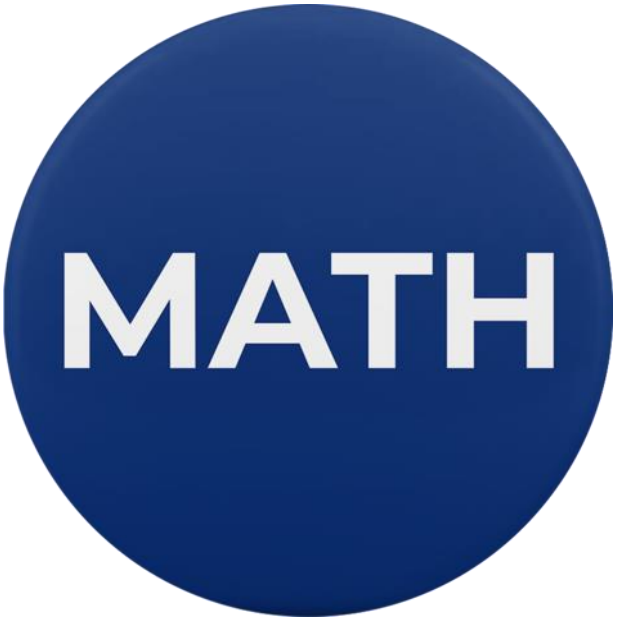
Mathematics Testing Program



Candidate's full name: Joshua Bourton

Candidate number: 6011

Centre number: 55233

NEA A LEVEL PROJECT

CONTENTS

ANALYSIS

1.1 Problem description

In order to succeed in A Level Mathematics, students must frequently review a broad range of question types that aim to challenge their ability and reinforce what they've previously learnt in class. Many students struggle to find online resources that can accommodate their individual academic needs and the programs and websites currently in place that aim to meet these requirements are either limited in functionality (and therefore usefulness to the student), or crammed full of an overwhelming number of features, making it difficult and frustrating for the student to navigate through.

Furthermore, many of the more useful systems currently in place do not tailor their content to the individual student: the most common form of testing a student's mathematical ability in these programs is a generic, pre-set series of math questions that are presented to the student in a test-like format. These programs will often only mark questions correct or incorrect without worked solutions being presented to the student, and rarely consider the areas that the student needs to improve upon. This lack of tailored content results in ineffective use of the student's revision time, often impacting that student's ability to achieve their target grade.

This is why the end user – an A Level mathematics teacher - has requested that a program be written to provide a more suitable platform on which students can learn maths; the *Mathematics Teaching Program* aims to address the issues of the current systems in place by providing an easy-to-use UI on which students can access content tailored to their specific level of ability.

1.2. Background to problem and research complete

Research outline

My research will be based on the strengths and weaknesses of an online website called mathsmadeeasy, a tool which the end user often uses to direct his students to practice maths questions in preparation for upcoming assessments.

There are a handful of features of the current system that are very well suited to the end user's needs, and I feel that it would be appropriate to retain these features for the end user and his students by incorporating them into the *Mathematics Teaching Program*. However, as shown in my research below, I have identified a number of features from the current system that are either unnecessary for the end user, lacking in usefulness or missing entirely. By remodelling these features to better suit the end user's needs I aim to be able to increase the functionality of the program whilst simultaneously decreasing its complexity.

This will make the new program more helpful to the end user and his students, as it will be made easier for them to navigate whilst being tailored to the needs of his classes.

The school-wide system currently in place, Moodle, is another system that the end user uses to make resources available to his students. However, I have made the decision not to incorporate this into my research because if I were to attempt to recreate a system similar to Moodle I would encounter a technical limitation; files are uploaded to a central server, and as I do not have access to a server I therefore cannot create a system with similar functionality.

Analysis of current system

The system currently used by the end user to set students practice questions is a website called *mathsmadeeasy*. The aim of the website is to provide students with a broad range of maths questions tailored to the student's level of study and presented in a clear, quiz-like fashion.

One of the end user's main problems with *mathsmadeeasy* is its user interface. The UI on mathsmadeeasy is difficult to navigate, visually busy and lacks a consistent colour scheme. All of this makes it difficult for students to find the questions that have been assigned to them by the end user, and he often finds that students must ask him for help when finding the specific webpage. Therefore, incorporating a simpler, more visually appealing UI whilst designing the *Maths Testing Program* will help save time and frustration for both the end user and his students.

Figure 1 shows the homepage of the website. It consists of a range navigational buttons with links to areas such as as the mathsmadeeasy online shop, an exam booking page, a login system, an online tutor-booking page and an area where user's can leave a review of the website.

The end user only uses the "Revise" and "Past Papers" navigational buttons, so when creating the Mathematics Testing Program I plan to simplify the UI by making these buttons large and obvious to the user, whilst removing the buttons that are unused by the end user and his students.

Figure 1: The mathsmadeeasy

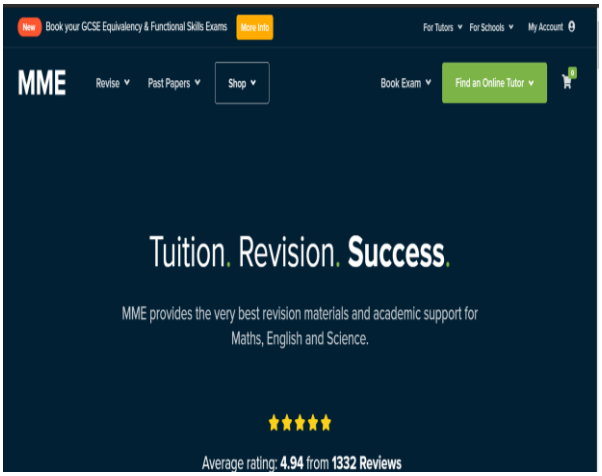


Figure 2: The “Revise” drop-down

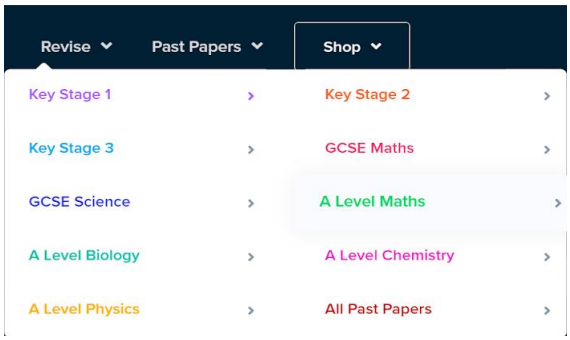


Figure 3: The “Past Papers” drop-down menu

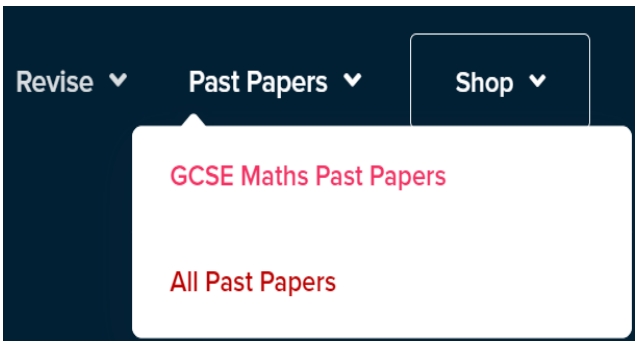


Figure 2 shows the “Revise” drop-down menu available to students that allow them to select a subject or key-stage out of a list of available options, and then takes the user to a page containing subject content relevant to their particular selection.
As the end user only teaches A Level maths, these options are not relevant to him and his students, and therefore need not be included. Furthermore, the end user wants a consistent colour scheme and the wide range of colours used does not fall in line with this requirement.

Figure 3 shows the “Past Papers” drop-down menu, which gives the student the option to select either “GCSE Maths Past Papers” or “All Past Papers”. The end user feels that this is a poor way to organise the website because it makes it difficult for his A Level maths students to find past papers relevant their exam board and level of study.

Figure 4 shows a section of the webpage accessed by clicking on the “Past Papers” navigational button. It is laid out with mathsmadeeasy’s own predicted papers on the left-hand side. These papers are not organised by either exam board or level of study, which is messy and a subsequently confusing for students.

From **Figure 4** and **Figure 5** we can see that on the right-hand side, all the past papers are A level, and organised by exam board.
On the left-hand side following on from mathsmadeeasy’s own predicted papers, past papers are organised exam board for exclusively GCSE past papers.

Although the end user only teaches Edexcel maths, he likes the organised style of this webpage and would like similar organisational menus in his new maths program.

Figure 4: The “Past Papers” webpage

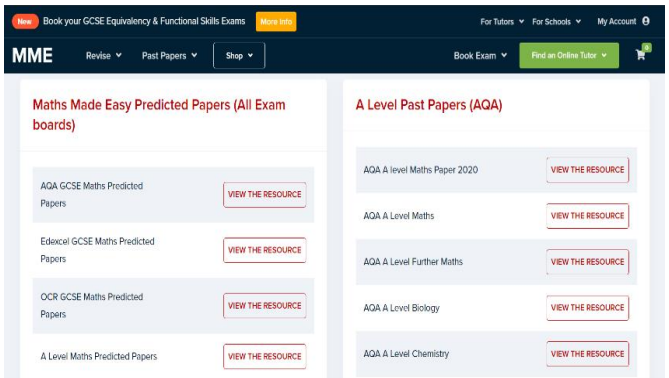


Figure 5: A continuation of the “Past Papers” webpage

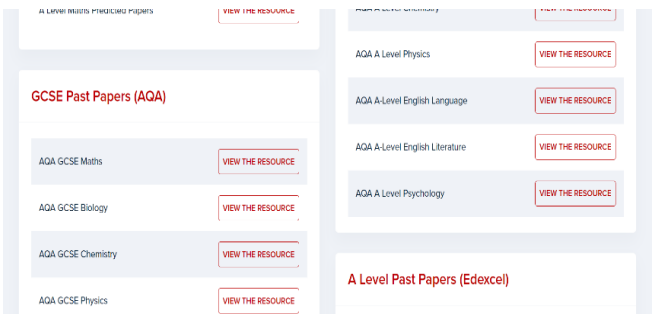


Figure 6 and Figure 7 show the webpage accessed from clicking on the “A Level Maths” section shown in green in **Figure 2**.

Figure 6 shows the first section of the webpage, which contains an advertisement for maths flashcards sold by mathsmadeeasy and a search bar to access specific maths papers. Problems with the search bar are described below. The advertisement is an unnecessary feature of the webpage because the end user has stated that they wish for all features of the *Mathematics Testing Program* to be made available to all students.

Figure 7 shows further down the same webpage. As commented on above with **Figure 2**, the colour scheme here is inconsistent and not visually appealing. There are several options available to the student on this section of the webpage. The end user often directs students to the “MME A Level Maths Online Test” section (highlighted), which will be one of the main areas that the *Mathematics Testing Program* will aim to improve, so this section will be bigger, bolder and more prominent than it currently is.

Figure 6: The “A Level Maths” webpage

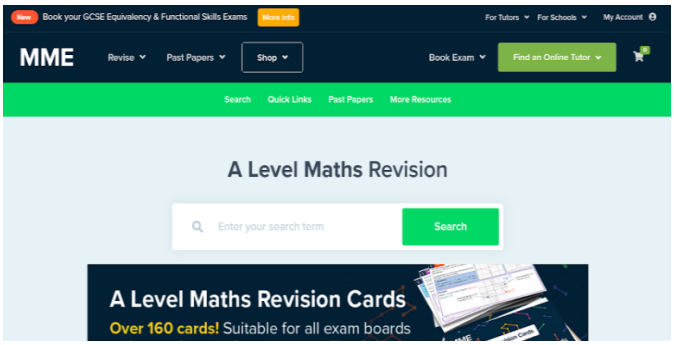


Figure 7: A continuation of the “A Level Maths webpage

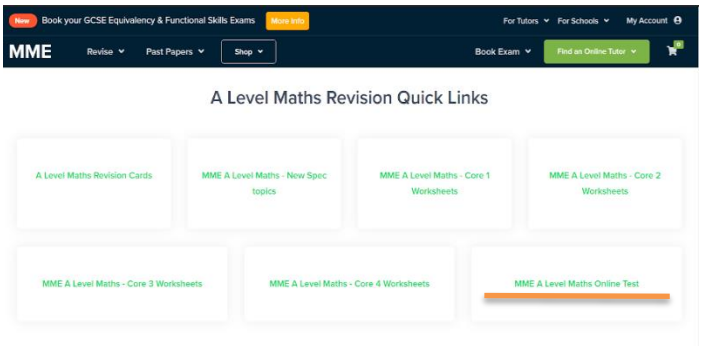


Figure 8

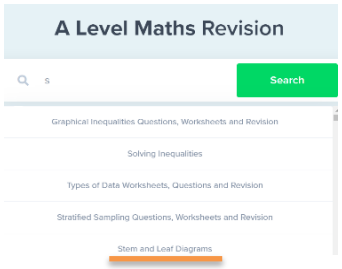
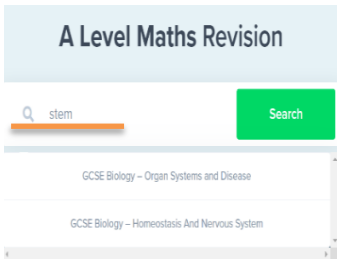


Figure 9



Students have reported to the end user that the search bar does not function as intended. **Figure 8** and **Figure 9** show the search bar in action. The user is attempting to search for the maths topic “Stem and Leaf Diagrams”. As underlined in **Figure 8**, the topic shows up when the user begins to type in the word “Stem”. However, when the user finishes typing the word, the topic suggestion disappears. This suggests that the algorithms used in the search bar are flawed, and if it were to be replicated in the Mathematics Testing Program, would need to be modified to improve the accuracy of the search.

Figure 10: The “MME A Level Maths Online Tests” webpage

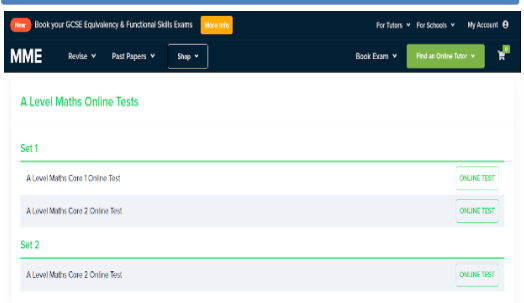


Figure 10 shows the webpage accessed by clicking on the “MME A Level Maths Online Tests” section (shown underlined in **Figure 7**). There are a range of different tests available for the student to take; this is a useful feature of the current system and one which will be retained in the *Mathematics Testing Program*.

Figure 11: The instructions for a practice test

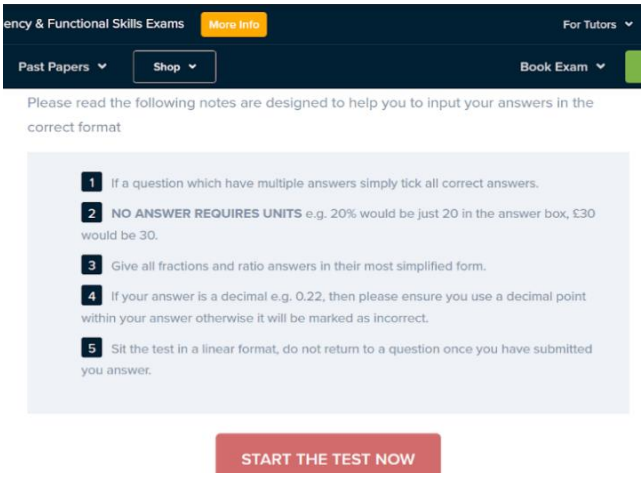


Figure 11 shows the page accessed by clicking on one of the test buttons shown in **Figure 10**. The instructions shown are presented as clear and numbered, making it easy for students to read and understand.

Where instructions for the user are necessary in the *Mathematics Testing Program* they will be presented in a clear, numbered in a similar fashion as the instructions shown **Figure 11**.

Figure 12: The beginning of a practice test

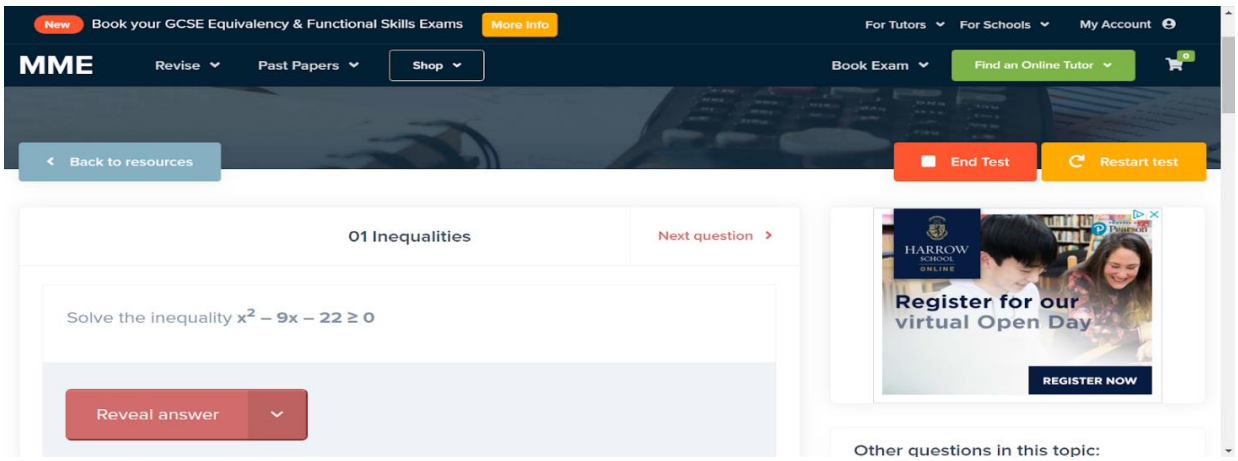


Figure 12 shows the beginning of one of the practice papers, accessed by pressing the “Start the test now” button shown in **Figure 11**. There are several features available to the student throughout the test. There are “End Test” and “Restart Test” buttons in the top right corner of the screen. The restart button is an issue for the end user as students can get the question purposefully wrong to look at the right answer and then restart the test to answer the question correct. There is also a “Next question” button and a “Reveal answer” button, along with a “Back to resources” button that takes the user back to the menu shown in **Figure 11**. Upon pressing the “Next question” button shown in **Figure 12** the student is able to access the next question without needing to complete the previous one, which means that students can skip any questions they find difficult.

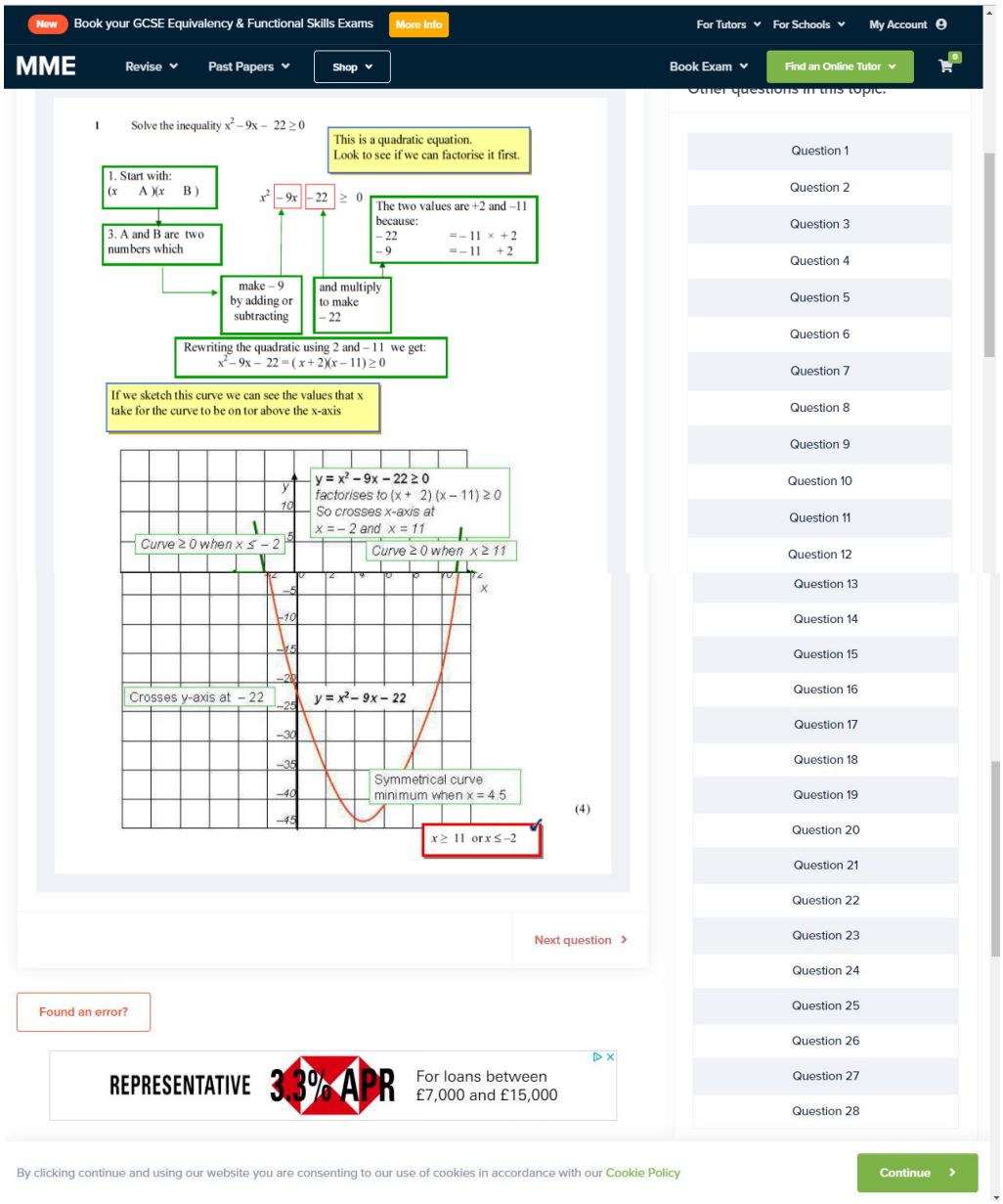


Figure 13: The full webpage shown when the “Reveal Answer” button is clicked.

Figure 13 shows the mark scheme the student has access to when pressing the “Reveal Answer” button. There are several major areas here that need to be improved upon: Firstly, the webpage does not take in any answers, so cannot mark and record the students’ scores. Furthermore, the number of questions in a test (28 in this test, as shown in the right-hand column of **Figure 13**) is too long for most students to complete in one sitting and consequently needs to be shortened in order to account for this. However, the mark scheme which the student has access to is clear and methodical, with a graph and step-by-step instructions on how to answer the question, which is a helpful feature that will be included in the *Mathematics Testing Program*. Furthermore, when the test is restarted the same questions show up in the same order; there is not attempt to randomise which questions students will complete first, which means that the questions at the beginning of the test are more likely to be completed more often than the questions at the end.

Figure 14 shows the webpage accessed after pressing the “End test” button shown in **Figure 13**. There is an option to restart the test or to go back to the menu webpage shown in **Figure 10**. The limited selection of options here is useful for students because it makes the webpage easier to navigate.

Figure 15 shows further down the same page. There is an option to call or search for tutors; this is an advertisement but could be turned into a useful feature by changing contact details to the end user’s email address.

Figure 14: The “End test” webpage

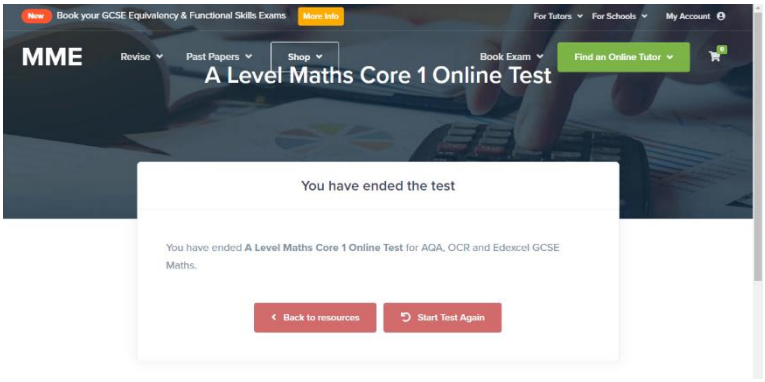


Figure 15: The “End test” webpage further down

Background to the problem

After analysis of the current system, I have identified several underlying problems. These are as followed:

- Most of the available features on the website are unlikely to be used by A level students and their respective teachers. This is because the website is targeted at a range of different ages and abilities, and as such contains many resources which are not useful to A level students. This makes the website feel unnecessarily busy.
- The features that are likely to be used by A level students are difficult to locate, which is inconvenient for the student.
- The current colour scheme used on the website is inconsistent and chaotic, making it unpleasant and confusing for students to navigate through.
- The website contains a plethora of advertisements aimed at persuading students to buy resources such as bespoke practice past papers from mathsmadeeasy’s own shop. This is a feature that hinders all the website’s users, because alongside making the website feel crowded it also puts some of the useful features of the website behind a paywall.
- Students can skip any question they find difficult, which is not ideal as the temptation will arise for students to not attempt questions related to their weakest topics. This will subsequently limit their ability to improve at A Level maths. Instead of allowing students to skip each question, The Mathematics Testing Program could incorporate a “Hint” button to guide students through each question. The use of this button by the student could also be recorded, allowing the program to identify if the student is not

fully sure on how to answer the question. This data could be made available to the student's teacher.

- The "Restart" button allows for students to see the mark scheme before attempting the question, which means that students can easily cheat on the tests they take.
- The program does not take in answers, which means that it cannot mark the question and store the result of the student's attempt at the quiz on their account. As a result, questions cannot be tailored to the individual student's areas of weaknesses, and the student's teacher cannot track the student's progress.
- The quantity and numerical order of questions are fixed; the test length is often too long for students to complete in one sitting, and this means that the questions at the beginning of the test are more likely to be completed more frequently than questions at the end of the test, regardless of how difficult the student finds each question. This could be improved by shortening the length of each test and randomising the order that each question shows up in.

1.3 Identification of the end user

The program will be designed with two different types of users in mind: teachers and students. As the program will gather data about students, it is essential that the person that has access to this data is permitted to see it, and that they will use it in a responsible way that is beneficial to the student. Consequently, I have chosen Mr Pape, an A Level Maths teacher at Poole Grammar School, the end user throughout the duration of this project.

Mr Pape uses several different maths programs to direct students to resources and set homework, with his most frequently used program being a website called mathsmadeeasy, which has been analysed in *1.2: Background to the problem and research completed*. He has found that there are many areas of the website that are not ideal for his use of the program, and as such there many areas of the website that he would like improved and adapted for his specific uses as an A level Mathematics teacher.

Students will also indirectly be the end users for this program, but all requirements for the system will go through Mr Pape who will provide design suggestions for the program with the benefit to his students in mind. As both teachers and students are the end users for this program, a separate area of the program will be needed to account for this; this will be implemented in the form of separate logins for each respective class of end user.

1.4 Interview

START OF INTERVIEW……

Joshua Bourton: What websites, apart from Moodle, do you use to direct students towards practice maths questions and tests?

Mr Pape: So we use Maths Genie, Physics and Maths tutor, Doctor Frost maths, and mathsmadeeasy which is very good, there's a kind of self-assessment on there as well which is very useful.

Joshua Bourton: How do you feel about the organisation and user interface of these websites?

Mr Pape: So Doctor Frost maths is very well structured, that's because the teacher is a computer scientist as well. He has a login on his website and students can login with a username, if they've forgotten their password they can recover it, it keeps a record of schools that have signed up to his website, etcetera. It has also got lots of resources which students can access as well. The other ones are just really depositories of lots of exam questions and papers, topic tests and things like that, so they obviously have a different design and user interface to account for that.

Joshua Bourton: Which areas of these websites do you find particularly useful?

Mr Pape: I think having resources for lots of exam questions and topic-based questions is very useful.

Joshua Bourton: What problems are there with these websites that you would like addressed in the new program?

Mr Pape: I think it would be good to have a setup where you can access certain questions from a topic more easily and maybe check answers quite easily between answering each question, instead of just getting a mark scheme for all the questions after completing the practice test, that would be good. And then practicing similar skills in the same test, but having the questions getting progressively more difficult. This is what a number of the websites don't do, they have topic questions that are randomly selected and therefore you have a very difficult question as the first question and then a much easier question following on, so if they can be in order of difficulty that would be much better.

Joshua Bourton: Do you have suggestions to reduce or eliminate some of those problems?

Mr Pape: So I think ranking the questions in order of difficulty would be good, and being careful to select the type of skills that are required on each topic for each question is useful. You could have, for example, a box for every question that lists what skills would be required for that question. It would also be useful to combine topics so when students are revising when it comes to exam season, students can take a practice test with a combination of questions from different topics.

Joshua Bourton: Which level of study would you like the program to be aimed at, for example, GCSE, AS level or A level?

Mr Pape: A level maths, because that is what I spend the majority of my time teaching, and where some of the details current systems could be improved upon.

Joshua Bourton: How will students input answers into the program? So for example, there could be text boxes to input answers or tick boxes to pick the correct answer to the question out of a list of available questions.

Mr Pape: Well obviously in maths you can have numerical answers so an input box would definitely be an appropriate method for inputting answers. The only issue with that is if the student gives the answer to the wrong accuracy, as in a different number of significant figures to what is required from the question, and then that gets marked incorrect then that could be a problem. I don't want the students to be marked fully incorrect when they get the correct answer and they know what they're doing. It is difficult to test the method, and actually getting students to show their working is more difficult on a computer system, so if

you were to include other types of answers then it would have to be something like “fill in the missing line of working” or “what should go in this part of the calculation”. Some multiple-choice answers would be fine as well. On Edexcel [the exam board we use at Poole Grammar] some of the practice questions we have are of the style “What’s wrong with this method”. This style of question could be incorporated by asking the student “which line is the incorrect calculation in” or “which line is incorrect in this working” or “is this entirely correct” or “is there a mistake in it” etcetera, so there might be possibilities for checking methods by showing a picture of the method and then asking the student to identify the mistake.

Joshua Bourton: What format or formats should the outputs be presented as? For example, a simple tick or cross, a standard mark scheme or bullet-pointed answers showing step-by-step solutions on how to solve the problem.

Mr Pape: I think it would be helpful if the student has access to the full method straight after they’ve submitted their answer to each question, so a mark scheme would be really good. It would also be good to have some kind of visual feedback on whether the answer is correct or not and a scoring system to show the student’s total marks on each practice test. So I’ll leave it up to you to identify a clear way of presenting the student’s results for each practice test they complete.

Joshua Bourton: Should the program have a login feature, and if so should teachers and students have separate logins?

Mr Pape: Yes, it would be good to have logins. If teachers could have a different type of login account to the students, that would be useful because we could see all the feedback from the students who also have accounts on the program.

Joshua Bourton: What features should students and teachers be able to access on their separate logins?

Mr Pape: Students should be able to select the topic they want to revise, and they should be able to see their results on each topic and to track their performance over time. For teachers, it would be helpful to be able to see the students results, and to see the type of questions they’ve been attempting.

Joshua Bourton: Would it be useful for you as a teacher to receive a graph that depicts each individual student's performance over multiple attempts of the same topics and questions?

Mr Pape: Yes, I think it would be helpful to be able to track a student’s progress over time, as that will allow me to identify topics that students are consistently stuck on, and then maybe go over the topic again in class. It would also allow me to identify which students are falling behind so I can give them the extra help they need to get back on track to meeting their target grade. Also, it would be useful to have a graph that shows the performance of the whole class on each topic, and maybe even to be able to compare different maths classes in the same year group.

Joshua Bourton: Do you think it would be helpful if the program tailors the topic and level of difficulty to the individual student’s ability?

Mr Pape: Yes, I think it would be brilliant if the program is designed to not just give generic practice questions to all students but to instead tailor the type of questions given to students based on their strengths and weaknesses. It would be really good if the program almost worked with students, as in it starts off by giving them easier style exam questions, and then,

as the student gets better at answering those questions, slowly increases the difficulty of the questions given, allowing the student to work up to their target grade in steps.

Joshua Bourton: Should the questions be pre-set during programming or should there be an option for the teacher to add and remove questions?

Mr Pape: That's a difficult one. My personal preference is that all the questions would be already there for the teachers to direct students towards, but if the teacher is able to put in questions or take them out then that would be a good way to make the program better. I can understand that if it takes too long that would be something that could be left out of the programming as isn't an essential feature that will significantly aid the functionality of the program, so only include it in the program if you have enough time near the end of the design.

Joshua Bourton: Should there be a search bar incorporated for students to access specific topics and search for specific content areas?

Mr Pape: Well, there could be a search bar but I think a more simpler and effective alternative would be to have a structure in place that allows the student to narrow down the options until they get to a specific question type for each sub-topic. So what by that is, there would be three buttons to choose from: stats, mechanics and pure maths. Clicking on one of those buttons would take you to a list of all the available topics for that particular area of the course, and then clicking on one of those topics would present the student with a list of practice questions for each sub-topic.

Joshua Bourton: Will personal data such as names/email addresses be entered into the system, and if so does the system need to be made secure?

Mr Pape: Yes and yes.

Joshua Bourton: Should all the content on the new system be free for all students to access?

Mr Pape: Yes.

Joshua Bourton: How many questions would you like there to be on each quiz?

Mr Pape: I would suggest that three questions would be the minimum for each quiz, but three to ten questions on each quiz would be a good amount, because three would probably cover the main types of questions in a particular sub-topic, and if you had about ten questions then you could cover quite a range of different question style, but there would be a bit of overlap between the questions.

END OF INTERVIEW……..

1.5 Initial user requirements

The proposed new system, based around testing student's A level mathematical skills, will be designed to meet the following criteria:

1) A login system consisting of a username and password, that will allow users to access the program from the machine the program is installed on.

1a) The ability to create an account from within the program.

1b) The login system will consist of two different styles of user accounts: an account for teacher's and an account for student's.

1c) Personal data, including the username, password and email address of each person using the program, will be taken in by the program upon signup and stored securely in a database using encryption.

2) A clean, simple user interface that will allow for easy navigation of the program will be built for the *Student* user type. Features that the end user and his students do not use will be excluded from the program, and features that they do use will be made more prominent to the users.

2a) The UI for the student account will consist a menu screen with buttons that will enable the student to navigate to their desired sub-topic to by selecting a module.

2b) The student be able to select a topic from a list of available topics.

2c) Upon clicking on a topic, the student will be presented with a list of sub-topics that each contain a quiz consisting of questions exclusively related to that sub-topic.

2d) The UI for the teacher user account will consist of a menu screen containing a search bar.

2e) This search bar will be used to search for specific students.

2f) Upon selecting a student, data gathered on that student by the program will be presented to the teacher, and the student's progress in particular topics over time will be displayed in graph form.

2g) The menu will also consist of an option to track the class's progress as a whole, and will compare the students in each class with each other.

2h) A consistent colour scheme and background image will be used across both categories of user accounts to make the program easier to navigate.

3) A collection of practice questions that the program has access too. These programs will be used in during the quizzes.

These questions will initially be pre-set during programming. However, the end user has specified that if there is enough time to incorporate the feature, having the ability to add questions to the collection of questions stored by the program whilst the program is running would be an added bonus.

4) The ability for the student to be able to select a specific mathematical topic and then be presented with a quiz containing practice questions relevant to that topic.

Questions related to each sub-topic will be displayed when the student selects a particular sub-topic from a topic page. These questions will be presented to the student in order of difficulty.

5) A mixed quiz consisting of a variety of different types of practice questions from a range of different topics.

5a) The questions will be selected based off the individual student's areas of strengths and weaknesses, which will be gathered from the student's previous attempts at questions.
5b) If insufficient data is held about the student, then practice questions will be randomly generated and presented to the student in order of difficulty.

6) A mark scheme for each individual question, which will be presented to the student immediately after they have submitted their answer to the question. A mark scheme containing worked solutions for all the questions that were in the test will also be made available to the student at the end of the test.

   The mark scheme will be presented in the same way as the exam board the end user teaches (Edexcel) present their mark schemes.

7) Questions will be input into the program in the following ways:
   - Input boxes to input numerical answers for worked questions.
   - Tick boxes to answer multiple choice questions.

When inputting numerical values into the program, the following syntax will be used:
- *Square root: **Sqrt**(value)*
- *Raising to an exponent: x**^**(value)*
- *Logarithms: **log/base/value**, **ln/base/value***

8) The skills required to answer each question will be presented to the student on every question.

   - *This will be in the form a list, containing bullet points of the key skills required to answer the question.*
   - *These skills will also be used by the algorithm that will determine the difficulty level of each question.*

9) The font and button size will be large and obvious so that the program is more accessible to users with certain learning difficulties or poor eyesight. Where appropriate, text will be in bold so as to draw attention to certain parts of the program.

1.6 Prototype and Modelling

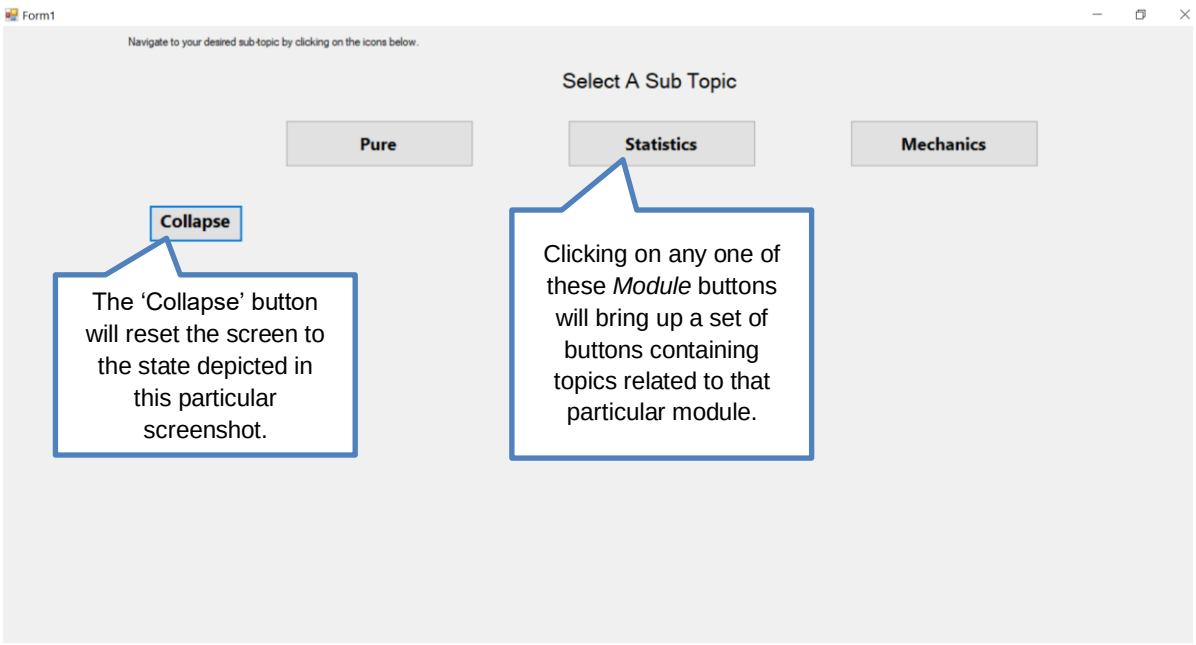
1.6.1 Overview of the prototype

Note: The following prototype is an abstraction of how the final program will turn out in the following ways:

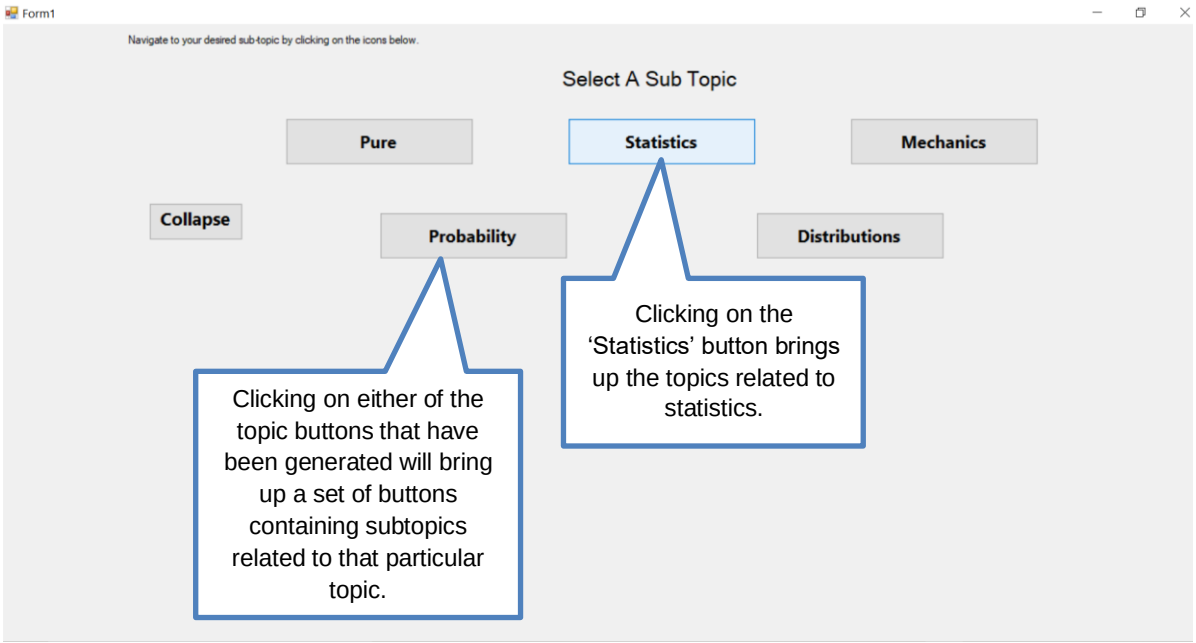
- *The '**Login'** and '**Create An Account'** section have been purposefully excluded from the model for simplification purposes; These modules are fairly generic to a wide variety of programs and websites, so there was deemed to be very little value in creating a prototype for these modules.*
- *The '**Teacher**' section of the program (data modelling) is not included in the prototype, as much of how the data will be displayed in this part of the program is described in 2.1 Modular structure of the system, so the planning for this particular section has been completed to a sufficient level.*
- *Background images, variation in font size & format and the inclusion of logos will not be incorporated within this prototype to help save time and simplify the design.*

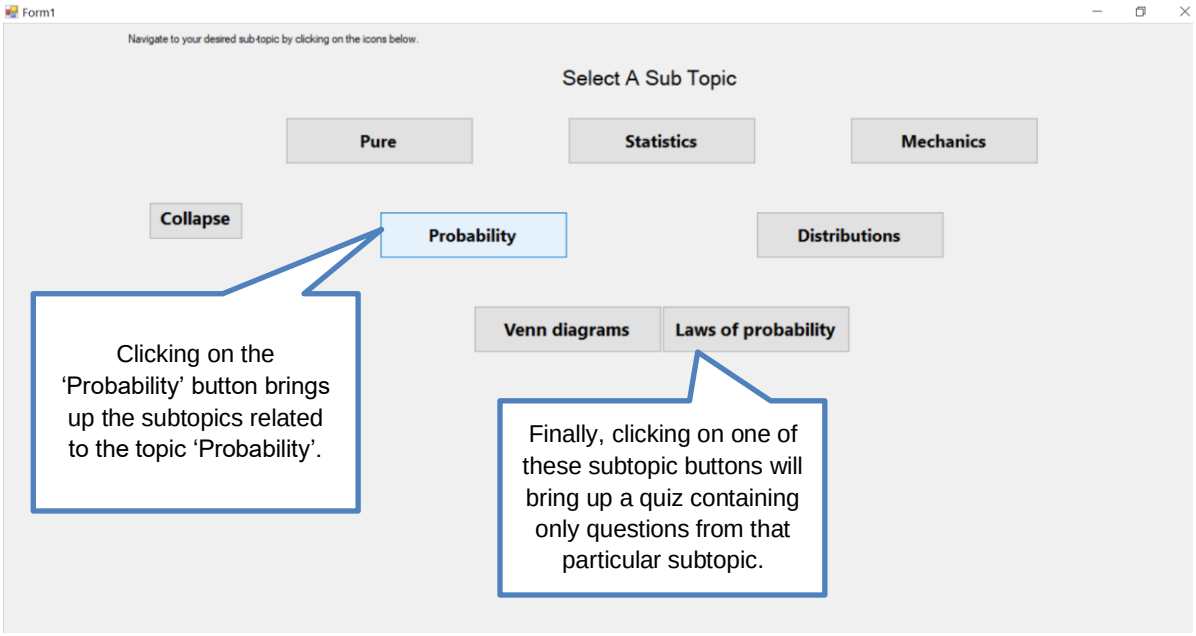
Figure 1: Selecting A subtopic

The following set of screenshots depict how a student would navigate to a quiz containing questions related to a specific subtopic of their choice.



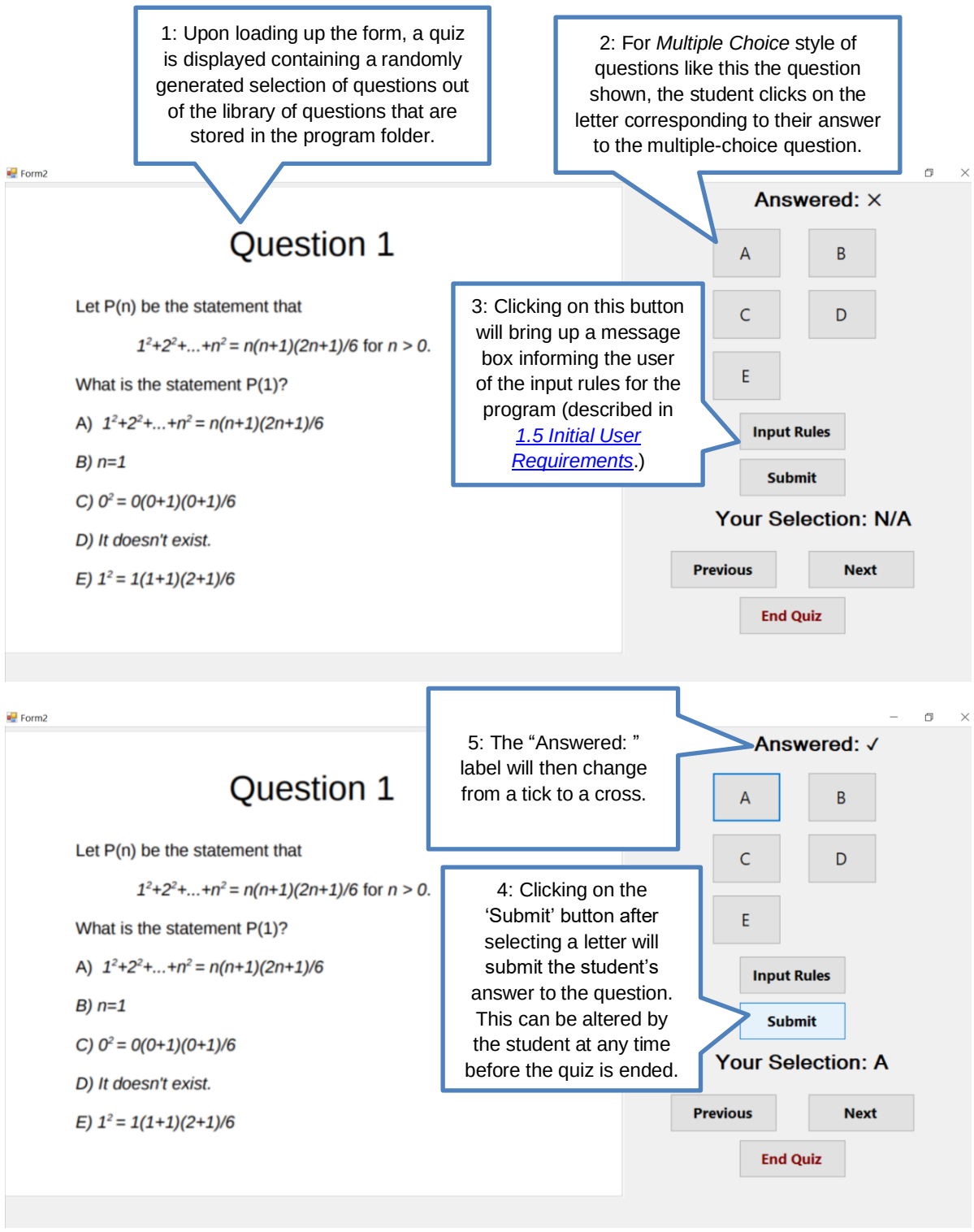
Comp 4





*Note: At this stage in the final product, an adaptation of the **Randomised Quiz** module would have been loaded up with questions related to the subtopic selected by the student. However, for simplicity's sake, this part of the program has not been included in the prototype.*

Figure 2: Taking a quiz
The following set of screenshots depicts the UI that the student is presented with when taking a quiz within the program.



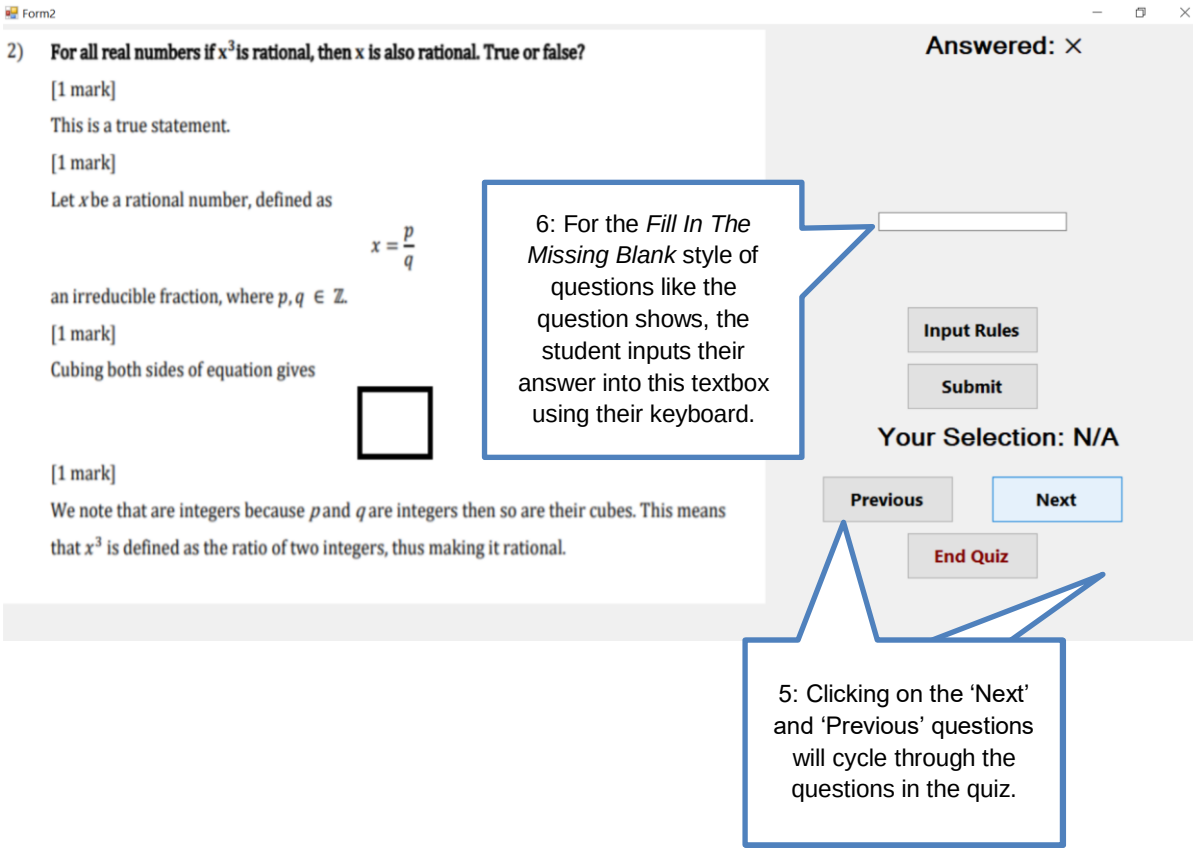
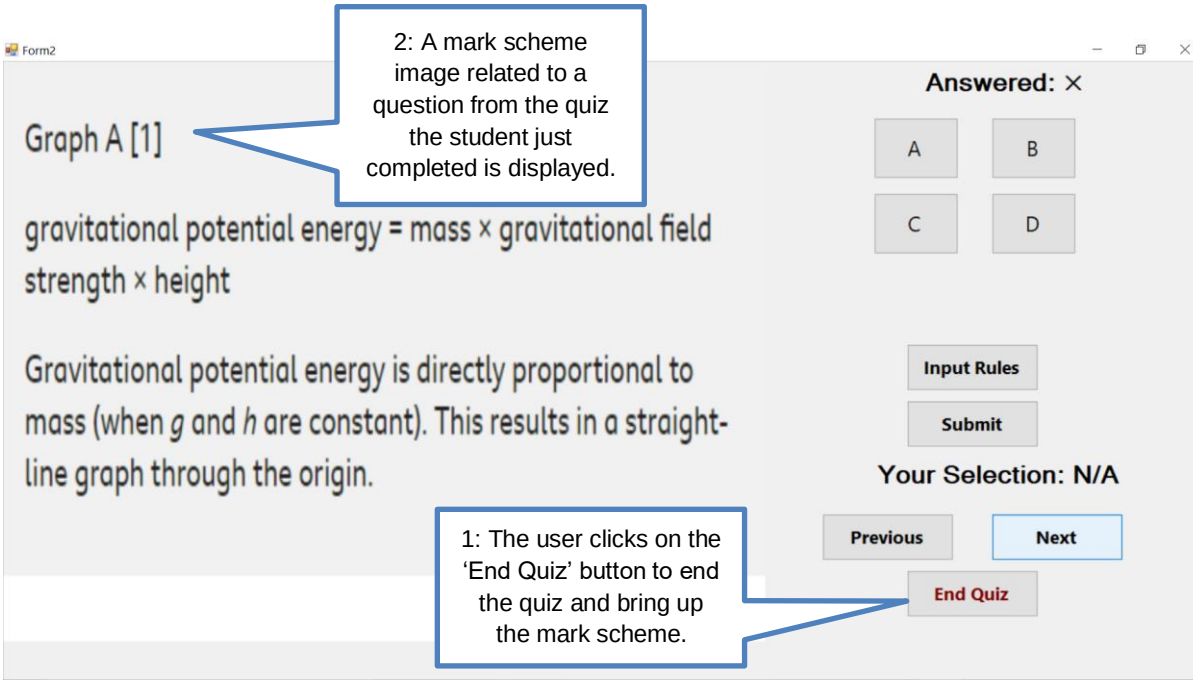
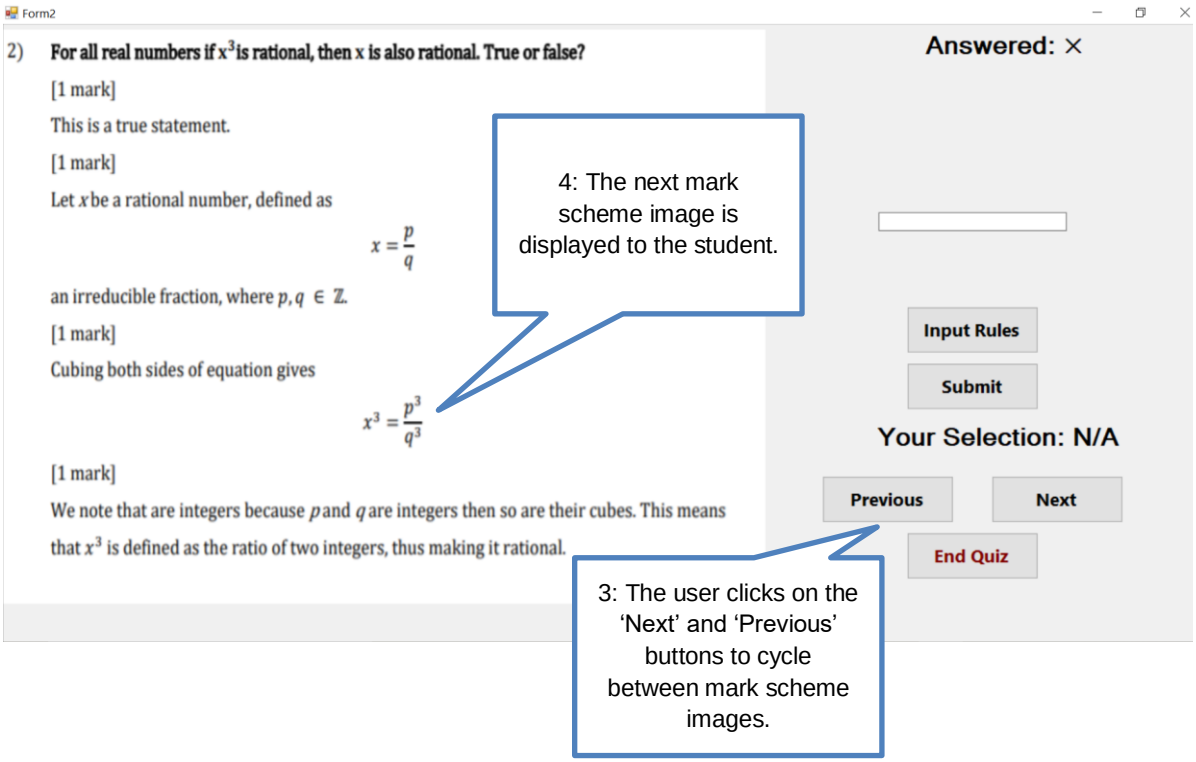


Figure 3: Displaying a mark scheme
The following set of screenshots demonstrate how the student can view a mark scheme for the questions in the quiz.





1.6.2 User feedback from the prototype

To gather feedback on the suitability of the prototype to solve the problem described in 1.1 Problem Description, a Google form was created and sent to the end user.

The form demonstrated the functionality of the program to the end user via a video clip, which gave a guided demonstration of how the program worked.

The end user was then asked some questions related to the prototype, and the statistics from that response were then collected and analysed.

Below are screenshots of the form, and the end user's response to it.

The form used to gather data feedback from the end user

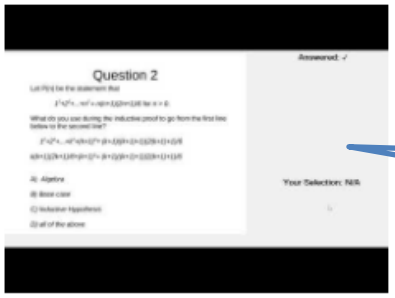
Comp 4

Section 1 of 2

'Randomised Quiz' prototype

This section of the form will ask a few questions related to the suitability of the 'Randomised Quiz' model that has been designed as a prototype.

Please watch the following video. This video gives an overview of how the 'Randomised Quiz' section of the program will work.



A video demonstrating the 'Randomised Quiz' section was included in the form to give an overview of how the prototype works.

Of the two types of questions presented in the prototype, which is the most useful to test the student on their mathematical ability.

- 'Multiple Choice' style of questions
- 'Fill in the missing blank' style of questions
- Both of the above

On a scale of 1 to 5, how does the inclusion of a mark scheme in the prototype add value to the student's overall learning experience?

	1	2	3	4	5	
It adds no value whatsoever	○	○	○	○	○	It adds significant value

On a scale of 1 to 5, how clear is the 'Answered' label at indicating to the user whether or not they have answered a particular question?

	1	2	3	4	5	
Not clear at all	○	○	○	○	○	Very clear

Do you have any additional comments regarding this section of the prototype?

Long-answer text

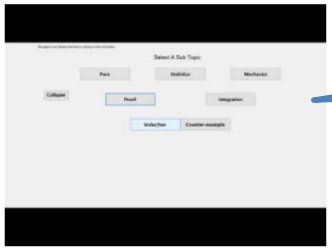
Comp 4

Section 2 of 2

'Select a subtopic' prototype

This section of the form will ask a few questions related to the suitability of the 'Select a subtopic' model that has been designed as a prototype. Please note that once the user has navigated to one of the subtopic buttons in the final program, clicking on that button will bring up a quiz containing questions related to that particular subtopic.

Please watch the following video. This video gives an overview of how the 'Select a subtopic' section of the program will work.



A video demonstrating the 'Select A Subtopic' module was also included.

On a scale of 1 to 5, how user-friendly is the 'reverse triangle' structure presented in the prototype for allowing the student to easily navigate to a subtopic of their choice?

	1	2	3	4	5	
It is not user-firendly at all	○	○	○	○	○	It is very user-friendly

Do you think that a 'Collapse' button to reset the screen to its former state is better suited to navigating to a specific subtopic than a file system such as the one used in Windows (as shown in the picture below)?

An image of the windows file system for comparison was included in the question.

- ○ Yes
- ○ No
- ○ Neither function is more suitable than the other

Do you have any additional comments regarding this section of the prototype?

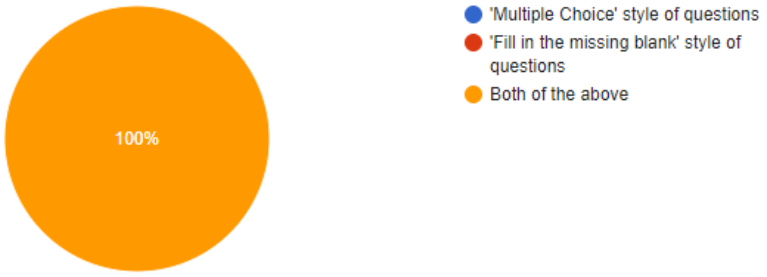
Long-answer text

<u>End user's response to the form</u>

Comp 4

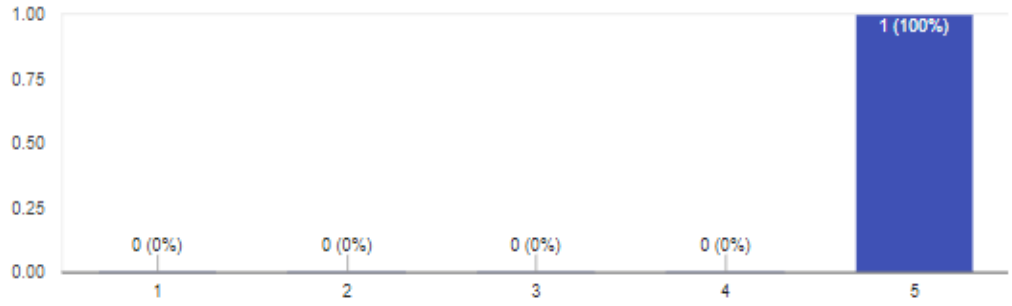
Of the two types of questions presented in the prototype, which is the most useful to test the student on their mathematical ability.

1 response



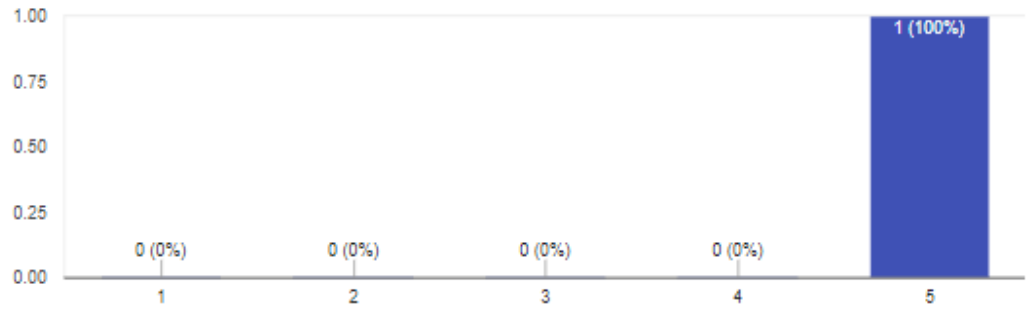
On a scale of 1 to 5, how clear is the 'Answered' label at indicating to the user whether or not they have answered a particular question?

1 response



On a scale of 1 to 5, how clear is the 'Answered' label at indicating to the user whether or not they have answered a particular question?

1 response



Comp 4

Do you have any additional comments regarding this section of the prototype?

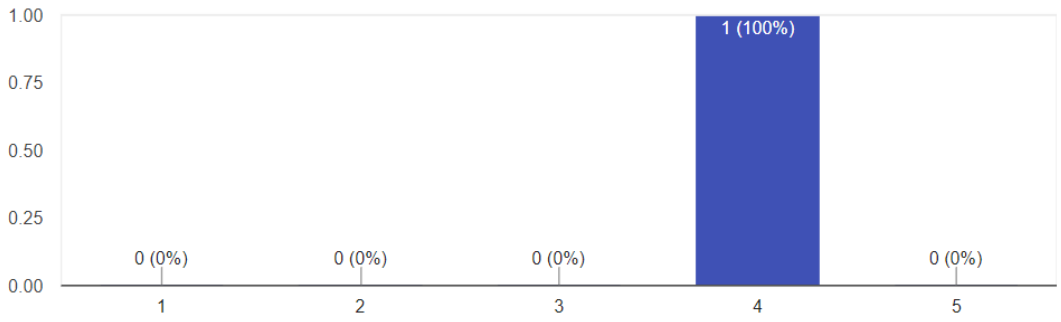
1 response

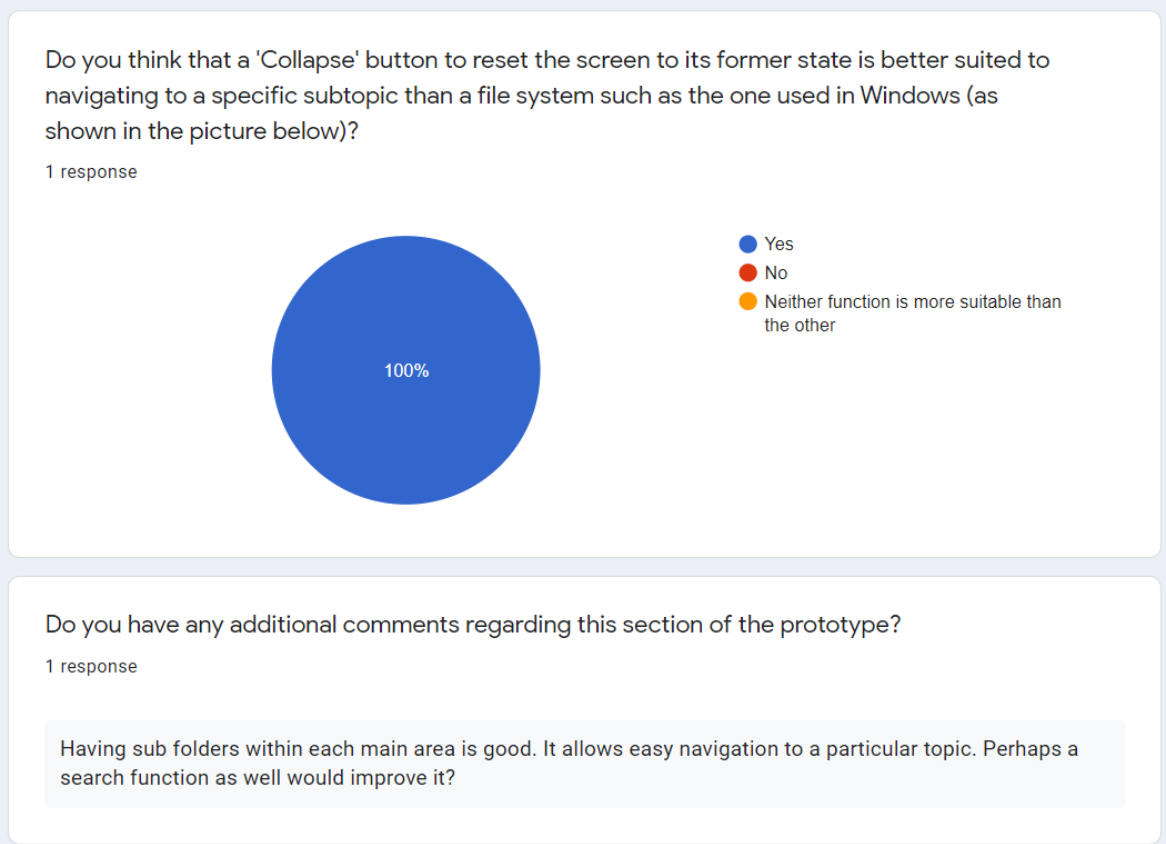
Seems very easy to use eg submission of answers and navigation between questions. I alos like the way you can only see the mark scheme when you have clicked on End quiz.

'Select a subtopic' prototype

On a scale of 1 to 5, how user-friendly is the 'reverse triangle' structure presented in the prototype for allowing the student to easily navigate to a subtopic of their choice?

1 response





Analysis of user feedback

Overall, the feedback received from the end user was largely positive, which is a good indicator that the proposed program will solve the majority of the problems described in *1.2 Background to the problem and research complete*. Going forward, I will keep the core design of the program for the *Student* user type similar to that of the prototype, and will expand upon areas that need improvment, such as the UI. For the *Teacher* section of the program, the set of requirements that have been described in *1.7 Final user requirements* will be used as the basis upon which to program the statistical models needed in each of the *Teacher* modules.

The feasability of the possible inclusion of a search function for the *Select A Subtopic* module will be researched further to determine whether or not it makes the user experience more enjoyable and simplistic.

1.7 Final User Requirements

1.7.1 Updated list of user requirements

After creating the prototype and gathering feedback from the end user, some of the requirements for the program have changed, both to make the aims of the program more realistic within the given timeframe and to make the program more user-friendly. A compiled

list of the final user requirements is given below. Requirements which have been added are highlighted like this, and requirements which have been removed are ~~crossed out like so~~.

The newly updated proposed new system, based around testing student's A level mathematical skills, will be designed to meet the following criteria:

- A login system consisting of a username and password, that will allow users access to the program from the machine the program is installed on.

  1a) The ability to create an account from within the program.

  1b) The login system will consist of two different styles of user accounts: an account for teachers and an account for students.

  1c) Personal data, including the username, password, forename and surname ~~and email address~~ of each person using the program, will be taken in by the program upon signup and stored securely in a text file ~~database~~ using encryption.

- ~~A consistent colour scheme and background image will be used across both categories of user accounts to make the program easier to navigate.~~ Different background images and colour schemes will be used to more easily differentiate between the different types of user accounts.

- A clean, simple user interface that will allow for easy navigation of the different kinds of quizzes will be built for the *Student* user type. Features that A Level mathematics students did not use in the Maths Made Easy website (as described in 1.2 Background to the problem and research complete) will be excluded from the program and features that they do use will be made more prominent to the users.

- The UI for the **Student** user account will consist of a navigation bar at the top of the program which allows the user to navigate between the two different **Student** modules.

  ~~The UI for the student account will consist a menu screen with buttons that will enable the student to navigate to their desired sub-topic to by selecting a module.~~

- The option for a student to select a subtopic from a list of available subtopics from within the **Select A Subtopic** section of the program.

  Upon clicking on a module button followed by a topic button, the student will be presented with a list of sub-topics that each contain a quiz consisting of questions exclusively related to that sub-topic.

- An easy-to-navigate UI will be created for the **Teacher** user account that will allow the teacher to easily view a range of statistics collected by the program.

- The UI for the teacher user account will consist of a navigation bar at the top of the program which allows the user to navigate between the three different **Teacher** modules.

~~The UI for the teacher user account will consist of a menu screen containing a search bar.~~

- ~~This search bar will be used to search for specific students.~~ The teacher will be able to select a student from within the **Student Overview** module.
Upon selecting a student, data gathered on that student by the program will be presented to the teacher in both graphical and numerical forms.

- The menu will also consist of a **Class Overview** module, which will allow the teacher to track the class's progress as a whole, and will compare the students in each class with each other.

The statistics for the **Class Overview** will be displayed to the teacher in both graphical and numerical forms to make the data more helpful and easier to understand.

- A collection of practice questions which assess a wide range of the topics and skills needed for A Level Mathematics are stored in the program folder and can be accessed by the program whilst it is running. These questions will be used during the quizzes.

~~*These questions will initially be pre-set during programming. However, the end user has specified that if there is enough time to incorporate the feature, having the ability to add questions to the collection of questions stored by the program whilst the program is running would be an added bonus.*~~

- A mixed quiz consisting of a variety of different types of practice questions from a range of different topics.

~~*11a) The questions will be selected based off the individual student's areas of strengths and weaknesses, which will be gathered from the student's previous attempts at questions.*~~
~~*11b) If insufficient data is held about the student, then practice questions will be randomly generated and presented to the student in order of difficulty.*~~

This specific set of user requirements was deemed to be too difficult to achieve during the creation of the prototype, so will not be included. Instead, all questions in a particular quiz will be randomly generated by the program based off of the subtopics which are viable to be included in the quiz.

- ~~A mark scheme for each individual question, which will be presented to the student immediately after they have submitted their answer to the question. A mark~~

~~scheme containing worked solutions for all the questions that were in the test will also be made available to the student at the end of the test.~~

The mark scheme will be presented to the student in the same format as the questions will be such that the student will be able to cycle between the mark scheme images by clicking on the 'Next' and 'Previous' buttons.

The mark scheme will be presented in the same way as the exam board the end user teaches (Edexcel) present their mark schemes.

- Questions will be input into the program in the following ways:
  - Input boxes to input numerical answers for worked questions.
  - Tick boxes to answer multiple choice questions.

- When inputting numerical values into the program, the following syntax will be used:
  - Square root: Sqrt(value)
  - Raising to an exponent: x^(value)
  - Logarithms: log/base/value, ln/base/value

- The font and button size will be large and obvious so that the program is more accessible to users with certain learning difficulties or poor eyesight. Where appropriate, text will be in bold so as to draw attention to certain parts of the program.

- ~~The skills required to answer each question will be presented to the student on every question.~~

  a. ~~*This will be in the form a list, containing bullet points of the key skills required to answer the question.*~~
  b. ~~*These skills will also be used by the algorithm that will determine the difficulty level of each question.*~~

  This particular requirement was deemed unnecessary, as it is unlikely to help the student in learning how to answer the questions presented to them due to the questions being either 1-line answers or multiple choice questions.

1.7.2 Acceptable limitations of the system

The following is a list of features that are not essential to the program design, and so are acceptable to leave out during the design phase of the program.

- For the **Teacher** user account, the ability to compare specific classes to each other will not add significant value to the user experience, so it will not need to be included in the statistics that will be displayed to the teacher.

- The ability for a teacher to add their own questions to the collection of questions that can be displayed to the student during a quiz is not pivotal to the program design, so will only be included if there is spare time left over at the end of the project.

1.7.3 Realistic appraisal of the feasibility of potential solutions

Incrementing a Microsoft Access database into the program as an alternative to text files could potentially make the process of reading and writing data between the program and the data storage method more efficient. Furthermore, it could also make the data easier to view and edit, as an administrator of the program would be able to open up the database and edit the database however they choose. If a text file were used then the data would be stored in a less user-friendly format, resulting in the data being more difficult to read and understand if it needed to be edited by an administrator.

However, it would present significant challenges in the following areas:

- Researching and implementing the code which would need to be input into the program in order to link the database. Whilst conducting some initial research on the feasibility of incorporating a database into the project, it was found that the most efficient way to do so was by use of a data class. As this is a programming technique which I have not yet tried, attempting to incorporate a data class into the program could lead to unexpected errors due to overlooking certain aspects of how a class works.
- Planning out which section of the database will be used to store which data. As there are two different user accounts, determining if separate tables are needed for e.g. different user accounts would take extensive planning, and the need for testing the different methods of reading and writing data to the database may give rise to the need to carry out additional tests to determine which method is the most efficient.
- Deciding whether or not to have two (or three if a separate database for the login details is to be included) separate databases for the **Teacher** user account and the **Student** user account. As the data stored about the two different types of users will be fundamentally different (aside from the login details), storing the data across several different databases would perhaps make more sense on the security side of the program, as if the database holding data about one of the user types was somehow compromised, the data held about the other user type would still be safe. However, this would present the further difficulty of linking the two databases to each other, and then linking them to the Visual Studio IDE, which would present a significant challenge on the technical side of project.

The alternative to a database would be to use file handling which, for the purpose of this project, is deemed to be the most suitable method of storing the data. This is because the volume of data taken in by the program is relatively low compared to other programs which make use of a database, so the increased efficiency at reading and writing data when using a database instead of a text file will likely be negligible. Furthermore, using text files also allows for the data stored about each student to be stored in its own individual text file, which would help an administrator to easily edit the data stored on one particular student, instead of them having to search a database for that student.

Additionally, as the **Teacher** section of the program is built around displaying statistics gathered from students taking quizzes, ensuring that there is a sufficient level data that can be used by the program to generate these statistics is pivotal to thoroughly testing this aspect of it. Manually taking many quizzes over a range of different user accounts will take a significant amount of time. Instead, quiz results will be manually input into the **Scores** text file for each of the students in the test sample, to be read in by the program when a teacher wishes to generate statistics for that specific student, or for a class in which that student is a part of.

1.7.4 Justification of chosen solutions

The program will be written using a series of text files to store information about students scores, students account details and teachers account details. Aside from being significantly simpler to implement and, at its core, achieving the same task as a database does (storing data to be accessed upon request), the benefit of increased efficiency at reading and writing data that is gained from using a database will have very little, if any, improvement to the overall function of the system, given the low volume of data that will be used in all areas of the program.

I have also decided to use Visual Basic to program the *Mathematics Testing Program* in. Part of this decision is because this language works well with the Visual Studio IDE, which has the advantage of offering a Forms application. The use of different forms to show the different sections of the program is a simple and intuitive method of implementing the user interface, as a smooth transition between one form and the next will allow the users of the program to be guided through the program in a user-friendly manner. Ensuring that the program is easily accessible for people without any technical training whatsoever is important, because the end user does not wish for students to miss out on their mathematics education because they have difficulties navigating the system (which happens to be an underlying issue with the *Maths Made Easy* website which was analysed in 1.2 Background and Research.

Furthermore, the Visual Basic programming language is relatively simple to program in, and there is a good level of documentation for the language available on the internet should I come across any issues when programming. Using the Visual Studio IDE will allow me to make use of MDI Parent and Child forms, which will allow for tabulated pages which the user can easily navigate between to access different parts of the program for their user type.

Design

2.1 Modular structure of the system

The program will be split into five separate modules; two of these modules will be for the student user-type section of the program, and the other three modules will be for the teacher user-type section. These modules will be made accessible to the relevant user-type from a tab interface, and the labels on the tabs will vary between two separate user-types. The relevant home screen can be accessed once the user has successfully passed through the login section of the program.

The first module will be a randomised quiz option for the student user-type. This page will contain a button that, when loaded up, will generate a quiz containing a fully randomised quiz containing five questions from any of the subtopics used in the program.

This module will also gather data from the student's inputs at run-time and then display that information in graphical and numerical forms to the **Teacher** user type in the modules described below. As the student is likely to improve on some topics and fall behind on other topics over time, the data collected will help to track the student's progress and show the correlation between their progress in quizzes and their number of attempts.

The second module will also be for the student user-type. This module will provide a search tool for students to navigate to specific sub-topics and will present students with a set of practice questions related to that sub-topic.

A tree structure will be used as an alternative to a file system (such as the one used in the Windows operating system) because it offers a simpler method of navigating to a desired subtopic than a folder structure.

This tree structure will consist of a row of three buttons with the module names at the top. When one of these module buttons is clicked on, three topic buttons will be generated related to the module. When one of these topics buttons is clicked on, two subtopic buttons will be generated that are related to the selected topic. This is opposed to a search bar or a scroll bar, as both of these methods have their flaws. The student may not know the exact name of the topic they're looking for, so the search bar would be difficult to use in this particular situation. The scroll bar would increase the time taken to search for a particular topic, and so would make the page feel busier and more confusing to navigate.

The third module will be for the teacher user-type. This module will allow the teacher to add a class to the list of classes that they teach. It will in the names of the students that the teacher wishes to add to the new class along with the name that the teacher wishes to give

the new class and then and store this information in a data structure. Instead of needing to type out the names of each student in the class separately, a list box containing a list of all the students that have signed up to the program will be incorporated, and the teacher can select students from the scroll bar to add to a list of students they want to add to their new class. Furthermore, the option to search for a student by their ID will be included as an alternative to using the 'Add' and 'Remove' buttons so that the teacher can choose the most efficient way to add student's to their new class.

This module will also display a list of the other classes that have been created by teachers so that the teacher does not attempt to create a class that is similar to one that has been previously created.

The fourth module will also be for the teacher user-type. This module will allow the teacher to view the overall performance of their classes over time. The teacher will be able to select a class and then have relevant statistics about that class displayed to them. These stats will include a percentage increase or decrease for each topic over time and will also include the average performance in the current academic year of each class on each of the topics. Statistics will be displayed both arithmetically and graphically in order to help the teacher to extract relevant information from them.

The fifth module, again for the teacher user-type, will consist of a series of graphs and information that will allow the teacher to gain an insight into a specific student's progress by selecting their name from a list of students that are signed up to the program.

The information available to the teacher will consist of relevant statistics collected by the program that are specific to that particular student. This information will be displayed in an output window in both arithmetic and graphical forms for the teacher's ease of understanding. The relevant statistics that will be displayed will include a table depicting the student's average score in each module and a bar chart and line graph that will show the percentage score over all the quizzes taken for each student in the class. The teacher will be able to select the subtopic for which they wish to view the students score on and each time a new subtopic is selected by the teacher the contents of the graph will be updated to display the relevant scores.

Hierarchy charts

The following diagrams are hierarchy charts for both the current and proposed systems. Each branch represents a possible path that a user could take when navigating through the program.

Note: Branches which end on sections of the website that are not related to A Level maths have not been continued because they are not relevant to the end user's requirements. However, these portions of the website are organised in a similar manner to the sections of the website that are related to A Level maths.

Figure 1: The hierarchy chart for the current system

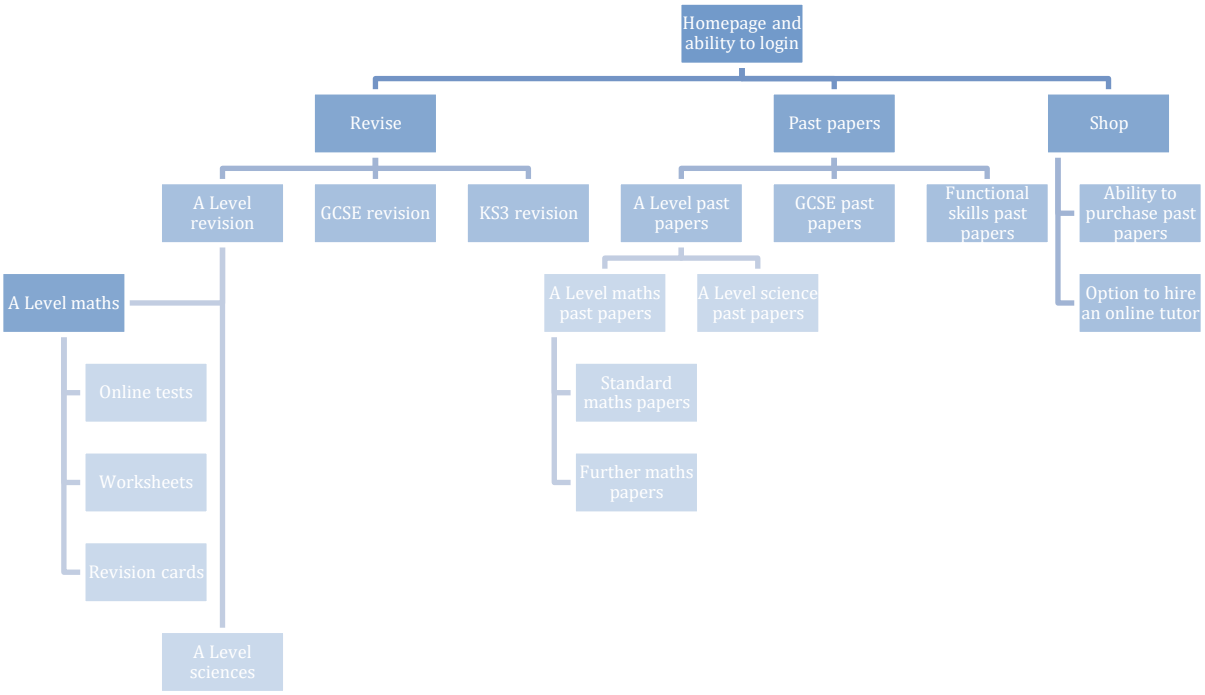
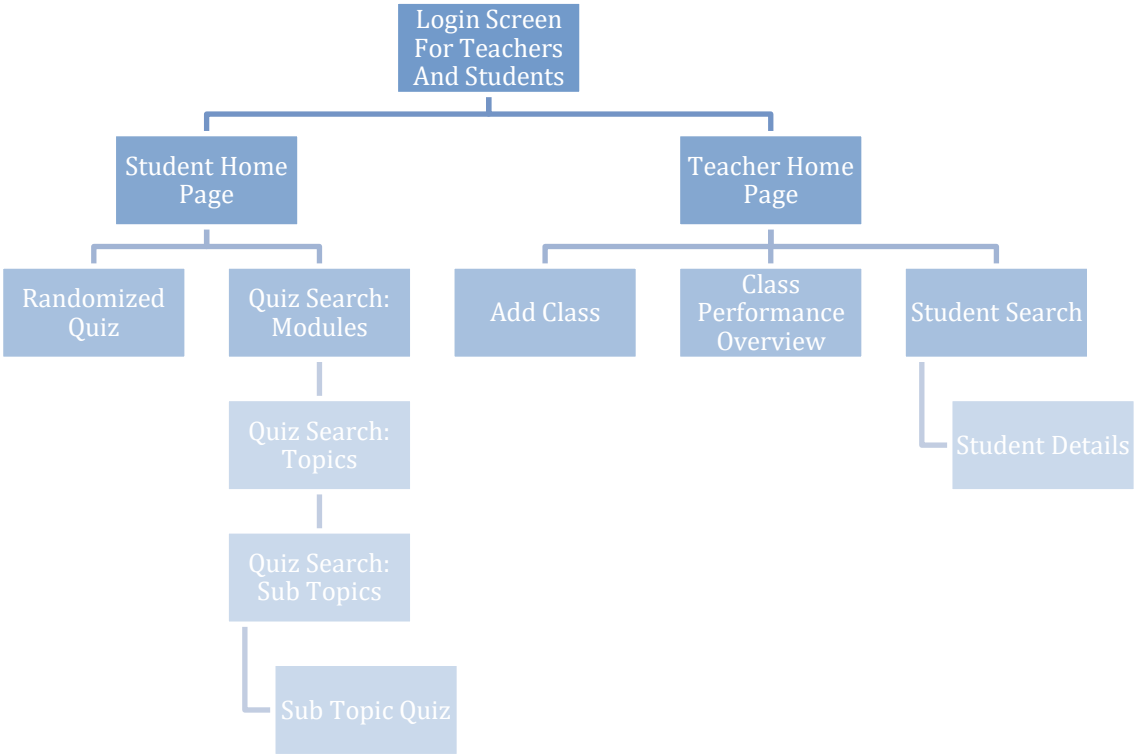


Figure 2: The hierarchy chart for the proposed system



2.2 Data structures and data dictionary

Data structures

Structure: **AccountDetails**
- `Username`: String of fixed length 20
- `Password`: String of fixed length 20
- `Forename`: String of fixed length 20
- `Surname`: String of fixed length 20

Structure: **WriteAccountDetails**
- `filename`: String
- `EncryptedUsername`: String of fixed length 20
- `EncryptedPassword`: String of fixed length 20
- `EncryptedForename`: String of fixed length 20
- `EncryptedSurname`: String of fixed length 20

Structure: **LoginValidation**
- `InputDataValid`: Boolean
- `CorrectFieldsFilled`: Boolean
- `StudentUsernameEmpty`: Boolean
- `StudentPasswordEmpty`: Boolean
- `TeacherUsernameEmpty`: Boolean
- `TeacherPasswordEmpty`: Boolean
- `Student`: Boolean
- `Teacher`: Boolean
- `LoginDetailsFound`: Boolean
- `filename`: String
- `username`: String
- `password`: String
- `DecryptedCipherText`: String

Structure: **Question**
- `QuestionTitle`: String
- `QuestionType`: String
- `NumberOfButtons`: Integer
- `ModuleType`: String
- `Topic`: String
- `SubTopic`: String
- `QuestionNumber`: Integer
- `CorrectAnswer`: Boolean
- `DifficultyLevel`: Integer
- `IsAnswered`: Boolean

`IsCorrect`: Boolean

Structure: **TestResults**
- `TotalScore`: String
- `PureScore`: String
- `StatisticsScore`: String
- `MechanicsScore`: String
- `ProofScore`: String
- `IntegrationScore`: String
- `CollisionsScore`: String
- `EnergyScore`: String
- `InductionScore`: String
- `CounterexampleScore`: String
- `Integrating_E_Score`: String
- `Integrating_LN_Score`: String
- `VennDiagramsScore`: String
- `LawsOfProbability`: String
- `BinomialScore`: String
- `NormalScore`: String
- `ElasticIn2DScore`: String
- `ObliqueScore`: String
- `KineticScore`: String
- `GravitationalScore`: String

Structure: **CreateNewClass**
- `ClassName`: String of fixed length 5
- `ClassSize`: Integer of fixed length 2
- `Students()`: Array containing the names of students that are to be added to the new class.

- `NumOfAttempts`: String

Structure: **SelectedFolders**
- `MathsModule`: String
- `Topic`: String

Structure: **SearchListBox**
- `SearchID`: Integer
- `FullStudentName`: String
- `DecryptedForename`: String
- `DecryptedSurname`: String
- `StudentAlreadyInList`: Boolean

Data Classes

Class: **StudentInformation**
- StudentName: String
- StudentScores: String
- StudentAttempts: String

Data Dictionary

The following is a data dictionary for all the variables in the *Create An Account* and *Login* modules.

Variable Name	Description	General Data Type	Example	Validation
`Username`	The user's username.	String (fixed length 20 characters)	"NewtonApple"	**Presence check, length check** Maximum length of 20 characters permitted.
`Password`	The user's password.	String (fixed length 20 characters)	"Physics_Student123"	**Presence check, length check** Maximum length of 20 characters permitted.
`Forename`	The user's forename.	String (fixed length 20 characters)	"Johnathon"	**Presence check, length check** Maximum length of 20 characters permitted.

Comp 4

Variable Name	Description	General Data Type	Example	Validation
`Surname`	The user's surname	String (fixed length 20 characters)	"Rodgers"	**Presence and length check** Maximum length of 20 characters permitted.
`filename`	The name of the file that the signup data will be written too.	String	"filename1.txt"	**N/A** (Automatically assigned based on selected user type.)
`EncyrptedUsername`	The username entered by the user into the input box, which then has a Caesar shift performed upon it.	String	"X~@JH"	**Presence and length check** There must be a data entry for username in the input box, this data entry must be less than 20 characters long and it cannot be in use by another user.
`EncryptedPassword`	The password entered by the user into the input box, which then has a Caesar shift performed upon it.	String	"^YU)S"	**Presence and length check** There must be a data entry for password in the input box, this data entry must be less than 20 characters long and it cannot be in use by another user.
`EncryptedForename`	The forename entered by the user into the input box, which then has a Caesar shift performed upon it.	String	"@90i1"	**Presence and length check** There must be a data entry for forename in the input box and the data entry must be less than 20 characters long.

Comp 4

Variable Name	Description	General Data Type	Example	Validation
`EncryptedSurname`	The surname entered by the user into the input box, which then has a Caesar shift performed upon it.	String	“7Up£G”	**Presence and length check** There must be a data entry for forename in the input box and the data entry must be less than 20 characters long.
`InvalidUsername`	This variable used to hold the result of validating whether the user’s input into the username box is valid.	Boolean	True	**N/A**
`i`	A Counter variable	Integer	1	**N/A**
`NextRecord`	Holds the record number, which determines the position in which to write the *NewUser* structure to the signup details text file.	Integer	5	**N/A**
`EncryptionFactor`	Determines the shift that will be used when performing a Caesar shift on the data that has been input by the user.	Integer	3	**N/A**
`InputDataValid`		Boolean	False	**N/A**
`CorrectFieldsFilled`		Boolean	False	**N/A**
`StudentUsernameEmpty`	This variable is used to determine whether there is an input in the student username input box.	Boolean	True	**Presence Check** Designed to check for the presence of data in the student username input box.

Variable Name	Description	General Data Type	Example	Validation
`StudentPassword Empty`	This variable is used to determine whether there is an input in the student password input box.	Boolean	True	**Presence Check** Designed to check for the presence of data in the student password input box.
`TeacherUsername Empty`	This variable is used to determine whether there is an input in the teacher username input box.	Boolean	False	**Presence Check** Designed to check for the presence of data in the teacher username input box.
`TeacherPassword Empty`	This variable is used to determine whether there is an input in the teacher password input box.	Boolean	False	**Presence Check** Designed to check for the presence of data in the teacher password input box.
`Student`	For this variable to be true, the student username and password variables must both be true, and the teacher username and password variables must both be false.	Boolean	True	**N/A**
`Teacher`	For this variable to be true, the teacher username and password variables must both be true, and the student username and password variables must both be false.	Boolean	False	**N/A**

Variable Name	Description	General Data Type	Example	Validation
`DecryptedCipherText`	Holds the result of decrypting the cipher text for a specific part of one of the text files containing login details.	String	“JoeBloggs”	**N/A**
`LoginDetailsFound`	This variable is used to hold the result of whether the login was successful.	Boolean	True	**N/A**

The following is a data dictionary for all the variables in the *Randomised Quiz* and *Select A Subtopic* modules:

Variable Name	Description	General Data Type	Example	Validation
`QuestionList`	Holds the names of the randomly generated questions and is used to retrieve those questions from the file system.	List(Of String)	QuestionList(“Induction 1”, “Binomial1”, “Normal2”, “Kinetic1”, “Graviational2”)	**N/A**
`FromSubTopic`	This variable is ensure that all the questions selected for the quiz are valid subtopic names.	Boolean	True	**Presence Check** Ensures that only valid question names are present in the quiz.
`MultipleChoiceAnswer`	Holds the letter that the student has selected when answering a *Multiple Choice* style of question.	String	“A”	**N/A**
`NextQuestionNumber`	Used to determine which question the quiz should display once the ‘Next’ or ‘Previous’ buttons are clicked.	Integer	1	**N/A**
`Rndm`	Used to randomly select a question name.	VB Random		**N/A**

Comp 4

Variable Name	Description	General Data Type	Example	Validation
UserAnswer	Holds the user's answer to the *Fill In The Missing Blank* style of question.	String	True	**Presence Check, Format** Ensure that the user's answer is present, and that it follows the input rules of the program.
Markscheme	Holds the result of whether the user selected to view the mark scheme, and then generates the mark scheme if true.	Boolean	True	**N/A**
SW	Used to the student's results for the quiz to their results text file using the VB StreamWriter method.	VB Stream Writer	**N/A**	**Presence Check with Try Catch** If the text file cannot be accessed then the program safely catches the error with a try catch and does not crash.
QuestionTitle	Holds the name of a specific question.	String		**N/A**
QuestionType	Holds the style of question for a specific question.	String	"Multiple_Choice" or "Fill_In_The_Missing_Blank".	**Presence Check** If the question type is not one of the two possible options the question is not loaded.
NumberOfButtons	This variable is used to hold the number of multiple choice answers to a specific question, which is then used to indicate to the program how many buttons to dynamically generate.	String	4	**N/A**

Comp 4

Variable Name	Description	General Data Type	Example	Validation
`ModuleType`	Holds the name of the module that the specific question belongs too.	String	“Mechanics”	**N/A**
`Topic`	Holds the name of the topic that the specific question belongs too.	String	“Energy”	**N/A**
`SubTopic`	Holds the name of the subtopic that the specific question belong too.	String	“Kinetic”	**N/A**
`QuestionNumber`	Assigned a number based on the order that the specific question was generated (e.g. if it was the first question generated then `QuestionNumber` would be 1.	Integer	1	**N/A**
`CorrectAnswer`	Holds the correct answer to a specific question.	String	“1/2*m*v^2”	**Format Check** The contents of the `CorrectAnswer` variable must follow the input rules of the program.
`DifficultyLevel`	This variable is used to rank the difficulty level of a specific question on a scale of 1 to 5.	Integer	5	**N/A**

Comp 4

Variable Name	Description	General Data Type	Example	Validation
`IsAnswered`	This variable is used to track whether a specific question has been answered by the student in a quiz.	Boolean	False	**Presence and Format Check** The question must have been answered by the student, and must follow the input rules of the program if the answer to to a *Fill In The Missing Blank* style of question.
`IsCorrect`	This variable is used to track whether the student has correctly answered a specific question in a quiz.	Boolean	False	**Format Check** This variable is compared to the user's answer to identify whether or not they are the same.
`btnMultipleChoice(NumberOfButtons)`	This list is used to hold the information needed to generate the multiple-choice dynamic buttons.	List(Of Button)	`btnMultipleChoice(1) = New Button`	**N/A**
`btnAdditionalControls(5)`	This list is used to hold the information needed to generate the additional controls dynamics buttons.	List(Of Button)	`btnAdditionalControls(1) = New Button`	**N/A**
X	Holds the X-Coordinate used for positioning dynamically generated buttons on the screen.	Integer	900	**N/A**
Y	Holds the Y-Coordinate used for positioning dynamically generated buttons on the screen.	Integer	300	**N/A**
`count`	This variable is used to loop through the list of buttons.	Integer	1	**N/A**

Comp 4

Variable Name	Description	General Data Type	Example	Validation
`letters`	This variable is used to assign a letter to each of the generated multiple choice buttons.	String	"ABCDEFGH"	**N/A**
`txtAnswer`	Holds the details used to dynamically generate the input box for answering the *Fill In The Missing Blank* style of questions.	`System.Windows.Forms.TextBox()`	`Me.Controls.Add(txtAnswer)`	**N/A**
`ContainsTextbox`	Used to determine whether the form currently contains a text box; if so, the contents of the form will be wiped ready to be populated with multiple choice buttons instead.	Boolean	True	**Presence Check** Multiple choice buttons can only be generated once any input boxes are removed from the form.
`NextQuestion`	This structure is used to cycle through the different questions (and answers once the quiz is ended).	A `Question` structure (described above)	**N/A**	**N/A**
`TotalScore`	Holds the total score of the students result in the quiz, to be displayed to the student once the quiz is ended.	Integer	3	**N/A**
`Successful`	This variable is used to determine whether the result of reading the student's information from a text file is successful.	Boolean	True	**N/A**
`ViewMarkScheme`	This variable is used to hold the result of whether the user wishes to view the mark scheme; if True, a mark scheme is generated.	Boolean	True	**N/A**

Comp 4

Variable Name	Description	General Data Type	Example	Validation
`forename`	Holds the forename of the student who is currently signed into the program.	String	"Bob"	**Presence and length check** Maximum length of 20 characters permitted.
`surname`	Holds the surname of the student who is currently signed into the program.	String	"Ross"	**Presence check, length check** Maximum length of 20 characters permitted.
`filename`	This variable is used to hold the filename for the student who is logged in and taking the quiz, so their results can be written to their file.	String	"BobRoss.txt"	**Presence check, length check** Comprised of the `forename` and `surname` variables, so goes through the same validation checks as those variables do.
`avgScore`	Holds the result of the calculation of the student's average score over all the quizzes taken by that student.	String	"/00085" – meaning 85%	**Presence, length and format check** The scores stored in the student's text files must all be present (meaning the text file must be a set length), each score must comprise of 5 numbers and these scores must be preceded by a backslash.

Comp 4

Variable Name	Description	General Data Type	Example	Validation
`ResultsString`	This variable temporarily holds a section of the student's text file relating to a part of their score, so that score can be incremented by one if the student scored a correct answer in a particular area.	String	"/00010/00050" – meaning 10/50	**Presence, length and format check** The scores stored in the student's text files must all be present (meaning the text file must be a set length), each score must comprise of 10 numbers and these scores must be preceded and ended with a backslash.
`ResultsSubstring`	This variable holds one of the sides of `ResultsSubstring` in order to increment the value using the `score` variable.	String	"/00010"	**Length and format check** This string must comprise of 5 numbers and must be preceded with a backslash.
`ModuleNames`	This list to hold the names of the different modules being assessed, from which a module will be randomly selected by the program in order to generate a question.	List(Of String)	`ModuleNames("Pure Maths", "Mechanics", "Statistics")`	**N/A**
"X"`TopicNames`	Where X is the name of a topic. These lists are used to hold the names of the topics being assessed, from which a topic will be randomly selected by the program in order to generate a question.	List(Of String)	`PureMaths("Integration", "Proof")`	**N/A**

Comp 4

Variable Name	Description	General Data Type	Example	Validation
“Y”SubtopicNames	Where Y is the name of a subtopic. These lists are used to hold the names of the subtopics being assessed, from which a subtopic will be randomly selected by the program in order to generate a question.	List(Of String)	Proof(“Induction”, “Counter-example”	**N/A**
FilePath	This variable holds the file path for the set of questions selected in the *Select A Subtopic* module.	String	“C:/Users…/Mechanics/Energy/Kinetic”	**Presence Check** A module, topic and subtopic name must be present in order to generate a valid file path.
MathsModule	Holds the name of the module selected by the student in the *Select A Subtopic* module, and is used to dynamically generate the relevant Topic buttons.	String	“Mechanics”	**N/A**
Topic	Holds the name of the topic selected by the student in the *Select A Subtopic* module, and is used to dynamically generate the relevant Subtopic buttons.	String	“Energy”	**N/A**
SubTopicQuestions	This list holds the names of the set of subtopics that have been selected by the student in the *Select A Subtopic* module.	List(Of String)	SelectSubTopicQuestin(“Kinetic”, Graviational”)	**N/A**

The following is a data dictionary for all the variables in the *Create A Class Module* (excluding the variables described above):

Variable Name	Description	General Data Type	Example	Validation
`Classname`	The name that the new class will be given. This is used when selecting a class in the *View Class Progress* module.	String (fixed length 5)	"13/CS"	**Presence, length and format check** The teacher is not able to create a class until a class name has been selected. The class name must also be five characters long and start with two numbers followed by a backslash.
`Classize`	The number of students in the new class.	String (fixed length 3)	"012"	**Length and format check** The class size must be a string of three letters, with preceding zeros if the integer value is less than a three figure value.
`Students`	A combination of	String (fixed length 150)	"James Murray, Amy Ophilieo, Bob Ross"	**Presence and length Check** At least one student must be selected for the teacher to proceed with creation of a new class. This string must also be less than or equal to 150 characters.

Variable Name	Description	General Data Type	Example	Validation
`SearchID`	This is used to search for a student name by direct access.	Integer	1	**Range Check** The search ID is checked to ensure that it is greater than zero and less than the number of students signed up to the program.
`StudentAlreadyInList`	This variable is used to validate that the student selected by the teacher to add to the new class has not already been added.	Boolean	False	**Presence Check** This variable is set to false if the student is already present in the new class list box.
`FullStudentName`	The combination of the first name and surname of the student that the teacher has selected to add to the new class.	String	“James Murray”	**Presence and range check** The selected index of the student is checked to ensure that it is greater than zero and less than the number of students signed up to the program, and that it is not already in the list of selected students for the new class.

Variable Name	Description	General Data Type	Example	Validation
`plaintext`	This holds the decrypted contents of the "*students.txt*" text file.	String	**N/A**	**Presence and range check** The text file cannot be empty, and the number of characters in the text file must a fixed multiple of a specific length due to the structure used to read data from the text file, which contains variables of fixed lengths.
`NextRecord`	Holds the record number that is used to write data to the "*classes.txt*" text file.	Integer	1	**N/A**
`Numbers`	This is used to validate that the class name contains numbers as its first two letters.	String	"1234567890"	**Format Check** The class name must have any one of these numbers as its first two letters.

The following is a data dictionary for all the variables in the *View Student Progress* and *View Class Progress* modules (excluding any variables described above):

Variable Name	Description	General Data Type	Example	Validation
`SR`	This variable is used to read data from the student *Results* and *Attempts* text file.	Visual Basic Stream Reader	**N/A**	**N/A**
`ResultsLine`	This variable holds the result of reading a line from a student's *Attempts* text file.	String	`ResultsLine = SR.Readline()`	**Presence check** This variable is checked to ensure that it is not empty; if it is found to be empty the point is not plotted.

Comp 4

Variable Name	Description	General Data Type	Example	Validation
`formgraphics`	This variable is used to draw the axis on the *View Student Progress* module.	Visual Basic Drawing. Graphics	`formgraphics .DrawLine(Pens.Blank, 100, 200, 100, 400)`	**N/A**
`Y1 and Y2`	These variables are used to change the position of the pointer on the y-axis in order to draw the line graph.	Integer	100	**Range check** These variables cannot exceed the height in pixels of the full-screen Visual Studio form.
`lblXAxis(10) and lblYAxis(10)`	Used to label the X and Y axis respectively.	Visual Basic Label	**N/A**	**N/A**
`loc(10)`	Holds the points which are to be plotted on the drawn-on axis.	Visual Basic Point	(100, 200)	**N/A**
`avg1 and avg2`	These variables are used to populate the bar chart in the *View Student Progress* module.	Integer	85(%)	**Range check** This average must be less than or equal to 100(%), otherwise it is not accepted as a valid average.
`subtopic1 and subptopic2`	These variables are used to label the bar chart in the *View Student Progress* module.	String	“Binomial”	**N/A**
`StudentName`	This variable is used to hold the names of the students in the selected class sequentially (for the *View Class Progress* module) in order to label the generated graphs.	String	“JamesFerret”	**Presence check** Try Catch validation is used when reading the text files related to the student name to ensure that the files are present and not empty.

Variable Name	Description	General Data Type	Example	Validation
`StudentScores`	This variable is used to hold the scores of the student in the selected class sequentially (for the *View Class Progress* module) in order to plot the graphs.	String	**N/A**	**Presence check** Try Catch validation is used when reading the text files related to the student name to ensure that the files are present and not empty.
`StudentAttempts`	This variable is used to hold the *Attempts* for a student, in order to generate a percentage score for that student.	String	"010020030"	**Presence check** Try Catch validation is used when reading the text files related to the student name to ensure that the files are present and not empty.
`ArrayOfStudentNames()`	This array is used to hold the names of all of the selected students in the selected class.	Array of strings	`ArrayOfStudentNames("James Ferret, Chloe Green")`	**N/A**
`Results(3)`	This variable is used to hold the result of calculating the average of each of the three sections of the student *Attempts* line.	Array of integers	`Results(10, 20, 30)`	**N/A**
`first and last`	The variables that are to be swapped round when executing the quicksort function.	Integers	**N/A**	**N/A**
`j and k`	Placeholder variables used to perform the quicksort function.	Integers	**N/A**	**N/A**

Variable Name	Description	General Data Type	Example	Validation
`ClassAvg()`	This array is used to hold the overall average score for each of the students in the selected class so the graphs can be plotted.	Integers	`ClassAvg( 50, 30, 40)`	**N/A**

2.3 File handling and data processing

Overview of why text files were chosen over a database

The original idea for the program was to store the data gathered by the program in a Microsoft Access database. However, upon further analysis it was found that using a database would add very little (if any) value when compared with a series of text files.
The arguments promoting the advantages of using a set of text files over a database is as followed:

- A database is significantly more time consuming to implement than a series of text files are.
- The database code would be used in a similar manner across several different program modules, so the inclusion of a data class would be needed. This would add an unnecessary layer of complexity to the program.
- The data gathered by the program does not need to be analysed and sorted through (which is one of the major advantages of using a database over a text file), so very little, if any, improvement will be seen in terms of analysis of the collected data when using a database.
- A potential advantage of using a database would be the higher level of efficiency at gathering large volumes of data. However, the volume of data gathered by the program is relatively low so there will be no apparent advantage of a database over a text file in this area.
- Data can be stored separately across a series of text files, meaning that if one text file is compromised, the data contained in the other text files is safe. However, if an Access database is used then all of the data is stored in one place and so all of the data could be compromised or corrupted at once.
- There is a small chance of an Access database being corrupted when it is sent over the internet (which is how the program would be distributed) due to the formatting used in an Access database. However, when using a text file there is a lower chance of the data being corrupted because there is no formatting used.
- The relations between different tables in the database would be time-consuming to plan out and normalising those tables to prevent duplication of data would take even more time. Storing the data over a series of text files is a significantly simpler method of achieving the same purpose.
- Finally, text files are easier to edit and view, take up less space and can be viewed on any machine without additional software. If a Microsoft Access database were

used and a program administrator wished to view the database they would have to have Access installed on their computer.

Due to the issues discussed above with using a Microsoft Access database the program will instead use exclusively text files to store all the data that is gathered from the quizzes. The uses of text files within the *Mathematics Testing Program* are described below.

Types of text files used by the program

Creating an account and logging into the program

In order to create an account within the program, login details must be taken in by the program. These details are gathered in the *Create An Account* module, and include the forename and surname of the user along with a username and password used for signing into program and subsequently collecting data on performance within the program (for the student user type). As there are two different user types in the program, the details for each type of user are stored in separate encrypted text files, such that there is a text file containing the student login details and another containing the teacher login details.

Saving a student's scores

Each student signed up to the program has a *Scores* text file, which holds the percentage scores for each of the modules, topics and subtopics. These scores are stored in format "*/Number of times answered correctly/Total number of times attempted*" and are used to produce a variety of different graphs in the **Teacher** section of the program. This text file is not encrypted, because the data stored within the text file is not confidential.

Saving a student's score for each attempt

Each student also has an *Attempts* text file. This text file holds the percentage score for each of the three modules for each attempt at a quiz in the format "*Pure Maths %, Statistics %, Mechanics %*", so for example a quiz in which a student scores 80% in pure maths questions, 50% in statistics questions and 20% in mechanics questions would result in the string "080050020" being written to that particular student's *Attempts* text file. This is used alongside the student's *Scores* text file to generate the statistics used to analyse student's performance in the *Teacher* section of the program. Specifically, the *Attempts* text file is used to generate the data in the list box that displays the student's average overall score and their average score in each of the three modules by calculating an average of each line of the text file. Again, this text file is not encrypted, because the data stored within the text file is not confidential.

Creating new classes and storing the names of the students in those classes

Within the *Create A Class* module in the *Teacher* section of the program, the teacher is able to create a new class of students. The advantage to this is that the student's performances can then be compared against each other in the *View Class Progress* module, which will help the teacher to understand which students are struggling and which students are excelling.

To create a new class, the teacher inputs a class name and selects students to add to the class from a list box containing the names of all the student's signed up to the program. Then the user clicks the enter button, and providing that all the validation requirements are

met, the class name and names of the students in the class are written to the *Classes* text file. This text file contains the names of all the classes in the program, and creating a new class adds the details of that class to the end of the text file. It is not encrypted.

The text files used to hold question information

For each question, a text file will be used to hold the following information about a question:

- The question title.
- The style of question (*Multiple Choice* or *Fill In The Missing Blank*).
- The number of labelled buttons that need to be generated (this is zero if the style of question is *Fill In The Missing Blank*.
- The name of the module that the question falls under.
- The name of the topic that the question falls under.
- The name of the subtopic that the question falls under.
- The question number. This is used to determine the order in which to display the questions.
- The correct answer to the question.
- The variable `IsAnswered`, which is set to false and changed during the quiz based on whether or not the student has answered a particular question.
- The variable `IsCorrect`, which is also set to false and changed during the quiz. This variable is used at the end of the quiz to determine whether the student has correctly answered the question.

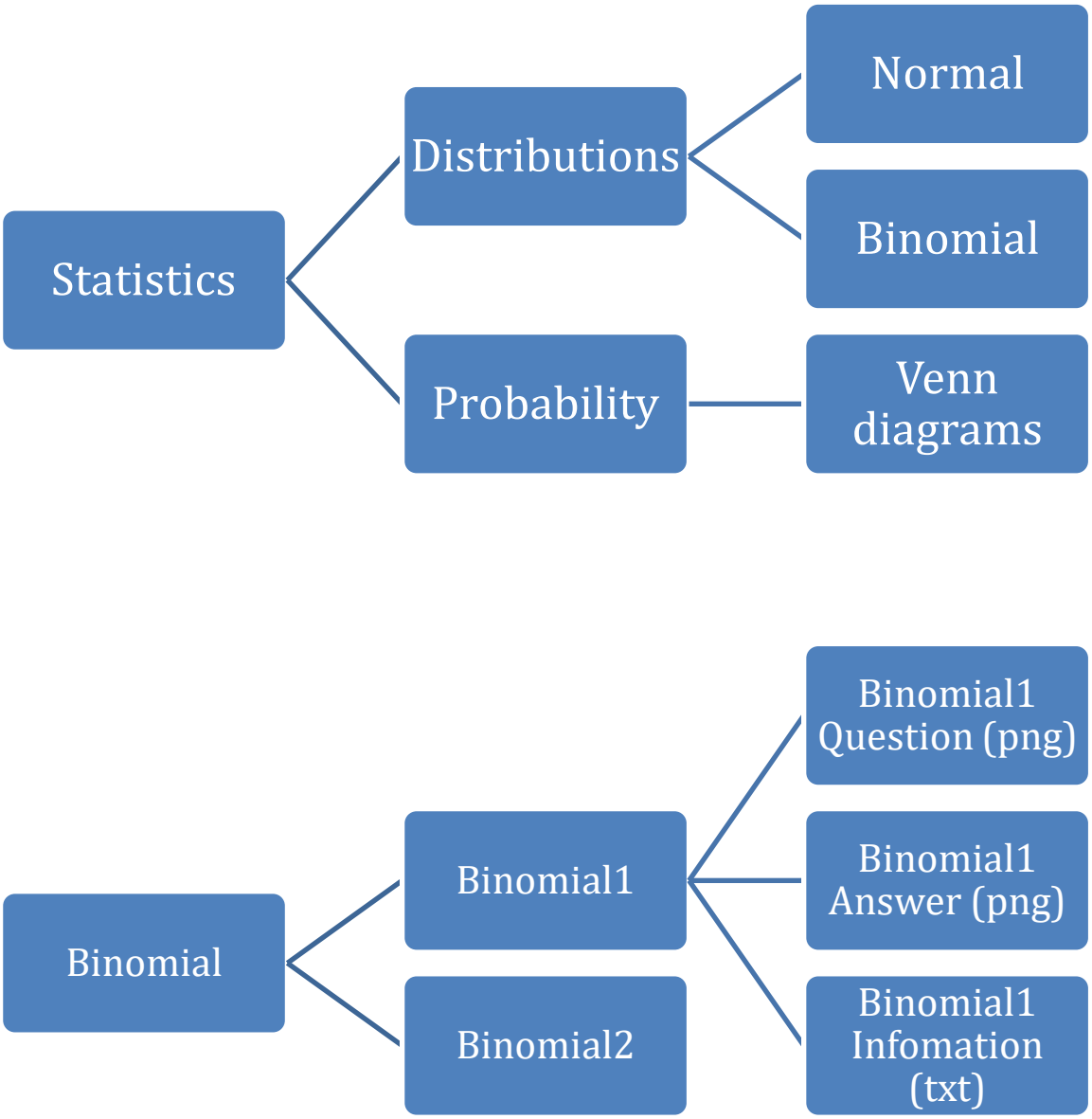
This information is used to populate the **Question** structure upon loading of the '*Randomised quiz'* and '*Select a subtopic***'** modules.

Folder hierarchy

For each question there will be three files required: the question (in the form of a PNG image), the answer (also in PNG form) and a question information text file (described above).

The three files needed for each question will be stored within a hierarchical file system. A folder named *Modules* will be used to hold the *Module* folders. Within each of these *Module* folders will be the *Topic* folders. Each of these *Topic* folders will contain the *Subtopic* folders. Within these *Subtopic* folders will be the *Question numbers* folders for each question and finally, within these *Question numbers* folders will be the three documents required for the question (described above).

The following hierarchy charts visually demonstrates how the folder structure for the *Statistics* module will be laid out. The first chart shows the possible paths that can be taken to access a specific subtopic within the *Statistics* folder. The second chart is a continuation from the first chart, and shows the hierarchy of a *Subtopic* folder, specifically the *Binomial* subtopic from within the *Statistics* module.



2.4 Pseudocode for main algorithms

Note 1: Some of the variable names used in the pseudocode below may differ to the variable names in the actual program.

Note 2: Any sentences coloured green and preceded by a hashtag are comments on the code, designed to help the user understand what the code is doing.

01. Pseudocode for creating a new class (Teacher user account)

The Teacher user type will enter details into the *Create A Class* form; a list box named *lstStudents* will contain the names of all the students who have signed up to the program. This list box will allow the teacher to individually select students to add to the new class they are creating. The Teacher will also have the option to search for a student by ID using direct access.
An input box named *txtClassName* will be used to take in the name of the new class.

Once these details have been entered and validated, they will be used to populate the **CreateNewClass** structure. The values stored in the structure will then be written to the *classes.txt* text file.

```
# lstStudents contains the list of all the students who have signed up to the
program
# lstNewClass contains the list of students which the teacher has added to the new
class

# This procedure copies a student name from lstStudents to lstNewClass, provided
the student is not already present in lstNewClass
PROCEDURE AddStudent
	StudentAlreadyInList <-- FALSE

	IF an item in lstStudents is selected THEN
		FOR each item in lstNewClass
			IF item == selected item in lstStudents THEN
				StudentAlreadyInList <-- TRUE
			ENDIF
		ENDFOR
		IF StudentAlreadyInList == FALSE THEN
			ADD selected item in lstStudents to lstNewClass
		ENDIF
	ENDIF
END PROCEDURE

# This procedure uses direct access to search for the student by ID
PROCEDURE Search
	StudentAlreadyInList <-- FALSE

	OPEN FILE "studentaccounts.txt"
	RETRIEVE student information at index SearchID
	CLOSE FILE

	StudentFullName <-- StudentForename + " " + StudentSurname

	# Determines if the student name is already in lstNewClass
	FOR n = 1 TO number of items in lstNewClass
		IF lstNewClass.Items(n) == StudentFullName
			StudentAlreadyInList <-- TRUE
		ENDIF
	ENDFOR
	IF StudentAlreadyInList == FALSE
		ADD StudentFullName TO lstNewClass
```

```
        ENDIF
END PROCEDURE

# This procedure populates the CreateNewClass Structure with values and then writes
those values to the classes.txt text file
PROCEDURE CreateClass

        # Transferring contents of lstNewClass to NewClassToAdd.Students()
        FOR n = 1 TO number of items in lstNewClass
                NewClassToAdd.Students(n) <-- the item stored at index n in
                lstNewClass
        ENDFOR

        FOR n = 1 to number of items in NewClassToAdd.Students()
                strStudentNames <-- NewClassToAdd.Students(n)
        ENDFOR

        New.ClassSize = number of items in lstNewClass

        OPEN TEXT FILE

        INSERT INTO TEXTFILE(ClassName, ClassSize, StudentNames)
        VALUES(NewClassToAdd.ClassName, NewClassToAdd.ClassSize, strStudentNames)

END PROCEDURE
```

02. Pseudocode for displaying information about a specific student (Teacher user account)

The following pseudocode demonstrates how the Teacher user type will be able to access statistics gathered by the program on the students who have signed up. A list box containing the names of all those students will be populated from a the data stored in a text file, and the teacher can select a student to view data on. Once a student has been selected, a list box and a bar chart will both be populated with that student's data, showing the student's performance on one of the options selected by the teacher: module, topic and sub-topic.

```
# Using the AddAccountDetails structure to read student names from a text file into a
list box
PROCEDURE Main
    OPEN FILE "studentaccounts.txt"
    FOR EACH LINE IN studentaccounts.txt
            READ DATA INTO StudentDetails
            ADD StudentDetails.Forename + StudentDetails.Surname INTO listbox
    ENDFOR
END PROCEDURE

# When the select buttons is clicked, the selected student's details are read from the
student's Scores text file
PROCEDURE btnSelect_click
    OPEN text file "studentscores"
    FOR EACH COLUMN IN studentscores
            Read column into relevant array
    ENDFOR
```

```
    DisplayScores()
ENDPROCEDURE

# Fuction to calculate the overall score of a student
FUNCTION CalculateMean(Scores() BY VALUE)
    FOR EACH Item IN Scores()
            Mean --> Mean + Item
    ENDFOR
    Mean = Mean / Scores.NumberOfItems
    RETURN Mean
END PROCEDURE

PROCEDURE DisplayScores()
# Taking parameters from the user, that will be used to display relevant information in
graphical and text form

    OUTPUT("Enter a module")
    Module = INPUT

    OUTPUT("Enter a topic")
    Topic --> INPUT

    OUTPUT("Enter a sub-topic")
    SubTopic = INPUT

    # Displaying student's scoring information in a list box
    # lstStudentData is the list box that will hold student data

    ADD ("Total score for module is: " + CalculateMean(ModuleScores)) INTO
lstStudentData
    ADD ("Total score for topic is: " +  CalculateMean(TopicScores)) INTO
lstStudentData
    ADD ("Total score for sub-topic is: " + CalculateMean(SubTopicScores)) INTO
lstStudentData

    # Displaying the student's scoring information graphically
    # grStudentData is the name of the bar chart that will be used to display student's
scores

    OUPTPUT("Select a graph to view: Module, Topic or Sub-Topic")
    GraphView = INPUT
    IF GraphView == Module
            DRAW barchart ON grStudentData USING ModuleScores()
    ELSE IF GraphView == Topic
            DRAW barchart ON grStudentData USING TopicScores()
    ELSE IF GraphView == Sub-Topic
            DRAW barchart ON grStudentData USING SubTopicScores()
    END IF
END PROCEDURE
```

03. Pseudocode for creating an overview of class progress (Teacher user account)

The following pseudocode demonstrates how the Teacher user type will be able to access data stored about the about the classes they have created within the *Create New Class* module (*see 2.1: Modular structure of the system)*. The information stored about each class,

including the names of the students within that class, will be read from each of the student's *Scores* and *Attempts* text files. Upon accessing the data of each individual student in the class, a class average for each statistic will be calculated, and then displayed both graphically and in a text format.

This pseudocode requires access to the "classes.txt" text file. It will read data from this text file into a list box.

Note: The first part of the *Classes Overview* module will use the same general method that has been outlined in *02. Pseudocode for displaying information about a specific student* to read student file, and as such has not been shown in the below pseudocode.

```
PROCEDURE Main
        AddClasses()
        # This is one of the subroutines demonstrated in 02
        DisplayStudentDetails()
END PROCEDURE

# This procedure copies adds the name of the class and the names of the students in
that class to a list box
PROCEDURE AddClasses()
        OPEN TEXTFILE "Classes.txt"
        FOR EACH LINE IN Classes.txt
                READ DATA INTO ClassToRead
                ADD (ClassToRead.ClassName) INTO lstStudents
        ENDFOR
END PROCEDURE

# This procedure will calcalate a mean score for a class in a particular area
# It will be called from within the DisplayStudentDetails() procedure
PROCEDURE CalculateClassMean(Students() BY VALUE, DataToDisplay BY VALUE)
        # As mentioned in the description, the code to read in student data is not
shown

        DECLARE Mean AS INTEGER

        FOR EACH name IN Students()
        # This subroutine returns the data stored on the student for either module,
topic or sub-topic
                Mean =  Mean + ReadStudentDetails(name BY VALUE, DataToDisplay BY
VALUE)
        ENDFOR
        Mean = Mean / Students.NumberOfItems
        RETURN Mean
END PROCEDURE
```

04. Pseudocode for navigating a math quiz (Student user account)

The following pseudocode demonstrates how the Student user type will be able to submit answers to images, and how they will be able to move back and forth between different questions.

The images for each question will be stored using the file structure that was described in *2.3 File Handling and Data* and will be accessible to the program from the resource folder in the visual studio IDE. Each time the student moves between questions the program will retrieve the relevant image from the resource folder and display the image in the picture box.

```
# There will be 5 questions in the randomised quiz, so 10 instances of Question will
be declared in the program

# Randomly selecting 5 different questions using a subroutine containing lists of
strings with the question names in them
FOR i = 1 to 5
       SelectRandomQuestion()
END FOR

# Each instance of Question will be populated with data stored in a text file
specific to that question
# Reading contents of unique text file into the relevant instance of Question.

PictureBox image = GET IMAGE(Q1.jpg)

# Marks the student's answer as correct or incorrect
# Note: CurrentQuestion is the instance of Question that the user is currently
answering
PROCEDURE SubmitAnswer
INPUT UserAnswer
CurrentQuestion.IsAnswered <-- TRUE
IF UserAnswer == CurrentQuestion.IsAnswered
       CurrentQuestion.IsCorrect <-- TRUE
END IF
END PROCEDURE

# Changes the current instance of Question, and changes the image in the PictureBox
to match that question
# Note: next question will be set to true if the user wants to move the next
question, or false if the user wants to move the previous question
PROCEDURE ChangeQuestion(NextQuestion AS BOOLEAN)
    # Cycling back round the question numbers once the end of the questions has been
reacher
    IF NextQuestion == TRUE
        NextQuestionNumber = CurrentQuestion.QuestionNumber + 1
        IF NextQuestionNumber > 5 THEN
        NextQuestionNumber = 1
        END IF
    ELSE
        NextQuestionNumber = CurrentQuestion.QuestionNumber - 1
        IF NextQuestion < 1 THEN
        NextQuestionNumber = 5
                END IF
        LOAD DYNAMIC BUTTONS
        # Displays the next question
        PictureBox Image = (Question + NextQuestionNumber)
ENDIF

# iterates through all the instances of Question to identify the next question
FOR all instances of Question
       IF CurrentInstanceOfQuesiton.QuestionNumber = NextQuestion THEN
```

```
        CurrentQuestion = CurrentInstanceOfQuesiton
        END IF
        END FOR

        # updates the image in the PictureBox to match the current question
        PictureBox.image = GET IMAGE(CurrentQuestion.QuestionNumber.jpg)
END PROCEDURE
```

05. Pseudocode for saving a student's quiz results

The following pseudocode demonstrates how the results for the student user type quiz is saved. Using the *Results* and *Attempts* text files described in *2.3 File handling and data processing*, the student's score for the quiz is saved and their overall score whilst using the program is updated to be used in the **Teacher** section of the program when displaying student results.

```
PROCEDURE ReadData(Sucsessful BY REFERENCE)
        TRY
                i <-- 1
                OPEN TEXTFILE filename
                FOR EACH LINE IN filename
                # SaveResults is a data structure that has been previously declared
                        READ DATA INTO SaveResults
                ENDFOR
                CLOSE TEXTFILE

                # Incrementing the student's recorded number of attempts
                Attempts <-- SaveResults.Attempts
                Attempts <-- Score + 1
                SaveResults.Attempts <-- Attempts
        END TRY
END PROCEDURE

PROCEDURE WriteData(filename BY VALUE)
        # The following code overwrites contents of results file with values stored
        in an instance of the TestResults structure
        TRY
                OPEN TEXTFILE filename
                        WITH SaveResults
                                WRITE EACH VARIABLE TO filename
                        END WITH
                CLOSE TEXTFILE

                # Changing filename so data can now be written to the Attempts text
                file
                INSERT "Attempts" INTO filename

                # Writing percetage score to student attempts text file
                OPEN TEXTFILE filename
                        WRITE avgScore TO filename
                CLOSE TEXTFILE
        END TRY
END PROCEDURE
```

2.5 Validation

Note: The following bullet points contain examples of some of the validation that will be incorporated into the program. All examples are written in pseudocode, and may vary slightly to the visual basic code, but the overall structure and purpose of the code will remain the same.

01. Try Catch validation for reading student names from a text file

The following Pseudocode is using a Try Catch validation method to prevent the program from crashing if it encounters an error, and instead outputting the appropriate error message to the user. In this case the potential error may occur when reading student details from a text file containing instances of the **AccountDetails** structure, selecting only the **Forename** and **Surname** variables and then writing those names into a list box.

```
TRY
        OPEN FILE "studentaccounts.txt"
        RETRIEVE student information at index SearchID
        CLOSE FILE

        StudentFullName <-- StudentForename + " " + StudentSurname

        FOR n = 1 TO number of items in lstNewClass
                IF lstNewClass.Items(n) == StudentFullName
                        StudentAlreadyInList <-- TRUE
                ENDIF
        ENDFOR
        IF StudentAlreadyInList == FALSE
                ADD StudentFullName TO lstNewClass
        ENDIF
CATCH ex AS EXCEPTION
        OUTPUT --> "An error occurred whilst attempting to retrieve student data."
        Please try again."
END TRY
```

02. IF statement and Try Catch validation for performing a format check on the class name input

The following Pseudocode is intended to ensure that all inputs for a new class name follow a set format; the class name is exactly 5 character in length, and the first 2 characters of the class name are numbers (to represent the year group).

```
IF LENGTH OF NewClassToAdd.ClassName == 5 THEN
        TRY
                CONVERT first 2 characters of NewClassToAdd.ClassName TO INTEGER
        CATCH ex AS EXCEPTION
                OUTPUT --> "The first 2 characters of the class name must begin with
                numbers."
                ValidDataEntry <-- FALSE
```

```
        END TRY
ENDIF
```

03. IF statement and Try Catch validation for logging the user into the program

The purpose of the IF statement in the following Pseudocode is to ensure that the user can only attempt a login if they have both the student username and student password, or both the teacher username and teacher password text boxes filled out by the user without any other text boxes having any inputs.
The Try Catch statement is intended to stop the programming from crashing in the event of it encountering an error, which in this case would be when it attempts to read student data from a text file.

```
# All variables beginning with txt represent input boxes, and are appropriately
named to reflect the inputs they accept

# IF statement Validation for determining if both textboxes for the student
usertype or both textboxes for the teacher usertype are correctly filled out
ValidTextboxesFilled <-- TRUE

IF (txtStudentUsername IS EMPTY AND txtStudentPassword IS EMPTY) OR
(txtStudentUsername IS EMPTY AND txtStudentPassword IS NOT EMPTY) OR
(txtStudentUsername IS NOT EMPTY AND txtStudentPassword IS EMPTY) THEN
        IF (txtTeacherUsername IS EMPTY AND txtTeacherPassword IS EMPTY) OR
(txtTeacherUsername IS EMPTY AND txtTeacherPassword IS NOT EMPTY) OR
(txtTeacherUsername IS NOT AND txtTeacherPassword IS EMPTY) THEN
                ValidTextboxesFilled <-- FALSE
        ENDIF
ELSEIF (txtStudentUsername IS NOT EMPTY AND txtTeacherUsername IS NOT EMPTY) OR
(txtStudentUsername IS NOT EMPTY AND txtTeacherPassword IS NOT EMPTY) OR
(txtStudentPassword IS NOT EMPTY AND txtTeacherUsername IS NOT EMPTY) OR
(txtStudentPassword IS NOT EMPTY AND txtTeacherPassword IS NOT EMPTY) THEN
        ValidTextboxesFilled <-- FALSE
ENDIF

IF ValidTextboxesFilled == FALSE THEN
        OUTPUT --> "Please enter both a username and password for both student or
teacher usernames."
ENDIF

# Try Catch validation for reading login data from a text file
OPEN FILE "students.txt"
RETRIEVE contents of file
IF username == username in file THEN
        IF password == password in file THEN
                OUTPUT --> "Successful login"
                Log the user in
        ELSE
                OUTPUT --> "Invalid password"
        ENDIF
ELSE
        OUTPUT --> "Could not find an account with that username"
ENDIF
CLOSE FILE
```

04. Validation for ensuring that the data stored in the *Scores* text file is valid

In the following pseudocode, validation to ensure that the average score for a student's results in a quiz is less than or equal to 100%.

```
# Calculating average of a variable in the TestResults structure
FUNCTION CalculateAverage(strToAverage BY REFERENCE)

        # Ensuring average score is <= 100%
        TRY
                # If the first half of the string from which an average is calculted
                from is less than the second half
                IF (CONVERT TO INTEGER(strToAverage.Substring(1, 5))) <= (CONVERT TO
                INTEGER(strToAverage.Substring(7, 5))) AND strToAverage.Length() = 12
                THEN
                        # then calculate the average
                        avg <-- (CInt(strToAverage.Substring(1, 5)) /
                        CInt(strToAverage.Substring(7, 5))) * 100
                ELSE
                        OUTPUT --> "Unable to calculate average. Data file needs
                        erasing."
                        avg = 0
        END TRY
END FUNCTION
```

2.6 User Interface

Note: The following sketches are hand-drawn during the initial planning of the program. They are intended to represent the general layout of the UIs for the individual modules in the system so as to consolidate with the end user what the general structure of the program will be like before the design of those UIs begin. As a result, these sketches may not be a fully accurate representation of the final user interface.

For example, the Student Homepage module (depicted in Figure 3) shows 2 buttons that, when clicked, will take the user to the two 'Student' Modules (See 2.1: Modular structure of the system) whereas in the actual program a tab interface will be used to allow the user to navigate back and forth between modules, instead of moving directly to a module with no way to get back to the homepage.

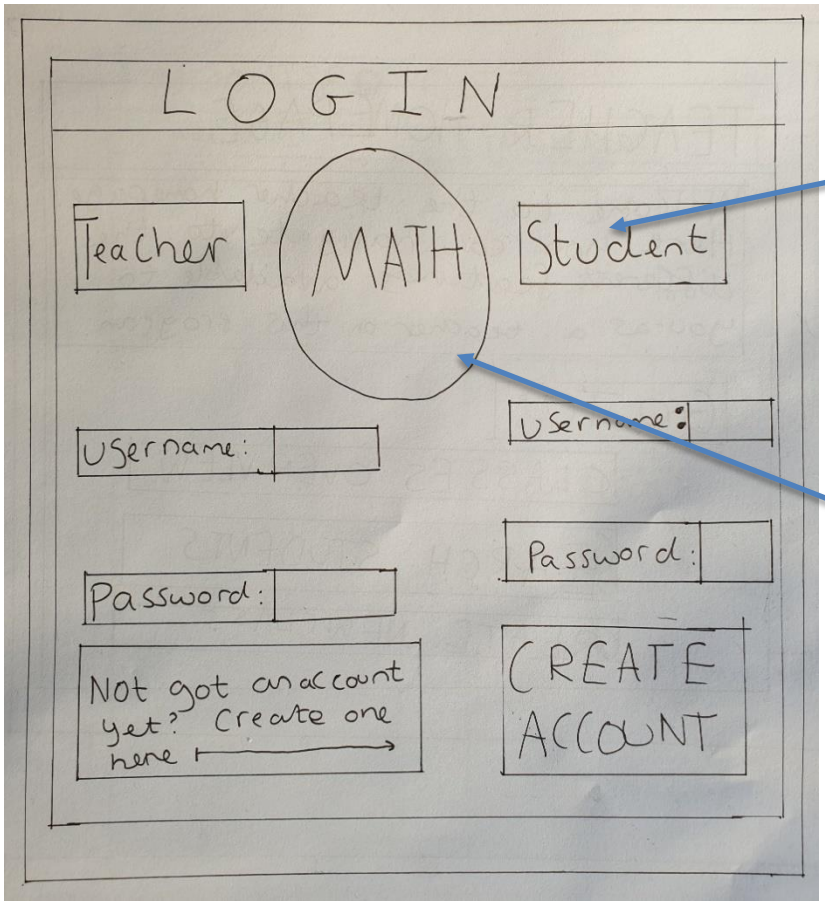


Figure 1: Sketch of the login page

The 'Student' and 'Teacher' labels will indicate to the user which column of login details to fill out.

The program logo will be printed on the login system to make the form more aesthetically pleasing.

Bold text and a smart, professional font will be consistently used on all elements of the form.

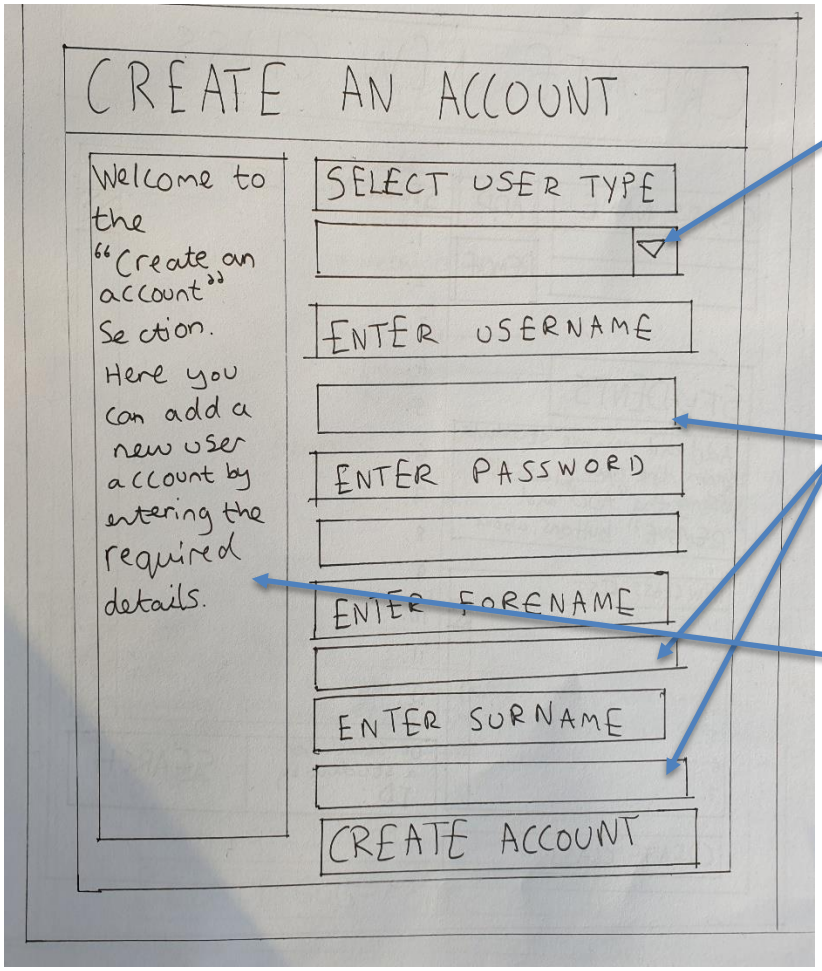


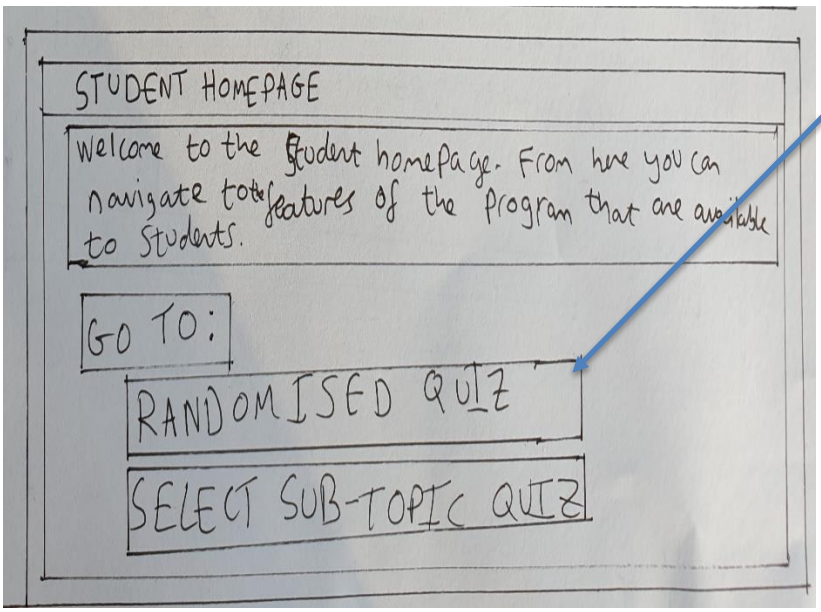
Figure 2: Sketch of the 'Create Account' page

A drop-down menu will be used to make the process of creating a new account quicker and simpler.

The fields in which the user enters new account details.

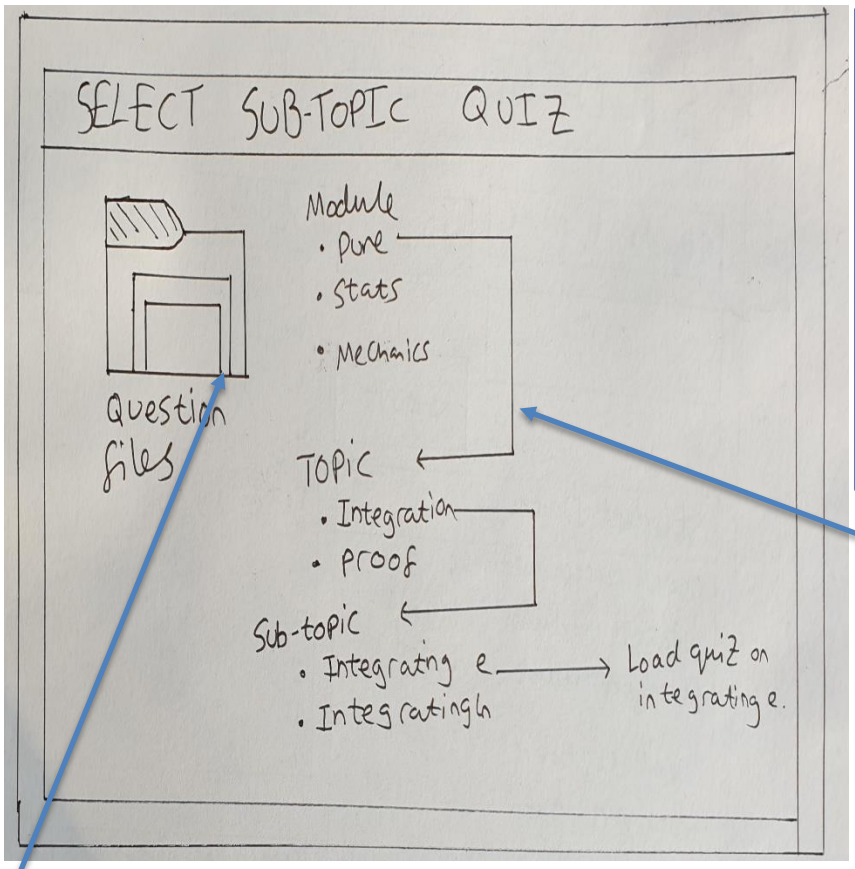
A welcome message or piece of advice will be consistently displayed across all forms in order to make the program simple and accessible to new users on the program.

Figure 3: Sketch of the 'Student Homepage' page



Buttons that, when clicked, will take the student to the 'Student' modules in the program (see *2.1: Modular Structure of the System*).

Figure 4: Sketch of the 'Select a Sub-Topic Quiz' page

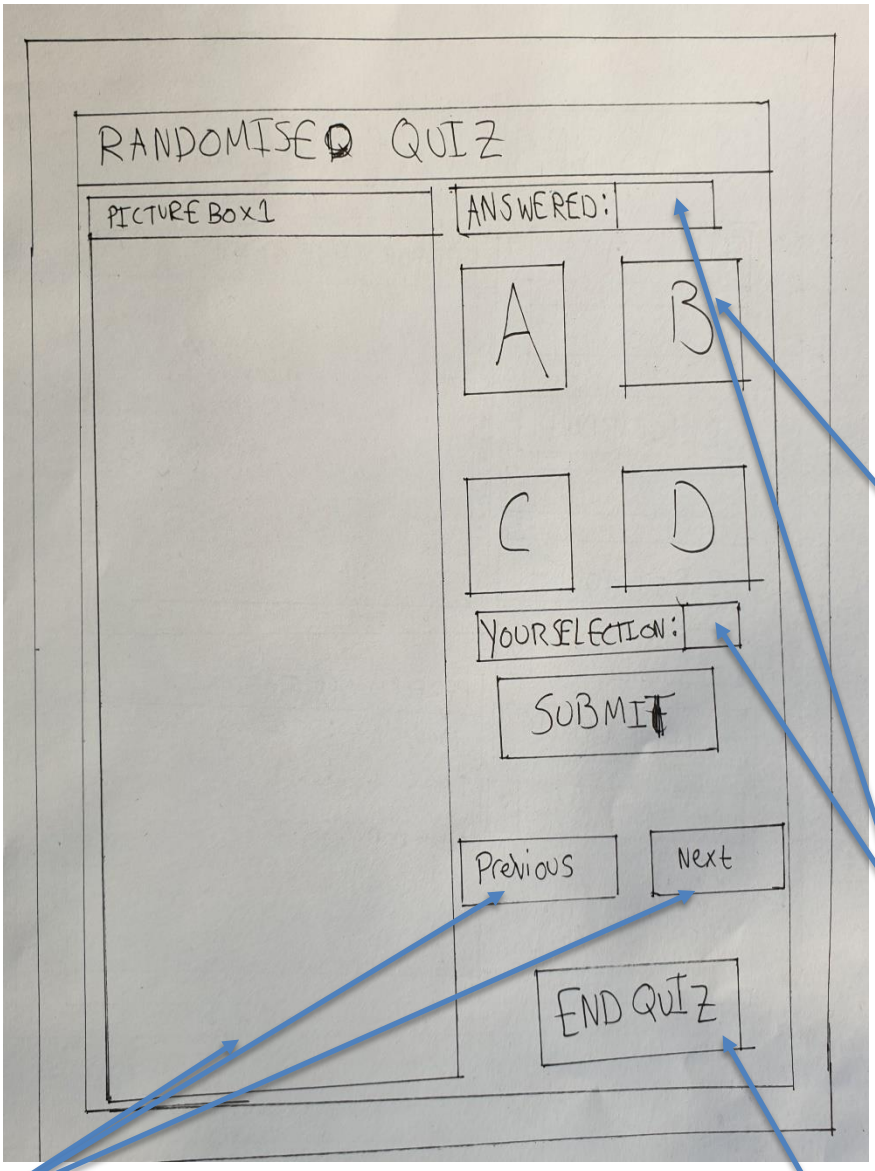


Note: This sketch visually depicts the path that the student will take to load a specific sub-topic quiz (this particular example show the user selecting the path 'Pure → Integration → Integrating e' leads to a quiz on 'Integrating e'. It does not show the path to all the possible sub-topic quizzes.

Arrows will be used to visually indicate to the user the file paths they have chosen.

An image folder symbol like the one used in the Windows operating system file management system will be displayed to visually indicate to the user that they can search a file directory for specific files.

Figure 5: Sketch of the 'Randomised Quiz' page



Note: The UI for the randomised quiz will be the same as the UI for the sub-topic quiz. The difference between the two quizzes is that the randomised quiz will display randomly selected questions, and the sub-topic quiz will display questions that have been specifically selected by the user.

Buttons depicting letters will be used to allow users to answer multiple choice questions.

Text boxes will also be displayed as an alternative to buttons, to allow users to answer questions that are not multiple choice.

'Answered' and 'Your Selection' labels will be used to indicate to the user whether they have answered the question, and what the letter they have selected is respectively.

A picture box will be used to display the question images in. The images displayed in this box will change to display the relevant question when the 'Previous' and 'Next' buttons are clicked.

An 'End Quiz' button will always be present, giving the user the option to end the quiz without answering all the questions, should they choose too.

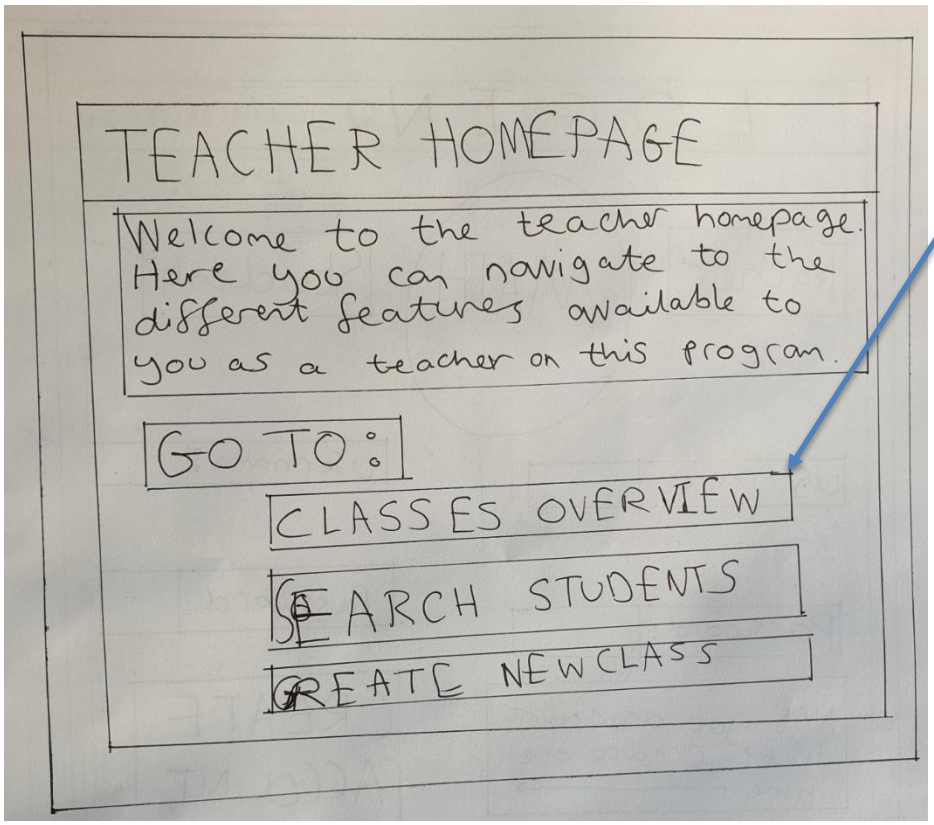
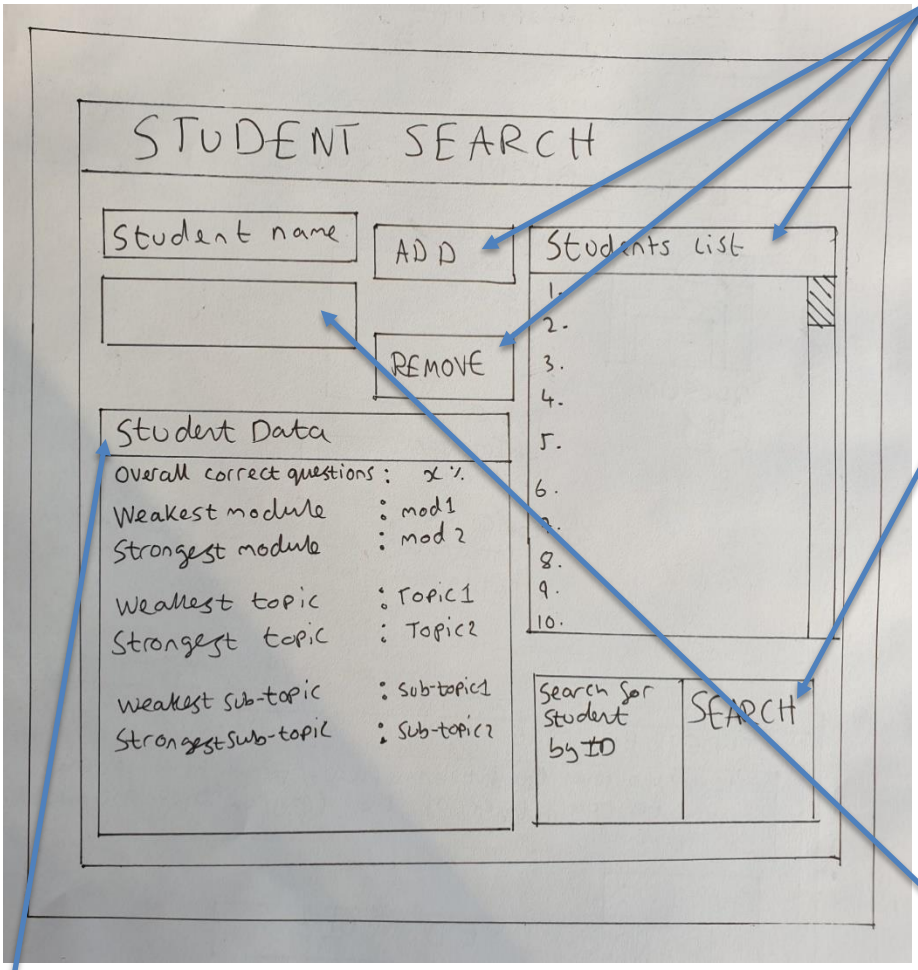


Figure 6: Sketch of the 'Teacher Homepage' page

Similar to the 'Student Homepage' page depicted in *Figure 3*, this page contains buttons that will take the teacher to the 'Teacher' modules (see *2.1: Modular Structure of the System*)

Figure 7: Sketch of the 'Student Search' page



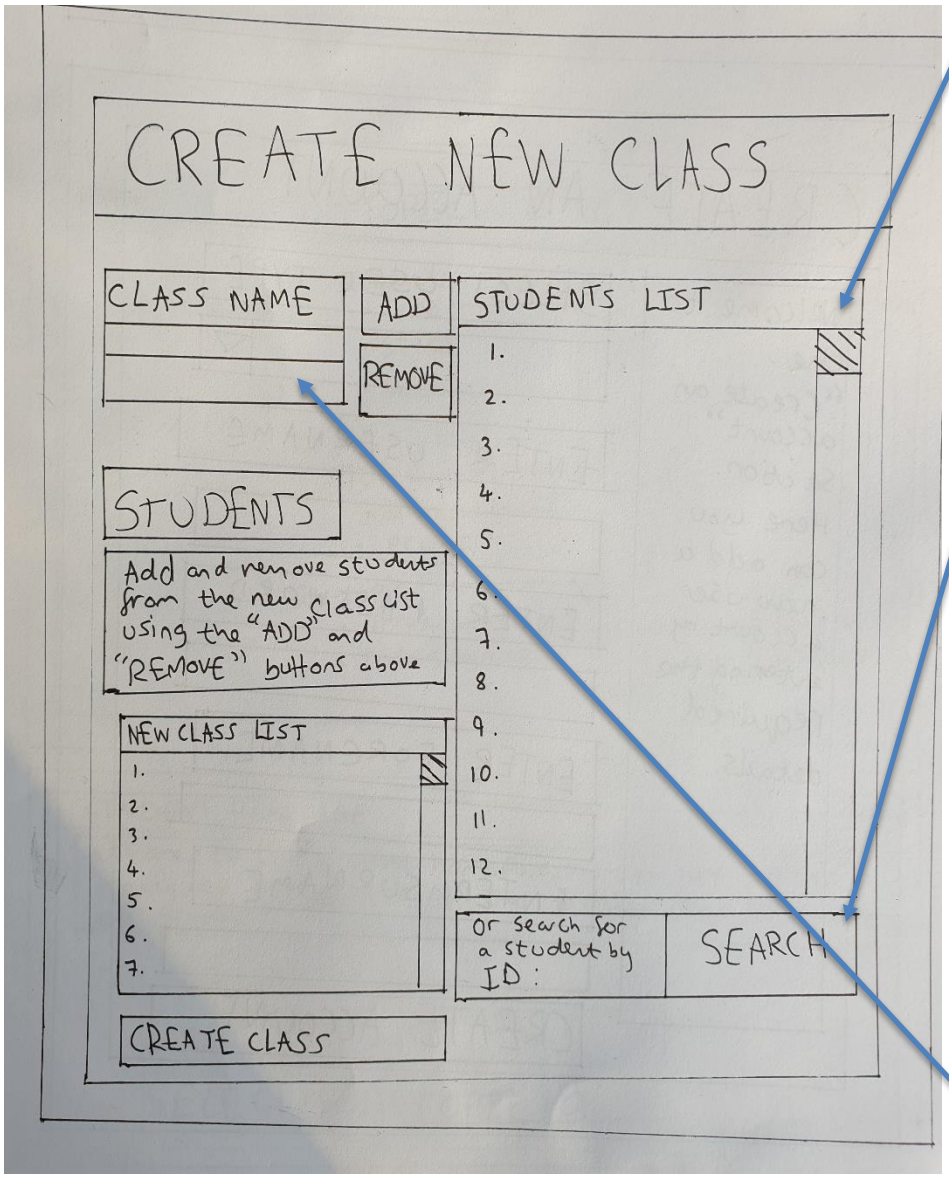
A list box will be used that will contain the names of all the students who have signed up to the program. The teacher will be able to select and delete a name using the 'Add' and 'Remove' buttons respectively and then have relevant data about that student displayed to them.

A search box will also be available to the user. When the 'Search' button is clicked, a textbox will appear that will allow the user to input the ID of a particular student. If an ID equal to the ID input by the user is found, the program will then display information about that student in the 'Student Data' table.

The selected student's name will appear in this textbox to remind the teacher which student's data they are viewing.

Once a student has been selected, a table containing data about that student's performance in the program will be presented to the teacher. This data will include percentages for that student's progress in different modules, and their overall progress.

Figure 8: Sketch of the 'Create New Class' page



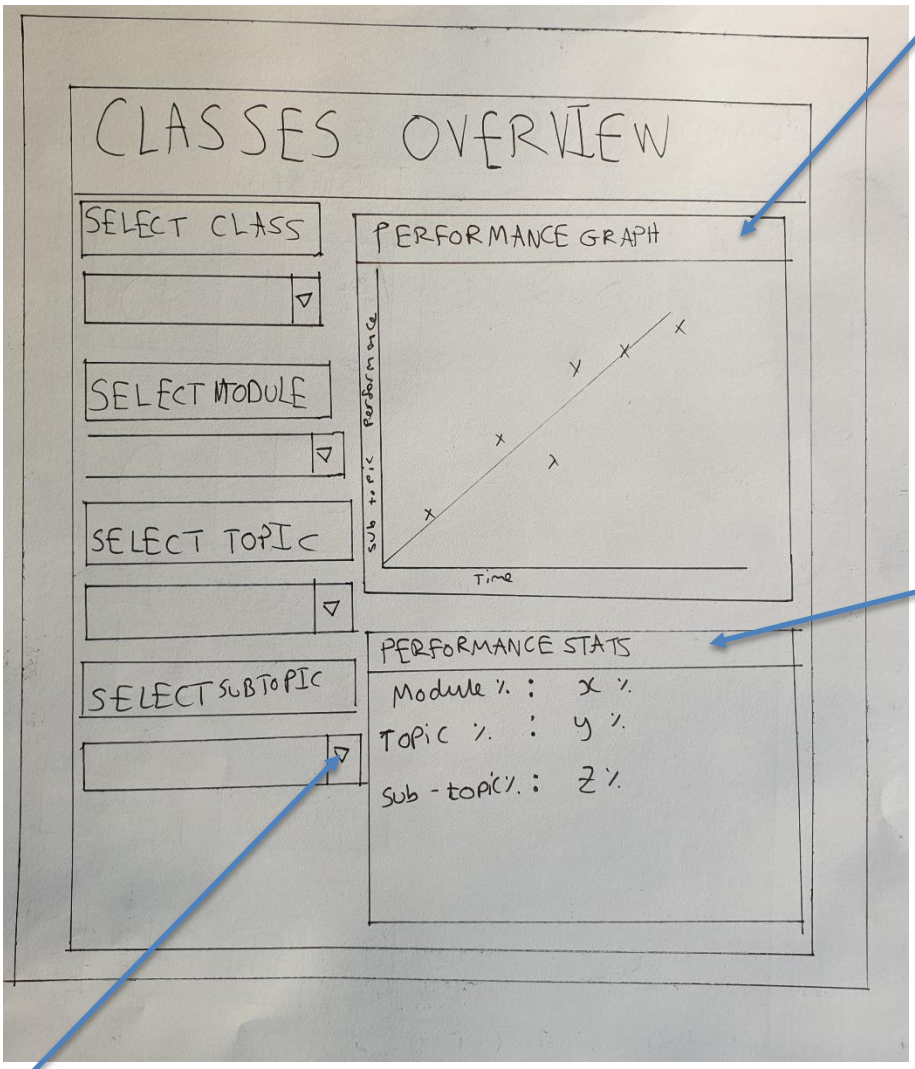
Similar to **Figure 7**, a list box will be used that will contain the names of all the student's who have signed up to the program. The teacher will be able to copy names across from the 'Students' list to the 'New Class' list by selecting a student from the 'New Class' list and clicking the 'Add' buttons.

Again, similar to **Figure 7**, A search box will be available to the user. When the 'Search' button is clicked, a textbox will appear that will allow the user to input the unique of a particular student. If a student with an ID equal to the ID input by the user is found, the name of the student will be copied from the 'Students' list into the 'New Class' list.

The 'Class Name' textbox will take an input for the name of the new class that the teacher wishes to create.

The 'New Class' list will display the names of the students that the teacher has selected to add to the new class. When the 'Create Class' button is clicked, these names will be stored by the program along with the value stored in the 'Class Name' textbox.

Figure 9: Sketch of the 'Classes Overview' page



A graph will be used to visually display the classes performance in a specific area of the course to the teacher. This graph will be a line graph, and will contain a line of best fit to help the teacher gain a better idea of the general class performance.

A table containing statistics about the selected area of the course will also be provided, allowing the teacher to view the specified data in an alternative format.

A series of drop down menus will be used to allow the teacher to refine their searches. Upon specifying which areas of the course they want to search for, they will then be presented with data stored about pupils' performances from specific areas of the course.

IMPLEMENTATION

3.1 Complexity List

Complexity Type	Page Numbers
Using local variables	87
Selection (Case, If)	• If…Endif statement: 74 • Select Case statement: 99
Iteration (For…Next, While…End While, Repeat…Until)	• For…Next statement: 77 • While…End While statement: 76 • Do…Until: 100
Functions	87
Parameter passing (ByVal and ByRef)	• ByVal: 82 • ByRef: 90
Good error handling (Try…Catch, Range check, Length check, Format check)	• Try…Catch: 76 • Range check: 78 • Presence check: 111 • Length check: 112 • Format check: 112
Arrays	104
User defined records (structures)	81
File handling routines (serial, direct)	• Serial access: 116 • Direct access: 111

Comp 4

Search routines	Linear search: 76
Sort routines (bubble, quick, merge)	Quicksort: 112
Encryption	Caesar shift: 79
Recursion	122
Graphics and drawing routines	115
Different access levels	76
Dynamic creation of objects	82
Advanced controls (MDI container, use of panels)	• MDI containers: 80 • Panels: 123
Bespoke processing/ calculation routines: - A subroutine that calculates the average percentage score for each topic, used for filling out a scores table. - A subroutine that calculates the average score against number of attempts, used for plotting a line graph. - Calculating y-axis position of student scores to be plotted.	• Average percentage score for each topic calculator: 121 • Average score against attempts calculator: 121 • Calculating y-axis position of a point to be plotted: 116
Other complexity: - Use of lists - Use of classes - Regex expression - LINQ and lambda expressions	• List of strings: 107 • List of a data class: 119 • Class: 119 • Regex expression: 122 • LINQ and lambda expression: 115
Total lines of code:	**3017**

3.2 Code Listing

Code for ‘Login’ module

```
Public Class frmLoginPage
    Private Sub frmLogin_Load(sender As Object, e As EventArgs) Handles MyBase.Load
        Me.WindowState = FormWindowState.Maximized
    End Sub

    Public Structure AccountDetails
        <VBFixedString(20)> Public Username As String
        <VBFixedString(20)> Public Password As String
        <VBFixedString(20)> Public Forename As String
        <VBFixedString(20)> Public Surname As String
    End Structure

    Structure LoginValidation
        Public InputDataValid As Boolean
        Public CorrectFieldsFilled As Boolean
        Public StudentUsernameEmpty As Boolean
        Public StudentPasswordEmpty As Boolean
        Public TeacherUsernameEmpty As Boolean
        Public TeacherPasswordEmpty As Boolean
        Public Student As Boolean
        Public Teacher As Boolean
        Public LoginDetailsFound As Boolean
        Public filename As String
        Public username As String
        Public password As String
        Public DecryptedCiphertext As String
    End Structure

    Public UserLoginStructure As AccountDetails
    Private LoginAttempt As LoginValidation

    Private Sub BtnCreateAccount_Click(sender As Object, e As EventArgs) Handles
btnCreateAccount.Click
        ClearFields()
        Me.Hide()
        frmCreateAccount.Show()
    End Sub

    Private Sub btnLogin_Click(sender As Object, e As EventArgs) Handles
btnLogin.Click, MyBase.Click
        LoginAttempt.InputDataValid = False
        LoginAttempt.CorrectFieldsFilled = False

        ' Presence check: determine which text boxes are empty
        If (String.IsNullOrEmpty(txtStudentUsername.Text)) Then
            LoginAttempt.StudentUsernameEmpty = True
        End If
        If (String.IsNullOrEmpty(txtStudentPassword.Text)) Then
            LoginAttempt.StudentPasswordEmpty = True
        End If
        If (String.IsNullOrEmpty(txtTeacherUsername.Text)) Then
            LoginAttempt.TeacherUsernameEmpty = True
        End If
        If (String.IsNullOrEmpty(txtTeacherPassword.Text)) Then
            LoginAttempt.TeacherPasswordEmpty = True
```

```
        End If
        ' Determines if both of the text boxes in one field, but neither of the
textboxes in the other field, have been filled
        If LoginAttempt.StudentUsernameEmpty And LoginAttempt.StudentPasswordEmpty And
LoginAttempt.TeacherUsernameEmpty And LoginAttempt.TeacherPasswordEmpty Then
            ' Both catgories are fully empty
            MessageBox.Show("Please input values into either the teacher or student
category to login", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error)
        ElseIf (Not LoginAttempt.StudentUsernameEmpty Or Not
LoginAttempt.StudentPasswordEmpty) And (Not LoginAttempt.TeacherUsernameEmpty Or Not
LoginAttempt.TeacherPasswordEmpty) Then
            ' Both teacher and student categories have data input into one of the
relevant texboxes
            MessageBox.Show("Please do not attempt to fill in both teacher and student
login", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error)
        ElseIf (LoginAttempt.StudentUsernameEmpty And Not
LoginAttempt.StudentPasswordEmpty) And (LoginAttempt.TeacherUsernameEmpty And
LoginAttempt.TeacherPasswordEmpty) Then
            ' The teacher category is empty but the student category has a password
but no username
            MessageBox.Show("Please enter a value for the the student username",
"Error", MessageBoxButtons.OK, MessageBoxIcon.Error)
        ElseIf (Not LoginAttempt.StudentUsernameEmpty And
LoginAttempt.StudentPasswordEmpty) And (LoginAttempt.TeacherUsernameEmpty And
LoginAttempt.TeacherPasswordEmpty) Then
            ' The teacher category is empty but the student category has a username
but no password
            MessageBox.Show("Please enter a value for the the student password",
"Error", MessageBoxButtons.OK, MessageBoxIcon.Error)
        ElseIf (LoginAttempt.StudentUsernameEmpty And
LoginAttempt.StudentPasswordEmpty) And (LoginAttempt.TeacherUsernameEmpty And Not
LoginAttempt.TeacherPasswordEmpty) Then
            ' The student catgory is empty but the teacher category has a password but
not username
            MessageBox.Show("Please enter a value for the teacher username", "Error",
MessageBoxButtons.OK, MessageBoxIcon.Error)
        ElseIf (LoginAttempt.StudentUsernameEmpty And
LoginAttempt.StudentPasswordEmpty) And (Not LoginAttempt.TeacherUsernameEmpty And
LoginAttempt.TeacherPasswordEmpty) Then
            ' The student catgory is empty but the teacher category has a username but
no password
            MessageBox.Show("Please enter a value for the the teacher password",
"Error", MessageBoxButtons.OK, MessageBoxIcon.Error)
        ElseIf (Not (LoginAttempt.StudentUsernameEmpty And
LoginAttempt.StudentPasswordEmpty)) And (LoginAttempt.TeacherUsernameEmpty And
LoginAttempt.TeacherPasswordEmpty) Then
            ' Both student fields are full, and neither of the teacher fields
            LoginAttempt.filename = "studentaccounts.txt"
            LoginAttempt.CorrectFieldsFilled = True
            LoginAttempt.Student = True
        ElseIf (LoginAttempt.StudentUsernameEmpty And
LoginAttempt.StudentPasswordEmpty) And (Not (LoginAttempt.TeacherUsernameEmpty And
LoginAttempt.TeacherPasswordEmpty)) Then
            ' Both teacher fields are full, and neither of the student fields
            LoginAttempt.filename = "teacheraccounts.txt"
            LoginAttempt.CorrectFieldsFilled = True
            LoginAttempt.Teacher = True
        Else : MessageBox.Show("Logic error", "Error", MessageBoxButtons.OK,
MessageBoxIcon.Error)
        End If

        If LoginAttempt.CorrectFieldsFilled Then
```

```
            ' Take username & password input, assign them to respective varibles
            If LoginAttempt.Student Then
                LoginAttempt.username = txtStudentUsername.Text
                LoginAttempt.password = txtStudentPassword.Text
            ElseIf LoginAttempt.Teacher Then
                LoginAttempt.username = txtTeacherUsername.Text
                LoginAttempt.password = txtTeacherPassword.Text
            Else

            End If

            ' Read contents of file to check if input username & password are present
            Try
                Dim i As Integer = 1
                FileOpen(1, LoginAttempt.filename, OpenMode.Random, , ,
Len(UserLoginStructure))
                While Not EOF(1)
                    FileGet(1, UserLoginStructure, i)
                    LoginAttempt.DecryptedCiphertext =
UserLoginStructure.Username.Trim
                    ' Decrypt username & password to compare with input
                    ASCIIDecryption(LoginAttempt.DecryptedCiphertext)
                    If LoginAttempt.username = LoginAttempt.DecryptedCiphertext Then
                        LoginAttempt.DecryptedCiphertext =
UserLoginStructure.Password.Trim
                        ASCIIDecryption(LoginAttempt.DecryptedCiphertext)
                        ' Notify user of sucsessful login
                        If LoginAttempt.password = LoginAttempt.DecryptedCiphertext
Then
                            MessageBox.Show("Successful login", "Notice",
MessageBoxButtons.OK, MessageBoxIcon.Information)
                            LoginAttempt.LoginDetailsFound = True
                        End If
                    End If
                    i += 1
                End While
                ' Notify user that no match was found
                If Not LoginAttempt.LoginDetailsFound Then
                    MessageBox.Show("Login details not found", "Error",
MessageBoxButtons.OK, MessageBoxIcon.Error)
                End If
                FileClose(1)
            Catch ex As Exception
                MessageBox.Show("An error occured whilst attempting to log you in.
Please try again.", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error)
            End Try
        End If

        ClearFields()

        ' Display relevant homepage if login details correct
        If LoginAttempt.LoginDetailsFound Then
            If LoginAttempt.Teacher Then
                frmTeacherHomepage.Show()
                Me.Hide()
            Else
                frmStudentHomepage.Show()
                Me.Hide()
            End If
        End If

        ResetLoginStructureValues(LoginAttempt)
```

```
    End Sub

    Public Sub ASCIIDecryption(ByRef CiphertextToDecrypt As String)
        ' Take in the encoded ASCII value and convert back to plaintext
        Dim EncryptionFactor As Integer = 3
        Dim plaintext As String = ""
        For Each letter As String In CiphertextToDecrypt
            plaintext += (Chr(Asc(letter) - EncryptionFactor))
        Next
        CiphertextToDecrypt = plaintext
    End Sub

    Private Sub ClearFields()
        ' Clear contents of all textboxes
        For Each control As Control In Me.Controls
            If TypeOf control Is TextBox Then
                control.Text = String.Empty
            End If
        Next
    End Sub

    Private Sub ResetLoginStructureValues(ByRef LoginAttempt As LoginValidation)
        ' Resests an instance of the LoginValidation structure
        LoginAttempt.InputDataValid = False
        LoginAttempt.CorrectFieldsFilled = False
        LoginAttempt.StudentUsernameEmpty = False
        LoginAttempt.StudentPasswordEmpty = False
        LoginAttempt.TeacherUsernameEmpty = False
        LoginAttempt.TeacherPasswordEmpty = False
        LoginAttempt.Student = False
        LoginAttempt.Teacher = False
        LoginAttempt.LoginDetailsFound = False
        LoginAttempt.filename = ""
        LoginAttempt.username = ""
        LoginAttempt.password = ""
        LoginAttempt.DecryptedCiphertext = ""
    End Sub
End Class
```

Code for ‘Create An Account’ module

```
Public Class frmCreateAccount
    Private Sub frmCreateAccount_Load(sender As Object, e As EventArgs) Handles
MyBase.Load
        Me.WindowState = FormWindowState.Maximized
    End Sub

    Structure AccountDetails
        <VBFixedString(20)> Public Username As String
        <VBFixedString(20)> Public Password As String
        <VBFixedString(20)> Public Forename As String
        <VBFixedString(20)> Public Surname As String
    End Structure
    Dim NewUser As AccountDetails

    Structure WriteAccountDetails
        Public filename As String
        Public EncryptedUsername As String
        Public EncryptedPassword As String
        Public EncryptedForename As String
```

```
        Public EncryptedSurname As String
    End Structure
    Dim AccountCreation As WriteAccountDetails

    Private Sub btnReturn_Click(sender As Object, e As EventArgs) Handles
btnReturn.Click
        Me.Hide()
        frmLoginPage.Show()
    End Sub

    Private Sub btnCreateAnAccount_Click(sender As Object, e As EventArgs) Handles
btnCreateAccount.Click
        ' Type check: ensuring only teacher or student has been selected for the
usertype
        If Not (String.IsNullOrEmpty(cmbUserType.Text) Or
String.IsNullOrEmpty(txtCreateUsername.Text) Or
String.IsNullOrEmpty(txtCreatePassword.Text) Or
String.IsNullOrEmpty(txtCreateForename.Text) Or
String.IsNullOrEmpty(txtCreateSurname.Text)) Then
            If cmbUserType.Text = "Student" Then
                AccountCreation.filename = "studentaccounts.txt"
            ElseIf cmbUserType.Text = "Teacher" Then
                AccountCreation.filename = "teacheraccounts.txt"
            Else
                MessageBox.Show("Invalid user type entered, account has not been
created.", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error)
            End If

            ASCIIEncryption(txtCreateUsername.Text, txtCreatePassword.Text,
txtCreateForename.Text, txtCreateSurname.Text, AccountCreation.EncryptedUsername,
AccountCreation.EncryptedPassword, AccountCreation.EncryptedForename,
AccountCreation.EncryptedSurname)

            Try
                Dim InvalidUsername As Boolean
                Dim i As Integer = 1
                Dim NextRecord As Integer
                FileOpen(1, AccountCreation.filename, OpenMode.Random, , ,
Len(NewUser))
                ' Iterate through txt file to identify if input username is already in
use by another user
                While Not EOF(1)
                    FileGet(1, NewUser, i)
                    If AccountCreation.EncryptedUsername = NewUser.Username.Trim Then
                        MessageBox.Show("This username is already in use. Please enter
a different username.", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error)
                        InvalidUsername = True
                    End If
                    i += 1
                End While
                If AccountCreation.EncryptedUsername.Length >= 20 Or
AccountCreation.EncryptedPassword.Length >= 20 Or
AccountCreation.EncryptedForename.Length >= 20 Or
AccountCreation.EncryptedSurname.Length >= 20 Then
                    MessageBox.Show("Maximum length of data entry for signup must be
less than or equal to 20", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error)
                    InvalidUsername = True
                End If
                If Not InvalidUsername Then
                    NextRecord = (LOF(1) / Len(NewUser)) + 1
                    With NewUser
                        .Username = AccountCreation.EncryptedUsername
```

```
                        .Password = AccountCreation.EncryptedPassword
                        .Forename = AccountCreation.EncryptedForename
                        .Surname = AccountCreation.EncryptedSurname
                    End With
                    FilePut(1, NewUser, NextRecord)
                    FileClose(1)
                    MessageBox.Show("Successful account creation", "Notice",
MessageBoxButtons.OK, MessageBoxIcon.Information)
                Else
                    FileClose(1)
                End If
            Catch ex As Exception
                MessageBox.Show("An error occured whilst attempting to log you in.
Please try again.", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error)
                FileClose(1)
            End Try
            ResetControls()
        Else
            MessageBox.Show("Please input values into all fields to complete sign-up",
"Error", MessageBoxButtons.OK, MessageBoxIcon.Error)
        End If
    End Sub

    Private Sub ASCIIEncryption(ByVal username As String, ByVal password As String,
ByVal forename As String, ByVal surname As String, ByRef EncryptedUsername As String,
ByRef EncryptedPassword As String, ByRef EncryptedForename As String, ByRef
EncryptedSurname As String)
        ' Perform ceasear shift with pad of 3
        Dim EncryptionFactor As Integer = 3
        For Each letter As String In username
            EncryptedUsername = EncryptedUsername + (Chr(Asc(letter) +
EncryptionFactor))
        Next
        For Each letter As String In password
            EncryptedPassword = EncryptedPassword + (Chr(Asc(letter) +
EncryptionFactor))
        Next
        For Each letter As String In forename
            EncryptedForename = EncryptedForename + (Chr(Asc(letter) +
EncryptionFactor))
        Next
        For Each letter As String In surname
            EncryptedSurname = EncryptedSurname + (Chr(Asc(letter) +
EncryptionFactor))
        Next
    End Sub

    Private Sub ResetControls()
        For Each control As Control In Me.Controls
            If TypeOf control Is TextBox Then
                control.Text = String.Empty
            ElseIf TypeOf control Is ComboBox Then
                cmbUserType.SelectedIndex = 0
            End If
        Next
    End Sub
End Class
```

Code for ‘Student Homepage’ module

```
Option Strict On
Public Class frmStudentHomepage
    ' Declare instances of the quiz modules
    Dim RandomisedTest As New frmRandomisedQuiz
    Dim SubTopicQuiz As New frmSubTopicQuiz
    Private Sub frmStudentHomepage_Load(sender As Object, e As EventArgs) Handles
MyBase.Load
        ' Display Randomised Quiz on load
        Me.WindowState = FormWindowState.Maximized
        RandomisedTest.MdiParent = Me
        RandomisedTest.WindowState = RandomisedTest.WindowState.Maximized
        RandomisedTest.Show()
    End Sub

    Private Sub tls_lbl_RandomisedQuiz_Click(sender As Object, e As EventArgs) Handles
tls_lbl_RandomisedQuiz.Click
        ' Show Randomised Quiz form
        frmRandomisedQuiz.FromSubTopic = False
        MinimiseForms()
        RandomisedTest.MdiParent = Me
        RandomisedTest.WindowState = RandomisedTest.WindowState.Maximized
        RandomisedTest.Show()
    End Sub

    Private Sub tbl_lbl_SubTopicQuiz_Click(sender As Object, e As EventArgs) Handles
tbl_lbl_SubTopicQuiz.Click
        ' Show Sub-topic Quiz form
        frmRandomisedQuiz.FromSubTopic = True
        MinimiseForms()
        SubTopicQuiz.MdiParent = Me
        SubTopicQuiz.WindowState = SubTopicQuiz.WindowState.Maximized
        SubTopicQuiz.Show()
    End Sub

    Private Sub MinimiseForms()
        ' Hide all open forms
        For Each form As Form In Me.MdiChildren
            form.WindowState = FormWindowState.Minimized
        Next
    End Sub
End Class
```

Code for ‘Randomised Quiz’ Module

```
Imports System.IO
Public Class frmRandomisedQuiz
    Public QuestionList As New List(Of String)
    Public FromSubTopic As Boolean

    Private MultipleChoiceAnswer As String = "N/A"
    Private NextQuestionNumber As Integer = 1
    Private Rndm As New Random
    Private UserAnswer As String
    Private Markscheme As Boolean
    Private SW As StreamWriter
```

```
    Structure Question
        Public QuestionTitle As String
        Public QuestionType As String
        Public NumberOfButtons As Integer
        Public ModuleType As String
        Public Topic As String
        Public SubTopic As String
        Public QuestionNumber As Integer
        Public CorrectAnswer As String
        Public DifficultyLevel As Integer
        Public IsAnswered As Boolean
        Public IsCorrect As Boolean
    End Structure

    Dim Question1 As New Question
    Dim Question2 As New Question
    Dim Question3 As New Question
    Dim Question4 As New Question
    Dim Question5 As New Question

    Structure TestResults
        <VBFixedString(12)> Dim TotalScore As String

        <VBFixedString(12)> Dim PureScore As String
        <VBFixedString(12)> Dim StatisticsScore As String
        <VBFixedString(12)> Dim MechanicsScore As String

        <VBFixedString(12)> Dim ProofScore As String
        <VBFixedString(12)> Dim IntegrationScore As String

        <VBFixedString(12)> Dim ProbabilityScore As String
        <VBFixedString(12)> Dim DistributionsScore As String

        <VBFixedString(12)> Dim CollisionsScore As String
        <VBFixedString(12)> Dim EnergyScore As String

        <VBFixedString(12)> Dim InductionScore As String
        <VBFixedString(12)> Dim CounterexampleScore As String

        <VBFixedString(12)> Dim Integrating_E_Score As String
        <VBFixedString(12)> Dim Integrating_LN_Score As String

        <VBFixedString(12)> Dim VennDiagramsScore As String
        <VBFixedString(12)> Dim LawsOfProbabilityScore As String

        <VBFixedString(12)> Dim BinomialScore As String
        <VBFixedString(12)> Dim NormalScore As String

        <VBFixedString(12)> Dim ElasticIn2DScore As String
        <VBFixedString(12)> Dim ObliqueScore As String

        <VBFixedString(12)> Dim KinecticScore As String
        <VBFixedString(12)> Dim GravitationalScore As String

        <VBFixedString(6)> Dim NumOfAttempts As String
    End Structure
    Dim SaveResults As New TestResults

    Private Sub RandomisedQuiz_Load(sender As Object, e As EventArgs) Handles
MyBase.Load
        Me.WindowState = FormWindowState.Maximized
        For Each item In QuestionList
```

```
            MessageBox.Show(item)
        Next
        LoadQuiz(QuestionList, FromSubTopic)
    End Sub

Private Sub LoadQuizButtons(ByVal NumberOfButtons As Integer)
        Dim btnMultipleChoice(NumberOfButtons) As Button
        Dim btnAdditonalControls(5) As Button

        ' Dynamically generate multiple choice input controls
        Dim X, Y As Integer
        Dim count As Integer
        Dim letters As String = "ABCDEFGH"
        ' Value of zero buttons is stored as 100 (value cannot be null)
        If NumberOfButtons > 8 Then
            X = 930
            Y = 200
            ' Generate text box
            AddNewTextBox(X, Y)

        ElseIf NumberOfButtons Mod 2 <> 0 Then
            ' Generate odd number of buttons
            X = 925
            Y = 60
            count = 0
            For i = 1 To (NumberOfButtons / 2)
                For j = 1 To 2
                    count += 1
                    btnMultipleChoice(count) = New Button
                    btnMultipleChoice(count).Location = New Point(X, Y)
                    btnMultipleChoice(count).Width = 90
                    btnMultipleChoice(count).Height = 65
                    btnMultipleChoice(count).Text = letters.Substring(count - 1, 1)
                    btnMultipleChoice(count).Name = "btn" & letters.Substring(count -
1, 1)
                    btnMultipleChoice(count).Font = New Font("Segoe UI", 16.0)
                    Me.Controls.Add(btnMultipleChoice(count))
                    X += 125
                Next
                Y += 80
                X = 925
            Next
            ' Add additional odd button
            count += 1
            X = 925
            btnMultipleChoice(count) = New Button
            btnMultipleChoice(count).Location = New Point(X, Y)
            btnMultipleChoice(count).Width = 90
            btnMultipleChoice(count).Height = 65
            btnMultipleChoice(count).Text = letters.Substring(count - 1, 1)
            btnMultipleChoice(count).Name = "btn" & letters.Substring(count - 1, 1)
            btnMultipleChoice(count).Font = New Font("Segoe UI", 16.0)
            Me.Controls.Add(btnMultipleChoice(count))
        Else
            ' Generate even number of buttons
            X = 925
            Y = 60
            count = 0
            For i = 1 To (NumberOfButtons / 2)
                For j = 1 To 2
                    count += 1
                    btnMultipleChoice(count) = New Button
```

```
                    btnMultipleChoice(count).Location = New Point(X, Y)
                    btnMultipleChoice(count).Width = 90
                    btnMultipleChoice(count).Height = 65
                    btnMultipleChoice(count).Text = letters.Substring(count - 1, 1)
                    btnMultipleChoice(count).Name = "btn" & letters.Substring(count -
1, 1)
                    btnMultipleChoice(count).Font = New Font("Segoe UI", 16.0)
                    Me.Controls.Add(btnMultipleChoice(count))
                    X += 125
                Next
                Y += 80
                X = 925
            Next
        End If


        ' Dynamically generate additional control buttons
        count = 0
        For i = 1 To 5
            count += 1
            btnAdditonalControls(count) = New Button
            btnAdditonalControls(count).Width = 140
            btnAdditonalControls(count).Height = 50
            btnAdditonalControls(count).Font = New Font("Segoe UI", 14.0,
FontStyle.Bold)
            Me.Controls.Add(btnAdditonalControls(count))
        Next

        btnAdditonalControls(1).Location = New Point(960, 360)
        btnAdditonalControls(1).Text = "Submit"
        btnAdditonalControls(1).Name = "btnSubmit"

        btnAdditonalControls(2).Location = New Point(870, 480)
        btnAdditonalControls(2).Text = "Previous"
        btnAdditonalControls(2).Name = "btnPrevious"

        btnAdditonalControls(3).Location = New Point(1050, 480)
        btnAdditonalControls(3).Text = "Next"
        btnAdditonalControls(3).Name = "btnNext"

        btnAdditonalControls(4).Location = New Point(960, 540)
        btnAdditonalControls(4).Text = "End Quiz"
        btnAdditonalControls(4).Name = "btnEndQuiz"
        btnAdditonalControls(4).ForeColor = Color.Maroon

        btnAdditonalControls(5).Location = New Point(960, 300)
        btnAdditonalControls(5).Text = "Input Rules"
        btnAdditonalControls(5).Name = "btnInputRules"

        ' Add handlers
        Select Case NumberOfButtons
            Case = 1
                AddHandler btnMultipleChoice(1).Click, AddressOf btnClickA
            Case = 2
                AddHandler btnMultipleChoice(1).Click, AddressOf btnClickA
                AddHandler btnMultipleChoice(2).Click, AddressOf btnClickB
            Case = 3
                AddHandler btnMultipleChoice(1).Click, AddressOf btnClickA
                AddHandler btnMultipleChoice(2).Click, AddressOf btnClickB
                AddHandler btnMultipleChoice(3).Click, AddressOf btnClickC
            Case = 4
                AddHandler btnMultipleChoice(1).Click, AddressOf btnClickA
```

```
                AddHandler btnMultipleChoice(2).Click, AddressOf btnClickB
                AddHandler btnMultipleChoice(3).Click, AddressOf btnClickC
                AddHandler btnMultipleChoice(4).Click, AddressOf btnClickD
            Case = 5
                AddHandler btnMultipleChoice(1).Click, AddressOf btnClickA
                AddHandler btnMultipleChoice(2).Click, AddressOf btnClickB
                AddHandler btnMultipleChoice(3).Click, AddressOf btnClickC
                AddHandler btnMultipleChoice(4).Click, AddressOf btnClickD
                AddHandler btnMultipleChoice(5).Click, AddressOf btnClickE
            Case = 6
                AddHandler btnMultipleChoice(1).Click, AddressOf btnClickA
                AddHandler btnMultipleChoice(2).Click, AddressOf btnClickB
                AddHandler btnMultipleChoice(3).Click, AddressOf btnClickC
                AddHandler btnMultipleChoice(4).Click, AddressOf btnClickD
                AddHandler btnMultipleChoice(5).Click, AddressOf btnClickE
                AddHandler btnMultipleChoice(6).Click, AddressOf btnClickF
            Case = 7
                AddHandler btnMultipleChoice(1).Click, AddressOf btnClickA
                AddHandler btnMultipleChoice(2).Click, AddressOf btnClickB
                AddHandler btnMultipleChoice(3).Click, AddressOf btnClickC
                AddHandler btnMultipleChoice(4).Click, AddressOf btnClickD
                AddHandler btnMultipleChoice(5).Click, AddressOf btnClickE
                AddHandler btnMultipleChoice(6).Click, AddressOf btnClickF
                AddHandler btnMultipleChoice(7).Click, AddressOf btnClickG
            Case = 8
                AddHandler btnMultipleChoice(1).Click, AddressOf btnClickA
                AddHandler btnMultipleChoice(2).Click, AddressOf btnClickB
                AddHandler btnMultipleChoice(3).Click, AddressOf btnClickC
                AddHandler btnMultipleChoice(4).Click, AddressOf btnClickD
                AddHandler btnMultipleChoice(5).Click, AddressOf btnClickE
                AddHandler btnMultipleChoice(6).Click, AddressOf btnClickF
                AddHandler btnMultipleChoice(7).Click, AddressOf btnClickG
                AddHandler btnMultipleChoice(8).Click, AddressOf btnClickH
        End Select


        AddHandler btnAdditonalControls(1).Click, AddressOf btnSubmit_Click
        AddHandler btnAdditonalControls(2).Click, AddressOf btnPrevious_Click
        AddHandler btnAdditonalControls(3).Click, AddressOf btnNext_Click
        AddHandler btnAdditonalControls(4).Click, AddressOf btnEndQuiz_Click
        AddHandler btnAdditonalControls(5).Click, AddressOf btnInputRules_Click
    End Sub

    Private Function AddNewTextBox(ByVal X As Integer, ByVal Y As Integer) As
System.Windows.Forms.TextBox
        Dim txtAnswer As New System.Windows.Forms.TextBox()
        Me.Controls.Add(txtAnswer)
        txtAnswer.Location = New Point(X, Y)
        txtAnswer.Name = "txtAnswerBox"
        txtAnswer.Width = 200
        txtAnswer.Height = 60
        Return txtAnswer
    End Function

    Private Sub DeleteControls()
        For i As Integer = Me.Controls.Count - 1 To 0 Step -1
            If TypeOf Me.Controls(i) Is Button Or TypeOf Me.Controls(i) Is TextBox
Then
                Me.Controls.RemoveAt(i)
            End If
        Next
    End Sub
```

```
Private Sub DetermineQuestionToShow(ByVal CurrentQuestion As Question)
        ' Displays question image related to question name
        Select Case CurrentQuestion.QuestionTitle
            Case = "INDUCTION1"
                pcbQuestions.Image = My.Resources.INDUCTION1_QUE
            Case = "INDUCTION2"
                pcbQuestions.Image = My.Resources.INDUCTION2_QUE
            Case = "COUNTEREXAMPLE1"
                pcbQuestions.Image = My.Resources.COUNTEREXAMPLE1_QUE
            Case = "COUNTEREXAMPLE2"
                pcbQuestions.Image = My.Resources.COUNTEREXAMPLE2_QUE
            Case = "INTEGRATING_E1"
                pcbQuestions.Image = My.Resources.INTEGRATING_E1_QUE
            Case = "INTEGRATING_LN1"
                pcbQuestions.Image = My.Resources.INTEGRATINGLN_1_QUE
            Case = "VENNDIAGRAMS1"
                pcbQuestions.Image = My.Resources.VENNDIAGRAMS1_QUE
            Case = "LAWSOFPROBABILITY1"
                pcbQuestions.Image = My.Resources.LAWSOFPROBABILITY1_QUE
            Case = "BINOMIAL1"
                pcbQuestions.Image = My.Resources.BINOMIAL1_QUE
            Case = "NORMAL1"
                pcbQuestions.Image = My.Resources.NORMAL1_QUE
            Case = "ELASTICIN2D1"
                pcbQuestions.Image = My.Resources.ELASTICIN2D1_QUE
            Case = "ELASTICIN2D2"
                pcbQuestions.Image = My.Resources.ELASTICIN2D2_QUE
            Case = "OBLIQUE1"
                pcbQuestions.Image = My.Resources.OBLIQUE1_QUE
            Case = "KINETIC1"
                pcbQuestions.Image = My.Resources.KINETIC1_QUE
            Case = "GRAVITATIONAL1"
                pcbQuestions.Image = My.Resources.GRAVITATIONAL1_QUE
            Case Else
                MessageBox.Show("ERROR: IMAGE COULD NOT BE FOUND: STRING TO BE PARSED
WAS " & CurrentQuestion.QuestionTitle, "Error", MessageBoxButtons.OK,
MessageBoxIcon.Error)
        End Select
    End Sub

    Private Sub DetermineAnswerToShow(ByVal CurrentAnswer As Question)
        ' Displays answer image related to answer name
        Select Case CurrentAnswer.QuestionTitle
            Case = "INDUCTION1"
                pcbQuestions.Image = My.Resources.INDUCTION01_ANS
            Case = "INDUCTION2"
                pcbQuestions.Image = My.Resources.INDUCTION02_ANS
            Case = "COUNTEREXAMPLE1"
                pcbQuestions.Image = My.Resources.COUNTEREXAMPLE1_ANS
            Case = "COUNTEREXAMPLE2"
                pcbQuestions.Image = My.Resources.COUNTEREXAMPLE2_ANS
            Case = "INTEGRATING_E1"
                pcbQuestions.Image = My.Resources.INTEGRATING_E1_ANS
            Case = "INTEGRATING_LN1"
                pcbQuestions.Image = My.Resources.INTEGRATING_LN1_ANS
            Case = "VENNDIAGRAMS1"
                pcbQuestions.Image = My.Resources.VENNDIAGRAMS1_ANS
            Case = "LAWSOFPROBABILITY1"
                pcbQuestions.Image = My.Resources.LAWSOFPROBABILITY1_ANS
            Case = "BINOMIAL1"
```

```
                pcbQuestions.Image = My.Resources.BINOMIAL1_ANS
            Case = "NORMAL1"
                pcbQuestions.Image = My.Resources.NORMAL1_ANS
            Case = "ELASTICIN2D1"
                pcbQuestions.Image = My.Resources.ELASTICIN2D1_ANS
            Case = "ELASTICIN2D2"
                pcbQuestions.Image = My.Resources.ELASTICIN2D2_ANS
            Case = "OBLIQUE1"
                pcbQuestions.Image = My.Resources.OBLIQUE1_ANS
            Case = "KINETIC1"
                pcbQuestions.Image = My.Resources.KINETIC1_ANS
            Case = "GRAVITATIONAL1"
                pcbQuestions.Image = My.Resources.GRAVITATIONAL1_ANS
            Case Else
                MessageBox.Show("ERROR: IMAGE COULD NOT BE FOUND: STRING TO BE PARSED
WAS " & CurrentAnswer.QuestionTitle, "Error", MessageBoxButtons.OK,
MessageBoxIcon.Error)
        End Select
    End Sub

    Private Sub btnClickA(ByVal sender As System.Object, ByVal e As System.EventArgs)
        MultipleChoiceAnswer = "A"
        lblSelection.Text = "Your Selection: A"
    End Sub


Private Sub btnClickB(ByVal sender As System.Object, ByVal e As System.EventArgs)
        MultipleChoiceAnswer = "B"
        lblSelection.Text = "Your Selection: B"
    End Sub

    Private Sub btnClickC(ByVal sender As System.Object, ByVal e As System.EventArgs)
        MultipleChoiceAnswer = "C"
        lblSelection.Text = "Your Selection: C"
    End Sub

    Private Sub btnClickD(ByVal sender As System.Object, ByVal e As System.EventArgs)
        MultipleChoiceAnswer = "D"
        lblSelection.Text = "Your Selection: D"
    End Sub




Private Sub btnClickE(ByVal sender As System.Object, ByVal e As System.EventArgs)
        MultipleChoiceAnswer = "E"
        lblSelection.Text = "Your Selection: E"
    End Sub

    Private Sub btnClickF(ByVal sender As System.Object, ByVal e As System.EventArgs)
        MultipleChoiceAnswer = "F"
        lblSelection.Text = "Your Selection: F"
    End Sub

    Private Sub btnClickG(ByVal sender As System.Object, ByVal e As System.EventArgs)
        MultipleChoiceAnswer = "G"
        lblSelection.Text = "Your Selection: G"
    End Sub


Private Sub btnClickH(ByVal sender As System.Object, ByVal e As System.EventArgs)
        MultipleChoiceAnswer = "H"
```

```
        lblSelection.Text = "Your Selection: H"
    End Sub

    Private Sub btnInputRules_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs)
        MessageBox.Show("The rules for inputting values into textboxes are as follows:
" & Environment.NewLine & "Exponents: ^ " & Environment.NewLine & "Multiplication: *"
& Environment.NewLine & "Division: /", "Input Rules", MessageBoxButtons.OK,
MessageBoxIcon.Information)
    End Sub

    Private Function FormContainsTextbox()
        ' Check if textbox is present on form
        Dim ContainsTextbox As Boolean
        For i As Integer = Me.Controls.Count - 1 To 0 Step -1
            If TypeOf Me.Controls(i) Is TextBox Then
                ContainsTextbox = True
            End If
        Next
        Return ContainsTextbox
    End Function



    Private Sub btnSubmit_Click(sender As Object, e As EventArgs)
        ' Determine if form contains a textbox
        Dim ContainsTextbox As Boolean
        ContainsTextbox = FormContainsTextbox()

        Dim ActiveQuesiton As New Question
        ActiveQuesiton = DetermineNextQuestion()

        If ContainsTextbox Then
            ' Search the form for the textbox
            For i As Integer = Me.Controls.Count - 1 To 0 Step -1
                If TypeOf Me.Controls(i) Is TextBox Then
                    ' Check if textbox is empty
                    If Not Me.Controls(i).Text Is Nothing Then
                        ActiveQuesiton.IsAnswered = True
                        ' Compare textbox value to correct answer
                        If Me.Controls(i).Text = ActiveQuesiton.CorrectAnswer Then
                            ActiveQuesiton.IsCorrect = True
                        End If
                        Me.lblAnswered.Text = "Answered: ✓"
                        TransferAnswer(ActiveQuesiton)
                    Else
                        MessageBox.Show("No answer has been input", "Error",
MessageBoxButtons.OK, MessageBoxIcon.Error)
                    End If
                End If
            Next
        Else
            ' If form does not contain textbox, determine if a button has been pressed
            If Not Me.lblSelection.Text = "Your Selection: N/A" Then
                Me.lblSelection.Text = "Your Selection: N/A"
                ActiveQuesiton.IsAnswered = True
                Me.lblAnswered.Text = "Answered: ✓"
                ' Determine if the button pressed is the correct answer
                If MultipleChoiceAnswer = ActiveQuesiton.CorrectAnswer Then
                    ActiveQuesiton.IsCorrect = True
                Else
                    ActiveQuesiton.IsCorrect = False
```

```
                End If
                TransferAnswer(ActiveQuesiton)
            Else
                MessageBox.Show("No answer has been input", "Error",
MessageBoxButtons.OK, MessageBoxIcon.Error)
            End If
        End If
        MultipleChoiceAnswer = "N/A"
    End Sub

    Private Sub btnNext_Click(sender As Object, e As EventArgs)
        ' Cycle through questions
        Dim NextQuestion As Question
        NextQuestionNumber += 1
        If NextQuestionNumber > 5 Then
            NextQuestionNumber = 1
        End If
        ' Fetching next question details
        NextQuestion = DetermineNextQuestion()
        DeleteControls()
        ' Reloading buttons for next question
        LoadQuizButtons(NextQuestion.NumberOfButtons)
        ' Displaying next question
        If Markscheme Then
            DetermineAnswerToShow(NextQuestion)
        Else
            DetermineQuestionToShow(NextQuestion)
        End If
        UpdateAnswerLabel()
    End Sub

    Private Sub btnPrevious_Click(sender As Object, e As EventArgs)
        ' Cycle through questions
        Dim NextQuestion As Question
        NextQuestionNumber -= 1
        If NextQuestionNumber < 1 Then
            NextQuestionNumber = 5
        End If
        ' Fetching next question details
        NextQuestion = DetermineNextQuestion()
        DeleteControls()
        ' Reloading buttons for next question
        LoadQuizButtons(NextQuestion.NumberOfButtons)
        ' Displaying next question
        If Markscheme Then
            DetermineAnswerToShow(NextQuestion)
        Else
            DetermineQuestionToShow(NextQuestion)
        End If
        UpdateAnswerLabel()
    End Sub


Private Function DetermineNextQuestion()
        ' Fetches next question number
        Dim ActiveQuestion As New Question
        If NextQuestionNumber = Question1.QuestionNumber Then
            ActiveQuestion = Question1
        ElseIf NextQuestionNumber = Question2.QuestionNumber Then
            ActiveQuestion = Question2
        ElseIf NextQuestionNumber = Question3.QuestionNumber Then
            ActiveQuestion = Question3
```

```
        ElseIf NextQuestionNumber = Question4.QuestionNumber Then
            ActiveQuestion = Question4
        ElseIf NextQuestionNumber = Question5.QuestionNumber Then
            ActiveQuestion = Question5
        Else
            MessageBox.Show("An error has occured whilst attempting to switch
questions", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error)
        End If
        Return ActiveQuestion
    End Function

    Private Sub TransferAnswer(ByVal ActiveQuestion As Question)
        ' Transfer values over from placeholder structure to current question
structure
        If NextQuestionNumber = Question1.QuestionNumber Then
            Question1.IsAnswered = ActiveQuestion.IsAnswered
            Question1.IsCorrect = ActiveQuestion.IsCorrect
        ElseIf NextQuestionNumber = Question2.QuestionNumber Then
            Question2.IsAnswered = ActiveQuestion.IsAnswered
            Question2.IsCorrect = ActiveQuestion.IsCorrect
        ElseIf NextQuestionNumber = Question3.QuestionNumber Then
            Question3.IsAnswered = ActiveQuestion.IsAnswered
            Question3.IsCorrect = ActiveQuestion.IsCorrect
        ElseIf NextQuestionNumber = Question4.QuestionNumber Then
            Question4.IsAnswered = ActiveQuestion.IsAnswered
            Question4.IsCorrect = ActiveQuestion.IsCorrect
        ElseIf NextQuestionNumber = Question5.QuestionNumber Then
            Question5.IsAnswered = ActiveQuestion.IsAnswered
            Question5.IsCorrect = ActiveQuestion.IsCorrect
        End If
    End Sub


Private Sub btnEndQuiz_Click(sender As Object, e As EventArgs)
        Dim TotalScore As Integer
        Dim Sucsessful As Boolean

        Dim ViewMarkScheme As String

        ' Determine if the user has already been asked whether or not they want to
view the markscheme
        If Not Markscheme Then
            ' Determine total score
            If Question1.IsCorrect = True Then TotalScore += 1
            If Question2.IsCorrect = True Then TotalScore += 1
            If Question3.IsCorrect = True Then TotalScore += 1
            If Question4.IsCorrect = True Then TotalScore += 1
            If Question5.IsCorrect = True Then TotalScore += 1

            MessageBox.Show("Your total score is: " & TotalScore)

            ' Display markscheme if user specifies to do so
            ViewMarkScheme = MsgBox("Thank you for taking the quiz. Would you like to
view the markscheme?", vbQuestion + vbYesNo + vbDefaultButton2, "Notice")
            If ViewMarkScheme = vbYes Then
                Markscheme = True
                DetermineAnswerToShow(Question1)
            End If
        End If

        ' Read student's question data from file into SaveResults structure
        ReadData(Sucsessful)
```

```
        ' Place quiz results into SaveResults structure
        If Sucsessful Then
            UpdateResults(Question1)
            UpdateResults(Question2)
            UpdateResults(Question3)
            UpdateResults(Question4)
            UpdateResults(Question5)
        End If

        ' Overwrite contents of files with updated SaveResults structure & module
averages
        WriteData()

        ' Reset Question structure values
        Question1.IsCorrect = False
        Question1.IsAnswered = False
        Question2.IsCorrect = False
        Question2.IsAnswered = False
        Question3.IsCorrect = False
        Question3.IsAnswered = False
        Question4.IsCorrect = False
        Question4.IsAnswered = False
        Question5.IsCorrect = False
        Question5.IsAnswered = False
    End Sub

    Private Sub ReadData(ByRef Sucsessful As Boolean)
        Dim filename As String = FetchFilename()
        Dim Score As Integer

        ' Populate instance of TestResults structure with values from a file
        Sucsessful = False
        Try
            Dim i As Integer = 1
            FileOpen(1, filename, OpenMode.Random, , , Len(SaveResults))
            While Not EOF(1)
                FileGet(1, SaveResults, i)
                i += 1
            End While
            FileClose(1)

            ' Increment SaveResults.NumOfAttempts by 1
            Score = CInt(SaveResults.NumOfAttempts.Substring(1, 5))
            Score += 1
            SaveResults.NumOfAttempts = CStr(Score)
            InsertLeadingZeros(SaveResults.NumOfAttempts)
            SaveResults.NumOfAttempts = "/" & SaveResults.NumOfAttempts

            Sucsessful = True
        Catch ex As Exception
            MessageBox.Show("An error occured whilst attempting to fetch your data",
"Error", MessageBoxButtons.OK, MessageBoxIcon.Error)
            FileClose(1)
        End Try
    End Sub

    Private Function FetchFilename()
        ' Fetch students's names from the login form
        Dim forename As String
        Dim surname As String
        Dim filename As String
```

```
        forename = frmLoginPage.UserLoginStructure.Forename.Trim()
        surname = frmLoginPage.UserLoginStructure.Surname.Trim()

        frmLoginPage.ASCIIDecryption(forename)
        frmLoginPage.ASCIIDecryption(surname)

        filename = forename + surname + ".txt"
        Return filename
    End Function



Private Sub WriteData()
        Dim filename As String = FetchFilename()
        Dim avgScore As String = CalculateAverageScore()

        ' Overwrite contents of results file with values stored in an instance of the
TestResults structure
        MessageBox.Show("Total score is: " & SaveResults.TotalScore)
        Try
            FileOpen(2, filename, OpenMode.Random, , , Len(SaveResults))
            With SaveResults
                .TotalScore = SaveResults.TotalScore
                .PureScore = SaveResults.PureScore
                .StatisticsScore = SaveResults.StatisticsScore
                .MechanicsScore = SaveResults.MechanicsScore
                .ProofScore = SaveResults.ProofScore
                .IntegrationScore = SaveResults.IntegrationScore
                .ProbabilityScore = SaveResults.ProbabilityScore
                .DistributionsScore = SaveResults.DistributionsScore
                .CollisionsScore = SaveResults.CollisionsScore
                .EnergyScore = SaveResults.EnergyScore
                .InductionScore = SaveResults.InductionScore
                .CounterexampleScore = SaveResults.CounterexampleScore
                .Integrating_E_Score = SaveResults.Integrating_E_Score
                .Integrating_LN_Score = SaveResults.Integrating_LN_Score
                .VennDiagramsScore = SaveResults.VennDiagramsScore
                .LawsOfProbabilityScore = SaveResults.LawsOfProbabilityScore
                .BinomialScore = SaveResults.BinomialScore
                .NormalScore = SaveResults.NormalScore
                .ElasticIn2DScore = SaveResults.ElasticIn2DScore
                .ObliqueScore = SaveResults.ObliqueScore
                .KinecticScore = SaveResults.KinecticScore
                .GravitationalScore = SaveResults.GravitationalScore
                .NumOfAttempts = SaveResults.NumOfAttempts
            End With
            FilePut(2, SaveResults, 1)
            FileClose(2)

            ' Update filename
            filename = filename.Insert(filename.Length - 4, "Attempts")

            ' Write percentage score to student's Attempt file
            SW = New StreamWriter(filename, True)
            SW.WriteLine(avgScore)
            SW.Close()
        Catch ex As Exception
            MessageBox.Show("An error occured whilst attempting to save your test
result", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error)
            FileClose(2)
```

```
        End Try
    End Sub

    Private Sub IncrementIfIsTopic(ByVal QuestionX As Question, ByVal ModuleType As
String, ByRef TotalNum As Integer, ByRef TotalNumCorrect As String)
        ' Increments the counter variables to keep track of how many questions for a
particular topic have been answered, and how many are correct
        If QuestionX.ModuleType = ModuleType Then
            TotalNum += 1
            If QuestionX.IsCorrect = True Then TotalNumCorrect += 1
        End If

    End Sub

    Private Sub AddAverageToString(ByRef CombinedAvg As String, ByVal TotalNum As
Integer, ByVal TotalNumCorrect As Integer)
        ' Add topic average onto combined average string
        Dim avg As String
        If TotalNum = 0 Then
            CombinedAvg += "///"
        Else
            avg = CStr(Math.Round((TotalNumCorrect / TotalNum) * 100, 0))
            ' Add leading zeros if necessary so avg is 2 chars long
            If avg.Length = 1 Then
                CombinedAvg = CombinedAvg + "00" + avg
            ElseIf avg.Length = 2 Then
                CombinedAvg = CombinedAvg + "0" + avg
            Else
                CombinedAvg += avg
            End If
        End If
    End Sub




Private Sub ResetTotalNumVariables(ByRef TotalNum As Integer, ByRef TotalNumCorrect As
Integer)
        ' Set TotalNum variables to 0
        TotalNum = 0
        TotalNumCorrect = 0
    End Sub

    Private Function CalculateAverageScore()
        ' Return a single string containing the number of attempts (4 chars) and
percentage correct (2 chars) for each topic
        Dim TotalNum As Integer
        Dim TotalNumCorrect As Integer
        Dim CombinedAvg As String

        ' Calculate pure maths average
        IncrementIfIsTopic(Question1, "PUREMATHS", TotalNum, TotalNumCorrect)
        IncrementIfIsTopic(Question2, "PUREMATHS", TotalNum, TotalNumCorrect)
        IncrementIfIsTopic(Question3, "PUREMATHS", TotalNum, TotalNumCorrect)
        IncrementIfIsTopic(Question4, "PUREMATHS", TotalNum, TotalNumCorrect)
        IncrementIfIsTopic(Question5, "PUREMATHS", TotalNum, TotalNumCorrect)
        AddAverageToString(CombinedAvg, TotalNum, TotalNumCorrect)
        ResetTotalNumVariables(TotalNum, TotalNumCorrect)

        ' Calculate statistics average
        IncrementIfIsTopic(Question1, "STATISTICS", TotalNum, TotalNumCorrect)
        IncrementIfIsTopic(Question2, "STATISTICS", TotalNum, TotalNumCorrect)
```

```
        IncrementIfIsTopic(Question3, "STATISTICS", TotalNum, TotalNumCorrect)
        IncrementIfIsTopic(Question4, "STATISTICS", TotalNum, TotalNumCorrect)
        IncrementIfIsTopic(Question5, "STATISTICS", TotalNum, TotalNumCorrect)
        AddAverageToString(CombinedAvg, TotalNum, TotalNumCorrect)
        ResetTotalNumVariables(TotalNum, TotalNumCorrect)

        ' Calculate mechanics average
        IncrementIfIsTopic(Question1, "MECHANICS", TotalNum, TotalNumCorrect)
        IncrementIfIsTopic(Question2, "MECHANICS", TotalNum, TotalNumCorrect)
        IncrementIfIsTopic(Question3, "MECHANICS", TotalNum, TotalNumCorrect)
        IncrementIfIsTopic(Question4, "MECHANICS", TotalNum, TotalNumCorrect)
        IncrementIfIsTopic(Question5, "MECHANICS", TotalNum, TotalNumCorrect)
        AddAverageToString(CombinedAvg, TotalNum, TotalNumCorrect)
        ResetTotalNumVariables(TotalNum, TotalNumCorrect)
        Return CombinedAvg
    End Function

    Private Sub UpdateResults(ByVal CurrentQuestion As Question)
        ' Each results string is in format Total attempts correct (6 chars) / Total
attempts incorect (6 chars)

        Dim ResultsString As String

        ' Update .TotalScore
        ResultsString = SaveResults.TotalScore
        IncrementResultsString(ResultsString, CurrentQuestion.IsCorrect)
        SaveResults.TotalScore = ResultsString

        ' Update SaveResults to include new scores for module, topic & subtopic
        Select Case CurrentQuestion.SubTopic
            Case = "INDUCTION"
                ' Update module score
                ResultsString = SaveResults.PureScore
                IncrementResultsString(ResultsString, CurrentQuestion.IsCorrect)
                SaveResults.PureScore = ResultsString

                ' Update topic score
                ResultsString = SaveResults.ProofScore
                IncrementResultsString(ResultsString, CurrentQuestion.IsCorrect)
                SaveResults.ProofScore = ResultsString

                ' Update sub topic score
                ResultsString = SaveResults.InductionScore
                IncrementResultsString(ResultsString, CurrentQuestion.IsCorrect)
                SaveResults.InductionScore = ResultsString
            Case = "COUNTEREXAMPLE"
                ' Update module score
                ResultsString = SaveResults.PureScore
                IncrementResultsString(ResultsString, CurrentQuestion.IsCorrect)
                SaveResults.PureScore = ResultsString

                ' Update topic score
                ResultsString = SaveResults.ProofScore
                IncrementResultsString(ResultsString, CurrentQuestion.IsCorrect)
                SaveResults.ProofScore = ResultsString

                ' Update sub topic score
                ResultsString = SaveResults.CounterexampleScore
                IncrementResultsString(ResultsString, CurrentQuestion.IsCorrect)
                SaveResults.CounterexampleScore = ResultsString
            Case = "INTEGRATING_E"
                ' Update module score
```

```
        ResultsString = SaveResults.PureScore
        IncrementResultsString(ResultsString, CurrentQuestion.IsCorrect)
        SaveResults.PureScore = ResultsString

        ' Update topic score
        ResultsString = SaveResults.IntegrationScore
        IncrementResultsString(ResultsString, CurrentQuestion.IsCorrect)
        SaveResults.IntegrationScore = ResultsString

        ' Update sub topic score
        ResultsString = SaveResults.Integrating_E_Score
        IncrementResultsString(ResultsString, CurrentQuestion.IsCorrect)
        SaveResults.Integrating_E_Score = ResultsString
    Case = "INTEGRATING_LN"
        ' Update module score
        ResultsString = SaveResults.PureScore
        IncrementResultsString(ResultsString, CurrentQuestion.IsCorrect)
        SaveResults.PureScore = ResultsString

        ' Update topic score
        ResultsString = SaveResults.IntegrationScore
        IncrementResultsString(ResultsString, CurrentQuestion.IsCorrect)
        SaveResults.IntegrationScore = ResultsString

        ' Update sub topic score
        ResultsString = SaveResults.Integrating_LN_Score
        IncrementResultsString(ResultsString, CurrentQuestion.IsCorrect)
        SaveResults.Integrating_LN_Score = ResultsString
    Case = "VENNDIAGRAMS"
        ' Update module score
        ResultsString = SaveResults.StatisticsScore
        IncrementResultsString(ResultsString, CurrentQuestion.IsCorrect)
        SaveResults.StatisticsScore = ResultsString

        ' Update topic score
        ResultsString = SaveResults.ProbabilityScore
        IncrementResultsString(ResultsString, CurrentQuestion.IsCorrect)
        SaveResults.ProbabilityScore = ResultsString

        ' Update sub topic score
        ResultsString = SaveResults.VennDiagramsScore
        IncrementResultsString(ResultsString, CurrentQuestion.IsCorrect)
        SaveResults.VennDiagramsScore = ResultsString
    Case = "LAWSOFPROBABILITY"
        ' Update module score
        ResultsString = SaveResults.StatisticsScore
        IncrementResultsString(ResultsString, CurrentQuestion.IsCorrect)
        SaveResults.StatisticsScore = ResultsString

        ' Update topic score
        ResultsString = SaveResults.ProbabilityScore
        IncrementResultsString(ResultsString, CurrentQuestion.IsCorrect)
        SaveResults.ProbabilityScore = ResultsString

        ' Update sub topic score
        ResultsString = SaveResults.LawsOfProbabilityScore
        IncrementResultsString(ResultsString, CurrentQuestion.IsCorrect)
        SaveResults.LawsOfProbabilityScore = ResultsString
    Case = "BINOMIAL"
        ' Update module score
        ResultsString = SaveResults.StatisticsScore
        IncrementResultsString(ResultsString, CurrentQuestion.IsCorrect)
```

```
    SaveResults.StatisticsScore = ResultsString

    ' Update topic score
    ResultsString = SaveResults.DistributionsScore
    IncrementResultsString(ResultsString, CurrentQuestion.IsCorrect)
    SaveResults.DistributionsScore = ResultsString

    ' Update sub topic score
    ResultsString = SaveResults.BinomialScore
    IncrementResultsString(ResultsString, CurrentQuestion.IsCorrect)
    SaveResults.BinomialScore = ResultsString
Case = "NORMAL"
    ' Update module score
    ResultsString = SaveResults.StatisticsScore
    IncrementResultsString(ResultsString, CurrentQuestion.IsCorrect)
    SaveResults.StatisticsScore = ResultsString

    ' Update topic score
    ResultsString = SaveResults.DistributionsScore
    IncrementResultsString(ResultsString, CurrentQuestion.IsCorrect)
    SaveResults.DistributionsScore = ResultsString

    ' Update sub topic score
    ResultsString = SaveResults.NormalScore
    IncrementResultsString(ResultsString, CurrentQuestion.IsCorrect)
    SaveResults.NormalScore = ResultsString
Case = "ELASTICIN2D"
    ' Update module score
    ResultsString = SaveResults.MechanicsScore
    IncrementResultsString(ResultsString, CurrentQuestion.IsCorrect)
    SaveResults.MechanicsScore = ResultsString

    ' Update topic score
    ResultsString = SaveResults.CollisionsScore
    IncrementResultsString(ResultsString, CurrentQuestion.IsCorrect)
    SaveResults.CollisionsScore = ResultsString

    ' Update sub topic score
    ResultsString = SaveResults.ElasticIn2DScore
    IncrementResultsString(ResultsString, CurrentQuestion.IsCorrect)
    SaveResults.ElasticIn2DScore = ResultsString
Case = "OBLIQUE"
    ' Update module score
    ResultsString = SaveResults.MechanicsScore
    IncrementResultsString(ResultsString, CurrentQuestion.IsCorrect)
    SaveResults.MechanicsScore = ResultsString

    ' Update topic score
    ResultsString = SaveResults.CollisionsScore
    IncrementResultsString(ResultsString, CurrentQuestion.IsCorrect)
    SaveResults.CollisionsScore = ResultsString

    ' Update sub topic score
    ResultsString = SaveResults.ObliqueScore
    IncrementResultsString(ResultsString, CurrentQuestion.IsCorrect)
    SaveResults.ObliqueScore = ResultsString
Case = "KINETIC"
    ' Update module score
    ResultsString = SaveResults.MechanicsScore
    IncrementResultsString(ResultsString, CurrentQuestion.IsCorrect)
    SaveResults.MechanicsScore = ResultsString
```

```
                ' Update topic score
                ResultsString = SaveResults.EnergyScore
                IncrementResultsString(ResultsString, CurrentQuestion.IsCorrect)
                SaveResults.EnergyScore = ResultsString

                ' Update sub topic score
                ResultsString = SaveResults.KinecticScore
                IncrementResultsString(ResultsString, CurrentQuestion.IsCorrect)
                SaveResults.KinecticScore = ResultsString
            Case = "GRAVITATIONAL"
                ' Update module score
                ResultsString = SaveResults.MechanicsScore
                IncrementResultsString(ResultsString, CurrentQuestion.IsCorrect)
                SaveResults.MechanicsScore = ResultsString

                ' Update topic score
                ResultsString = SaveResults.EnergyScore
                IncrementResultsString(ResultsString, CurrentQuestion.IsCorrect)
                SaveResults.EnergyScore = ResultsString

                ' Update sub topic score
                ResultsString = SaveResults.GravitationalScore
                IncrementResultsString(ResultsString, CurrentQuestion.IsCorrect)
                SaveResults.GravitationalScore = ResultsString
        End Select
    End Sub

Private Sub ResetTestResultsStructure(ByRef ResultsStructure As TestResults)
        ' Resets the values of a TestResults structure to default
        ResultsStructure.TotalScore = "/00000/00000"
        ResultsStructure.PureScore = "/00000/00000"
        ResultsStructure.StatisticsScore = "/00000/00000"
        ResultsStructure.MechanicsScore = "/00000/00000"
        ResultsStructure.ProofScore = "/00000/00000"
        ResultsStructure.IntegrationScore = "/00000/00000"
        ResultsStructure.ProbabilityScore = "/00000/00000"
        ResultsStructure.DistributionsScore = "/00000/00000"
        ResultsStructure.CollisionsScore = "/00000/00000"
        ResultsStructure.EnergyScore = "/00000/00000"
        ResultsStructure.InductionScore = "/00000/00000"
        ResultsStructure.CounterexampleScore = "/00000/00000"
        ResultsStructure.Integrating_E_Score = "/00000/00000"
        ResultsStructure.Integrating_LN_Score = "/00000/00000"
        ResultsStructure.VennDiagramsScore = "/00000/00000"
        ResultsStructure.LawsOfProbabilityScore = "/00000/00000"
        ResultsStructure.BinomialScore = "/00000/00000"
        ResultsStructure.NormalScore = "/00000/00000"
        ResultsStructure.ElasticIn2DScore = "/00000/00000"
        ResultsStructure.ObliqueScore = "/00000/00000"
        ResultsStructure.KinecticScore = "/00000/00000"
        ResultsStructure.GravitationalScore = "/00000/00000"
        ResultsStructure.NumOfAttempts = "/00000"
    End Sub


Private Sub IncrementResultsString(ByRef ResultsString As String, ByVal Correct As
Boolean)
        Dim Score As Integer
        Dim ResultsSubstring As String
        Try
            If Correct Then
                ' Fetch first part of ResultsString
```

```
                ResultsSubstring = ResultsString.Substring(1, 5)

                ' Increment int value
                Score = CInt(ResultsSubstring)
                Score += 1

                ' Convert back to string
                ResultsSubstring = CStr(Score)

                ' Insert leading zeros
                InsertLeadingZeros(ResultsSubstring)

                ' Insert incremented score into ResultsString
                ResultsString = ResultsString.Substring(0, 1) & ResultsSubstring &
ResultsString.Substring(6, 6)
            Else
                ' Fetch second part of ResultsString
                ResultsSubstring = ResultsString.Substring(7, 5)

                ' Increment int value
                Score = CInt(ResultsSubstring)
                Score += 1

                ' Convert back to string
                ResultsSubstring = CStr(Score)

                ' Insert leading zeros
                InsertLeadingZeros(ResultsSubstring)

                ' Insert incremented score into ResultsString
                ResultsString = ResultsString.Substring(0, 7) & ResultsSubstring
            End If

        Catch ex As Exception
            Console.WriteLine("Error")
        End Try
    End Sub

    Private Sub InsertLeadingZeros(ByRef str As String)
        Select Case str.Length
            Case = 1
                str = "0000" & str
            Case = 2
                str = "000" & str
            Case = 3
                str = "00" & str
            Case = 4
                str = "0" & str
            Case Else
                ' Do nothing
        End Select
    End Sub


Private Sub UpdateAnswerLabel()
        ' Update answer label to indicate to user whether current question has been
answered
        If NextQuestionNumber = Question1.QuestionNumber Then
            If Question1.IsAnswered = False Then
                Me.lblAnswered.Text = "Answered: ✗"
            Else
                Me.lblAnswered.Text = "Answered: ✓"
```

```
            End If
        ElseIf NextQuestionNumber = Question2.QuestionNumber Then
            If Question2.IsAnswered = False Then
                Me.lblAnswered.Text = "Answered: ✗"
            Else
                Me.lblAnswered.Text = "Answered: ✓"
            End If
        ElseIf NextQuestionNumber = Question3.QuestionNumber Then
            If Question3.IsAnswered = False Then
                Me.lblAnswered.Text = "Answered: ✗"
            Else
                Me.lblAnswered.Text = "Answered: ✓"
            End If
        ElseIf NextQuestionNumber = Question4.QuestionNumber Then
            If Question4.IsAnswered = False Then
                Me.lblAnswered.Text = "Answered: ✗"
            Else
                Me.lblAnswered.Text = "Answered: ✓"
            End If
        ElseIf NextQuestionNumber = Question5.QuestionNumber Then
            If Question5.IsAnswered = False Then
                Me.lblAnswered.Text = "Answered: ✗"
            Else
                Me.lblAnswered.Text = "Answered: ✓"
            End If
        End If
    End Sub

    Private Function SelectRandomQuestion() As String
        ' Randomly select a question name

        Dim rnd = New Random()
        Dim ModuleName As String
        Dim TopicName As String
        Dim SubTopicName As String

        Dim ModuleNames As New List(Of String)

        Dim PureTopicNames As New List(Of String)
        Dim StatisticTopicNames As New List(Of String)
        Dim MechanicsTopicNames As New List(Of String)

        Dim PureSubTopic_Proof As New List(Of String)
        Dim PureSubTopic_Integration As New List(Of String)

        Dim StatisticsSubTopic_Probability As New List(Of String)
        Dim StatisticsSubTopic_Distributions As New List(Of String)

        Dim MechanicsSubTopic_Collisions As New List(Of String)
        Dim MechanicsSubTopic_Energy As New List(Of String)

        ModuleNames.AddRange(New String() {"PUREMATHS", "STATISTICS", "MECHANICS"})

        PureTopicNames.AddRange(New String() {"PROOF", "INTEGRATION"})
        StatisticTopicNames.AddRange(New String() {"PROBABILITY", "DISTRIBUTIONS"})
        MechanicsTopicNames.AddRange(New String() {"COLLISIONS", "ENERGY"})

        PureSubTopic_Proof.AddRange(New String() {"INDUCTION1", "INDUCTION2",
"COUNTEREXAMPLE1", "COUNTEREXAMPLE2"})
        PureSubTopic_Integration.AddRange(New String() {"INTEGRATING_E1",
"INTEGRATING_LN1"})
```

```
        StatisticsSubTopic_Probability.AddRange(New String() {"VENNDIAGRAMS1",
"LAWSOFPROBABILITY1"})
        StatisticsSubTopic_Distributions.AddRange(New String() {"BINOMIAL1",
"NORMAL1"})

        MechanicsSubTopic_Collisions.AddRange(New String() {"ELASTICIN2D1",
"ELASTICIN2D2", "OBLIQUE1"})
        MechanicsSubTopic_Energy.AddRange(New String() {"KINETIC1", "GRAVITATIONAL1"})

        ModuleName += ModuleNames(rnd.Next(0, ModuleNames.Count))

        Select Case ModuleName
            Case = "PUREMATHS"
                TopicName = PureTopicNames(rnd.Next(0, PureTopicNames.Count))
                Select Case TopicName
                    Case = "PROOF"
                        SubTopicName = PureSubTopic_Proof(rnd.Next(0,
PureSubTopic_Proof.Count))
                    Case = "INTEGRATION"
                        TopicName = "INTEGRATION"
                        SubTopicName = PureSubTopic_Integration(rnd.Next(0,
PureSubTopic_Integration.Count))
                    Case Else
                        MessageBox.Show("An error has occured whilst selecting the
pure topic", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error)
                End Select
            Case = "STATISTICS"
                TopicName = StatisticTopicNames(rnd.Next(0,
StatisticTopicNames.Count))
                Select Case TopicName
                    Case = "PROBABILITY"
                        SubTopicName = StatisticsSubTopic_Probability(rnd.Next(0,
StatisticsSubTopic_Probability.Count))
                        TopicName = "PROBABILITY"
                    Case = "DISTRIBUTIONS"
                        SubTopicName = StatisticsSubTopic_Distributions(rnd.Next(0,
StatisticsSubTopic_Distributions.Count))
                        TopicName = "DISTRIBUTIONS"
                    Case Else
                        MessageBox.Show("An error has occured whilst selecting the
statistics topic", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error)
                End Select
            Case = "MECHANICS"
                TopicName = MechanicsTopicNames(rnd.Next(0,
StatisticTopicNames.Count))
                Select Case TopicName
                    Case = "COLLISIONS"
                        SubTopicName = MechanicsSubTopic_Collisions(rnd.Next(0,
MechanicsSubTopic_Collisions.Count))
                        TopicName = "COLLISIONS"
                    Case = "ENERGY"
                        TopicName = "ENERGY"
                        SubTopicName = MechanicsSubTopic_Energy(rnd.Next(0,
MechanicsSubTopic_Energy.Count))
                    Case Else
                        MessageBox.Show("An error has occured whilst selecting the
mechanics topic", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error)
                End Select
            Case Else
                MessageBox.Show("An error has occured whilst selecting the module",
"Error", MessageBoxButtons.OK, MessageBoxIcon.Error)
        End Select
```

```
        Return SubTopicName
    End Function


Private Sub LoadQuiz(ByVal QuestionList As List(Of String), ByVal FromSubTopic As
Boolean)
        Dim FileName As String
        Markscheme = False

        If Not FromSubTopic Then
            ' Delete contents of QuestionList
            QuestionList.Clear()
            ' Randomly select names of 5 questions for the quiz
            Dim ProceduralllyGeneratedQuestionNames(4) As String
            Dim strQuestionName As String
            For i = 0 To ProceduralllyGeneratedQuestionNames.Count - 1
                strQuestionName = SelectRandomQuestion()
                ' Prevent repeats of questions from being selected
                Do Until Not
(ProceduralllyGeneratedQuestionNames.Contains(strQuestionName))
                    strQuestionName = SelectRandomQuestion()
                Loop
                ProceduralllyGeneratedQuestionNames(i) = strQuestionName
            Next
            ' Transfer contents of generated names over to QuestionList
            For Each item In ProceduralllyGeneratedQuestionNames
                QuestionList.AddRange(New String() {item})
            Next
        End If

        ' Reset values of SaveResults structure
        ResetTestResultsStructure(SaveResults)

        ' Read question information from text files
        Using reader As TextReader = New
StringReader(My.Resources.ResourceManager.GetObject(QuestionList(0) & "_INF"))
            Question1.QuestionTitle = reader.ReadLine
            Question1.QuestionType = reader.ReadLine
            Question1.NumberOfButtons = reader.ReadLine
            Question1.ModuleType = reader.ReadLine
            Question1.Topic = reader.ReadLine
            Question1.SubTopic = reader.ReadLine
            Question1.CorrectAnswer = reader.ReadLine
            Question1.DifficultyLevel = reader.ReadLine
            Question1.IsAnswered = reader.ReadLine
            Question1.IsCorrect = reader.ReadLine
        End Using

        DetermineFileName(FileName, QuestionList, 1)
        Using reader As TextReader = New
StringReader(My.Resources.ResourceManager.GetObject(FileName & "_INF"))
            Question2.QuestionTitle = reader.ReadLine
            Question2.QuestionType = reader.ReadLine
            Question2.NumberOfButtons = reader.ReadLine
            Question2.ModuleType = reader.ReadLine
            Question2.Topic = reader.ReadLine
            Question2.SubTopic = reader.ReadLine
            Question2.CorrectAnswer = reader.ReadLine
            Question2.DifficultyLevel = reader.ReadLine
            Question2.IsAnswered = reader.ReadLine
            Question2.IsCorrect = reader.ReadLine
        End Using
```

```
        DetermineFileName(FileName, QuestionList, 2)
        Using reader As TextReader = New
StringReader(My.Resources.ResourceManager.GetObject(FileName & "_INF"))
            Question3.QuestionTitle = reader.ReadLine
            Question3.QuestionType = reader.ReadLine
            Question3.NumberOfButtons = reader.ReadLine
            Question3.ModuleType = reader.ReadLine
            Question3.Topic = reader.ReadLine
            Question3.SubTopic = reader.ReadLine
            Question3.CorrectAnswer = reader.ReadLine
            Question3.DifficultyLevel = reader.ReadLine
            Question3.IsAnswered = reader.ReadLine
            Question3.IsCorrect = reader.ReadLine
        End Using

        DetermineFileName(FileName, QuestionList, 3)
        Using reader As TextReader = New
StringReader(My.Resources.ResourceManager.GetObject(FileName & "_INF"))
            Question4.QuestionTitle = reader.ReadLine
            Question4.QuestionType = reader.ReadLine
            Question4.NumberOfButtons = reader.ReadLine
            Question4.ModuleType = reader.ReadLine
            Question4.Topic = reader.ReadLine
            Question4.SubTopic = reader.ReadLine
            Question4.CorrectAnswer = reader.ReadLine
            Question4.DifficultyLevel = reader.ReadLine
            Question4.IsAnswered = reader.ReadLine
            Question4.IsCorrect = reader.ReadLine
        End Using

        DetermineFileName(FileName, QuestionList, 4)
        Using reader As TextReader = New
StringReader(My.Resources.ResourceManager.GetObject(FileName & "_INF"))
            Question5.QuestionTitle = reader.ReadLine
            Question5.QuestionType = reader.ReadLine
            Question5.NumberOfButtons = reader.ReadLine
            Question5.ModuleType = reader.ReadLine
            Question5.Topic = reader.ReadLine
            Question5.SubTopic = reader.ReadLine
            Question5.CorrectAnswer = reader.ReadLine
            Question5.DifficultyLevel = reader.ReadLine
            Question5.IsAnswered = reader.ReadLine
            Question5.IsCorrect = reader.ReadLine
        End Using

        Question1.QuestionNumber = 1
        Question2.QuestionNumber = 2
        Question3.QuestionNumber = 3
        Question4.QuestionNumber = 4
        Question5.QuestionNumber = 5

        LoadQuizButtons(Question1.NumberOfButtons)
        DetermineQuestionToShow(Question1)
        Me.lblSelection.Text = "Your Selection: N/A"
    End Sub

    Private Sub DetermineFileName(ByRef FileName As String, ByVal QuestionList As
List(Of String), ByVal n As Integer)
        ' Assign FileName string the value held at position n in the list; if there
are less than n items, assign FileName the final item in the list
        If QuestionList.Count > n Then
```

```
            FileName = QuestionList(n)
        Else
            FileName = QuestionList(QuestionList.Count - 1)
        End If
    End Sub
End Class
```

Code for ‘Select A Subtopic’ Module

```
Imports System.IO
Public Class frmSubTopicQuiz

    Private MultipleChoiceAnswer As String = "N/A"
    Private NextQuestionNumber As Integer = 1
    Private QuestionNumbers As New List(Of String)
    Private Rndm As New Random
    Private UserAnswer As String
    Private Markscheme As Boolean
    Private FilePath As String
    Private SubTopicQuestions As New List(Of String)

    Structure Question
        Public QuestionTitle As String
        Public QuestionType As String
        Public NumberOfButtons As Integer
        Public ModuleType As String
        Public Topic As String
        Public SubTopic As String
        Public QuestionNumber As Integer
        Public CorrectAnswer As String
        Public DifficultyLevel As Integer
        Public IsAnswered As Boolean
        Public IsCorrect As Boolean
    End Structure
    Dim Question1 As Question
    Dim Question2 As Question

    Structure TestResults
        <VBFixedString(12)> Dim TotalScore As String

        <VBFixedString(12)> Dim PureScore As String
        <VBFixedString(12)> Dim StatisticsScore As String
        <VBFixedString(12)> Dim MechanicsScore As String

        <VBFixedString(12)> Dim ProofScore As String
        <VBFixedString(12)> Dim IntegrationScore As String

        <VBFixedString(12)> Dim ProbabilityScore As String
        <VBFixedString(12)> Dim DistributionsScore As String

        <VBFixedString(12)> Dim CollisionsScore As String
        <VBFixedString(12)> Dim EnergyScore As String

        <VBFixedString(12)> Dim InductionScore As String
        <VBFixedString(12)> Dim CounterexampleScore As String

        <VBFixedString(12)> Dim Integrating_E_Score As String
        <VBFixedString(12)> Dim Integrating_LN_Score As String

        <VBFixedString(12)> Dim VennDiagramsScore As String
```

```
        <VBFixedString(12)> Dim LawsOfProbabilityScore As String

        <VBFixedString(12)> Dim BinomialScore As String
        <VBFixedString(12)> Dim NormalScore As String

        <VBFixedString(12)> Dim ElasticIn2DScore As String
        <VBFixedString(12)> Dim ObliqueScore As String

        <VBFixedString(12)> Dim KinecticScore As String
        <VBFixedString(12)> Dim GravitationalScore As String
    End Structure
    Dim SaveResults As New TestResults

    Structure SelectedFolders
        Public MathsModule As String
        Public Topic As String
    End Structure
    Dim QuizPath As New SelectedFolders


    Private Sub frmSubTopicQuiz_Load(sender As Object, e As EventArgs) Handles
MyBase.Load
        Me.WindowState = FormWindowState.Maximized
        LoadStartupElements()
        LoadModuleButtons()
    End Sub

Private Sub LoadStartupElements()
        ' Create picturebox
        Dim pcbFolder As New PictureBox
        pcbFolder.Name = "pcbFolder"
        pcbFolder.Width = 100
        pcbFolder.Height = 80
        pcbFolder.Image = My.Resources.Blue_Folder
        pcbFolder.Location = New Point(155, 85)
        pcbFolder.SizeMode = PictureBoxSizeMode.StretchImage
        Me.Controls.Add(pcbFolder)

        ' Create label
        Dim lblInfo As New Label
        lblInfo.Name = "lblInfo"
        lblInfo.Text = "Navigate to your desired sub-topic by clicking on the icons
below."
        lblInfo.Width = 320
        lblInfo.Height = 20
        lblInfo.Location = New Point(130, 10)
        Me.Controls.Add(lblInfo)

        ' Create collapse button
        Dim btnCollapse As New Button
        btnCollapse.Name = "btnCollapse"
        btnCollapse.Location = New Point(155, 190)
        btnCollapse.Width = 100
        btnCollapse.Height = 40
        btnCollapse.Text = "Collapse folders"
        btnCollapse.Font = New Font("Segoe UI", 14.0, FontStyle.Bold)
        Me.Controls.Add(btnCollapse)

        ' Add handler
        AddHandler btnCollapse.Click, AddressOf btnCollapse_Click
    End Sub
```

```
    Private Sub LoadModuleButtons()
        ' Load names of modules into dynamically created buttons
        Dim ModuleButtons(3) As Button
        Dim X, Y As Integer

        ' Declare starting position of first button
        X = 300
        Y = 100

        ' Add shared button properties
        For i = 1 To 3
            ModuleButtons(i) = New Button
            ModuleButtons(i).Location = New Point(X, Y)
            ModuleButtons(i).Width = 200
            ModuleButtons(i).Height = 50
            ModuleButtons(i).Font = New Font("Segoe UI", 13.0, FontStyle.Bold)
            Me.Controls.Add(ModuleButtons(i))
            X += 300
        Next

        ' Add specific button properties
        ModuleButtons(1).Name = "btnPure"
        ModuleButtons(2).Name = "btnStatistics"
        ModuleButtons(3).Name = "btnMechanics"

        ModuleButtons(1).Text = "Pure"
        ModuleButtons(2).Text = "Statistics"
        ModuleButtons(3).Text = "Mechanics"

        ' Add handlers
        AddHandler ModuleButtons(1).Click, AddressOf btnPure_Click
        AddHandler ModuleButtons(2).Click, AddressOf btnStatistics_Click
        AddHandler ModuleButtons(3).Click, AddressOf btnMechanics_Click
    End Sub

    Private Sub LoadTopicButtons()
        ' Load names of topics related to selected module into dynamically created
buttons
        Dim TopicButtons(2) As Button
        Dim X, Y As Integer

        ' Declare starting position of first button
        X = 400
        Y = 200

        ' Add shared button properties
        For i = 1 To 2
            TopicButtons(i) = New Button
            TopicButtons(i).Location = New Point(X, Y)
            TopicButtons(i).Width = 200
            TopicButtons(i).Height = 50
            TopicButtons(i).Font = New Font("Segoe UI", 13.0, FontStyle.Bold)
            Me.Controls.Add(TopicButtons(i))
            X += 400
        Next

        TopicButtons(1).Name = "btnTopic1"
        TopicButtons(2).Name = "btnTopic2"

        ' Add specific button properties based off selected module
        Select Case QuizPath.MathsModule
            Case = "Pure"
```

```
                TopicButtons(1).Text = "Proof"
                TopicButtons(2).Text = "Integration"
            Case = "Statistics"
                TopicButtons(1).Text = "Probability"
                TopicButtons(2).Text = "Distributions"
            Case = "Mechanics"
                TopicButtons(1).Text = "Collisions"
                TopicButtons(2).Text = "Energy"
        End Select

        ' Add handlers
        AddHandler TopicButtons(1).Click, AddressOf btnTopic1_Click
        AddHandler TopicButtons(2).Click, AddressOf btnTopic2_Click
    End Sub

    Private Sub LoadSubTopicButtons()
        ' Load names of sub topics related to selected topic into dynamically created
buttons
        Dim SubTopicButtons(2) As Button
        Dim X, Y As Integer

        ' Declare starting position of first button
        X = 500
        Y = 300

        ' Add shared button properties
        For i = 1 To 2
            SubTopicButtons(i) = New Button
            SubTopicButtons(i).Location = New Point(X, Y)
            SubTopicButtons(i).Width = 200
            SubTopicButtons(i).Height = 50
            SubTopicButtons(i).Font = New Font("Segoe UI", 14.0, FontStyle.Bold)
            Me.Controls.Add(SubTopicButtons(i))
            X += 200
        Next

        SubTopicButtons(1).Name = "btnSubTopic1"
        SubTopicButtons(2).Name = "btnSubTopic2"

        ' Add specific button properties based off selected module
        Select Case QuizPath.Topic
            Case = "Proof"
                SubTopicButtons(1).Text = "Induction"
                SubTopicButtons(2).Text = "Counter-example"
            Case = "Integration"
                SubTopicButtons(1).Text = "Integrating e"
                SubTopicButtons(2).Text = "Integrating ln"
            Case = "Probability"
                SubTopicButtons(1).Text = "Venn diagrams"
                SubTopicButtons(2).Text = "Laws of probability"
            Case = "Distributions"
                SubTopicButtons(1).Text = "Binomial"
                SubTopicButtons(2).Text = "Normal"
            Case = "Collisions"
                SubTopicButtons(1).Text = "Elastic in 2D"
                SubTopicButtons(2).Text = "Oblique"
            Case = "Energy"
                SubTopicButtons(1).Text = "Kinetic"
                SubTopicButtons(2).Text = "Gravitational"
        End Select

        ' Add handlers
```

```
        AddHandler SubTopicButtons(1).Click, AddressOf btnSubTopic1_Click
        AddHandler SubTopicButtons(2).Click, AddressOf btnSubTopic2_Click
    End Sub

Private Sub btnPure_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
        ' Load topic buttons related to the pure module
        QuizPath.MathsModule = "Pure"

        ' Display next hierarchy of buttons
        LoadTopicButtons()
    End Sub
    Private Sub btnStatistics_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs)
        ' Load topic buttons related to the statistics module
        QuizPath.MathsModule = "Statistics"

        ' Display next hierarchy of buttons
        LoadTopicButtons()
    End Sub
    Private Sub btnMechanics_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs)
        ' Load topic buttons related to the mechanics module
        QuizPath.MathsModule = "Mechanics"

        ' Display next hierarchy of buttons
        LoadTopicButtons()
    End Sub

    Private Sub btnTopic1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs)
        ' Assign Topic variable in SelectedFolders structure
        Select Case QuizPath.MathsModule
            Case = "Pure"
                QuizPath.Topic = "Proof"
            Case = "Statistics"
                QuizPath.Topic = "Probability"
            Case = "Mechanics"
                QuizPath.Topic = "Collisions"
        End Select

        ' Display next hierarchy of buttons
        LoadSubTopicButtons()
    End Sub

    Private Sub btnTopic2_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs)
        ' Assign Topic variable in SelectedFolders structure
        Select Case QuizPath.MathsModule
            Case = "Pure"
                QuizPath.Topic = "Integration"
            Case = "Statistics"
                QuizPath.Topic = "Distributions"
            Case = "Mechanics"
                QuizPath.Topic = "Energy"
        End Select

        ' Display next hierarchy of buttons
        LoadSubTopicButtons()
    End Sub

    Private Sub btnSubTopic1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs)
```

```
        Select Case QuizPath.Topic
            Case = "Proof"
                SubTopicQuestions.AddRange(New String() {"INDUCTION1", "INDUCTION2"})
            Case = "Integration"
                SubTopicQuestions.AddRange(New String() {"INTEGRATING_E1"})
            Case = "Probability"
                SubTopicQuestions.AddRange(New String() {"VENNDIAGRAMS1"})
            Case = "Distributions"
                SubTopicQuestions.AddRange(New String() {"BINOMIAL1"})
            Case = "Collisions"
                SubTopicQuestions.AddRange(New String() {"ELASTICIN2D1",
"ELASTICIN2D2"})
            Case = "Energy"
                SubTopicQuestions.AddRange(New String() {"KINETIC1"})
        End Select
        DisplayQuiz()
    End Sub

    Private Sub btnSubTopic2_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs)
        Select Case QuizPath.Topic
            Case = "Proof"
                SubTopicQuestions.AddRange(New String() {"COUNTEREXAMPLE1",
"COUNTEREXAMPLE2"})
            Case = "Integration"
                SubTopicQuestions.AddRange(New String() {"INTEGRATING_LN1"})
            Case = "Probability"
                SubTopicQuestions.AddRange(New String() {"LAWSOFPROBABILITY1"})
            Case = "Distributions"
                SubTopicQuestions.AddRange(New String() {"NORMAL1"})
            Case = "Collisions"
                SubTopicQuestions.AddRange(New String() {"OBLIQUE1"})
            Case = "Energy"
                SubTopicQuestions.AddRange(New String() {"GRAVITATIONAL1"})
        End Select
        DisplayQuiz()
    End Sub

    Private Sub btnCollapse_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs)
        ' Reset controls
        DeleteControls()

        ' Reset SelectedFolders structure values
        QuizPath.MathsModule = ""
        QuizPath.Topic = ""

        ' Load controls back in
        LoadStartupElements()
        LoadModuleButtons()
    End Sub


   Private Sub DeleteControls()
        ' Delete all controls
        For i As Integer = Me.Controls.Count - 1 To 0 Step -1
            If TypeOf Me.Controls(i) Is Button Or TypeOf Me.Controls(i) Is TextBox Or
TypeOf Me.Controls(i) Is PictureBox Then
                Me.Controls.RemoveAt(i)
            End If
        Next
    End Sub
```

```
    Private Sub DisplayQuiz()
        ' Transfer selected question names to a list in frmRandomisedQuiz
        frmRandomisedQuiz.QuestionList.Clear()
        For Each item In SubTopicQuestions
            frmRandomisedQuiz.QuestionList.AddRange(New String() {item})
        Next
        ' Load frmRandomisedQuiz
        frmRandomisedQuiz.Show()
    End Sub
End Class
```

Code for ‘Teacher Homepage’ module

```
Option Strict On
Public Class frmTeacherHomepage
    Dim CreateClass As New frmCreateNewClass
    Dim ViewStudentProgress As New frmViewStudentProgress
    Dim ViewClassProgress As New frmViewClassProgress
    Private Sub Teacher_Homepage_Load(sender As Object, e As EventArgs) Handles
MyBase.Load
        Me.WindowState = FormWindowState.Maximized
        CreateClass.MdiParent = Me
        CreateClass.WindowState = CreateClass.WindowState.Maximized
        CreateClass.Show()
    End Sub

    Private Sub tlb_lbl_CreateNewClass_Click(sender As Object, e As EventArgs) Handles
tlb_lbl_CreateNewClass.Click
        MinimiseForms()
        CreateClass.MdiParent = Me
        CreateClass.WindowState = CreateClass.WindowState.Maximized
        CreateClass.Show()
    End Sub

    Private Sub tbl_lbl_ViewStudentProgress_Click(sender As Object, e As EventArgs)
Handles tbl_lbl_ViewStudentProgress.Click
        MinimiseForms()
        ViewStudentProgress.MdiParent = Me
        ViewStudentProgress.WindowState = ViewStudentProgress.WindowState.Maximized
        ViewStudentProgress.Show()
    End Sub

    Private Sub tbl_lbl_ViewClassProgress_Click(sender As Object, e As EventArgs)
Handles tbl_lbl_ViewClassProgress.Click
        MinimiseForms()
        ViewClassProgress.MdiParent = Me
        ViewClassProgress.WindowState = ViewClassProgress.WindowState.Maximized
        ViewClassProgress.Show()
    End Sub

    Private Sub MinimiseForms()
        ' Hide all open forms
        For Each form As Form In Me.MdiChildren
            form.WindowState = FormWindowState.Minimized
        Next
    End Sub
End Class
```

Code for ‘Create A Class’ module

```
Public Class frmCreateNewClass
    Structure AccountDetails
        <VBFixedString(20)> Public Username As String
        <VBFixedString(20)> Public Password As String
        <VBFixedString(20)> Public Forename As String
        <VBFixedString(20)> Public Surname As String
    End Structure

    Structure AddClass
        <VBFixedString(5)> Public ClassName As String
        <VBFixedString(3)> Public ClassSize As String
        <VBFixedString(150)> Public Students As String
    End Structure

    Structure SearchListBox
        Public SearchID As Integer
        Public FullStudentName As String
        Public DecryptedForename As String
        Public DecryptedSurname As String
        Public StudentAlreadyInList As Boolean
    End Structure

    Dim StudentInfo As AccountDetails
    Dim NewClassToAdd As AddClass
    Dim SearchStudents As SearchListBox

    Private Sub frmClassesOverview_Load(sender As Object, e As EventArgs) Handles
MyBase.Load
        ClearControls()
        GetStudents()
    End Sub

    Private Sub ClearControls()
        For Each control As Control In Me.Controls
            If TypeOf control Is TextBox Then
                control.Text = String.Empty
            End If
        Next
        lstStudents.Items.Clear()
    End Sub


Private Sub GetStudents()
        ' Retrieve student names from textfile
        Try
            Dim DecryptedForename As String
            Dim DecryptedSurname As String
            Dim i As Integer = 1
            FileOpen(1, "studentaccounts.txt", OpenMode.Random, , , Len(StudentInfo))
            While Not EOF(1)
                FileGet(1, StudentInfo, i)
                DecryptedForename = StudentInfo.Forename.Trim
                DecryptedSurname = StudentInfo.Surname.Trim
                ' Decrypt contents of file
                ASCIIDecryption(DecryptedForename)
                ASCIIDecryption(DecryptedSurname)
                ' Add name to listbox
```

```
                lstStudents.Items.Add(i.ToString + ".  " + DecryptedForename + " " +
DecryptedSurname)
                i += 1
            End While
            FileClose(1)
        Catch ex As Exception
            MessageBox.Show("An error occured whilst attempting to retrive your data.
Please try again.", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error)
        End Try
    End Sub

    Public Sub ASCIIDecryption(ByRef CiphertextToDecrypt As String)
        ' Take in encoded ASCII value and convert back to plaintext
        Dim EncryptionFactor As Integer = 3
        Dim plaintext As String = ""
        For Each letter As String In CiphertextToDecrypt
            plaintext += (Chr(Asc(letter) - EncryptionFactor))
        Next
        CiphertextToDecrypt = plaintext
    End Sub

    Private Sub btnAdd_Click(sender As Object, e As EventArgs) Handles btnAdd.Click
        ' Adds selected name from students list box to new class list box
        Dim StudentAlreadyInList As Boolean = False
        If lstStudents.SelectedIndex > -1 Then
            ' Validation to ensure name isn't already added
            For n As Integer = 0 To lstNewClass.Items.Count - 1
                If lstNewClass.Items(n) = lstStudents.SelectedItem.Remove(0, 4) Then
                    StudentAlreadyInList = True
                End If
            Next
            If Not StudentAlreadyInList Then
                Dim StudentName As String = lstStudents.SelectedItem
                lstNewClass.Items.Add(StudentName.Remove(0, 4))
            End If
        End If
        lstStudents.SelectedIndex = -1
    End Sub

    Private Sub btnRemove_Click(sender As Object, e As EventArgs) Handles
btnRemove.Click
        ' Remove selected name in students list box from new class list box
        Dim StudentAlreadyInList As Boolean = False
        If lstStudents.SelectedIndex > -1 Then
            ' Validation to ensure name to remove is present
            For n As Integer = 0 To lstNewClass.Items.Count - 1
                If lstNewClass.Items(n) = lstStudents.SelectedItem.Remove(0, 4) Then
                    StudentAlreadyInList = True
                End If
            Next
            If StudentAlreadyInList Then
                lstNewClass.Items.Remove(lstStudents.SelectedItem.Remove(0, 4))
            End If
        End If
        lstStudents.SelectedIndex = -1
    End Sub
Private Sub btnSearch_Click(sender As Object, e As EventArgs) Handles btnSearch.Click
        SearchStudents.StudentAlreadyInList = False

        ' Use id number of student to search for their name
        Try
```

```
            SearchStudents.SearchID = InputBox("Please enter the ID of the student you
wish to search for")
            If SearchStudents.SearchID < 1 Or SearchStudents.SearchID >
lstStudents.Items.Count Then
                MessageBox.Show("SearchID is out of range", "Error",
MessageBoxButtons.OK, MessageBoxIcon.Error)
            End If

            ' Search for student with the input id
            FileOpen(2, "studentaccounts.txt", OpenMode.Random, , , Len(StudentInfo))
            FileGet(2, StudentInfo, SearchStudents.SearchID)
            SearchStudents.DecryptedForename = StudentInfo.Forename.Trim
            SearchStudents.DecryptedSurname = StudentInfo.Surname.Trim
            ASCIIDecryption(SearchStudents.DecryptedForename)
            ASCIIDecryption(SearchStudents.DecryptedSurname)
            SearchStudents.FullStudentName = SearchStudents.DecryptedForename + " " +
SearchStudents.DecryptedSurname
            FileClose(2)

            For n As Integer = 0 To lstNewClass.Items.Count - 1
                If lstNewClass.Items(n) = SearchStudents.FullStudentName Then
                    SearchStudents.StudentAlreadyInList = True
                End If
            Next
            If Not SearchStudents.StudentAlreadyInList Then
                lstNewClass.Items.Add(SearchStudents.FullStudentName)
            End If
            lstStudents.SelectedIndex = -1
        Catch ex As Exception
            MessageBox.Show("An error has occured whilst attempting to locate the
student at the given id", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error)
        End Try
    End Sub

    Private Sub TextBox_Validate(sender As Object, e As EventArgs) Handles
txtClassName.TextChanged
        ' Allow user to create class after classname has been entered
        If Not String.IsNullOrWhiteSpace(txtClassName.Text) Then page.Enabled = True
    End Sub

    Private Sub btnCreateClass_Click(sender As Object, e As EventArgs) Handles
btnCreateClass.Click
        Dim NextRecord As Integer
        Dim Valid As Boolean = CheckInputValidity()

        ' Convert number of students to be added into string
        Dim strClassSize = CStr(lstNewClass.Items.Count)
        AddTrailingBackslashes(strClassSize)

        If Valid Then
            Try
                ' Retrieve values from form
                NewClassToAdd.ClassName = txtClassName.Text
                NewClassToAdd.ClassSize = strClassSize
                NewClassToAdd.Students = CombinelstStudentItemsToString()

                ' Write data to file
                FileOpen(1, "Classes.txt", OpenMode.Random, , , Len(NewClassToAdd))
                NextRecord = (LOF(1) / Len(NewClassToAdd)) + 1
                With NewClassToAdd
                    .ClassName = NewClassToAdd.ClassName
                    .ClassSize = NewClassToAdd.ClassSize
```

```
                    .Students = NewClassToAdd.Students
                End With
                FilePut(1, NewClassToAdd, NextRecord)
                FileClose(1)
                MessageBox.Show("Class " & NewClassToAdd.ClassName & " has been
sucsessfully created")
            Catch ex As Exception
                MessageBox.Show("An error has occured whilst attempting to create a
new class", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error)
            End Try
        End If
    End Sub

    Private Function CombinelstStudentItemsToString()
        ' Combine student names in lstNewClass into a single string
        Dim Students = String.Empty
        For Each item In lstNewClass.Items
            If Students <> String.Empty Then
                Students &= ", "
            End If
            Students &= item
        Next
        Return Students
    End Function


Private Sub AddTrailingBackslashes(ByRef strNum As String)
        ' Ensure that number string is exactly 3 characters long
        Select Case strNum.Length
            Case = 1
                strNum = strNum + "//"
            Case = 2
                strNum = strNum + "/"
            Case > 3
                strNum = strNum.Substring(0, 3)
            Case Else
        End Select
    End Sub

    Private Function CheckInputValidity()
        ' Validate that classname starts with 2 numbers and a backslash, and that at
least one student has been selected
        Dim Valid As Boolean = False
        Dim Numbers As String = "1234567890"
        If txtClassName.Text.Length = 5 Then
            If Numbers.Contains(txtClassName.Text.Substring(0, 1)) And
Numbers.Contains(txtClassName.Text.Substring(1, 1)) And txtClassName.Text.Substring(2,
1) = "/" Then
                If lstNewClass.Items.Count >= 1 Then
                    Valid = True
                Else
                    MessageBox.Show("Error: Could not create the class. At least 1
student must be added to the new class.", "Error", MessageBoxButtons.OK,
MessageBoxIcon.Error)
                End If
            Else
                MessageBox.Show("Error: Could not create the class. Classname must
begin with 2 numbers representing the year group, followed by a backslash.", "Error",
MessageBoxButtons.OK, MessageBoxIcon.Error)
            End If
        Else
```

```
                MessageBox.Show("Error: Could not create the class. Classname must be
exactly 5 characters long.", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error)
        End If
        Return Valid
    End Function
```

Code for ‘Student Overview’ module

```
Imports System.IO
Public Class frmViewStudentProgress
    Structure AccountDetails
        <VBFixedString(20)> Public Username As String
        <VBFixedString(20)> Public Password As String
        <VBFixedString(20)> Public Forename As String
        <VBFixedString(20)> Public Surname As String
    End Structure

    Structure SearchListBox
        Public SearchID As Integer
        Public FullStudentName As String
        Public DecryptedForename As String
        Public DecryptedSurname As String
        Public StudentAlreadyInList As Boolean
    End Structure

    Structure TestResults
        <VBFixedString(12)> Dim TotalScore As String

        <VBFixedString(12)> Dim PureScore As String
        <VBFixedString(12)> Dim StatisticsScore As String
        <VBFixedString(12)> Dim MechanicsScore As String

        <VBFixedString(12)> Dim ProofScore As String
        <VBFixedString(12)> Dim IntegrationScore As String

        <VBFixedString(12)> Dim ProbabilityScore As String
        <VBFixedString(12)> Dim DistributionsScore As String

        <VBFixedString(12)> Dim CollisionsScore As String
        <VBFixedString(12)> Dim EnergyScore As String

        <VBFixedString(12)> Dim InductionScore As String
        <VBFixedString(12)> Dim CounterexampleScore As String

        <VBFixedString(12)> Dim Integrating_E_Score As String
        <VBFixedString(12)> Dim Integrating_LN_Score As String

        <VBFixedString(12)> Dim VennDiagramsScore As String
        <VBFixedString(12)> Dim LawsOfProbabilityScore As String

        <VBFixedString(12)> Dim BinomialScore As String
        <VBFixedString(12)> Dim NormalScore As String

        <VBFixedString(12)> Dim ElasticIn2DScore As String
        <VBFixedString(12)> Dim ObliqueScore As String

        <VBFixedString(12)> Dim KinecticScore As String
        <VBFixedString(12)> Dim GravitationalScore As String

        <VBFixedString(6)> Dim NumOfAttempts As String
```

```
    End Structure

    Dim StudentInfo As AccountDetails
    Dim SearchStudents As SearchListBox
    Dim RetrieveResults As New TestResults

    Dim SR As StreamReader
    Dim Filename As String

    Private Sub frmViewStudentProgress_Load(sender As Object, e As EventArgs) Handles
MyBase.Load
        Me.WindowState = FormWindowState.Maximized
        ' Load student names into listbox
        GetStudents()
    End Sub

    Private Sub btnSelect_Click(sender As Object, e As EventArgs) Handles
btnSelect.Click
        ResetTestResultsStructure(RetrieveResults)
        ' Extract filename from selected index
        Filename = lstStudents.SelectedItem.Remove(0, 4)
        Filename = RemoveWhitespace(Filename)
        Filename = Filename
        Try
            Dim i As Integer = 1
            FileOpen(1, Filename & ".txt", OpenMode.Random, , , Len(RetrieveResults))
            While Not EOF(1)
                FileGet(1, RetrieveResults, i)
            End While
            FileClose(1)
        Catch ex As Exception
            MessageBox.Show("An error occured whilst attempting to retrieve student
data. Please try again.", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error)
        End Try
    End Sub

Private Sub GetStudents()
        ' Retrieve student names from textfile
        Try
            Dim i As Integer = 1
            FileOpen(1, "studentaccounts.txt", OpenMode.Random, , , Len(StudentInfo))
            While Not EOF(1)
                FileGet(1, StudentInfo, i)
                SearchStudents.DecryptedForename = StudentInfo.Forename.Trim
                SearchStudents.DecryptedSurname = StudentInfo.Surname.Trim
                ' Decrypt contents of file
                frmCreateNewClass.ASCIIDecryption(SearchStudents.DecryptedForename)
                frmCreateNewClass.ASCIIDecryption(SearchStudents.DecryptedSurname)
                ' Add name to listbox
                lstStudents.Items.Add(i.ToString + ".  " +
SearchStudents.DecryptedForename + " " + SearchStudents.DecryptedSurname)
                i += 1
            End While
            FileClose(1)
        Catch ex As Exception
            MessageBox.Show("An error occured whilst attempting to retrive your data.
Please try again.", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error)
        End Try
    End Sub

    Function RemoveWhitespace(fullString As String) As String
        ' Remove all whitespaces from a string
```

```
        Return New String(fullString.Where(Function(x) Not
Char.IsWhiteSpace(x)).ToArray())
    End Function

    Private Sub ResetTestResultsStructure(ByRef ResultsStructure As TestResults)
        ' Resets the values of a TestResults structure to default
        ResultsStructure.TotalScore = "/00000/00000"
        ResultsStructure.PureScore = "/00000/00000"
        ResultsStructure.StatisticsScore = "/00000/00000"
        ResultsStructure.MechanicsScore = "/00000/00000"
        ResultsStructure.ProofScore = "/00000/00000"
        ResultsStructure.IntegrationScore = "/00000/00000"
        ResultsStructure.ProbabilityScore = "/00000/00000"
        ResultsStructure.DistributionsScore = "/00000/00000"
        ResultsStructure.CollisionsScore = "/00000/00000"
        ResultsStructure.EnergyScore = "/00000/00000"
        ResultsStructure.InductionScore = "/00000/00000"
        ResultsStructure.CounterexampleScore = "/00000/00000"
        ResultsStructure.Integrating_E_Score = "/00000/00000"
        ResultsStructure.Integrating_LN_Score = "/00000/00000"
        ResultsStructure.VennDiagramsScore = "/00000/00000"
        ResultsStructure.LawsOfProbabilityScore = "/00000/00000"
        ResultsStructure.BinomialScore = "/00000/00000"
        ResultsStructure.NormalScore = "/00000/00000"
        ResultsStructure.ElasticIn2DScore = "/00000/00000"
        ResultsStructure.ObliqueScore = "/00000/00000"
        ResultsStructure.KinecticScore = "/00000/00000"
        ResultsStructure.GravitationalScore = "/00000/00000"
        ResultsStructure.NumOfAttempts = "/00000"
    End Sub


    Private Sub drawline(ByVal ModuleType As String, ByVal StudentName As String)
        Dim ResultsLine As String
        Dim formgraphics As Drawing.Graphics
        Dim Y1, Y2 As Integer
        Dim lblXAxis(10) As Label
        Dim lblYAxis(10) As Label
        Dim loc(10) As Point
        Randomize()
        formgraphics = Me.CreateGraphics

        ' Draw axis; y-axis(start, end), x-axis(start, end)
        formgraphics.DrawLine(Pens.Black, 725, 10, 725, 330)
        formgraphics.DrawLine(Pens.Black, 725, 330, 1250, 330)

        ' Label x axis in intervals of multiples of i; i determines number of labels
        For i = 0 To 5
            ' Set interval between labels on x axis
            loc(i).X = i * 100 + 725
            ' Set vertical position of x axis label
            loc(i).Y = 340
            lblXAxis(i) = New Label
            lblXAxis(i).Location = loc(i)
            lblXAxis(i).Text = i + 1
            Me.Controls.Add(lblXAxis(i))
        Next i

        ' Change co-ordinates to label the y axis
        For i = 0 To 5
            loc(i).Y = i * 62.5 + 10
            loc(i).X = 625
```

```
            lblYAxis(i) = New Label
            lblYAxis(i).Location = loc(i)
            lblYAxis(i).Text = Math.Abs(20 * (5 - i))
            Me.Controls.Add(lblYAxis(i))
        Next i

        SR = New StreamReader(StudentName & "Attempts.txt", True)
        ResultsLine = SR.ReadLine()
        Y1 = PercentageOfYAxis(ResultsLine, ModuleType)
        For x = 725 To 1175 Step 100
            ResultsLine = SR.ReadLine()
            If Not IsNothing(ResultsLine) Then
                Y2 = PercentageOfYAxis(ResultsLine, ModuleType)
                formgraphics.DrawLine(Pens.Red, x, Y1, x + 100, Y2)
                Y1 = Y2
            End If
        Next
        SR.Close()
    End Sub

    Private Function PercentageOfYAxis(ByVal ResultsLine As String, ByVal ModuleType
As String)
        ' Calculate position of a point to be plotted on the y axis
        Dim Y1 As Integer
        Select Case ModuleType
            Case = "P"
                Y1 = CInt((1 - (ResultsLine.Substring(0, 3) / 100)) * 320) + 10
            Case = "S"
                Y1 = CInt((1 - (ResultsLine.Substring(3, 3) / 100)) * 320) + 10
            Case = "M"
                Y1 = CInt((1 - (ResultsLine.Substring(6, 3) / 100)) * 320) + 10
        End Select
        Return Y1
    End Function

    Private Sub btnGenGraph_Click(sender As Object, e As EventArgs) Handles
btnGenGraph.Click
        ' Display user scores for the selected module; P = puremaths, S = statistics &
M = mechanics
        If cmbSelectTopic.Text <> Nothing Then
            ' Delete current graph
            Me.Refresh()
            Select Case cmbSelectModule.SelectedIndex
                Case = 0
                    drawline("P", Filename)
                Case = 1
                    drawline("S", Filename)
                Case = 2
                    drawline("M", Filename)
            End Select
        Else
            MessageBox.Show("Please select a module and topic to continue", "Error",
MessageBoxButtons.OK, MessageBoxIcon.Error)
        End If

        lstStudentData.Items.Clear()
        PopulateBarChart(cmbSelectTopic.Text)
        PopulateListBox()

        lblShowGraphX.Text = "Showing graph for: " & cmbSelectModule.Text
        lblShowChartX.Text = "Showing chart for: " & cmbSelectTopic.Text
        ' Might need to refresh this instead
```

```
        cmbSelectModule.SelectedIndex = -1
        cmbSelectTopic.SelectedIndex = -1
    End Sub


Private Sub PopulateBarChart(ByVal Topic As String)
        ' Reset bar chart
        chtDisplayTopic.Series.Clear()
        chtDisplayTopic.Series.Add("Sub Topic")

        ' Display average score for each topic on a bar chart
        Dim avg1 As Integer
        Dim avg2 As Integer
        Dim SubTopic1 As String
        Dim SubTopic2 As String

        ' Identify sub-topics to display on bar chart
        Select Case Topic
            Case = "Proof"
                SubTopic1 = "Induction"
                SubTopic2 = "Counter-example"
                avg1 = CalculateAverage(RetrieveResults.InductionScore)
                avg2 = CalculateAverage(RetrieveResults.CounterexampleScore)
            Case = "Integration"
                SubTopic1 = "Integrating e"
                SubTopic2 = "Integrating ln"
                avg1 = CalculateAverage(RetrieveResults.Integrating_E_Score)
                avg2 = CalculateAverage(RetrieveResults.Integrating_LN_Score)
            Case = "Probability"
                SubTopic1 = "Venn Diagrams"
                SubTopic2 = "Laws Of Probability"
                avg1 = CalculateAverage(RetrieveResults.VennDiagramsScore)
                avg2 = CalculateAverage(RetrieveResults.LawsOfProbabilityScore)
            Case = "Distributions"
                SubTopic1 = "Binomial"
                SubTopic2 = "Normal"
                avg1 = CalculateAverage(RetrieveResults.BinomialScore)
                avg2 = CalculateAverage(RetrieveResults.NormalScore)
            Case = "Collisions"
                SubTopic1 = "Elastic in 2D"
                SubTopic2 = "Oblique"
                avg1 = CalculateAverage(RetrieveResults.ElasticIn2DScore)
                avg2 = CalculateAverage(RetrieveResults.ObliqueScore)
            Case = "Energy"
                SubTopic1 = "Kinetic"
                SubTopic2 = "Gravitational"
                avg1 = CalculateAverage(RetrieveResults.KinecticScore)
                avg2 = CalculateAverage(RetrieveResults.GravitationalScore)
        End Select

        ' Plot bars
        Me.chtDisplayTopic.Series("Sub Topic").Points.AddXY(SubTopic1, avg1)
        Me.chtDisplayTopic.Series("Sub Topic").Points.AddXY(SubTopic2, avg2)
    End Sub

    Private Sub PopulateListBox()
        ' Populate lstStudentData with student's scores in a variety of categories
        lstStudentData.Items.Add("Overall score: " &
CalculateAverage(RetrieveResults.TotalScore) & "%")
        lstStudentData.Items.Add("Pure Maths score: " &
CalculateAverage(RetrieveResults.PureScore) & "%")
```

```
        lstStudentData.Items.Add("Statistics score: " &
CalculateAverage(RetrieveResults.StatisticsScore) & "%")
        lstStudentData.Items.Add("Mechanics score: " &
CalculateAverage(RetrieveResults.MechanicsScore) & "%")
    End Sub

    Private Function CalculateAverage(ByRef strToAverage As String)
        ' Calculate average of a variable in the TestResults structure
        Dim avg As Integer
        ' Ensuring average <= 100%
        Try
            If CInt(strToAverage.Substring(1, 5)) <= CInt(strToAverage.Substring(7,
5)) And strToAverage.Length() = 12 Then
                avg = (CInt(strToAverage.Substring(1, 5)) /
CInt(strToAverage.Substring(7, 5))) * 100
            Else
                avg = 0
            End If
        Catch ex As Exception
            MessageBox.Show("Please parse a valid scores string into the subroutine",
"Error", MessageBoxButtons.OK, MessageBoxIcon.Error)
            avg = 0
        End Try
        Return avg
    End Function

    Private Function GetResults(ByRef strToAverage As String)
        ' Calculate average of a variable in the TestResults structure
        Dim avg As Integer
        ' Ensuring average <= 100%
        Try
            If CInt(strToAverage.Substring(1, 5)) <= CInt(strToAverage.Substring(7,
5)) And strToAverage.Length() = 12 Then
                avg = (CInt(strToAverage.Substring(1, 5)) /
CInt(strToAverage.Substring(7, 5))) * 100
            Else
                avg = 0
            End If
        Catch ex As Exception
            MessageBox.Show("Please parse a valid scores string into the subroutine",
"Error", MessageBoxButtons.OK, MessageBoxIcon.Error)
            avg = 0
        End Try
        Return avg
    End Function




Private Sub btnSelectModule_Click(sender As Object, e As EventArgs) Handles
btnSelectModule.Click
        ' Select values for topics to populate the combo box based on the user's
selection in the module combobox
        cmbSelectTopic.Items.Clear()
        Select Case cmbSelectModule.SelectedIndex
            Case = 0
                cmbSelectTopic.Items.Add("Proof")
                cmbSelectTopic.Items.Add("Integration")
            Case = 1
                cmbSelectTopic.Items.Add("Probability")
                cmbSelectTopic.Items.Add("Distributions")
            Case = 2
```

```
                cmbSelectTopic.Items.Add("Collisions")
                cmbSelectTopic.Items.Add("Energy")
            Case Else
                MessageBox.Show("Please select a module to continue", "Error",
MessageBoxButtons.OK, MessageBoxIcon.Error)
        End Select
    End Sub
```

Code for 'Class Overview' module

```
Imports System.Text.RegularExpressions
Public Class frmViewClassProgress
    Structure AddClass
        <VBFixedString(5)> Public ClassName As String
        <VBFixedString(3)> Public ClassSize As String
        <VBFixedString(150)> Public Students As String
    End Structure
    Dim PopulateClass As AddClass

    Public Class StudentInformation
        Public StudentName As String
        Public StudentScores As String
        Public StudentAttempts As String
    End Class
    Dim sInfo As List(Of StudentInformation) = New List(Of StudentInformation)()

    Private Sub frmViewClassProgress_Load(sender As Object, e As EventArgs) Handles
MyBase.Load
        Me.WindowState = Me.WindowState.Maximized
        Me.Visible = True

        ' Display names of classes in cmbSelectClass
        GetClassnames()
    End Sub

    Private Sub GetClassnames()
        ' Retrieve class names from textfile
        Try
            Dim i As Integer = 1
            FileOpen(1, "classes.txt", OpenMode.Random, , , Len(PopulateClass))
            While Not EOF(1)
                FileGet(1, PopulateClass, i)
                ' Add name to listbox
                cmbSelectClass.Items.Add(PopulateClass.ClassName)
                i += 1
            End While
            FileClose(1)
        Catch ex As Exception
            MessageBox.Show("An error occured whilst attempting to retrive your data.
Please try again.", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error)
        End Try
    End Sub

Private Sub btnSelectClass_Click(sender As Object, e As EventArgs) Handles
btnSelectClass.Click
        ' Retrieve student names into an instance of the student scores structure

        ' Fetch current class
        RetrieveClassStruc(cmbSelectClass.SelectedItem)
```

```
        ' Add class name to label
        lblStudents.Text = "Students in class: " & cmbSelectClass.SelectedItem

        ' Clear contents of sInfo
        sInfo.Clear()

        ' Fetch student names from file
        PopulateStudentList()

        ' Display student names
        PopulatelstStudentNames()

        ' Populate bar chart with average scores for each student in the class
        PopulateBarChart()

        ' Populate line graph with average scores for each attempt across the class
        PopulateLineGraph()
    End Sub

  Private Sub RetrieveClassStruc(ByVal ClassName As String)
        ' Fetch instance of AddClass structure that contains the selected class name
        Try
            Dim i As Integer = 1
            FileOpen(1, "classes.txt", OpenMode.Random, , , Len(PopulateClass))
            Do Until EOF(1) Or PopulateClass.ClassName = ClassName
                FileGet(1, PopulateClass, i)
                i += 1
            Loop
            FileClose(1)
        Catch ex As Exception
            MessageBox.Show("An error occured whilst attempting to retrive your data.
Please try again.", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error)
        End Try
    End Sub

    Private Sub PopulateStudentList()
        ' Split PopulateClass.Students by the commma
        Dim ArrayOfStudentNames As String() = PopulateClass.Students.Split(New Char()
{","c})
        Dim StudentName As String
        ' Add each student to lstStudentNames
        For Each StudentName In ArrayOfStudentNames
            ' Collect together data stored about the student's results from .txt files
            GatherStudentInformation(StudentName)
        Next
    End Sub

    Private Sub GatherStudentInformation(ByVal StudentName As String)
        ' Add student name to sInfo list
        Dim StudentScores As String =
My.Computer.FileSystem.ReadAllText(RemoveWhitespace(StudentName) & ".txt")
        Dim StudentAttempts As String =
My.Computer.FileSystem.ReadAllText(RemoveWhitespace(StudentName) & "Attempts.txt")
        sInfo.Add(New StudentInformation() With {.StudentName = StudentName,
.StudentScores = StudentScores, .StudentAttempts = StudentAttempts})
    End Sub

    Function RemoveWhitespace(fullString As String) As String
        Return New String(fullString.Where(Function(x) Not
Char.IsWhiteSpace(x)).ToArray())
    End Function
```

```
Private Sub PopulatelstStudentNames()
        ' Clear lstStudentNames
        lstStudentNames.Items.Clear()
        For i = 0 To sInfo.Count - 1
            lstStudentNames.Items.Add(sInfo(i).StudentName.Trim())
        Next
    End Sub

    Private Sub PopulateBarChart()
        ' Reset bar chart
        chtStudentAverages.Series.Clear()
        chtStudentAverages.Series.Add("Students")

        ' Declare array to hold student scores
        Dim StudentScores(sInfo.Count - 1) As Integer

        For i = 0 To sInfo.Count - 1
            StudentScores(i) =
CalculateStudentScoresAvg(sInfo(i).StudentScores.Substring(0, 12))
        Next

        ' Quicksort student scores
        InitiateQuicksort(StudentScores)

        ' Plot average score for each student on bar chart in descending order
        For i = sInfo.Count - 1 To 0 Step -1

Me.chtStudentAverages.Series("Students").Points.AddXY(sInfo(i).StudentName,
StudentScores(i))
        Next
    End Sub

    Private Function CalculateStudentScoresAvg(ByRef strToAverage As String)
        ' Calculate average of the StudentScores variable in the StudentInformation
structure
        Dim avg As Integer
        ' Ensuring average <= 100%
        Try
            If CInt(strToAverage.Substring(1, 5)) <= CInt(strToAverage.Substring(7,
5)) And strToAverage.Length() = 12 Then
                avg = (CInt(strToAverage.Substring(1, 5)) /
CInt(strToAverage.Substring(7, 5))) * 100
            Else
                avg = 0
            End If
        Catch ex As Exception
            MessageBox.Show("Please parse a valid scores string into the subroutine",
"Error", MessageBoxButtons.OK, MessageBoxIcon.Error)
            avg = 0
        End Try
        Return avg
    End Function

Private Function CalculateStudentAttemptsAvg(ByVal ResultsLine As String)
        ' Return the average of a single line from sInfo(n).StudentAttempts
        Dim Results(3) As Integer
        For i = 0 To 2
            Results(i) = CInt(ResultsLine.Substring((3 * i), 3))
        Next
        Return Math.Round(Results.Average())
    End Function
```

```
    Sub InitiateQuicksort(ByRef StudentScores() As Integer)
        Dim First As Integer = 0
        Dim Last As Integer = sInfo.Count - 1

        Dim str As String = ""

        Call Quicksort(First, Last, StudentScores)

        For i = 0 To sInfo.Count - 1
            str += StudentScores(i) & " "
        Next
    End Sub

    Sub Quicksort(ByVal First As Integer, ByVal Last As Integer, ByRef StudentScores()
As Integer)
        ' Perform a quicksort on the array of student scores
        Dim j As Integer
        Dim k As Integer
        If First < Last Then
            j = First
            k = Last + 1
            Do
                Do
                    j += 1
                Loop Until StudentScores(j) >= StudentScores(First) Or j = Last

                Do
                    k -= 1
                Loop Until StudentScores(k) <= StudentScores(First)

                If j < k Then
                    Call Swap(StudentScores(j), StudentScores(k))
                End If
            Loop Until j >= k
            Call Swap(StudentScores(First), StudentScores(k))
            Call Quicksort(First, k - 1, StudentScores)
            Call Quicksort(k + 1, Last, StudentScores)
        End If
    End Sub

    Private Sub Swap(ByRef A As String, ByRef B As String)
        Dim Temp As String
        Temp = A
        A = B
        B = Temp
    End Sub

Private Sub PopulateLineGraph()
        ' Populate line graph with combined average of each student's average score
        Dim arrOfStudentAttempts As String()
        Dim avgOfStudentAttempts(5) As Integer
        Dim ClassAvg(sInfo.Count - 1) As Integer

        ' Calculate average score for each student
        For i = 0 To sInfo.Count - 1
            ' Split each line of an instance of StudentAttempts into an array using
Regex
            arrOfStudentAttempts = Regex.Split(sInfo(i).StudentAttempts,
Environment.NewLine)

            ' Declare new integer array to hold average of each individual line
```

```
            ReDim Preserve avgOfStudentAttempts(arrOfStudentAttempts.Length - 1)

            For j = 0 To arrOfStudentAttempts.Length - 1
                avgOfStudentAttempts(j) =
CalculateStudentAttemptsAvg(arrOfStudentAttempts(j).Trim())
            Next

            ClassAvg(i) = avgOfStudentAttempts.Average()
        Next

        For k = 0 To ClassAvg.Length - 1
            Me.chtAvgOfAttempts.Series("Average Score Per Attempt").Points.AddXY(k +
1, ClassAvg(k))
        Next
    End Sub

    Private Sub btnGoToGraph2_Click(sender As Object, e As EventArgs) Handles
btnGoToGraph2.Click
        Panel1.Visible = False
        Panel2.Visible = True
    End Sub

    Private Sub btnGoToGraph1_Click(sender As Object, e As EventArgs) Handles
btnGoToGraph1.Click
        Panel2.Visible = False
        Panel1.Visible = True
    End Sub
End Class
```

3.3 Program Demonstration

The following screenshots give an overview of the *Login* and *Create An Account* modules, along with the two different ways that the user can interact with the *Mathematics Testing Program* – either as a teacher or as a student.

3.3.1 Program Demonstration – Creating an account

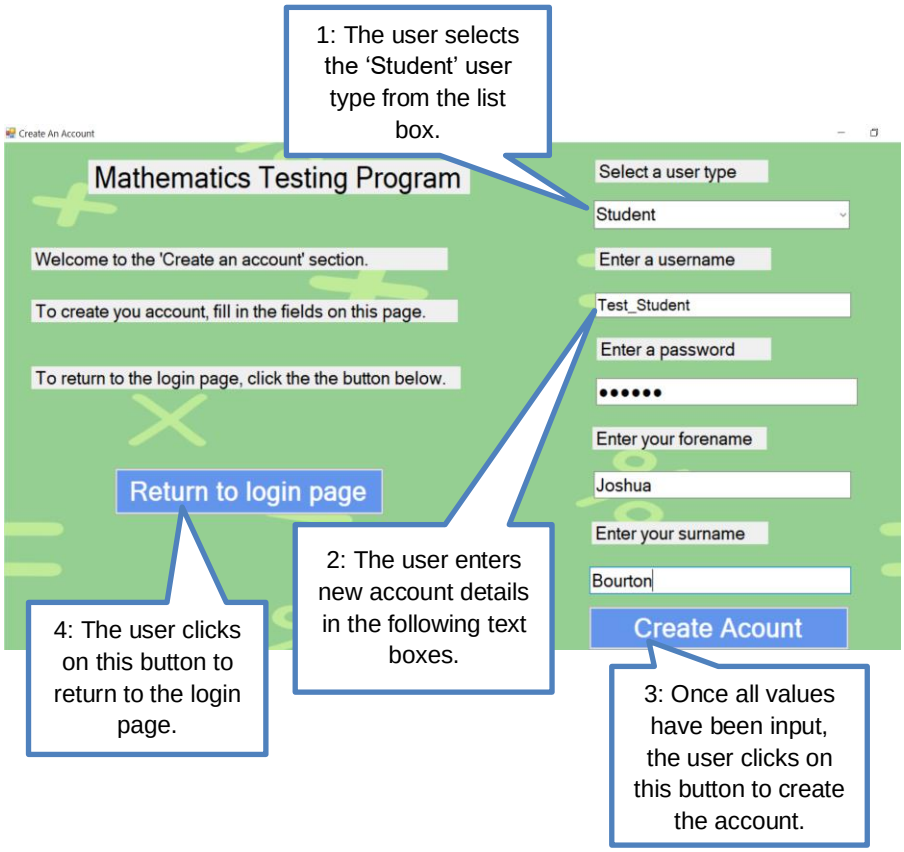
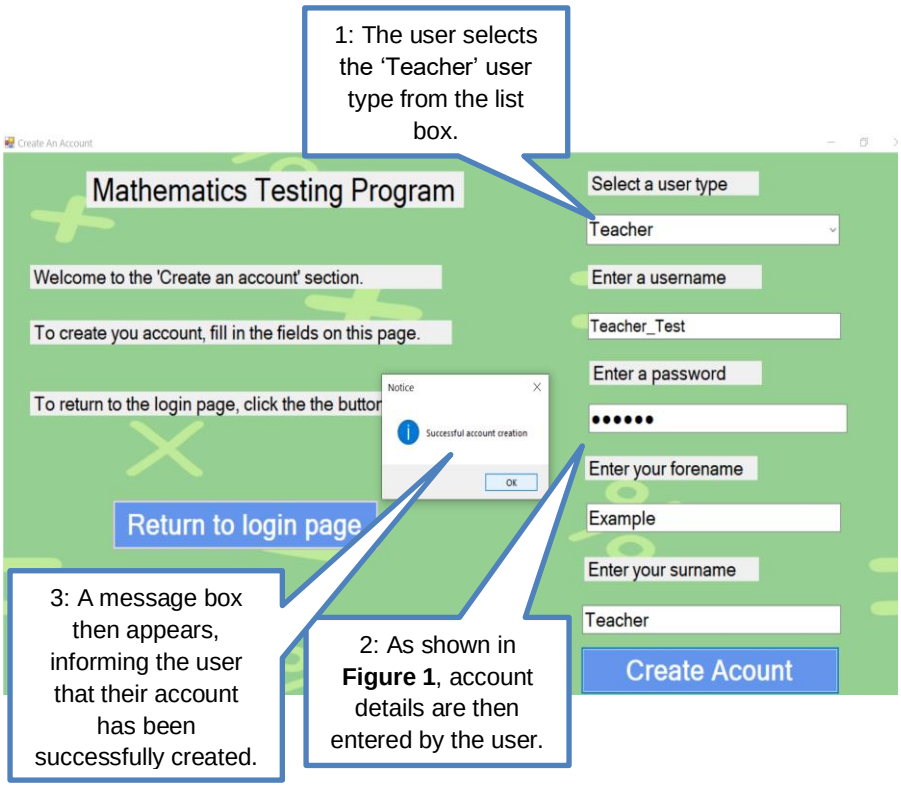


Figure 1: 'Create An Account' form – Student user type.
The following screenshot shows the *Create An Account* form in use when a student attempts to sign up to the program.

Figure 2: 'Create An Account' form – Teacher user type.
The following screenshot shows the *Create An Account* form in use when a teacher attempts to sign up to the program.



3.3.2 Program Demonstration – Logging into the program

Figure 3: 'Login' form – Student user type.
The following screenshot shows the login form in use when a student is attempting to login to the program using the account details created in the *Create An Account* section.

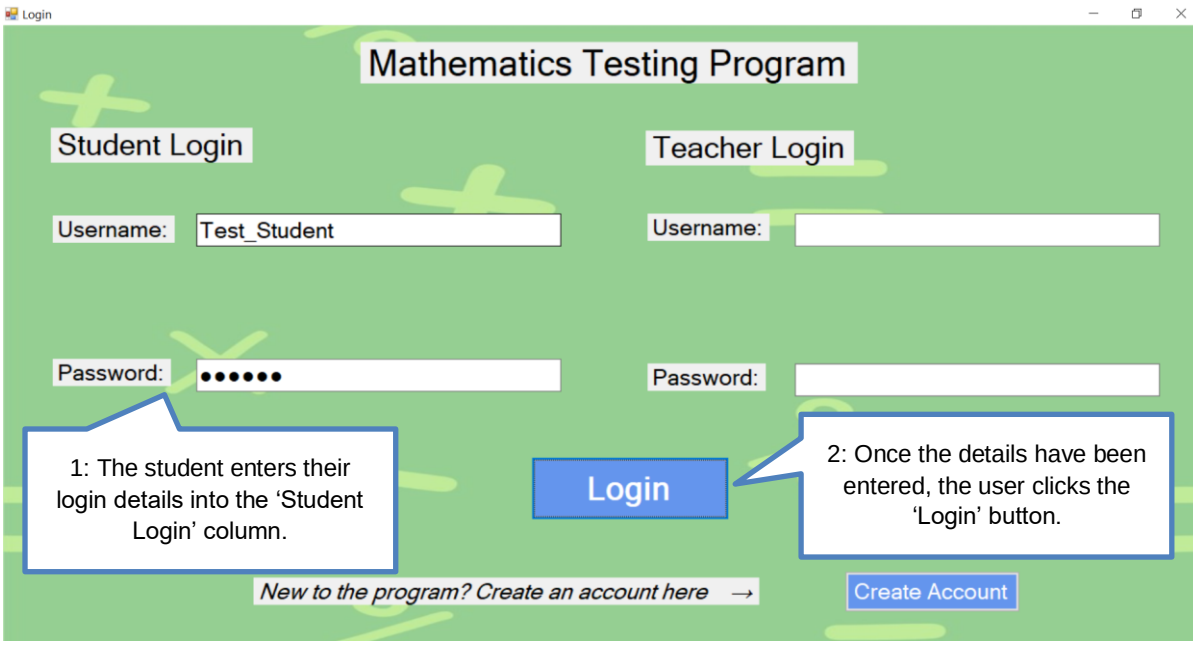
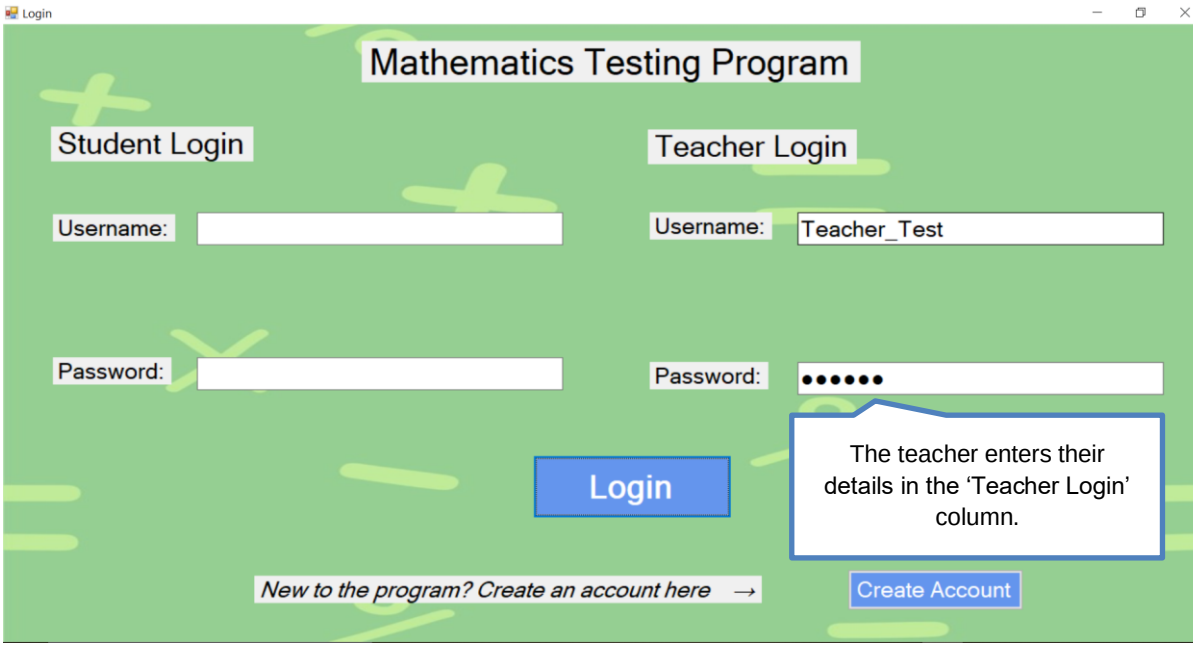


Figure 4: 'Login' form – Teacher user type.
The following screenshot shows the login form in use when a teacher is attempting to login to the program using the account details created in the *Create An Account* section.



3.3.3 Program Demonstration – Randomised Quiz (*Student* user type)

Figure 5: Randomised Quiz – *Fill In The Missing Blank* style of question.
The following screenshot shows the *Randomised Quiz* section of the program when a *Fill In The Missing Blank* style of question is presented to the student.

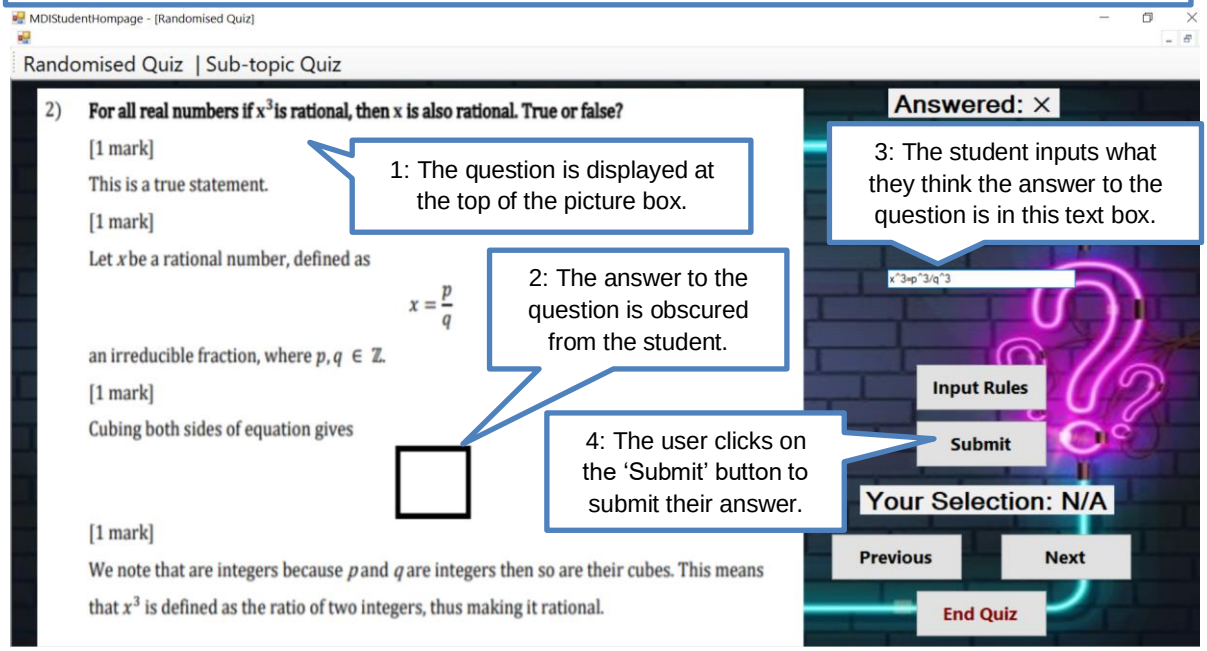
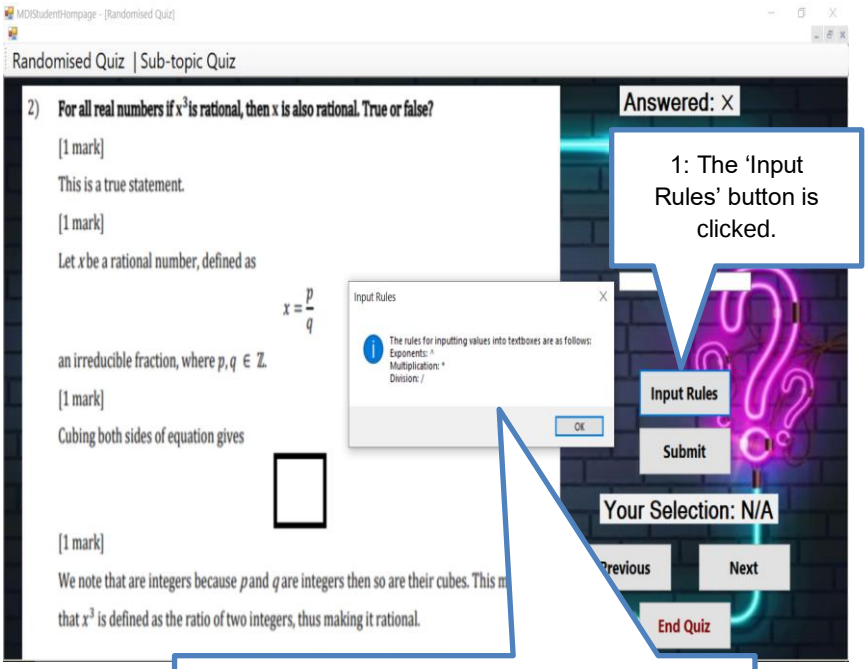


Figure 6: Viewing the input rules
This screenshot depicts how the "Input Rules" button works to give guidance to the student on how to correctly format their answer for a *Fill In The Missing Blank* style of question.



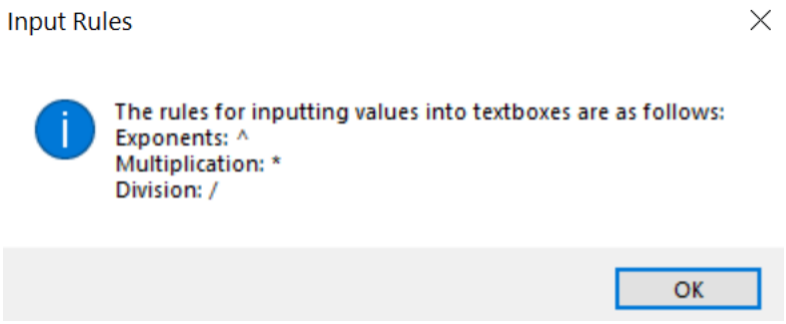


Figure 7: A closeup of the message box shown when the 'Input Rules' button (shown in **Figure 6**) is clicked).

Figure 8: Randomised Quiz – *Multiple Choice* style of question.
The following screenshot shows the *Randomised Quiz* section of the program when a *Multiple Choice* style of question is presented to the student.

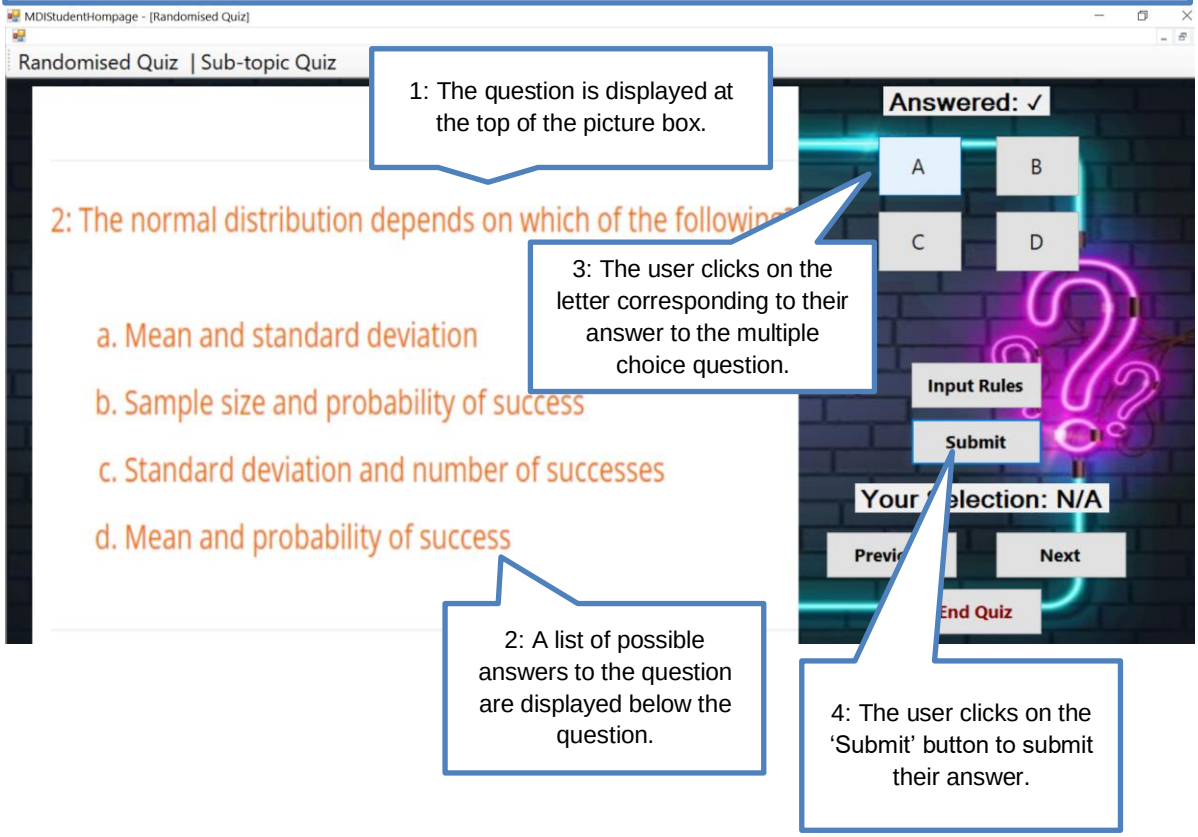
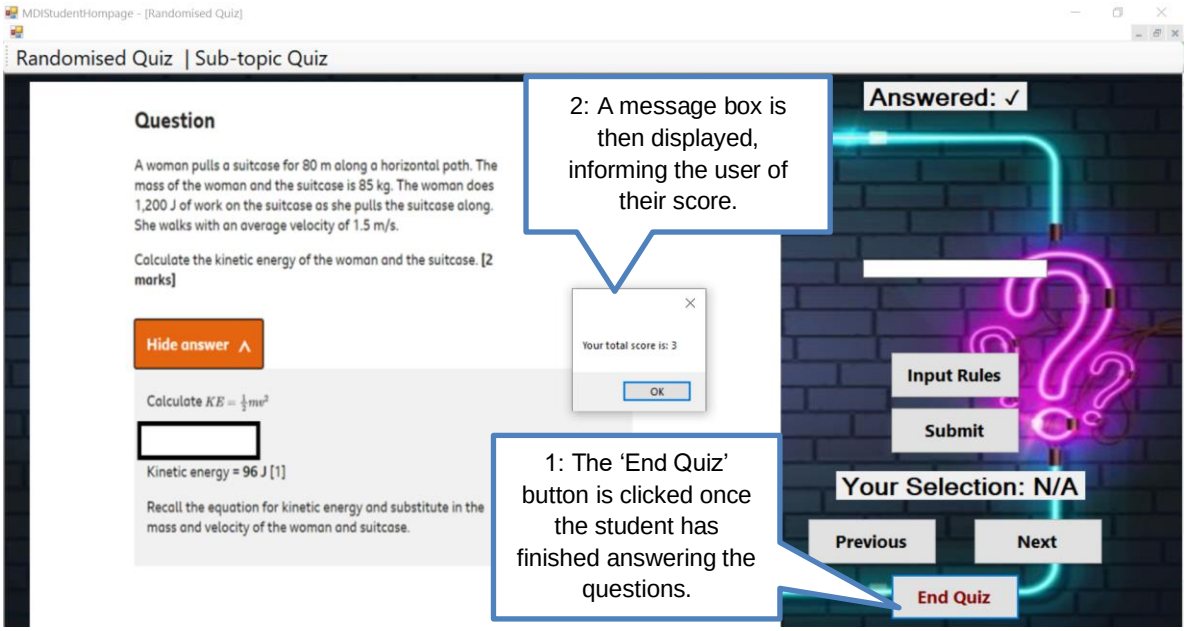
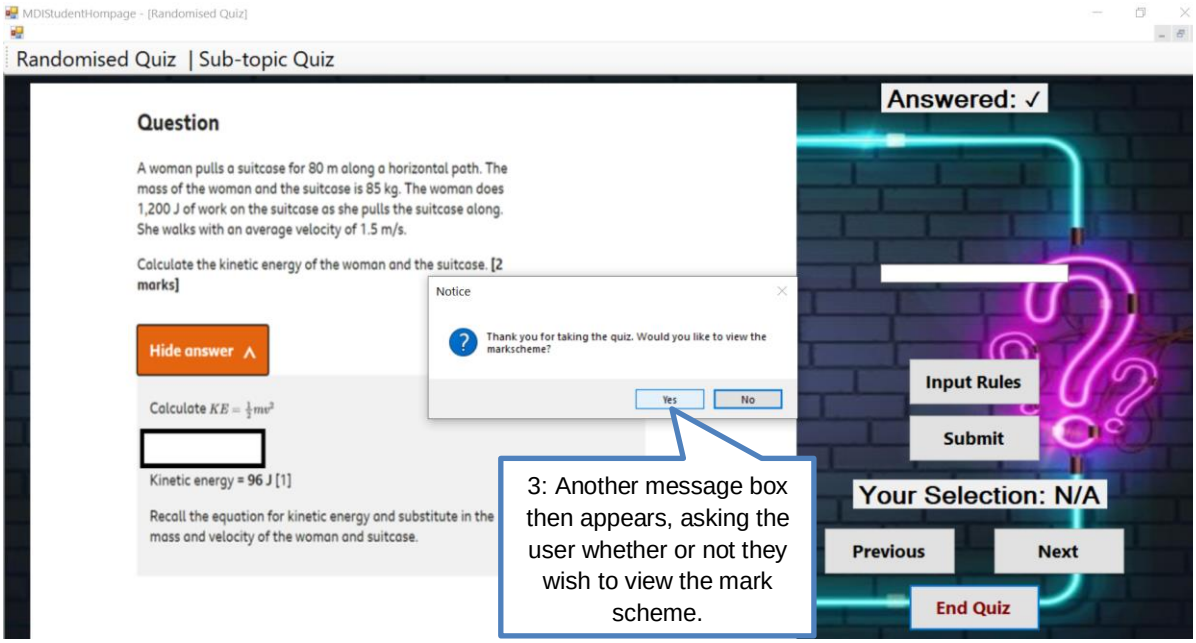
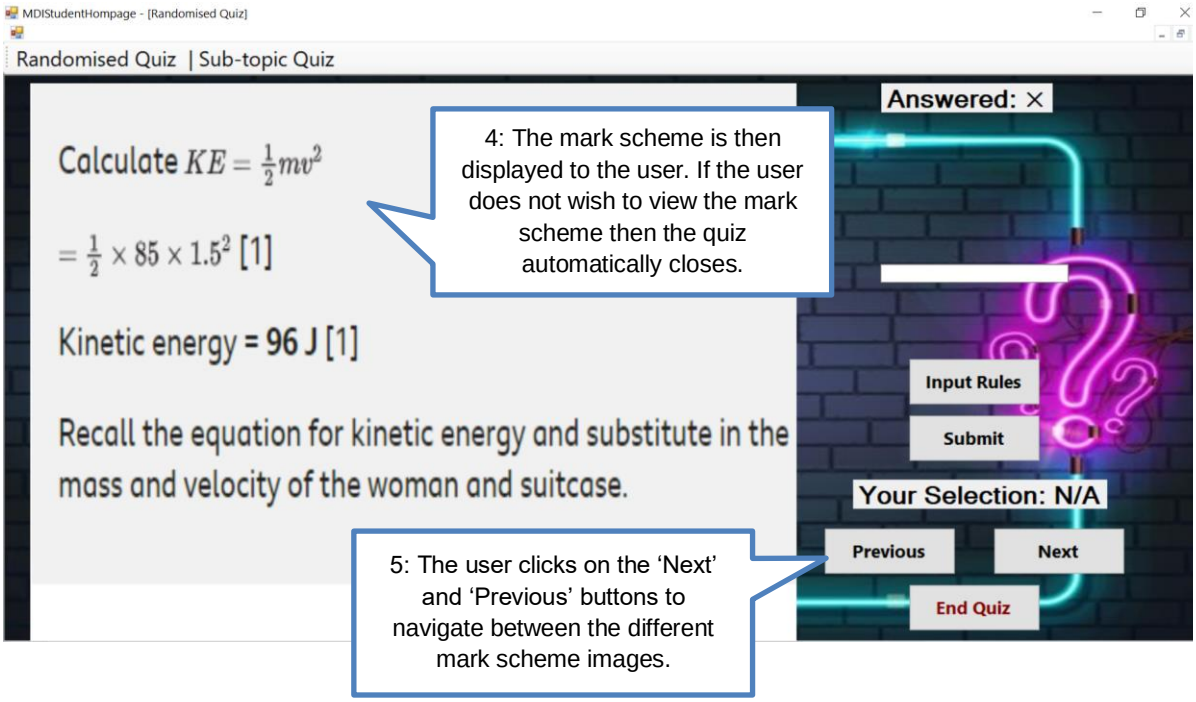


Figure 9: Randomised Quiz – Viewing the mark scheme
The following screenshot set depicts how the student is given a choice as to whether they wish to view the mark scheme for the questions they attempted during the quiz.

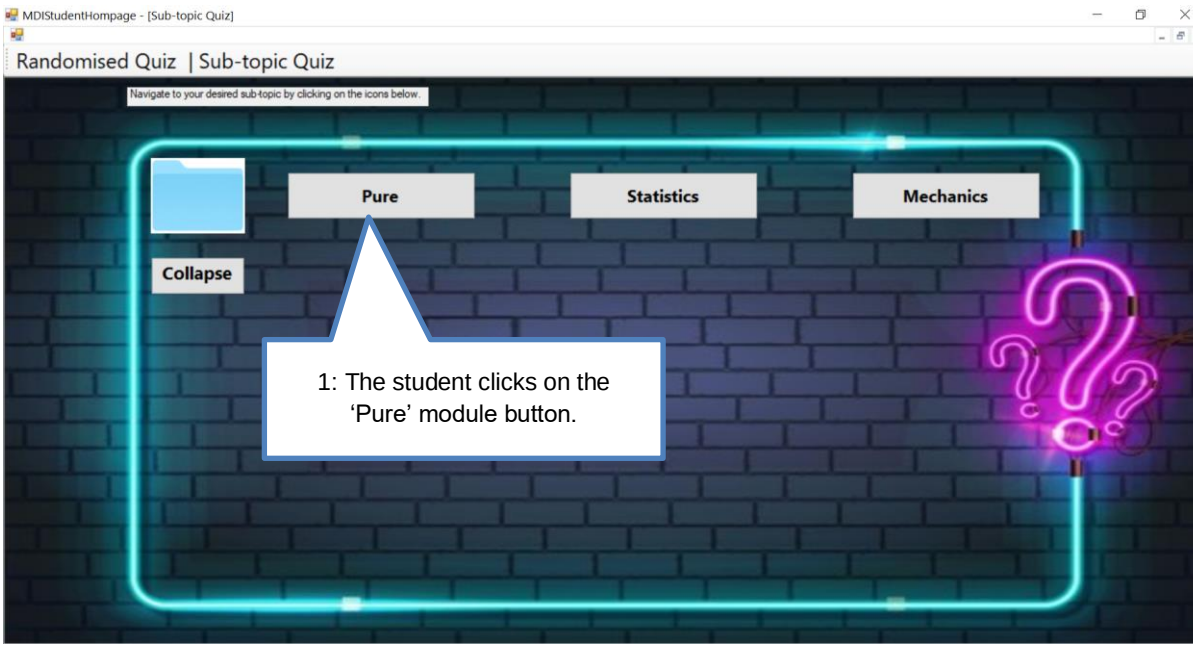


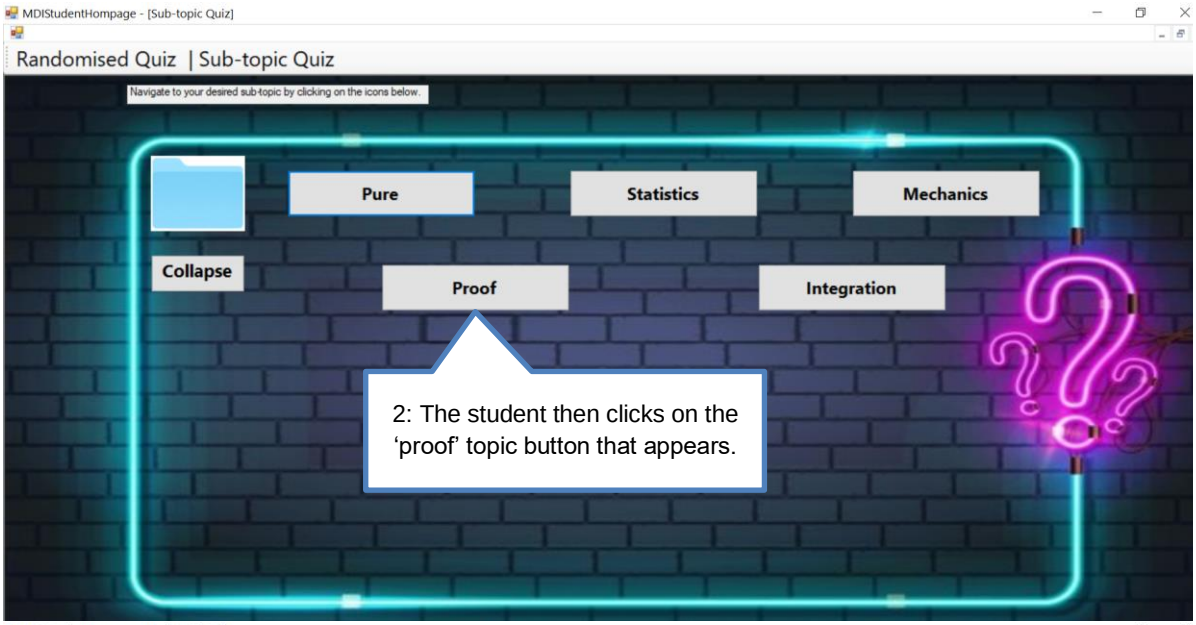


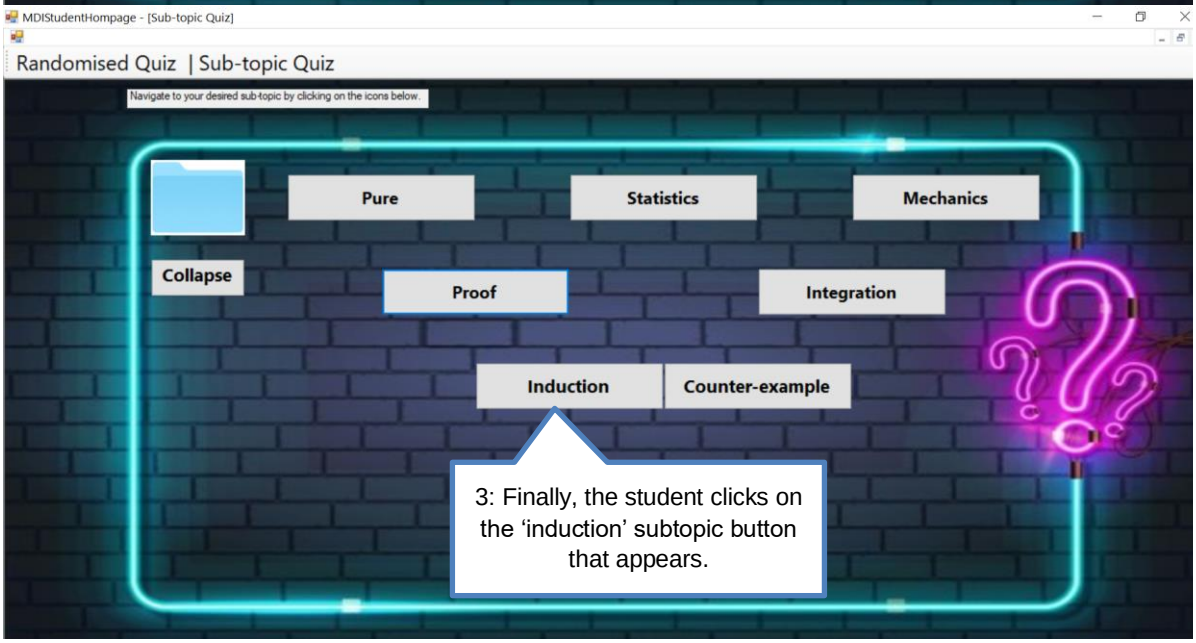


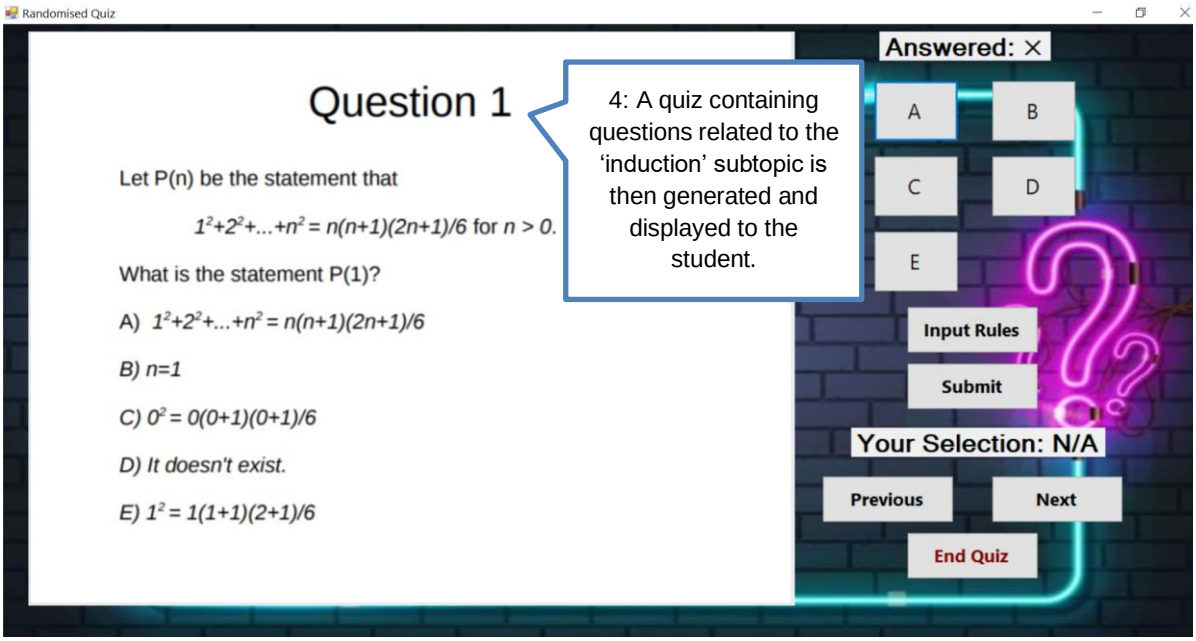
3.3.4 Program Demonstration – Select A Subtopic (*Student* user type)

Figure 10: The following screenshots demonstrate how a student user would bring up a set of questions related to the 'induction' subtopic from within the *Select A Subtopic* section of the program.




MDIStudentHompage - [Sub-topic Quiz]
Randomised Quiz | Sub-topic Quiz
Navigate to your desired sub-topic by clicking on the icons below.
Pure
Statistics
Mechanics
Collapse
Proof
Integration
2: The student then clicks on the 'proof' topic button that appears.


MDIStudentHompage - [Sub-topic Quiz]
Randomised Quiz | Sub-topic Quiz
Navigate to your desired sub-topic by clicking on the icons below.
Pure
Statistics
Mechanics
Collapse
Proof
Integration
Induction
Counter-example
3: Finally, the student clicks on the 'induction' subtopic button that appears.



3.3.5 Program Demonstration – Creating A Class (*Teacher* user type)

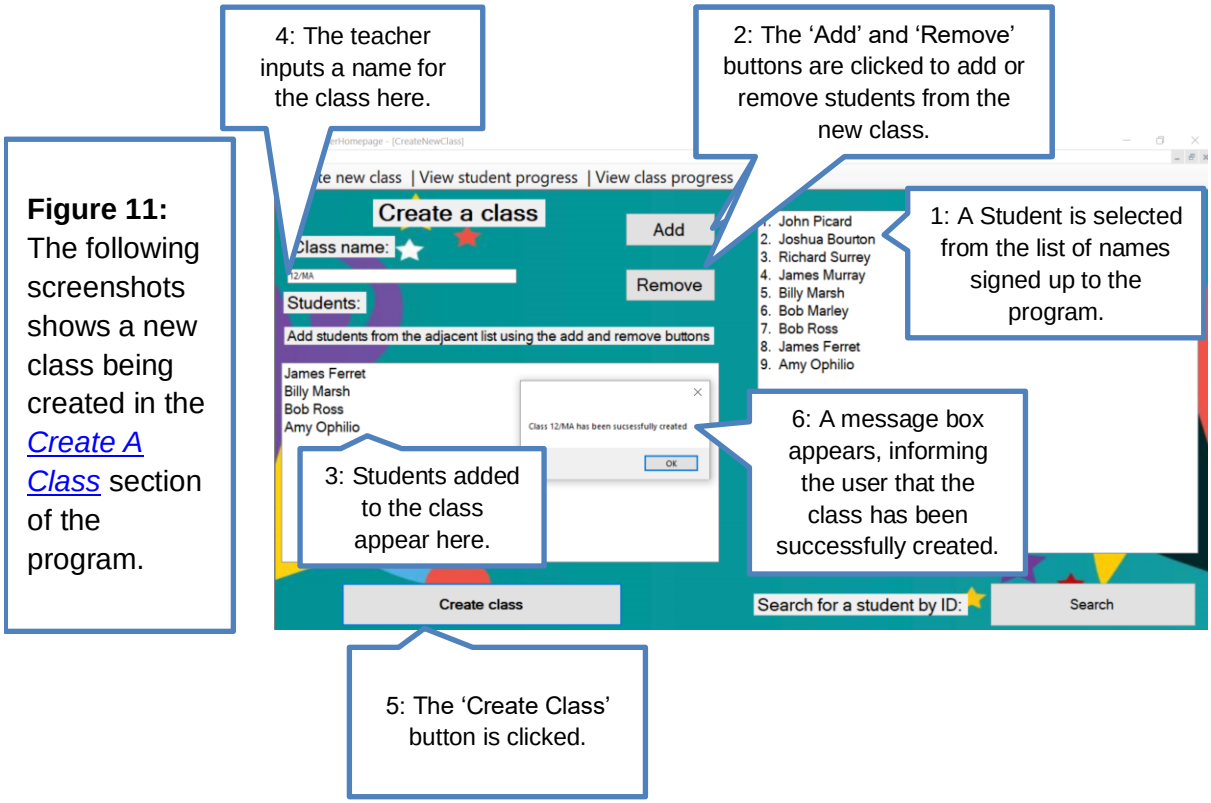


Figure 11: The following screenshots shows a new class being created in the *Create A Class* section of the program.

3.3.6 Program Demonstration – Student Overview (*Teacher* user type)

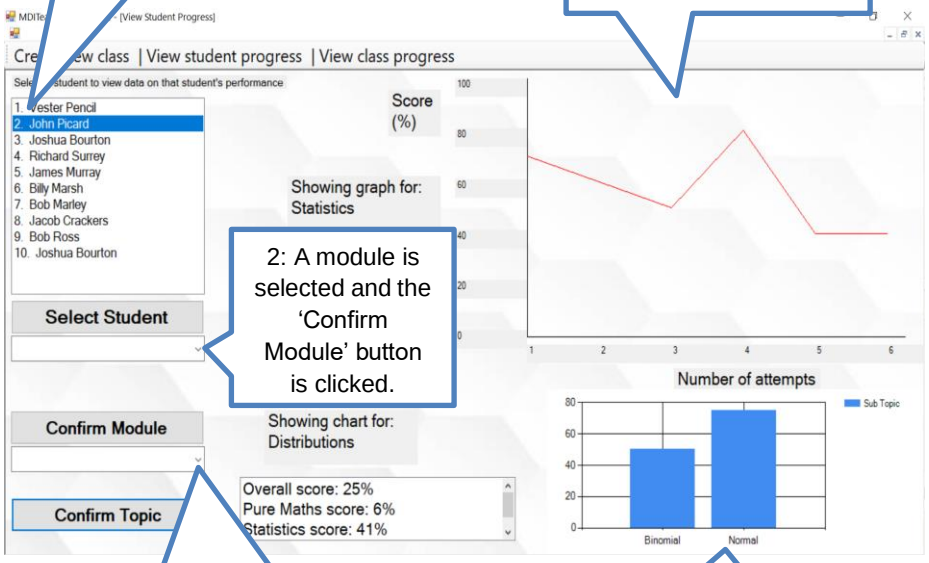
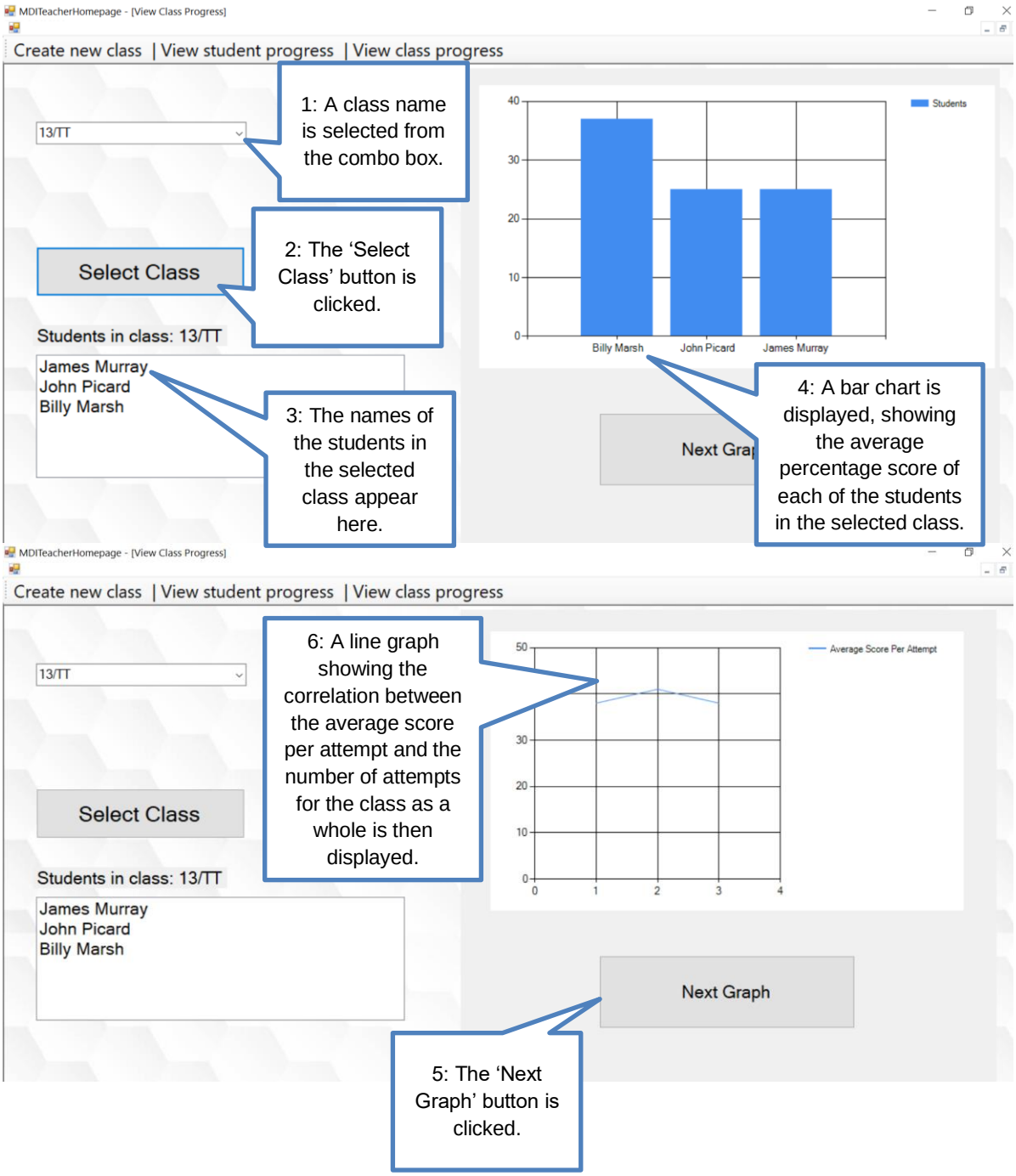


Figure 12: The following screenshots demonstrate how a teacher would bring up statistics collected by the program on an individual student.

3.3.7 Program Demonstration – Class Overview (*Teacher* user type)

Figure 13: The following screenshots show how a teacher would bring up statistics gathered by the program related to the overall performance of their created classes.



TESTING

4.1 Data Sets

Note: In the following datasets, fields that are left blank represent an absence of input into that particular field.

Data Set 1 – Student Registration and Login

The following dataset lists the dummy accounts that will be used to test the '**Create An Account**' and '**Login**' section of the program.

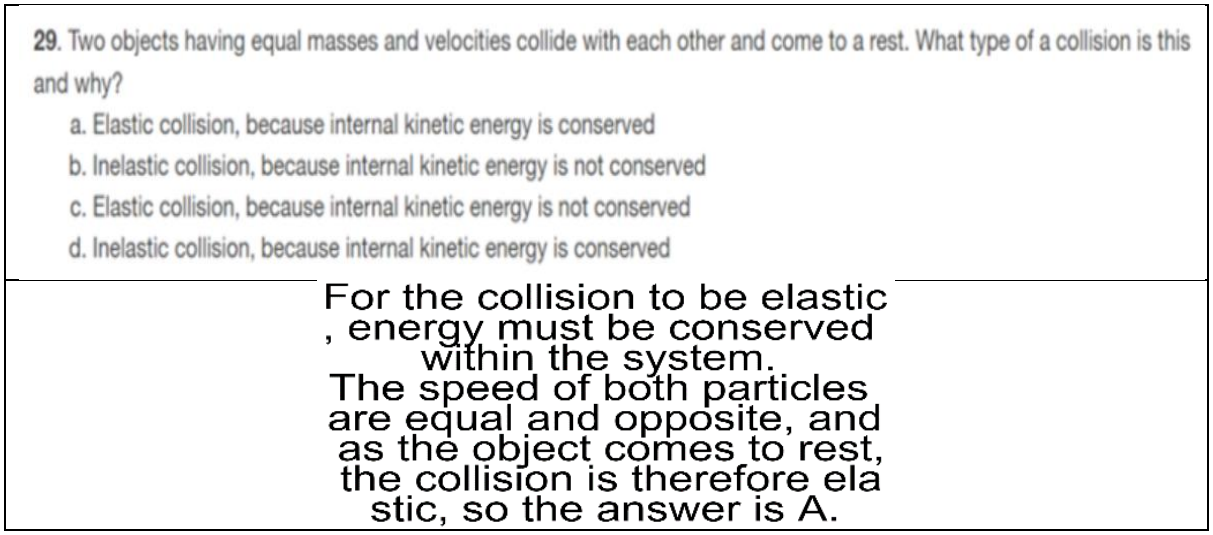
Number	Forename	Surname	Username	Password
1	John	Picard	LongJohn	Pirate-ys
2	Joshua	Bourton	JackRyan	SkyDive09
3	Richard	Surrey	12Terabytes	07USB
4	James	Murray	PandaConservation	SaveThePandas
5	Billy	Marsh	CalculatorWizz	Magic3Math
6	Bob	Ross	1Mandalorian2	ThisIsTheWay
7	James	Ferret	ExampleStudent	12345
8	Anna	Grizfolk	ThisIsAReallyLongName	12345
9	Amy	Ophilio	MathWorks	arithMatic9
10	Winter	Reeves	DarkMathmagician	ThisIsAReallyLong Password

Data Set 2 – Example Question 1 (multiple choice)

The following question will be used to test that the program can determine whether a student's answer to a *multiple choice* style question is correct or incorrect. The first image shows the question, and the second image shows the mark scheme answer for this particular question.

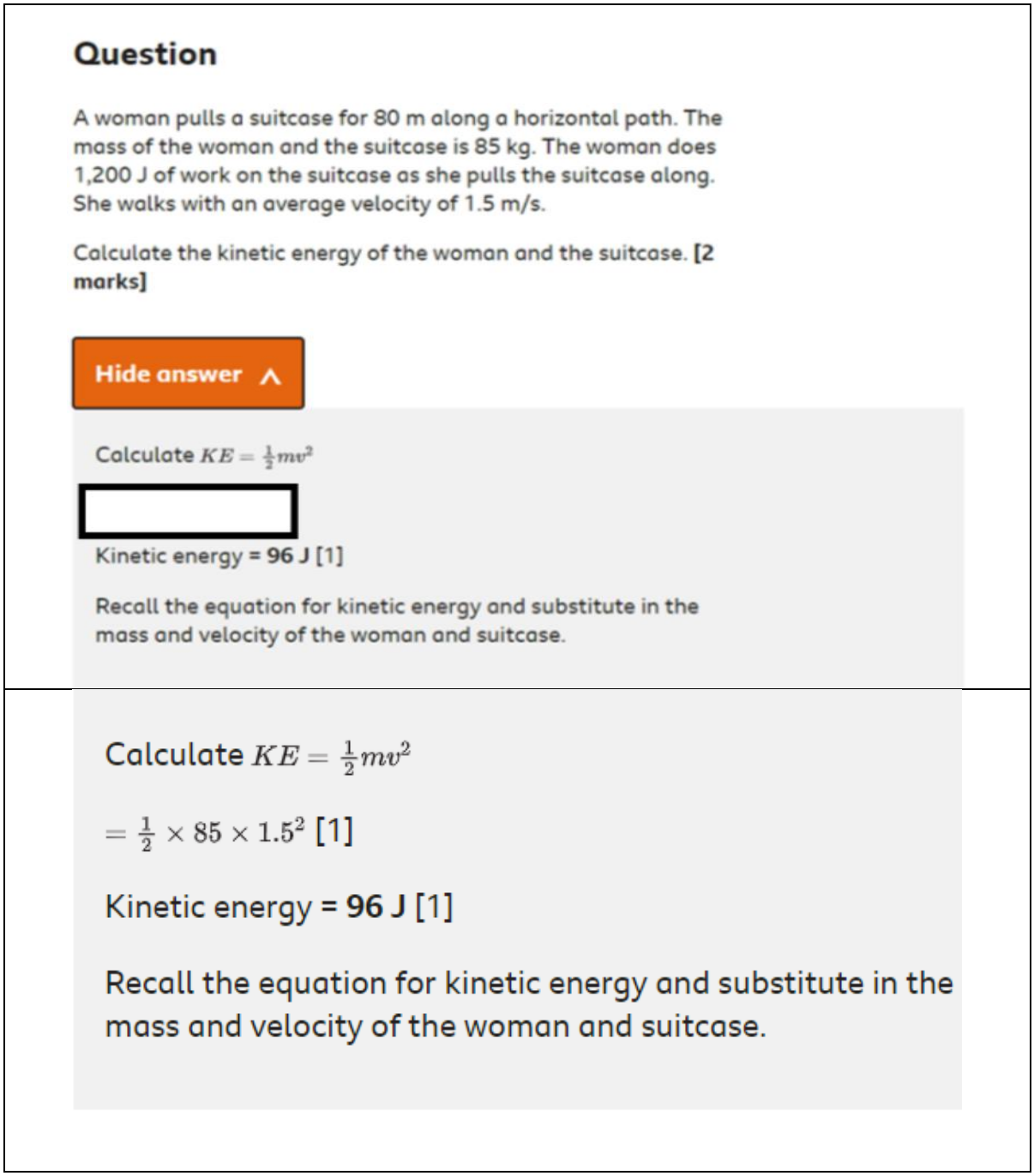
Input 1	Input 2
A	D

The question and answer for this question, respectively.



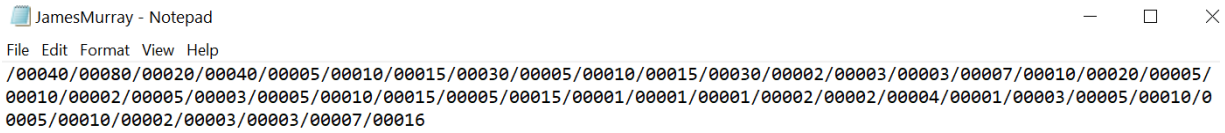
Data Set 3 – Example Question 2 (fill in the missing blank)
The following question will be used to test that the program can determine whether a student's answer to a *fill in the missing blank* style question is correct or incorrect. The first image shows the question, and the second image shows the mark scheme answer for this particular question.

Input 1	Input 2
1/2*85*1.5^2	Incorrect_Answer



Data Set 4 – Results text file

The following dataset shows the contents of the text file that holds the *quiz results* data. This data is used in generating the statistics and bar chart for the example student “James Murray”.



Data Set 5 – Attempts text file

The following dataset shows the contents of the text file that holds the *attempts* data. This data is used in generating a line graph of percentage score against number of attempts for the example student “James Murray”.

4.2 Table of tests

In order to test that the program functions as intended and can withstand invalid user inputs, multiple tests were carried out across all the different areas of the program. These tests are described in the table below.

Test Number	Description	Test Data	Expected Output	Result
"Create An Account" Section				
1	Clicking the "Create An Account" button in the when no values are input into the field.	"Create An Account" button click. Erroneous	Error message displayed. Account is not created.	As expected
2	Clicking the "Create An Account" button when valid values for creating a **teacher account** are input into all the fields.	"ExampleTeacher", "12345", "John", "Nixon" Typical	Message appears telling the user that account creation is successful, and the user is returned to the login form.	As expected – **See SS1**
3	Clicking the "Create An Account" button when valid values for creating a **student account** are input into all the fields.	"ExampleStudent", "12345", "James", "Ferret" Typical	Message appears telling the user that account creation is successful, and the user is returned to the login form.	Error message appears stating that the inputs are over the 20 character limit - **See SS2** **RETEST NEEDED**
4 ***Retest for Test #3***	As above	As above	As above	As expected - **See SS3**
5	Attempting to sign up to the program with a username that is already in use.	Clicking the "Create An Account" button Erroneous	Message appears telling user that the username is already in use. Account is not created.	As expected - **See SS4**
6	Attempting to sign up to the program with a username that is over 20 characters long.	"ThisIsAReallyLongName" Erroneous	Message appears telling user that username is too long.	As expected - **See SS5**

"Login" Section				
7	Clicking the "Login" button when no values are input into any of the fields.	N/A Erroneous	Message appears asking the user to input values into the field and the user is not logged in.	As expected
8	Clicking the "Login" button when a valid value is entered into the **student username** input box, but none of the other sections.	Student, "OnlyUsername", N/A Erroneous	The user is told to input a value into the student password field.	As expected
9	Clicking the "Login" button when a valid value is input into the **teacher password** section, but no other sections.	N/A, "12345" Erroneous	The user is told to input a value into the teacher username field.	As expected
10	Entering valid login details for the **student** user type into the **student login** section.	"ExampleStudent", "12345" Typical	The user is logged in as a student user type.	As expected
11	Entering valid login details for the **teacher** user type into the **teacher login** section.	"ExampleTeacher", "12345" Typical	The user is logged in as a teacher user type.	As expected
12	Entering valid **student** login details into the **teacher login** section.	"ExampleStudent", "12345" Typical	An error message occurs, stating that the login credentials are incorrect.	As expected
13	Entering valid **teacher** login details into the **student login** section.	"ExampleTeacher", "12345" Typical	An error message occurs, stating that the login credentials are incorrect.	As expected
"Randomised Quiz" Section				
14	Testing that the program registers a <u>correct</u> answer to a *multiple choice* style question.	**Data Set 2, Input 1** ("A") Typical	The question is marked as correct, and the user scores 1/5.	As expected - **See SS6**

15	Testing that the program registers an incorrect answer to a *multiple choice* style question.	**Data Set 2, Input 2** ("D") Erroneous	The question is marked as incorrect, and the user scores 0/5.	As expected - **See SS7**
16	Testing that the program registers a correct answer to the *fill in the missing blank* style question.	**Data Set 3, Input 1** ("1/2*85*1.5^2") Typical	The question is marked as correct, and the user scores 1/5.	As expected - **See SS8**
17	Testing that the program registers an incorrect answer to the *fill in the missing blank* style question.	**Data Set 3, Input 2** ("Incorrect_Answer") Erroneous	The question is marked as incorrect, and the user scores 0/5.	As expected - **See SS9**
18	Ensuring that the "Next" button works as intended when on Question 1.	Clicking the "Next" button when on Question 1. Typical	Question 2 is shown.	As expected
19	Ensuring that the "Next" button works as intended when on Question 5.	Clicking the "Next" button when on Question 5. Typical	Cycles back through to Question 1.	As expected
20	Ensuring that the "Previous" button works as intended when on Question 5.	Clicking the "Previous" button when on Question 5. Typical	Question 4 is shown.	As expected
21	Ensuring that the "Previous" button works as when on Question 1.	Clicking the "Previous" button when on Question 1. Typical	Cycles back through to Question 5.	As expected
22	Ensuring that clicking on the "Submit" button works with "fill in the missing blank" style of question.	Clicking on the "Submit" button. Typical	The answer in the textbox is successfully submitted and a tick is added to the "answered" label.	As expected

23	Ensuring that clicking on the "Submit" button works with the "multiple choice" style of question.	Clicking on the "Submit" button. Typical	The selected letter is successfully logged and a tick is added to the "answered" label.	As expected
24	The "End Quiz" button works as intended when no questions are answered.	Clicking on the "End Quiz" button. Typical	The quiz is ended and the user is asked whether or not they wish to view the mark scheme.	As expected
25	The "End Quiz" button works as intended when all 5 questions have been answered.	Clicking on the "End Quiz" button. Typical	The quiz is ended and the user is asked.	As expected
26	The "End Quiz" button works as intended when 2 questions have been answered.	Clicking on the "End Quiz" button. Typical	The quiz is ended and the user is asked.	As expected
27	The "View Mark scheme" option works as intended.	Clicking on the "Yes" button after correctly answering **Question X_1** Typical	The mark scheme is generated for the 5 questions in the quiz.	As expected - **See SSS10**
"Select A Subtopic" Section				
28	Testing that clicking on a subtopic button successfully loads up a quiz containing questions related to that subtopic.	Clicking on the "Induction" subtopic button Typical	A quiz containing exclusively questions from the selected subtopic is generated.	As expected – **See SSS11**
"Create A Class" Section				
29	Testing that students from **Data Set 1** are correctly loaded into the list box ready for selection.	**Data Set 1** Typical	All the names of the students from **Data Set 1** are decrypted and loaded into the selection list box.	As expected – **See SS12**
30	Testing that adding a student to the new class list box works as intended.	Selecting the name "Richard Surrey" from the selection list box and then clicking the "Add" button.	The name selected in the selection list box is copied over to	As expected – **See SS13**

		Typical	the new class list box.	
31	Testing that removing a student from the new class list box works an intended.	"Richard Surrey" Typical	The name selected in the new class list box is removed from the new class list box.	Student is not removed from the new class list box - **See SS14 RETEST NEEDED**
32 ***Retest for test #29***	As above	As above	As above	As expected – **See SS15**
33	Testing if creating a new class using all the students in **Data Set 1** and a valid class name is successful.	"13/CS", **Data Set 1** Boundary (Boundary because the maximum number of student possible are used to create a new class)	A message box appears informing the user that their attempt to create a class was successful.	As expected – **See SS16**
34	Testing if creating a class using all the students in **Data Set 1** and invalid class name is successful.	N/A, **Data Set 1** Erroneous	Message appears informing the user that the class name is invalid. The class is not created.	As expected
35	Searching for the student "James Murray" by ID.	ID Number: 4 Typical	The name "James Murray" is copied over from the student names list box to the new class list box.	As expected
36	Searching for the student "John Picard" by ID.	ID Number: 1 Boundary (Boundary because the first ID number in the list of students is used)	The name "John Picard" is copied over from the student names list box to the new class list box.	As expected – **See SSS17**
37	Searching for a student with an ID that is out of range.	ID Number: -1 Erroneous (Erroneous because the ID number can only be between one and the number of students signed up to the program)	An error message is displayed to the user, stating that the search ID is out of range.	As expected – **See SSS18**
"View Student Progress" Section				
38	Testing that all the modules are loaded into the "Select Module" combo box.	Clicking the arrow on the "Select Module" combo box. Typical	All three of the modules are displayed in the combo box.	As expected

39	Testing that all the topics related to a particular module are correctly loaded into the "Select Topic" combo box.	Selected Module: Pure Maths. Typical	The topics "Proof" and "Integration" are loaded into the "Select Topic" combo box.	As expected – **See SS19**
40	Testing that the statistics for the student "Richard Surrey" are correctly calculated and subsequently displayed into the student data list box.	**Data Set 4** Selected Module: Pure Maths. Selected Topic: Proof. Typical	*Total: 50%* *Pure: 50%* *Statistics: 50%* *Mechanics: 50%* is displayed in the student data list box.	The program becomes unresponsive upon selecting the name "Richard Surrey" – **See SS20** **RETEST NEEDED**
41 ***Retest for test #40***	As above	As above	As above	As expected – **See SS21**
42	Testing that the "Score against number of attempts" graph is correctly populated using the statistics in **Data Set 4**.	As above.		As expected – **See SS21**
43	Testing that the subtopic bar chart for "Proof" is correctly displayed using the statistics in **Data Set 4**.	**Data Set 5**	A bar chart is generated displaying the following statistics: *Induction: 40%* *Counter-example: 60%*	As expected – **See SS21**
"View Class Progress" Section				
44	Inputting an invalid class name into the class name combo box to test whether the program crashes or not.	"invalid_class_name" Erroneous	An error message appears informing the user that the class name they entered is invalid, and asking them to select a name from the combo box.	The program crashes – **See SSS22** **RETEST NEEDED**
45 ***Retest for test #44***	As above	As above	As above	As expected – **See SS23**
46	Determining if the students	Class name: "13/TT" Typical	The student names "James	As expected – **See SS24**

	assigned to class “13/TT” are all correctly displayed in the student names list box.		Murray”, “John Picard” and “Billy Marsh” are displayed in the student names list box.	
47	Determining if the percentage score bar chart for each student in class “13/TT” is correctly displayed.	As above.	Bars indicating scores of 50%, 38% and 25% for “Billy Marsh”, “John Picard” and “James Murray” respectively are displayed.	As expected – **See SS24**
48	Determining if the line chart is correctly displayed.	As above.	A line chart line chart showing the relationship between the scores of students in class “13/TT” is displayed.	As expected – **See SS25**
49	Determining if the results for class “00/ZZ” is correctly displayed.	Class name: “00/ZZ” Boundary (Boundary because the smallest number that can be used to create a class name is zero)	The class analysis is correctly displayed.	An error message is displayed, informing the user that the class name is out of the allowable bounds – **See SS26** **RETEST NEEDED**
50 ***Retest for Test #49***	As above	As above	As above	As expected – **See SS27**

4.3 Hand calculations

4.3.1 Hand calculation 1

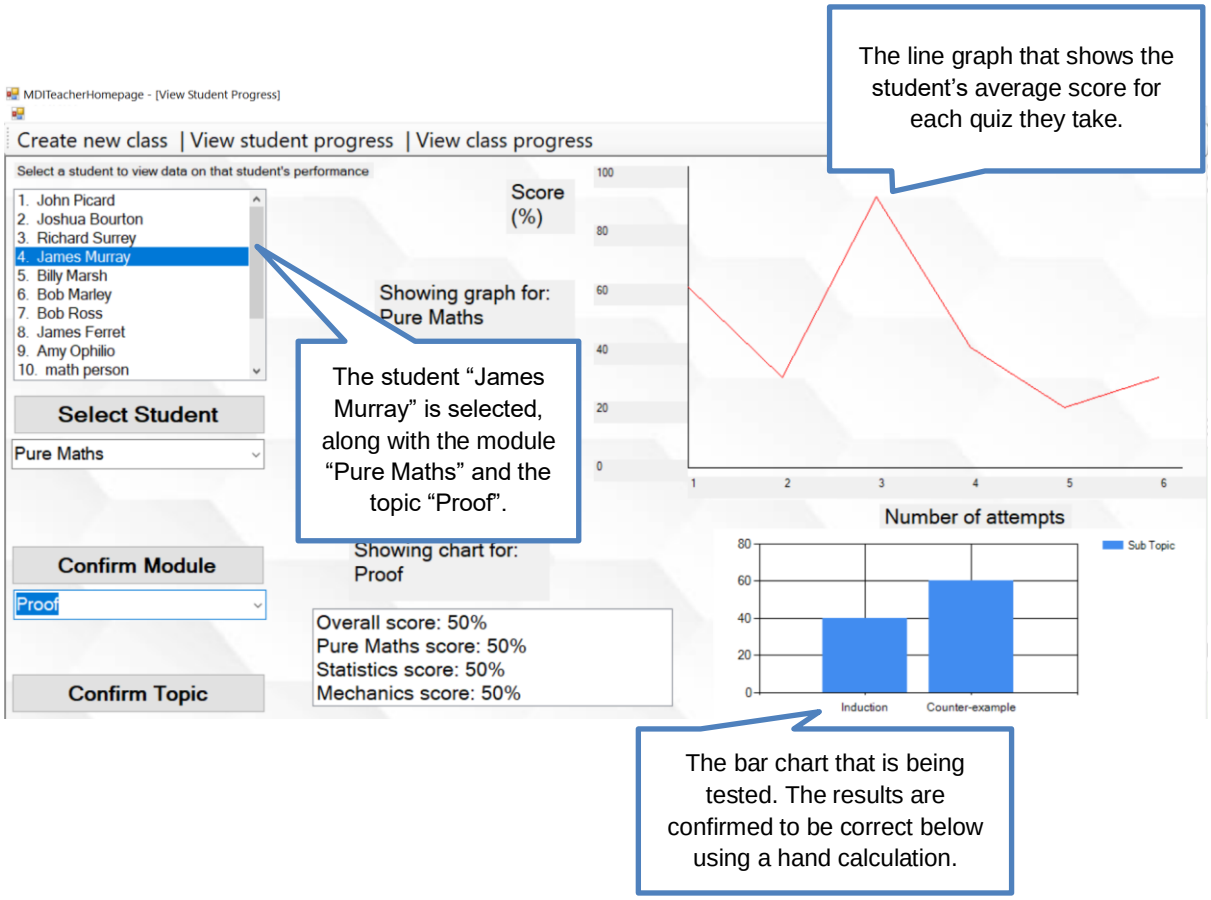
In the *View Student Progress* section of the program the teacher has the ability to select a student and then view two different graphs based on their performance on the quizzes they have took. One such graph is a bar chart that depicts the average percentage score of the student in each of the two subtopics related to the selected topic.

This average percentage score is calculated using the following line of code:

```
avg = (CInt(strToAverage.Substring(1, 5)) / CInt(strToAverage.Substring(7, 5))) * 100
```

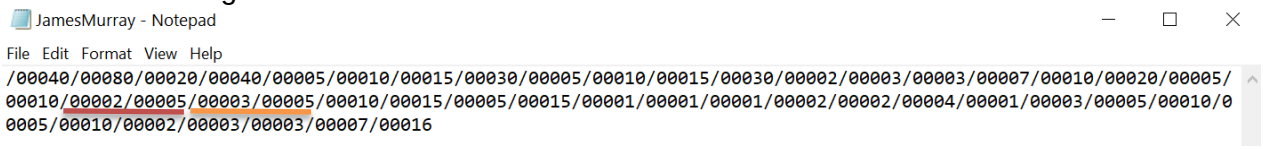
The example student that will be used to confirm that the calculation works as intended is the student user “James Murray”, the example module is “Pure Maths” and the example topic is “Proof”. The details are entered into the input boxes as shown in the screenshot below. The bar chart (bottom right) that is being tested is then generated and displayes the following scores:

- Induction: 40%
- Counter-example: 60%



Text file

The text used to generate the bar chart is shown below:



The underlined results are used to calculate the average percentage scores related to the selected topic (in this case 'Proof'). The names of these subtopics are 'Induction' and 'Counter-example'.

The format of this text file (described in *2.3 File handling and data processing*) is *Number of times that type of question was correctly answered / Total number of attempts for that type of question.*

Hand calculation

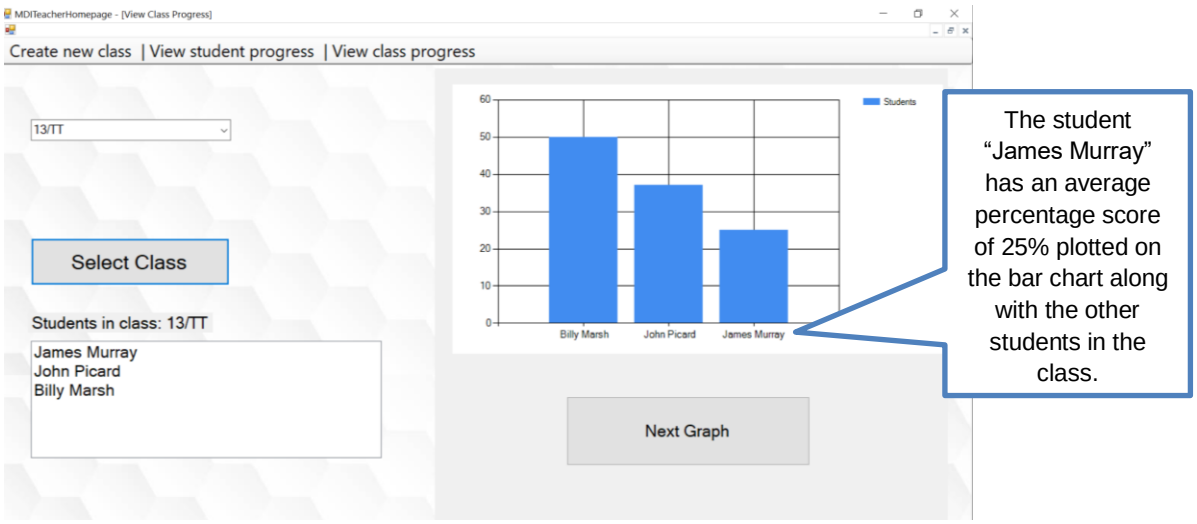
The student's results on first subtopic, Induction, is underlined in red. The student's results on second subtopic, Counter-example, is underlined in orange.

The student's percentage score on the first subtopic is therefore:
(2 / 5) * 100 = 40%
And the student's percentage score on the second subtopic is:
(3/5) * 100 = 60%

The values shown in the bar chart are 40% and 60% respectively. Therefore, the outcome of the test is as expected.

4.3.2 Hand calculation 2

The bar chart displayed in the *View Student Progress* shows the student's average percentage score for each quiz they take. To test that this bar chart correctly displays a student's average results against number of attempts, the same example student 'James Murray' is used.

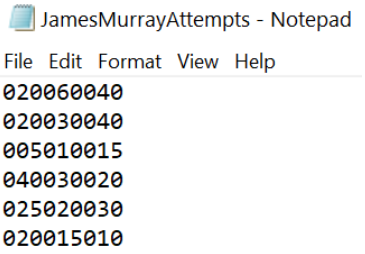


The following code is used to calculate the average for each attempt:

```
For i = 0 To 2
        Results(i) = CInt(ResultsLine.Substring((3 * i), 3))
Next
Return Math.Round(Results.Average())
```

Text file

To plot a student's average percentage score in each attempt against number of attempts the *Attempts* text file is used. The *Attempts* text file for example student 'James Murray' is shown below. Each line represents a single attempt at a quiz. The format of the text file for each line is *Pure Maths score, Statistics score, Mechanics score*, with each of the modules occupying three characters on a line.



To calculate the average (mean) score for attempt the average of each module percentage is calculated. So, for *JamesMurrayAttempts.txt*:

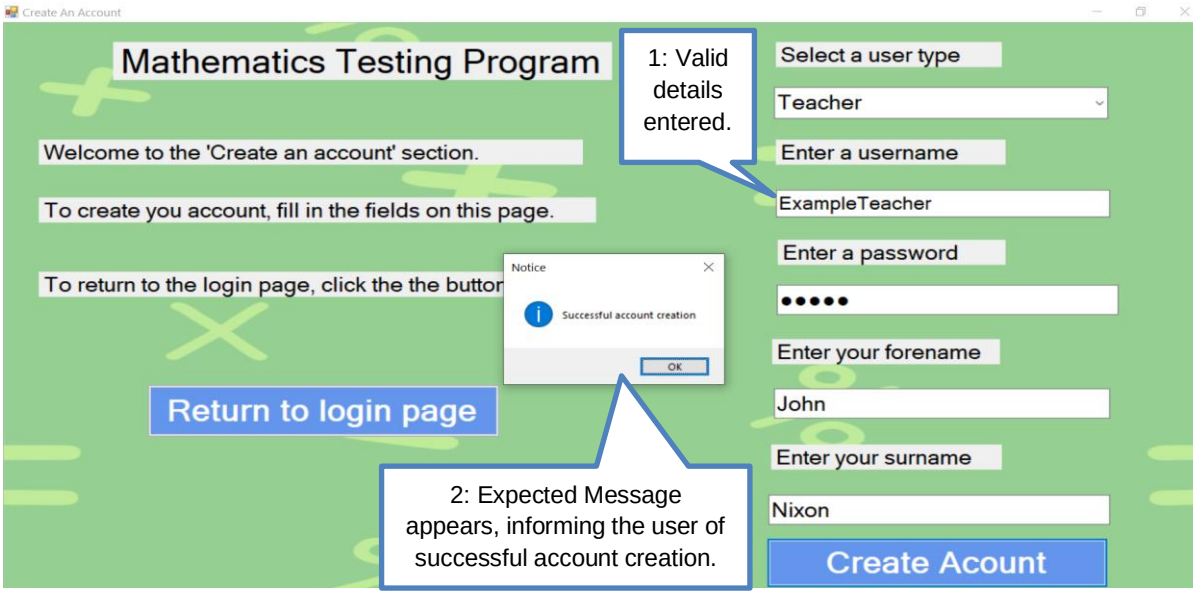
- 1st line: (20 + 60 + 40) / 3 = 40(%)
- 2nd line: (20 + 30 + 40) / 3 = 30(%)
- 3rd line: (5 + 10 + 15) / 3 = 10(%)
- 4th line: (40 + 30 + 20) / 3 = 30(%)
- 5th line: (25 + 20 + 30) / 3 = 25(%)
- 6th line: (20 + 15 + 10) / 3 = 15(%)

The average of these averages is then found by the following calculation:
(40 + 30 + 10 + 30 + 25 + 15) / 6 = 25(%)

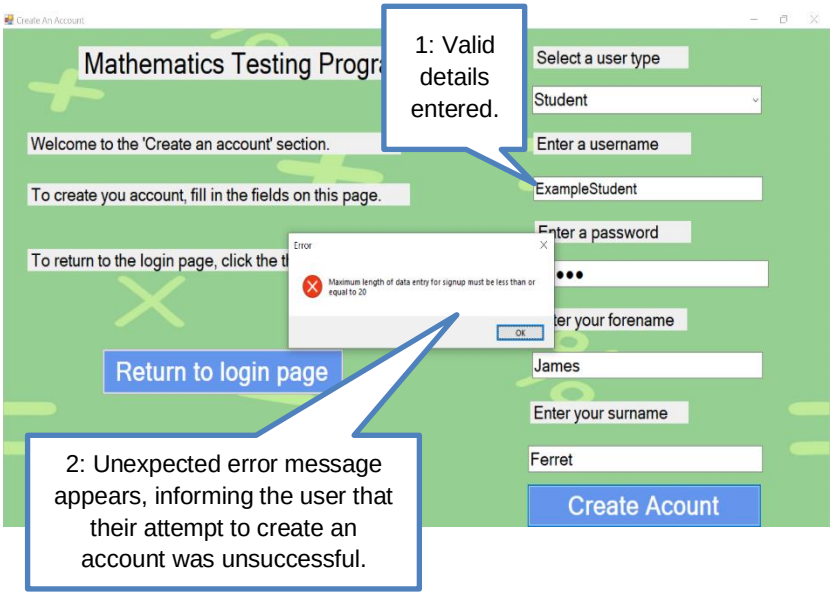
Which is the expected value. Therefore, the test is successful.

4.4 Screenshots of tests

Screen Shot 1
The following screenshot illustrates a successful attempt at signing up to the program as a teacher as detailed in **Test 2**.

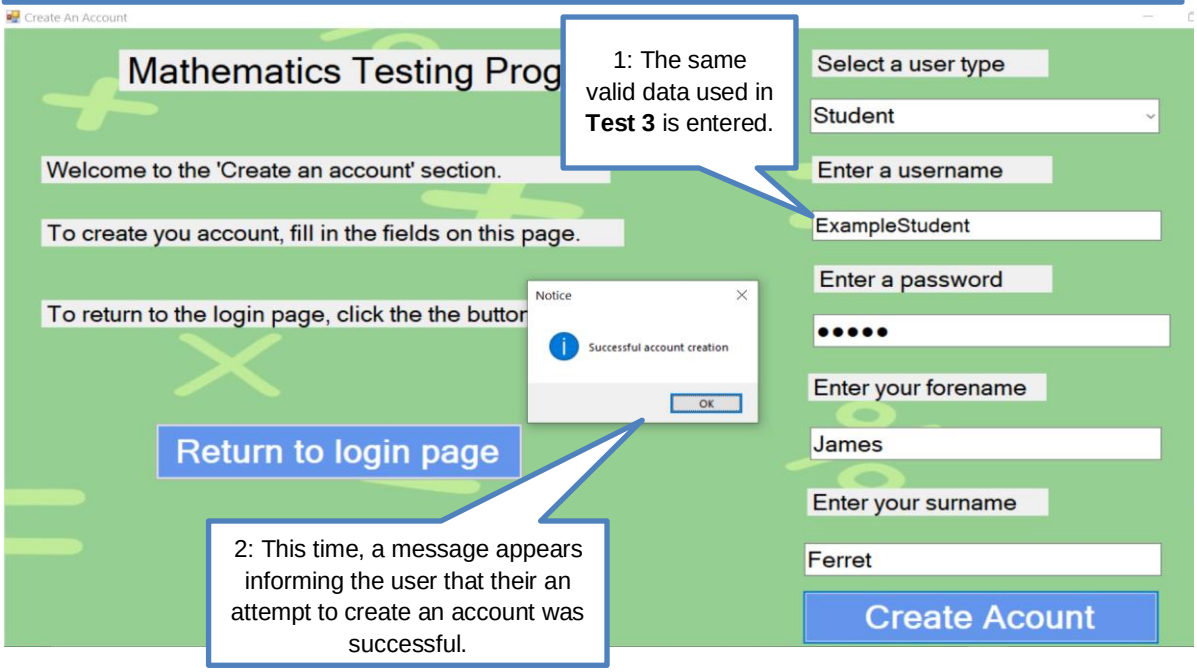


Screen Shot 2
The following screenshot illustrates an unexpected output whilst using **Data Set 1** to attempt to create an account.



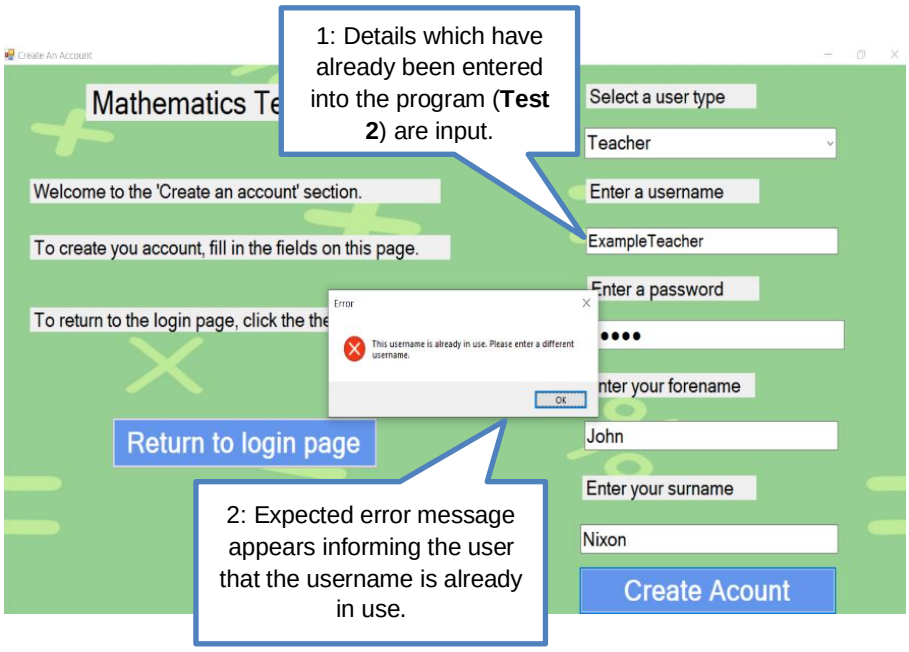
Screen Shot 3

The following screenshot depicts the retest of **Test 3** in order to correct the error found (shown in **Screen Shot 1**)



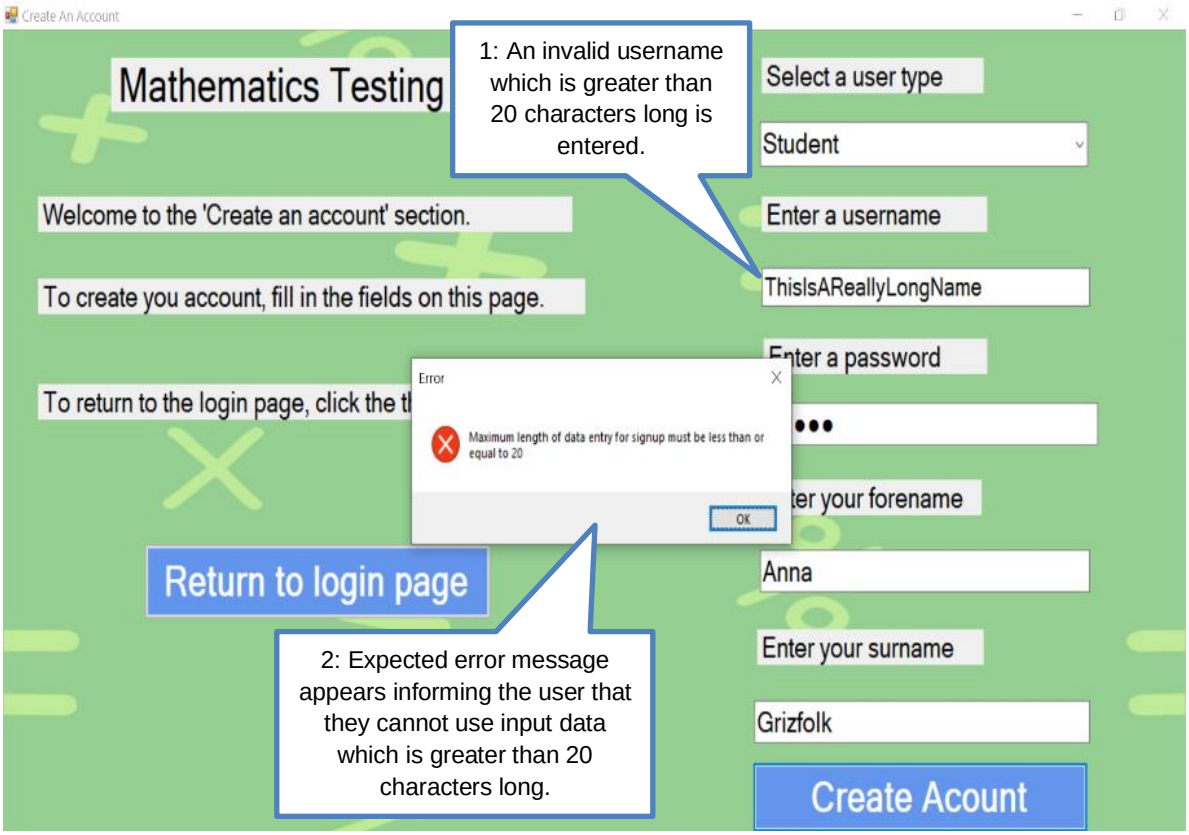
Screen Shot 4

The following screenshot illustrates an expected error message when attempting to sign up to the program with a username that is already in use, as detailed in **Test 5**.

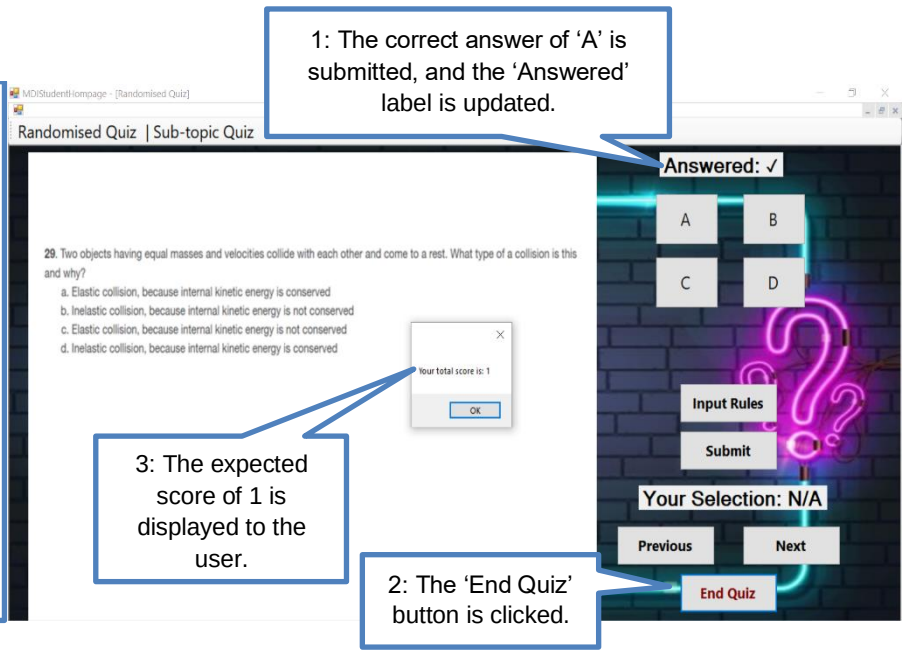


Screen Shot 5

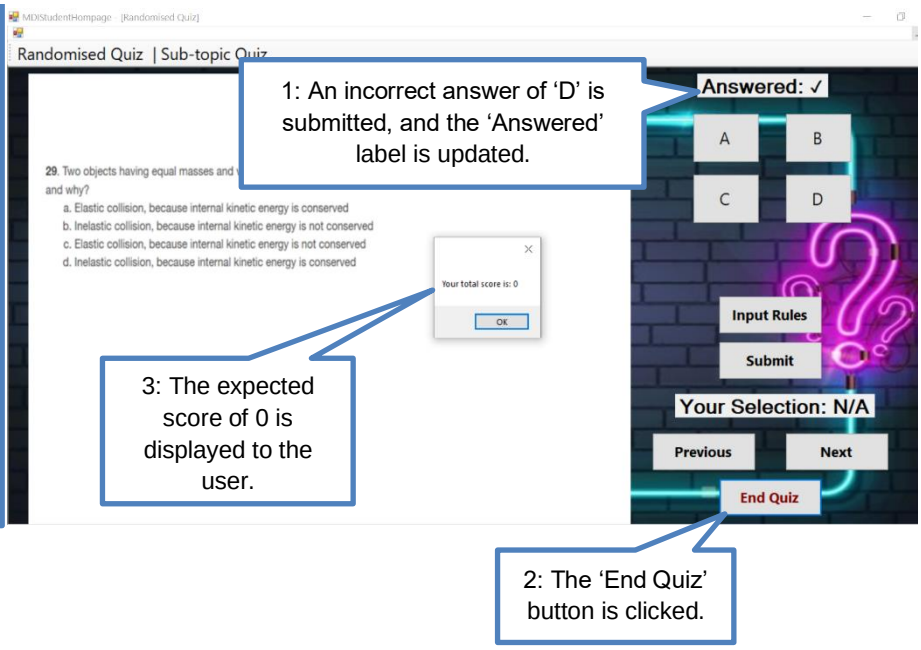
The following screenshot demonstrates what happens when the user attempts to sign up to the program with a username that is over 20 characters long.



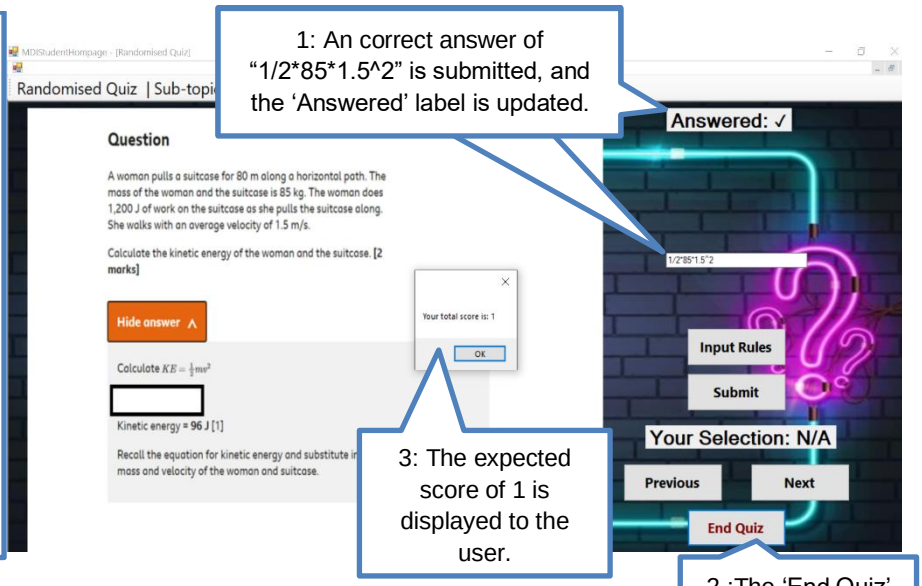
Screen Shot 6
The following screenshot illustrates what the program outputs when **Input 1** (correct answer) is used to answer **Question X_1**.



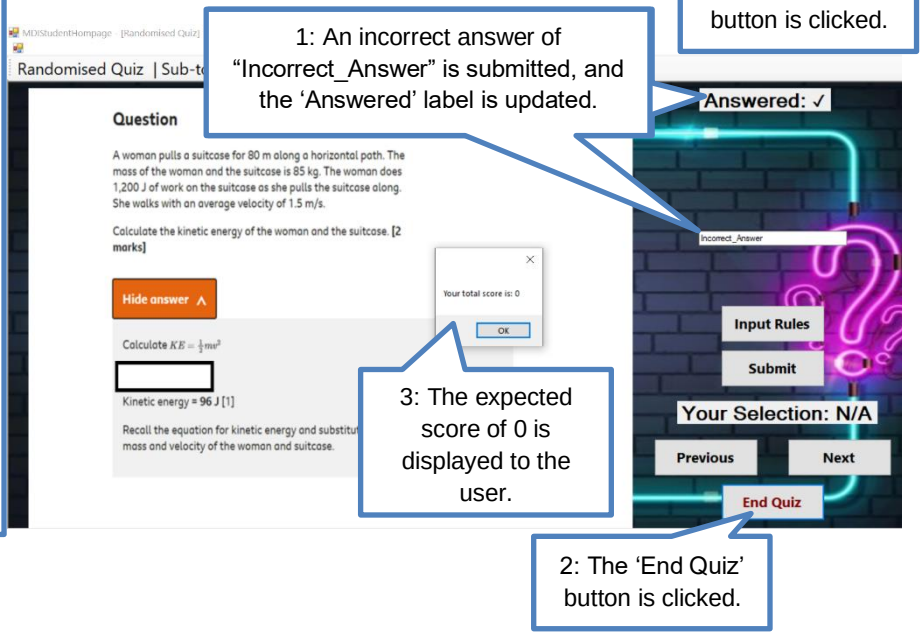
Screen Shot 7
The following screenshot illustrates what the program outputs when **Input 2** (incorrect answer) is used to answer **Question X_1**.



Screen Shot 8
The following screenshot illustrates what the program outputs when **Input 1** (correct answer) is used to answer **Question X_2**.

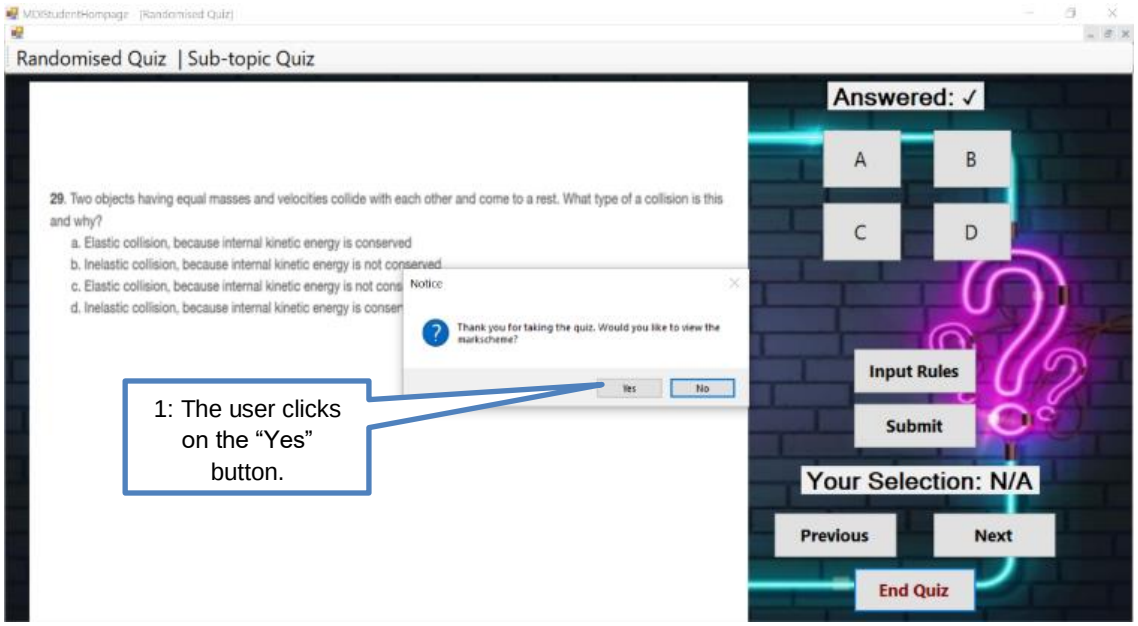


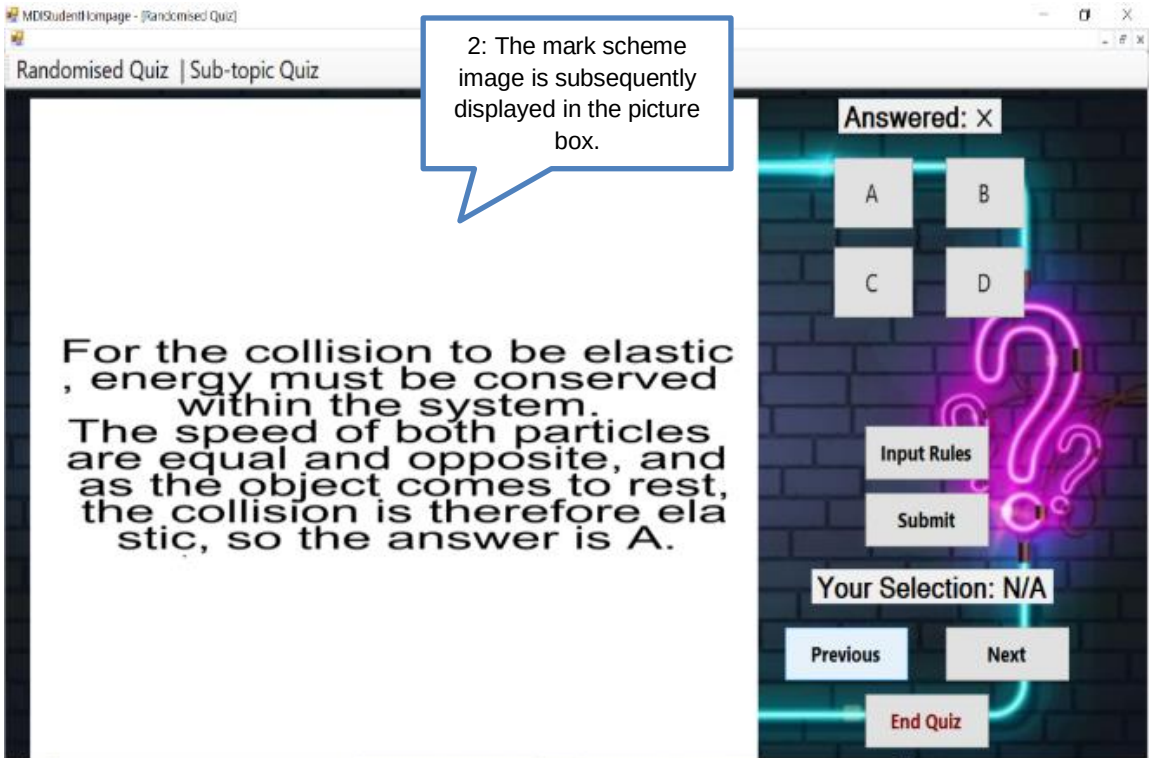
Screen Shot 9
The following screenshot illustrates what the program outputs when **Input 2** (incorrect answer) is used to answer **Question X_2**.



Screen Shot Set 10

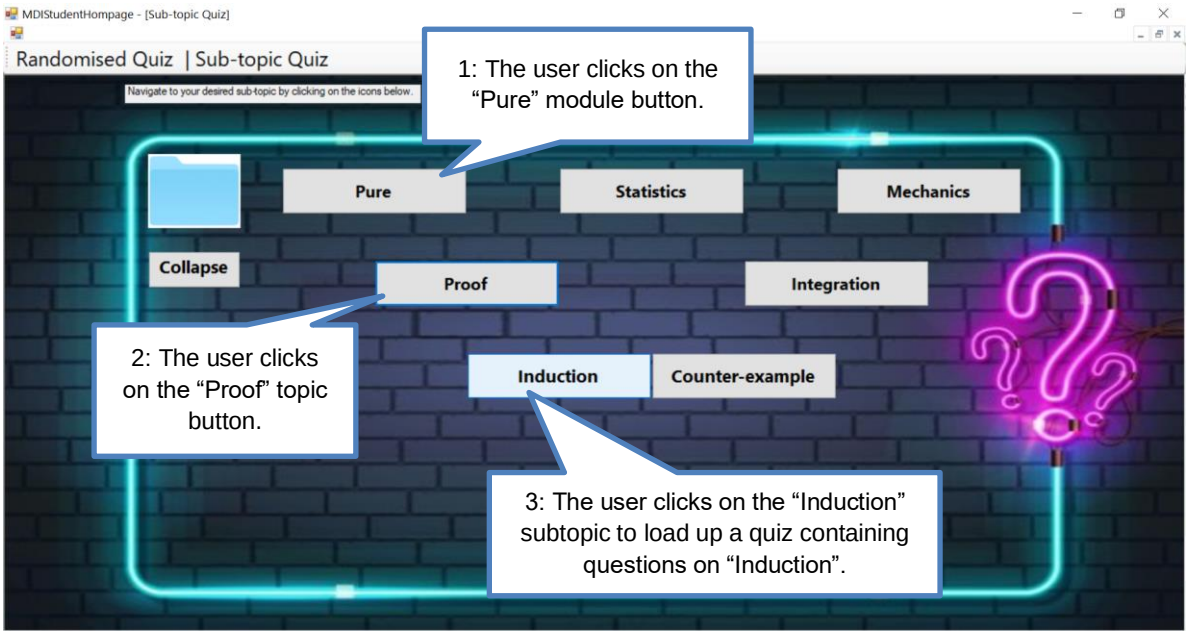
The following screenshot set illustrates what happens when the user presses the “Yes” button in response to being asked whether or not they wish to view the markscheme.



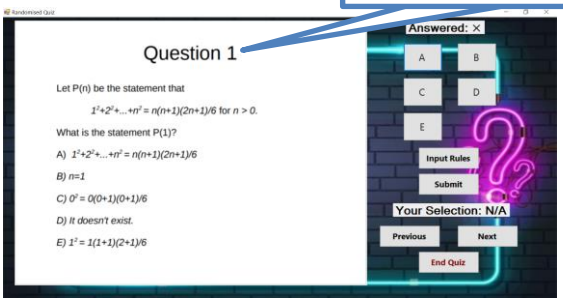


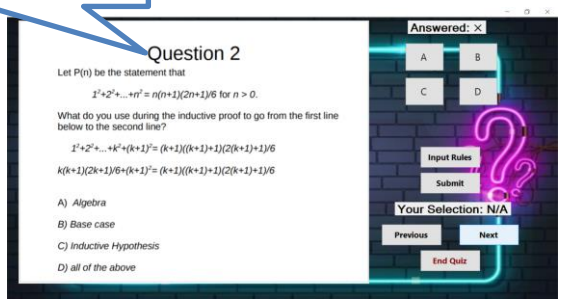
Screen Shot Set 11

The following screenshot set illustrates what happens when the user clicks on a subtopic button in the “Select A Subtopic” section of the program.

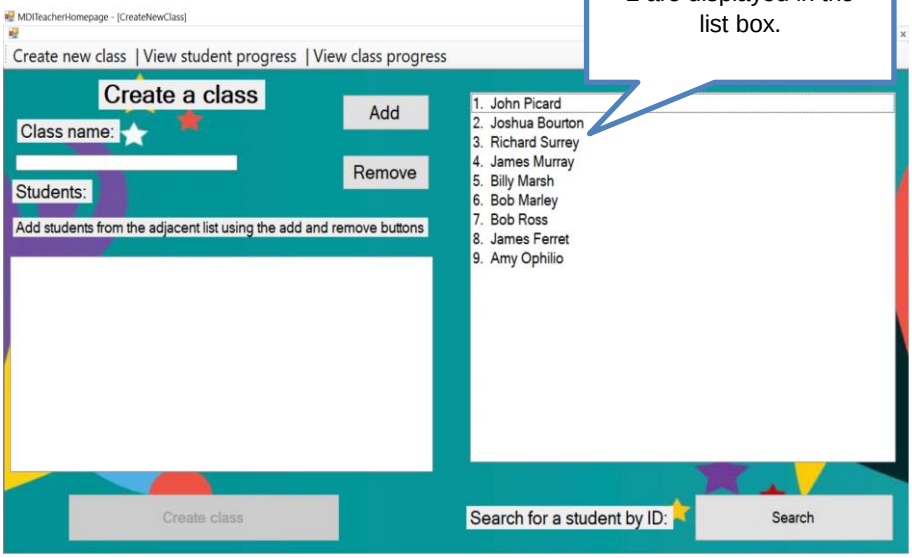


4: The relevant questions are then loaded into the picture box.

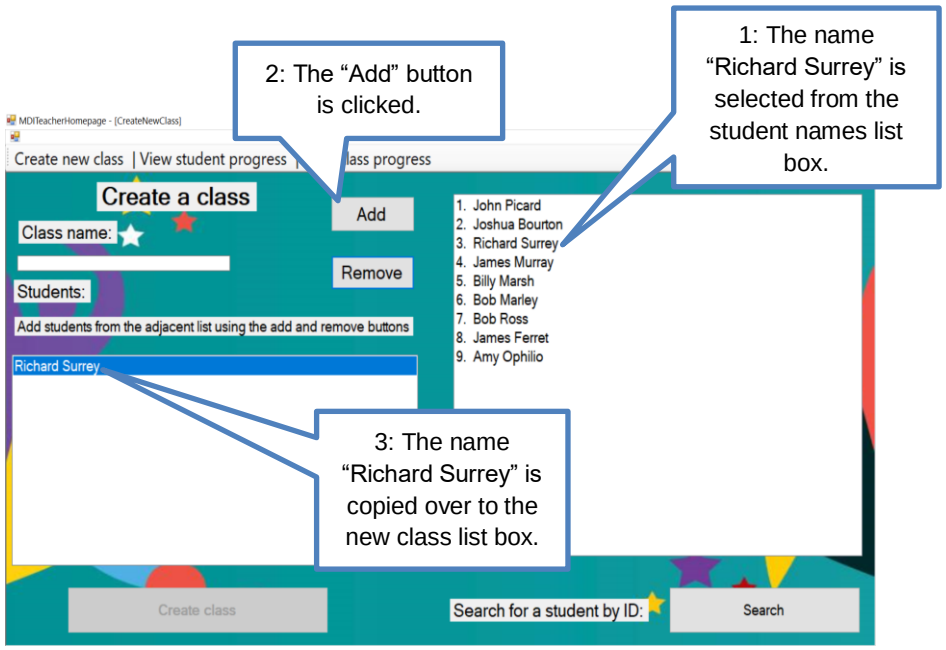




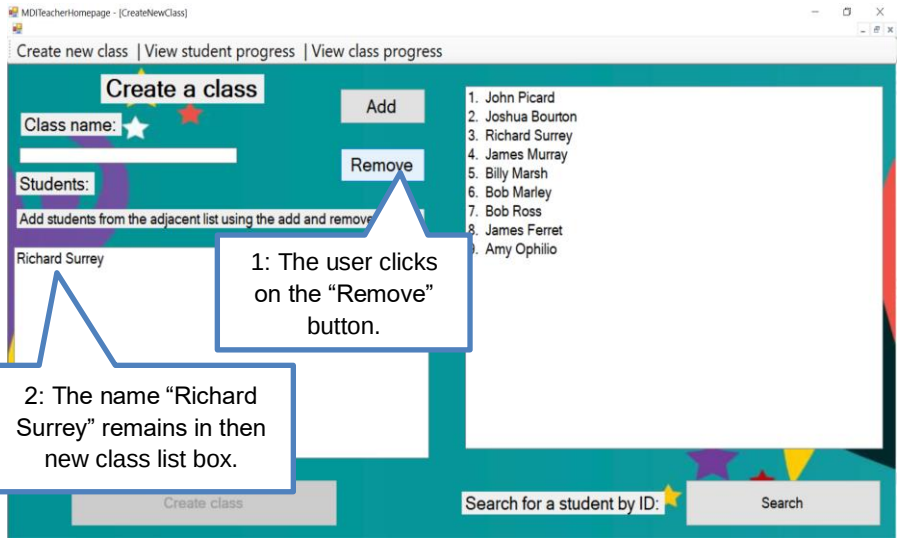
Screen Shot 12
The following screenshot shows demonstrates how the students from **Data Set 1** are correctly loaded into a list box ready for selection.



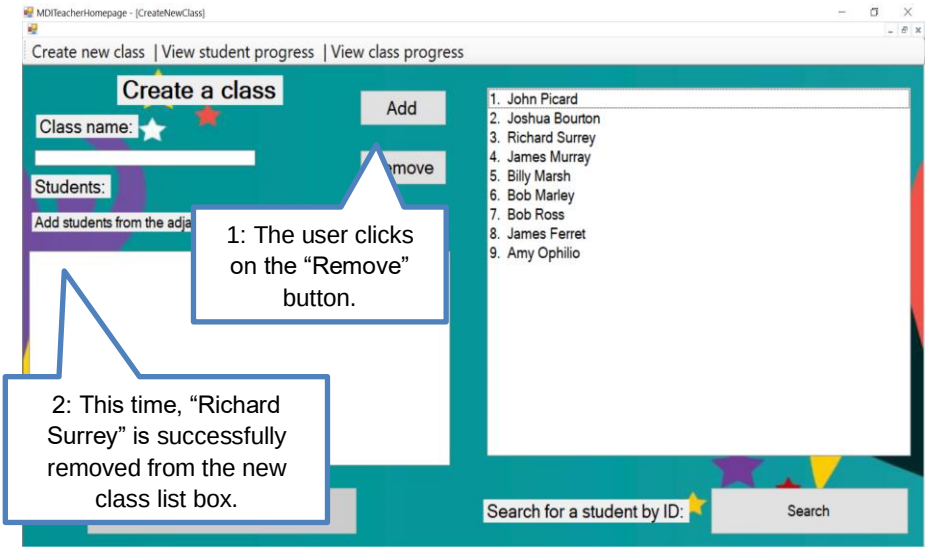
Screen Shot 13
The following screenshot illustrates the test for adding a student to the new class list box.



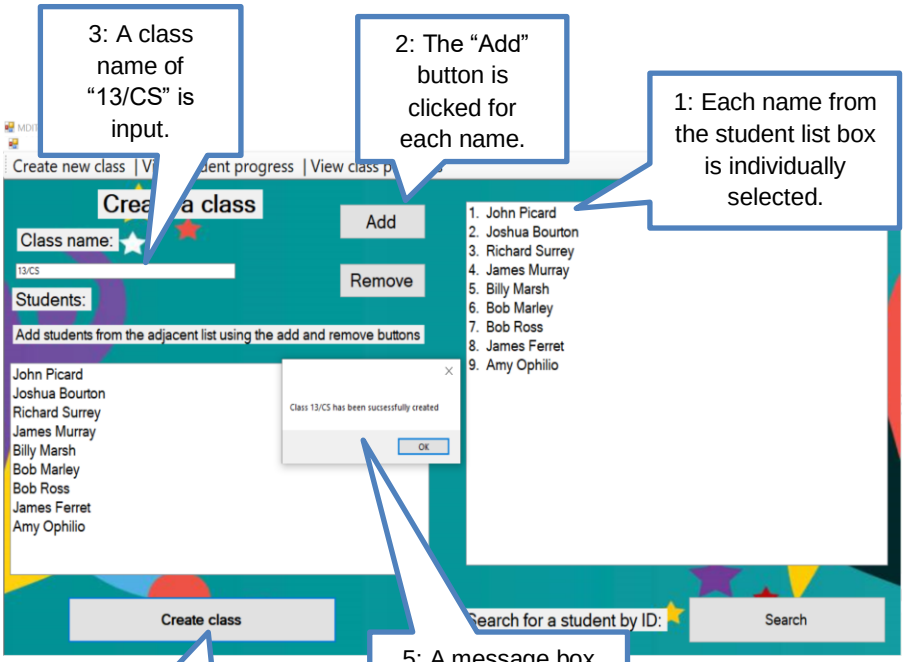
Screen Shot 14
The following screenshot identifies an error when attempting to remove a student from the new class list box.

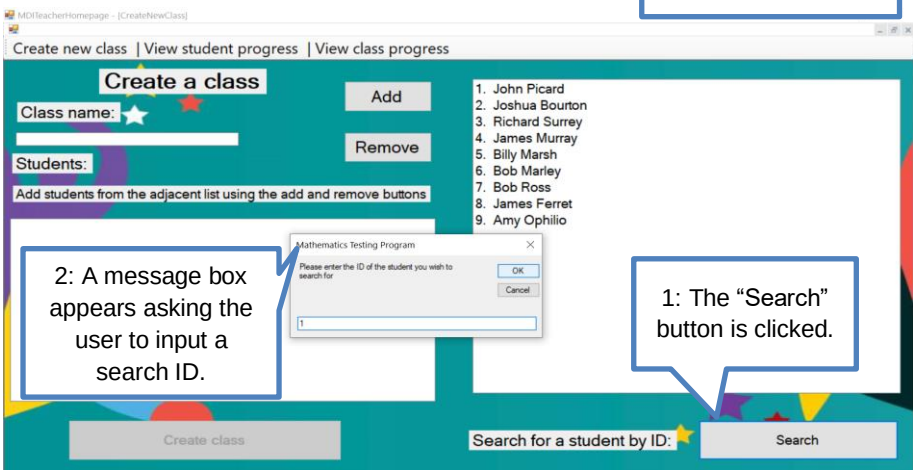


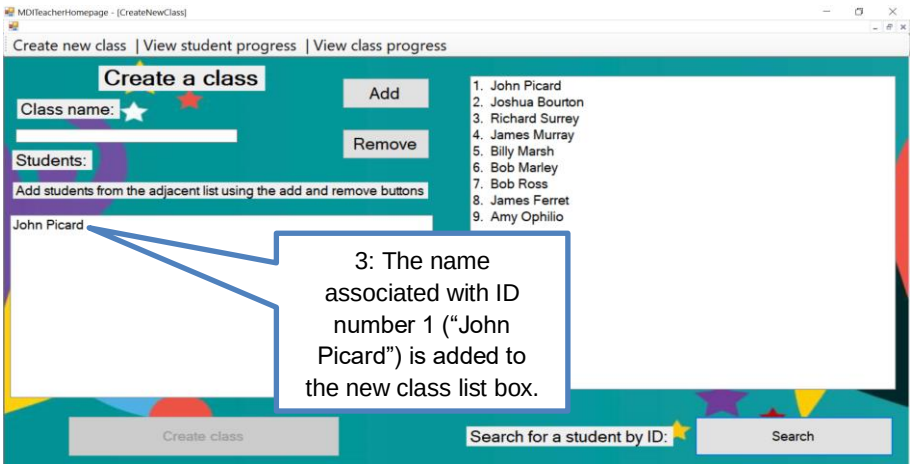
Screen Shot 15
The following screenshot demonstrates a successful retest of **Test #29**



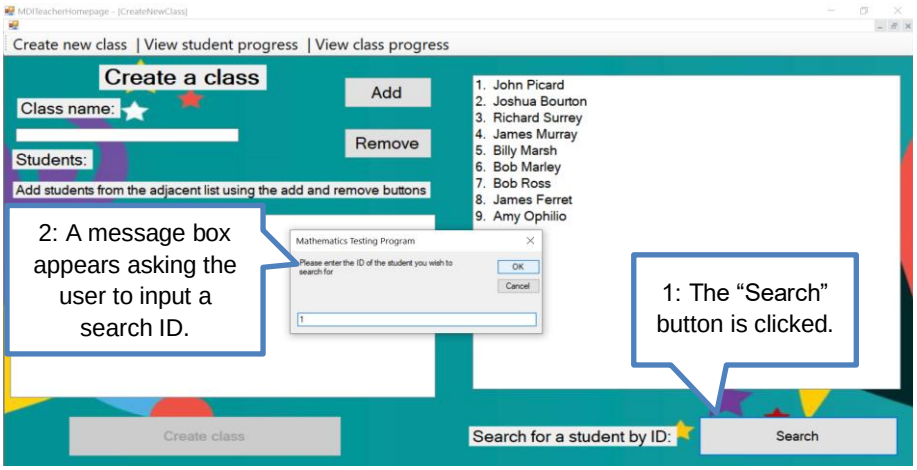
Screen Shot 16
The following screenshot shows a successful attempt at creating a new class using all the students in **Data Set 1** and a valid class name.

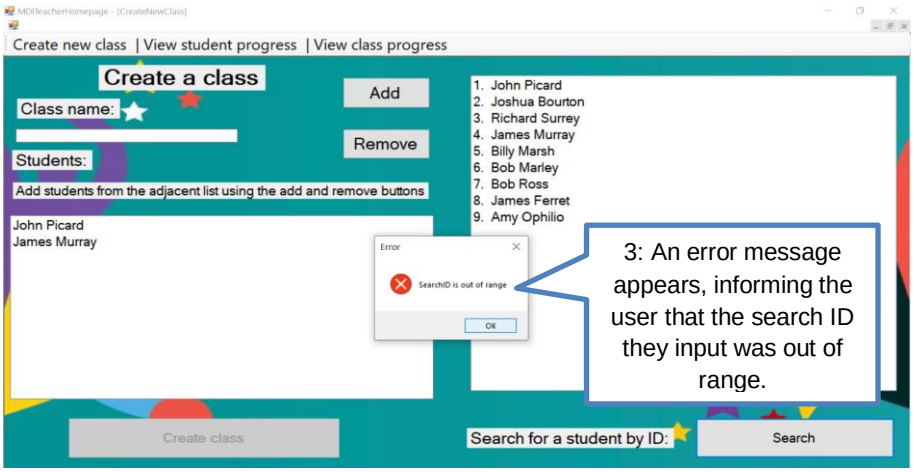






Screen Shot Set 17
The following screenshot shows a successful attempt to search for the student "John Picard" by ID



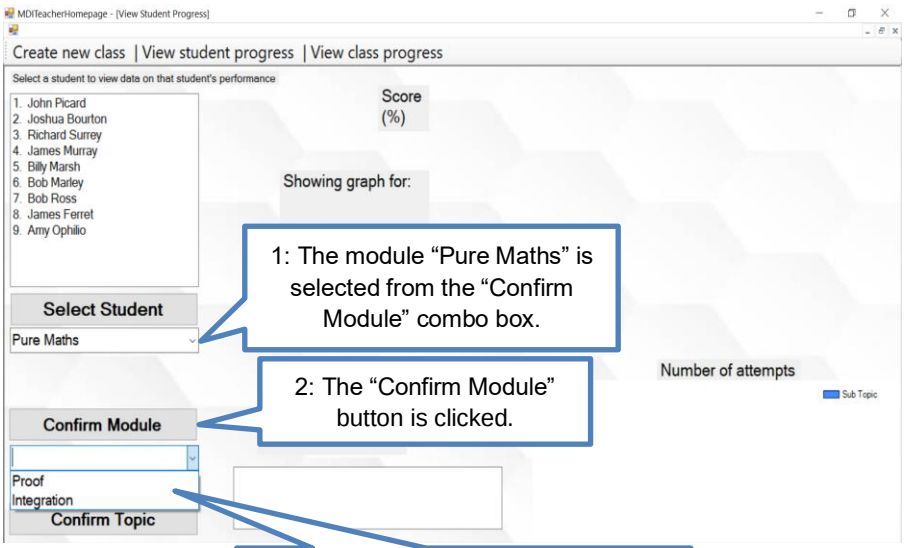


Screen Shot Set 18

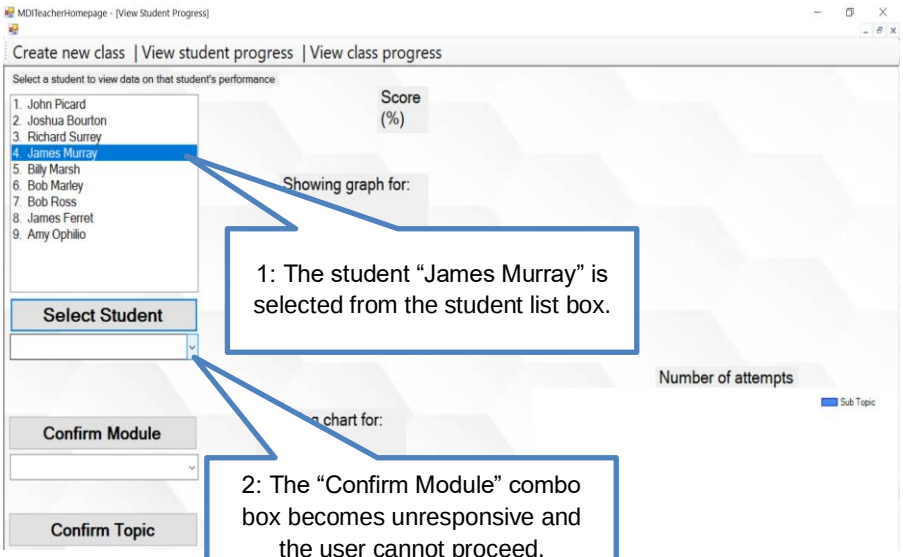
The following screenshot depicts what happens when the teacher attempts to search for a student by ID using an ID that is out of range.

Screen Shot 19

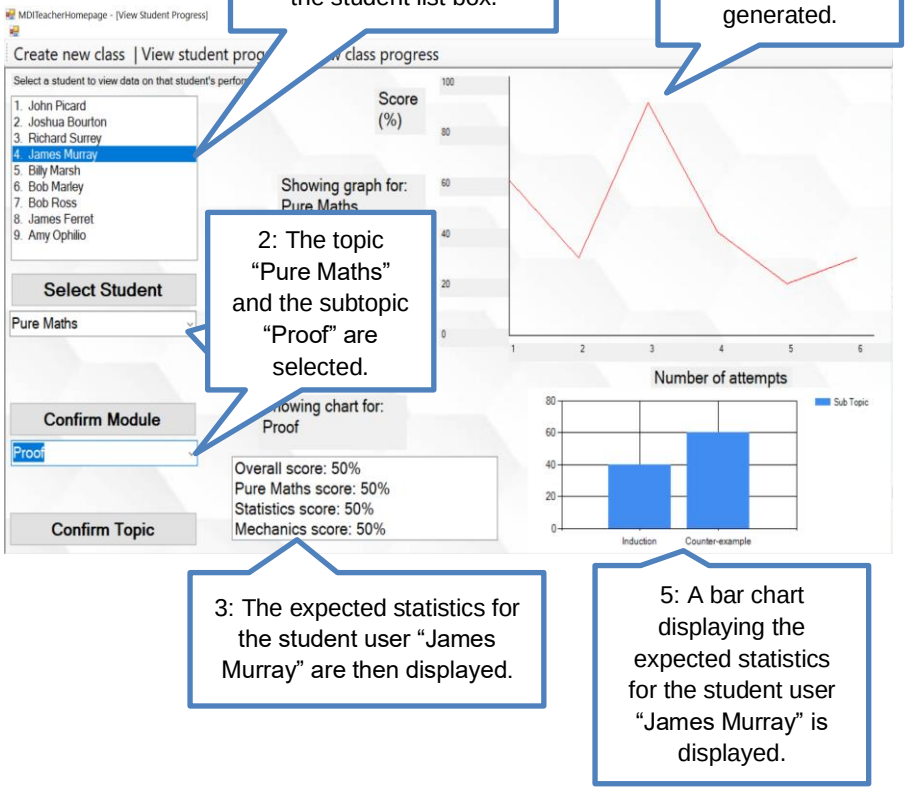
The following screenshot shows how the topics related to the module "Pure Maths" are loaded into the "Confirm Topic" combo box.



Screen Shot 20
The following screenshot shows that the program became unresponsive upon attempt to select the student "James Murray" from the student list box.

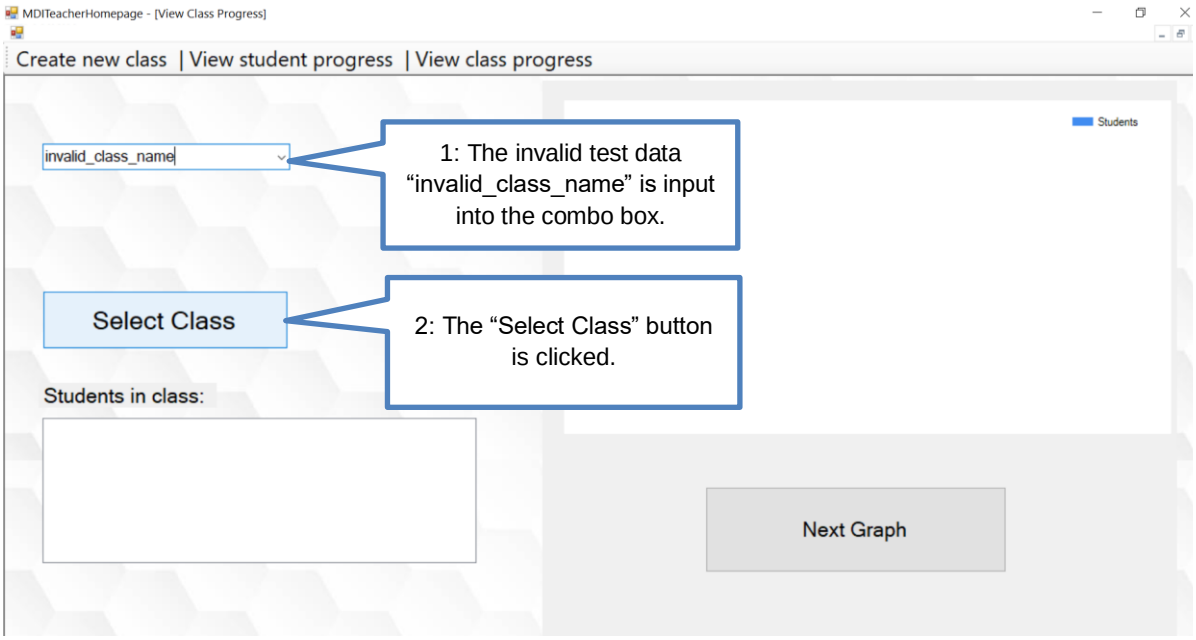


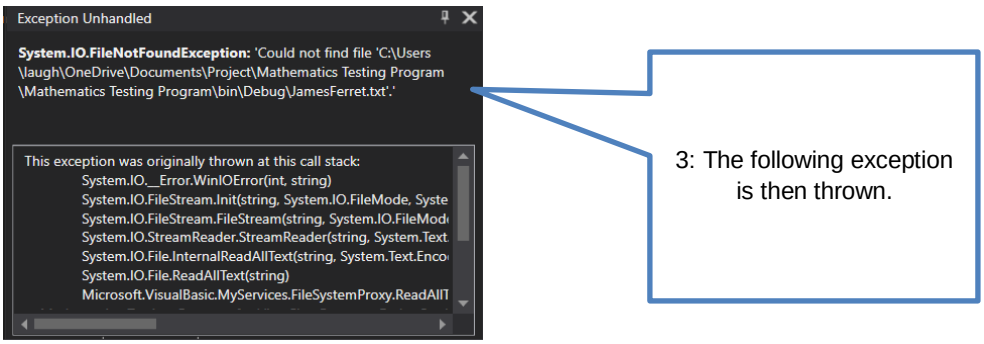
Screen Shot 21
The following screenshot shows a retest for **Test #40**

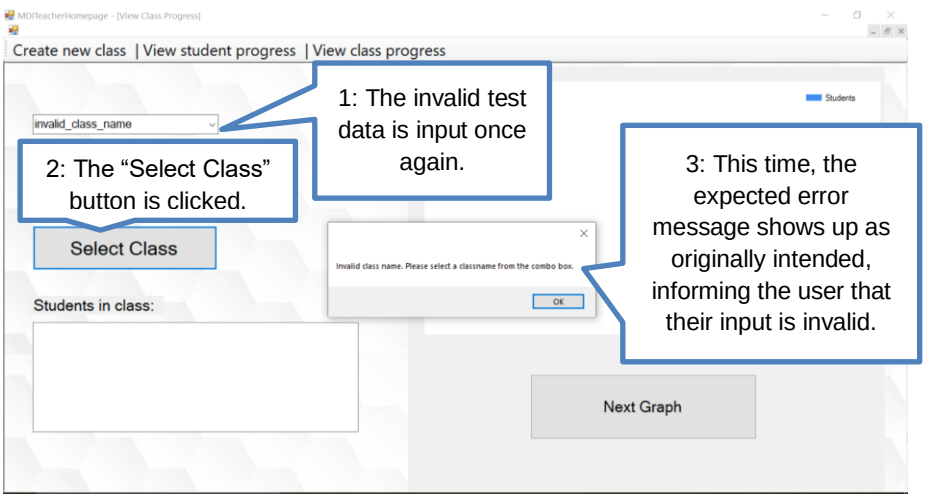


Screen Shot Set 22

The following screenshot shows what happens when an invalid class name is input into the class combo box.



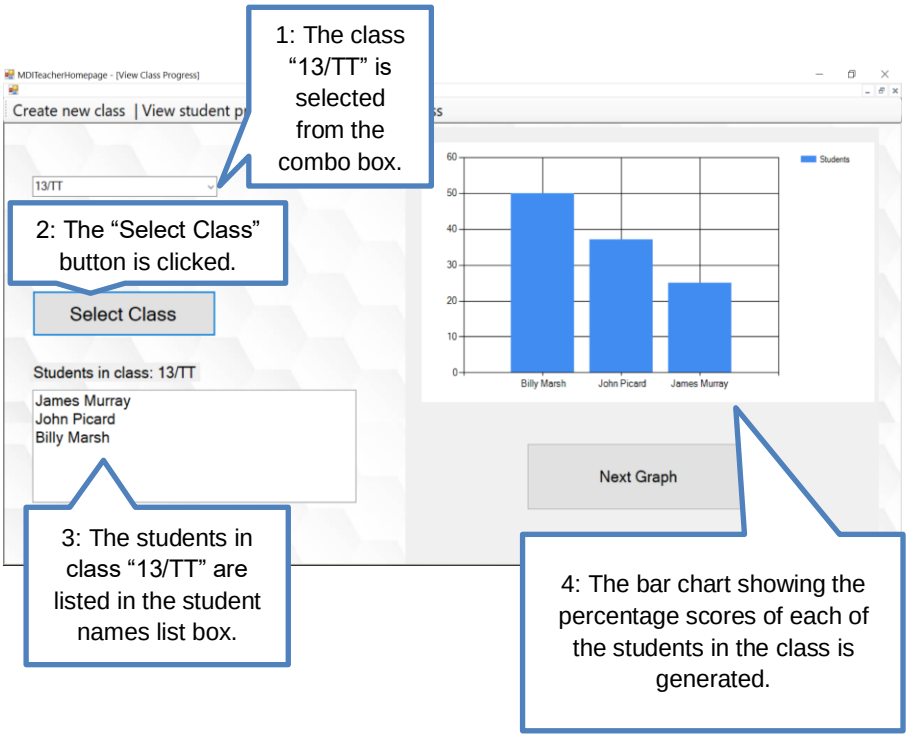




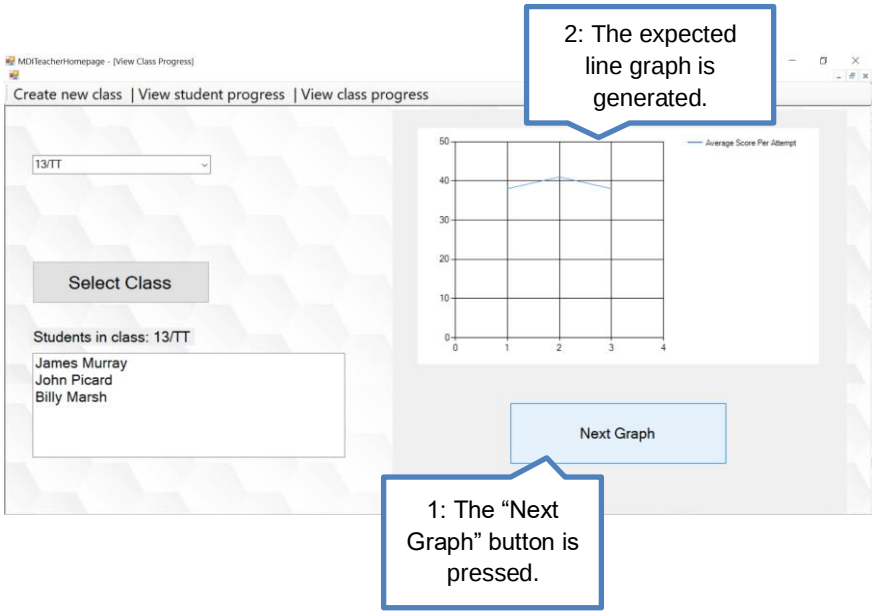
Screen Shot 23

The following screenshot shows a retest for **Test #44**.

Screen Shot 24
The following screenshot shows what the program displays when the test class "13/TT" is selected.

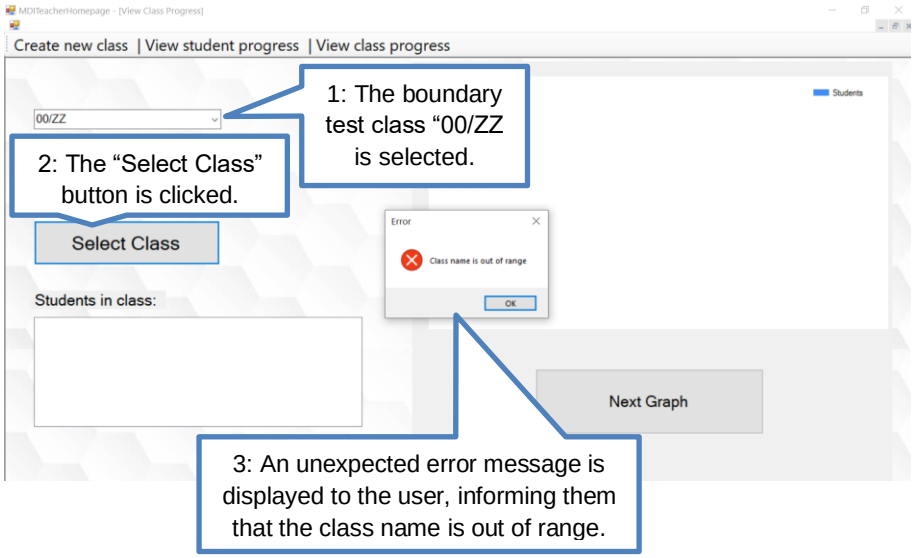


Screen Shot 25
The following screenshot shows the line graph that is generated by the program that shows the relationship between the student scores.

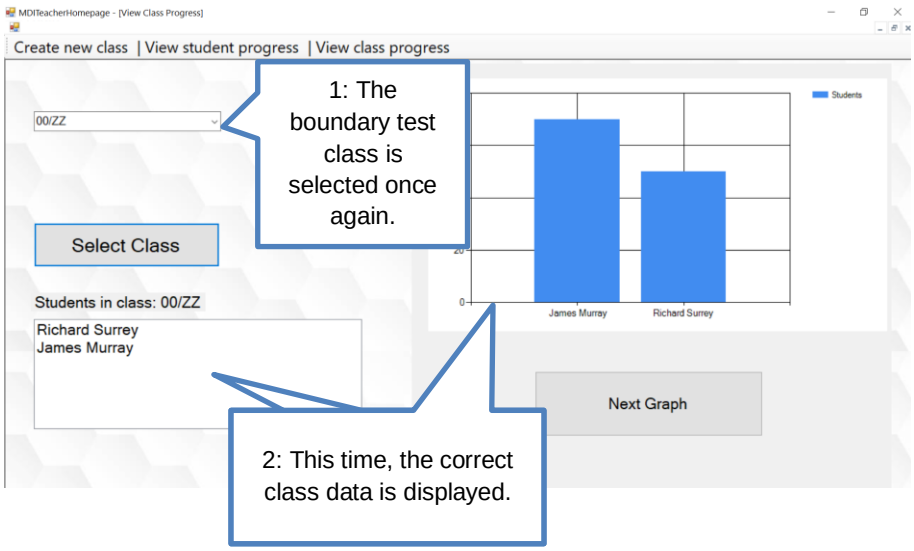


Note: The line graph here has only has three data points and so does not yet demonstrate a clear relationship between student's scores. The usefulness of this graph to the end user is significantly more pronounced when a large number of students are added to a class. The more students that there are in a class the better the graph is at providing a clear insight into class's progress in quizzes when compared with their peers.

Screen Shot 26
The following screenshot shows the results of selecting a class with boundary values in its name.



Screen Shot 27
The following screenshot shows the retest for **Test #49**.



EVALUATION

5.1 How well the project met the requirements

Requirement 1: A login system consisting of a username and password, that will allow users to access the program from the machine the program is installed on.

1a) The ability to create an account from within the program.

1b) The login system will consist of two different styles of user accounts: an account for teachers and an account for students.

1c) Personal data, including the username, password, forename and surname of each person using the program, will be taken in by the program upon signup and stored securely in a text file using encryption.

The program has two different styles of user accounts that each perform different functions, as described in 2.1 Modular structure of the system.

The program has a fully functioning login system which allows both students and teachers to sign into the section of the program that is relevant to them. The login details for all students are stored in a single text file called *Students.txt*, and the login details for all teachers are stored in another separate text file called *Teacher.txt*.

However, the extent to which the data is stored securely could be improved with the use of a more complex encryption algorithm. The program uses a Caesar shift to translate the details entered by the student or teacher upon signup to the program, and as such could be easily broken by brute force should the text files containing these login details be compromised.

Requirement 2: Different background images and colour schemes will be used to more easily differentiate between the different types of user accounts.

On paper, this requirement was fully met. However, the background images used were arguably more busy than they needed to be, and so add to the complexity of the UI. The possibility of changing this as a potential improvement has been discussed below.

Requirement 3: A clean, simple user interface that will allow for easy navigation of the different kinds of quizzes will be built for the *Student* user type. Features that A Level mathematics students did not use in the Maths Made Easy website will be excluded from the program and features that they do use will be made more prominent to the users.

and

<u>Requirement 4</u>: The UI for the Student user account will consist of a navigation bar at the top of the program which allows the user to navigate between the two different Student modules.

All the features which the end user and his students did not use in the *Maths Made Easy* website were purposefully excluded from the program, and a simplified and easy to use UI was implemented that allowed students and teachers to easily navigate between the features for their perspective user accounts.

An MDI container was incorporated that allowed for a tabular interface for both user accounts. This is easy to use for people of all technical abilities due to it being a prominent feature at the top of each form.

However, some parts of the user interface use a coloured image as a background, which slightly complicates the look and feel of the UI. To further simplify its design, a plain white or other neutral colour could have been used in the background image, combined with transparent shapes that blend into the background so as to highlight the mathematical nature of the program.

<u>Requirement 5</u>: The option for the student be able to select a subtopic from a list of available subtopics from within the Select A Subtopic section of the program.

Upon clicking on the module button followed by a topic button a topic, the student will be presented with a list of sub-topics that each contain a quiz consisting of questions exclusively related to that sub-topic.

The ability for a student to select a subtopic by clicking on a series of buttons was fully implemented. A ‘reverse triangle’ structure was created using dynamic button generation. This structure allows students to easily navigate to any one of the eighteen subtopics in just three clicks. When compared with a file system such as the one used in the Windows operating system, this particular method of enabling the student to select a specific subtopic is simpler and more user-friendly because it uses large, obvious buttons that easily capture the user’s attention.

<u>Requirement 6</u>: An easy-to-navigate UI will be created for the Teacher user account that will allow the teacher to easily view a range of statistics collected by the program.
and
<u>Requirement 7</u>: The UI for the teacher user account will consist of a navigation bar at the top of the program which allows the user to navigate between the three different teacher modules.

A tabular system using an MDI container was incorporated into the teacher UI (similar to the system mentioned above for the student UI) which allows teacher to easily navigate between the modules available to them.

As described above for the *Student* UI, the *Teacher* UI could also be simplified to a greater extent to make it feel less busy. Changing the background for the *Create A Class* module to a more neutral one is one possible way to achieve this. Another possible simplification of the

program would be to remove the ‘Search for student by ID’ function, as its only purpose is to provide an alternative method of searching for a particular student’s name to the ‘Add’ and ‘Remove’ buttons. This feature would become more helpful when a greater number of students are added to the class, and as class sizes for A Level Mathematics are relatively small, it therefore does not add significant value to the UX.

Requirement 8: The teacher will be able to select a student from within the Student Overview module.

Upon selecting a student, data gathered on that student by the program will be presented to the teacher in both graphical and numerical forms.
This requirement was fully met. The statistics gathered by the program on the student’s results are presented to the teacher once they select a student name. These statistics are displayed in the following formats:

- A table showing the students percentage score overall, and in each individual module.
- A line graph showing the student’s score against number of attempts.
- A bar chart showing the student’s percentage score for each of the subtopics that fall under the selected topic.

The potential inclusion of a data table that lists the student’s percentage score for each of the assessed subtopics in the program could be implemented in order to improve the overall user experience.

Requirement 9: The menu will also consist of an option to track a class’s progress as a whole and will compare the students in each class with each other.

The statistics will be displayed to the teacher in both graphical and numerical forms to make the data more helpful and easier to understand.

This requirement was partially met. A bar chart and a line graph are used in the program because representing data in a visual format helps to clearly highlight the relationship between different data points (the student’s scores) and is significantly easier to understand than plain text. Both graphs clearly compare the individual performances of each of the students in the selected class and, as such, help the teacher to easily identify which students are excelling at the quizzes and which students are falling behind.

However, whilst programming, I struggled to find a format of presenting the *Class* data numerically in a way that would add value to the user experience. As a result, I have decided to exclude this particular aspect of the requirement from the program, as it would unnecessarily overcomplicate the UI without improving the UX. This helped to ensure that user requirement 5 was fully met.

Requirement 10: A collection of practice questions which assess a wide range of the topics and skills needed for A Level Mathematics are stored in the program folder and

can be accessed by the program whilst it is running. These questions will be used during the quizzes.

Between one and three questions for each of the twelve subtopics are available to be used as questions in the quizzes. This meets the requirement as students can be assessed on every one of the subtopics used in the program, allowing for a full set of statistics to be gathered about the student's performance in A Level Mathematics and then to be output graphically and numerically to the *Teacher* user type.

However, when a student selects a type of subtopic to take a quiz in via the *Select A Subtopic* module, the quiz that is generated only contains between one and three questions. This is an issue, as the initial aim was to have five questions per quiz. In order to fix this, the number of questions for each subtopic should be bought up to five.

Requirement 11: All of the questions in a particular quiz will be randomly generated by the program based off of the subtopics which are viable to be included in the quiz.

This requirement was fully met. Questions were generated using a built-in randomisation function within the Visual Basic programming language. A potential (but perhaps unnecessary) improvement would be to use true randomisation to generate a question name.

Requirement 12: The mark scheme will be presented to the student in the same format as the questions will be, such that the student will be able to cycle between the mark scheme images by clicking on the 'Next' and 'Previous' buttons. The mark scheme will be presented in the same way as the exam board the end user teaches (Edexcel) present their mark schemes.

This requirement was partially met. The student is able to access the mark scheme by clicking on an 'End Quiz' button once they have finished answering the questions in the quiz. Once they have clicked on that button, the mark scheme images are displayed in the picture box in the same order as the questions were (so for example the first image in the quiz is also the first image in the mark scheme). The student is able to navigate between the different mark scheme images using the 'Next' and 'Previous' buttons in the same way that they can cycle through the quiz images.

However, for some of the questions that are used in the program the mark schemes that were written for those questions were not available to be accessed and used. As a result, a mark scheme had to be written for those questions and then saved as a PNG file, so they may not be fully representative of the format in which the exam board Edexcel write their mark schemes in. Furthermore, the formatting for these 'homemade' mark schemes are not perfect, as some of the images have stretched text. This does not take away from the value that the mark schemes add the student's learning experience but does perhaps detract from the appeal of the UI, so using questions with pre-written mark schemes is a possible improvement that could be made.

Requirement 13: Questions will be input into the program in the following ways:

- **Input boxes to input numerical answers for worked questions.**

- **Tick boxes to answer multiple choice questions.**
  and

<u>Requirement 14</u>: When inputting numerical values into the program, the following syntax will be used:

- **Square root: Sqrt(value)**
- **Raising to an exponent: x^(value)**
- **Logarithms: log/base/value, ln/base/value.**

These requirements were fully met. Both multiple choice style questions and questions with an input box are used in the quizzes. The student is able to answer the *Fill In The Missing Blank* style questions using the exact input rules described above. An alternative method that could possibly have been incorporated is to use a button-based system in which the user can click on a button with a particular symbol on it (such as the square root symbol) and then that symbol would automatically be copied over into the input box.

<u>Requirement 15</u>: The font and button size will be large and obvious so that the program is more accessible to users with certain learning difficulties or poor eyesight. Where appropriate, text will be in bold so as to draw attention to certain parts of the program.

This requirement was fully met. The large and clear font combined with the carefully structured layout of the buttons came together to form a simple to use UI. The end user comments below on how he feels that the font and button sized used was an appropriate fit for the program.

5.2 User feedback

5.2.1 Feedback from the end user

In order to gather feedback from the end user regarding the level of success the project achieved, a second interview was conducted. Below is a transcript of that interview.

START OF INTERVIEW……

Joshua Bourton: Overall, would you say that the program was successful at testing students on their mathematical abilities?

Mr Pape: Yes, I think there's a good selection of topics that you can go into, and there are ways of choosing a specific set of questions or having randomised questions across the topics that I like, and a couple of different ways of taking in answers as well. Those different types of answers – multiple choice and one-line answers – they seem to be quite effective at testing students on how well they're doing.

Joshua Bourton: Overall, do you think the user interface is visually appealing? What could be done to improve it?

Mr Pape: Yes, I think it's very clear. The text is a good size and the buttons are clearly laid out across the screen so for instance with the login page it's very clear to understand what you need to do to get into the program. The buttons to navigate to the next page are very clear, submitting an answer to a question is very easy to do and viewing the mark scheme is

simple and effect. I like the way that you can add and remove students from a class with the buttons in that section of the program, it's very well laid out.

Joshua Bourton: Do the graphs in the *Teacher* section of the program give a good insight into answering A Level Maths questions.

Mr Pape: Yes. So, I like that way that you can see how a student has progressed on a particular topic over a certain number of attempts and hopefully see an improvement as they spend more time on the program, so that's a nice feature rather than just having one score and then just having a total at the end. And then having an average of the scores is a really good way of seeing that actually this student is struggling or actually they're doing really well. So I think there are a lot of good features, and the graphical representation of them really did help. Being able to compare the different areas across multiple attempts is really helpful.

Joshua Bourton: Are the style of questions used in the quizzes sufficient? Should there be different styles included such as long answer questions that span multiple lines?

Mr Pape: So yeah, I think that would be a good improvement, to have more types of questions. The problem with that would be how you would mark it, and how you would compare answers with the mark scheme, but having some slightly longer answers would be beneficial. But then its just a matter of how you mark it because that might present some difficulties.

Joshua Bourton: Are there any features that are missing from the program that could have been added?

Mr Pape: Well, there's not many things that you could improve but maybe having a "hints" section, which would include having a button that the student gets stuck that when it's clicked on, a link to a particular website that details information about a particular topic, or maybe a sheet of text that explains the method, so some way of actually students being able to help themselves through some other resources that are linked to a particular question type.

Joshua Bourton: Are there any areas of the program which could be simplified?

Mr Pape: That's a difficult one. Maybe with the multiple choice questions you could select the answer and then that tells you whether you're right or wrong straight away. But I don't know whether that really improves it much, you'd have to look into that one to find out. I think overall I really like the way that you can select a topic. The way that you can branch down to a sub level and then a sub level beneath that, I think that really makes it clear so I changing that part would improve the accessibility of the program to the students. Overall I would say that having the option to take quizzes in different formats is a really good thing.

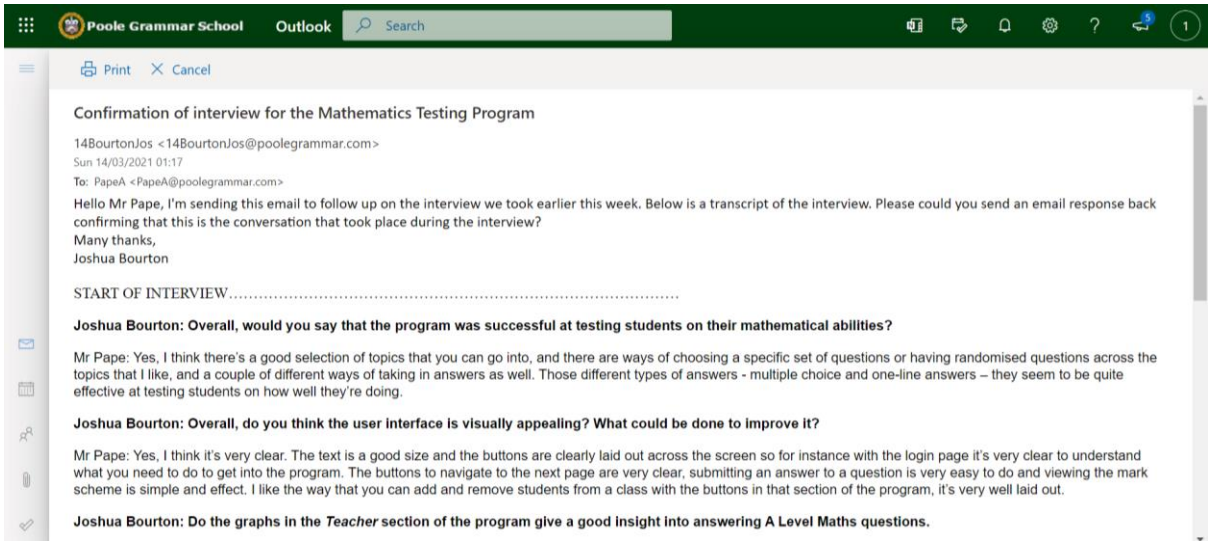
Joshua Bourton: Are there any areas of the program which should be removed?

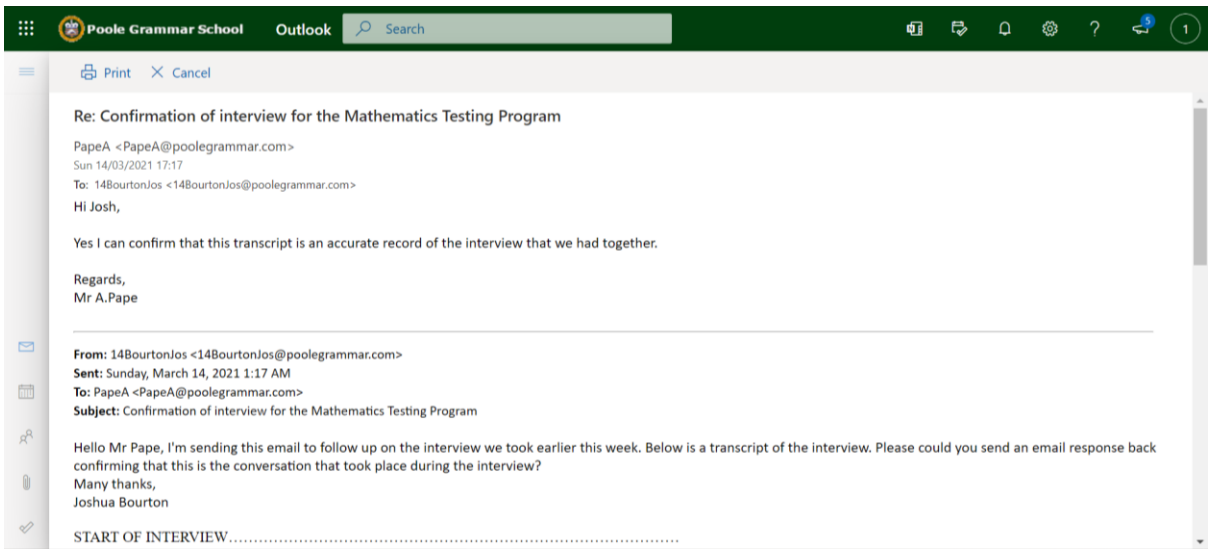
Mr Pape: I don't think so. All the modules seem to add value to the program. If anything, I think that the textbox answers are perhaps not as effective as they could be at catering to every possible answer so that could be an issue if there were questions that had two or more separate answers that were both valid. Otherwise, everything is really good.

END OF INTERVIEW……

5.2.2 Confirmation of the validity of the interview

Below are images of an email thread with the end user in which he confirms that the transcripted interview is an accurate record of the conversation that took place.
The first screenshot depicts the email that was sent to the end user in order to ask for validation that the transcript is representivive of the verbal interview. The second screenshot depicts the end user's repsonse to that email and subsequent confirmation that the transcript is accurate, and also provides evidence that the emails are related and are in the same thread.





5.2.3 Response to the end user's feedback

Overall, the feedback received from Mr Pape with regard to the success of the project at meeting the arranged requirements have been very positive. He has confirmed that the graphs and data generated in the *Teacher* section of the program are helpful in gaining an insight into student performance within the program, and also that the styles of questions used in the *Student* section of the program were sufficient for testing a student's mathematical ability.

Mr Pape was also very positive when discussing the UI. He mentioned that the font and buttons were a good size and that navigating through the program was clear and simple. The UI was designed so that the user could be guided through the program without having to change any settings or click on a back button to revert to the previous page, so it is good to know that this has been achieved. Using large, bold and obvious buttons and fonts helps to make the program more accessible to students with learning difficulties or poor eyesight, so it is also good to know that this has been successfully achieved as well.

The end user has also highlighted some potential changes that could be made to the program in order to increase its functionality and better fulfil its purpose at testing and analysing student's results in the quizzes. Some of these changes would be relatively easy to implement but others may be more difficult to include in the program for the reasons described below.

5.3 Possible improvements

Possible improvement 1 – Incorporating longer, multiple-line questions into the quizzes

One thing that the end user mentioned as a possible improvement was to add a new style of question that spans multiple lines of working. This would help to test different areas of students' mathematical abilities to the areas that are currently tested through the selection of question types available in the program, so incorporating this suggestion could help to increase the effectiveness of the program at testing and improving a student's mathematical ability. A multi-line textbox would be used to implement this suggestion, with the student inputting their answers for each stage in their workings to a question into each line.

The answers to each of these styles of questions would span multiple lines in the information text file (described in *2.3 File handling and data processing*) and each line would be read by the program sequentially when marking the student's answer.

However, as the end user has mentioned, problems may arise when laying out the rules for how this type of question will be marked. For example, there could be multiple methods of achieving the final result when answering the same question, so potentially every possible method of answering a question may need to be considered. This issue could potentially be mitigated by expanding upon the input rules of the program such that students can only answer certain types of questions in specific ways, but this could have a negative impact as students would potentially be deterred from answering a question using a method that would otherwise be valid.

Possible improvement 2 – Adding a 'hints' button

Another thing that the end user suggested adding to the program was a 'hints' button. He stated that, when clicked on, it should take the student to a website that describes how to answer questions on that particular topic. As an alternative to linking to a website, he also stated that a sheet of text containing hints and tips for that particular topic could also be made available to the student. Either of these methods would help to increase the likelihood that the student will be improve their knowledge on the topic, and adding in one of these

options would be a valuable addition to the program. There would also be little difficulty in adding a link to a website from within the program because that can easily be embedded into the text file used by the program to collect information about the question. An information sheet would also be easy to incorporate into the program as well; a textbox could be generated, displaying the text contained within the document.

The possibility of having dynamic objects such as pointers and textboxes within the program could also be explored, so for example if the student clicks on a '*Need a hint?*' button, an arrow could appear and point to a section of the question and a textbox on the other end of that arrow could give the student a hint on how to answer that section of the question.

Possible Improvement 3 – Adding interactive elements to the program

Following on from above, another possible way to make the program more engaging would be to add interactive elements such as video tutorials on how to work through certain styles of questions, which can be accessed by the user clicking a help button. There could be one video for every subtopic, and each video would give a detailed walkthrough of how to go about solving the style of problem that the student is stuck on. However, this would increase the file size of the program significantly and the videos would take a great deal of time to make, so discussion with the end user on how suitable this improvement is would need to take place. The program could potentially contain links to online videos, but this would mean that the program would need internet access in order for it to have full functionality which is a potential downside of incorporating this particular improvement.

The inclusion of audio cues could also benefit the program. For example, a bell sound could chime when the student submits their answer to a question, or an upbeat celebrational noise could sound when the student beats their personal best. There could also be soft background music that can be played and paused from a settings menu. Taken together, these sounds could help to increase the interactivity of the program, thus increasing the amount of time students spend taking quizzes.

Possible improvement 4 – Improving the overall security of the program

As previously mentioned, the program uses a Caesar shift to encrypt the signup data taken in by the program. Due to the low security offered by this particular method of encryption combined with the sensitive nature of the data being encrypted (names and passwords), the end user would potentially be put more at ease if this data had a higher level of security applied to it.
Therefore, using a more complex algorithm for encryption such as a hash function would go a long way in increasing the security of the program and would consequently increase user satisfaction.

Furthermore, the data stored about each student's performances in the quizzes they take is not encrypted due to it having a much less sensitive nature than the program login details. However, encrypting this data as well as the login data would potentially help to increase user satisfaction on the student's end because students may not want their scores for the quizzes they take to be stored by the program without some form of encryption.

Possible improvement 5 – Simplifying the design of the UI

Overall, the end user's comments on the user interface have been positive. However, as mentioned above, the user interface background images used in some of the modules are quite colourful and busy. This potentially detracts from the cleanness of the UI and therefore possibly makes the program more confusing to navigate around than it needs to be. A plain white background could be used but this runs the risk of decreasing user engagement as the user would possibly feel that the program is overly simplistic and boring. So a good compromise would be to use a pre-defined colour palette that is consistent throughout the program and that is comprised of only neutral colours.

Additionally, the possibility of using the outline of basic, semi-transparent mathematical shapes and symbols in the background would help to make the UI feel more mathematical in nature and therefore possibly increase the engagement for the student users.

Possible improvement 6 – Increasing the clarity of the graphs

Another area that could potentially be improved upon is the clarity of the graphs presented in the **Teacher** section of the program. There are several formatting issues that have arisen when discussing the functionality of the graphs with the end user which, when altered, will help to increase the clarity of the graphs.

The first issue that has been identified is that the bar chart generated in the *View Student Progress* module always displays a label that is one level higher than the student's average result in a subtopic. So for example, if a student scores 100% in all of the 'Venn Diagrams' questions they have attempted then the highest label that the graph will display is 120%, which is an impossible score. Therefore, to increase the clarity of the bar chart the issue needs to be addressed by ensuring that the graph always has labels that range from 0%-100%.

The other issue is that not all of the graphs have both axes clearly labelled due to space constraints when designing the UI. Consequently, this makes interpreting the graphs more difficult than it needs to be for the end user. To improve the clarity of these graphs, the UI for **Teacher** section of the program could be slightly remodelled in order to free up space for both axis labels.