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:

```
*********************************************************
*       ST1507 DSAA: Morse Code Message Analyser        *
*-------------------------------------------------------*
*                                                       *
*       - Done By: Ethan Tan (2012085)                  *
*       - Class: DAAA/2B/03                             *
*                                                       *
*********************************************************
Please select your choice (1, 2, 3 or 4):
               1.  Change printing mode
               2.  Convert plain text to morse code
               3.  Analyse morse code message
               4.  Exit
Enter your choice:
```

As indicated above, the 4 choices are:

1. Change Printing Mode

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

```
Current print mode is h
Enter 'h' for horizontal or 'v' for vertical, then press enter:
```

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.

```
Enter 'h' for horizontal or 'v' for vertical, then press enter: h
The print mode has been changed to horizontal
```

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

```
Enter 'h' for horizontal or 'v' for vertical, then press enter:
Operation cancelled by user. The print mode remains as horizontal
```

2. Convert Plain Text to Morse Code

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

Either horizontally:

```
Enter text to be converted:
SOS WE NEED HELP

...,---,...,  ,.--,.,  ,-.,.,.,-..,  ,....,.,.-..,.--.
```

Or vertically:

```
SOS WE NEED HELP


          . ..
.-. .     - . --
.-. -   -   . . .-
.-. -. .... ....
```

Multi-line input is also supported.

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

3. 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. 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.

```
Enter your choice: 3
Enter input file:  abc
Invalid input file
Enter input file:  __main__.py
Morse code in file __main__.py is in invalid format. Aborting...
```

```
Enter input file:  data/morse_sos.txt
Enter output file: report.txt

*** Decoded morse text
SOS SOS OUR SHIP IS SINKING
PLEASE HELP

*** Morse words with frequency = 2
...,---,...
[SOS] (2) [(0, 0), (0, 1)]

*** Essential Message
SOS SHIP SINKING HELP
```

report.txt

```
1   *** Decoded morse text
2   SOS SOS OUR SHIP IS SINKING
3   PLEASE HELP
4
5   *** Morse words with frequency = 2
6   ...,---,...
7   [SOS] (2) [(0, 0), (0, 1)]
8
9   *** Essential Message
10  SOS SHIP SINKING HELP
```

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

A. 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/Improvements Made:

- Code was distributed into modules to improve structure and design of the application, and reduce repetitiveness

B. 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-text 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/Improvements Made:

- This class did not originally exist
- It improves organisation of code by using a "namespace" to group related functions

C. 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/Improvements Made:

- None

D. 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/Improvements Made:

- I had problems overloading the __iadd__ magic method
- The solution was to return self

E. 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/Improvements Made:

- I realised that I had the wrong impression of vertical printing – I thought it was bottom up when it was actually top down
- Solution was to reverse the word string in the constructor

F. 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/Improvements Made:

- Challenge was to overload the less than operator in 2 different ways
- Initial solution was to implement a custom key when sorting
- Final solution was to add 2 subclasses (Message_Breakdown_Word and Essential_Message_Word)

G. Message_Breakdown_Word

Description:

- This is the class of the word objects specific to the __get_message_breakdown method

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/Improvements Made:

- Traded space complexity for time complexity by keeping track of an Essential_Message_Word alongside a Message_Breakdown_Word in __get_frequencies

H. Essential_Message_Word

Description:

- This is the class of the word objects specific to the __get_essential_message method

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_position 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/Improvements Made:

- Traded space complexity for time complexity by keeping track of a Message_Breakdown_Word alongside an Essential_Message_Word in __get_frequencies

Algorithms Implemented

A. 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

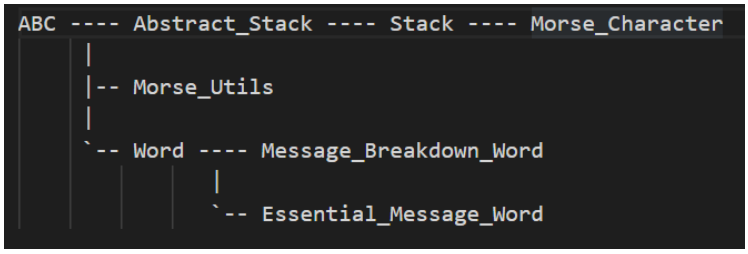
Reasons:

- 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 $O(n^2)$

Challenges Faced/Improvements Made:

- Initially, the list was sorted in-place
- Afterward, it was changed to return a sorted copy instead
- Had to tune the boundaries a few times to get partition working right

Inheritance Tree



Appendix

1. morse_code_analyser.py

```
# Name:     Ethan Tan
# Admin:    P2012085
# Class:    DAAA/2B/03

# Import Dependencies
from typing import Dict, List, Literal, Set, Tuple, Union
from .data_structures.morse_character import Morse_Character
from .data_structures.message_breakdown_word import Message_Breakdown_Word
from .data_structures.essential_message_word import Essential_Message_Word
from .sorting.quicksort import quicksort
from .utils.io_utils import strip_special_characters, simple_input,
multi_line_input, file_input, clear_console
from .utils.morse_utils import Morse_Utils
from os.path import exists, isfile


# Main object
class Morse_Code_Analyser:
    def __init__(self, config=None):
        self.__load_config(config=config)

    # Load custom configuration options
    def __load_config(self, config=None):
        DEFAULT_CONFIG = {
            'author': {
                'name': 'John Doe',
                'admin': '1234567',
                'class': 'UNKNOWN',
                'module': 'UNKNOWN',
            },
            'print_mode': 'h',
            'stopwords_file': 'data/stopwords.txt',
            'min_significant_frequency': 1
        }

        # Defaults to default configuration if key is not specified in
config
        CUSTOM_CONFIG = DEFAULT_CONFIG.copy()
        if config is not None:
            for k, v in config.items():
                CUSTOM_CONFIG[k] = v

        # Details to be displayed
        self.__author: Dict[str, Dict[str, str]] = CUSTOM_CONFIG['author']
```

```
        # Default printing mode
        self.__print_mode: Union[Literal['h'],
                                 Literal['v']] = CUSTOM_CONFIG['print_mode']

        # Minimum lowest frequency to display in the get_message_breakdown
method in Option 3
        self.__min_significant_frequency: int =
CUSTOM_CONFIG['min_significant_frequency']

        # Relative path to file containing stop words
        self.__set_stop_words(file=CUSTOM_CONFIG['stopwords_file'])

    # Operational Methods
    # Runs the program
    def run(self):
        choice = 0
        while choice < 4:
            clear_console()

            # Displays the author's information
            self.__print_info()

            # Get the user's choice
            choice = self.__get_choice()

            if choice == 1:
                self.__change_printing_mode_1()
            elif choice == 2:
                self.__convert_text_to_morse_2()
            elif choice == 3:
                self.__analyse_morse_message_3()
            elif choice == 4:
                self.__exit_4()
                break
            input('\nPress Enter to continue...')

    # Allows the user to change the printing mode for Option 2
    # Empty input will return to the menu without changing the printing mode
    def __change_printing_mode_1(self):
        mode = self.__print_mode
        modes = {
            'h': 'horizontal',
            'v': 'vertical'
        }
        print(f'Current print mode is {mode}')
        while True:
            try:
```

```
                mode = simple_input(
                    f'Enter \'h\' for horizontal or \'v\' for vertical, then
press enter: ')
                assert mode == 'h' or mode == 'v' or mode == ''
                break
            except Exception:
                print('Invalid input')
        if mode != '':
            self.__print_mode = mode
            print('The print mode has been changed to', modes[mode])
        else:
            print('Operation cancelled by user. The print mode remains as',
                  modes[self.__print_mode])

    # Converts a (multi-line) message in plain text to morse code and prints
it out
    #   based on the current printing mode
    def __convert_text_to_morse_2(self):
        print('Enter text to be converted:')
        lines = multi_line_input()
        morse = Morse_Utils.encode_morse(lines)
        if self.__print_mode == 'h':
            self.__print_morse_h(morse)
        else:
            self.__print_morse_v(morse)

    # Converts a morse code message in a file to plain text
    # Displays a breakdown of the frequencies of each word in the message
    # Displays the essential message, with stop words removed
    def __analyse_morse_message_3(self):
        try:
            input_file = self.__get_input_file()
            decoded_text = self.__get_decoded_message(input_file=input_file)
            output_file = self.__get_output_file()
            stripped_decoded_text = strip_special_characters(decoded_text)
            message_breakdown_ls, essential_message_ls =
self.__get_frequencies(
                text=stripped_decoded_text)
            message_breakdown = self.__get_message_breakdown(
                word_ls=message_breakdown_ls)
            essential_message = self.__get_essential_message(
                stop_words=self.__stop_words, word_ls=essential_message_ls)
            report = self.__build_report(
                decoded_message=decoded_text,
message_breakdown=message_breakdown, essential_message=essential_message)
            print('\n', report, sep='')
            self.__save_report(message=report, file=output_file)
        except AssertionError as err:
```

```
            print(err, 'Aborting...')

    # Displays a friendly farewell message
    def __exit_4(self):
        print('Bye, thanks for using {}: Morse Code Analyser!'.format(
            self.__author['module']))

    # Utility Methods

    # Option 2

    # Prints the encoded string horizontally
    @staticmethod
    def __print_morse_h(morse: str):
        print(morse)

    # Prints the encoded string vertically
    @staticmethod
    def __print_morse_v(morse: str):
        lines = morse.splitlines()
        for line in lines:
            ls = []
            # A list is used to collect the characters as
            #   repeated appending has an amortized worst case time
complexity of O(n) whereas
            #   repeated string concatenation has a time complexity of
O(n^2)
            print_ls = []
            for char in line.split(sep=','):
                ls.append(Morse_Character(
                    morse_char=char, pad_char=' ', padding=5))

            # Each morse character has a maximum length of 5
            for _ in range(5):
                for char in ls:
                    print_ls.append(char.pop())
                print_ls.append('\n')
            print(''.join(print_ls))

    # Option 3

    # Loads the stop words from the file specified
    def __set_stop_words(self, file: str):
        with open(file=file, mode='r') as f:
            stop_words = {w.upper() for w in f.read().splitlines()}
        self.__stop_words = stop_words

    # Tries recursively to obtain a valid input file from the user
```

```
    def __get_input_file(self):
        try:
            input_file = file_input('Enter input file:  ')
            assert exists(input_file), 'Invalid input file'
            assert isfile(input_file), 'Not a file'
            return input_file
        except AssertionError as err:
            print(err)
            return self.__get_input_file()

    # Tries recursively to obtain a valid output file from the user
    # Empty input will cause report to not be saved
    def __get_output_file(self):
        try:
            output_file = file_input('Enter output file: ')
            if output_file == '':
                return ''
            with open(file=output_file, mode='w') as f:
                f.write('')
            return output_file
        except FileNotFoundError:
            print('Invalid output file name')
            return self.__get_output_file()

    # Decodes morse code message from input file
    def __get_decoded_message(self, input_file: str):
        decoded_message = Morse_Utils.decode_morse(input_file)
        return decoded_message

    # Returns a list of words to be used by __get_message_breakdown and
__get_essential_message methods
    def __get_frequencies(self, text: str):
        word_dict: Dict[str, Tuple[Message_Breakdown_Word,
                                   Essential_Message_Word]] = {}
        for line_index, line in enumerate(text.splitlines()):
            for word_index, word in enumerate(line.split(sep=' ')):
                if word in word_dict:
                    word_dict[word][0].addInstance((line_index, word_index))
                    word_dict[word][1].addInstance()
                else:
                    word_dict[word] = (
                        Message_Breakdown_Word(
                            word=word, first_pos=(line_index, word_index)),
                        Essential_Message_Word(
                            word=word, first_pos=(line_index, word_index))
                    )
        message_breakdown_word_list = []
        essential_message_word_list = []
```

```
        for mbw, emw in word_dict.values():
            message_breakdown_word_list.append(mbw)
            essential_message_word_list.append(emw)
        return message_breakdown_word_list, essential_message_word_list

    # Analyses frequencies and positions of the words in the morse code
message
    def __get_message_breakdown(self, word_ls:
List[Message_Breakdown_Word]):
        sorted_words: List[Message_Breakdown_Word] = quicksort(
            word_ls)
        try:
            previous_frequency = sorted_words[0].getFrequency()
        except IndexError:
            previous_frequency = self.__min_significant_frequency - 1
        message_breakdown = []
        if previous_frequency >= self.__min_significant_frequency:
            message_breakdown.append(
                f'*** Morse words with frequency = {previous_frequency}\n')
        for word in sorted_words:
            current_frequency = word.getFrequency()
            if current_frequency < self.__min_significant_frequency:
                break
            if current_frequency < previous_frequency:
                message_breakdown.append(
                    f'\n*** Morse words with frequency =
{current_frequency}\n')
                previous_frequency = current_frequency
            message_breakdown.append(word.getDetails())
        return ''.join(message_breakdown)

    # Analyses essential message through sorting by frequency and first
position of the words
    def __get_essential_message(self, stop_words: Set[str], word_ls:
List[Essential_Message_Word]):
        sorted_words: List[Essential_Message_Word] = quicksort(
            word_ls)  # type: ignore
        essential_message = []
        for word in sorted_words:
            w = word.getWord()
            if w not in stop_words and w.isalpha():
                essential_message.append(w)
        return ' '.join(essential_message)

    # Builds report from the three components
    def __build_report(self, decoded_message: str, message_breakdown: str,
essential_message: str):
        report = '*** Decoded morse text\n' + decoded_message + '\n' \
```

```
            if decoded_message != '' \
            else '*** No decoded morse text\n'
        report += message_breakdown + '\n' \
            if message_breakdown != '' \
            else f'*** No morse words with frequency >=
{self.__min_significant_frequency}\n'
        report += '*** Essential Message\n' + essential_message \
            if essential_message != '' \
            else '*** No essential message'
        return report

    # Saves report to the output file, if specified
    def __save_report(self, message: str, file: str):
        if file != '':
            with open(file=file, mode='w') as f:
                f.write(message)

    # Generic Utility Methods

    # Retrieves valid option from user, recursively
    def __get_choice(self):
        print(
            'Please select your choice (1, 2, 3 or 4):\n \
            \t1.  Change printing mode\n \
            \t2.  Convert plain text to morse code\n \
            \t3.  Analyse morse code message\n \
            \t4.  Exit'.expandtabs(tabsize=4)
        )
        try:
            choice = int(input('Enter your choice: '))
            assert 1 <= choice <= 4
        except Exception:
            print('Invalid input')
            return self.__get_choice()
        return choice

    # Prints the author's details
    def __print_info(self):
        print('*' * 57)
        print(
            f'''*\t{self.__author['module']}: Morse Code Message
Analyser\t*''')
        print('*' + '-' * 55 + '*')
        print('*\t\t\t\t\t\t\t*')
        print(f'''*\t- Done By: {self.__author['name']}
({self.__author['admin']})\t\t\t*''')
        print(f'''*\t- Class: {self.__author['class']}\t\t\t\t*''')
        print('*\t\t\t\t\t\t\t*')
```

```
        print('*' * 57)
```

2. data_structures/abstract_stack.py

```
# Name:     Ethan Tan
# Admin:    P2012085
# Class:    DAAA/2B/03

# Import Dependencies
from abc import ABC, abstractmethod


# Template for Stack class
class Abstract_Stack(ABC):
    @abstractmethod
    def push():
        pass

    @abstractmethod
    def peek():
        pass

    @abstractmethod
    def pop():
        pass

    @abstractmethod
    def empty():
        pass

    @abstractmethod
    def size():
        pass
```

3. data_structures/essential_message_word.py

```
# Name:     Ethan Tan
# Admin:    P2012085
# Class:    DAAA/2B/03

# Import Dependencies
from typing import Tuple
from .word import Word


# Model for a word in the essential message
class Essential_Message_Word(Word):
    def __init__(self, word: str, first_pos: Tuple[int, int]):
```

```
        super().__init__(word=word)
        self.__first_position: Tuple[int, int] = first_pos

    def addInstance(self) -> None:
        self._frequency += 1

    def getFirstPos(self) -> Tuple[int, int]:
        return self.__first_position

    def __lt__(self, otherWord) -> bool:
        return (-self.getFrequency(), self.getFirstPos()) < (-
otherWord.getFrequency(), otherWord.getFirstPos())
```

4. data_structures/message_breakdown_word.py

```
# Name:     Ethan Tan
# Admin:    P2012085
# Class:    DAAA/2B/03

# Import Dependencies
from typing import List, Tuple
from .word import Word
from ..utils.morse_utils import Morse_Utils


# Model for a word in the message breakdown
class Message_Breakdown_Word(Word):
    def __init__(self, word: str, first_pos: Tuple[int, int]):
        super().__init__(word=word)
        self.__all_positions: List[Tuple[int, int]] = [first_pos]

    def addInstance(self, instance_position: Tuple[int, int]) -> None:
        self.__all_positions.append(instance_position)
        self._frequency += 1

    def getDetails(self) -> str:
        details = ''
        details += Morse_Utils.encode_morse(self._word)
        details += f'[{self._word}] ({self._frequency})
{self.__all_positions}\n'
        return details

    def __lt__(self, otherWord) -> bool:
        return (-self.getFrequency(), self.size(), self.getWord()) < (-
otherWord.getFrequency(), otherWord.size(), otherWord.getWord())
```

5. data_structures/morse_character.py

```
# Name:     Ethan Tan
# Admin:    P2012085
# Class:    DAAA/2B/03

# Import Dependencies
from .stack import Stack


# Models a morse character for vertical printing
class Morse_Character(Stack):
    def __init__(self, morse_char: str, pad_char: str = ' ', padding: int =
5):
        super().__init__()

        # Pads all characters to length 5, as longest morse character is 5,
        #   thus enforcing O(1) time complexity
        for symbol in morse_char[::-1]:
            self += symbol

        for _ in range(padding - self.size()):
            self.push(pad_char)
```

6. data_structures/node.py

```
# Name:     Ethan Tan
# Admin:    P2012085
# Class:    DAAA/2B/03

# Class for an individual node in the Stack class
class Node:
    def __init__(self, val, next_node=None):
        # Value property is immutable after initialisation
        self.__value = val

        # Next node is not encapsulated
        self.next = next_node

    # Getter for value
    def get_value(self):
        return self.__value
```

7. data_structures/stack.py

```
# Name:     Ethan Tan
```

```python
# Admin:    P2012085
# Class:    DAAA/2B/03

# Import Dependencies
from .abstract_stack import Abstract_Stack
from .node import Node


# Generic linear data structure implementing LIFO principle
class Stack(Abstract_Stack):
    def __init__(self):
        self.__head = None
        self.__size = 0

    def push(self, val):
        new_node = Node(val=val)
        if self.__head is None:
            self.__head = new_node
        else:
            new_node.next = self.__head
            self.__head = new_node
        self.__size += 1

    def peek(self):
        if self.__head is not None:
            return self.__head.get_value()
        else:
            return None

    def pop(self):
        tmp_val = None
        if self.__head is not None:
            tmp_val = self.__head.get_value()
            self.__head = self.__head.next
            self.__size -= 1
        return tmp_val

    def __len__(self) -> int:
        return self.__size

    def __iadd__(self, val):
        self.push(val=val)
        return self

    def empty(self):
        self.__size = 0
        self.__head = None
        return self
```

```
    def size(self) -> int:
        return self.__size
```

8. data_structures/word.py

```
# Name:     Ethan Tan
# Admin:    P2012085
# Class:    DAAA/2B/03

# Import Dependencies
from abc import ABC, abstractmethod


# Models a word in the morse code message
class Word(ABC):
    def __init__(self, word: str):
        self._word: str = word.upper()
        self._frequency: int = 1

    def size(self) -> int:
        return len(self._word)

    def getWord(self) -> str:
        return self._word

    def getFrequency(self) -> int:
        return self._frequency

    @abstractmethod
    def addInstance(self):
        pass

    @abstractmethod
    def __lt__(self, otherWord):
        pass
```

9. sorting/quicksort.py

```
# Name:     Ethan Tan
# Admin:    P2012085
# Class:    DAAA/2B/03

# type: ignore
# Import Dependencies
from typing import List
```

```
# Swap 2 values in the list by specified indices
def swap(ls, i, j):
    ls[j], ls[i] = ls[i], ls[j]


# Recursive body of the algorithm
def partition(ls, l, h):
    if l < h:
        partition_index = pointer = l - 1
        partition_num = ls[h]
        for el in ls[l:h]:
            pointer += 1
            if el < partition_num:
                partition_index += 1
                swap(ls, pointer, partition_index)
        partition_index += 1
        swap(ls, h, partition_index)
        partition(ls, l, partition_index - 1)
        partition(ls, partition_index + 1, h)


# Main function
# Sorts an arbitrary list using the quicksort algorithm
# Returns a new list, sorted
def quicksort(ls: List):
    ls_copy = ls.copy()
    partition(ls_copy, 0, len(ls) - 1)
    return ls_copy
```

10. utils/io_utils.py

```
# Name:     Ethan Tan
# Admin:    P2012085
# Class:    DAAA/2B/03

# Import Dependencies
import os
import re


# Clears console, imitates reloading the screen
def clear_console():
    os.system('cls' if os.name == 'nt' else 'clear')


# Removes all internal whitespace
```

```
def whitespace_reducer(text: str):
    return re.sub(pattern='[\\s]+', repl=' ', string=text)


# Converts non alphanumeric characters to whitespace
def strip_special_characters(text: str):
    return re.sub(pattern='[^A-Za-z0-9\\s]', repl=' ', string=text)

# Retrieves formatted multi-line input
def multi_line_input():
    lines = []
    while True:
        line = input()
        if line:
            lines.append(whitespace_reducer(line))
        else:
            break
    return "\n".join(lines)


# Retrieves formatted input
def simple_input(prompt: str):
    return whitespace_reducer(input(prompt)).strip().lower()


# Retrieves formatted file path input
def file_input(message: str):
    return whitespace_reducer(input(message)).strip()
```

11. utils/morse_utils.py

```
# Name:     Ethan Tan
# Admin:    P2012085
# Class:    DAAA/2B/03

# Import Dependencies
from abc import ABC, abstractmethod


# Merely a namespace for morse-to-text translation functions
class Morse_Utils(ABC):
    @abstractmethod
    def __init__(self):
        pass

    # Encodes plain text to morse code
    @staticmethod
    def encode_morse(plain_text: str):
```

```
        TEXT_TO_MORSE = {
            "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": "--..",
            "1": ".----",
            "2": "..---",
            "3": "...--",
            "4": "....-",
            "5": ".....",
            "6": "-....",
            "7": "--...",
            "8": "---..",
            "9": "----.",
            "0": "-----",
        }
        morse_ls = []
        contents = plain_text.upper().splitlines()
        for line in contents:
            for char in line:
                morse_ls.append(
                    TEXT_TO_MORSE[char] if char in TEXT_TO_MORSE else char)
                morse_ls.append(",")
            morse_ls.pop()
            morse_ls.append('\n')
        return ''.join(morse_ls)
```

```python
    # Decodes morse code to plain text
    @staticmethod
    def decode_morse(file: str):
        MORSE_TO_TEXT = {
            ".-": "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",
            ".----": "1",
            "..---": "2",
            "...--": "3",
            "....-": "4",
            ".....": "5",
            "-....": "6",
            "--...": "7",
            "---..": "8",
            "----.": "9",
            "-----": "0",
        }
        try:
            with open(file=file) as f:
                contents = f.read().splitlines()
                text_ls = []
                for line in contents:
                    for char in line.split(","):
```

```
                        text_ls += MORSE_TO_TEXT[char] if "." in char or "-"
in char else char
                    text_ls.append("\n")
            return "".join(text_ls)
        except KeyError:
            raise AssertionError(
                f'Morse code in file {file} is in invalid format.')
```

~ END ~