# Assessment Brief: Mini Apps

|  |  |  |
| --- | --- | --- |
| **Module Leader:** Tonderai Maswera | | **Level:** 4 |
| **Module Name:** Programming for Computer Science | | **Module Code:** 55-407816 |
| **Assignment Title:** Mini Apps | | |
| **Individual Task** | **Weighting:** 50% | **Magnitude:** 20 hours (notionally) |
| **Submission date/time:**  24th November 2022 at 3pm | **Blackboard submission:** Y **Turnitin submission:** N | **Format:** Single C# file |
| **Planned feedback date:**  15th December 2022 | **Mode of feedback:** Blackboard | **In-module retrieval available:** Yes |
|  | | |
| **Module Learning Outcomes**   * Describe, recognise, and deploy key concepts that relate to designing small imperative and basic object-oriented applications and algorithms. * Describe, recognise and deploy essential features of a mainstream programming language and use them to implement solutions to a variety of programming problems, selecting appropriate control constructs, data structures and objects. * Select and apply appropriate software tools and program testing and validation techniques on small programs. | | |

## Assessment Brief

In this task you will develop a series of mini applications that are designed to get you thinking logically about constructing code in C#. The concepts required to implement these applications form only a small and introductory selection of features found in the language, although you will need to carefully consider how to put those building blocks together. Alongside the code, you will need to provide a testing plan for one of the mini-apps along with code comments.

This task can be tackled using the techniques covered in class, however aspects can be better coded with more advanced features. If you choose to research and deploy those advanced features of the language, then you will receive higher marks (please refer to [Grading Guidelines](#_Grading_Guidelines)).

There are four mini applications to create, alongside a top-level menu system. The following sections outline the core details and requirements.

### Menu System

* Create a top-level menu system that allows the user to select which of the mini applications to run when the application is launched.

P4CS Mini Applications

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

Please select an option:

1) Keep Counting

2) Square Root Calculator

3) Encrypt Text (Caesar Cipher)

4) Decrypt Text (Caesar Cipher)

9) Quit

Please enter option:

* If the user presses a number not in the list of options, display a message and show the menu again
* Once the selected application has finished, the menu should be displayed, and the user asked which app to run (or exit)
* When the user selects the quit option, your code should naturally terminate. **Do not use** System.Environment.Exit() to end your application but do allow control paths to naturally converge at the end of the main method
* **Do not use** any form of recursion to implement the menu system. Use appropriate loops and methods to help split up your code and create a sensible iterative control flow

### Mini App 1: Keep Counting

* Starting from two random numbers, the user is presented with 10 addition or subtraction questions

Keep Counting

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

You will be presented with 10 arithmetic questions. After the first question, the left-hand operand is the result of the previous addition.

Question 1: 9 - 6 =

3

Question 2: 3 + 3 =

6

Question 3: 6 - 3 =

3

Question 4: 3 + 4 =

6

INCORRECT: ANSWER IS 7

Question 5: 7 + 4 =

11

Question 6: 11 + 4 =

15

Question 7: 15 + 9 =

20

INCORRECT: ANSWER IS 24

Question 8: 24 - 9 =

15

Question 9: 15 + 10 =

25

Question 10: 25 + 7 =

32

You got 8 out of 10 questions right

* The [true] answer to the previous question becomes the first operand of the next question (after the first question)
* The type of question (plus or minus) and the second operand to the next question are randomly generated
* The randomly generated operands of the mathematical questions presented to the user should be in the whole number range of 1 and 10 inclusive (answers can be outside of this range though)
* The choice of presenting the user with addition or subtraction should be done randomly for each question.
* You can assume the user enters valid input for the data type expected
* The application keeps a tally of the number of correct answers
* If the user enters an incorrect answer:
  + a suitable message is shown and includes the correct answer
  + the correct answer is used as the first operand of the next question (i.e. not the user’s incorrect answer)
* After the user has answered all 10 questions, they are told how many questions they got right

### Mini App 2: Square Root Calculator

* The user enters a whole, positive number (**as an integer**) and your app will calculate its square root using numerical methods
* The user enters the number of decimal places for the precision square root result (**as an integer**). This should be in the range of 1 to 6 decimal places
* The square root of the input is calculated iterative using a lower and upper bound that represent guesses towards the solution. The lower bound is the last known value that the true square root must be above and the upper bound is the last know value that the true square root must be below. You will need to initialise these with sensible values

Square Root Calculator

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

Please enter a positive number:

5

How many decimal places do you want the solution calculated to:

3

The square root of 5 to 3 decimal places is 2.236

* You refine your lower and upper bounds by:
  + taking the average of the upper and lower bounds
  + squaring that average
  + compare the square of the average against the number we are calculating the square root for
  + update the lower or upper bound accordingly using the average
* You will repeatedly refine your lower and upper bounds until the different between the two bounds is less than your decimal place precision.
  + if the user asks for a precision of 1 decimal place, then your accuracy can be taken as 0.1., for two decimal places, it would be 0.01, for three decimal places, it would be 0.001, and so on.
* Once the level of precision has been reached, output the value of the square root, formatted to the specified number of decimal places
* **Do not use the built-in square root function for this at any point; you must implement this the numerical recipe given above**
* This might seem like a complex task at first but work it through on paper a few times to make sure you know what you are coding. The implementation is reasonably small. You just need to think about this first. Here is a partial worked example (purposely not including the initial values).

| **Input** | 5 | **Decimal Places** | 3 | |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
| **lower** | **upper** | **average** | **average \* average** | **[action]** |
| … | … | … | … |  |
| 2.1875 | 2.34375 | 2.265625 | 5.133056640625 | Update upper |
| 2.1875 | 2.265625 | 2.2265625 | 4.95758056640625 | Update lower |
| 2.2265625 | 2.24609375 | 2.236328125 | 5.001163482666016 | Update upper |
| … | … | … | … | … |

### Mini App 3: Encrypt Text (using Caesar Cipher)

* Input a plain-text message and encrypt it using the Caesar Cipher – See [Caesar cipher - Wikipedia](https://en.wikipedia.org/wiki/Caesar_cipher) for a full description about how this shift cipher works
* The valid alphabet symbols that the user can input are:
  + A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, [space]
  + [space] is the usual space character between words
  + This alphabet is fixed – do not add or remove characters from those given above
* Lower-case characters are accepted as their uppercase versions and share the same character in the alphabet – i.e. if the user enters ‘a’ or ‘A’, they are both taken as an uppercase ‘A’
* If a user provides an input that includes any invalid characters, they are given a message and asked to enter their message again
* After the user has entered a valid input (that has been checked), they are asked for a valid shift
  + This must be in the range of range of 1 and 36 inclusive, which is the extent of our alphabet
* If the shift is out of range, the user is asked to enter another shift until a valid shift value is provided
* The plain-text message is encrypted using the Caesar Cipher and the user-entered shift. The encrypted messaged is displayed to the user

A transcript of the encryption algorithm running is given below:

Example 1:

Encrypt Text

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

Please enter text to encrypt:

HELLO-INVALID\_CHARACTERS?'

Invalid input text

Please enter text to encrypt:

This is my sample message

Please enter shift (between 1 and 36)

-4

Invalid shift amount

Please enter shift (between 1 and 36)

37

Invalid shift amount

Please enter shift (between 1 and 36)

5

Encoded/Decoded string is: 'YMNXENXER3EXFRUQJERJXXFLJ'

Example 2:

Encrypt Text

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

Please enter text to encrypt:

One is 1 and 2 is Two

Please enter shift (between 1 and 36)

15

Encoded/Decoded string is: '32TOX7OFOP2SOGOX7O8 3'

### Mini App 4: Decrypt Text (using the Caesar Cipher)

* Provide a decryption application that takes a Caesar Cipher-encoded message and decrypts it
* The user inputs and validations should follow the same structure as those outlined in the Encrypt Text mini-app (i.e. check no invalid characters exist in the message and a valid shift is given in the range of 1 and 36 inclusive)
* Refactor and reuse code between this mini-app and the encryption algorithm that is common between the two applications.

A transcript of the decryption algorithm running is given below:

Example 1:

Decrypt Text

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

Please enter text to decrypt:

#5672fgdrfghf

Invalid input text

Please enter text to decrypt:

N66NPXM10M6URMVQR5M1SMZN4PU

Please enter shift (between 1 and 36)

-2

Invalid shift amount

Please enter shift (between 1 and 36)

40

Invalid shift amount

Please enter shift (between 1 and 36)

13

Encoded/Decoded string is: 'ATTACK ON THE IDES OF MARCH'

Example 2: (remember, without the correct shift, the message will not decode correctly)

Decrypt Text

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

Please enter text to decrypt:

N66NPXM10M6URMVQR5M1SMZN4PU

Please enter shift (between 1 and 36)

20

Encoded/Decoded string is: '4MM46D3HG3MA83B78L3H93F4K6A'

## Testing

Provide a test plan for the Square Root Calculator (the table used above to demonstrate the algorithm is not a test plan – it just shows internal working steps). **Put this above the main function for that app as a comment block.**  The test plan should include inputs, expected outputs, actual outputs, and should be sufficient to ensure your code is tested to a suitable level.

## General Principles, Code Development and Hints

* While you can assume the user enters input valid for the data type expected, you will obtain marks for researching and implementing techniques that guards against invalid input types. Minimally, you should check for range correctness (e.g. the menu system should give an error when “123” is entered)
* **Use sensible methods and functions** to divide the complete application up into logical sections. There is a minimum expectation that each of the mini-apps will be in their own method/function, which itself may use more functions.
* Avoid repeating sections of code by **reusing methods and functions** where possible (for example, to get valid input from the user in a specified range)
* **Avoid all global variables** – all variables should be declared within the scope of a method or function or further nested construct
* You will get credit for what you have achieved so do not worry if you don’t get it all working
* **Comment your code** to an appropriate level, use sensible naming and make the code readable using a consistent and sensible layout.
* There is no single right solution, but some code is better than others… think about what you are doing, what you want to achieve and implement it in a sensible, robust and efficient way
* **Evolve your code and regularly test what you have written**; when we put together applications, we do not sit down and write it in one go and expect it to run… we build the application in small stages, progressively adding in functionality and testing and debugging as we go (yes, everyone needs to test and debug code they write no matter how long they have been coding for so get practiced at doing so)

## Grading Guidelines

The marking of this task is done based on your ability to successfully choose and apply appropriate programming structures to the complete coding task. You will need to fully implement the require functionality to obtain the highest marks, although marks will still be awarded for partial solutions and subject to the level of completion.

Marks will be awarded against the following areas of competence

| **Area of Competence** | **Description** | **Percentage Available** |
| --- | --- | --- |
| Data types | Appropriate selection of base types to suit the data being stored and manipulated. Avoiding use of global variables. | 10% |
| Control Structures | Sensible control structures have been selected and demonstrated: if, elseif – for-loops, while loops and do-while loops. Sensible use of break | 30% |
| Use of methods | Methods have been used to break the code down into sensible sections, including using return value and parameters | 15% |
| Conciseness of code | Have you written your code in such a way that you are not unnecessarily repeating code, and have you written it concisely – the more convoluted the solution, the more code you will need to write and the more likely it is you will have errors and find it harder to maintain. Keep it simple. Do control paths naturally converge? Avoid having a contrived runtime pathway through your code. | 10% |
| Testing | A test plan (expected output against input across a sensible choice of inputs) | 15% |
| Defensive coding | Coding to guard against invalid user input – from basic if-test to try and catch, etc | 10% |
| Formatting | Meaningful names for methods, variables, and consistent use of case. Consistent formatting including indentation and appropriate use of whitespace (new lines). Use of code comments – this should include introducing what each method does at the start and inter-code comments | 10% |

Each area of competence will be graded using the [University level 4 Grade Descriptor](#_University_Grade_Descriptors) against the (0 – 16) grade point scale and categories. Your final mark will be calculated using the weightings given in above table and projected into a grade point category. Please refer to the assessment criteria rubric at the end of this document, which reproduces the level 4 grade descriptor. Standard university grading is being used for this task. The marking scheme embeds the concept of extended work by rewarding the highest marks to those who demonstrate evidence of independent investigation, learning, critical thought, and problem analysis (via good code solutions).

## Submission Process

Your assignment should be submitted electronically through the module Blackboard site as a **single, uncompressed .cs file**. This will ensure you can preview the format and submission within the Blackboard preview window.

Your last on-time attempt will be viewed and graded (as per university regulations).

Make sure that you upload the correct file by checking once you have submitted (i.e. preview your submission in Blackboard and check it and/or download it again from Blackboard). Mistakes discovered after the deadline cannot be corrected; it is your responsibility to ensure that you submit the correct files by the deadline. You may be asked to provide a walkthrough of your code during which you will need to discuss all aspects of the work you submitted with your grade being subject to a successful walkthrough and discussions of your work.

## Avoiding Plagiarism

Looking for snippets of code on the internet and searching for materials that will help solve parts of a problem is part of software development. If you do this, you must add a comment in your code stating:

* the URL for snippets of code that have heavily influenced your solution and
* a short summary of what you have taken / learnt to demonstrate you understand

You should not directly copy from the internet (or elsewhere), even once referenced; make the code your own. There is no need to reference core learning that you apply to this task – for example, if you have learnt how to use lambda expression and applied it here, that is fine.

This assessment is about you demonstrating your knowledge and understanding when it comes to programming and not the internet’s, your friend’s, or Discord’s. If you try to pass off work as your own that isn’t, then an academic conduct meeting will be arranged, and you’ll likely get zero marks. A second offense can lead to withdrawal from the course, so please ask if you’re unsure.

# [University Grade Descriptors – Level 4](https://students.shu.ac.uk/regulations/assessment_awards/University%20Grade%20Descriptors%20(Level%204;%20new%20Level%207).pdf)

