2800ICT Assignment Milestone (30 marks)

Note: Try to make the best use of appropriate C++ features.

Encryption and Decryption System

This assignment requires you to independently solve problems and develop software that **integrates various aspects of C++ programming covered in this course from Week1 to Week7**. This program should maximize the use of object-oriented C++ programming and appropriate modern C++ features.

1, Task Description

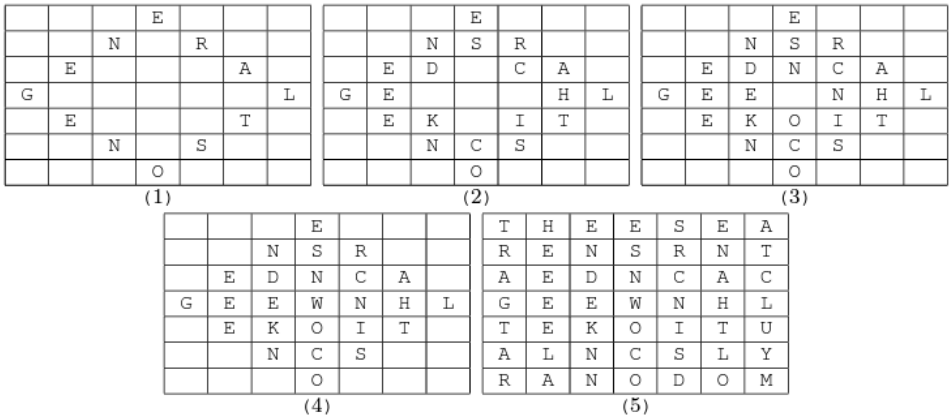
You are tasked with developing a C++ program capable of performing both encryption and decryption using a custom diamond-shaped grid algorithm. The encryption process begins by inserting the message into a square grid, following a specific diamond traversal pattern. Starting at the middle cell of the leftmost column, the message is written diagonally upwards to the top edge, then continues diagonally downwards to the right edge, forming a diamond shape. Once the outermost diamond is filled, the process repeats inwardly with successively smaller diamonds until the message ends. Any remaining empty cells—either outside or inside the final diamond—are filled with random uppercase letters (A–Z) to mask the actual content. The encrypted message is then produced by concatenating all characters column by column, from left to right and top to bottom within each column.

The grids below, with **a grid size of 7**, illustrate the steps involved in one-round encryption of the message (ignoring the blank spaces):

GENERAL TSO NEEDS CHICKEN NOW

The first grid shows the encoding after the first diamond has been filled in. The second, third and fourth grids show successively smaller diamonds being filled in. Finally, the last grid shows the final message after random letters have been placed in the empty squares around the grid. The encrypted message will be

TRAGTARHEEEELAENDEKNNESNWOCOSRCNISDENAHTLOATCLUYM

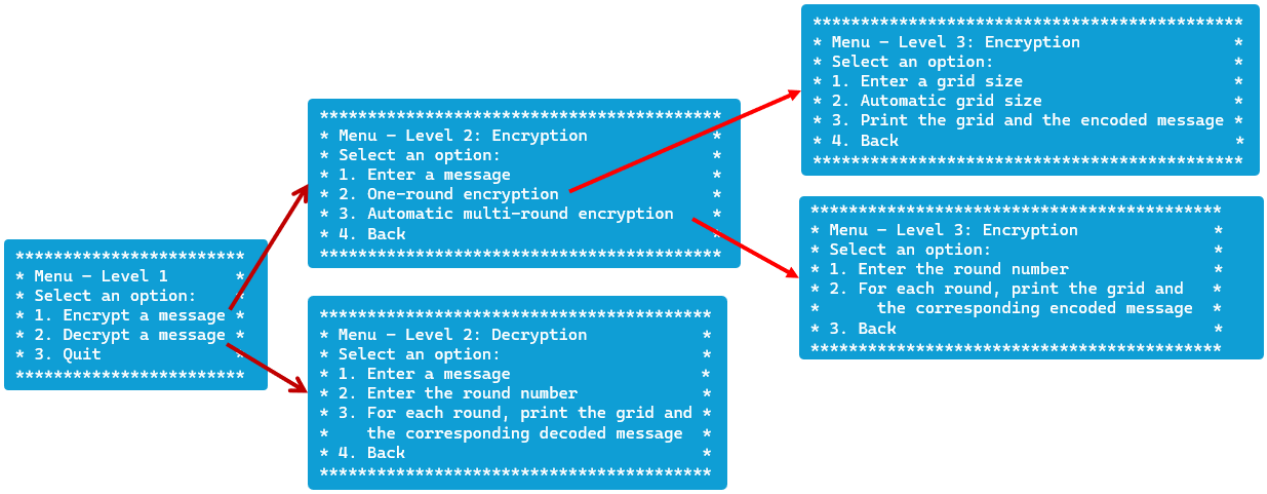


To indicate the end of the original plain message, a full stop (i.e., **.**) should be appended at the end if it is not already included. This appending should occur only in the first round, if necessary. This ensures that during decryption, the program can accurately identify where the message ends, even if random filler characters remain. The length of the encrypted string would be a perfect square of an odd integer (e.g., 49, 81, 121), forming a square grid suitable for both encryption and decryption.

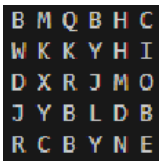
Your program must also support multi-round encryption and decryption: the output of the first round is fed as input into the second round, and so on, up to a user-specified number of rounds N. This number will serve as a secret shared between sender and receiver and must be used during decryption to accurately reverse the process.

2, Other Implementation Requirements:

- Your program should maximize the use of object-oriented C++ programming and appropriate modern C++ features, integrating proper key C++ concepts/skills/knowledge covered in the course from Week 1 to Week 7.
- Properly design custom classes and their relationships to ensure a clear and logical program structure. The program must include a minimum of **three classes**.
- Your program must implement **the below textual user interface**. After executing each selected option, a proper menu should be displayed.



- A grid could be displayed like below. You are welcome to design a more visually appealing format – surprise us!



- This program **can always continue**, even if a user enters any incorrect data, **until the user quit it**. This means that the system should be able to handle various exceptions that may occur during execution.
- Your program should be able to remove spaces in the input message and add a full stop when appropriate.
- Be well written
  - sensible variable names
  - no too long functions (less < 50 lines of C++ code) except for the main function
  - appropriate comments.
- Please refer to **"3, Submit - 3, week11.pdf or week11.docx"** below for additional requirements related to the document.

3, Submit:

1, **week11.cpp or week11.zip**

- If your code is organized into separate files, please compress all *.cpp and *.hpp files into a single ZIP file and submit it.
- Organizing your code into separate files is recommended but not mandatory.

2, **week11.txt**

- This file must be submitted **separately** and **should not be** included in a ZIP file.
- A .txt file containing all source code must be included for plagiarism check. *(If you are unable to create the .txt file correctly, please configure your system to display file extensions.)*

3, **week11.pdf or week11.docx**

- The file must be submitted **separately** and **should not be** included in a ZIP file.
- **The PDF or DOCX file must contain clearly labelled sections**:
  - **Section 1: UML**

    Your UML diagram should provide a clear overview of all classes used in your program. Each class should include the following information:

    (You are encouraged to use the free online tool https://app.diagrams.net for UML creation – a simple tutorial https://www.drawio.com/blog/uml-class-diagrams. Hand-drawn diagrams are also accepted.)
    - Inheritance/aggregation relationship if applicable
    - The name of the class
    - The name and data type of each member variable
    - For each member function
      - ✓ Its name

- ✓ Its return data type
- ✓ For each parameter: its name and data type

- All member variables and functions should clearly show their visibility (i.e. private(-)/public(+)/protected(#))

- **Section 2: Normal Execution**

  Include screenshots demonstrating the normal execution of **each option** in the textual user interface across **all menu levels** for the following three scenarios.
  - ***Scenario 1***: Perform **one-round encryption** on **your full name** using a manually entered valid grid size. Then, decrypt the corresponding encrypted message. *It should include screenshots demonstrating the entire process.*
  - ***Scenario 2***: Perform **one-round encryption** on **your full name** using an automatic grid size. Then, decrypt the corresponding encrypted message. *It should include screenshots demonstrating the entire process.*
  - ***Scenario 3***: Perform **automatic multi-round encryption** with **three** rounds on **the message below**. Then, decrypt the final encrypted output. *It should include screenshots demonstrating the entire process.*

    `I ENJOY THIS COURSE.`
- **Section 3: Abnormal Execution**
  - This program should be able to **always continue**, even if a user enters any incorrect data, **until the user quit it**. This section should include screenshots showing how the program handles **various abnormal inputs**, at least **three** abnormal inputs, for ***every option that requires user input*** at **each menu level**, e.g.,
    - ✓ empty input
    - ✓ wrong type
    - ✓ not desired input
    - ✓ etc.
- **Section 4: Application of Course Concepts/Knowledge from week1 to week7**

  List the relevant C++ concepts and skills from Weeks 1–7 that are effectively used in your program. **They must be organized in the following format**:

  Week 1: xxx, xxx, …
  Week 2: xxx, xxx, …
  Week 3: xxx, xxx, …
  … …

Please refer to the submission page for the Marking Rubric.