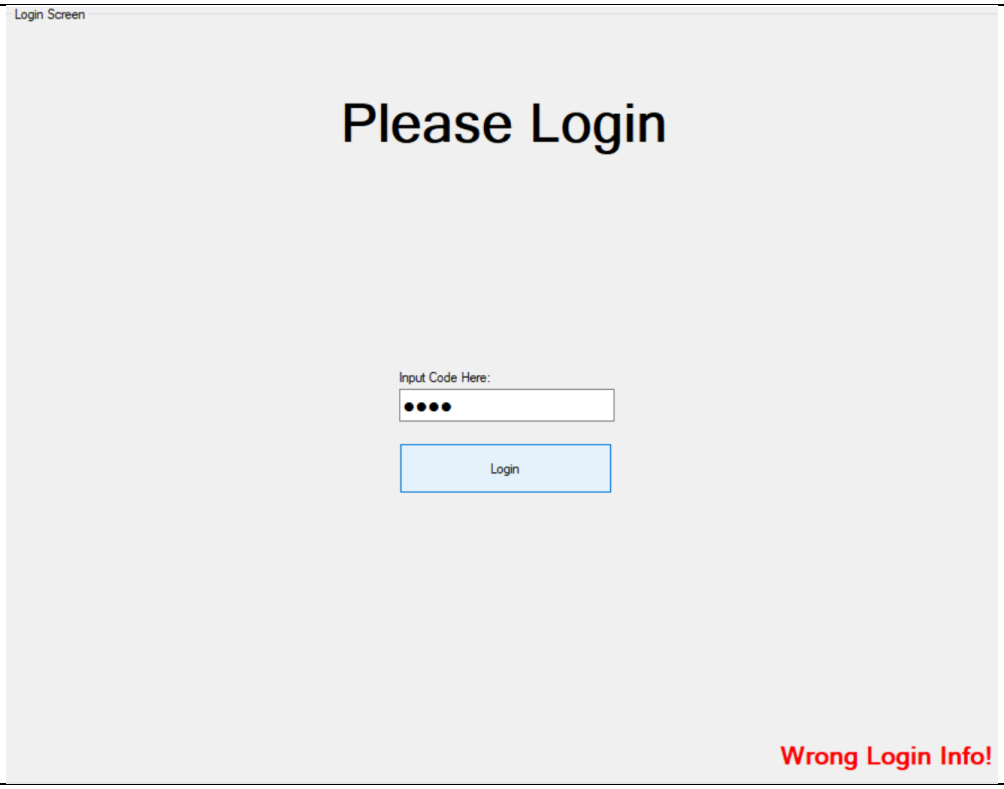
21 b

```
DO THE INPUT:xyz+ucv

z*y*x + v*c*u

DIFFERENTIATE? (y/n)
If it is multivariable, note that the evaluation is d(f(x,y,z...))/dx.
Partial Derivatives that evaluate as ∂(f(x))/∂x *dx will become ∂(f(x)
Partial Derivatives of any other form will produce a ratio derivative in
- Such as ∂(f(x))/∂x * dy/dx
This form is correct for all multivariable calculations, no matter the nu
y

d(xyz+ucv)/dx = z*y + ((z*x*dy + y*x*dz + v*u*dc + c*u*dv + v*c*du)/dx)
```

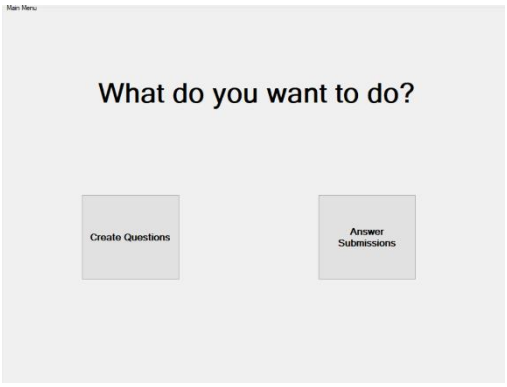
22 a

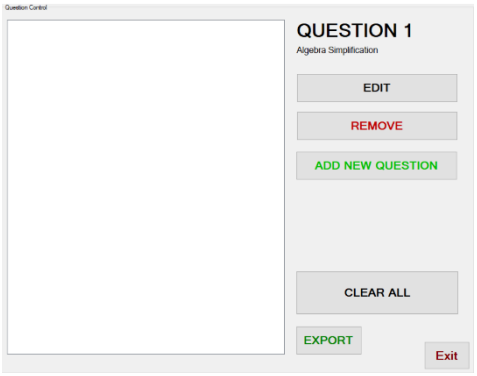


22 b

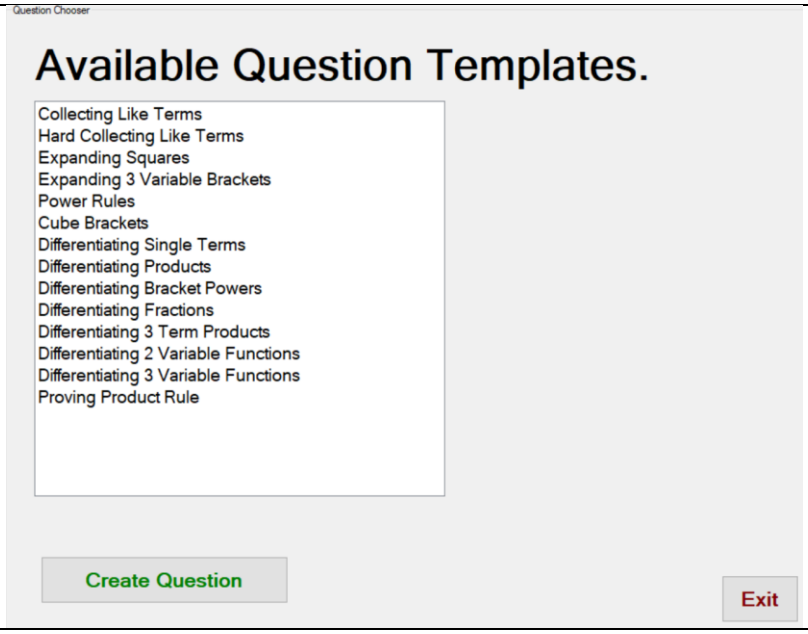

Login Screen
Please Login
Input Code Here:
Login
Wrong Login Info!

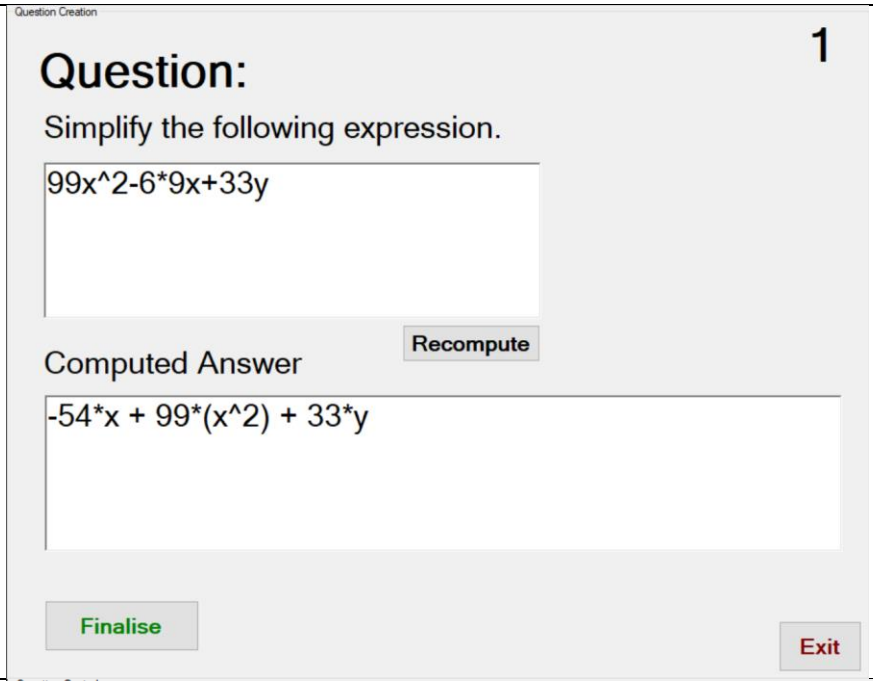
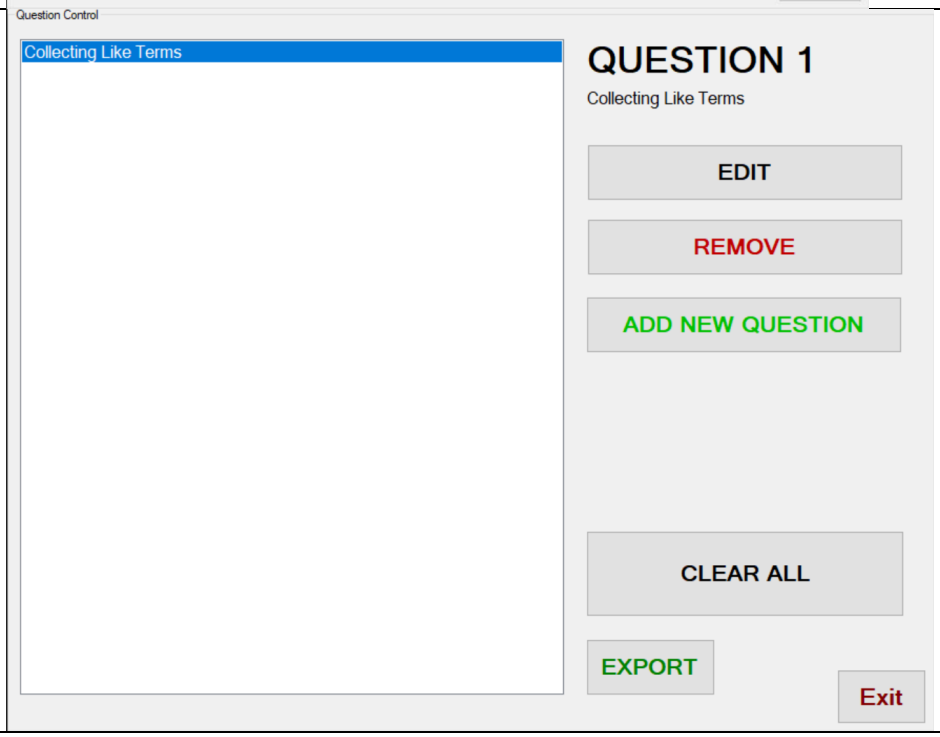
23


What do you want to do?
Create Questions
Answer Submissions

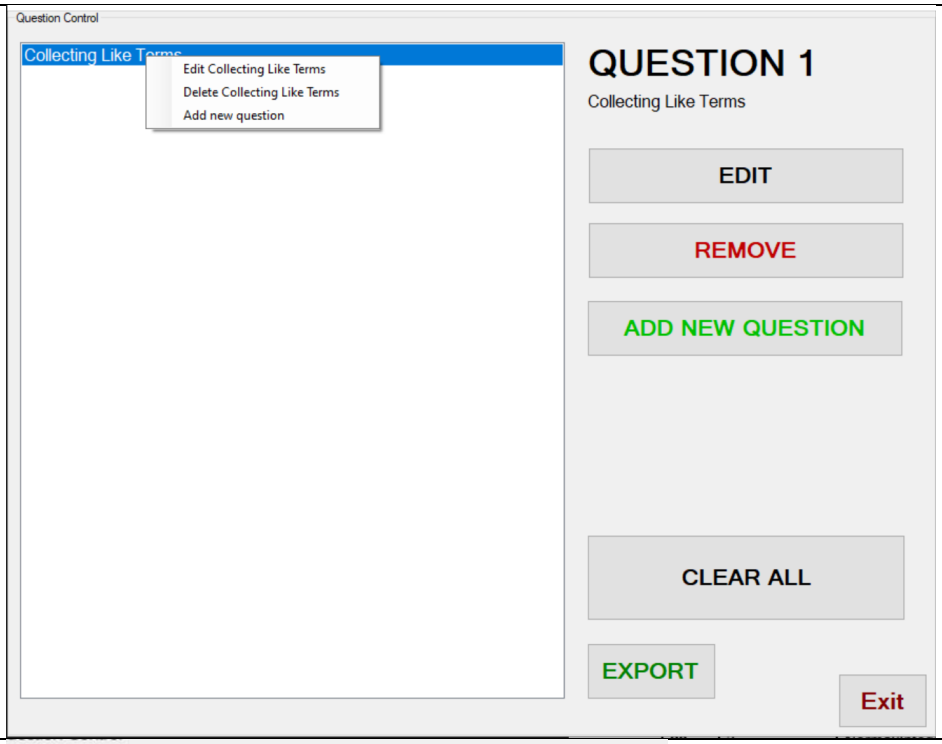

QUESTION 1
Algebra Simplification
EDIT
REMOVE
ADD NEW QUESTION
CLEAR ALL
EXPORT
Exit

24

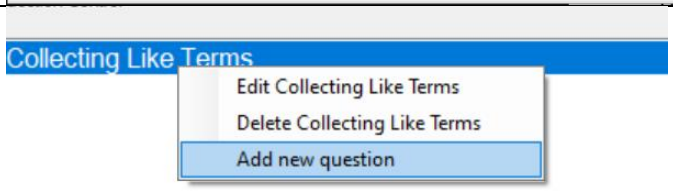

Question Chooser
Available Question Templates.
Collecting Like Terms
Hard Collecting Like Terms
Expanding Squares
Expanding 3 Variable Brackets
Power Rules
Cube Brackets
Differentiating Single Terms
Differentiating Products
Differentiating Bracket Powers
Differentiating Fractions
Differentiating 3 Term Products
Differentiating 2 Variable Functions
Differentiating 3 Variable Functions
Proving Product Rule
Create Question
Exit

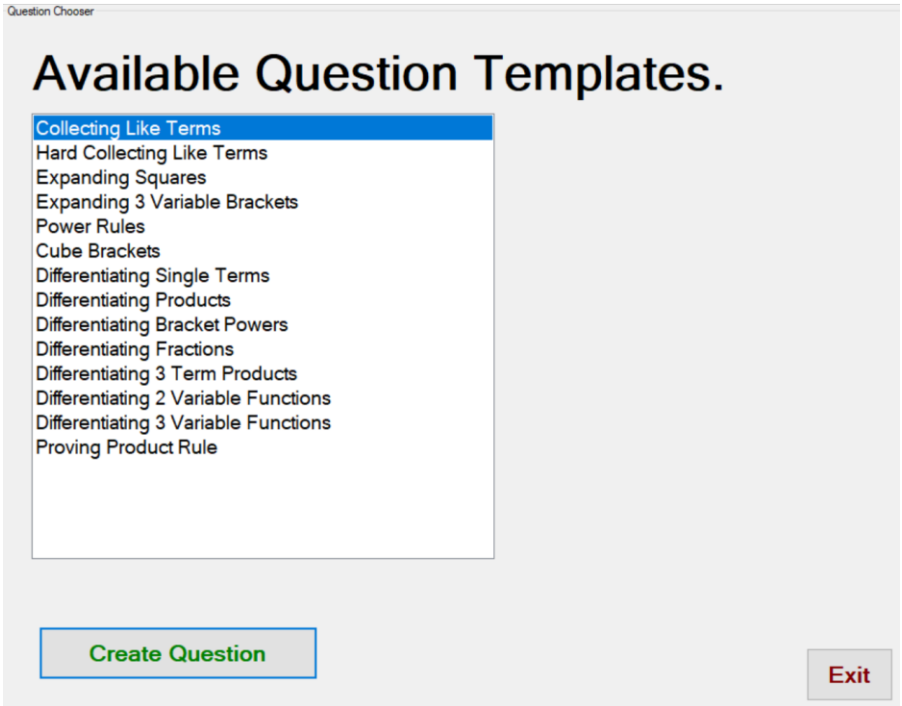
25	 Question Creation 1 Question: Simplify the following expression. 99x^2-6*9x+33y Recompute Computed Answer -54*x + 99*(x^2) + 33*y Finalise Exit
26	 Question Control Collecting Like Terms QUESTION 1 Collecting Like Terms EDIT REMOVE ADD NEW QUESTION CLEAR ALL EXPORT Exit

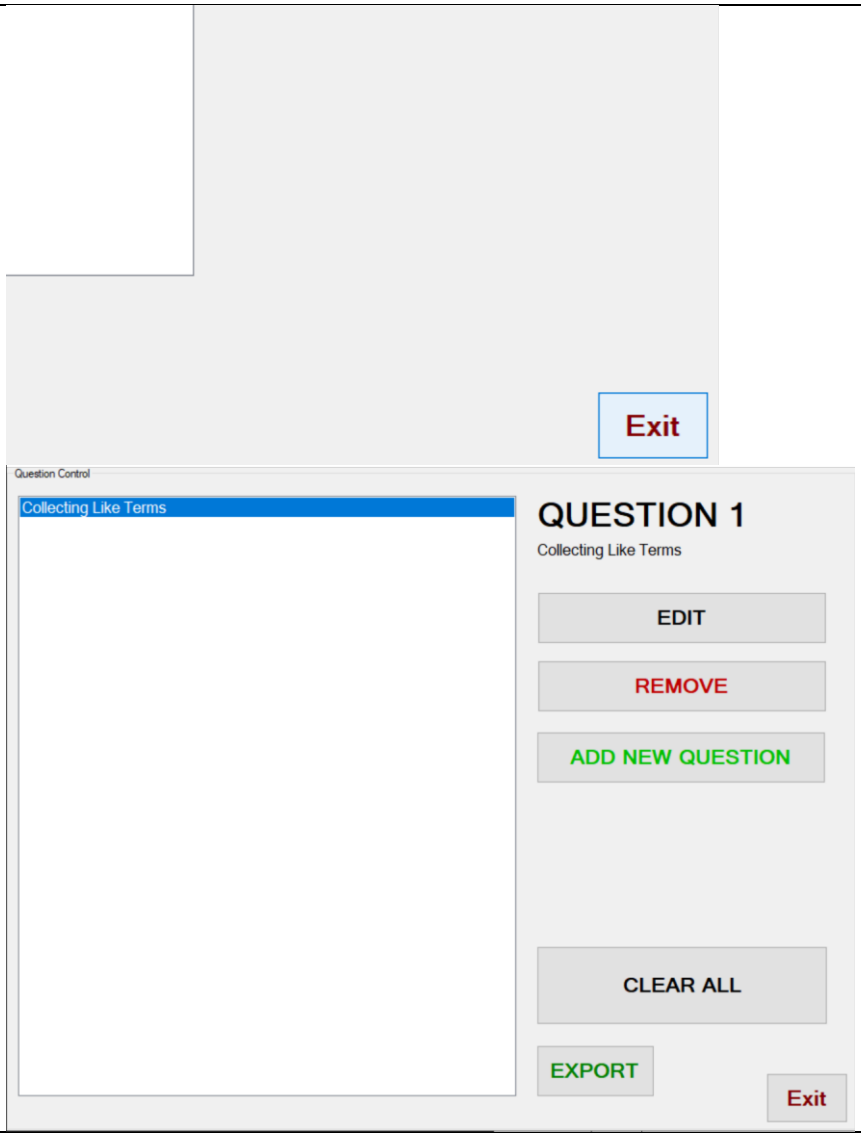
27


Question Control
Collecting Like Terms
Edit Collecting Like Terms
Delete Collecting Like Terms
Add new question
QUESTION 1
Collecting Like Terms
EDIT
REMOVE
ADD NEW QUESTION
CLEAR ALL
EXPORT
Exit

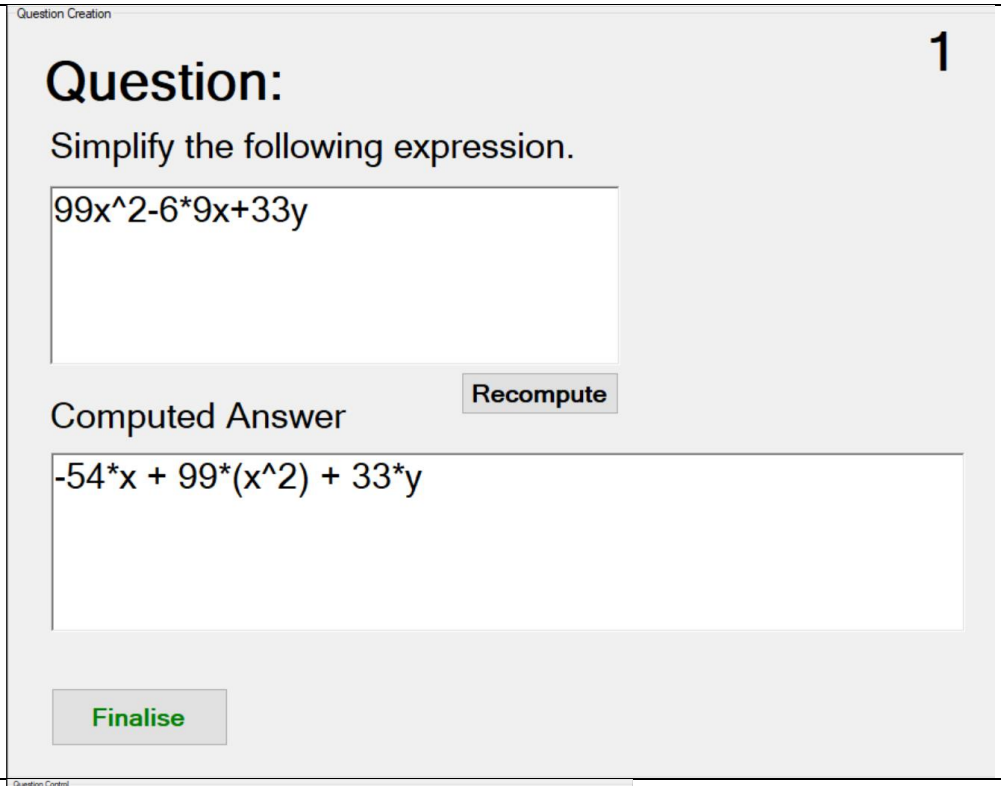
28


Collecting Like Terms
Edit Collecting Like Terms
Delete Collecting Like Terms
Add new question

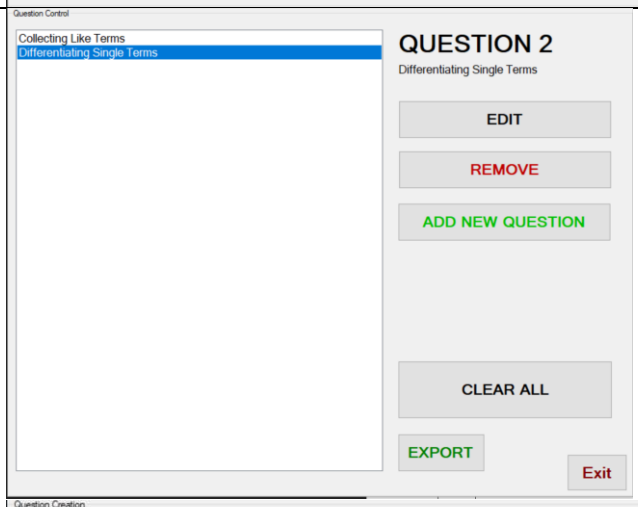

Question Chooser
Available Question Templates.
Collecting Like Terms
Hard Collecting Like Terms
Expanding Squares
Expanding 3 Variable Brackets
Power Rules
Cube Brackets
Differentiating Single Terms
Differentiating Products
Differentiating Bracket Powers
Differentiating Fractions
Differentiating 3 Term Products
Differentiating 2 Variable Functions
Differentiating 3 Variable Functions
Proving Product Rule
Create Question
Exit

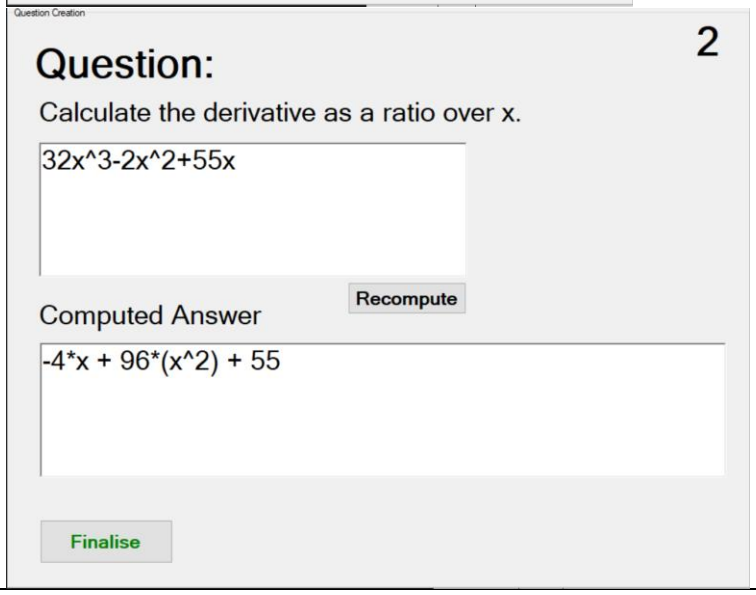
29	 Exit Question Control Collecting Like Terms QUESTION 1 Collecting Like Terms EDIT REMOVE ADD NEW QUESTION CLEAR ALL EXPORT Exit

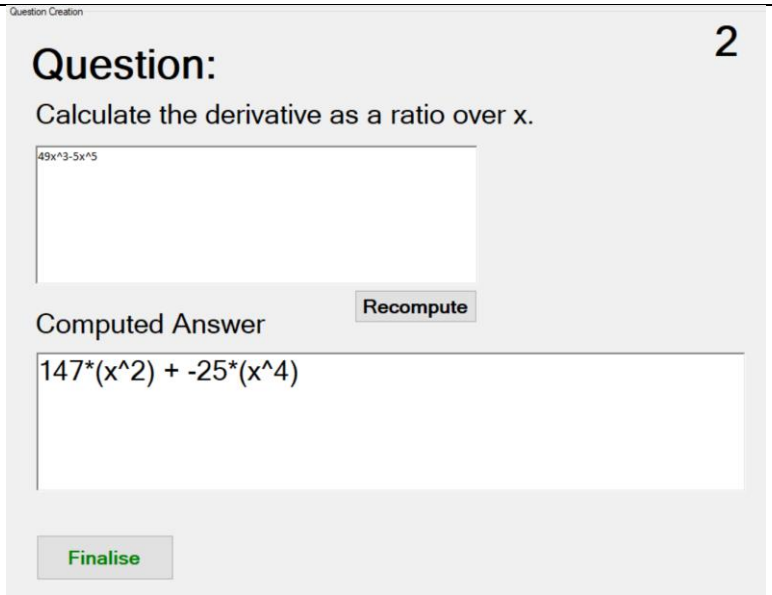
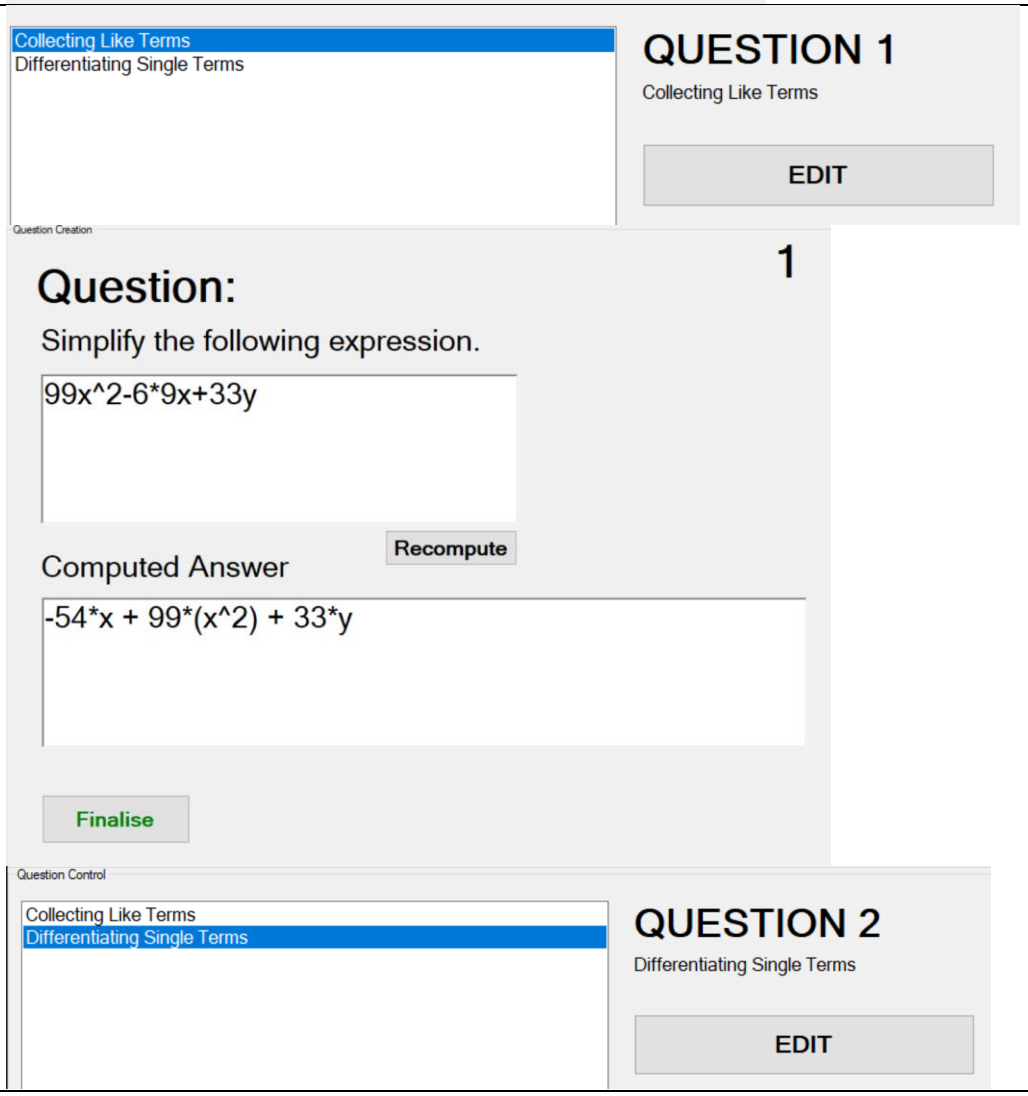
30

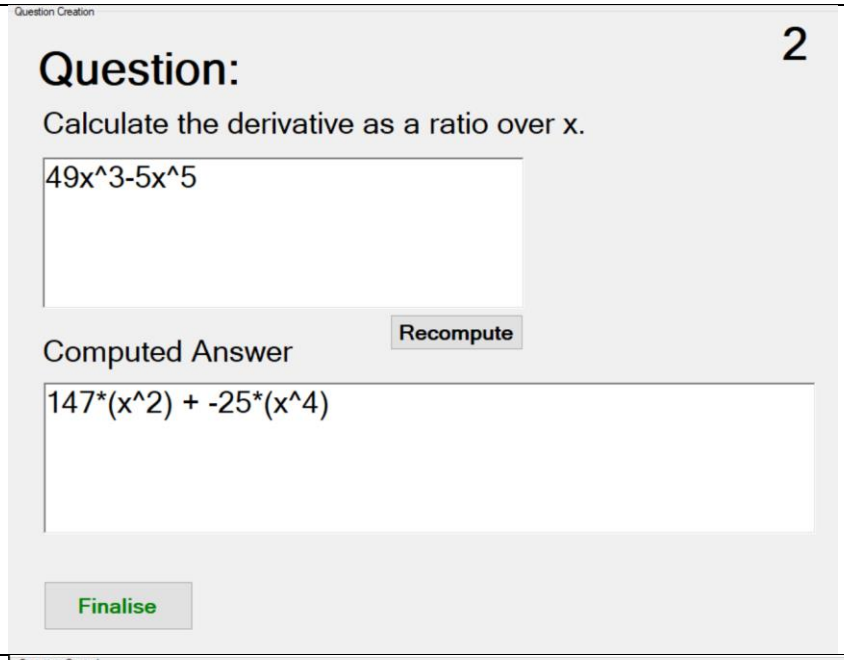

Question Creation
1
Question:
Simplify the following expression.
99x^2-6*9x+33y
Recompute
Computed Answer
-54*x + 99*(x^2) + 33*y
Finalise

31

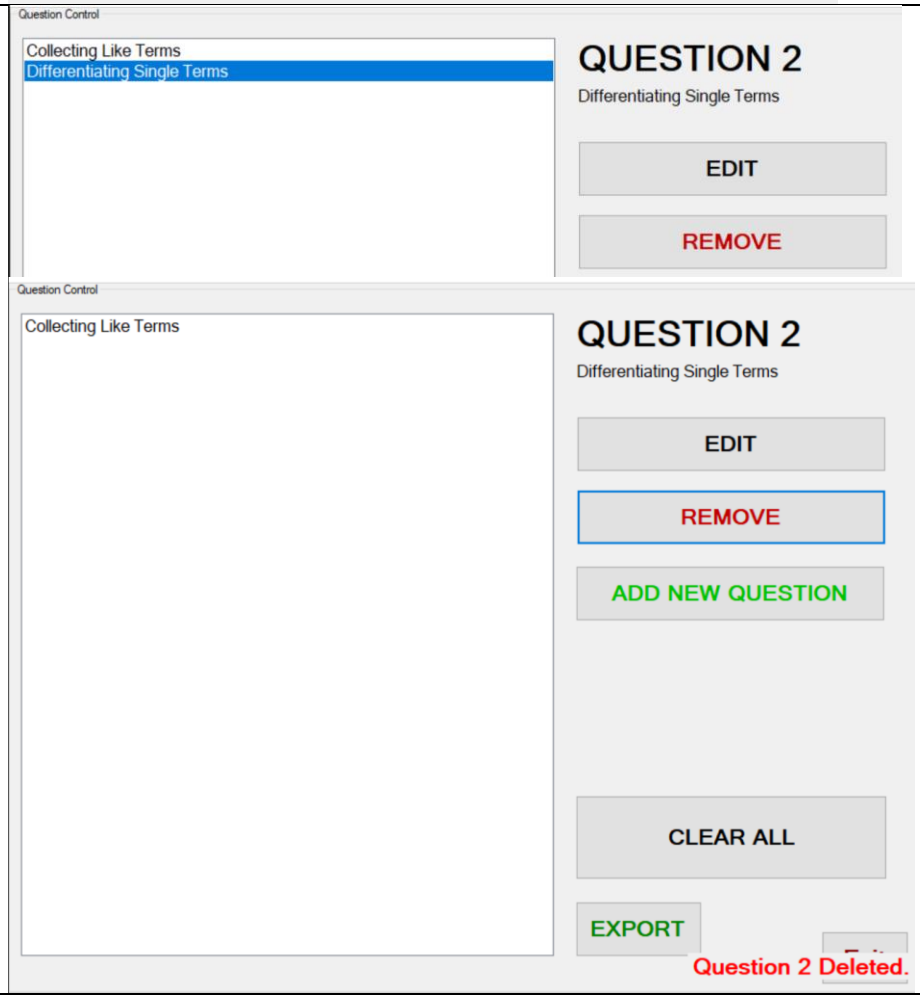

Question Control
Collecting Like Terms
Differentiating Single Terms
QUESTION 2
Differentiating Single Terms
EDIT
REMOVE
ADD NEW QUESTION
CLEAR ALL
EXPORT
Exit


Question Creation
2
Question:
Calculate the derivative as a ratio over x.
32x^3-2x^2+55x
Recompute
Computed Answer
-4*x + 96*(x^2) + 55
Finalise

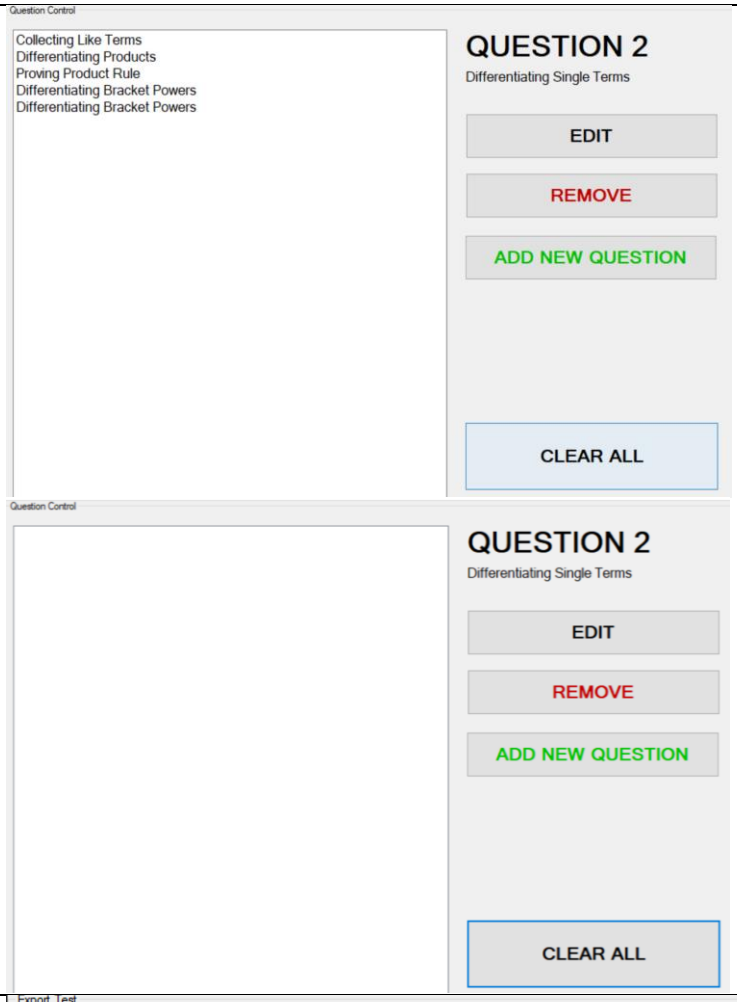
32	 Question Creation 2 Question: Calculate the derivative as a ratio over x. 49x^3-5x^5 Recompute Computed Answer 147*(x^2) + -25*(x^4) Finalise
33	 Collecting Like Terms Differentiating Single Terms QUESTION 1 Collecting Like Terms EDIT Question Creation 1 Question: Simplify the following expression. 99x^2-6*9x+33y Recompute Computed Answer -54*x + 99*(x^2) + 33*y Finalise Question Control Collecting Like Terms Differentiating Single Terms QUESTION 2 Differentiating Single Terms EDIT


Question Creation
2
Question:
Calculate the derivative as a ratio over x.
49x^3-5x^5
Recompute
Computed Answer
147*(x^2) + -25*(x^4)
Finalise

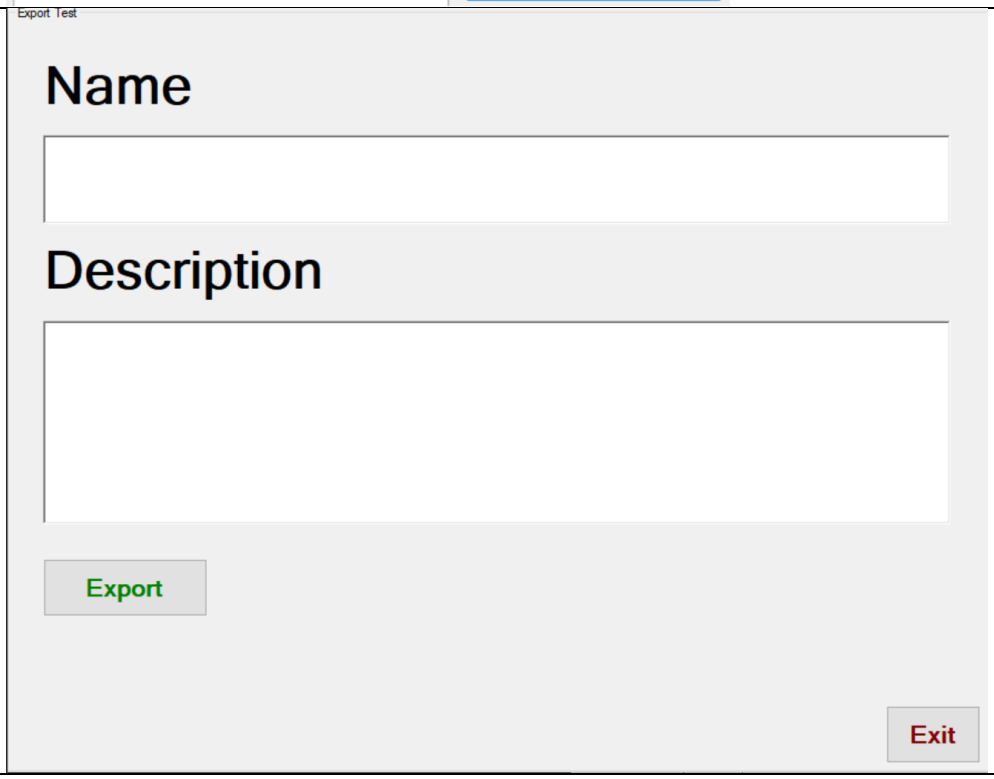
34


Question Control
Collecting Like Terms
Differentiating Single Terms
QUESTION 2
Differentiating Single Terms
EDIT
REMOVE
Question Control
Collecting Like Terms
QUESTION 2
Differentiating Single Terms
EDIT
REMOVE
ADD NEW QUESTION
CLEAR ALL
EXPORT
Question 2 Deleted.

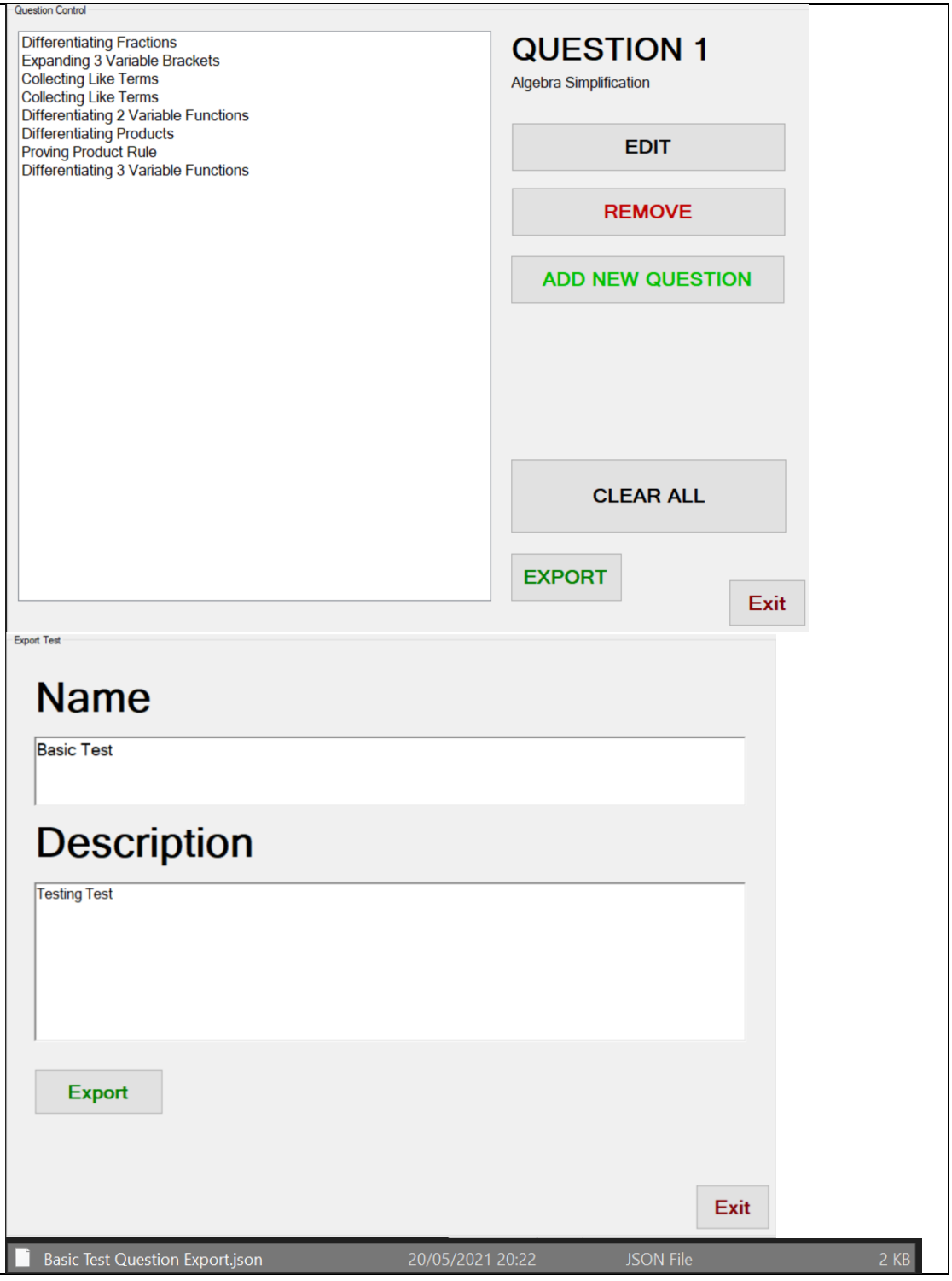
35


Question Control
Collecting Like Terms
Differentiating Products
Proving Product Rule
Differentiating Bracket Powers
Differentiating Bracket Powers
QUESTION 2
Differentiating Single Terms
EDIT
REMOVE
ADD NEW QUESTION
CLEAR ALL
Question Control
QUESTION 2
Differentiating Single Terms
EDIT
REMOVE
ADD NEW QUESTION
CLEAR ALL

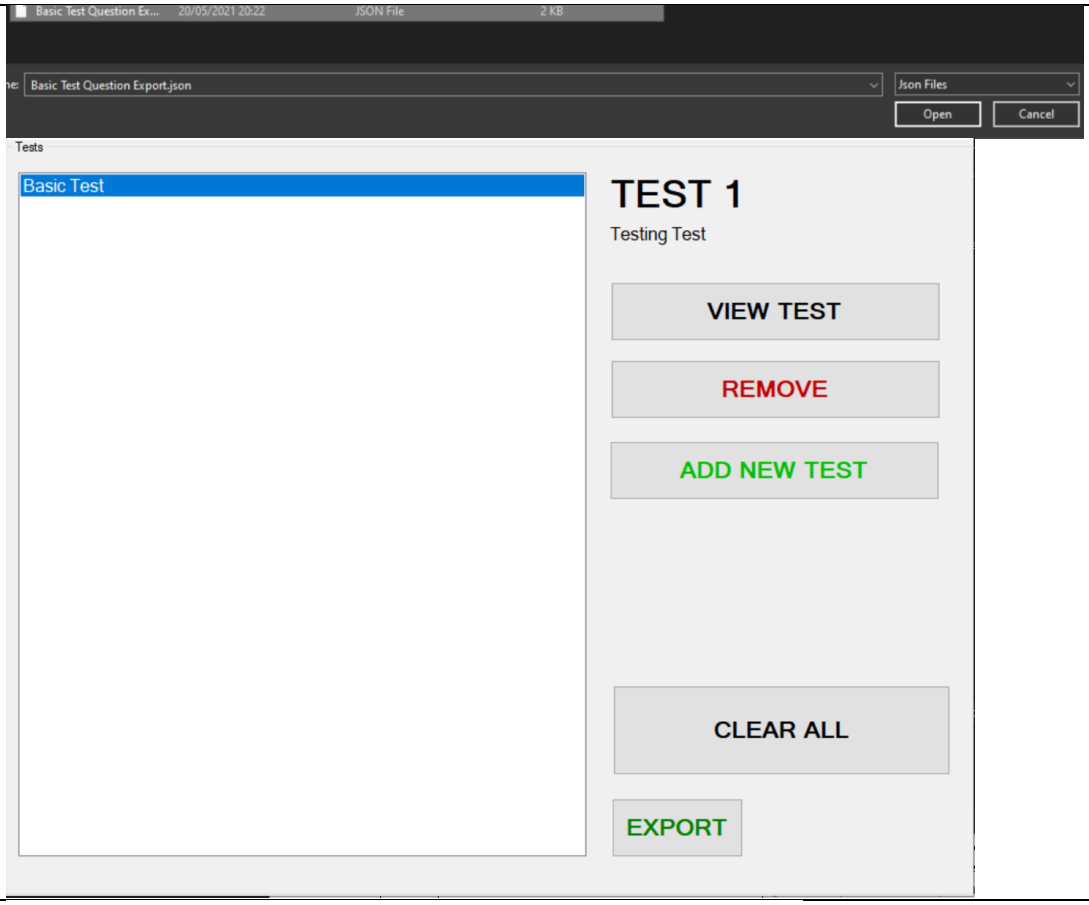
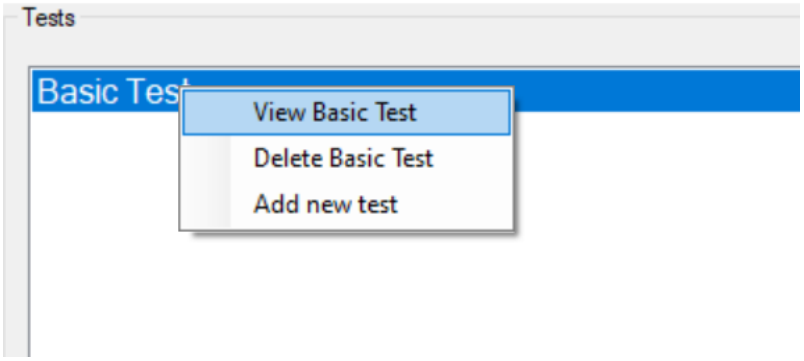
36


Export Test
Name
Description
Export
Exit

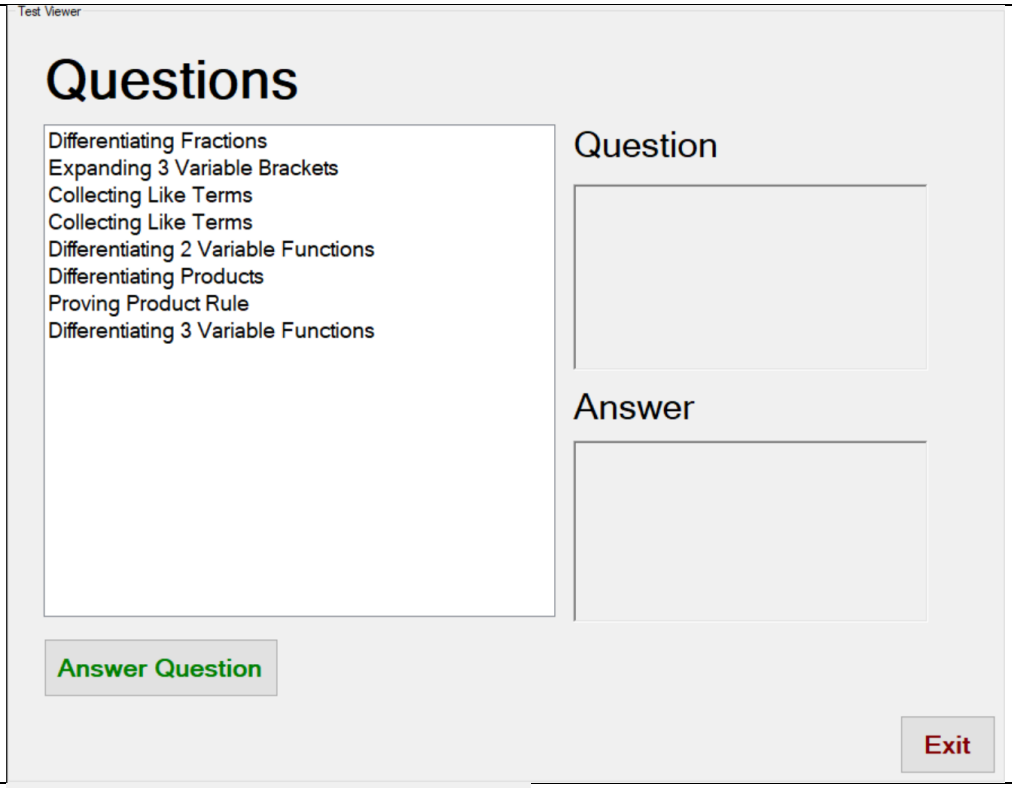
37


Question Control
Differentiating Fractions
Expanding 3 Variable Brackets
Collecting Like Terms
Collecting Like Terms
Differentiating 2 Variable Functions
Differentiating Products
Proving Product Rule
Differentiating 3 Variable Functions
QUESTION 1
Algebra Simplification
EDIT
REMOVE
ADD NEW QUESTION
CLEAR ALL
EXPORT
Exit
Export Test
Name
Basic Test
Description
Testing Test
Export
Exit
Basic Test Question Export.json
20/05/2021 20:22
JSON File
2 KB

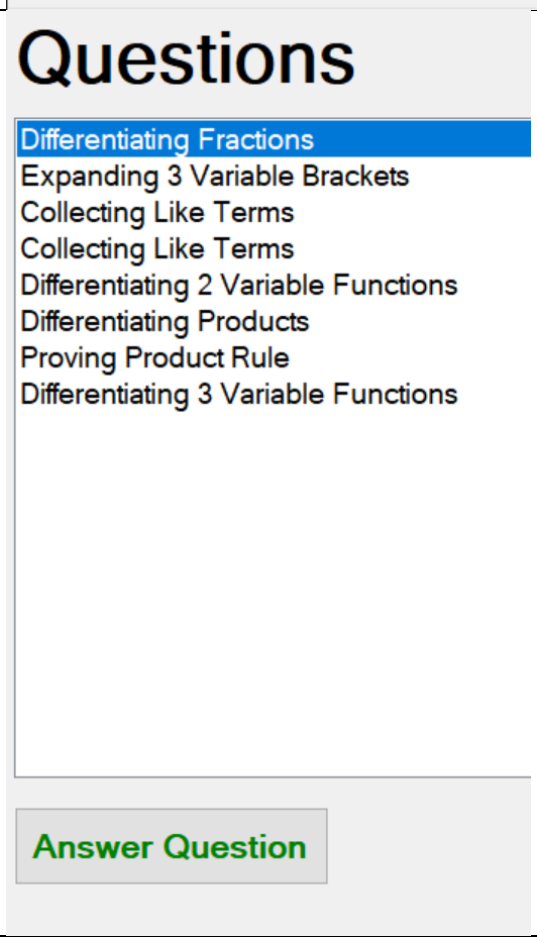
```
▼ array [9]
  ▼ 0 {4}
      NAME : Basic Test
      DESCRIPTION : Testing Test
      MARKED : False
      STUDENT NAME : John Joe
  ▼ 1 {7}
      QUESTION : (32x^2+1)/((5x-2)^2)
      QUESTION TITLE : Calculate the derivative as a ratio over x.
      QUESTION TYPE : DIFFERENTIATION
      ANSWER : Test
      TYPE : Differentiating Fractions
      TEACHER EDITED : False
      STATUS : CORRECT
  ▼ 2 {7}
      QUESTION : (6x^2-y+10z)^2
      QUESTION TITLE : Simplify the following expression.
      QUESTION TYPE : SIMPLIFICATION
      ANSWER : -12x^2y+120x^2z+36x^4+y^2-20yz+100z^2
      TYPE : Expanding 3 Variable Brackets
      TEACHER EDITED : False
      STATUS : CORRECT
  ▼ 3 {7}
      QUESTION : 3x+9z-10x^2+9y+10x-2y
      QUESTION TITLE : Simplify the following expression.
      QUESTION TYPE : SIMPLIFICATION
      ANSWER : 9x^2
      TYPE : Collecting Like Terms
      TEACHER EDITED : False
      STATUS : CORRECT
  ▼ 4 {7}
      QUESTION : 99x^2-6*9x+33y
      QUESTION TITLE : Simplify the following expression.
      QUESTION TYPE : SIMPLIFICATION
      ANSWER : 12
      TYPE : Collecting Like Terms
      TEACHER EDITED : False
      STATUS : CORRECT
  ▼ 5 {7}
      QUESTION : (2y+3x)^2
      QUESTION TITLE : Calculate the derivative as a ratio over x.
      QUESTION TYPE : DIFFERENTIATION
      ANSWER : 2x
      TYPE : Differentiating 2 Variable Functions
      TEACHER EDITED : False
      STATUS : CORRECT
  ▼ 6 {7}
      QUESTION : ((5x-10)^36)*(5x+10)
      QUESTION TITLE : Calculate the derivative as a ratio over x.
      QUESTION TYPE : DIFFERENTIATION
      ANSWER : 5*(5x-10)^36+180*(5x+10)*(5x-10)^35
      TYPE : Differentiating Products
      TEACHER EDITED : False
      STATUS : CORRECT
  ▼ 7 {7}
      QUESTION : uv
      QUESTION TITLE : Calculate the derivative as a ratio over x.
      QUESTION TYPE : DIFFERENTIATION
      ANSWER : uc
      TYPE : Proving Product Rule
      TEACHER EDITED : False
      STATUS : CORRECT
  ▼ 8 {7}
      QUESTION : (5y-7x-3yu)*((13y^2+3)^3)
      QUESTION TITLE : Calculate the derivative as a ratio over x.
      QUESTION TYPE : DIFFERENTIATION
      ANSWER : a
      TYPE : Differentiating 3 Variable Functions
      TEACHER EDITED : False
      STATUS : CORRECT
```

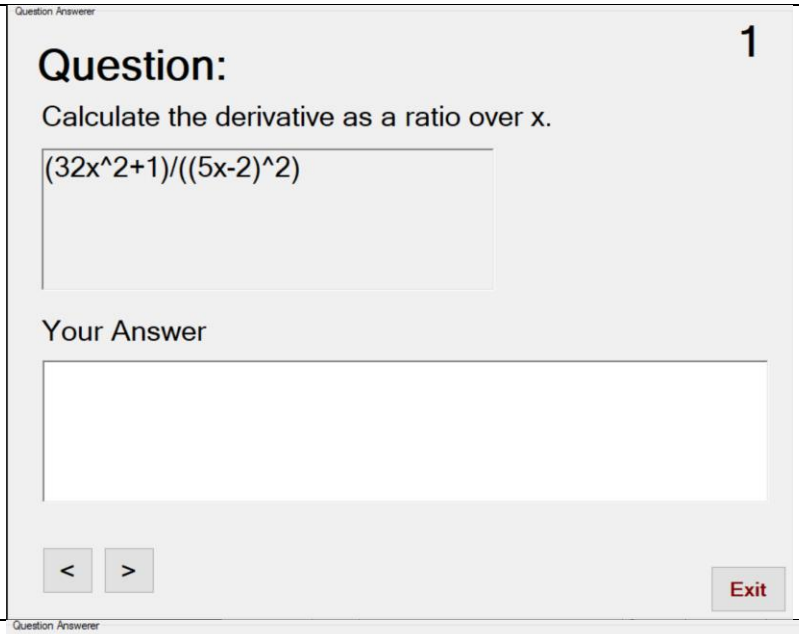
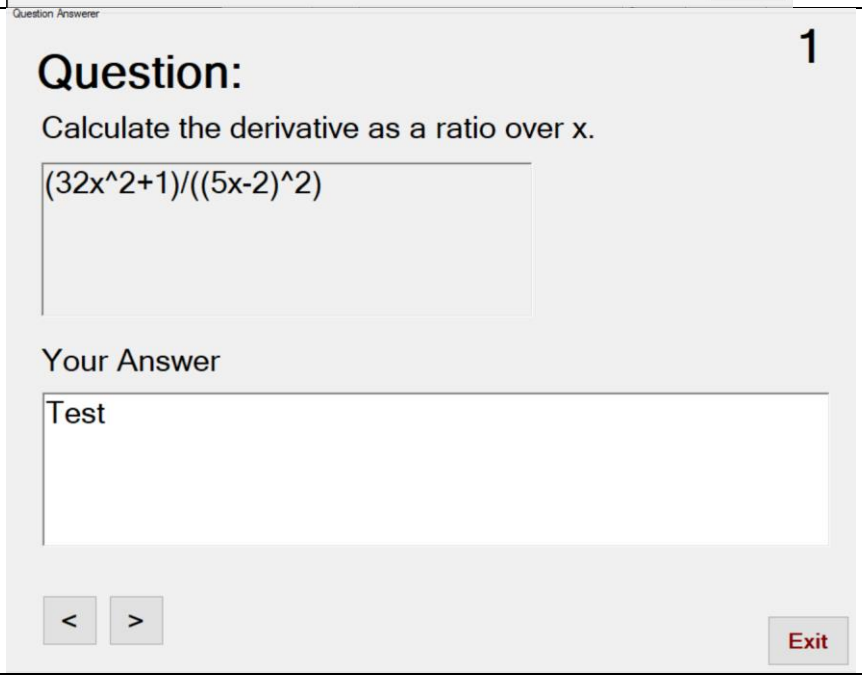
38	 Basic Test Question Ex... 20/05/2021 20:22 JSON File 2 KB Basic Test Question Export.json Json Files Open Cancel Tests Basic Test TEST 1 Testing Test VIEW TEST REMOVE ADD NEW TEST CLEAR ALL EXPORT
39	 Tests Basic Test View Basic Test Delete Basic Test Add new test

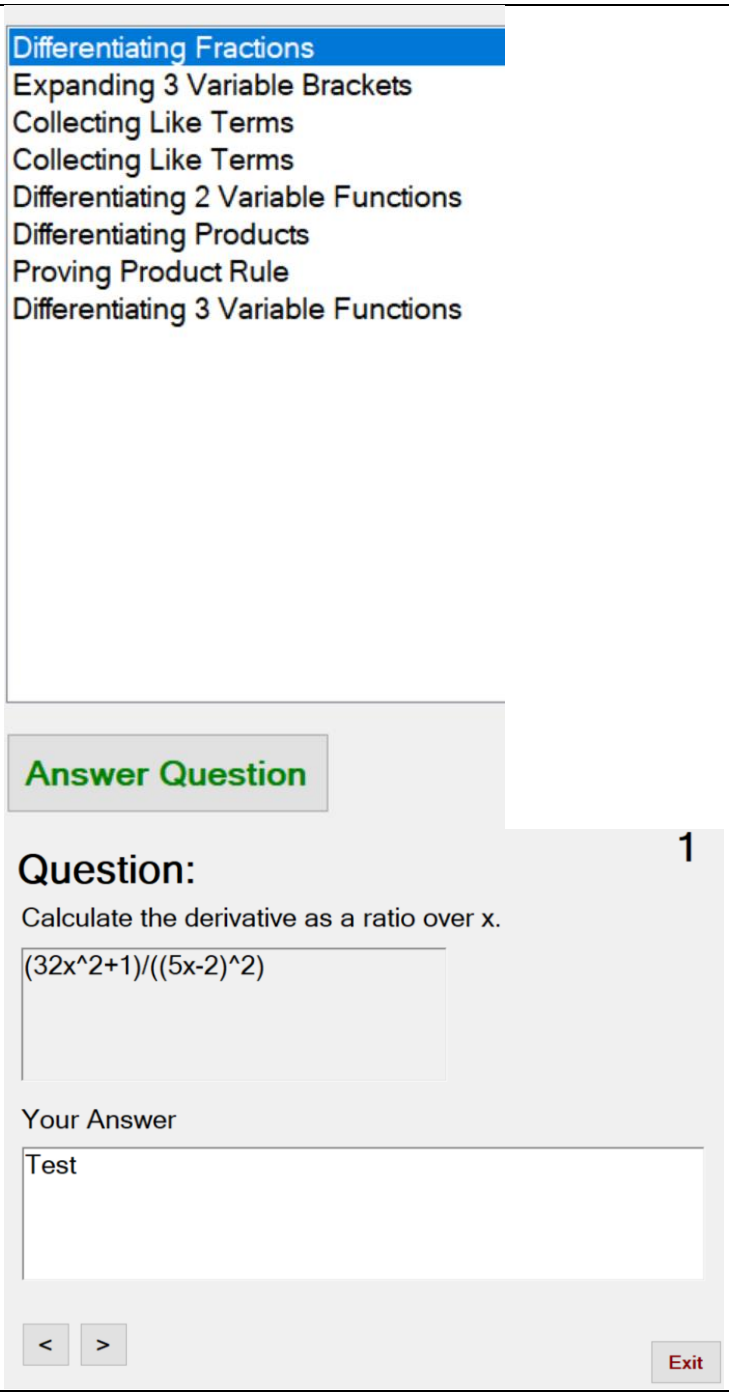
40


Test Viewer
Questions
Differentiating Fractions
Expanding 3 Variable Brackets
Collecting Like Terms
Collecting Like Terms
Differentiating 2 Variable Functions
Differentiating Products
Proving Product Rule
Differentiating 3 Variable Functions
Question
Answer
Answer Question
Exit

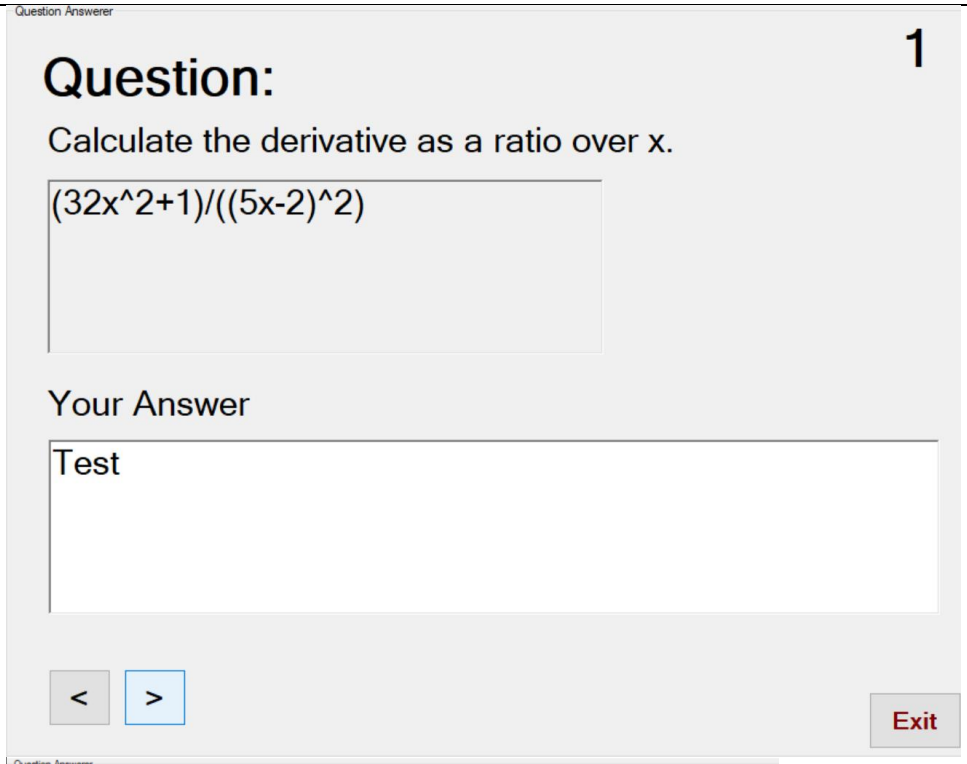
41

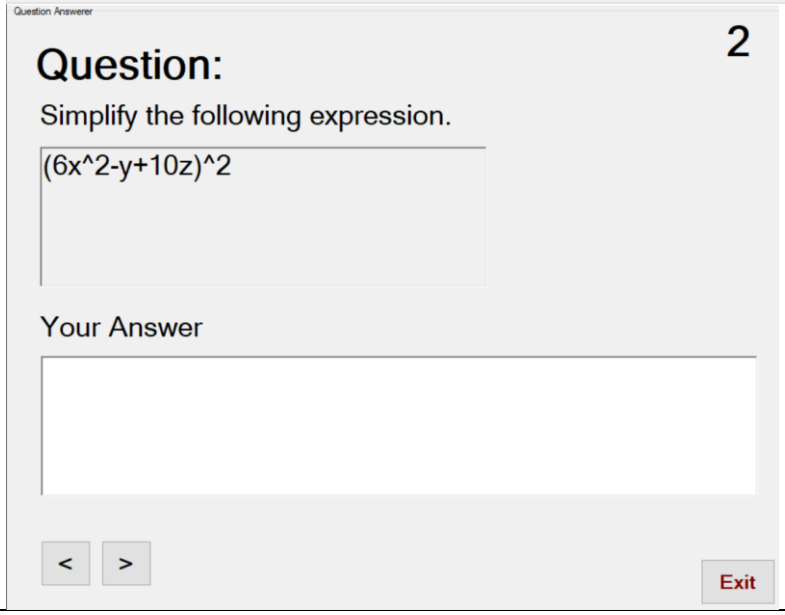

Questions
Differentiating Fractions
Expanding 3 Variable Brackets
Collecting Like Terms
Collecting Like Terms
Differentiating 2 Variable Functions
Differentiating Products
Proving Product Rule
Differentiating 3 Variable Functions
Answer Question

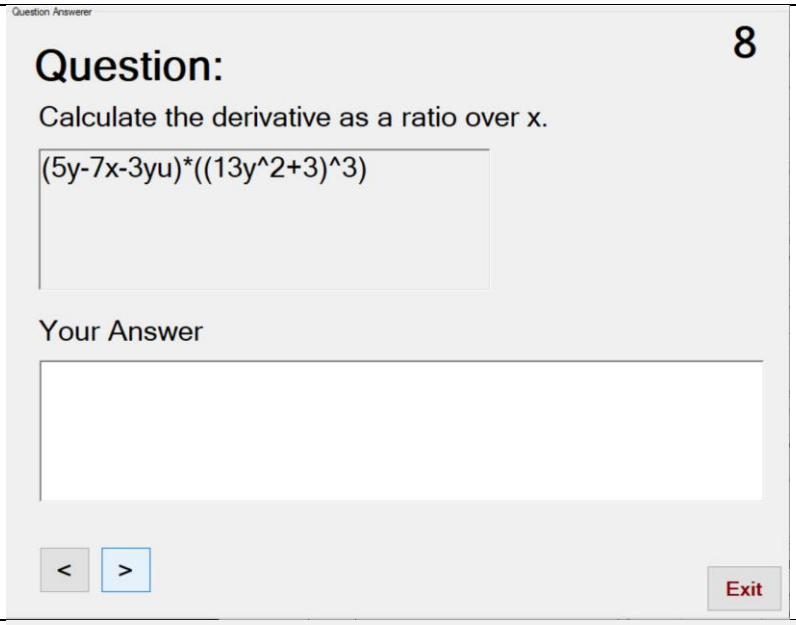
	 Question Answerer 1 Question: Calculate the derivative as a ratio over x. (32x^2+1)/((5x-2)^2) Your Answer < >Exit
42	 Question Answerer 1 Question: Calculate the derivative as a ratio over x. (32x^2+1)/((5x-2)^2) Your Answer Test < >Exit


Differentiating Fractions
Expanding 3 Variable Brackets
Collecting Like Terms
Collecting Like Terms
Differentiating 2 Variable Functions
Differentiating Products
Proving Product Rule
Differentiating 3 Variable Functions
Answer Question
Question:
1
Calculate the derivative as a ratio over x.
(32x^2+1)/((5x-2)^2)
Your Answer
Test
<
>
Exit

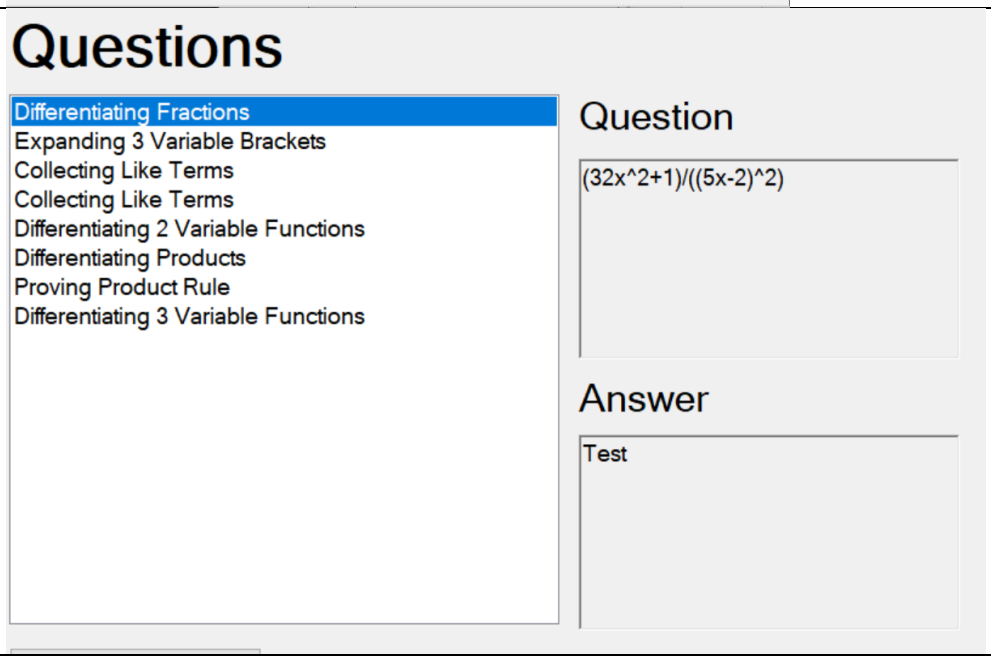
43

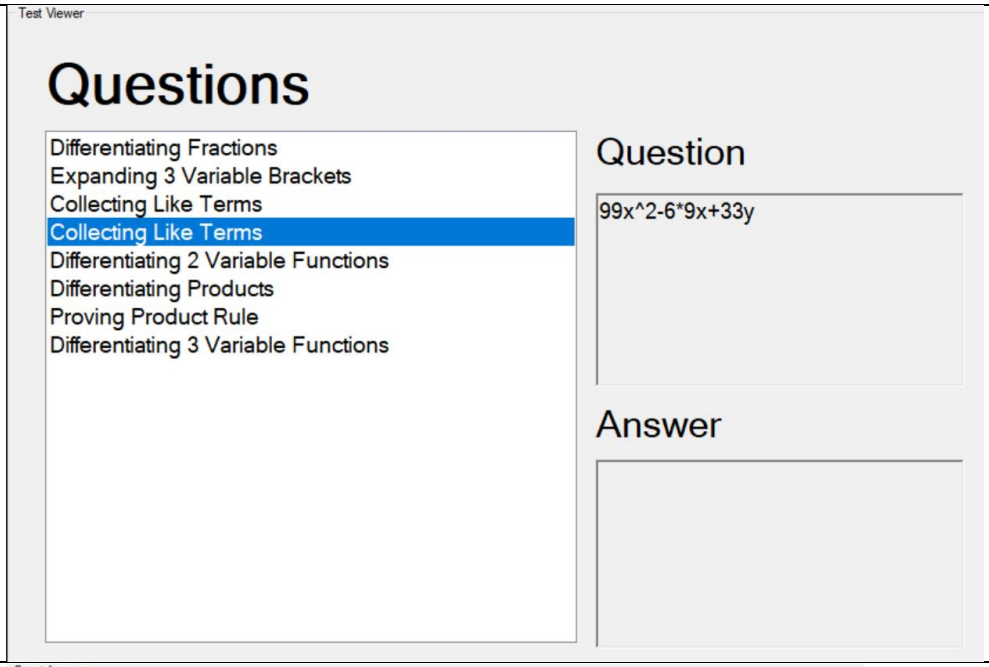

Question Answerer
1
Question:
Calculate the derivative as a ratio over x.
(32x^2+1)/((5x-2)^2)
Your Answer
Test
<
>
Exit


Question Answerer
2
Question:
Simplify the following expression.
(6x^2-y+10z)^2
Your Answer
<
>
Exit

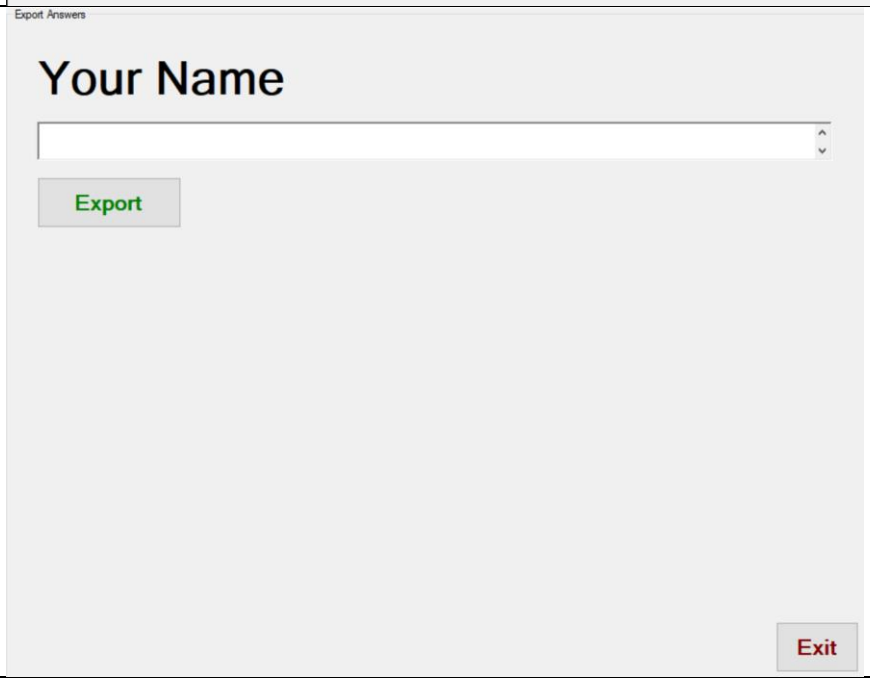

Question Answerer
8
Question:
Calculate the derivative as a ratio over x.
(5y-7x-3yu)*((13y^2+3)^3)
Your Answer
<
>
Exit

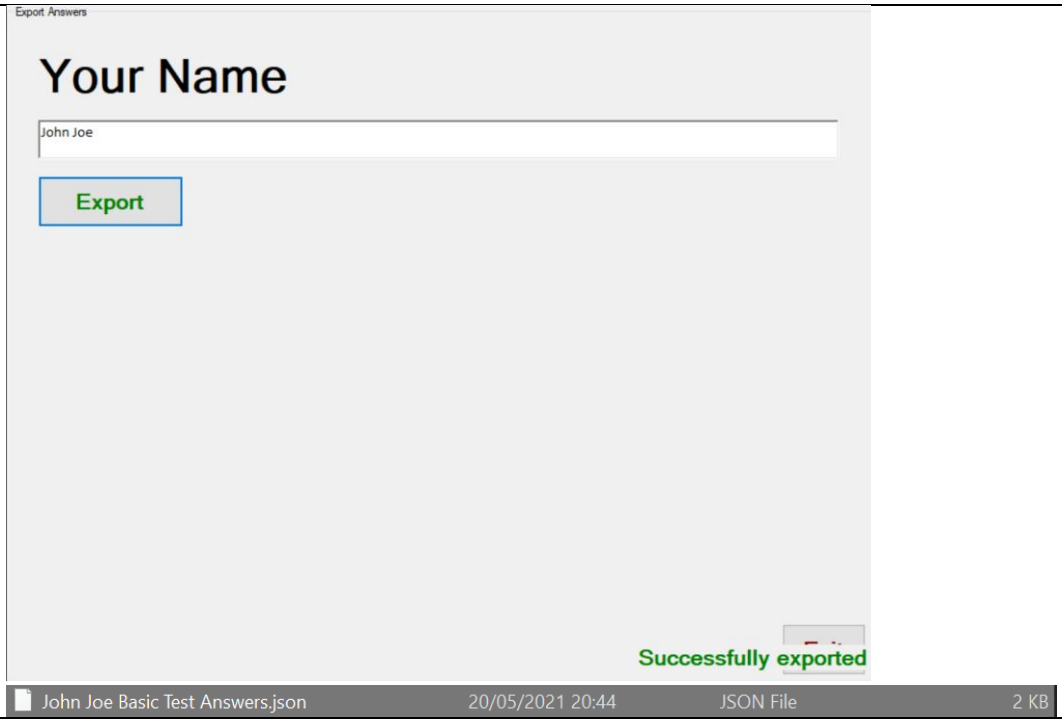
44


Questions
Differentiating Fractions
Expanding 3 Variable Brackets
Collecting Like Terms
Collecting Like Terms
Differentiating 2 Variable Functions
Differentiating Products
Proving Product Rule
Differentiating 3 Variable Functions
Question
(32x^2+1)/((5x-2)^2)
Answer
Test


Test Viewer
Questions
Differentiating Fractions
Expanding 3 Variable Brackets
Collecting Like Terms
Collecting Like Terms
Differentiating 2 Variable Functions
Differentiating Products
Proving Product Rule
Differentiating 3 Variable Functions
Question
99x^2-6*9x+33y
Answer

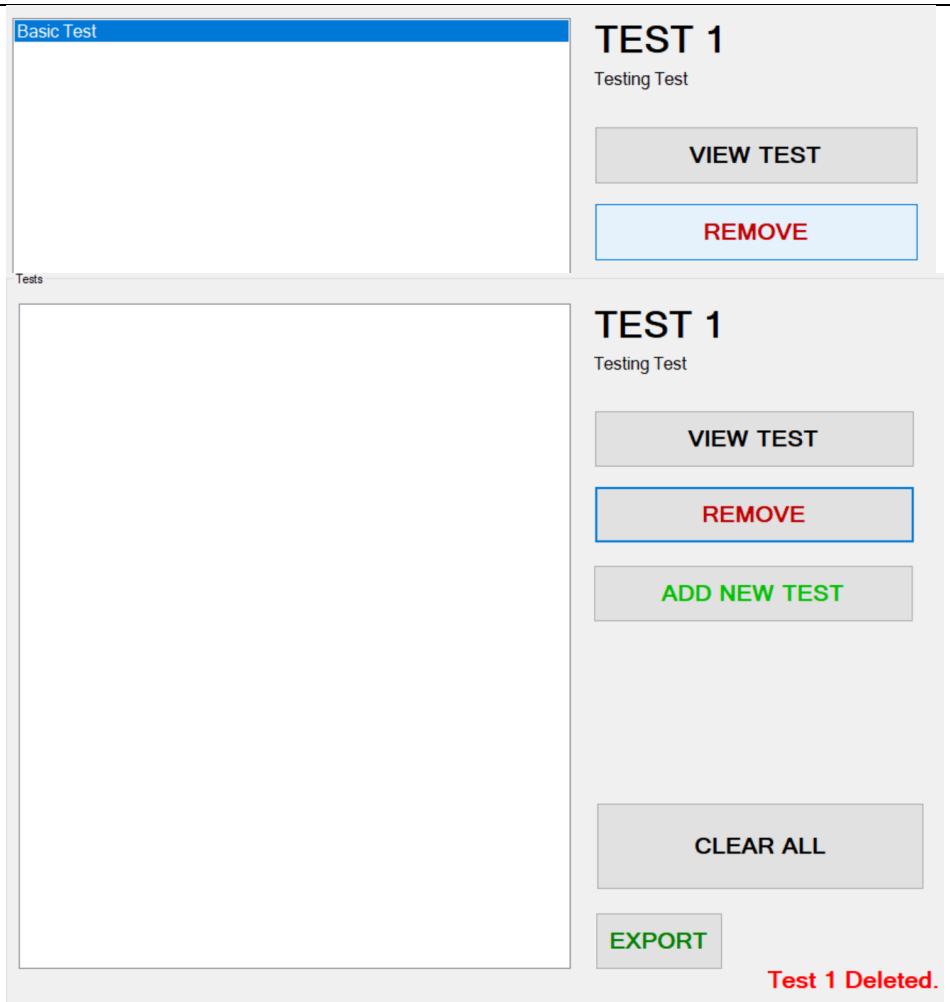
45


Export Answers
Your Name
Export
Exit

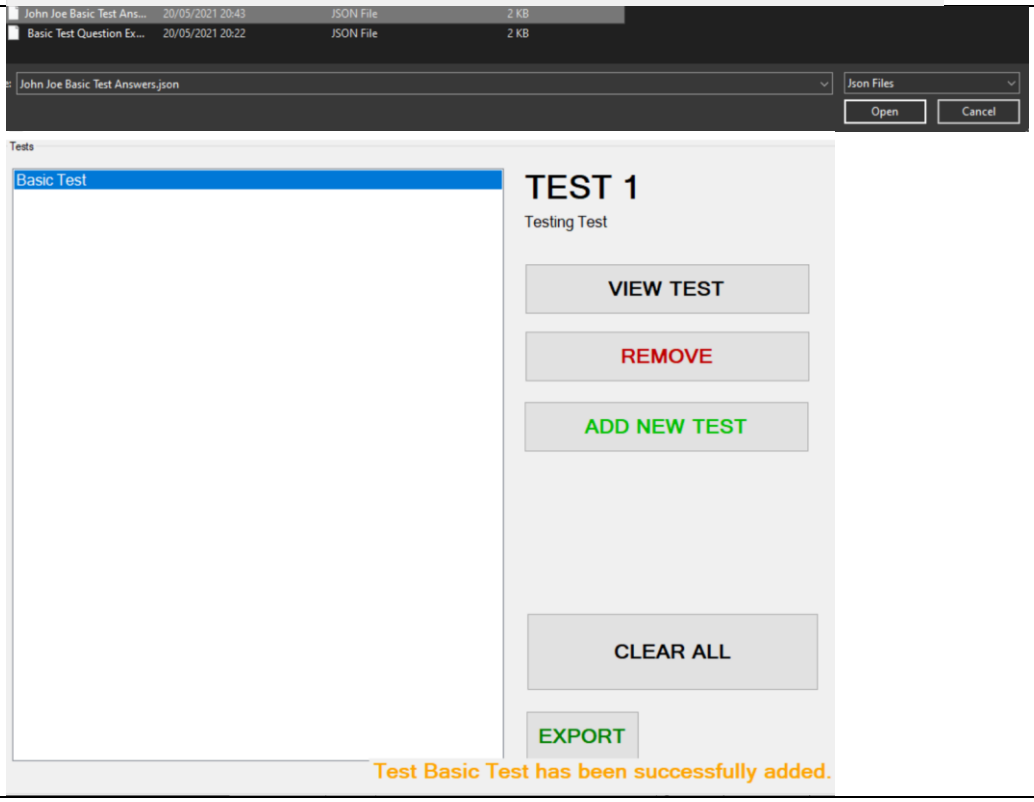
46	 Export Answers Your Name John Joe Export Successfully exported John Joe Basic Test Answers.json 20/05/2021 20:44 JSON File 2 KB

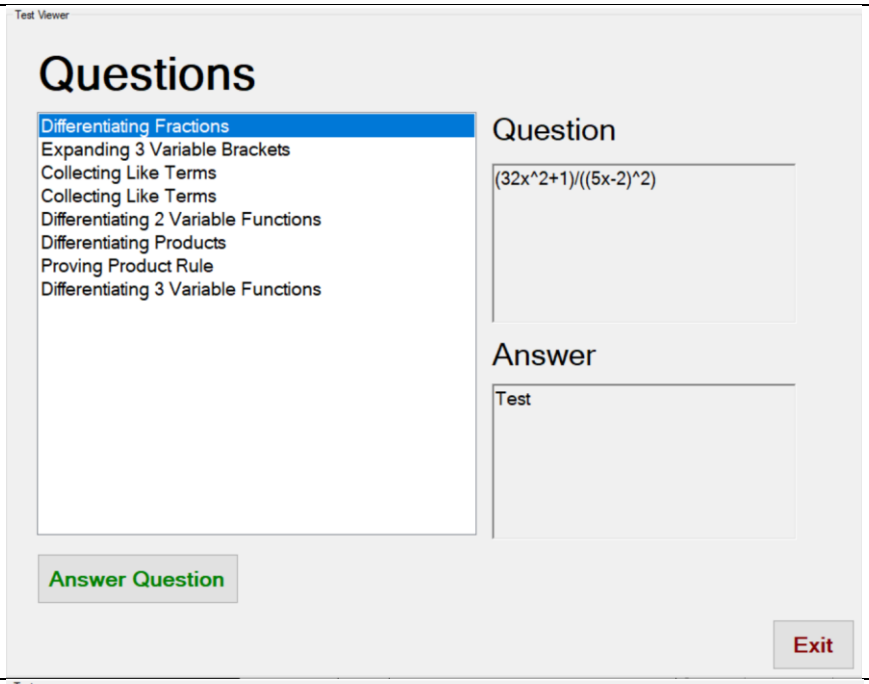
```
▼ array [9]
  ▼ 0 {4}
      NAME : Basic Test
      DESCRIPTION : Testing Test
      MARKED : False
      STUDENT NAME : John Joe
  ▼ 1 {7}
      QUESTION : (32x^2+1)/((5x-2)^2)
      QUESTION TITLE : Calculate the derivative as a ratio over x.
      QUESTION TYPE : DIFFERENTIATION
      ANSWER : Test
      TYPE : Differentiating Fractions
      TEACHER EDITED : False
      STATUS : CORRECT
  ▼ 2 {7}
      QUESTION : (6x^2-y+10z)^2
      QUESTION TITLE : Simplify the following expression.
      QUESTION TYPE : SIMPLIFICATION
      ANSWER : -12x^2y+120x^2z+36x^4+y^2-20yz+100z^2
      TYPE : Expanding 3 Variable Brackets
      TEACHER EDITED : False
      STATUS : CORRECT
  ▼ 3 {7}
      QUESTION : 3x+9z-10x^2+9y+10x-2y
      QUESTION TITLE : Simplify the following expression.
      QUESTION TYPE : SIMPLIFICATION
      ANSWER : 9x^2
      TYPE : Collecting Like Terms
      TEACHER EDITED : False
      STATUS : CORRECT
  ▼ 4 {7}
      QUESTION : 99x^2-6*9x+33y
      QUESTION TITLE : Simplify the following expression.
      QUESTION TYPE : SIMPLIFICATION
      ANSWER : 12
      TYPE : Collecting Like Terms
      TEACHER EDITED : False
      STATUS : CORRECT
  ▼ 5 {7}
      QUESTION : (2y+3x)^2
      QUESTION TITLE : Calculate the derivative as a ratio over x.
      QUESTION TYPE : DIFFERENTIATION
      ANSWER : 2x
      TYPE : Differentiating 2 Variable Functions
      TEACHER EDITED : False
      STATUS : CORRECT
  ▼ 6 {7}
      QUESTION : ((5x-10)^36)*(5x+10)
      QUESTION TITLE : Calculate the derivative as a ratio over x.
      QUESTION TYPE : DIFFERENTIATION
      ANSWER : 10
      TYPE : Differentiating Products
      TEACHER EDITED : False
      STATUS : CORRECT
  ▼ 7 {7}
      QUESTION : uv
      QUESTION TITLE : Calculate the derivative as a ratio over x.
      QUESTION TYPE : DIFFERENTIATION
      ANSWER : uc
      TYPE : Proving Product Rule
      TEACHER EDITED : False
      STATUS : CORRECT
  ▼ 8 {7}
      QUESTION : (5y-7x-3yu)*((13y^2+3)^3)
      QUESTION TITLE : Calculate the derivative as a ratio over x.
      QUESTION TYPE : DIFFERENTIATION
      ANSWER : a
      TYPE : Differentiating 3 Variable Functions
      TEACHER EDITED : False
      STATUS : CORRECT
```

47

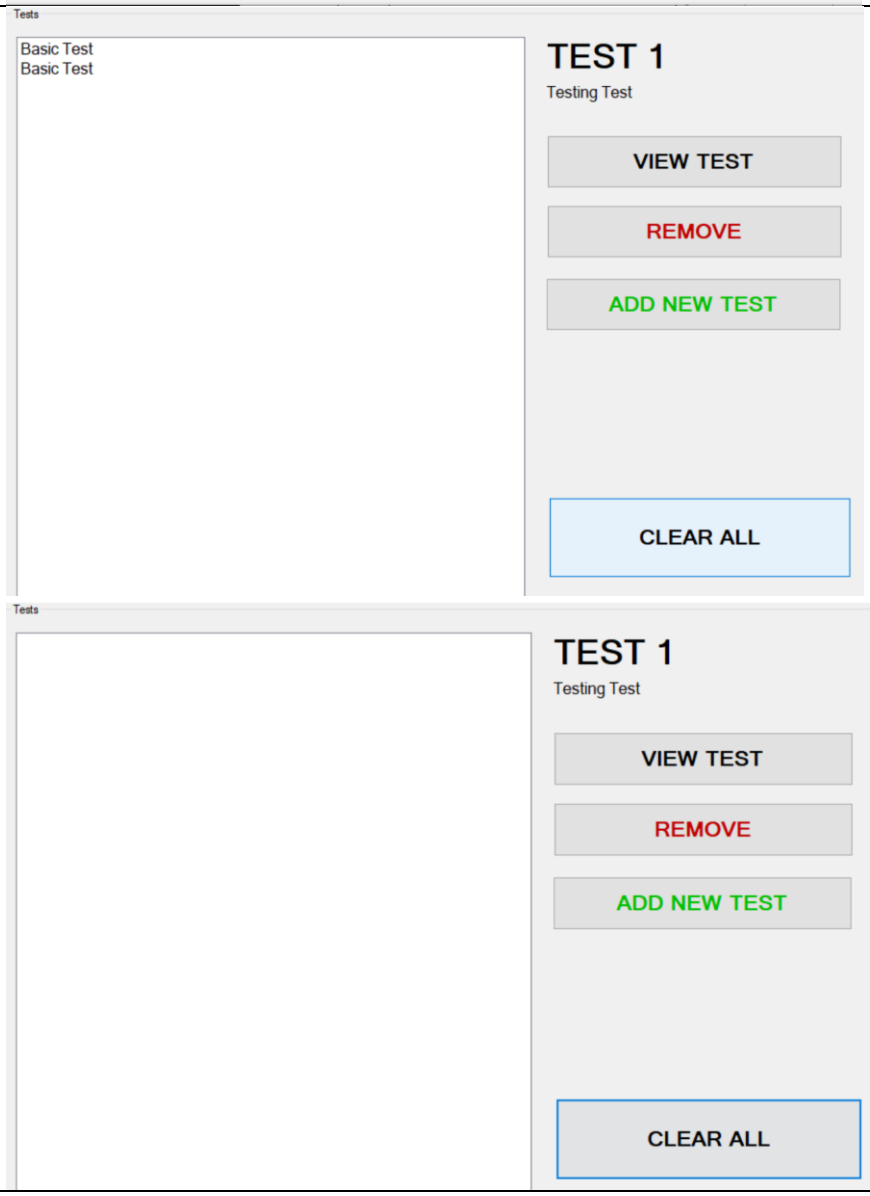

Basic Test
TEST 1
Testing Test
VIEW TEST
REMOVE
Tests
TEST 1
Testing Test
VIEW TEST
REMOVE
ADD NEW TEST
CLEAR ALL
EXPORT
Test 1 Deleted.

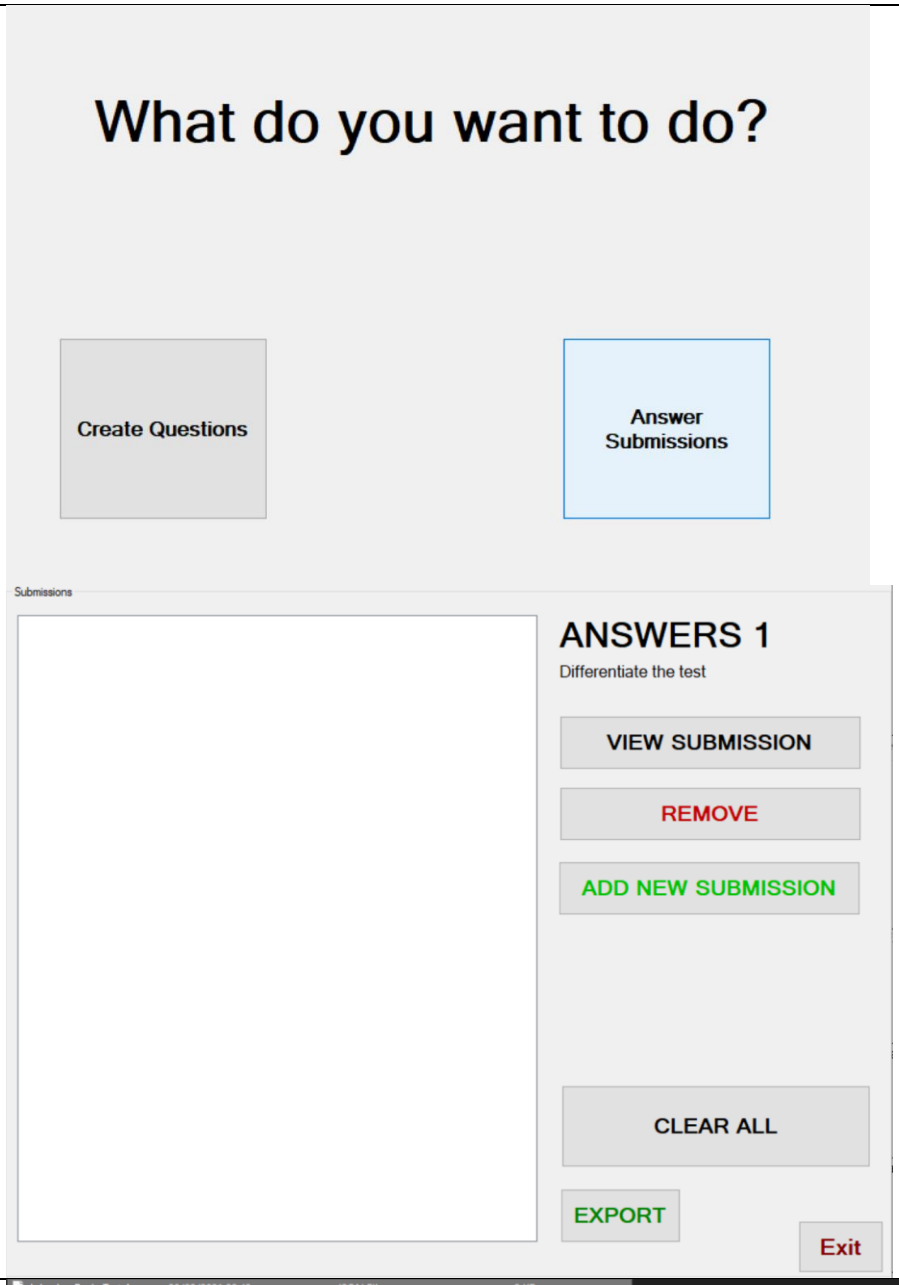
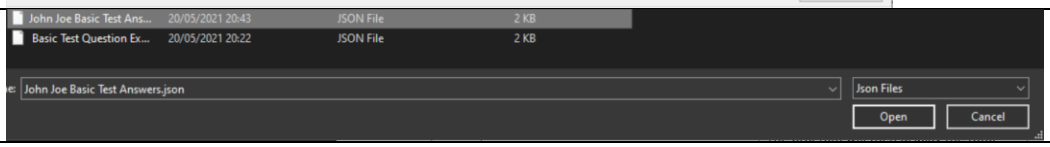
48

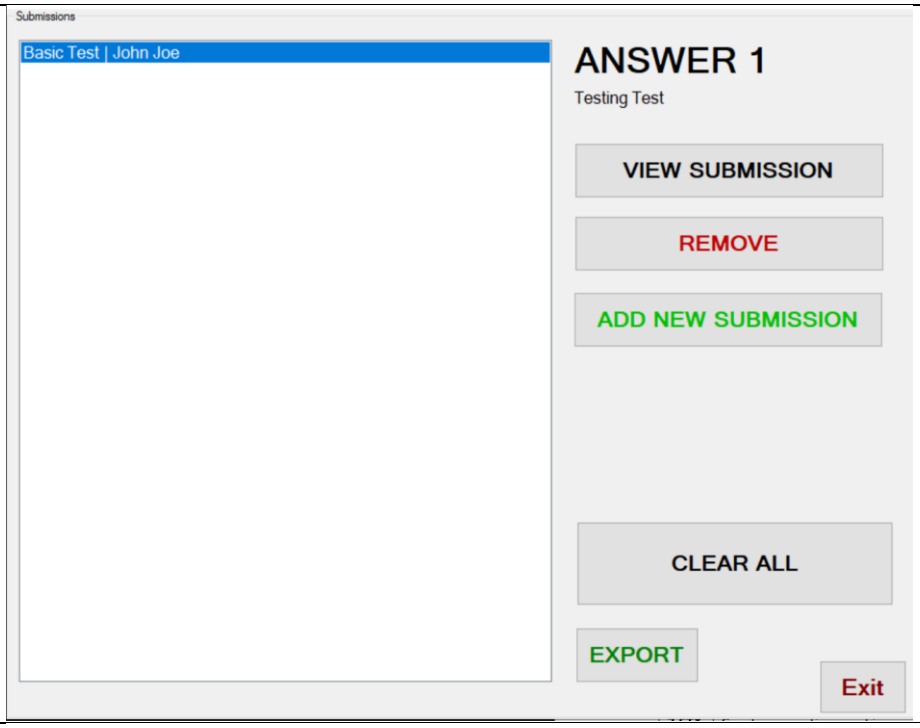

John Joe Basic Test Ans... 20/05/2021 20:43 JSON File 2 KB
Basic Test Question Ex... 20/05/2021 20:22 JSON File 2 KB
John Joe Basic Test Answers.json
Json Files
Open
Cancel
Tests
Basic Test
TEST 1
Testing Test
VIEW TEST
REMOVE
ADD NEW TEST
CLEAR ALL
EXPORT
Test Basic Test has been successfully added.


Test Viewer
Questions
Differentiating Fractions
Expanding 3 Variable Brackets
Collecting Like Terms
Collecting Like Terms
Differentiating 2 Variable Functions
Differentiating Products
Proving Product Rule
Differentiating 3 Variable Functions
Question
(32x^2+1)/((5x-2)^2)
Answer
Test
Answer Question
Exit

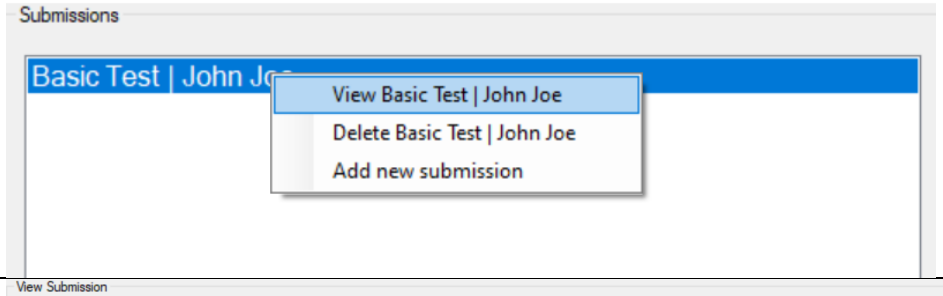
49


Tests
Basic Test
Basic Test
TEST 1
Testing Test
VIEW TEST
REMOVE
ADD NEW TEST
CLEAR ALL
Tests
TEST 1
Testing Test
VIEW TEST
REMOVE
ADD NEW TEST
CLEAR ALL

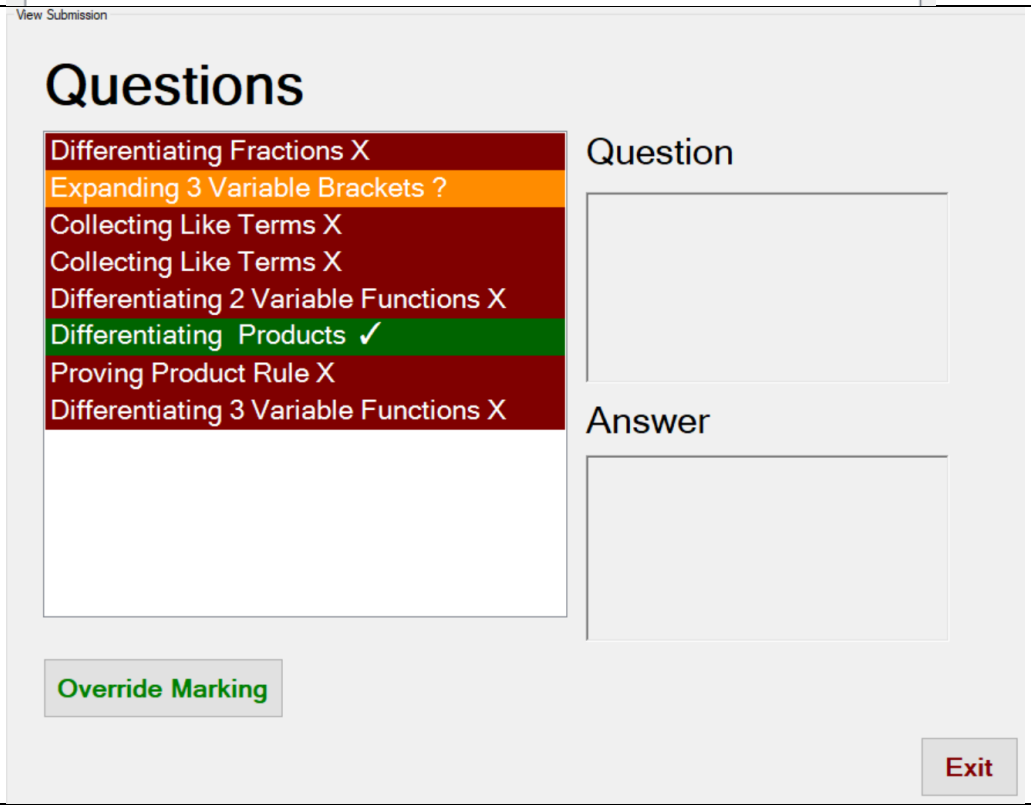
50	 What do you want to do? Create Questions Answer Submissions Submissions ANSWERS 1 Differentiate the test VIEW SUBMISSION REMOVE ADD NEW SUBMISSION CLEAR ALL EXPORT Exit
51	 John Joe Basic Test Ans... 20/05/2021 20:43 JSON File 2 KB Basic Test Question Ex... 20/05/2021 20:22 JSON File 2 KB John Joe Basic Test Answers.json Json Files Open Cancel


Submissions
Basic Test | John Joe
ANSWER 1
Testing Test
VIEW SUBMISSION
REMOVE
ADD NEW SUBMISSION
CLEAR ALL
EXPORT
Exit

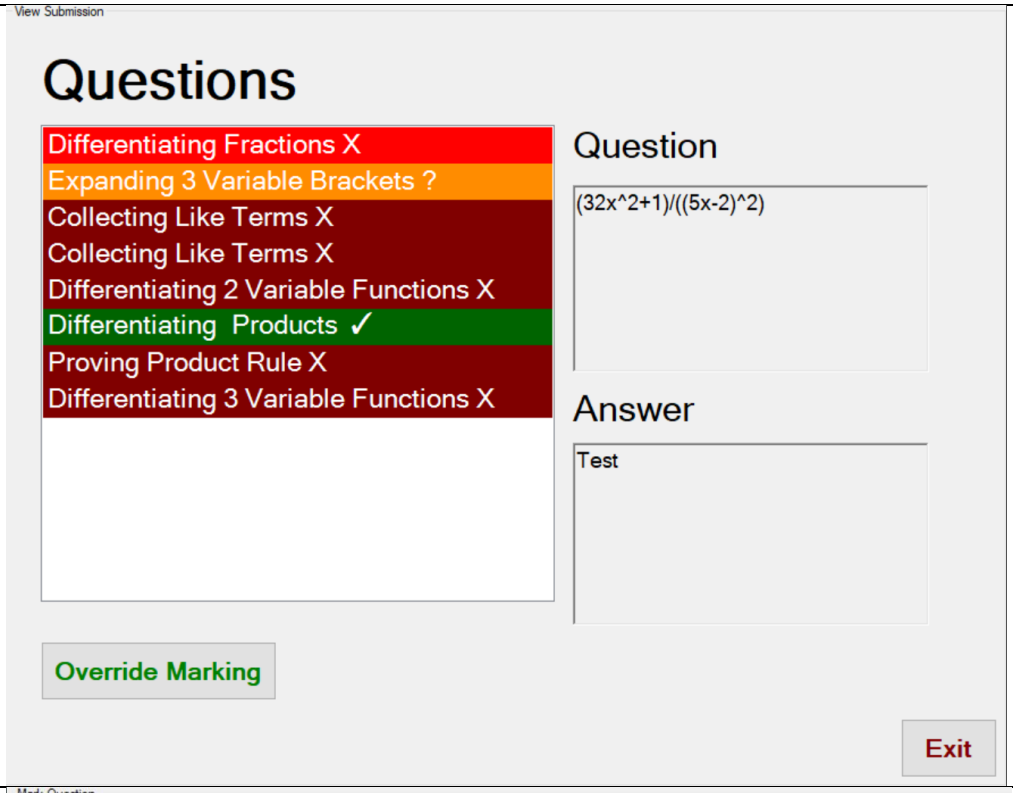
52


Submissions
Basic Test | John Joe
View Basic Test | John Joe
Delete Basic Test | John Joe
Add new submission

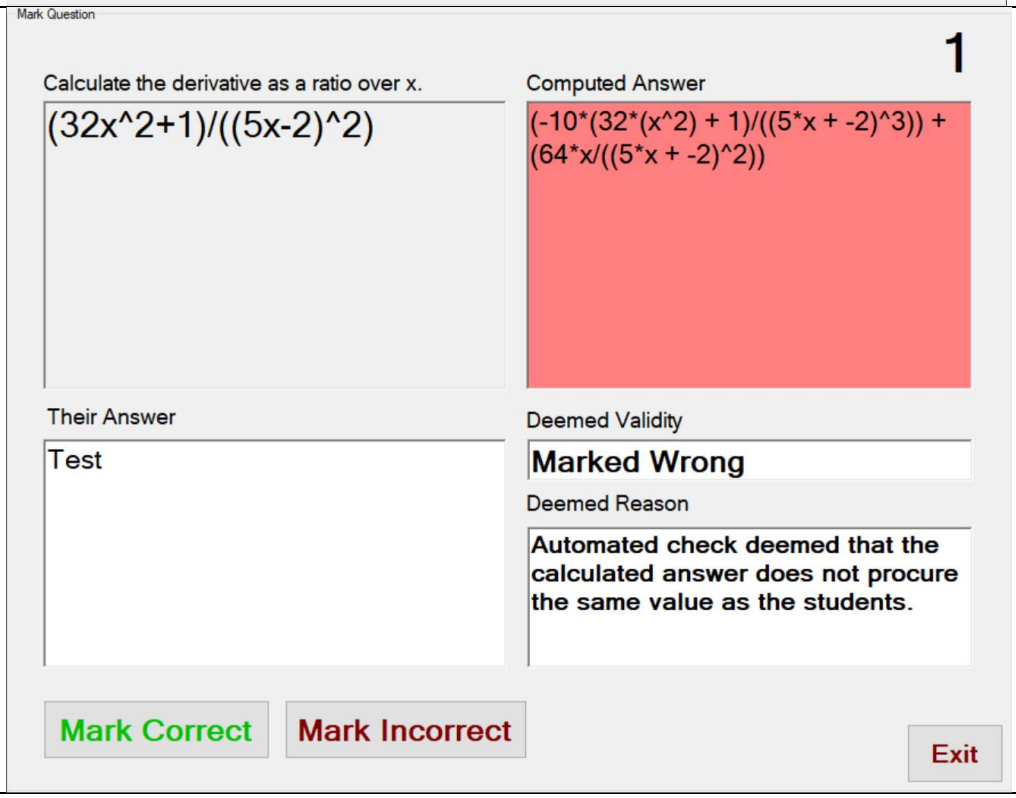
53


View Submission
Questions
Differentiating Fractions X
Expanding 3 Variable Brackets ?
Collecting Like Terms X
Collecting Like Terms X
Differentiating 2 Variable Functions X
Differentiating Products ✓
Proving Product Rule X
Differentiating 3 Variable Functions X
Question
Answer
Override Marking
Exit

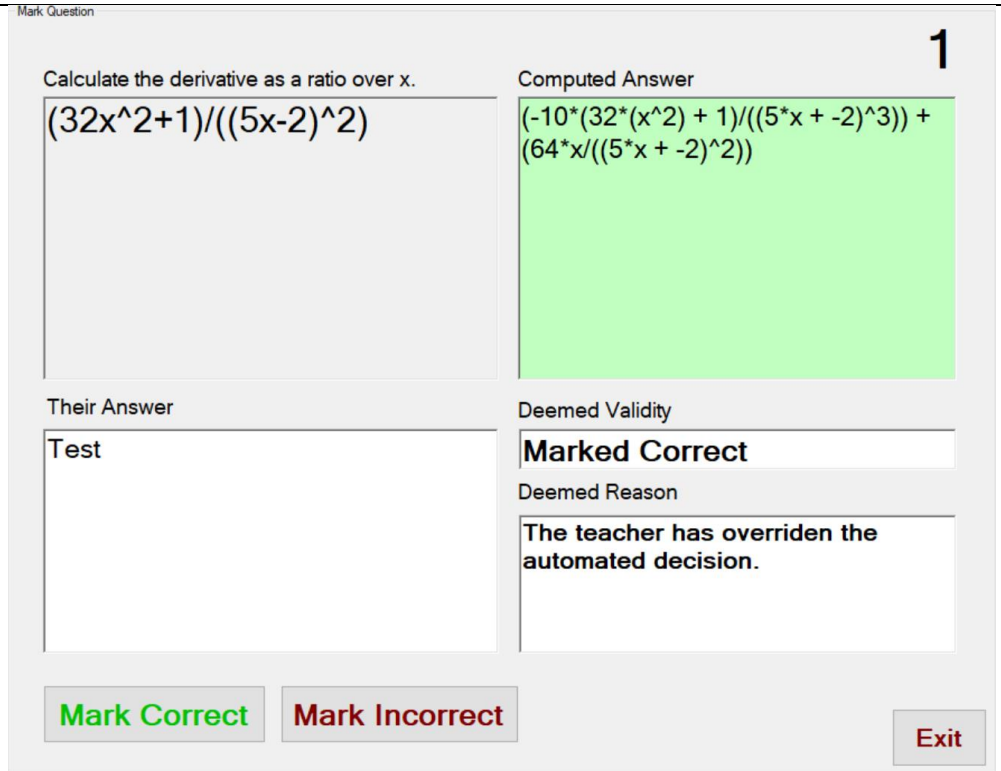
54


View Submission
Questions
Differentiating Fractions X
Expanding 3 Variable Brackets ?
Collecting Like Terms X
Collecting Like Terms X
Differentiating 2 Variable Functions X
Differentiating Products ✓
Proving Product Rule X
Differentiating 3 Variable Functions X
Question
(32x^2+1)/((5x-2)^2)
Answer
Test
Override Marking
Exit

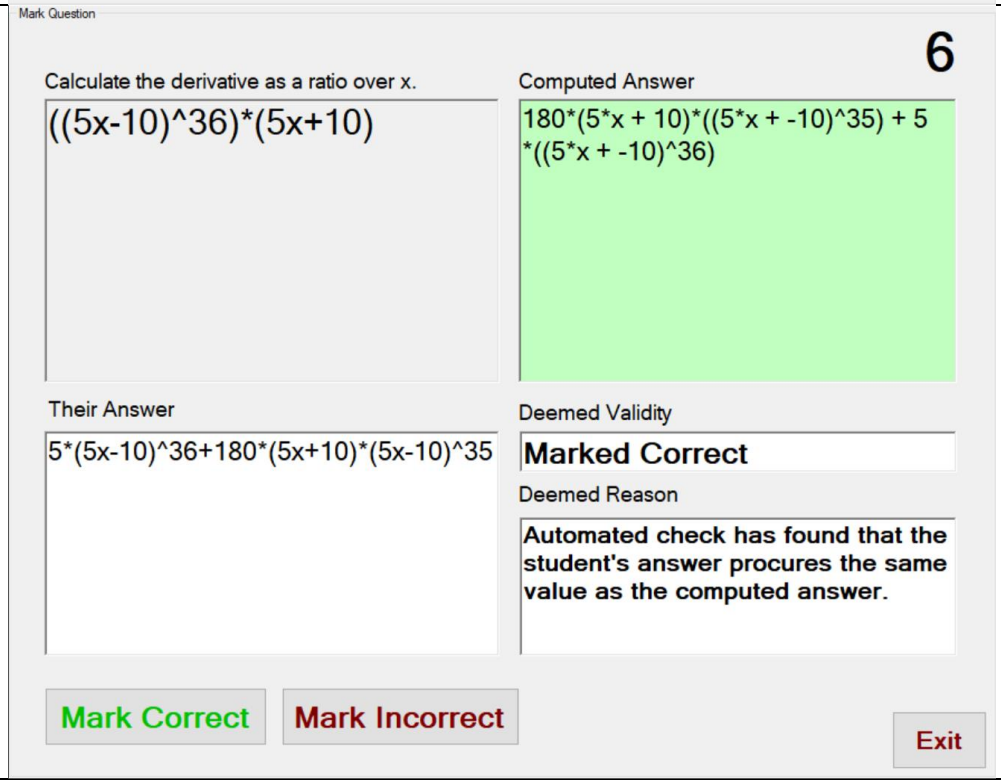
55


Mark Question
1
Calculate the derivative as a ratio over x.
(32x^2+1)/((5x-2)^2)
Computed Answer
(-10*(32*(x^2) + 1)/((5*x + -2)^3)) + (64*x/((5*x + -2)^2))
Their Answer
Test
Deemed Validity
Marked Wrong
Deemed Reason
Automated check deemed that the calculated answer does not procure the same value as the students.
Mark Correct
Mark Incorrect
Exit

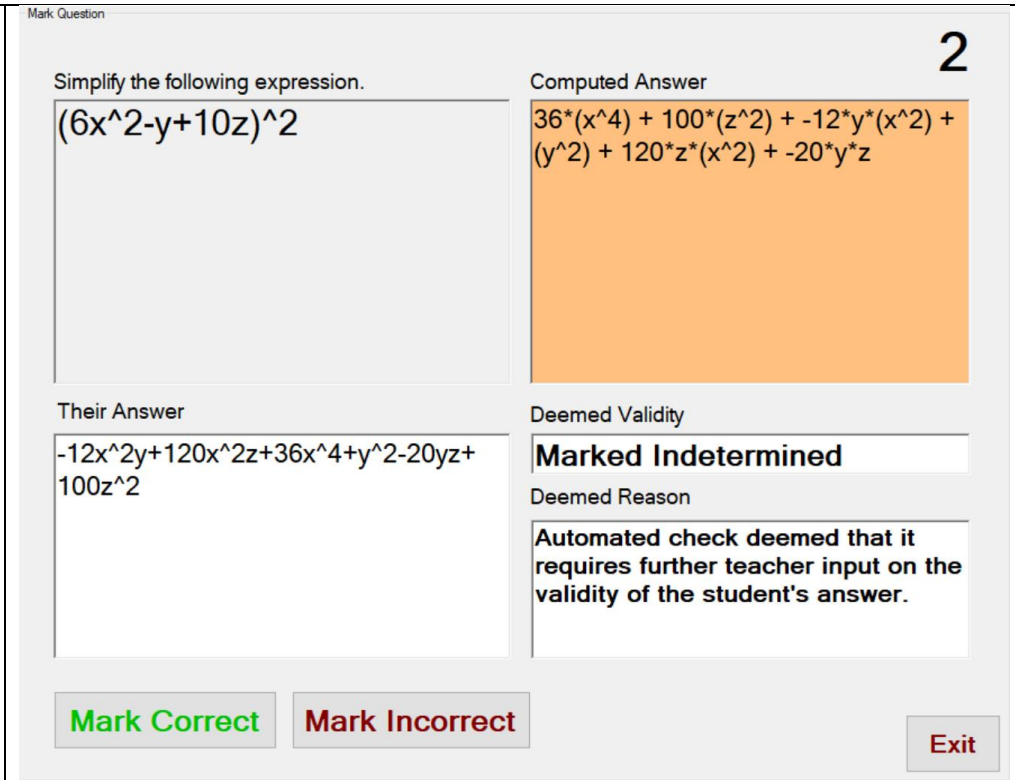
56


Mark Question
1
Calculate the derivative as a ratio over x.
(32x^2+1)/((5x-2)^2)
Computed Answer
(-10*(32*(x^2) + 1)/((5*x + -2)^3)) + (64*x/((5*x + -2)^2))
Their Answer
Test
Deemed Validity
Marked Correct
Deemed Reason
The teacher has overriden the automated decision.
Mark Correct
Mark Incorrect
Exit

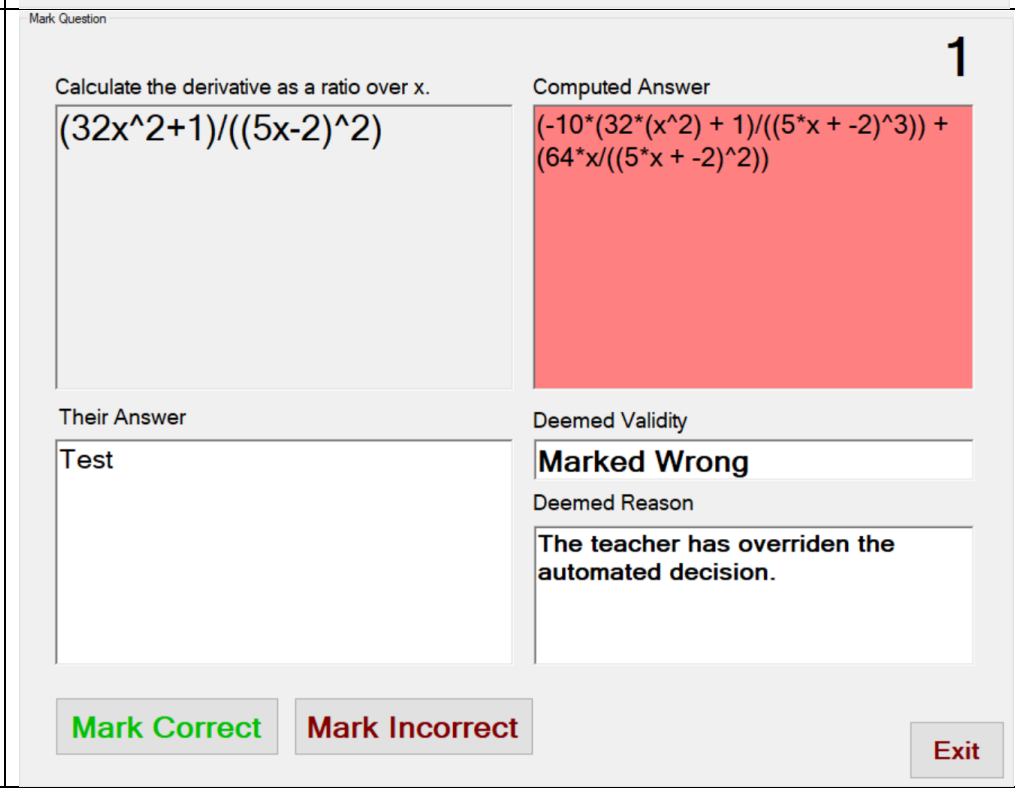
57

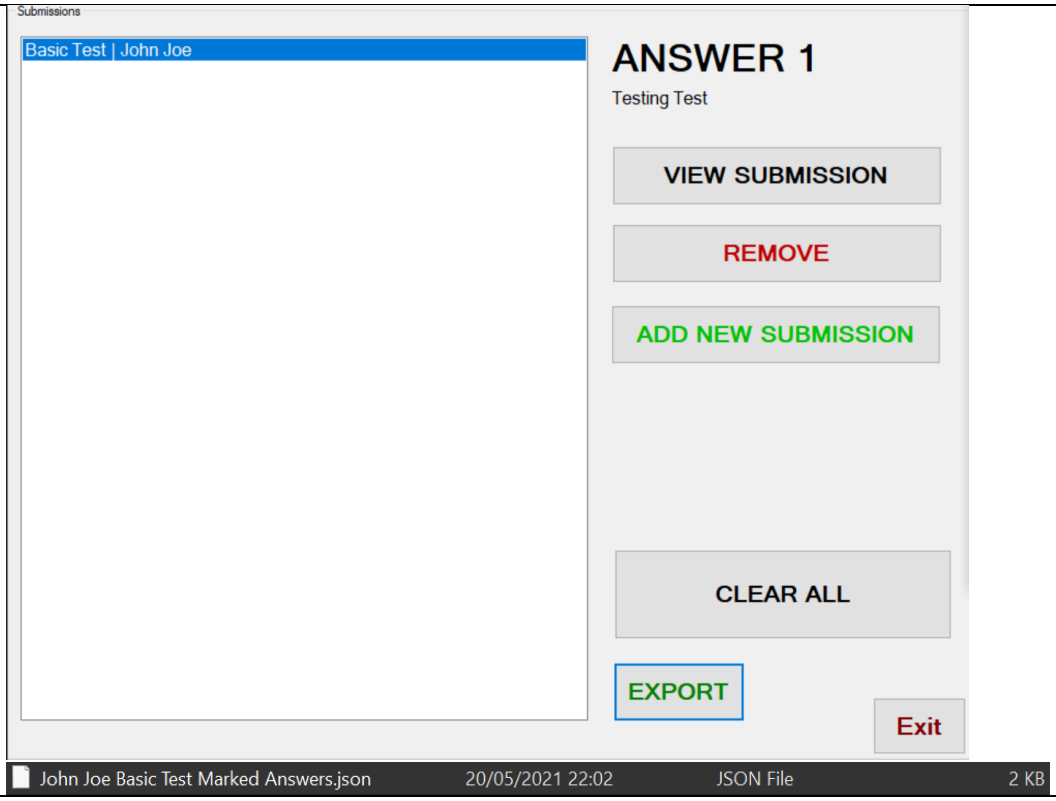

Mark Question
6
Calculate the derivative as a ratio over x.
((5x-10)^36)*(5x+10)
Computed Answer
180*(5*x + 10)*((5*x + -10)^35) + 5*((5*x + -10)^36)
Their Answer
5*(5x-10)^36+180*(5x+10)*(5x-10)^35
Deemed Validity
Marked Correct
Deemed Reason
Automated check has found that the student's answer procures the same value as the computed answer.
Mark Correct
Mark Incorrect
Exit

58


Mark Question
2
Simplify the following expression.
(6x^2-y+10z)^2
Computed Answer
36*(x^4) + 100*(z^2) + -12*y*(x^2) + (y^2) + 120*z*(x^2) + -20*y*z
Their Answer
-12x^2y+120x^2z+36x^4+y^2-20yz+100z^2
Deemed Validity
Marked Indetermined
Deemed Reason
Automated check deemed that it requires further teacher input on the validity of the student's answer.
Mark Correct
Mark Incorrect
Exit

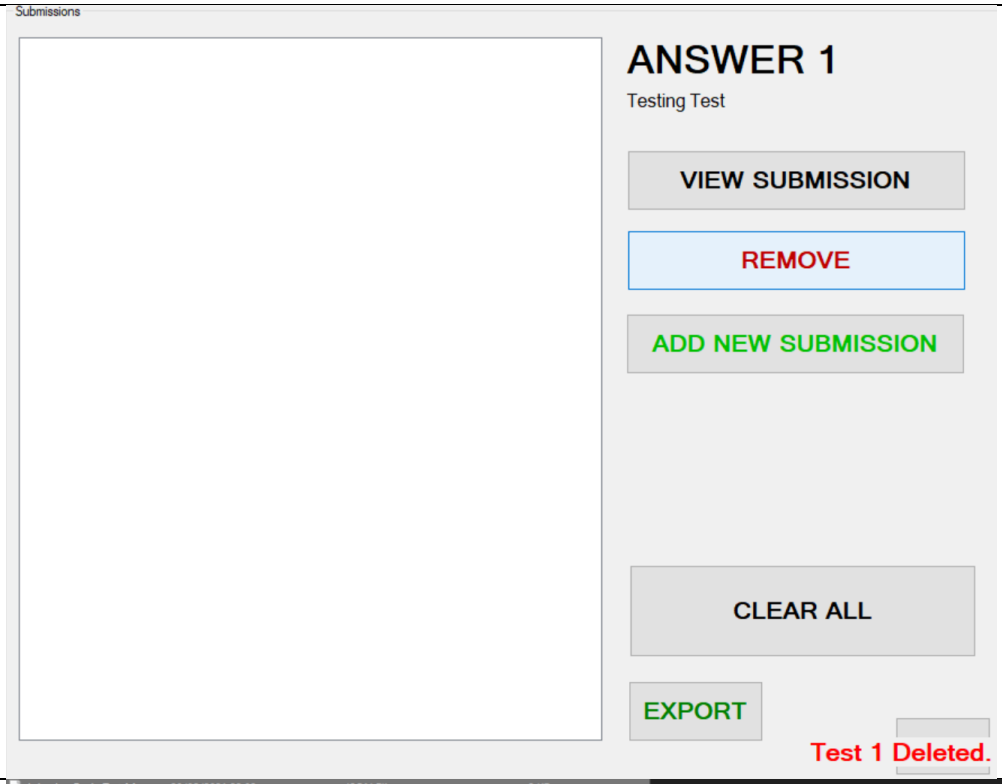
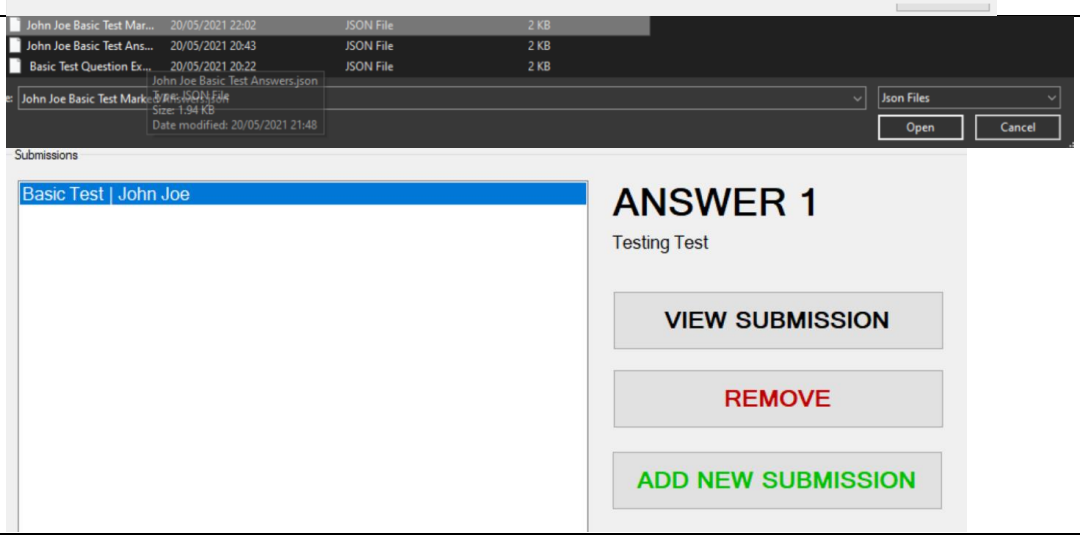
59

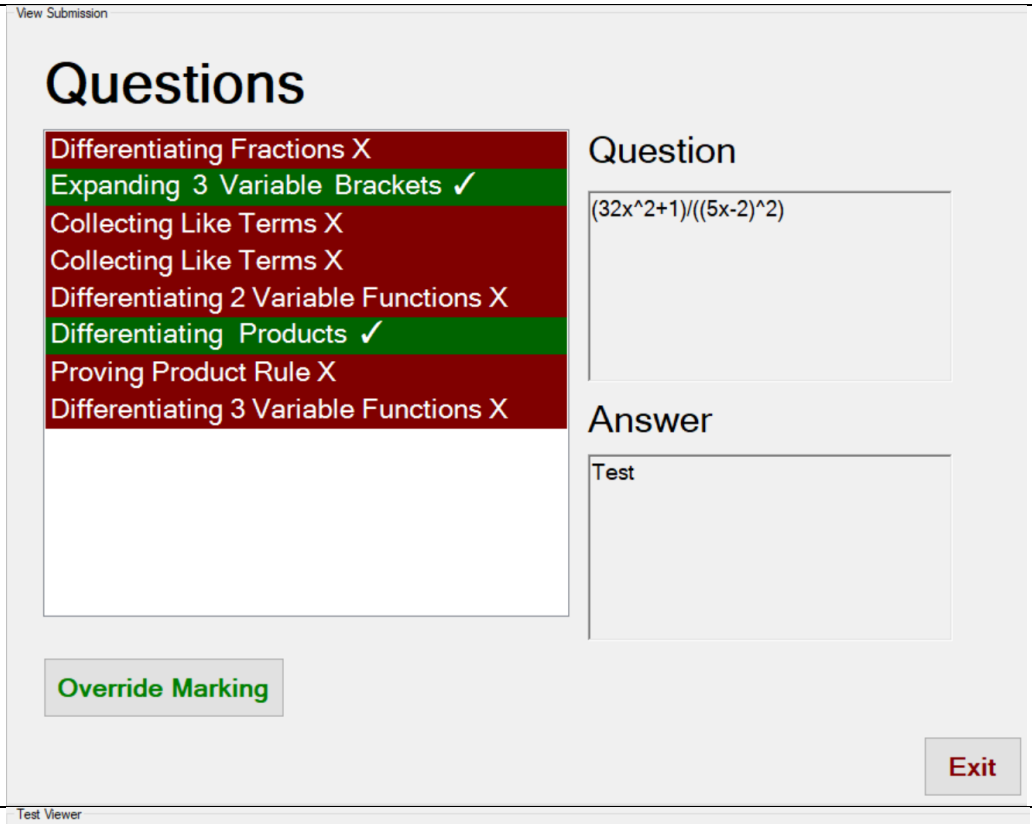
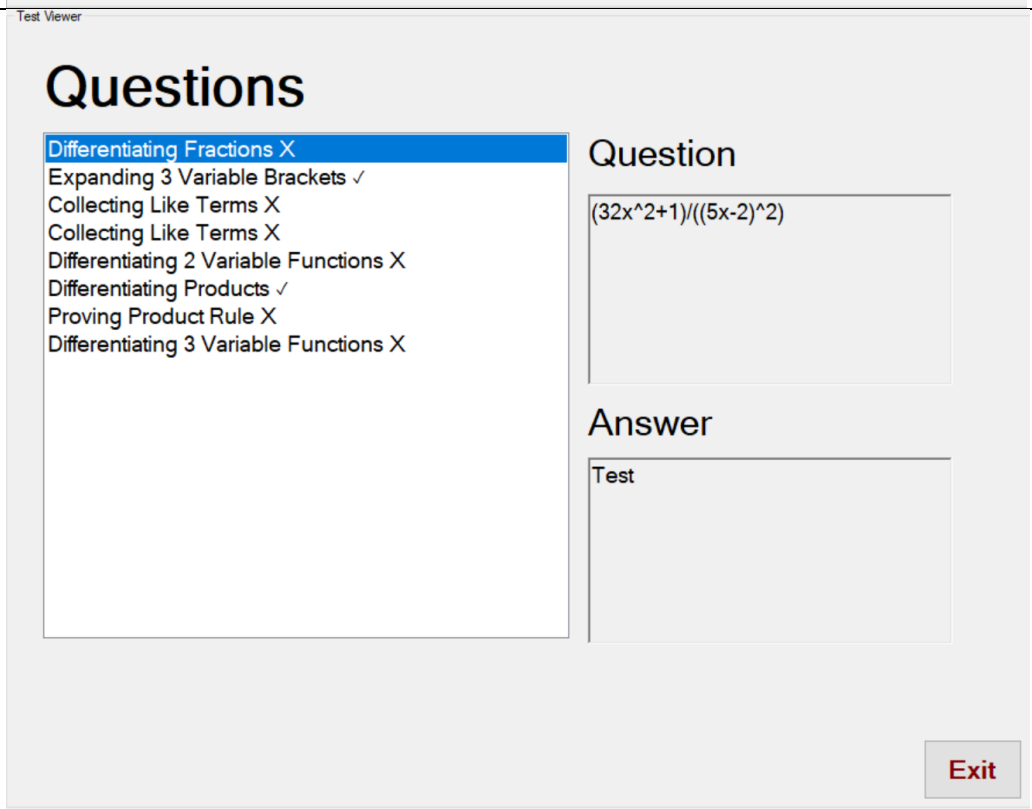

Mark Question
1
Calculate the derivative as a ratio over x.
(32x^2+1)/((5x-2)^2)
Computed Answer
(-10*(32*(x^2) + 1)/((5*x + -2)^3)) + (64*x/((5*x + -2)^2))
Their Answer
Test
Deemed Validity
Marked Wrong
Deemed Reason
The teacher has overriden the automated decision.
Mark Correct
Mark Incorrect
Exit

60	

Submissions
Basic Test | John Joe
ANSWER 1
Testing Test
VIEW SUBMISSION
REMOVE
ADD NEW SUBMISSION
CLEAR ALL
EXPORT
Exit
John Joe Basic Test Marked Answers.json
20/05/2021 22:02
JSON File
2 KB

```
▼ array [9]
  ▼ 0 {4}
      NAME : Basic Test
      DESCRIPTION : Testing Test
      MARKED : True
      STUDENT NAME : John Joe
  ▼ 1 {7}
      QUESTION : (32x^2+1)/((5x-2)^2)
      QUESTION TITLE : Calculate the derivative as a ratio over x.
      QUESTION TYPE : DIFFERENTIATION
      ANSWER : Test
      TYPE : Differentiating Fractions
      TEACHER EDITED : True
      STATUS : WRONG
  ▼ 2 {7}
      QUESTION : (6x^2-y+10z)^2
      QUESTION TITLE : Simplify the following expression.
      QUESTION TYPE : SIMPLIFICATION
      ANSWER : -12x^2y+120x^2z+36x^4+y^2-20yz+100z^2
      TYPE : Expanding 3 Variable Brackets
      TEACHER EDITED : True
      STATUS : CORRECT
  ▼ 3 {7}
      QUESTION : 3x+9z-10x^2+9y+10x-2y
      QUESTION TITLE : Simplify the following expression.
      QUESTION TYPE : SIMPLIFICATION
      ANSWER : 9x^2
      TYPE : Collecting Like Terms
      TEACHER EDITED : False
      STATUS : WRONG
  ▼ 4 {7}
      QUESTION : 99x^2-6*9x+33y
      QUESTION TITLE : Simplify the following expression.
      QUESTION TYPE : SIMPLIFICATION
      ANSWER : 12
      TYPE : Collecting Like Terms
      TEACHER EDITED : False
      STATUS : WRONG
  ▼ 5 {7}
      QUESTION : (2y+3x)^2
      QUESTION TITLE : Calculate the derivative as a ratio over x.
      QUESTION TYPE : DIFFERENTIATION
      ANSWER : 2x
      TYPE : Differentiating 2 Variable Functions
      TEACHER EDITED : False
      STATUS : WRONG
  ▼ 6 {7}
      QUESTION : ((5x-10)^36)*(5x+10)
      QUESTION TITLE : Calculate the derivative as a ratio over x.
      QUESTION TYPE : DIFFERENTIATION
      ANSWER : 5*(5x-10)^36+180*(5x+10)*(5x-10)^35
      TYPE : Differentiating Products
      TEACHER EDITED : False
      STATUS : CORRECT
  ▼ 7 {7}
      QUESTION : uv
      QUESTION TITLE : Calculate the derivative as a ratio over x.
      QUESTION TYPE : DIFFERENTIATION
      ANSWER : uc
      TYPE : Proving Product Rule
      TEACHER EDITED : False
      STATUS : WRONG
  ▼ 8 {7}
      QUESTION : (5y-7x-3yu)*((13y^2+3)^3)
      QUESTION TITLE : Calculate the derivative as a ratio over x.
      QUESTION TYPE : DIFFERENTIATION
      ANSWER : a
      TYPE : Differentiating 3 Variable Functions
      TEACHER EDITED : False
      STATUS : WRONG
```

61	 Submissions ANSWER 1 Testing Test VIEW SUBMISSION REMOVE ADD NEW SUBMISSION CLEAR ALL EXPORT Test 1 Deleted.
62	 John Joe Basic Test Mar... 20/05/2021 22:02 JSON File 2 KB John Joe Basic Test Ans... 20/05/2021 20:43 JSON File 2 KB Basic Test Question Ex... 20/05/2021 20:22 JSON File 2 KB John Joe Basic Test Answers.json Size: 1.94 KB Date modified: 20/05/2021 21:48 Json Files Open Cancel Submissions Basic Test | John Joe ANSWER 1 Testing Test VIEW SUBMISSION REMOVE ADD NEW SUBMISSION

	 View Submission Questions Differentiating Fractions X Expanding 3 Variable Brackets ✓ Collecting Like Terms X Collecting Like Terms X Differentiating 2 Variable Functions X Differentiating Products ✓ Proving Product Rule X Differentiating 3 Variable Functions X Question (32x^2+1)/((5x-2)^2) Answer Test Override Marking Exit
63	 Test Viewer Questions Differentiating Fractions X Expanding 3 Variable Brackets ✓ Collecting Like Terms X Collecting Like Terms X Differentiating 2 Variable Functions X Differentiating Products ✓ Proving Product Rule X Differentiating 3 Variable Functions X Question (32x^2+1)/((5x-2)^2) Answer Test Exit

Evaluation

Objectives Achieved

I will list out all the objectives I have achieved and evaluate how I done so. I won't reference them as that will be annoying to constantly look back up for.

Index	Business Requirements	Evaluation
1	A login system to differentiate students and teachers.	**COMPLETELY ACHIEVED** I have created a class 'ACCOUNT' that is able to differentiate from a student and teacher via its properties, and then thus the user's choice at the first screen. This is documented in testing at **test #22** and in **Interface Design.**
2	A set of question templates that increase in difficulty. They should be made from the start of A level to the hardest areas (calculus is a focus).	**COMPLETELY ACHIEVED** I have created a set of templates within the DATA_HANDLER class that contain 3 different questions that are examples of that template. The system allows the teacher to choose from these templates, and edit them if they so please. The template currently includes 'Simplification' and 'Differentiation' systems. Differentiation reaches university level difficult with multivariable calculus, whilst simplification is more focused on lower years in ks3-ks5. These templates can be added to if I add more features to the system, but these two different systems allow me to create a number of them that focus of different areas of math – and thus varying levels of difficulty. This aspect is shown in **Interface Design** and in **tests #25-27**.
3	A basic method of solving algebraic questions. This includes simplification and differentiation.	**COMPLETELY ACHIEVED** This was undoubtedly the hardest to achieve, as it required to create an entire expression tree system from scratch. My first two prototypes (1&2) are entirely focused on this system, which has been called 'backend maths'. I first started with basic simplification function, ability to input expressions with strings, and then I moved onto differentiation in prototype 3 when I had built a sturdy enough foundation. This required numerous functions to be created in the 'OPTIMISER' class, each checking for a different case to simplify. After over 2000 lines of code it had reached a level of maturity that can handle general input and solve certain types of questions.

		Its capabilities in its near entirety have been tested on **tests #1-21**.
4	Design a functional user interface.	**COMPLETELY ACHIEVED** This is a more ambiguous **br**, but it basically involved me creating a user interface that is traversable. In **interface design** I showed the entirety of the interface I created, which involved a system of Groupboxes to traverse around. The user interface serves its purpose of encapsulating my first two prototypes (backend maths), and allowing the user to create, serve, and mark tests.
5 FT*	Create a question file creator area.	**COMPLETELY ACHIEVED** This entailed me to create an interface area where I can view the available templates I have, the questions I have created, and have the ability to export these as a JSON file that can then be viewed from a student. I have demonstrated this with **tests #23-24 and 28-37.** It has also been documented in **Interface design.**
6 FS*	Create a question answer area.	**COMPLETELY ACHIEVED** I needed to create an interface for the students to answer questions given by the teacher. This was quite easy, as I could just copy the design from the teacher area and modify it a bit. The ability to export was changed, but its underlying function remained the same; this allows the students to export and then re-add their tests if they want to do it over a couple sessions. I have demonstrated this **br** in **tests #38-49** and in **interface design** where I show an example of how someone would use the system.
7 FT*	Create a question marking area.	**COMPLETELY ACHIEVED** This needed a couple new interface designs to be made working. The marking area was focused on, where I colour marked each different question to easily show which are wrong, right, or indetermined. I also made it easy to manage the marked tests, much like the other two areas, as it builds off the same systems. This continuity allows for ease of use for the system, I believe, as there is very little varying interface for the similar systems. This area is displayed and tested in **tests #50-62** and I**nterface Design**.

8 FT*	Allow teachers to export results to a spreadsheet.	**Partially Achieved** I was not able to achieve the spreadsheet export system due to time constraints, however the teacher is still able to export the data in the form of **JSON**. This JSON can be viewed by the student and is shown in **test #63** and in **Interface Design.** It may not achieve the **br** outright, but it is still able to export some form of data for the student.

FS dictates that this BR is for the student module. **FT** dictates the teacher module.

Feedback

9	The program must be easy to use.	**COMPLETELY ACHIEVED** **TEACHER:** "I found the interface to be logical and easy to follow. There were no sudden changes in style, and the methods of creating and marking questions were easy to understand. I particularly found the colour coding of the marking to make my life easier, as well as the preview function for the questions. The exit buttons made traversing easy if I made a mistake, too. However, I would sometimes run into a situation where the program would crash, but this was rare and normally due to me inputting a question with wrong syntax." **STUDENT:** "I liked how I could save my progress and come back whenever I liked. The interface was easy to follow once I understood how it worked, and the method of moving through questions was intuitive and fast. I would like to say that it may be beneficial to have some sort of method area, like a place where I could show how I got to my answer. I understand why this is not here now, but I think it may be nice for teachers to have." **How could this outcome be improved?** The issue with crashing is due to the handling of wrong inputs in for the backend maths system. Currently I have made few circumventions to stop the program from crashing when this occurs, so I need to add checks on the inputs of the user so that they are notified of their wrong input, instead of having the program crash. As for the student, the 'method' area was an idea I had for a while. However, the reason I never implemented it was that it would have added extra complexity to an already complex project. It would require me to create a 'mark' system, where instead of just being right or wrong, the student can get marks for method. This is impossible to computationally mark, as that would require machine

		learning that I do not have the capability to procure. However, it still seems like a system that could be thought out to complement the marking, so that in the case where the student is wrong, they could still get 50% of the marks for their method. This would need the teacher to mark more on their part, though, and creates a conflict with the reason I created this program in the first place.