You can complete this assessment after going through chapters 1-5 of Ezust - the recommended book.

Question 1

Write a class named SentenceProcessor to process a sentence as depicted in the UML class diagram below:

The function implementations should achieve the following:

- getWordNumber() takes a sentence and returns the number of words in it. You may assume that words are separated by spaces in a sentence.
- getVowelNumber() takes a sentence and returns the number of vowels in it.
- isReversible() takes a sentence and returns true if the sentence is the same with the words

  reversed in it. An example of a reversible sentence is “All for one and one for all”. On the other

  hand the sentence, “I am Sam” is not a reversible sentence.

- wordsReversed() takes a sentence and returns a sentence with the words reversed in it. For

  example, when the words are reversed in the sentence “I am Sam”, then it will read “Sam am I”.

- formatSentence() takes a sentence and returns a sentence with the first letter in upper case

  and one period at the end of the sentence. It should not add a period if the original sentence already has a period.

  Make sure you reuse functions of Qt classes such as QString and QStringList to realise the functions in SentenceProcessor.

Test SentenceProcessor in a main()function by requesting the user to enter a sentence using aQInputDialog. The output of the all the operations should be displayed to the user using a QMessageBox. Give the user a choice to test other sentences before quitting the application. Ensure that the user enters at least two words as input.

Question 2

Consider the following code defining a composition relationship between two classes:

```
class Squab {
private:

int score;

    Position pos;
public:
    Squab(int x, int y);
    int getScore() const;
    void incScore();
    Position getPos() const;
    void move(char dn, int de);
    bool lineOfSight(Squab s) const;
    Squab * clone() const;

};

class Position {
private:

int x;

int y;

public:
    Position(int x, int y);
    void changePos(int x, int y);
    int getX() const;
    int getY() const;

};
```

A Squab is meant to represent a creature in a computer game that is played on a rectangular grid. Positions on the grid are specified by two coordinates (x and y). Directions are represented by four letters: N, S, E and W. Movement in direction E causes an increase in the x coordinate, and W a decrease. Movement in direction N causes an increase in the y coordinate, and S a decrease.

NB: The rules of the game are not important. You are NOT required to write a program to play the game. You must simply write a program to test the implementations of the Squab and Position classes.

Firstly, implement the classes as follows:

- The Squab constructor should initialise the score to 0 and the position to the specified coordinates.
- getScore() should return the score, and incScore() should increment it by 1.
- getPos() should return a copy of the current position.
- move() should move the instance in a straight line in a given direction a given distance.
- lineOfSight() checks whether the current instance is in the same row or column as another Squab instance.
- clone() creates (and returns a pointer) to a new instance of Squab with the same position and score as the current instance.
- The member functions of Position are self-explanatory.

Note that the score data member and the incScore() member function are intended to allow a Squab instance to accumulate a score. The situations in which the score is increased are part of the rules of the game and are not important for this exercise. Similarly, the situations in which Squabs may be moved or cloned are not important for this exercise. You must just make sure that the member functions do what is specified above.

Next, write a console app to test all the member functions of the Squab class. It should create a single instance of Squab and then clone it twice, testing whether the cloning works properly.

Finally, change the Squab class so that the pos data member is a pointer to a Position. You will also need to implement the Big Three (copy constructor, assignment operator and destructor) for the Squab class. (See the additional notes on pointer data members in the comments on Section 2.14 in Tut Letter 102.) No changes to the Position class are required.

Test the changed Squab class with the program you wrote previously.

You must submit the following:

The header and source files for the (changed) Squab and Position classes. The testing program (containing a main() function).

A Qt project (.pro) file for all this code.