# Data Structures and Algorithms (AI) CA1 Assignment Report

Name: Ethan Tan

Admin: P2012085

Class: DAAA/2B/03

## Description

This report entails the documentation of a Morse Code Analyser program written in the Python programming language. The program has been designed to be user-friendly and configurable. It supports the encoding of plain text to morse code, printed either horizontally or vertically, and the decoding and opinionated analysis of a message written in morse code.

## User Guidelines

When run, the program will display a menu which offers users 4 options to choose from:

Text

Description automatically generated

As indicated above, the 4 choices are:

### Change Printing Mode

This option allows users to switch from horizontal printing mode to vertical and vice versa.



The default print mode is horizontal. Users can select horizontal/vertical mode by entering h/v respectively. Whitespace is ignored and the user’s input is case insensitive.



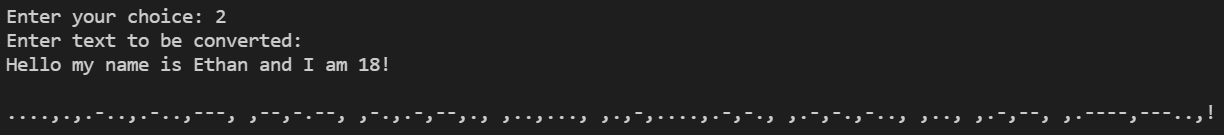
If the user does not enter any input, the current printing mode will be retained.



### Convert Plain Text to Morse Code

This option encodes a plain text message to morse code and prints it out;

Either horizontally:



A picture containing text, night sky

Description automatically generatedOr vertically:

A picture containing text, computer, night sky

Description automatically generated

Multi-line input is also supported for both printing modes:

Text

Description automatically generated with low confidence

*\*Note: Do not include periods (.), dashes (-) or commas (,) as these are symbolic to the function*

### Analyse Morse Code Message

This option first decodes a morse code message in a file, then provides a breakdown of the frequencies of the words therein and constructs an essential message therefrom with the stop words filtered out.

The user will be prompted to enter an input file which contains the morse code message and an output file to write the report of the analysis to. If the input file does not exist or the output file name is invalid, the user will be prompted again correspondingly. If the user does not provide an output file, the report will not be saved.

Text

Description automatically generated with medium confidence

If the input file’s contents are in an invalid format, the process will be aborted, and the user will be redirected to the menu screen.

Text

Description automatically generated Text

Description automatically generated

The decoded morse text will be displayed, followed by a breakdown of the individual words in the message [encoded word; frequency; positions – (line, word)], followed by the essential message.

## Data Structures Implemented

##### Morse\_Code\_Analyser

#### Description:

This is the class for the main object to be run in the main program. It integrates all the core operational functionality mentioned above into one centralised object.

#### Purpose:

* Centralises all the major functionality related to operation of the program

#### Properties:

* \_\_author (contains details of the author)
* \_\_print\_mode (default printing mode)
* \_\_min\_significant\_frequency (minimum frequency of words to be shown in Option 3)

#### Methods (Only time complexities are analysed):

* run (the only public method)
* \_\_change\_printing\_mode\_1 [O(i); i = frequency of invalid input]
* \_\_convert\_text\_to\_morse\_2 [O(s); s = size of input]
* \_\_analyse\_morse\_message\_3 [O(i + o + s *log*(s)); i = frequency of invalid input files, o = frequency of invalid output files, s = size of input file’s contents]
* \_\_exit\_4 [O(1)]

*\*Note: size of input is the total length of the flattened input*

#### Reasons:

* Properties are all **encapsulated** as they should not be changed after instantiation
* Methods (besides run) are all **encapsulated** as they are for internal use only

#### Challenges Faced:

##### Morse\_Utils

#### Description:

* This is an abstract class which contains all the functionality related to morse-to-plaintext translation. It cannot be instantiated, and its methods are all static.

#### Purpose:

* Centralises all the functionality related to morse-to-plaintext translation

#### Methods:

* encode\_morse [O(s); s = size of the plaintext message]
* decode\_morse [O(s); s = size of the morse message]

*\*Note: size of input is the total length of the flattened input*

#### Reasons:

* Class is abstract as it is merely a namespace to hold correlated functionality
* Methods are public as they are meant for external use

#### Challenges Faced:

##### Abstract\_Stack

#### Description:

* This is an abstract interface for the Stack class. It merely declares what methods the Stack class should support

#### Purpose:

* Ensure that the Stack class supports all necessary functionality

#### Methods:

* push [O(1)]
* pop [O(1)]
* peek [O(1)]
* empty [O(1)]
* size [O(1)]

#### Reasons:

* Methods are all abstract as they are meant to be overridden by the Stack class

#### Challenges Faced:

##### Stack

#### Description:

* This is a generic Stack class, which support the common push, pop, peek, empty and size methods at O(1) time complexity

#### Purpose:

* Data structure with the LIFO constraint

#### Properties:

* \_\_head (top item of stack)
* \_\_size (size of the stack)

#### Methods:

* \_\_len\_\_ [O(1)] (alias for size method)
* \_\_iadd\_\_ [O(1)] (alias for push method)

*\*Note: with the addition of all the methods declared in Abstract\_Stack*

#### Reasons:

* **Polymorphism** is implemented as method overriding of methods in Abstract\_Stack
* **Polymorphism** is also implemented as operator overloading for the alias methods
* Properties are **encapsulated** as they are meant to be for internal use

#### Challenges Faced:

##### Morse\_Character

#### Description:

* This is the class which is used to embody encoded text for vertical printing as in Option 2. This class inherits the Stack class

#### Purpose:

* Provide a logical data structure to meet the need for vertical printing

#### Properties/Methods:

* \_\_init\_\_ [O(1)] (loads a morse character in string format)

*\*Note: with all the properties and methods as defined in the Stack class it inherits from*

#### Reasons:

* Vertical printing is implemented using a list of Morse\_Character objects, as the morse characters are printed from the back to front – adheres to LIFO principle
* Morse\_Character class is used instead of the generic Stack class as it makes loading the string morse characters easier and more explicit

#### Challenges Faced:

##### Word

#### Description:

* This is an abstract class which is meant to model a generic word in the morse code message for Option 3

#### Purpose:

* Model a word object

#### Properties:

* \_word (word in string form)
* \_frequency (number of occurrences in message)

#### Methods:

* size [O(1)]
* getWord [O(1)]
* getFrequency [O(1)]
* addInstance [abstract]
* \_\_lt\_\_ [abstract]

#### Reasons:

* Properties are prefixed with single underscores as they are meant to be **protected** (private but inheritable)
* Abstract methods are to be overridden in subclasses, Message\_Breakdown\_Word and Essential\_Message\_Word

#### Challenges Faced:

##### Message\_Breakdown\_Word

#### Description:

* This is the class of the word objects specific to the get\_message\_breakdown method in Morse\_Code\_Analyser class

#### Purpose:

* Provide an easy way to compare words for sorting

#### Properties:

* \_\_all\_positions (stores all occurrences of the word in the message)

#### Methods:

* addInstance [O(1)] (register a new instance of the word, noting its position in the message)
* getDetails [O(w + f); w = length of word, f = frequency of word] (return details on the word)
* \_\_lt\_\_ [O(1)] (compares frequency descending, followed by size, followed by word)

#### Reasons:

* **Overloaded** < operator enables easy sorting of words by the conditions as stated in the brief
* getDetails collects all the necessary details and returns it to be inserted in the report

#### Challenges Faced:

##### Essential\_Message\_Word

#### Description:

* This is the class of the word objects specific to the get\_essential\_message method in Morse\_Code\_Analyser class

#### Purpose:

* Provide an easy way to compare words for sorting

#### Properties:

* \_\_first\_position (stores the initial position of the word in the message)

#### Methods:

* addInstance [O(1)] (increments frequency by 1)
* getFirstPos [O(1)] (getter for the \_\_first\_pos property)
* \_\_lt\_\_ [O(1)] (compares frequency descending, followed by first position)

#### Reasons:

* **Overloaded** < operator enables easy sorting of words by the conditions as stated in the brief

#### Challenges Faced:

## Algorithms Implemented

##### QuickSort

#### Description:

* This is an implementation of the commonly known quicksort algorithm, which performs efficiently for large lists through a divide-and-conquer approach

#### Purpose:

* Sort the list of word objects efficiently

#### Time Complexity (Average/Best):

* O(n *log*(n)); n = the number of items in the list

#### Reasons:

* Quicksort is used to sort the lists of Message\_Breakdown\_Word/Essential\_Message\_Word objects instead of SortedLists as it is more efficient
* SortedLists use insertion sort over many insertions to maintain state of being sorted, which has average time complexity of O(n­2)

#### Challenges Faced:

## Inheritance Tree

Text

Description automatically generated