Software Engineering 1
SIS 2075
Mini project

Members:
Ghingut M. Oomar (2016304)
Kholeepa B. Sadiyah (2014254)
Rajanah Mrishana (2011318)

Date: 22/04/2022

Table of contents

1.Introduction

In this report we will document the program MathCalc and explain the process behind its conception. It is first and foremost a calculator made to facilitate certain mathematical calculations. We will discuss about its features and functionalities in the next section.

2.Design Requirements and Structure

2.1.Base Program:

The base MathCalc program was conceptualised to contain the following features:

1. 3 types of mathematical calculations as: Area, volume and perimeter calculation for 2D and 3D shapes, Matrix computation and Polynomial operations.
2. Log files for each of one of the above types, to facilitate testing, and to give the user a way to see previously done calculations.
3. Menus for each sections, with a prompt after executing a calculation, so that the user can do repeated calculations in a specific section.
4. A main menu giving access to the all sections implemented, as well as to view the log files.

2.2.Base Features:

The following are the base feature set and functionality of each module to be implemented.

2.2.1 Area, volume and perimeter module:

This module covers formulas and techniques to allow the user to work with 2D plane shapes and 3D solid shapes.

It provides a selection of 6 different 2D shapes and 3D shapes, mainly:

2D shapes:

- Square
- Rectangle
- Circle
- Triangle
- Rhombus
- Parallelogram

3D shapes:

- Cube
- Cuboid
- Pyramid
- Cylinder
- Cone
- Sphere

For the 2D plane shapes section, the calculator will have the user input details about length, height, radius or angles for the different shapes as required for each and will then return the area and perimeter of the particular shape which is being used.

The same input principle is applied for 3D solid shapes whereby each of the latter will have the user input the parameters as required for calculations. This section will return the surface area and volume of the particular shape.

All inputs will be saved to the respective log text file.

2.2.2 Matrix Calculations:

This module computes the following operations:

- Matrix Addition: It takes two matrices, adds them and displays the sum.
- Matrix Subtraction: It takes two matrices, subtracts them and displays the difference.
- Matrix Scalar Multiplication: It takes a matrix and a real number and displays their product. Each entry in the matrix is multiplied by the given scalar.
- Matrix Multiplication: It takes two matrices, multiplies them and displays their product given that it satisfies the condition, I.e., if the number of rows in second matrix matches the number of columns the first matrix.
- All inputs and outputs are saved into the matrix log file.

2.2.3 Polynomial Operations:

This module shall compute the following:

- Polynomial Addition - Take two polynomials, add them and display the result.
- Polynomial Subtraction - Take two polynomials, subtract them and display the result.
- Polynomial Multiplication - Take two polynomials, multiply them and display the result.
- Polynomial Division - Take a polynomial as dividend, and a second as divisor, divide the two, and return quotient and remainder.
- Quadratic Roots - Take a quadratic expression and calculate its determinant and roots and display the latter two.
- Save all inputs and outputs from the different calculations to its respective log file.

2.3.Additional features:

These are additional features that will be considered upon completion of the base program. They are not worked on at the start as time to implement the base program can dictate whether we have time to implement these.

2.3.1 Complex Number Calculations:

This modules enables user to carry operations with complex numbers. It includes 4 sections, mainly:

- Complex numbers addition - It takes two complex numbers, perform addition and displays the sum.
- Complex numbers subtraction - It takes two complex numbers, subtract them and then displays the difference.
- Complex numbers multiplication - It takes two complex numbers, multiply them and displays the product result.
- Complex numbers division - It asks user to input two complex numbers, performs division using the first input as dividend and second input as divisor and finally returns and displays the resulting quotient.

All inputs and outputs will be saved to log file.

2.3.2 Inverse, Transpose and Determinant of Matrix:

These sections shall take as input a matrix, then compute the result as per the title of the section.

- Matrix Transpose: It takes a matrix, interchanges its rows into columns or columns into rows and displays it.
- Matrix Determinant: It takes a square matrix, computes and displays its determinant.
- Matrix Inverse: It takes a square matrix, calculates its determinant. If determinant is non-zero, it works out its cofactors, transposes the matrix of cofactors, divides it by its determinant and displays the inverse of the matrix. Else, if the condition is not satisfied, I.e., determinant is zero, it displays "Matrix has no inverse."

2.3.3 Area Under Curve -

MathCalc_Ar_Vol shall offer a section which allows the user to work with polynomial curve and expressions having degree between 2 to 5, inclusive, to calculate the area under the curve. The input required consists of the equation of the curve (degree 2 to 5 inclusive), limits/bounds within which the area is to be found, and the accuracy required. It shall calculate the output using Simpson's Rule and trapezoidal rule.

2.3.4 Scientific Calculator:

The scientific calculator should offer the following numerical operations:

- Addition
- Subtraction
- Multiplication
- Division
- Square Root
- Power
- 10^x
- Factorial
- Log10(x)
- Exponential
- Sin(x)
- Cos(x)
- Tan(x)
- Cosec(x)
- Cot(x)
- Sec(x)
- Sin Inverse
- Cos Inverse
- Tan Inverse
- Modulus
- Degree to Radian
- Radian to Degree

3.Implementation and Git Commands:

3.1 File Structure Creation

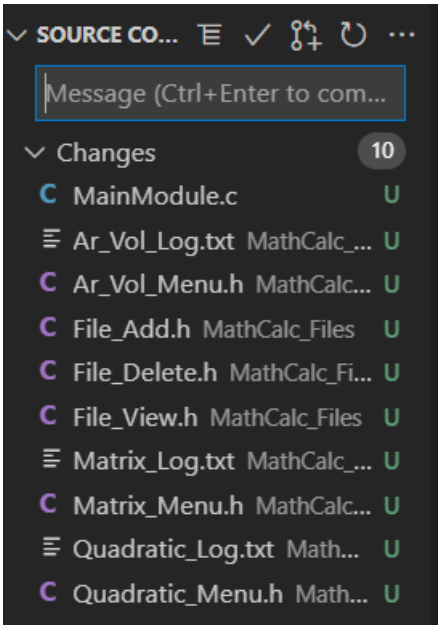
Based on our strategy, we will first create a basic folder and file structure to facilitate visualisation and to kickstart the development phase. We will use Git Bash terminal to do so as follows:

```
Admin@LAPTOP-BP2H6F7I MINGW64 ~
$ cd C:

Admin@LAPTOP-BP2H6F7I MINGW64 /c
$ cd Users/Admin/Desktop/Uni\ tings/Y2\ S1/

Admin@LAPTOP-BP2H6F7I MINGW64 ~/Desktop/Uni tings/Y2 S1
$ md SE_Project_Group1
bash: md: command not found

Admin@LAPTOP-BP2H6F7I MINGW64 ~/Desktop/Uni tings/Y2 S1
$ mkdir SE_Project_Group1

Admin@LAPTOP-BP2H6F7I MINGW64 ~/Desktop/Uni tings/Y2 S1
$ cd SE_
SE_G1_WS.code-workspace  SE_Project_Group1/

Admin@LAPTOP-BP2H6F7I MINGW64 ~/Desktop/Uni tings/Y2 S1
$ cd SE_Project_Group1/

Admin@LAPTOP-BP2H6F7I MINGW64 ~/Desktop/Uni tings/Y2 S1/SE_Project_Group1
$ mkdir MathCalc

Admin@LAPTOP-BP2H6F7I MINGW64 ~/Desktop/Uni tings/Y2 S1/SE_Project_Group1
$ cd MathCalc/

Admin@LAPTOP-BP2H6F7I MINGW64 ~/Desktop/Uni tings/Y2 S1/SE_Project_Group1/MathCalc
$ md MathCalc_Matrix MathCalc_Quadratic MathCalc_Ar_Vol MathCalc_Files
bash: md: command not found

Admin@LAPTOP-BP2H6F7I MINGW64 ~/Desktop/Uni tings/Y2 S1/SE_Project_Group1/MathCalc
$ mkdir MathCalc_Matrix MathCalc_Quadratic MathCalc_Ar_Vol MathCalc_Files
```

First, we entered the directory C:\Users\Admin\Desktop\Uni tings\Y2 S1 to create the project folder SE_Project_Group1.

Inside of it, we then used mkdir to create the MathCalc folder which is the main folder that will be uploaded to GitHub, and that will contain all modules. We used cd again to enter that folder, and then created all module folders using mkdir.

On Visual Studio Code (VSC), we then created the necessary .h files, .c and .txt files using the file manager on VSC directly.



3.2 Git and git commands:

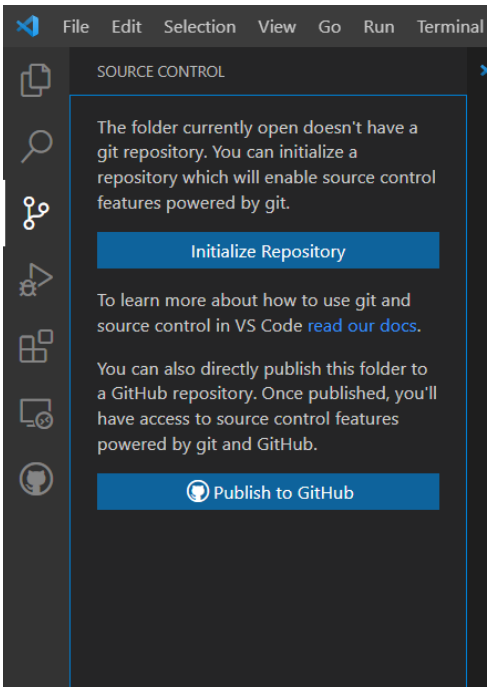
To facilitate and streamline development we are using git and github for versioning and source control.

These are the git / functionalities commands/functionalities we used to achieve this.

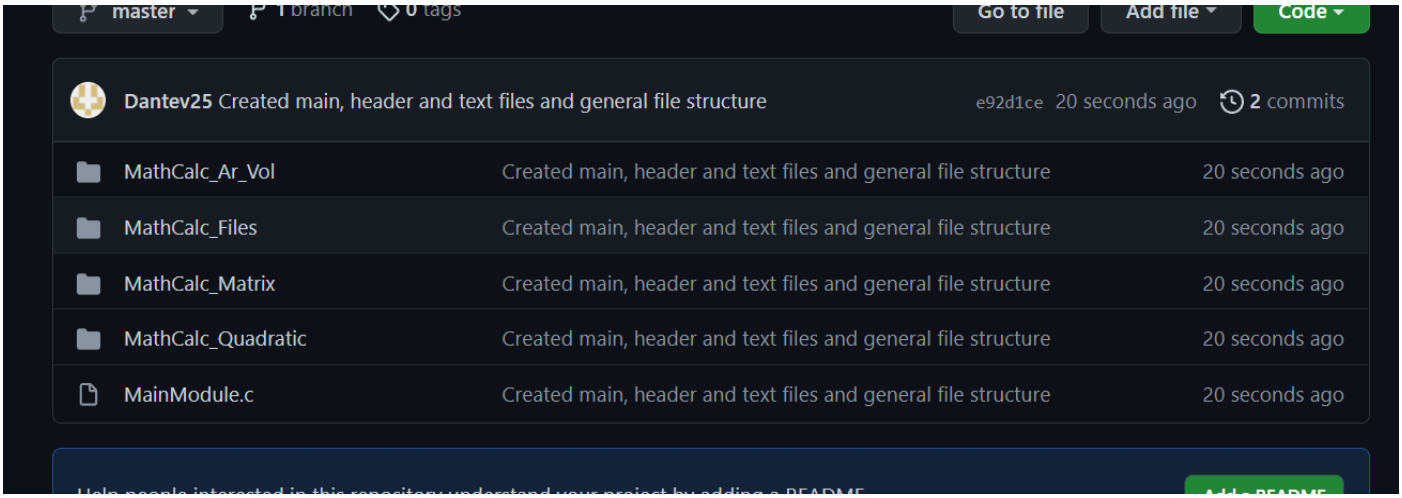
NOTE: As VSC is a powerful code editor, it has strong github integration. We shall use the VSC implementation of git and GitHub instead of using git code through console where it facilitates development.

1. Creating the repository:

Creating the repository is very easy using VSC:



We initialize the repository using the prompt on source control tab of VSC, then publish the repository. It will then appear on the GitHub account of the member who created it, in this case, Mr Ghingut's (alias Dantev25)



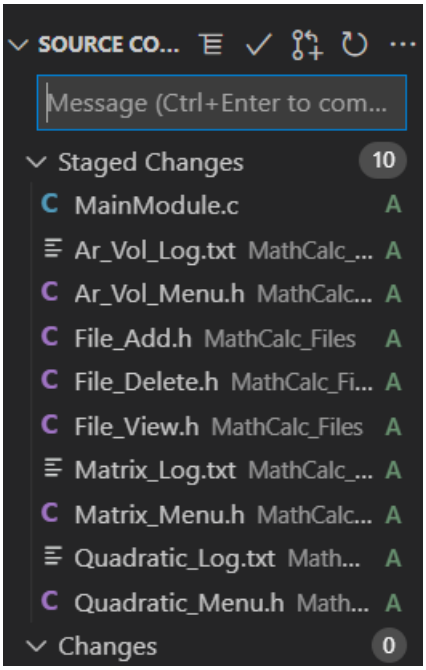
Now all files in this project will be tracked by GitHub as it has been successfully published.

2. Git Clone - For other members to get access to this created repository, they have to clone the repository to their local pc using commands as follows:

```
MINGW64:/c/Users/dell/OneDrive/Desktop/University/y2s1/SE_Project_Group1

dell@DESKTOP-4MM6U9O MINGW64 ~
$ cd C:/Users/dell/OneDrive/Desktop/University/y2s1/SE_Project_Group1

dell@DESKTOP-4MM6U9O MINGW64 ~/OneDrive/Desktop/University/y2s1/SE_Project_Group
1
$ git clone https://github.com/Dantev25/MathCalc.git
Cloning into 'MathCalc'...
remote: Enumerating objects: 9, done.
remote: Counting objects: 100% (9/9), done.
remote: Compressing objects: 100% (7/7), done.
remote: Total 9 (delta 0), reused 9 (delta 0), pack-reused 0
Receiving objects: 100% (9/9), done.
```

Here we find that Mrs. Kholeepa created the SE_Project_Group1 folder, and cloned the repository which contained folder MathCalc (main folder containing all modules and main program) inside. After it has successfully been cloned for both members, all members can now easily use the same repository for source control.

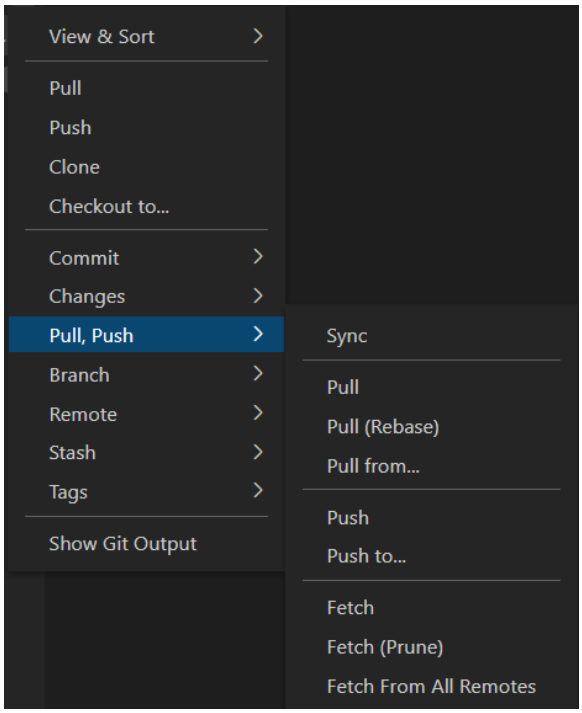
3. Git Stage and Commit - Once major changes have been made to files, and they are considered safe to push to GitHub. These changes are staged together as long as they are in the same workflow, so that the commit message represents the changes to the group using the VSC function:



To commit, a message is simply written inside the text box, and all the files in this staged section will be pushed to GitHub using the tick on the toolbar.

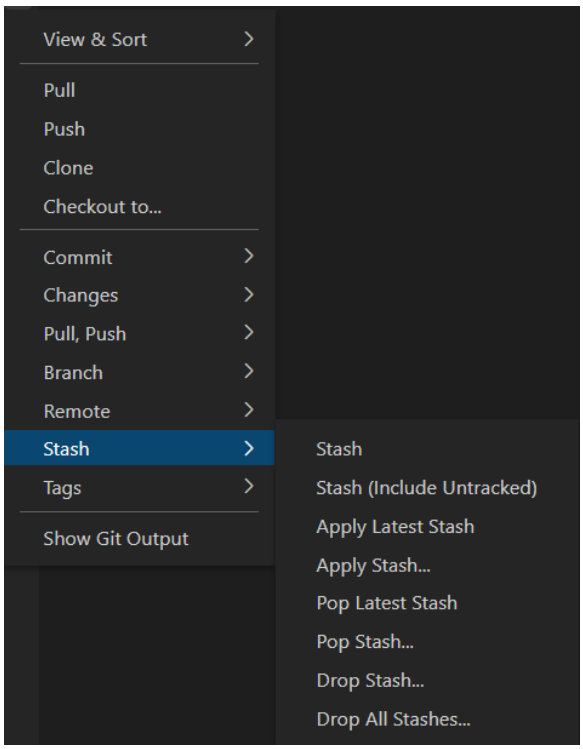
4. Git Push, Pull and Sync- To push staged changes onto the repository, we use the push function, and to download code that has been committed by other members, we can use pull.

However, to publish changes onto the repository, we must use a function called sync, which synchronizes our version of the source code with the version on GitHub. It also pulls all changes that have not yet been pulled. We can find all these commands using the menu on source control:



From this menu, we find that we can do all the aforementioned commands, and more.

5. Git Stash - In case we have a lot of staged/changed files pending, we usually will not be able to pull as there will be merging issues between master branches. However, if we are not ready to push these staged changes, or commit the changed ones, we can use the stash function to temporarily store these files for later use. Then we can pull the files from repository, after which we can just pop the stash using the menu below. All our files will be put in their respective categories again in VSC Source Control.



6. Git Status - Git status gives practically the same information as source control, but in more detail:

```
PS C:\Users\Admin\Desktop\Uni tings\Y2 S1\SE_Project_Group1\MathCalc> git status
On branch master
Your branch is up to date with 'origin/master'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   MathCalc_Matrix/Matrix_Determinant.h
        modified:   MathCalc_Matrix/Matrix_ScalarMultiplication.h

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   MathCalc_Polynomial/Polynomial_Addition.h
        modified:   MathCalc_Polynomial/Polynomial_Multiplication.h
        modified:   MathCalc_Polynomial/Polynomial_QuadRoots.h
```

It gives us the status of our branch, whether it is up to date, or we have commits we haven't pulled yet. Here, it specifically says our master branch is up to date, so we don't need to pull any commits.

We can easily see what files have been staged, and which files are changed, but not staged.

7. Git Log - This command displays the log file of the commits published to GitHub as shown below:

```
commit d5f09650b793c09a86fe9c9b94b5cef4294cb39b
Author: Dantev25 <oomargingo@gmail.com>
Date:   Mon Apr 18 23:01:40 2022 +0400

    main report

commit b12e19a162520a8296dac1190b2066e144482577
Merge: d550a35 2d722c2
Author: Mrishana <95587445+Mrishana@users.noreply.github.com>
Date:   Mon Apr 18 21:16:56 2022 +0400

    Merge pull request #6 from Dantev25/Scientific_Calc
```

As seen above, it gives us the commit code, the author, date and commit message, so we can easily identify who committed what at what time.

8.Git Branch:

Git branch is normally used when we want to try new methods, or add new features, without affecting a stable master program. The branches are independent of the master branch, and will not mess it up in case of bugs in the former. In the development of MathCalc, we will use branches when implementing or testing new and additional features, as well as for testing purposes.

Here, we will use git commands to create, publish and merge the branch as is below:

```
PS C:\Users\Admin\Desktop\Uni tings\Y2 S1\SE_Project_Group1\MathCalc> git branch -c polydiv_improvement
PS C:\Users\Admin\Desktop\Uni tings\Y2 S1\SE_Project_Group1\MathCalc> git branch
* master
  polydiv_improvement
PS C:\Users\Admin\Desktop\Uni tings\Y2 S1\SE_Project_Group1\MathCalc> git checkout polydiv_improvement
Switched to branch 'polydiv_improvement'
Your branch is up to date with 'origin/master'.
```

Here the branch polydiv_improvement has been created using the command git branch –c "filename" and we can see that it has been successfully created using git branch. Using **git checkout poly-div_improvement**, we can then switch from the master branch to this branch.

After finishing all the necessary work on this branch, the branch is then uploaded and merged with the master branch using command **git push origin head** as shown below:

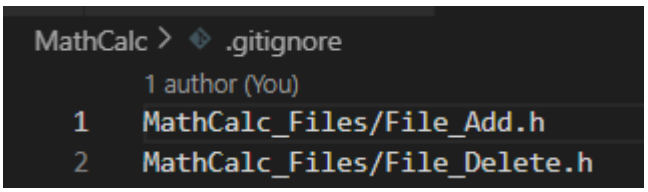
```
PS C:\Users\Admin\Desktop\Uni tings\Y2 S1\SE_Project_Group1\MathCalc> git push origin head
Enumerating objects: 203, done.
Counting objects: 100% (156/156), done.
Delta compression using up to 8 threads
Compressing objects: 100% (62/62), done.
Writing objects: 100% (110/110), 66.03 KiB | 4.40 MiB/s, done.
Total 110 (delta 77), reused 67 (delta 46), pack-reused 0
remote: Resolving deltas: 100% (77/77), completed with 28 local objects.
remote:
remote: Create a pull request for 'polydiv_improvement' on GitHub by visiting:
remote:      https://github.com/Dantev25/MathCalc/pull/new/polydiv_improvement
remote:
To https://github.com/Dantev25/MathCalc.git
 * [new branch]      head -> polydiv_improvement
PS C:\Users\Admin\Desktop\Uni tings\Y2 S1\SE_Project_Group1\MathCalc>
PS C:\Users\Admin\Desktop\Uni tings\Y2 S1\SE_Project_Group1\MathCalc> git merge
Already up to date.
PS C:\Users\Admin\Desktop\Uni tings\Y2 S1\SE_Project_Group1\MathCalc> git checkout master
Switched to branch 'master'
Your branch is behind 'origin/master' by 1 commit, and can be fast-forwarded.
  (use "git pull" to update your local branch)
PS C:\Users\Admin\Desktop\Uni tings\Y2 S1\SE_Project_Group1\MathCalc> git merge polydiv_improvement
Updating 49d99a3..2b372ff
Fast-forward
 MainTest.c                                |   5 +-
 MainTest.exe                              | Bin 79561 -> 61949 bytes
 MathCalc_Polynomial/Polynomial_Division.h | 122 ++++++++++++++++++++++++++--------
 MathCalc_Polynomial/Polynomial_Log.txt    |  10 +++
 MathCalc_Polynomial/Polynomial_QuadRoots.h |   2 +-
 5 files changed, 102 insertions(+), 37 deletions(-)
```

After the merge, we can easily see how many changes were made. In this example, it shows 5 files changed, 102 insertions and 37 deletions.

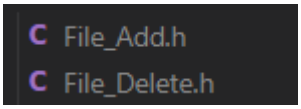
9. Git diff - Git diff is used to see what changes were implemented for all the files in a repository. It shows us the before and after of each change, and is specially useful in debugging changes that are causing problems.

```
diff --git a/MathCalc_Ar_Vol/Ar_Vol_Circle.h b/MathCalc_Ar_Vol/Ar_Vol_Circle.h
index 31db2e1..d131202 100644
--- a/MathCalc_Ar_Vol/Ar_Vol_Circle.h
+++ b/MathCalc_Ar_Vol/Ar_Vol_Circle.h
@@ -1,6 +1,7 @@
 #ifndef AR_VOL_CIRCLE_H_INCLUDED
 #define AR_VOL_CIRCLE_H_INCLUDED

+/*function to find area and perimeter of circle*/
 float circle()
 {
     float r, area, perimeter;
@@ -17,8 +18,8 @@ float circle()
         scanf("%f",&r);
     }

-    area = M_PI * r *r;
-    perimeter = 2 * M_PI * r;
+    area = M_PI * r *r;           //formula for area of circle
+    perimeter = 2 * M_PI * r;     //formula for perimeter of circle

     printf("\nArea of circle = %.2f\n",area);
     printf("perimeter of circle = %.2f\n",perimeter);
@@ -30,10 +31,12 @@ float circle()
         exit(1);
     }
```

10. Git ignore. - gitignore. is a file that is used to set untracked files as invisible to the repository. As such, files which are untracked stay local, and are not tracked by git, and do not appear in source control, no matter how many changes there are inside the file. We use this for temporary files that we do not want to upload to the repository, or files which we are not sure we will use in the repository. An example gitignore. File is shown below:



These two files then appeared greyed out in file management so as to show they are ignored in repository.



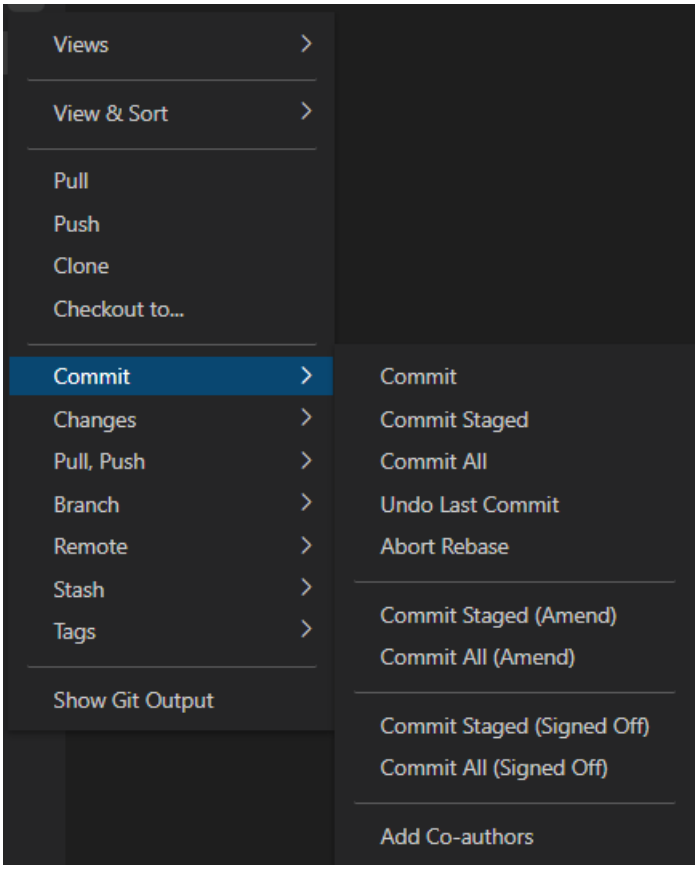
11. Undo commit

Sometimes, when a commit breaks the program, introduces undesirable conflicts or was just accidentally committed, we need to undo it.

There are two methods that can be used to undo commits:

Method 1: Through VSC

In the source control menu of VSC, we have an option to undo commits as shown below:



Using the 'Undo Last Commit' button in the commit sub menu, we can easily undo the last commit that is causing issues.

Method 2: Using commands

Using the 'git reset --soft HEAD~1', we can easily undo the last commit on the log. We shall introduce a test commit, and then remove it using the command as shown below:

```
PS C:\Users\Admin\Desktop\Uni tings\Y2 S1\SE_Project_Group1\MathCalc> git log
commit 90b1fb898e42745cba3339166d9b541ab7dc9fbc (HEAD -> polydiv_improvement, origin/polydiv_improvement)
Author: Dantev25 <oomargingo@gmail.com>
Date:   Thu Apr 21 21:30:31 2022 +0400

    added text to test undo commit
```

```
PS C:\Users\Admin\Desktop\Uni tings\Y2 S1\SE_Project_Group1\MathCalc> git reset --soft HEAD~1
PS C:\Users\Admin\Desktop\Uni tings\Y2 S1\SE_Project_Group1\MathCalc> git log
commit 0d7e4d3de700b8e6c8fd72462296b9c49151d041 (HEAD -> polydiv_improvement)
Merge: accb748 2b372ff
Author: Dantev25 <oomargingo@gmail.com>
Date:   Thu Apr 21 21:29:44 2022 +0400

    Merge branch 'polydiv_improvement' of https://github.com/Dantev25/MathCalc into polydiv_improvement
```

As we can see, after using the command, the commit in the first picture was undone.

3.3 Base Program Implementations:

In this section, we shall discuss the base program implementations as planned in section 2.1 and 2.2.

Libraries called:

- math.h
  - Library which contains necessary mathematical functions like ‘pow’ or ‘fabs’
- stdio.h
  - Standard input output c library
- stdlib.h
- - Standard c library
- ctype.h
  - Contains functions to classify characters, for example, ‘isupper’ which checks if a character is uppercase.
- time.h
  - Provides functions to easily implement time in source code

3.3.1 MathCalc_Ar_Vol

For all shapes, whether 2D or 3D, the user is asked to input required parameters to perform calculations. Then, using corresponding formulas, the program returns the correct output.

Module header files:

- Ar_Vol_Cylinder.h"
- Ar_Vol_Menu.h"
- Ar_Vol_Parallellogram.h"
- Ar_Vol_Pyramid.h"
- Ar_Vol_Rectangle.h"
- Ar_Vol_Rhombus.h"
- Ar_Vol_Sphere.h"
- Ar_Vol_Square.h"
- Ar_Vol_Triangle.h"
- Ar_Vol_Circle.h"
- Ar_Vol_Cone.h"
- Ar_Vol_Cube.h"
- Ar_Vol_Cuboid.h"

Note:

1) As validation, all inputs involving lengths, that is, length, breadth, height and radius should be greater than zero (positive integers only). Conditions are written for each one of them to make sure that the user inputs a correct and valid value. Examples for rectangle and circle are shown below.

Rectangle:

```
while((l <= 0)||(b <= 0)){
    if(l <= 0){
        printf("\nCan only input positive integer for length!\nInput again!\n");
        printf("Length = ");
        scanf("%f",&l);
    }

    else if(b <= 0){
       printf("\nCan only input positive integer for breadth!\nInput again!\n");
       printf("Breadth = ");
       scanf("%f",&b);
    }
}
```

Circle:

```
while(r <= 0){
    printf("\nCan only input positive integer for radius!\nInput again: ");
    scanf("%f",&r);
}
```

2) Angles are validated to be between 0 and π (the latter two not included), whereby conditions are included in the program to allow user input valid and corrects values. An example is shown below.

```
printf("\nInput value of the angle (in radians): ");
scanf("%f",&x);

while((x <= 0)||(x >= M_PI)){
    printf("\nAngle should have a value between 0 and pi!\nInput again: ");
    scanf("%f",&x);
}
```

3) The library <math.h> in c is used to implement certain mathematical concepts directly. π is used as M_PI which uses its exact value. Concepts like square root, power and cosine are also used as sqrt(), pow() and cos () respectively.

Example: square root of 4 = sqrt(4), x^2 = pow(x,2) and cos(1.75).

Input

The input and formulas used for each of them is now shown below:

- **Square**

Input: length of side, x

Formulas used:

```
area = x * x;
perimeter = x * 4;
```

- **Rectangle**

Input: length b and breadth b

Formulas used:

```
area = l * b;
perimeter = (l + b) * 2;
```

- **Circle**

Input: radius r

Formulas used:

```
area = M_PI * r *r;
perimeter = 2 * M_PI * r;
```

- **Triangle**

A menu is provided which allows user to work with 3 types of triangles:

i. Equilateral triangle

Input: length of side, s

Formulas used:

```
area = (sqrt(3)/4) * (pow(s,2));
perimeter = s * 3;
```

ii. Right – angled triangle

Input: base, height and length of third side

Formulas used:

```
area = 0.5 * base * height;
perimeter = base + height + s;
```

iii. Any other triangle

Input: length of 3 sides of triangle, that is, s1, s2 and s3

Formulas used:

```
s = (s1 + s2 + s3)/2;
area = sqrt(s * (s-s1) * (s-s2) * (s-s3));
perimeter = s1 + s2 + s3;
```

- **Parallelogram**

Input: length of side, base and height

Formulas used:

```
area = base * height;
perimeter = (side + base) * 2;
```

- **Rhombus**

A menu is provided which allows user to decide by which method he would want to carry the calculations for rhombus.

i. By using angle:

Input: length of side and angle x in radians

Formulas used:

To calculate the two diagonals -

```
a = pow(side,2);
d1 = sqrt(a + a - (2 * side * side * cos(x)));
d2 = sqrt((4 * a) - (pow(d1,2)));
```

To calculate area and perimeter -

```
area = (d1 * d2)/2;
perimeter = side * 4;
```

ii. By using diagonal:

Input: length of side and diagonal 1 (d1)

Formula used:

To calculate diagonal d2 -

```
d2 = sqrt((4 * pow(side,2)) - (pow(d1,2)));
```

Area and perimeter are calculated using the same formulas as that of the first method (that is, by using angle) .

- **Cube**

Input: length of side, x

Formulas used:

```
sa = (x * x) * 6;
vol = x * x * x;
```

Note: sa = surface area

Vol = volume

- **Cuboid**

Input: length h, breadth b and height h

Formulas used:

```
sa = ((b * l) + (l * h) + (h * b)) *2;
vol = l * b * h;
```

- **Pyramid**

Input: length l, breadth b of base and height h of pyramid

Formulas used:

```
x = l/2;
y = b/2;

sa = (l * b) + (l * sqrt((pow(y,2)) + (pow(h,2)))) + (b * sqrt((pow(x,2)) + (pow(h,2))));
vol = (l * b * h)/3;
```

- **Cylinder**

Input: radius r of circular base, height h of cylinder

Formulas used:

```
sa = (2 * M_PI * r * r) + (2 * M_PI * r * h);
vol = M_PI * r * r *h;
```

- **Cone**

Input: radius r of circular base, height h of cone

Formulas used:

```
sa = M_PI * r * (r + sqrt((pow(h,2)) + (pow(r,2))));
vol = M_PI * r * r * (h/3);
```

- **Sphere**

Input: radius r

Formulas used:

```
sa = 4* M_PI * r * r;
vol = (4.0/3) * M_PI * r * r * r;
```

A sample input and output is now shown below for rhombus (2D shape) and for cone (3D shape) when the program is run.

Rhombus:

```
                RHOMBUS

By which method would you like to carry the operations for rhombus:
1. By using angle
2. By using diagonal

Input your choice: 1
Input the length of the side of the rhombus: 4.53

Input value of the angle (in radians): 2.35

Calculated diagonal 1 = 8.36
Calculated diagonal 2 = 3.49

Area of rhombus = 14.60
Perimeter of rhombus = 18.12
```

Cone:

```
                CONE

Enter the radius of the circular base of the cone: 5.78

Enter the height of the cone: 9.8

Surface area of cone = 311.55
Volume of cone = 342.85
```

Entry in Ar Vol Log File:

Each time a calculation is executed for any particular shape, the details are entered and stored in a log file (Ar_Vol_Log.txt).

- The shape which was chosen to execute the calculations is stored as "Shape : <shape name>"
- The inputs required to perform operations for that specific shape are entered and stored.
- The output is also entered and stored, that is, area and perimeter for 2D shapes and surface area and volume for 3D shapes)

An example for circle is shown below:

```
/*Entry in log file*/
fprintf(fptr,"-----------------------------------------------\n");
fprintf(fptr,"Executed on: %s",ctime(&t));
fprintf(fptr,"Shape : Circle\n");
fprintf(fptr,"Input:\n\tradius = %.2f\n",r);
fprintf(fptr,"Output:\n\tArea = %.2f\n\tPerimeter = %.2f\n\n",area,perimeter);
```

- We shall also store the date and time at which each particular calculation is carried out using the <time.h> library in c.

```
time_t t;   // not a primitive datatype
time(&t);
```

All calculations entries for this module can be reviewed in the log file.

Below is the log entry for triangle and cone which was performed earlier as an example:

```
-----------------------------------------------
Executed on: Wed Apr 20 22:33:11 2022
Shape : Rhombus
Input:
    length of side = 4.53, angle (in radians) = 2.350
Output:
    Calculated diagonal 1 = 8.36, calculated diagonal 2 = 3.49
    Area = 14.60
    Perimeter = 18.12

-----------------------------------------------
Executed on: Wed Apr 20 22:39:02 2022
Shape : Cone
Input:
    radius of circular base = 5.78, height = 9.80
Output:
    Surface area = 311.55
    Volume = 342.85
```

Ar_Vol_Menu:

A menu was created for this module which provides a selection including all the 13 sections of MathCalc_Ar_Vol (additional area under graph feature included). It was implemented using switch/case function in c after having asked user to input his choice. An example for the two first cases is shown:

```
switch (choice)
{
case 1:
    printf("\n\t\tSQUARE\n\n");
    square();
    break;

case 2:
    printf("\n\t\tRECTANGLE\n\n");
    rectangle();
    break;
```

After operations are completed, the menu asks user whether or not they want to calculate again. As long as they choose 'Y' which stands for yes, the menu displays the selection list.

3.3.3 MathCalc_Matrix

User shall be able to specify the order of the matrix. User shall be able to add, subtract, multiply the matrices as well as finding the scalar multiplication of matrix/matrices.

Matrix_Menu.h File calls all the matrix operation files.

Module files:

- Matrix_Addition.h
- Matrix_Multiplication.h
- Matrix_ScalarMultiplication.h
- Matrix_Subtraction.h
- {Matrix_Determinant.h}
- {Matrix_Inverse.h}
- {Matrix_Transpose.h}

Note: Those between {} are additional feature files.

Input:

The user is asked to input the number of rows, and the number of columns. Then he will be asked to input the elements of the matrix as per their index:

```
for (i = 0; i < r; ++i){
  for (j = 0; j < c; ++j) {
    printf("Enter Element a[%d][%d]: ", i + 1, j + 1);
    scanf("%d", &a[i][j]);
  }
}
```

It uses two for loops to iterate between rows and columns, and allows input to the desired matrix.

```
Enter the Number of Rows : 2

Enter the Number of Columns : 3

Enter Elements of 1st matrix:

Enter Element a[1][1]: 11
Enter Element a[1][2]: 123
Enter Element a[1][3]: 44
Enter Element a[2][1]: 7
Enter Element a[2][2]: 99
Enter Element a[2][3]: 2
```

Computation method:

Addition

A nested loop is used in adding two matrices, we add the elements in each row and column to the respective elements in the row and column of the next matrix.

```
// adding two matrices
for (i = 0; i < r; ++i){
  for (j = 0; j < c; ++j){
    sum[i][j] = a[i][j] + b[i][j];
  }
}
```

Subtraction

A nested loop is used in subtracting two matrices, we subtract the elements in each row and column of the second matrix from the respective elements in the row and column of the first matrix.

```
// Subtracting two matrices
for (i = 0; i < r; ++i){
  for (j = 0; j < c; ++j){
    dif[i][j] = a[i][j] - b[i][j];
  }
}
```

Scalar Multiplication

The function for this operation multiplies every element of the matrix by the scalar input and returns it to the same matrix. The process is repeated until rows equal i and the scaled matrix is obtained.

```
for(rows = 0; rows < i; rows++){
    for(columns = 0; columns < j; columns++){
        Multiplication[rows][columns] = Number * Multiplication[rows][columns];
    }
}
```

Multiplication

A nested loop is used in multiplying the first and second matrices and storing it in sum. In turn this gets stored in the 2D array ‘multiply’ and sum is initialized to zero. This is repeated until c equals m and product matrix is computed.

```
for (c = 0; c < m; c++) {
  for (d = 0; d < q; d++) {
    for (k = 0; k < p; k++) {
      sum = sum + first[c][k]*second[k][d];
    }

    multiply[c][d] = sum;
    sum = 0;
  }
}
```

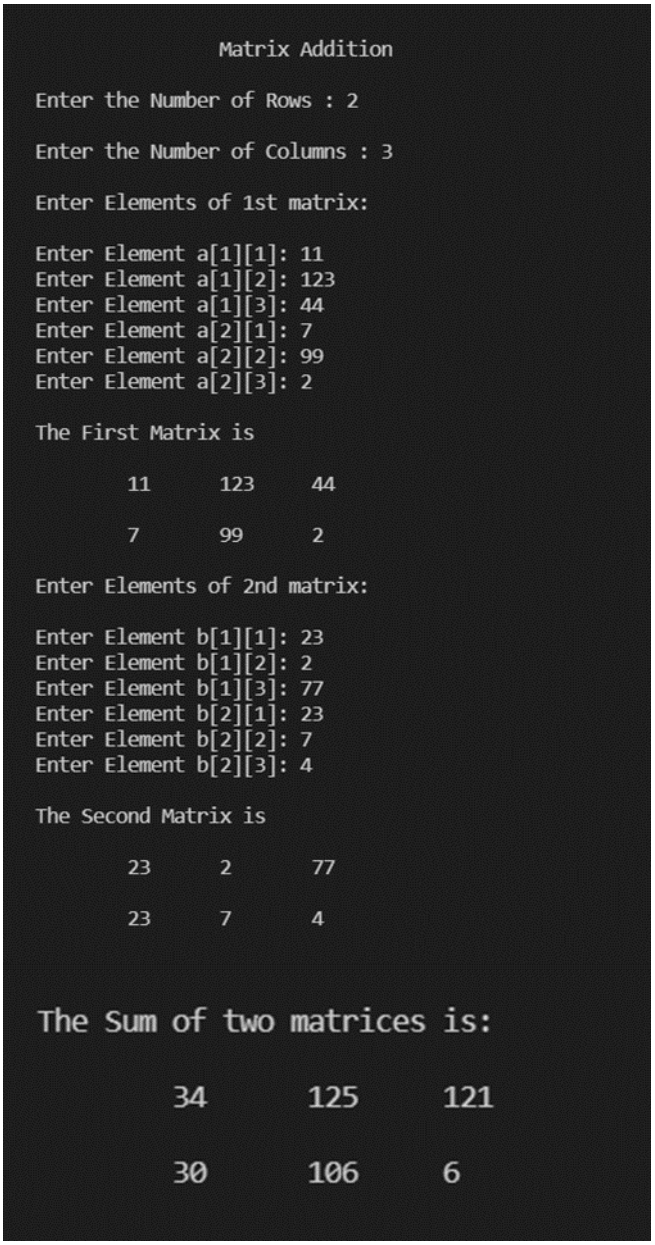
Entry in Matrix log file

Every operation saves to the Matrix_Log.txt . A few samples are shown below

```
Executed on: Tue Apr 19 20:43:20 2022

    Matrix Operation : Multiplication

Input:
First Matrix:

    12  23  1

    6   7   41


Second Matrix:

    24

    54

    122


Output:

    1652

    5524
```

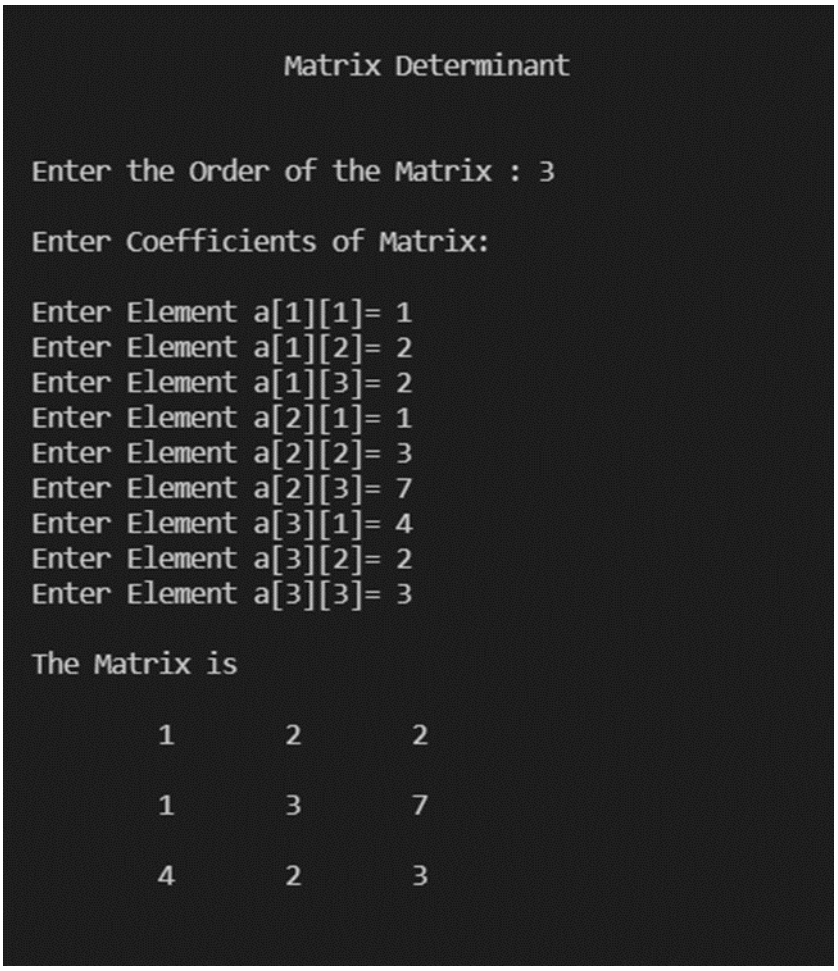
```
Executed on: Tue Apr 19 20:45:51 2022

    Matrix Operation : Determinant

Input:

    1   2   2

    1   3   7

    4   2   3


Output:

Determinant of the Matrix is 25.
```

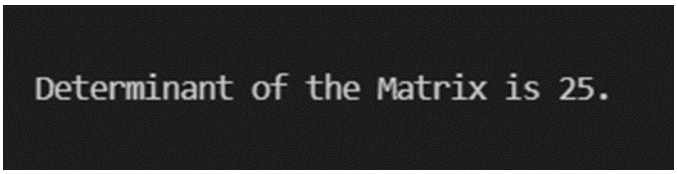
Sample input and output



```
                Matrix Multiplication

For the First Matrix
Enter the Number of Rows : 2

Enter the Number of Columns :3

Enter Elements of first matrix
Enter Element a[1][1]: 12
Enter Element a[2][1]: 6
Enter Element a[2][2]: 7
Enter Element a[2][3]: 41

The First Matrix is

        12      23      1

        6       7       41

For the Second Matrix
Enter the Number of Rows : 3

Enter the Number of Columns :1

Enter Elements of second matrix
Enter Element a[1][1]: 24
Enter Element a[2][1]: 54
Enter Element a[3][1]: 122

The Second Matrix is

        24

        54

        122
```

```
Output:

    1652

    5524
```


Matrix Determinant
Enter the Order of the Matrix : 3
Enter Coefficients of Matrix:
Enter Element a[1][1]= 1
Enter Element a[1][2]= 2
Enter Element a[1][3]= 2
Enter Element a[2][1]= 1
Enter Element a[2][2]= 3
Enter Element a[2][3]= 7
Enter Element a[3][1]= 4
Enter Element a[3][2]= 2
Enter Element a[3][3]= 3
The Matrix is
1 2 2
1 3 7
4 2 3


Determinant of the Matrix is 25.

```
                Matrix Inverse

Enter the Order of the Matrix : 3

Enter the Elements of the Matrix:
Enter Element a[1][1]= 1
Enter Element a[1][2]= 2
Enter Element a[1][3]= 3
Enter Element a[2][1]= 4
Enter Element a[2][2]= 5
Enter Element a[2][3]= 6
Enter Element a[3][1]= 7
Enter Element a[3][2]= 8
Enter Element a[3][3]= 9

The Matrix is

        1.00    2.00    3.00

        4.00    5.00    6.00

        7.00    8.00    9.00
```

```
Output:


The determinant is zero (0). Matrix has no inverse.
```

3.3.3 MathCalc_Polynomial:

Module files: Module files in this section are all done as header files, which will then be linked to the Polynomial_Menu.h header. This header file will then be linked to the main module c file. The header files created are as follows:

- Polynomial_Menu.h
- Polynomial_Addition.h
- Polynomial_Subtraction.h
- Polynomial_Multiplication.h
- Polynomial_Division.h
- Polynomial_QuadRoots.h

Inputs:

There are two methods used to implement polynomials input in the Polynomial section of MathCalc:

Method 1:

1. Input degree of first polynomial (in case of quadratic roots, only this polynomial is input). This degree is validated to be only positive.
2. Input first polynomial using a for loop which takes input to an array, from constant to increasing degree until the degree in 1. is met. The code sample below shows this input sequence:

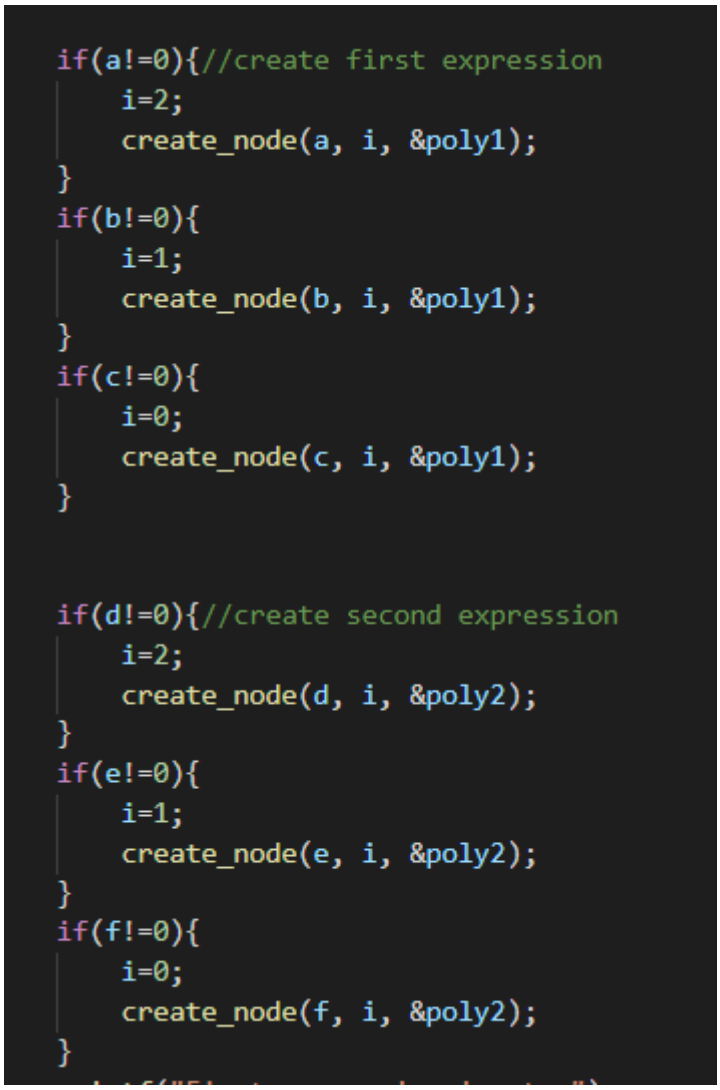
```
57      printf("\nPlease input the degree (highest power) of the first expression:");
58      scanf("%d",&n1);
59      deg = n1;
60      while (n1<=0){
61          printf("\nPlease input the degree (highest power) again, it cannot be negative:");
62          scanf("%d",&n1);
63      }
64      n1++;
65      printf("\nPlease input the first expression (starting from constant and ascending in power):\n");
66      for (i=0;i<n1;i++){
67          if (i==0){
68              printf("Constant = ");
69              scanf("%f",&a[i]);
70          }
71          else if (i==1){
72              printf("X = ");
73              scanf("%f",&a[i]);
74          }
75          else{
76              printf("X^%d = ",i);
77              scanf("%f",&a[i]);
78          }
79      }
```

This input method also uses if statements to better give visualisation to the user for inputting the correct numbers as it displays each power separately as shown below:

```
Please input the degree (highest power) of the first expression:2

Please input the first expression (starting from constant and ascending in power):
Constant = 2
X = 4
X^2 = 5
```

Method 2:

1. This method is based on method one and is used in module Division, in that it takes input to a set of variables using the same practical logic, but instead, it does it from highest degree to the constant.
2. Then, to allow for polynomial division, we then use the two sets of 3 uniquely defined variables, one for dividend and the other one for divisor to push the coefficients onto a linked list.
3. This method allows input for up to degree 2, as the input to the linked list cannot be setup in a loop. This is because the loop causes undefined behavior, and force exits the program. After inputting to the set as shown in 1. the function



```
if(a!=0){//create first expression
    i=2;
    create_node(a, i, &poly1);
}
if(b!=0){
    i=1;
    create_node(b, i, &poly1);
}
if(c!=0){
    i=0;
    create_node(c, i, &poly1);
}


if(d!=0){//create second expression
    i=2;
    create_node(d, i, &poly2);
}
if(e!=0){
    i=1;
    create_node(e, i, &poly2);
}
if(f!=0){
    i=0;
    create_node(f, i, &poly2);
}
```

If the element is not equal to zero, it is pushed onto the poly1 linked list (link list for dividend). Below is the function create node used to input to the link list:

.

.

.

.

.

.

CONTINUED ON NEXT PAGE

```
void create_node(float x, int y,
                 struct Node** temp)
{
    struct Node *r, *z;
    z = *temp;

    // If temp is NULL
    if (z == NULL) {

        r = (struct Node*)malloc(
            sizeof(struct Node));

        // Update coefficient and
        // power in the LL z
        r->coeff = x;
        r->pow = y;
        *temp = r;
        r->next = (struct Node*)malloc(
            sizeof(struct Node));
        r = r->next;
        r->next = NULL;
    }

    // Otherwise
    else {
        r->coeff = x;
        r->pow = y;
        r->next = (struct Node*)malloc(
            sizeof(struct Node));
        r = r->next;
        r->next = NULL;
    }
}
```

- If the list is empty, it allocates memory to the node using malloc, and then updates the power and co-efficient in the linked list.
- It then allocates memory to the next node, changes its pointer to it, and puts its next as null.
- If it is not empty, it just updates as above and creates the new node and sets the pointer to it, and then to null from there.

Computation Methods

Each module has its own specific formula we used to calculate the desired result. As such, we will describe each method below (assuming input is already done and validated):

Polynomial Addition:

As all the terms in the two expressions input have the same index referencing coefficients of the same degree, a simple for loop is used and an addition operation is used to calculate the result:

```
for(k=0;k<deg;k++){
    sum[k] = a[k] + b[k];
}
```

Polynomial Subtraction:

Same principle as for addition, only with a different formula:

```
for(k=0;k<deg;k++){
    Result[k] = a[k] - b[k];
}
```

Polynomial Multiplication:

Here, we will use a nested for loop. The outside loop shall iterate through expression a, and the inner loop through expression b. This in fact allows us to multiply every element in b, with the first in one, and then as the outer loop increments, the same is done for the second element in a with all elements in b. This is repeated until the degree of a is met.

As for the index of the product, as index represents power, the product matrix will take as index the addition of indexes of a and b for the specific elements multiplying. This is because multiplying terms with powers causes the addition of the latter. As such, all a*b multiples which have the same powers will just have their coefficients added as they will be saved at the same index.

```
for (int i=0; i<n1; i++){
    for (int j=0; j<n2; j++){
        prod[i+j] += a[i]*b[j];
    }
}
```

Quadratic Roots:

Here the degree is hardcoded to be 2, as we only need quadratics to find the roots.

We then calculate its determinant, $b^2 - 4ac$ as follows:

```
det = pow(a[1],2)-(4*a[2]*a[0]);
```

We use the function ‘pow’ from library ‘math.h’ to find the square of b and then use mathematical operators to construct the rest of the formula.

Depending on the determinant, we will use different methods to find the root:

```
if (det == 0){
    printf("\nEquation has only one root as b^2 - 4ac = 0\nRoot is at: ");
    root1 = (-a[1] + sqrt(det))/(2*a[2]);
    printf("x = %.2f",root1);
}
else if(det>0){
    printf("\nEquation has two roots as b^2 - 4ac > 0\nRoots are at: \n");      You, 4 days ago • F
    root1 = (-a[1] + sqrt(det))/(2*a[2]);
    root2 = (-a[1] - sqrt(det))/(2*a[2]);
    printf("Root1: x = %.2f",root1);
    printf("\nRoot2: x = %.2f",root2);
}
else{
    printf("\nRoots are imaginary as b^2 - 4ac <0.");
    real = -a[1] / (2 * a[2]);
    img = (sqrt(-det)) / (2 * a[2]);
    printf("\n\nComplex root 1 = %.2f + %.2fi\nComplex root 2 = %.2f - %.2fi\n",real,img,real,img);
}
```

- If determinant is zero, we just calculate the x position of the root with the quadratic formula as there is only one root

  Formula: (-b + square root of determinant)/2a

- If determinant is positive, we then use the quadratic formula again, but one root will be with

  (-b - square root of determinant)/2a

  and the other one with (-b - square root of determinant)/2a

- If the roots are imaginary, that is determinant is negative, we then calculate the real parts and imaginary parts separately. As one complex root is conjugate of the other, we can just output the two roots with opposite operators between real and im parts.

  Formula: Real part = -b /2a

  Im part = Root of (-det) /2a

Polynomial Division

After doing the input, the function divide_poly is called as below:

```
void divide_poly(struct Node* poly1,struct Node* poly2, FILE *fptr){
    // Initialize Remainder and Quotient
    struct Node *rem = NULL, *quo = NULL;
    quo = (struct Node*)malloc(
        sizeof(struct Node));
    quo->next = NULL;
    struct Node *q = NULL, *r = NULL;
    // Copy poly1, i.e., dividend to q
    copyList(poly1, &q);
    // Copy poly, i.e., divisor to r
    copyList(poly2, &r);
    // Perform polynomial subtraction till
    // highest power of q > highest power of divisor
    while (q != NULL&& (q->pow >= poly2->pow)) {        You, now • Unc
        // difference of power
        int diff = q->pow - poly2->pow;
        float mul_c = (q->coeff/ poly2->coeff);
        // Stores the quotient node
        store_quotient(mul_c, diff,quo);
        struct Node* q2 = NULL;
        // Copy one LL in another LL
        copyList(r, &q2);
        // formNewPoly forms next value
        // of q after performing the
        // polynomial subtraction
        formNewPoly(diff, mul_c, q2);

        struct Node* store = NULL;
        store = (struct Node*)malloc(sizeof(struct Node));
        // Perform polynomial subtraction
        sub(q, q2, store);
        // Now change value of q to the
        // subtracted value i.e., store
        q = store;
        free(q2);
    }
```

1. First, the two polynomials are copied onto temporary link lists r and q.
2. q is then used to calculate the division, and its result is stored in the quotient.
3. The remainder of the division is then saved onto q2, which is then used to subtract from the original q. The quotient is then stored in quo.
4. The subtracted value is then saved back onto q, so that the division can happen until there is no more possible division (degree dividend<degree divisor)
5. This sequence repeats until there are no longer possible division due to q being smaller than the divisor.
6. The remainder is saved to rem, as q is the remainder, and the quotient is quo.

Polynomial Log file

All the modules in the polynomial section save inputs and outputs to a log file. An example fot polynomial addition is shown below:

```
fprintf(fptr,"------------------------------------------------------------------------\n");
fprintf(fptr,"\n\nExecuted on: %s",ctime(&t));
fprintf(fptr,"Operation Done: Polynomial Addition\n");
fprintf(fptr,"Inputs: \n");
fprintf(fptr,"\tFirst expression : ");
PolySave(n1,a,fptr);
fprintf(fptr,"\tSecond expression : ");
PolySave(n2,b,fptr);
fprintf(fptr,"Output: ");
PolySave(deg,sum,fptr);
fclose(fptr);
```

First, the time at which the calculation was made is appended. Then all inputs are saved by using PolySave, which takes in an array and saves it to a text file in polynomial format. We then save the sum at the end. The other operations also follow the same principle and format.

```
------------------------------------------------------------------------
Executed on: Fri Apr 15 21:05:04 2022
Operation Done: Polynomial Addition
Inputs:
    First expression : 5.00X^2 + 4.00X + 3.00
    Second expression : 6.00X^2 + 5.00X + 1.00
Output: 11.00X^2 + 9.00X + 4.00
```

Below is some sample screenshots from the text file showing for different sections:

```
------------------------------------------------------------------------
Executed on: Fri Apr 15 21:12:12 2022
Operation Done: Polynomial Multiplication
Inputs:
    First expression : 6.00X^3 + 10.00X^2 + 5.00
    Second expression : 4.00X^2 + 2.00X + 1.00
Output: 24.00X^5 + 52.00X^4 + 26.00X^3 + 30.00X^2 + 10.00X + 5.00
------------------------------------------------------------------------
Executed on: Fri Apr 15 21:17:30 2022
Operation Done: Polynomial Subtraction
Inputs:
    First expression : 8.00X^2 + 7.00X + 5.00
    Second expression : 6.00X^2 + 4.00X + 1.00
Output: 2.00X^2 + 3.00X + 4.00

------------------------------------------------------------------------

Executed on: Fri Apr 15 21:35:16 2022
Operation Done: Polynomial Division
Inputs:
    First expression : 2.00X^2 + 4.00X^1 + 5.00
    Second expression : 3.00X^2 + 2.00X^1 + 1.00
Output:
    Quotient: 0.67
    Remainder: 2.67X^1 + 4.33
```

```
               Polynomial Addition


Please input the degree (highest power) of the first expression:2

Please input the first expression (starting from constant and ascending in power):
Constant = 1
X = 3
X^2 = 6

Please input the degree (highest power) of the second expression:3

Please input the second expression (starting from constant and ascending in power):
Constant = 6
X = 3
X^2 = 7
X^3 = 2

First expression : 6.00X^2 + 3.00X + 1.00
Second expression : 2.00X^3 + 7.00X^2 + 3.00X + 6.00

Sum of these two expressions is: 2.00X^3 + 13.00X^2 + 6.00X + 7.00
```

```
               Polynomial Subtraction


Please input the degree (highest power) of the first expression:3

Please input the first expression (starting from constant and ascending in power):
Constant = 4
X = 6
X^2 = 7
X^3 = 3

Please input the degree (highest power) of the second expression:2

Please input the second expression (starting from constant and ascending in power):
Constant = 1
X = 4
X^2 = 8

First expression : 3.00X^3 + 7.00X^2 + 6.00X + 4.00
Second expression : 8.00X^2 + 4.00X + 1.00

Result of these two expressions is: 3.00X^3 + -1.00X^2 + 2.00X + 3.00
```

```
               Polynomial Multiplication


Please input the degree (highest power) of the first expression:1

Please input the first expression (starting from constant and ascending in power):
Constant = 3
X = 5
Please input the degree (highest power) of the second expression:1
Please input the second expression (starting from constant and ascending in power):
Constant = 4
X = 6

First expression : 5.00X + 3.00
Second expression : 6.00X + 4.00

Product of these two expressions is: 30.00X^2 + 38.00X + 12.00
```

```
                Polynomial Division


Please input the first expression:
X^2 = 2
X = 3
Constant = 4

Please input the second expression:
X^2 = 1
X = 4
Constant = 5
First expression input: 2.00x^2 + 3.00x^1 + 4.00
Second expression input: 1.00x^2 + 4.00x^1 + 5.00

Quotient: 2.00
Remainder: -5.00x^1 - 6.00
```

```
               Quadratic Roots


Please input the quadratic expression (starting from constant and ascending in power):
Constant = 2
X = 3
X^2 = 5

The expressions input is: 5.00X^2 + 3.00X + 2.00

b^2 - 4ac = -31.00

Roots are imaginary as b^2 - 4ac <0.

Complex root 1 = -0.30 + 0.56i
Complex root 2 = -0.30 - 0.56i
```

3.3.4 Function to view log files:

This feature was implemented in the File_View section from the MathCalc_File module.

- It has a menu which provides a selection to view the log files of all the different modules in the mathCalc workspace and it was executed using if/else function in c after having asked user to input his choice.
- Each choice opens its corresponding .txt file in read mode.
- We then have a condition whereby, while the end of file is not reached, it fetches the characters in the file one by one (using fgetc) and proceeds to print them.
- The file is then closed after it has been viewed.

The implementation for viewing Complex_Log.txt is shown below:

```
else if(choice == 4){
    fptr = fopen("MathCalc_Complex/Complex_Log.txt","r");
    if(fptr == NULL){
        printf("\nFile not found\n");
    }
    else{
        printf("\nFile opened in read mode\n\n");
    }
    while((c = fgetc(fptr)) != EOF){
        printf("%c",c);
    }
    printf("\n\n");
    fclose(fptr);
}
```

The same code is used for all parts, except for the relative path of the log file is changed.

3.4 Additional Features:

After implementing the base program successfully, we implemented the following additional features. All the additional features were implemented using branching so as not to bug the main base program.

3.4.1 MathCalc_Complex

A module designed to carry out complex numbers operations was added and this was implemented using branch so as not to mess the main program in the master branch.

A branch ‘ Complex_branch ’ was created as shown:

```
PS C:\Users\dell\OneDrive\Desktop\University\y2s1\SE_Project_Group1\MathCalc> git branch -c Complex_branch
PS C:\Users\dell\OneDrive\Desktop\University\y2s1\SE_Project_Group1\MathCalc> git branch
  Ar_Vol_branch
  Complex_branch
  MathCalc_Files_branch
* master
PS C:\Users\dell\OneDrive\Desktop\University\y2s1\SE_Project_Group1\MathCalc> git checkout Complex_branch
Switched to branch 'Complex_branch'
Your branch is up to date with 'origin/master'.
```

Module files:

- Complex_Addition.h
- Complex_Subtraction.h
- Complex_Multiplication.h
- Complex_Division.h
- Complex_Menu.h

- **Complex number input**

This will be carried out as shown below and used in all sections:

```
printf("Enter a and b where a + ib is the first complex number:\n");
printf("a = ");
scanf("%f",&a.real);
printf("b = ");
scanf("%f",&a.img);

printf("\nEnter c and d where c + id is the second complex number:\n");
printf("c = ");
scanf("%f", &b.real);
printf("d = ");
scanf("%f", &b.img);
```

- **Complex number addition**

The addition operation is carried out by adding the real and imaginary part of the first complex to the real and imaginary part of the second complex respectively as shown:

```
c.real = a.real + b.real;
c.img = a.img + b.img;
```

- **Complex number subtraction**

The subtraction operation is carried out by subtracting the real and imaginary part of the second complex from the real and imaginary part of the first complex respectively as shown:

```
c.real = a.real - b.real;
c.img = a.img - b.img;
```

- **Complex number multiplication**

The product of the two complex numbers is calculated as shown below:

```
c.real = (a.real*b.real) - (a.img*b.img);
c.img = (a.img*b.real) + (a.real*b.img);
```

The real part of the product of the two inputs is calculated by subtracting the product of the imaginary part of both numbers from the product of the real part of the numbers.

The imaginary part is calculated by multiplying the imaginary part of the first one to the real part of the second one, and then, this is added to the product of real part of first input to imaginary part of second input.

- **Complex number division**

After user has been asked to input the complex numbers, we have a validation condition whereby, if the real and imaginary parts of the second complex number are both zero, user is asked to input again since division by zero is not allowed. Three parameters x, y and z are then calculated as follows:

```
x = a.real*b.real + a.img*b.img;
y = a.img*b.real - a.real*b.img;
z = b.real*b.real + b.img*b.img;
```

The modulus of x and z, and y and z are hence calculated.

1) If the remainder of both is zero, the real part of the resultant quotient is x/z while the imaginary part is y/z.
2) If the remainder of modulus of x and z is zero while that of modulus y and z is not zero, the real and imaginary parts of the quotient is the same as that in part 1).
3) If the remainder of modulus x and z is not zero while that of modulus of y and z is zero, then the real part of the quotient is x and the imaginary part is y / z^2.

A sample input and output for the division operation of complex numbers is shown below when the program is run.

```
                Complex Numbers Division

Enter a and b where a + ib is the first complex number:
a = 5
b = -3

Enter c and d where c + id is the second complex number:
c = 6
d = 1.2

Division of the complex numbers = 0.705 -0.641i
```

Entry in Complex log file:

Each time an operation using complex numbers is executed, the details are entered and stored in a log file (Complex_Log.txt).

- The operation which was chosen to execute the calculations is stored as "Operation : Complex numbers <Operation name>".
- Both complex numbers which are entered as input are stored.
- The output is also entered and stored, that is, the resultant complex number after performing the operations.
- We shall also store the date and time at which each particular calculation is carried out using the <time.h> library in c.

```
time_t t;   // not a primitive datatype
time(&t);
```

All calculations entries for this module can be reviewed in the log file.

Below is an example of the log file for a division and addition operation:

```
------------------------------------------------
Executed on: Thu Apr 21 12:29:59 2022
Operation : Complex numbers division
Input:
    complex number 1 = 5.00 -3.00i
    complex number 2 = 6.00 + 1.20i
Output:
    Quotient = 0.705 -0.641i

------------------------------------------------
Executed on: Thu Apr 21 12:55:07 2022
Operation : Complex numbers addition
Input:
    complex number 1 = 6.00 + 0.00i
    complex number 2 = -3.00 + 4.50i
Output:
    Sum = 3.00 + 4.50i
```

Complex_Menu:

A menu was created for this module which provides a selection including all the 4 sections to perform calculations for MathCalc_Complex. It was implemented using switch/case function in c after having asked user to input his choice. An example for the first case is shown:

```
switch (choice)
{
case 1:
    printf("\n\n\t\tComplex Numbers Addition\n\n");
    complexAdd();
    break;
```

After operations are completed, the menu asks user whether or not they want to calculate again. As long as they choose 'Y' which stands for yes, the menu displays the selection list.

After the module has been completed and tested thoroughly, the Complex_branch was merged to the master branch.

3.4.2 Matrix Determinant, Inverse and Transpose

Determinant

Here we are using Gauss Elimination Technique for transforming matrix to upper triangular matrix, with the use of nested loops.

```
//Applying Gauss Elimination
for(i=0;i< n;i++)
 {
    for(j=i+1;j< n;j++)
    {
        ratio = a[j][i]/a[i][i];

        for(k=0;k< n;k++)
        {
            a[j][k] = a[j][k] - ratio*a[i][k];
        }
    }
}
```

Then, with the use of a 'for' loop, we find the determinant by multiplying elements in principal diagonal elements.

```
for(i=0;i< n;i++)
{
   det = det * a[i][i];
}
```

To compute the transpose of a matrix, we interchange the rows into columns and vice versa with the use of a nested loop.

```
for (i = 0;i < r; i++)
{
  for (j = 0;j < r; j++)
 {
    b[i][j] = fac[j][i];
  }
}
```

Inverse

Matrix of Cofactors function

After finding the matrix of cofactors, the transpose function is called to find its adjugate matrix**.**

```
for (q = 0;q < f; q++)
{
  for (p = 0;p < f; p++)
   {
    m = 0;
    n = 0;
    for (i = 0;i < f; i++)
    {
      for (j = 0;j < f; j++)
       {
        if (i != q && j != p)
        {
          b[m][n] = num[i][j];
          if (n < (f - 2))
           n++;
          else
           {
            n = 0;
            m++;
            }
          }
       }
     }
    fac[q][p] = pow(-1, q + p) * ideterminant(b, f - 1);
  }
}
itranspose(num, fac, f);
```

Transpose function:

To compute the transpose of a matrix, we interchange the rows into columns and vice versa with the use of a nested loop.

```
for (i = 0;i < r; i++)
{
  for (j = 0;j < r; j++)
 {
    b[i][j] = fac[j][i];
  }
}
```

Inverse function:

Nested loop are applied to find the inverse of the matrix, the adjugate matrix gets divided by the matrix determinant.

```
for (i = 0;i < r; i++){
  for (j = 0;j < r; j++){
    inverse[i][j] = b[i][j] / d;
  }
}
```

3.4.3 Area Under Curve in MathCalc_Ar_Vol

A section to calculate area under polynomial curve (degree 2 to 5) was added to the module. This was implemented using branch so as not to mess the main program in the master branch.

A branch 'Ar_Vol_branch' was created and it can be found in the list of all branches:

```
PS C:\Users\dell\OneDrive\Desktop\University\y2s1\SE_Project_Group1\MathCalc> git branch
  Ar_Vol_branch
  Complex_branch
  MathCalc_Files_branch
* master
```

The section to implement the area under curve is as follows:

- User is asked to input the polynomial equation of the curve as well as limits between which the area is to be calculated, and the accuracy required.
- The area under curve is then calculated using two methods:

  i) Simpson's 3rd rule

  - see link " http://mathforcollege.com/nm/mws/gen/07int/mws_gen_int_txt_simpson13.pdf "

  ii) Trapezoid rule

  - see link " https://math24.net/trapezoidal-rule.html "
- The output is shown as the area under curve calculated by both methods.

A sample input and output is shown below:

```
                Area Under Curve


Please input the degree (highest power) of the expression, should be between 2-5 inclusive:4

Please input the quadratic expression (starting from constant and ascending in power):
Constant = 4
X = -2
X^2 = 5
X^3 = -7
X^4 = 3

Enter the initial limit: -2

Enter the final limit: 3

Enter the desired accuracy: 0.01

The integral using Simpson's Rule is: 124.595 for 18 sub-intervals.
The integral using Trapezoidal Rule is: 124.844 with 54 intervals
```

Here also, the operation, inputs and output are entered and stored in the Ar_Vol_Log.txt (using function polySaveDouble) as shown:

```
-----------------------------------------------
Executed on: Thu Apr 21 00:45:01 2022
Operation Done: Area Under Curve
Input:
    Expression : 3X^4 + -7X^3 + 5X^2 + -2X + 4
    Limits: between -2 and 3
    Accuracy: 0.01
Output:
    The integral using Simpson's Rule is: 124.595 for 18 sub-intervals.
    The integral using Trapezoidal Rule is: 124.844 with 54 intervals
```

After the program has been written and tested thoroughly, the branch was merged to the master branch.

3.3.4 Scientific Calculator:

The scientific calculator was also implemented as in 2.3.4 using mathematical operators and math.h functions.

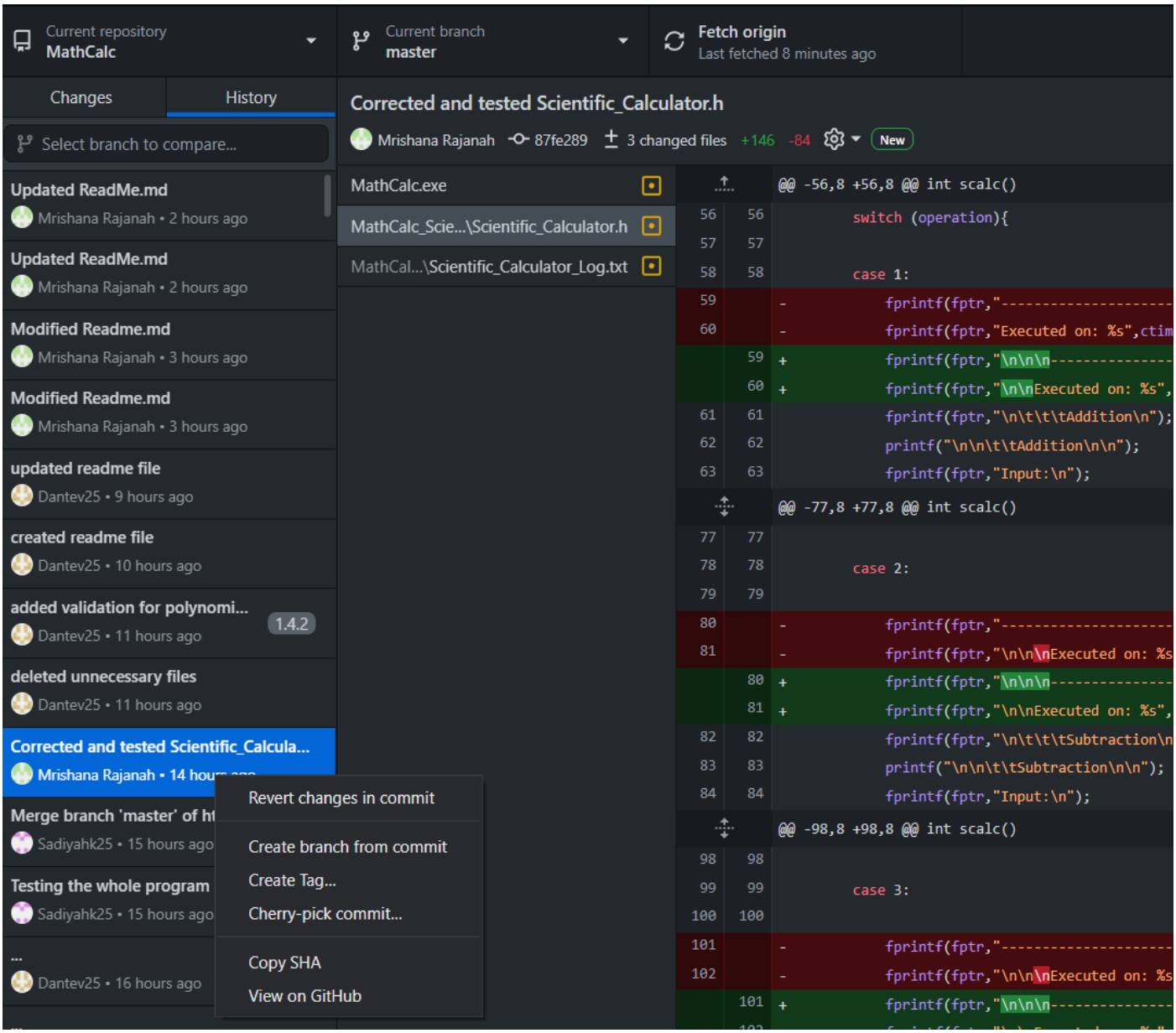
3.5 Versioning:

Using GitHub, it was very easy to name and define specific versions of the programs using tags. Tags create a snapshot of the program, and saves it under a version name. Our versioning naming convention is:

Version: X.Y.Z

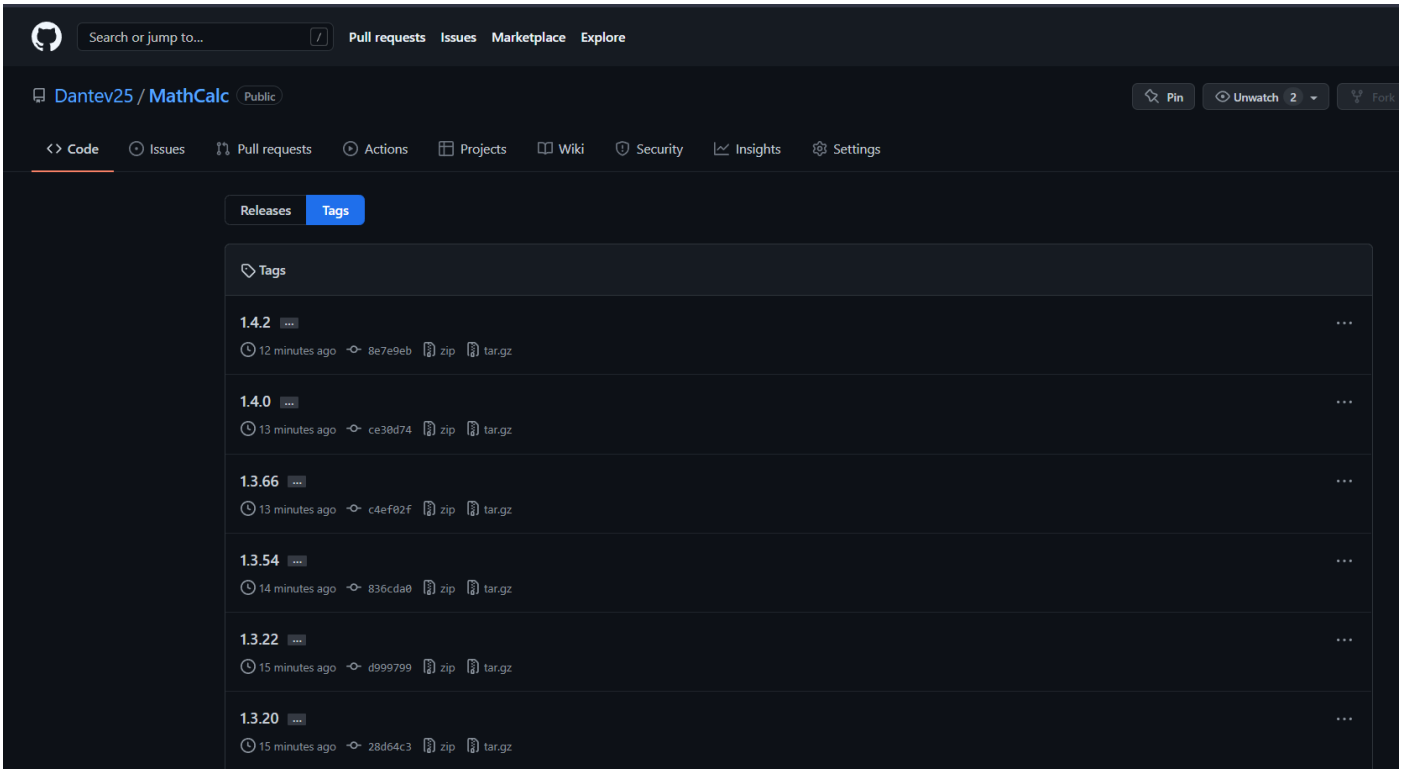
- X - 0 when the program is in beta/in testing and still unstable for release, 1 upon release. Incremented after major updates.
- Y - Incremented upon addition of a major feature.
- Z - Patches, fixes and minor optimisations.

To process versioning, we used a program called 'GitHub Desktop'. It is a very powerful app that allows us to see all the commits in a repository we collaborate in, see changes, branches, and set tags at any point in the repository history.



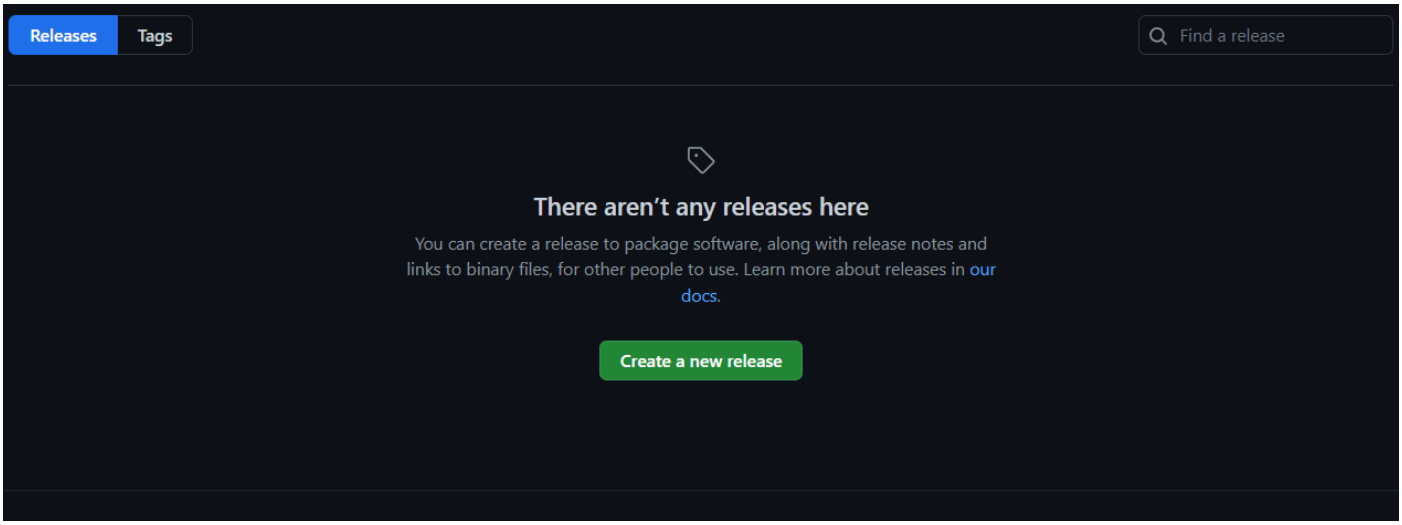
As seen above, it gives a very clear view of what is happening on the repository, and with just a right click on the commits, we can create tags to set the version of the program.

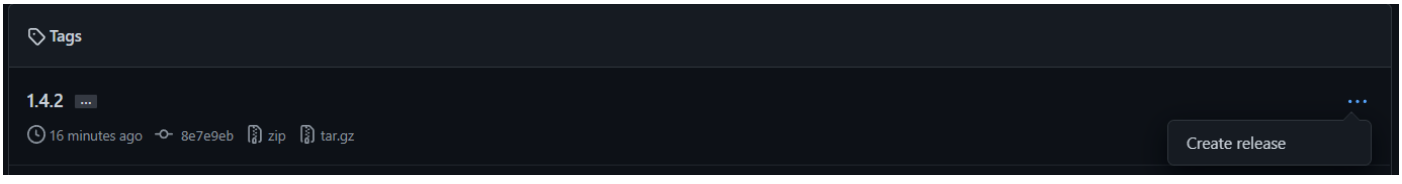
Below is how it appears in GitHub after setting the versions and pushing the tags onto the repository:



There, we can then decide which versions we want to release for public testing, or which version we want to release as definitive. Updates can also be released very easily from here.

Using the ‘Create Release’ button as shown below or by selecting the tags, we can easily publicly release the program as a package, with all necessary documentations and binary files:





4. Testing and Debugging:

Testing methodology:

1. All sections were first independently tested in a test main. Every variable used was tested using values in the normal range, extreme cases and abnormal values.
2. We then tested from the menu of the sections to see if there were any undefined or undesired behaviors.
3. In case of undefined behavior, the loops/variables are tested using varying cases until it satisfies the logic of the program.
4. Finally, after implementing all the menus and submenus, we tested from main, to each branch of the main menu, then to each branch of the different sections.

Below is an example testing the matrix input:

To verify if the number of rows is valid, a while function has been added whereby, if the user input an incorrect value, they will be asked to input again. Here, since we have allocated the matrix a 2D array of size 100 * 100, the user will be asked to type in a value not exceeding 99. Also, since we can have neither zero nor negative number of rows, the user will be requested to insert a value above 0.

```
                Matrix Transpose

Enter the Number of Rows : -1

You cannot have 0 or negative number of rows.
Please input a positive integer not exceeding 99 : 0

You cannot have 0 or negative number of rows.
Please input a positive integer not exceeding 99 : 100

You cannot have 0 or negative number of rows.
Please input a positive integer not exceeding 99 : 2

Enter the Number of Columns : 2
```

Unless the value fits the specified range, one will be asked to enter the value persistently as shown in the diagram. The same approach has been followed for every parameter that requires a verification and validation process for the code to work.

Strengths

- User-friendly - The user interface was designed such that it was intuitive for any user, including first time ones.
- Efficient - The calculator returns answer very fast, even for large computations
- Fully featured - 50 unique calculation possibilities
- Covers a wide range of mathematical computations - from shape measurements to complex numbers to polynomial, MathCalc covers a wide array of mathematical topics.

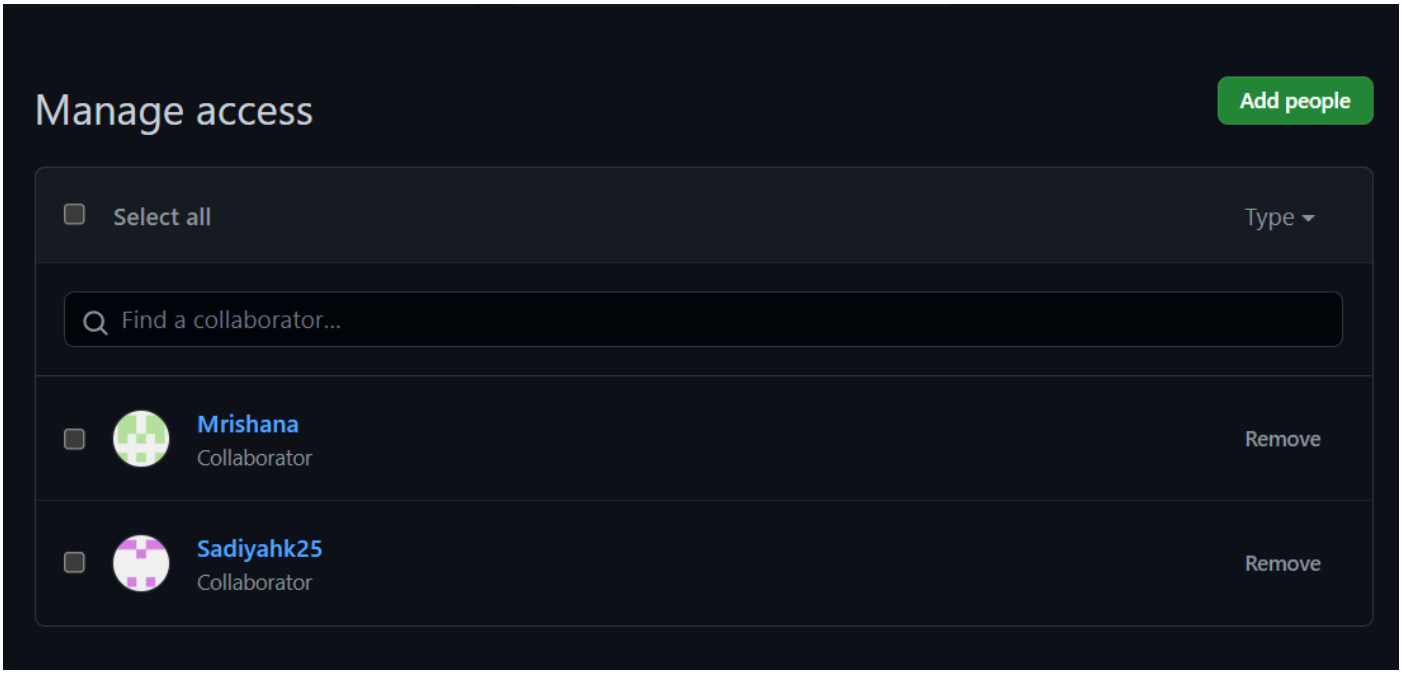
Limitations

- Owing to undefined behavior, polynomial division only goes up to degree 2.
- Area under curve goes up to degree 5, as input cannot be implemented in a loop.

5. Git hub and how it facilitates development

Through the use of github and git, we have been allowed to work remotely, without unnecessary uploads and downloads of files.

It took very little effort to setup, and to add collaborators were very easy, with the use of the invite button on the GitHub website.



From then on, we could easily manage the different sections and modules efficiently.

For example, to help someone debug a code, or to see their method, we simply had to see their committed files and do the necessary. It only took a single button press to synchronise our files and match versions.

Another advantage of GitHub is that many collaborators can work on a single large file system at once, and we did not need to manage much in terms of file organization or uploads and downloads as git already did that for us.

Through the use of git commit messages, we found that it was very easy to track our different changes and better got an estimate on how development is progressing. It also made our work more safe, as any commits could easily be reset to a previous one, thus making rollbacks to previous more stable versions of the program in development simple.

GitHub brings a sense of safety and improves efficiency to the point that workflow between members of a team becomes flawless and seamless due to the different powerful tools git offers.

An example could be git push and pulls, which allows us to update the files on a repository with a single click, instead of manually going and downloading 10's of files, and naming them so as to keep track etc.

Remote working is facilitated as it was easier to help someone having difficulties, to share code, or to debug as a team. This is due to the fact that GitHub allows multiple users to work on one project no matter where, we did not have to physically meet to work together seamlessly.

Finally, and most importantly, GitHub offered us a powerful way to track who did what where. This is extremely important when it comes to working on a large project, as it is easy to communicate to the person who made the changes. In addition to this, tracking the changes over versions of the program allowed us to choose the best solutions and to clearly see how our changes impacted the overall project.