Group Project Report: Evaluating and Sorting Assignment

Group 10: Goh Rui Zhuo, Jonathan Leo, Bharathan Bhaskaran
IDs: 2222329, 2222655 ,2201395
Class: DAAA2B05

Abstract

This report contains the idea of developing the evaluation and sorting assignment, with the use of Object Oriented Programming, with key ideas of encapsulation, function/ operator overloading polymorphism, and inheritance and the use of custom and pre-built data

1 Introduction (Application Description and User Guidelines)

1.1 Application Overview

The purpose of the application, the idea is to learn about the different data structures, object oriented programming, as to develop an application that can be solve assignment statements with mathematical expressions by using parse tree.

1.2 User Guidelines

This is a guide to teach how the user is able to run the application smoothly and without any errors.

1. Open your command line interface) such as Terminal on MacOS/Linux or Command Prompt/PowerShell on Windows. `python main.py` and press Enter after navigating to diectory.

Upon starting the application, users will be presented with a menu of six different options,

1. Add/Modify assignment statement
2. Display current assignment statements
3. Evaluate a single variable
4. Read assignment statements from file
5. Sort assignment statements
6. Exit

2 Approaches to Each Option

2.1 Option 1

In Option 1, the goal is to efficiently manage the storage and modification of assignment statements. When a variable already exists within the system, it is necessary to update the corresponding assignment statements accurately.

To achieve this, we first get the input from the users, which is then proceeded to be error handled. Once it passes the error handling, we store it into our data structures used for storage. We utilise a Hash Table for storing the assignments. The choice of a Hash Table is driven by its inherent characteristics such as efficient storage, fast access, and the ability for quick modification.

By leveraging the strengths of Hash Tables, Option 1 promises an efficient and reliable approach to managing assignment statements within the data storage system.

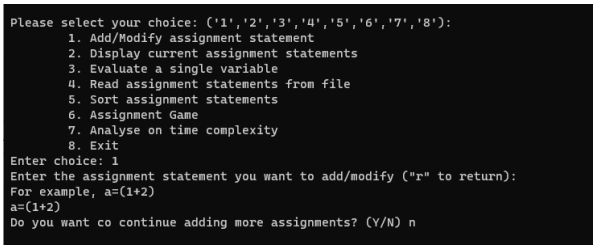

Figure 1: Usage of Option 1

2.1.1 Some Error Handling Mechanism

The input from the users was being check and error handled with the following ideas

- **Input Format:**
- **Variables Name:**
- **Circular Dependency**

2.2 Option 2

In Option 2, the goal is to be able to evalaute the assignment given the equation with the use of parse tree.

To achieve this, we retrieve the full data from our hash table data storage. Then, we proceed and use a dependency graph to get the dependency of each variables first so that we are able to evaluate variables in a concrete direction here

Then, we utilise topological sort to sort the variables in a given order to prevent circular dependency

Then, we proceeded to do equation clean up here such that if let's say the equation does not have the correct inner parenthesis, the equation cleanup class will help to add.

Next, we proceeded to build the parse tree to get the expression tree suitable for evaluation afterwards. The tree is then pass on to the evaluator and ther results is gotten through get results here.

This is then stored into a dictionary of values to provide efficient printing when required here.

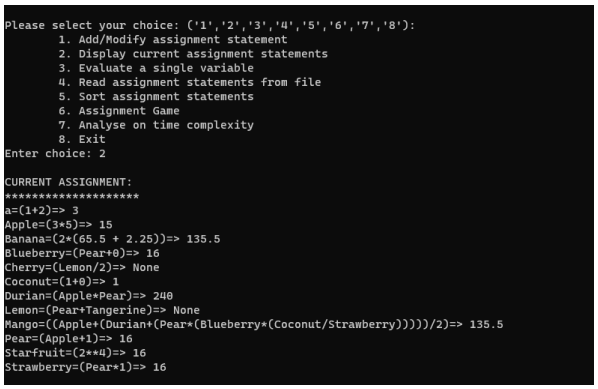

Figure 2: Usage of Option 2

2.3 Option 3

In Option 2, the goal is to be able to evalaute the assignment given the variable through the user input.

To achieve this, we first get the input from the users, which is then proceeded to be error handled. Once it passes the error handling, we will proceed to and us similar approach to option 2 by using a dependency graph to get the dependency of each variables first.

Then, also we utilise topological sort to sort the variables in a given order to prevent circular dependency

Next, we proceeded to build the parse tree to get the expression tree for that variable itself.

.

2.3.1 Some Error Handling Mechanism

The input from the users was being check and error handled with the following ideas

- **Variables Name:**

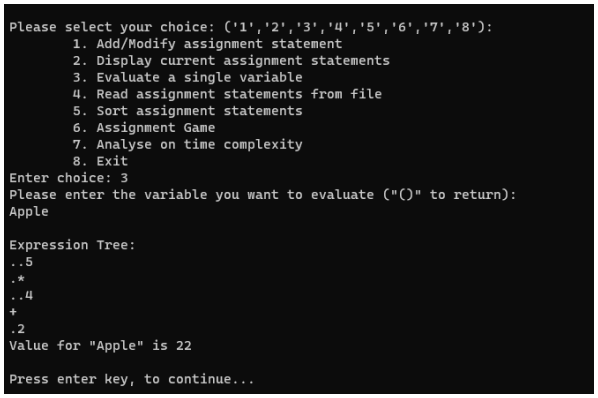

Figure 3: usage of Option 3

2.4 Option 4

In Option 4, the goal is to be able to upload a file and provide values and results of that file here.

To achieve this, we first get the input from the users on the file they want to upload here. This is then error handled along the way to provide correct validation of file here.

The file is split by newline so that each line can be evaluated accurately. Each part is then split into the variable and equation to store.

This is then stored into the overall data storage and is then put into run by utilising option 2 run here.

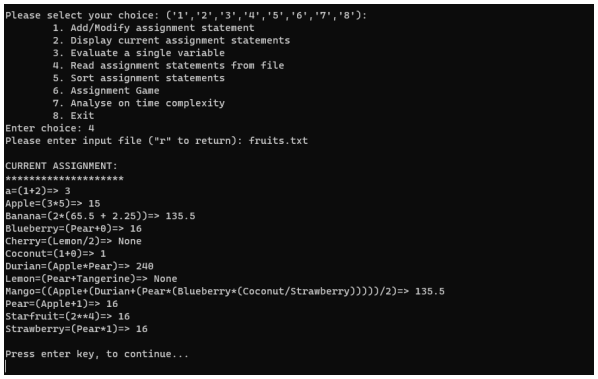

Figure 4: Usage of Option 4

2.4.1 Some Error Handling Mechanism

The input from the users was being check and error handled with the following ideas

- **Input Format:** Checking if the input file type is accurate and meets the expected format and type.
- **Circular Dependency**

2.5 Option 5

In Option 5, the is to store the current assignments into a file such that it make sure that the order are ordered by its values and if equal it is sort by the order of letters.

To achieve this, we first get the input from the users, which is then proceeded to be error handled. Once it passes the error handling, we will then sort the statements by the values first and provide

accurate inputs. Then if equal values occur, we will sort by the variable name in order of alphabets here.

This is then stored into the file required by the user.

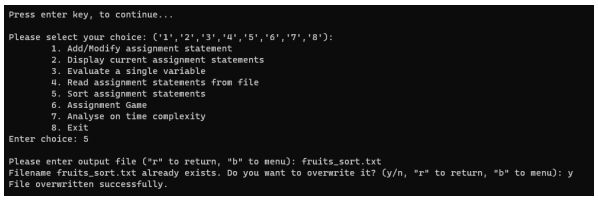

Figure 5: Usage of Option 5

2.5.1 Some Error Handling Mechanism

The input from the users was being check and error handled with the following ideas

- **Input Format:**
- **File Exist :**

3 Object-Oriented Programming Approach

3.1 binaryTree.py

The `BinaryTree` class is designed for creating and managing a binary tree structure. It encapsulates the complexity of tree operations, providing a user-friendly interface for interacting with binary trees.

Class Diagram

Class BinaryTree
Attr: key, left, right
Ops: init(), set_key(), get_key(), get_left(), get_right(), insertLeft(), insertRight(), printPreorder(), printInorder(), printPostorder()

Encapsulation: The class encapsulates the tree's nodes' data (key, left, right) and contains public methods (`set_key`, `get_key`, `get_left`, `get_right`, `insertLeft`, and `insertRight`) to access and modify them.

Abstraction: `BinaryTree` abstracts the complexity of tree traversals, with user-friendly methods (`printPreorder`, `printInorder`, and `printPostorder`)

3.2 dependency.py

The `DependencyGraph` class is designed to manage and represent dependencies among expressions.

Class Diagram

Class DependencyGraph
Attr: datas, graph
Ops: init(datas), add(var, exp), get(var), build(), __findDependency(exp), __str__()

Encapsulation

The class encapsulates the data members (datas, graph) and provides public methods to access and modify them, such as `add`, `get`, and `build`.

Data Hiding

The `__findDependency` method is a private method which is meant to access privately.

Operator Overloading

The class overloads the `__str__` method to provide a custom string representation of the dependency graph.

3.3 EquationCleanup.py

The `EquationCleanup` class is designed to clean up and evaluate mathematical expressions, effectively handling the input.

Class Diagram

Class EquationCleanup
Attr: exp
Ops: init(exp), __applyOperator(ops, values),
__pre(op), evaluate(), get_cleanup()

Encapsulation

The class encapsulates the cleanup process by keeping the expression manipulation methods private. The public method `get_cleanup` provides the interface through which users can obtain without understanding.

3.4 ExpressionEvaluator.py

The `ExpressionEvaluator` class is designed to evaluate mathematical expressions represented as binary trees.

Encapsulation

The class encapsulates the operations for evaluation in a private dictionary and private method where users can obtain the result of an expression by calling the public method `get_results`, which internally calls the `__evaluate` method.

Abstraction

The `ExpressionEvaluator` abstracts the complexity of expression evaluation. Users do not need to understand the recursive traversal and evaluation of the binary tree as they can just get it within the getters.

Class Diagram

Class ExpressionEvaluator
Attr: operations
Ops: init(), __evaluate(node, variable_values),
get_results(node, variable_values)

3.5 ExpressionManager.py

The `ExpressionManager` class, inheriting from `DependencyGraph`, is designed to manage expressions by handling dependencies and replacing variables with their respective values.

Class Diagram

Class ExpressionManager
Inherits: DependencyGraph
Ops: init(datas), topologicalSort(), replaceVar(exp, var_dict)

Encapsulation

The class encapsulates methods for managing expressions, such as `topologicalSort` and `replaceVar`, as well as the data related to expression dependencies

Inheritance

`ExpressionManager` inherits from `DependencyGraph`, hence reuse its functionality for building and managing a dependency graph

3.6 hashTable.py

The `hashTable` class is designed to implement a hash table structure, providing efficient storage and retrieval of key-value pairs.

Operator Overloading

The class overloads operators like `__setitem__`, `__getitem__`, and `__delitem__` to allow users to use the the hash table using familiar syntax

Class Diagram

The class diagram below represents the `hashTable` class:

Class hashTable
Attr: size, keys, buckets, count
Ops: init(size), hashFunc(key), rehashFunc(old_hash), resize(), items(), __insert(key, value), __setitem__(key, value), __getitem__(key), __delitem__(key), get(key, default), printHash()

3.7 User.py

The `User` class is designed to manage user-related information in the system.

Class Diagram

Class User
Attr: __name, __admin_no, __module_code, __class, __info
Ops: init(), set_name(name), set_adminNo(admin_no), set_moduleCode(module_code), set_class(className), set_info(info), get_names(), get_adminNos(), get_moduleCode(), get_class(), get_info(), __str__()

Encapsulation

The class encapsulates user details and provides getter and setter methods for each piece of information. It maintains private attributes for the name, admin number, module code, class, and other information, and exposes methods like `set_name`, `get_names`, etc., to interact with these attributes.

Operator Overloading

The class overloads the `__str__` method to provide a custom string representation of the the user information

3.8 Stack.py

The `Stack` class is designed to implement a stack data structure, providing operations for standard stack functionalities such as pushing, popping, and checking if the stack is empty.

Class Diagram

Class Stack
Attr: __list
Ops: init(), isEmpty(), size(), clear(), push(item), pop(), get(), __str__()

Encapsulation

The class encapsulates the stack operations (`__list`) and exposes methods like `push`, `pop`, `isEmpty`, `size`, and `clear` to interact with the stack.

Operator Overloading

The class overloads the `__str__` method to provide a custom string representation of the stack.

3.9 Options files

The `Options` class, contains didferent type of oop approaches

Class Diagram

Class Options
Inherits: OtherOptions
Attr: datas, errorVal
Ops: init(datas), run()

.

Inheritance

`Options` inherits from `other options`, utilizing and extending its functionality and leverages on the run method in option2

Polymorphism

`Options` overwrites the run function of other options

4 Discussion on Data Structures

The application leverages a combination of custom-developed and built-in data structures, each chosen for its specific advantages in terms of performance, complexity, and task suitability. Below, we discuss each data structure in detail, along with its performance and justification for its use.

4.1 Hash Table

Purpose/Usage: Efficient data retrieval and storage

- **Performance:** Hash Tables provides a constant time of O(1) lookup on the average where the number of items does not matter
- **Justification:** Hash tables provide constant-time complexity for key-based data retrieval, and therefore is crucial in this scenario when getting data from the storage. Comparing it with python dictionary, this is more memory efficient for the application due to the dictionary include overhead support to support advanced features.
- **Suitability:** This is used for where we require quick access to data is needed and also less memory used

4.2 Dependency Graph

Purpose/Usage: Managing and getting dependencies on assignment

- **Performance:** The complexity generally depends on the implementation, where the typical scenario will be $O(V + E)$ for traversal operations, where V is vertices and E is edges.
- **Justification:** Custom-built to handle complex dependencies in this scenario to handle the dependency on the assignment equations
- **Suitability:** This is used in parts where involve resolving dependencies or executing tasks in a specific order.

4.3 Stack

Purpose/Usage: Parsing expressions and evaluating expressions

- **Performance:** The complexity is general O(1) for both the push and pop operations.
- **Justification:** The stacks are suitable that require last-in-first-out (LIFO) access, such as function call management and expression parsing where it is utilise for the evaluation of assignments using parse tree.
- **Suitability:** This is used in parts of the application in the evaluation of assignments using parse tree

4.4 Binary Tree

Purpose/Usage: Storing and organising data for expressions

- **Performance:** The complexity for search, insert, and delete operations is generally O(log n) in a balanced binary tree.
- **Justification:** Binary trees are suitable for operations that require ordered access and hierarchical organization of data. They are fundamental in constructing binary search trees, and is use as parse tree in this scenario.
- **Suitability:** The binary tree is used in the application to manage sorted data and enable quick search operations. This is useful for developing parse tree and generally useful for this assignment.

4.5 List

Purpose/Usage: Maintaining ordered collections of items such as index

- **Performance:** The performance is $O(1)$ for accessing elements by index, $O(n)$ for operations when searching for an element and require huge storage of memory

- **Justification:** Lists provide flexible structures for maintaining an ordered collection of items, and the ability to delete, access push the data to the list.
- **Suitability:** This is used for maintaining sequences of data where the order is important and the data may need to be iterated over or accessed randomly such as sorting the index or storing the sentences.

4.6 Dictionary

Purpose/Usage: Short term storage of items

- **Performance:** The average case of $O(1)$ for search, insert, and delete operations but are not very memory efficient
- **Justification:** Its offer a efficient way to store and retrieve data using key-value pairs and ability to del from the dictionary
- **Suitability:** This is used for storing data that such as the values that was evaluated and the need to store data.

The idea behind the development or selection of these data structures really wants to get to the best optimised overall performance

Data Structure	Type (Custom/Built-in)	Purpose/Usage
Hash Table	Custom	Efficient data retrieval and storage
Dependency Graph	Custom	Managing and getting dependencies on assignment
Stack	Custom	Parsing expressions and evaluating expressions
Binary Tree	Custom	Help in the creation of parse tree
List	Built-in	Maintaining ordered collections of items such as index
Dictionary	Built-in	Short term storage of items

Table 1: Summary of Data Structures Used in the Application

5 Discussion on Algorithms

The application not only leverages efficient data structures but also integrates efficient algorithms to ensure optimal performance in data processing and task execution.Here, we are going to talk about tithe different algorithms used.

5.1 Topological Sort

Purpose/Usage: Sorting dependencies in the Dependency Graph

- **Performance:** The time complexity is $O(V + E)$, where V is the number of vertices and E is the number of edges in the graph.
- **Justification:** Topological sorting is needed for ordering tasks in a sequence such that each task comes before all tasks that depend on it. This is very important in the Dependency Graph to resolve dependencies efficiently.
- **Suitability:** Utilised in scenarios involving dependency resolution which is the solving from dependency graph.

5.2 Merge Sort

Purpose/Usage: Efficiently sorting data

- **Performance:** The time complexity $O(n \log n)$ time complexity, which is optimal for sorting.

- **Justification:** Merge Sort is chosen for its efficiency and the stable in sorting operations, ensuring that data is sorted in a predictable and efficient manner. Merge sort is even faster compared to some other sorting algorithm like bubble sort for example.
- **Suitability:** Utilised in the sorting of statements values and keys here.

5.3 Equation Cleanup Algorithms

Purpose/Usage: Optimising and simplifying mathematical expressions

- **Performance:** The time complexity can vary, but often involves traversing the expression tree, $O(n)$.
- **Justification:** Cleaning up equations is needed for providing full parenthesis to the equation
- **Suitability:** Utilised in the options where user input or have files that are not fully parenthesis and prompt them to get whether they want to continue here.

Each algorithm is chosen to address to special needs of each goals here.

6 Challenges and Learning Outcomes

6.1 Technical Challenges

The project posed technical challenges, primarily in handling circular dependencies, implementing new data structures like hashtables, and making error validation, each requiring in-depth analysis, research, and a methodical approach to resolve.

6.2 Group-work Challenges

Collaborative challenges included acclimating to new team dynamics and managing different schedules. We overcame these by putting time to understand each member's strengths.

6.3 Key Takeaways and Learning Achievements

The project offered numerous learning outcomes, notably in efficiently utilising hashtables for data management, understanding the requirement of data dependencies, and significantly enhancing our teamwork and problem-solving skills through effective communication and collaboration.

7 Roles and Contributions

Detail the roles and contributions of each team member, specifying who was responsible for each part of the project.

Table 2: Team Member Contributions

Names	**Roles**	**Contribution**
Goh Rui Zhuo	GUI creation, option 1, 2, 4, adding of error validation, improvement all	45%
Bharathan Bhaskaran	add and improve error validation Testing the code Doing Option 1 ,5	27.5%
Jonathan Leo	add and improve error validation Testing the code Doing Option 3,4	27.5%

8 Conclusion

Overall, this assignmnet was enriching and useful in guiding us

9 Appendices

9.1 Appendix A: Source Code

Depdendency.py

```
class DependencyGraph:
    '''
    Creating the initializer here of data and the graph
    '''
    def __init__(self, datas):
        self.datas = datas
        self.graph = {var: [] for var, exp in self.datas.items() if var is not
            ↪  None}

    '''
    Creating the adding of var and expression here
    '''
    def add(self, var, exp):
        self.graph[var] = self.__findDependency(exp)

    '''
    Creating the find dependency here
    '''
    def __findDependency(self, exp):
        tokenizer = Tokenizer(exp)
        t = tokenizer.get_tokenize_dep()
        return t

    '''
    Creating the get here to get variable from the graph
    '''
    def get(self, var):
        return self.graph.get(var, None)

    '''
    Creating the building of dependency graph here
    '''
    def build(self):
        # For each variable and expression
        for var, exp in self.datas.items():
            # Find each dependency here depending on the expression
            tokens = self.__findDependency(exp)
            for token in tokens:
                # If the token is an alphabet and not the same as var append
                    ↪ it to the dependency array
                if token.isalpha() and token != var:
                    self.graph[var].append(token)
        return self.graph

    # Overwriting print function here
    def __str__(self):
        return str(self.graph)
```

EquationCleanup.py (Done by: Goh Rui Zhuo

```


# Creating the equation clean up class here with the main use of inputing
    ↪ required paramthesis hgere
```

```
class EquationCleanup:

    '''
    Replacing all the space with empty space
    '''
    def __init__(self, exp):
        self.exp = exp[1:-1].replace("␣", "")

    '''
    Applying the operator here to aplly the last two operator to the last two
        ↪ values from the values stack
    '''
    def __applyOperator(self, ops, values):
        if ops:
            op = ops.pop()
            right = values.pop()
            left = values.pop()
            values.append(f'({left}{op}{right})')

    '''
    Checking the operator and encoding it to the value with the order of
        ↪ operation
    '''
    def __pre(self, op):
        if op == '**':
            return 3
        if op in ['*', '/']:
            return 2
        if op in ['+', '-']:
            return 1
        return 0

    '''
     Cleanup the equation here
    '''
    def evaluate(self):
        values = []
        ops = []
        i = 0

        # Handle negative or positive values at the start
        if self.exp[0] == '-':
            values.append('0')

        if self.exp[0] == '+':
            i += 1

        # While loop if the length is less than the length of expression
        while i < len(self.exp):

            # If the detected character is a digit or a float here and push to
                ↪  the stack
            if self.exp[i].isdigit() or (i > 0 and self.exp[i] == '.' and self
                ↪ .exp[i-1].isdigit()):
                j = i
                while j < len(self.exp) and (self.exp[j].isdigit() or self.exp
                    ↪ [j] == '.'):
                    j += 1

                # This is to convert it to either a float or a integer
                    ↪ depending on the type of it
                num_str = self.exp[i:j]
```

```
            num_val = float(num_str) if '.' in num_str else int(num_str)

            # Appending to the stack here
            values.append(str(num_val))
            i = j

        # IF the detected character is a exponent, then it will proceed to
            ↪  apply operator on the last two values with the use of
            ↪ apply operator
        elif self.exp[i:i+2] == '**':
            while ops and self.__pre(ops[-1]) >= self.__pre(self.exp[i:i
                ↪ +2]):
                self.__applyOperator(ops, values)

            # Appending it to the operations stack
            ops.append(self.exp[i:i+2])

            # Increment of 2 because of the operator ** containing two
                ↪ different characters
            i += 2

        # If the detected chracter is a usual operation it is also applied
            ↪  with apply operator here
        elif self.exp[i] in ['+', '-', '*', '/']:
            while ops and self.__pre(ops[-1]) >= self.__pre(self.exp[i]):
                self.__applyOperator(ops, values)

            # Appending it to the operations stack
            ops.append(self.exp[i])

            # Increment of 1 because of the operator containing only one
                ↪ character
            i += 1

        # If the detected character is a open paramthesis, it append the
            ↪ paramthesis into the operation stack
        elif self.exp[i] == '(':
            ops.append(self.exp[i])

            # Increment of 1 because of the operator containing only one
                ↪ character
            i += 1

         # If the detected character is a close paramthesis, it detects
            ↪ and until another opening paramthesis is seen, before
            ↪ applying the operator here
        elif self.exp[i] == ')':
            while ops and ops[-1] != '(':
                self.__applyOperator(ops, values)

            ops.pop()

            # Increment of 1 because of the operator containing only one
                ↪ character
            i += 1

        # IF the detected values is the otjhers  such as it is a variable
            ↪ then it is appended into the values stack
        else:
            j = i
            while j < len(self.exp) and not self.exp[j] in ['+', '-', '*',
                ↪  '/', '(', ')', '.', '␣','**']:
```

```
                    j += 1
                word = self.exp[i:j]
                values.append(word)
                i = j

        # If operation is not empty, apply last apply operation function to it
        while ops:
            self.__applyOperator(ops, values)
        return values[-1]

    '''
     Get clean up function as a getter
    '''
    def get_cleanup(self):
        result = self.evaluate()
        return result
```

expressionEvaluator.py (Done by: Goh Rui Zhuo

```
class ExpressionEvaluator:
    '''
    creating the operations for evaluation
    '''
    def __init__(self):
        self.operations = {
            '-': lambda x, y: x - y,
            '*': lambda x, y: x * y,
            '**': lambda x, y: x ** y,
            '+': lambda x, y: x + y,
            '/': lambda x, y: None if y == 0 else x / y,
        }

    '''
   private function that evaluate and takes in node and variable values ehre
    '''
    def __evaluate(self,  node, variable_values):

        # If the not node.left and node.right
        if not node.left and not node.right:

            # If int key ==  key mean that it is a integer
            # Define it to be an integer or else it is a float
            try:
                if int(node.key) == node.key:
                    return int(node.key)
                return float(node.key)

            # Return none if none to be fond
            except ValueError:
                return variable_values.get(node.key, None)

        # Evaluate the left and right node here using recursion as long as
            ↪ they exist
        left_val = self.__evaluate(node.left, variable_values) if node.left
            ↪ else None
        right_val = self.__evaluate(node.right, variable_values) if node.right
            ↪  else None

        # If the left and right are not none
        if left_val is not None and right_val is not None:
```

```
            # apply the operation
            operatorFunc = self.operations.get(node.key)

            # Return the operation result here
            return operatorFunc(left_val, right_val) if operatorFunc else None
        else:

            # Return none if does not exist
            return None

    '''
    creation of getter to get the results here
    '''
    def get_results(self ,  node, variable_values):
        return self.__evaluate( node, variable_values)
```

expressionManager.py (Done by: Goh Rui Zhuo

```
from .dependency import DependencyGraph
# Creating the manager here basing on the dependency graph
class ExpressionManager(DependencyGraph):

    '''
    Create a inherittance with dependency graph here and getting the datas as
        ↪ the attribute
    '''
    def __init__(self, datas):
        super().__init__(datas)

    '''
    Utilising topological sort here because to prevent circular dependency as
        ↪ it is able to detect it
    '''
    def topologicalSort(self):

        # Creating a list to store the final sorted variale here
        sorted_variables = []

        # Building the dependency graph here
        dependency_graph = DependencyGraph(self.datas).build()

        # While the dependency graph here is not empty
        while dependency_graph:

            # Setting acyclic to false to detect any form of cyclic later on
            acyclic = False

            # Looping through the dependency graph
            for var in list(dependency_graph):

                # For the dependency in each variable if it is in it then
                    ↪ break
                for dep in dependency_graph[var]:
                    if dep in dependency_graph:
                        break

                # Acyclic is set to true and deletion of variable from the
                    ↪ graph here
                else:
```

```
                    acyclic = True
                    del dependency_graph[var]

                    # Sorted variables append the new varaiable to be sorted
                    sorted_variables.append(var)

                    # Looping throguh the values in dependency graph,
                    for deps in dependency_graph.values():

                        # If the variable in depededncy graph values, remove
                            ↪ it here before breakling
                        if var in deps:
                            deps.remove(var)
                    break

            # Here this will detect the cycic dependency
            if not acyclic:
                return ("A cyclic dependency occurred")

        # Return the sorted variables to be evaluated
        return sorted_variables

    '''
    Replace the variable for with values once it is defined
    '''
    def replaceVar(self, exp,var_dict):

        # Split using tokenizer here
        tokens = exp.split()

        # Storing the result in a list here
        result = []

        # Looping throguh the tokens here
        for token in tokens:

            # IF the token is seen in the variables storage, it is then
                ↪ appended into the result with the values of it
            if token in var_dict:
                result.append(str(var_dict[token]))

            # Else it will just append the original token here
            else:
                result.append(token)

        # Form the replaced satement throguh here
        new_exp = ' '.join(result)
        return new_exp
```

hashTable.py (Done by: Goh Rui Zhuo

```
class hashTable:

    '''
    Create a init with different attributes here
    '''
    def __init__(self, size=5):
        self.size = size
        self.keys = [None] * self.size
        self.buckets = [None] * self.size
```

```
        self.count = 0


    '''
    Create a hash function that see if it is a int or a str and if it's a
        ↪ string it covnerts using the order ascii charater to store
    '''
    def hashFunc(self, key):

        # If it is an integer just return the key % self.size here
        if isinstance(key, int):
            return key % self.size

        # IF it is a string, then get the ascii character then % self.size for
            ↪  efficient storage
        elif isinstance(key, str):
            hashVal = 0
            for char in key:
                hashVal = (hashVal * 31 + ord(char)) % self.size
            return hashVal

    '''
    The rehashfunction incase of collison here to move to the next index here
    '''
    def rehashFunc(self, old_hash):
        return (int(old_hash) + 1) % self.size

    '''
    The resize function to resize incase the current storage is not enough
        ↪ here
    '''
    def resize(self):

        # Getting the old size here
        oldSize = self.size

        # Setting the new size to be two times of it
        self.size *= 2

        # Getting the old keys and buckets
        oldKeys = self.keys
        oldBuckets = self.buckets

        # Setting the current new keys and buckets to the new size here
        self.keys = [None] * self.size
        self.buckets = [None] * self.size

        # Insert back the old values into both the bcukets and keys
        for i in range(oldSize):
            if oldKeys[i] is not None:
                self.__insert(oldKeys[i], oldBuckets[i])

    '''
    The items function to get the items from the hash table here
    '''
    def items(self):
        for i in range(self.size):
            if self.keys[i] is not None:
                yield (self.keys[i], self.buckets[i])

    '''
```

```
    The resize function to resize incase the current storage is not enough
        ↪ here
    '''
    def __insert(self, key, value):

        # Getting the inex first
        index = self.hashFunc(key)

        # Check if the keys is not none then insert here
        while self.keys[index] is not None:
            index = self.rehashFunc(index)

        # Setting the new key and value here
        self.keys[index] = key
        self.buckets[index] = value

    '''
    Set item overloading set item function here
    '''
    def __setitem__(self, key, value):

        # Check if the need to resize the hashtable
        if self.count / self.size >= 0.8:
            self.resize()

        # get the index using the key
        index = self.hashFunc(key)

        # setting the start index here
        startIdx = index

        # While loop here
        while True:

            # If the index at both the buckets and keys is none or that they
                ↪ keys is equal to the key
            if self.buckets[index] is None or self.keys[index] == key:

                # If the buckets is none
                if self.buckets[index] is None:

                    # Increment the count by 1 here
                    self.count += 1

                # Setting the keys and buckets here
                self.keys[index] = key
                self.buckets[index] = value

                # Break the loop once all done
                break

            # Rehash the function to get the new index
            else:
                index  = self.rehashFunc(index)

    '''
    Get item overloading set item function here
    '''
    def __getitem__(self, key):

        # Getting the index base on the key
        index = self.hashFunc(key)
```

```
        # Set the start index here
        startIdx = index

        # While loop here
        while True:

            # If the keys at that indexis equal to what we are tyring to find
                ↪ then return the value
            if self.keys[index] == key:
                return self.buckets[index]

            # Else return none here
            elif self.keys[index] is None:
                return None

            # If unable to find and keys not none then rehash here
            index = self.rehashFunc(index)

            # If the rehash equal to the start index , it means it return to
                ↪ the original postion and therefore return none here
            if index == startIdx:
                return None

    '''
    Contains item overloading set item function here
    '''
    def __contains__(self, key):
        # Getting the index base on the key
        index = self.hashFunc(key)

        # Set the start index here
        startIdx = index

        # While loop here
        while True:

            # If the keys at that indexis equal to what we are tyring to find
                ↪ then return the true
            if self.keys[index] == key:
                return True

             # Else return false here
            elif self.keys[index] is None:
                return False

            # If unable to find and keys not none then rehash here
            index = self.rehashFunc(index)

            # If the rehash equal to the start index , it means it return to
                ↪ the original postion and therefore return none here
            if index == startIdx:
                return None

    '''
    Delete item overloading set item function here
    '''
    def __delitem__(self, key):

        # if the key exist here
        if key in self:
```

```
            # Getting the index here and set the start index
            index = self.hashFunc(key)
            startIdx = index

            # While loop here
            while True:

                # If the gotten key is equal to the target, proceed to set to
                    ↪ none
                if self.keys[index] == key:
                    self.keys[index] = None
                    self.buckets[index] = None

                    # Reduce the count by 1 and return
                    self.count -= 1
                    return
                else:
                    # If unable to find and keys not none then rehash here
                    index = self.rehashFunc(index)

                    # If the rehash equal to the start index , it means it
                        ↪ return to the original postion and therefore return
                        ↪  none here
                    if index == startIdx:
                        return None

            # Return the key not found here
            return (f"Key: {key} not found")
        else:
            return (f"Key: {key} not found")

    def get(self, key, default=None):
        # Getting the index base on the key
        index = self.hashFunc(key)

        # Set the start index here
        startIdx = index

        # While loop here
        while True:

            # If the keys at that indexis equal to what we are tyring to find
                ↪ then return the value
            if self.keys[index] == key:
                return self.buckets[index]

            # Else return none here
            elif self.keys[index] is None:
                return default

            # If unable to find and keys not none then rehash here
            index = self.rehashFunc(index)

            # If the rehash equal to the start index , it means it return to
                ↪ the original postion and therefore return none here
            if index == startIdx:
                return default

    ''' A print hash function to see what is inside here'''
    def printHash(self):

        # Looping through the hash table and printing the key value pair
```

```
        for i in range(self.size):
            if self.keys[i] is not None:
                print(f"Key: {self.keys[i]}, Value: {self.buckets[i]}")
```

mergeSort.py (Done by: Goh Rui Zhuo

```
class MergeSort:
    def __init__(self):
        pass
    def sort(self,l):
        # Checking that the array is indeed not empty
        if len(l) > 1:

            # Splitting the array into two different halves
            mid = len(l) // 2
            left = l[:mid]
            right = l[mid:]

            # sort both halves
            self.sort(left)
            self.sort(right)

            # Setting the index first
            l_index, r_index, m_index = 0, 0, 0

            # Setting a temporary copy here
            mergeList = l

            # While the left index and right index less than it respective
                ↪ array length
            while l_index < len(left) and r_index < len(right):

                # String both to covnert them to letters
                # COmpare the words base on uppering their case
                # If left less than right then the merge will input the left
                    ↪ value and the left index increment by 1
                if type(left[l_index]) in (int,float):
                    if left[l_index] < right[r_index]:
                        mergeList[m_index] = (left[l_index])
                        l_index += 1
                    else:
                    # The right index will then increment by 1 and added to
                        ↪ the merge list
                        mergeList[m_index] = right[r_index]
                        r_index += 1

                elif type(left[l_index][0]) == str or type(right[r_index][0])
                    ↪ == str :
                    if str(left[l_index][0].upper()) < str(right[r_index][0].
                        ↪ upper()):
                        mergeList[m_index] = (left[l_index])
                        l_index += 1
                    else:
                    # The right index will then increment by 1 and added to
                        ↪ the merge list
                        mergeList[m_index] = right[r_index]
                        r_index += 1
```

```
                # Increment of merge list index
                m_index += 1

            # Handle of remaing values in the left list
            while l_index < len(left):
                # insert to the merge list base on the left index
                mergeList[m_index] = left[l_index]
                l_index += 1
                m_index += 1

            while r_index < len(right):
                # insert to the merge list base on the right index
                mergeList[m_index] = right[r_index]
                r_index += 1
                m_index += 1
        return l
```

node.py (Done by: Goh Rui Zhuo

```
class Node:

    ''' Setting the value left and right attribute'''
    def __init__(self, value):
        self.value = value
        self.left = None
        self.right = None

    ''' Check if its leaf'''
    def is_leaf(self):
        return self.left is None and self.right is None

    '''Overloading the print function'''
    def __str__(self):
        return self.value
```

parseTree.py (Done by: Goh Rui Zhuo

```
from .binaryTree import BinaryTree
from .stack import Stack
from .tokenizer import Tokenizer

# Creating the parse tree building here
class ParseTreeBuilder:
    '''Initialse the expression'''
    def __init__(self, exp):
        self.exp = exp

    '''Building the parse tree method'''
    def build(self):

        # Initialise the tokenizer class here
        tokenizer = Tokenizer(self.exp)

        # Get the tokens base on the get tokenize parse
        tokens = tokenizer.get_tokenize_parse()

        # If the tokens is a digit or character by itself
        if len(tokens) == 3 and tokens[0] == '(' and tokens[2] == ')':
```

```
        # Try except block to handle if it is a float or integer
        try:
            value = int(tokens[1]) if tokens[1].isdigit() else float(
                ↪ tokens[1])

        # Return character if is netiher
        except ValueError:
            value = tokens[1]

        # Return the binary tree with the value here
        return BinaryTree(value)

    # Define the stack here
    stack = Stack()

    # Define the binary tree here
    tree = BinaryTree('?')

    # Push the binary tree into the stack here
    stack.push(tree)

    # Set the current to the tree define above
    current = tree

    # Looping through the tokens
    for i in tokens:

        # IF the token is an opening paramthesis, insert left
        if i == '(':
            current.insertLeft('?')

            # Push the current to the stack again
            stack.push(current)

            # Get the left key here
            current = current.get_left()

        # If it is an operation
        elif i in ['+', '-', '*', '/', '**']:

            # IF the current key here is set as '?'
            if current.get_key() == '?':

                # Set the key here on the current
                current.set_key(i)

                # Proceed to insert the right side here
                current.insertRight('?')

                # Push the current into the stack here
                stack.push(current)

                # Get the right key here
                current = current.get_right()

            # Else if the current is empty
            else:

                # Set a temporary binary tree here
                temp = BinaryTree(i)

                # Insert left with the current
```

```
                temp.insertLeft(current)

                # Proceed to insert right
                temp.insertRight('?')

                # If the stack is still not empty
                if stack:

                    # Get the last index here
                    parent = stack.get()

                    # If the parent left key is the current one
                    if parent.get_left() is current:

                        # Set the left child here to temp
                        parent.left = temp

                    # Set the right child here to temp
                    else:
                        parent.right = temp
                stack.push(temp)
                current = temp.get_right()

        # if it is not an operator
        elif i not in ['+', '-', '*', '/', '**', ')']:

            # Try except block to make sure the value are return in a
                ↪ correct format
            try:

                # Integer
                value = int(i)

            # Set for float or return the character
            except ValueError:
                try:
                    value = float(i)
                except ValueError:
                    value = i

            # Set the value here to current
            current.set_key(value)

            # If the stack is still not empty set the current to thelast
                ↪ value of stack
            if stack:
                current = stack.pop()

        # If it is the closing paramthesis here
        elif i == ')':

            # If the stack is still not emtpy pop the last element
            if stack:
                current = stack.pop()


        # FInal check of invalid expression here
        else:
            return ("Invalid␣token␣in␣expression")

    # Return the tree here
    return tree
```

stack.py (Done by: Goh Rui Zhuo

```
class Stack:
    def __init__(self):
        self.__list = []

    '''Stack is empty'''
    def isEmpty(self):
        return self.__list == []

    '''Size of stack'''
    def size(self):
        return len(self.__list)

    '''Removing all the elements'''
    def clear(self):
        self.__list.clear()

    '''Pushing the elements'''
    def push(self, item):
        self.__list.append(item)


    ''' Poping the elements'''
    def pop(self):
        if self.isEmpty():
            return None
        else:
            return self.__list.pop()
    '''Get the last element'''
    def get(self):
        if self.isEmpty():
            return None
        else:
            return self.__list[-1]
    '''Overaloding of print function'''
    def __str__(self):
        output = '<'
        for i in range(len(self.__list)):
            item = self.__list[i]
            if i < len(self.__list) - 1:
                output += f'{str(item)}, '
            else:
                output += f'{str(item)}'
        output += '>'
        return output
```

tokenzier.py (Done by: Goh Rui Zhuo

```
class Tokenizer:

    '''Initialse the index to track the current position in th eexprerssion
       ↪ and the expression'''
    def __init__(self, exp):
        self.exp = exp
        self.index = 0
```

```
    '''
    Tokenizer 1 based on the alphabet characters
    '''
    def __tokenize1(self):

        # List to store the tokens
        tokens = []

        # String to get the current token
        current_token = ''

        # For loop to check for the accurate tokens
        for char in self.exp:

            #  if it is an alphabet characters add alphabetic characters
            if char.isalpha():
                current_token += char
            else:

                # if current token is still not empty adding the completed
                    ↪ token to the list of tokens
                if current_token:
                    tokens.append(current_token)

                    # Reset current token
                    current_token = ''

         # If current token still exist add the last token
        if current_token:
            tokens.append(current_token)

        # Return the list of tokens here
        return tokens

    '''
    Tokenizer 1 based on alphanumeric characters and underscores, and special
        ↪ characters here
    '''
    def __tokenize2(self):

        # Set the tokens list, current tokens string and the set of special
            ↪ chracters
        tokens = []
        current_token = ''
        special_characters = {'+', '-', '*', '/', '(', ')', '␣', '**'}

        # For loop to loop throguh the expression
        for char in self.exp:

            # If it is an alphanumeric characters or underscores, it appends
                ↪ to the current tokens string
            if char.isalnum() or char == '_':
                current_token += char

            # Else
            else:
                # if current token is still not empty adding the completed
                    ↪ token to the list of tokens
                if current_token:
                    tokens.append(current_token)

                    # Reset current token
```

```
                current_token = ''

            # If character in special characters, just append the
                ↪ character
            if char in special_characters:
                tokens.append(char)

    # If current token still exist add the last token
    if current_token:
        tokens.append(current_token)

     # Return the list of tokens
    return tokens

'''
Tokenizer 1 getter to get base on dependency for depedency graph here
'''
def get_tokenize_dep(self):
    return self.__tokenize1()

'''
Tokenizer 1 getter to get base on replace variable here
'''
def get_tokenize_rv(self):
    return self.__tokenize2()

'''
Tokenizer 1 getter to get base on parse tree here
'''
def get_tokenize_parse(self):
    # Public method to access __tokenize3
    return self.__tokenize3()

'''
Returns the next character in the expression and increment the index
'''
def __next_char(self):

    # If the current index is less than the length of expression
    if self.index < len(self.exp):

        # Getting the character base on the index
        char = self.exp[self.index]

        # Increment the index by 1
        self.index += 1

        # Return the character
        return char

    # Return none here if the end of the expression
    return None

'''
Returns the next character in the expression and without increasing the
    ↪ index
'''
def __peek_char(self):

    # If the current index is less than the length of expression
    if self.index < len(self.exp):
        return self.exp[self.index]
```

```
        # Return none here if the end of the expression
        return None

    '''
    Get the next token here by identifying the numbers, operators and
        ↪ variables
    '''
    def get_next_token(self):

        # While loop such that when the index is less than the length of
            ↪ expression to skip whitespace here
        while self.index < len(self.exp) and self.exp[self.index].isspace():

            # Increment the index by 1 here
            self.index += 1

        # if the index is at the end here
        if self.index == len(self.exp):

            # Return none here
            return None

        # Get the character base on the index
        char = self.exp[self.index]

        # If the exponenet character here, return ** and increment the index
            ↪ by 2
        if char == '*' and self.index + 1 < len(self.exp) and self.exp[self.
            ↪ index + 1] == '*':
            self.index += 2
            return '**'

        # If the character is the other chracters
        if char in '+-*/()':

            # Increment the index by 1 and return the chracter
            self.index += 1
            return char

        # If the character is a digit here or float here
        if char.isdigit() or (char == '-' and self.index + 1 < len(self.exp)
            ↪ and self.exp[self.index + 1].isdigit()):

            # Set the start index and increment the index by 1
            start_index = self.index
            self.index += 1

            # while the index is less than lemgth and it is still digits or
                ↪ floats
            while self.index < len(self.exp) and (self.exp[self.index].isdigit
                ↪ () or self.exp[self.index] == '.'):

                # Increment the index by 1
                self.index += 1

            # Return the expression from start index to the current index - 1
            return self.exp[start_index:self.index]

        # If the chracter is an alphabet
        if char.isalpha():
```

```
            # Set the start index here
            start_index = self.index

            # Increment the index by 1 here
            self.index += 1

            # while loop to check as long as it is a alphabet here
            while self.index < len(self.exp) and self.exp[self.index].isalnum
                ↪ ():
                self.index += 1

            # Return the expression from start index to the current index - 1
            return self.exp[start_index:self.index]

        # Return the chracter if it is not a space and increment the index by
            ↪ 1
        if not char.isspace():

            self.index += 1
            return char

        # Return none if no valid token is found
        return None

    '''
    Tokenizes the expression using the get method
    '''
    def __tokenize3(self):

        # Store the tokens in a list
        tokens = []

        # Looping through the possible tokens
        while True:
            token = self.get_next_token()

            # If no more tokens are found, break the loop
            if token is None:
                break

            # Append it to the tokens list
            tokens.append(token)

        # Return the list of tokens
        return tokens
```

config.py (Done by: Goh Rui Zhuo

```
Configuration = {
    "name":['Goh␣Rui␣Zhuo','Bharathan␣Bhaskaran','Jonathan␣Leo'],
    'class':'DAAA/FT/2B/05',
    'admin_number':['2222329','2201395','2222655'],
    'welcome_message':"""ST1507 DSAA: Evaluating & Sorting Assignment
        ↪ Statements""",
    'module_code':'ST1507',
}
```

error.py (Done by: Goh Rui Zhuo

```
from classes.expressionManager import ExpressionManager
class Error:

    '''
    Validate the choices
    '''
    @staticmethod
    def validate_integer(inputs, minimum, maximum):
        try:
            value = int(inputs)
            # Check whether value input is within the range
            if not int(minimum) <= int(value) <= int(maximum):
                print(f'\nOnly values between {minimum} to {maximum} are
                    ↪ available, please try again:\n')
                return None
            return value
        # Print other errors
        except Exception as e:
            print(f'\nOnly values between {minimum} to {maximum} are available
                ↪ , please try again:\n')
            return None

    '''
    Validate whether the encrypt or decrypt option is press
    '''
    @staticmethod
    def validate_option(self, option, valid):
        if option.lower() not in valid:
            print('Invalid option.')
            return False
        return True

    '''
    Set the option to validate whether integer is press
    '''
    @staticmethod
    def validate_integerType(num):
        try:
            num = int(num)
            return num
        except Exception as e:
            print("Invalid input. Please enter a valid integer.")
            return None

    '''
    check user input
    '''
    @staticmethod
    def validate_text(text):
        if len(text) == 0:
            return None
        else:
            return text

    '''
    Set the option to validate assignment is press
    '''
    @staticmethod
```

```
    def validateAssignment(exp):

        # If equal is not in the expression return the error message here
        if '=' not in exp:
            return '\nThe assignment given did not contain an equal sign here\
                ↪ n', False

        # If the expresssion count of equal more than 1 and return the error
            ↪ message here
        if exp.count('=') > 1:
            return '\nThe assignment contains multiple equal sign , system does
                ↪  not support it \n',False

        # Split the equation after checking the statement is correct input
        var , eqn = exp.split('=',1)

        # If the left side is not a valid variable here
        if not var.isidentifier():
            return '\nThe left side of = must be a valid variable name here\n'
                ↪ , False

        # If the variable is an integer here
        if  var[0].isdigit():
            return '\nVariable name cannot begin with a value\n', False

        # If the equation is not starting or ending with a proper paramthesis
        if not(eqn.startswith('(') and eqn.endswith(')')):
            return '\nThe equation is not fully enclose in the paramthesis
                ↪ here.\n', False

        # If the count of paramthesis of left side and right side
        if eqn.count('(') != eqn.count(')'):
            return '\nThe equation provided are not balance\n', False

        # The equation is empty here
        if '()' in eqn:
            return '\nThe equation given is empty.\n', False

        # If double consective operation here is spotted , return the error
            ↪ message here
        for op in ['--','++','****','//']:
            if op in eqn:
                return '\nConsecutive operations are not allowed.\n', False

        # If the equation is started with an operator here
        if eqn[1] in '*/' or eqn[-2] in '+-*/' or  eqn[1:3] == '**' or eqn
            ↪ [-3:-1] == '**':
            return "\nEquation cannot start or end with an operator.\n",False

        # If the character is not valid here
        for char in eqn:
            if char.lower() not in '0123456789abcdefghilmnopqrstuvwxyz+-*/(). 
                ↪ ':
                return  "\nInvalid character in equation\n", False

        # return no error message if good here
        return '',True

    '''
    Validate circular dependency here to check
    '''
    @staticmethod
```

```
    def validateCircular(datas):
        m = ExpressionManager(datas)
        sorted_variables =   m.topologicalSort()

        # If circular dependency is detected
        if type(sorted_variables) == str:
            return '\nCircular Dependency Occur\n', False

        # If no error here return no error
        return '',True
```

main.py (Done by: Goh Rui Zhuo

```
from program import Program
from config import Configuration

# This is to run the program with main.py
if __name__ == "__main__":
    program = Program(Configuration)
    program.run()
```

option1.py (Done by: Goh Rui Zhuo

```
from tools import Tools
from error import Error

# Create the option 1
class OptionOne:

    '''
    Creating the init with data call in init and also create an initialising
        ↪ an error
    '''
    def __init__(self,datas):
        self.datas = datas
        self.errorVal = Error()

    '''
    Creating the run funcion to run the code for option 1 with the checking of
        ↪  input statement
    '''
    def run(self):

        # Create the try except block
        try:

            # Creating the while true block
            while True:

                # Getting the input of the assignment from the user
                user_input = input('Enter the assignment statement you want to
                    ↪ add/modify ("r" to return):\nFor example, a=(1+2)\n')

                # Provide option to return to main menu
                if user_input.lower() == 'r':
                    return self.datas
```

```
                # Removing the space from the user input
                user_input = user_input.replace(" ", "")

                # Validating the assignment input by the user
                message, isVal =  self.errorVal.validateAssignment(user_input)

                # if the validation is not valid, provide the message of the
                    ↪ error
                if not isVal:
                    print(message)
                    continue
                # If all else pass, split by the equal sign
                key, value = Tools.splitByEqual(user_input)

                # Getting the old value with the key
                oldVal = self.datas.get(key, None)

                # Storing the data into the hash table
                self.datas[key] = value

                # validation the circular statement
                message2, is_valid2 = self.errorVal.validateCircular(self.
                    ↪ datas)

                # if the validation is not valid, provide the message of the
                    ↪ error
                if not is_valid2:

                    # If old val exist then put back into the hashtable
                    if oldVal is not None:
                        self.datas[key] = oldVal
                    else:
                        del self.datas[key]

                    # print the error message here
                    print(message2)
                    continue

                # while block to add more assignment
                while True:
                    user_input2 = input('Do you want co continue adding more
                        ↪ assignments? (Y/N) ')

                    # Validating the input here
                    if user_input2.lower() in ['y','yes']:
                        break
                    elif user_input2.lower() in ['n','no']:
                        return self.datas
                    if user_input2.lower() not in ['y','n','yes','no']:
                        print('Invalid option.')
                        continue

            # Try except block here to print the error
            except Exception as e:
                print(e)
```

option2.py (Done by: Goh Rui Zhuo

```
from tools import Tools
from classes.hashTable import hashTable
```

```
from classes.expressionEvaluator import ExpressionEvaluator
from classes.expressionManager import ExpressionManager
from classes.dependency import DependencyGraph
from classes.parseTree import ParseTreeBuilder
from classes.mergeSort import MergeSort
from classes.equationCleanup import EquationCleanup

# Creating Option Two
class OptionTwo:
    '''
    Creating the init with data call in init and also create an initialising
        ↪ an error
    '''
    def __init__(self,datas):
        self.datas = datas

    '''
    Creating the run funcion to run the code for option 1 with the checking of
        ↪  input statement
    '''
    def run(self,  boolean=True):
        # Creating the expression manager that takes in the data
        m = ExpressionManager(self.datas)

        # Utilise topological sort to sort dependency graph to order the
            ↪ vertices such that every edge is ordered
        sorted_variables =   m.topologicalSort()

        # Getting the values
        variable_values = {}
        for var in sorted_variables:

            # Replacing the variables with values if it is already exist
            exp = m.replaceVar(self.datas[var], variable_values)

            # Cleanup the equation to include correct number of paramthesis
            exp = EquationCleanup(exp).get_cleanup()

            # Building the parse tree here to be able to evaluate next step
            tree = ParseTreeBuilder(exp).build()

            # Getting the results base on the expression evaluator class and
                ↪ get results methods
            value = ExpressionEvaluator().get_results(tree, variable_values)

            # Store the value here
            variable_values[var] = value

        # Use to provide inheritance for option 4
        if boolean:

            # Utilise merge sort to sort the values and items
            sorted_datas = MergeSort().sort( list(variable_values.items()))

            # Printing the results by looping through the sorted data
            for var, exp in sorted_datas:

                # If the variable exist in the keys
                if variable_values[var]:

                    # Conversion of float or integer base on the final values
```

```
                    if int(variable_values[var]) == variable_values[var]:
                        ↪ values = int(variable_values[var])
                    else: values = float(variable_values[var])

                # Printing out the statement here
                print(f'{var}={self.datas[var]}=> {variable_values[var]}')

        # return of results here
        return variable_values
```

option3.py (Done by: Goh Rui Zhuo

```
from tools import Tools
from classes.hashTable import hashTable
from classes.expressionEvaluator import ExpressionEvaluator
from classes.expressionManager import ExpressionManager
from classes.dependency import DependencyGraph
from classes.parseTree import ParseTreeBuilder

# Creating Option 3
class OptionThree:
    '''
    Creating the init with data call in init and also create an initialising
        ↪ an error
    '''
    def __init__(self,datas):
        self.datas = datas

    '''
    Creating the run funcion to run the code for option 1 with the checking of
        ↪  input statement
    '''
    def run(self):

        # While loop  to check for error handling
        while True:

            # Get the user to key in the variable to evaluate
            variable = input('Please enter the variable you want to evaluate
                ↪ ("()" to return):\n')

            # If the user want to return
            if variable == '()':
                return

            # If the variable does not exist
            if variable not in self.datas.keys:
                print(f'\nNo variable of {variable} available.\n')
                continue

            # Creating the expression manager that takes in the data
            m = ExpressionManager(self.datas)

            # Utilise topological sort to sort dependency graph to order the
                ↪ vertices such that every edge is ordered
            sorted_variables =   m.topologicalSort()

            # replacing the variables with values if it exist
            variable_values = {}
```

```
            original_exp = m.replaceVar(self.datas[variable], variable_values)

            # Building the original parse tree
            original_tree = ParseTreeBuilder(original_exp).build()

            # Getting the values here
            for var in sorted_variables:

                # Replacing the variables with the values if defined
                exp = m.replaceVar(self.datas[var], variable_values)

                # Building the parse tree for evaluation
                tree = ParseTreeBuilder(exp).build()

                # Getting the value through the use of get results
                value = ExpressionEvaluator().get_results(tree,
                    ↪ variable_values)

                # Storing the values here
                variable_values[var] = value

                # If the variable is the required variable, return
                if var == variable:

                    # Printint the expression tree
                    print(f'\nExpression␣Tree:')
                    original_tree.printInorder(0)

                    # Print the value heres
                    print(f'Value␣for␣"{variable}"␣is␣{value}')
                    return
```

option4.py (Done by: Goh Rui Zhuo

```
from tools import Tools
from classes.hashTable import hashTable
from classes.expressionEvaluator import ExpressionEvaluator
from classes.expressionManager import ExpressionManager
from classes.dependency import DependencyGraph
from classes.parseTree import ParseTreeBuilder
from classes.mergeSort import MergeSort
from option2 import OptionTwo
from error import Error

# Creating Option 4 with inheritance to option 2
class OptionFour(OptionTwo):
    '''
    Creating the init with data call in init and also create an initialising
        ↪ an error
    '''
    def __init__(self , datas):
        self.datas = datas
        self.errorVal = Error()

    '''
    Creating the get data function that retrieves all the file contents and
        ↪ evaluate it
    '''
    def get_data(self):
```

```
        while  True:
            # Get the file path here
            file_path = input('Please␣enter␣input␣file␣("r"␣to␣return):␣')

            # If user wants to return to main menu
            if file_path == 'r':
                return self.datas

            # Make sure that the file path exist and open file path
            while Tools.open_file(file_path):

                # Getting the file contents here
                text = Tools.open_file(file_path)

                # Splitting the file contents by new line
                text = text.split('\n')

                # For each new line in the file contents
                for line in text:

                    # Strip the line of adding leading spaces
                    line = line.strip()

                    # If the line is empty then continue to the next line here
                    if not line:
                        continue

                    # Try except block to fully handle error
                    try:

                        # while loop to handle errors here
                        while True:

                            # Error message and is validate was check here for
                                ↪  the key errors,
                            # errors are return and continue to the next line
                            message, isVal =  self.errorVal.validateAssignment
                                ↪ (line)
                            if not isVal:
                                print(message)
                                break

                            # Proceed to split by the equal sign here
                            key, value = Tools.splitByEqual(line)

                            # Stores the old values in case of possible
                                ↪ circular dependency
                            oldVal = self.datas.get(key, None)

                            # Storing the value here
                            self.datas[key] = value

                            # Check for circular dependency
                            message2, is_valid2 = self.errorVal.
                                ↪ validateCircular(self.datas)

                            # If it's a circular dependency error here
                            if not is_valid2:

                                # If the old value exist
                                if oldVal is not None:
```

```
                    # Store the old value
                    self.datas[key] = oldVal
                else:

                    # Delete the key from the keys
                    del self.datas[key]

                # Print out the error message
                print(message2)

                # Here is to check whether the user want to
                    ↪ redefine the variable
                while True:

                    # Getting the input on whether to redefine
                    inputQus = input(f'Do␣you␣want␣to␣redefine
                        ↪ ␣the␣variable␣{key}␣again?␣(Y/N):␣'
                        ↪ )

                    # if the users agree to redefine
                    if inputQus.lower() in ['y', 'yes']:

                        # Ask for the new assignment and user
                            ↪ select their decision
                        while True:

                            # Key in the assignment and no to
                                ↪ cancel
                            line = input('Key␣in␣the␣new␣
                                ↪ assignment␣statement␣for␣
                                ↪ the␣variable␣(N␣to␣cancel)
                                ↪ :\n')

                            # If the user press no, go to the
                                ↪ next line
                            if line.lower() == 'N':
                                break

                            # Checking whether the new line
                                ↪ entered is valid or not
                            message, isVal =  self.errorVal.
                                ↪ validateAssignment(line)

                            # If the new line is not valid
                                ↪ continue to the while loop
                                ↪ to ask the questions again
                            if not isVal:

                                # Print out the error message
                                    ↪ if it is not valid
                                print(message)
                                continue

                            # If it is valid, split the line
                                ↪ by the equal sign here
                            key2, value = Tools.splitByEqual(
                                ↪ line)

                            # If the key provided is not equal
                                ↪  to the original key in the
                                ↪  text, ask the user to key
                                ↪ in again
```

```
                                    if key2 != key:

                                        # Continue prompting and if
                                            ↪ the user want to stop
                                            ↪ they can and continue
                                            ↪ to the next line
                                        var  = input('Wrong key.
                                            ↪ Please enter the
                                            ↪ correct key to redefine
                                            ↪ or N to cancel.')
                                        if var == 'N':
                                            break

                                        # Continue prompting the
                                            ↪ keying in of quesion
                                        continue

                                    # Break if the key is equal to
                                        ↪ continue to the evaluation
                                        ↪ and next line
                                    else:
                                        break
                                break

                            # If the user input is no then break too
                            elif inputQus.lower() in ['n', 'no']:
                                break

                            # If the user input is neither in the
                                ↪ option, return invalid option here
                            if inputQus.lower() not in ['y', 'n', 'yes
                                ↪ ', 'no']:
                                print('Invalid option.')
                                continue

                        # If the user input is no then break here too
                        if inputQus.lower() in ['n', 'no']:
                            break

                    # Break if the sentence is valid and correct
                    else:
                        break

                # Except block to handle other possible error
                except Exception as e:
                    print(e)

            # Break the loop if it get to this point
            break
        break
    # Return the stored data to the hashtable in the main program
    return self.datas


'''
Run the full option here with inheritance of function from option 2
'''
def run(self):
    results = super().run(self.datas)
```

option5.py (Done by: Goh Rui Zhuo

```
from tools import Tools
from classes.hashTable import hashTable
from classes.expressionEvaluator import ExpressionEvaluator
from classes.expressionManager import ExpressionManager
from classes.dependency import DependencyGraph
from classes.parseTree import ParseTreeBuilder
from classes.mergeSort import MergeSort
from option2 import OptionTwo
import os

# Creating option 5 inheriting option 2
class OptionFive(OptionTwo):
    '''
    Creating the init with data call in init and also create an initialising
        ↪ an error
    '''
    def __init__(self,datas):
        super().__init__(datas)
        self.datas = datas

    '''
    Creating the run function to run the full code
    '''
    def run(self):

        # Inherit from option two the run option and getting the evaluated
            ↪ results
        results = super().run( boolean=False)

        # Retriving the value from the result and storing it as a list here
        vals = list(results.values())

        # If the storage is empty, return the program to the user
        if len(vals) == 0:
            print('\nThe␣storage␣is␣empty.␣Please␣add␣statements␣before␣saving
                ↪ ␣it.')
            return

        # Creating a filtered list to store the item here and removing
            ↪ repeated values
        filtered_list = [item for item in vals if item is not None]
        filtered_listUnique = list(set(filtered_list))

        # Using custom sort function, i sort the values here
        vals = MergeSort().sort(filtered_listUnique)[::-1] + [None] * (1 if
            ↪ len(vals) - len(filtered_list) > 0 else 0)

        # Initialising the string statement to store in the file
        statement = ''
        for i in (vals):

            # Creating the line for each statement here
            statement += f'***␣Statements␣with␣value=>␣{i}\n'

            # getting the keys that matches with the value
            keys = [key for key, val in results.items() if (i is val or i==
                ↪ val)]
```

```
        # Sorting the keys if the length of keys is more than or equal to
            ↪ 2
        keys = MergeSort().sort(keys)

        # Storing it into the list
        statement_list = [f'{var}={self.datas[var]}' for var in keys]

        # Adding to the original statement for storing later on
        statement += '\n'.join(statement_list) + '\n'*2

    # Statement remove the last two characters in the list
    statement = statement[:-2]

    # Creating the while loop
    while True:

        # Enter the output file for storage of contents
        file = input(f'\nPlease␣enter␣output␣file␣("r"␣to␣return,␣"b"␣to␣
            ↪ menu):␣')

        # If the r button is press, return to previous question
        if file.lower() == 'r':
            break

        # If the b button is pressed, return straight to main menu here
        if file.lower() == 'b':
            return

        # Try except block to check for all possible errors
        try:

            # Check if file path exist
            if os.path.exists(file):

                # Ask if user wants to continue
                while True:

                    # IF the file type provided is not acceptable, prompt
                        ↪ the user to input the filetype again
                    if file[-4:] !=  '.txt'  and file[-3:] != '.md':
                        print('File␣type␣not␣supported␣')
                        break

                    # If the file already exist, ask the user to overwite
                        ↪ it
                    question = input(f'Filename␣{file}␣already␣exists.␣Do␣
                        ↪ you␣want␣to␣overwrite␣it?␣(y/n,␣"r"␣to␣return,␣
                        ↪ "b"␣to␣menu):␣')

                    # If the r button is press, return to previous
                        ↪ question
                    if question.lower() == 'r':
                        break

                    # If the b button is pressed, return straight to main
                        ↪ menu here
                    if question.lower() == 'b':
                        return

                    # IF the user wants to cancle the operation
                    elif question.lower() == 'n':
                        print("Operation␣cancelled.")
```

```
                        break

                    # Overwrite file if the user input yes
                    elif question.lower() == 'y':

                        # Proceeed to write file to the file here
                        Tools.write_file(file , statement)
                        print("File overwritten successfully.")
                        return

                    # Return invalid option if invalid buttons was pressed
                    else:
                        print('Invalid Option')
                        continue

            # Write file if no error found
            else:

                # Create a inner while loop
                while True:

                    # IF the file type provided is not acceptable, prompt
                        ↪ the user to input the filetype again
                    if file[-4:] !=  '.txt'  and file[-3:] != '.md':
                        print('File type not supported ')
                        break

                    # Write file to the file path with the statement
                    else:
                        Tools.write_file(file , statement)
                        return

        # Exception to handle other errors
        except Exception as e:
            print(f"An error occurred while writing to the file: {e}")
            return
```

program.py (Done by: Goh Rui Zhuo

```
from userMenu import UserMenu
from tools import Tools
from user import User
from classes.stack import Stack
from option1 import OptionOne
from option2 import OptionTwo
from option3 import OptionThree
from option4 import OptionFour
from option5 import OptionFive
from feature1 import Feature1
from feature2 import Feature2
from classes.hashTable import hashTable
import os

class Program(UserMenu):
    '''
    Creating the inheritance here to username
    '''
    def __init__(self, config):
        # Create the inheritance here
```

```
        super().__init__()
        self.config = config

        # creating the hash table here
        self.datas =  hashTable()

    '''
    Creating the set details function of the details
    '''
    def set_details(self):
        users = User()
        users.set_name(self.config['name'])
        users.set_adminNo(self.config['admin_number'])
        users.set_class(self.config['class'])
        users.set_moduleCode(self.config['module_code'])
        users.set_info(self.config['welcome_message'])
        return users

    '''
        Setting all the run methods here
    '''
    def run(self):
        userObj = self.set_details()

        # Setting the starting page

        print(userObj)
        while True:
            # Define the data structure
            # Setting the different options

            self.choices = ['Add/Modify␣assignment␣statement', 'Display␣
                ↪ current␣assignment␣statements',
                            'Evaluate␣a␣single␣variable', 'Read␣assignment␣
                                ↪ statements␣from␣file',
                            'Sort␣assignment␣statements','Feature␣1␣(Bharathan
                                ↪ )','Feature2␣(Bharathan)','Exit']
            press_enter =   input("\nPress␣enter␣key,␣to␣continue...\n")

            # If the enter key is press here
            if press_enter == '' or press_enter:

                # Getting the user to input their choice
                user_choice = self.get_choice()

                # if the option choosen is one, run the run function here
                if user_choice == 1:
                    self.datas = OptionOne(self.datas).run()

                # if the option choosen is two, run the run function here
                elif user_choice == 2:
                    print('\nCURRENT␣ASSIGNMENT:')
                    print('*'*20)
                    OptionTwo(self.datas).run()

                # if the option choosen is three, run the run function here
                elif user_choice == 3:
                    OptionThree(self.datas).run()

                # if the option choosen is four, run the run function here
                elif user_choice == 4:
                    self.datas = OptionFour(self.datas).get_data()
```

```
                print('\nCURRENT ASSIGNMENT:')
                print('*'*20)
                OptionFour(self.datas).run()

            # if the option choosen is five, run the run function here
            elif user_choice == 5:
                OptionFive(self.datas).run()

            # if the option choosen is six, run the run function here
            elif user_choice == 6:
                Feature1(self.datas).run()

            # if the option choosen is seven, run the run function here
            elif user_choice == 7:
                Feature2(self.datas).run()

            # if the option choosen is eight, run the run function here
            elif user_choice == 8:
                print(f'\nBye, thanks for using {userObj.get_moduleCode()}
                    ↪  DSAA: Assignment Statement Evaluator & Sorter')
                return
```

tools.py (Done by: Goh Rui Zhuo

```
'''
Reasons for static method here: the use of tools class are not correlated
        so by putting it as a class function it is not suitable and does not
            ↪ make sense
'''


import matplotlib as plt
import os
from error import Error


class Tools:
    '''
    Set the split by equal file method here as static method
    '''
    @staticmethod
    def splitByEqual(statement):
        key, value = statement.split('=',1)
        return key.strip(), value.strip()

    '''
    Set the open file method here as static method
    '''
    @staticmethod
    def open_file( path):
        # If file has a wrong type
        if not path.endswith('.txt'):
            print(f'File with ".txt" are supported only.')
            return None

        # If file does not exist
        if not os.path.exists(path):
            print(f'File {path} does not exist')
            return None
```

```
        # If the file is empty, prompt the user again
        if os.stat(path).st_size == 0:
            print('File is empty. Please try again.')
            return None

        # Try opening the file
        try:
            with open(path, 'r') as f:
                return f.read()
        # Unexpected error handling
        except Exception as e:
            print('Unexpected error occured')
            return None

    '''
    Set the write file method here as static method
    '''
    @staticmethod
    def write_file(path, text):
        with open(path, 'w') as f:
            f.write(text)

    '''
    Set the get valid integer method here as static method
    '''
    @staticmethod
    def get_valid_integer(prompt):
        while True:
            user_input = input(prompt)
            valid = Error.validate_integerType(user_input)
            if valid is not None:
                return valid

    @staticmethod
    def delete_assignment(self, variable):
        if variable in self.assignments:
            del self.assignments[variable]
            print(f"Assignment statement for variable '{variable}' deleted.")
        else:
            print(f"Variable '{variable}' not found. No assignment statement
                ↪ deleted.")
```

user.py (Done by: Goh Rui Zhuo

```
class User:
    def __init__(self):
        # Setting the name and other infomation variable for start menu
        self.__name = None
        self.__admin_no = None
        self.__module_code = None
        self.__class = None
        self.__info = None

    '''
    Settting the name of the user
    '''
    def set_name(self, name):
        self.__name = name

    '''
```

```
    Setting the admin number of the user
    '''
    def set_adminNo(self, admin_no):
        self.__admin_no = admin_no

    '''
    Setting the module code of this project
    '''
    def set_moduleCode(self, module_code):
        self.__module_code = module_code

    '''
    Setting the class
    '''
    def set_class(self, className):
        self.__class = className

    '''
    Setting the infomation
    '''
    def set_info(self, info):
        self.__info = info

    '''
    Get the name of ther user
    '''
    def get_names(self):
        return self.__name

    '''
    Get the admin Number
    '''
    def get_adminNos(self):
        return self.__admin_no

    '''
    Get the module code
    '''
    def get_moduleCode(self):
        return self.__module_code

    '''
    Get the class
    '''
    def get_class(self):
        return self.__class

    '''
    Get the info
    '''
    def get_info(self):
        return self.__info

    '''
    Setting the overloading print function
    '''
    def __str__(self):

        # Setting the style of name when displaying it here

        namePairs = "␣&␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣*\n*␣␣␣␣␣␣␣␣␣␣␣␣␣".join(
```

```
            f"{name}({admin_no})" for name, admin_no in zip(self.get_names(),
                ↪ self.get_adminNos())
        )

        namePairs = namePairs[:90] + namePairs[97:]

        # Return the the final style here
        return (f"\n{'*'␣*␣59}\n*␣{self.get_info()}␣*\n{'*'␣+␣'-'␣*␣57␣+␣'*'}\
            ↪ n"
                f"{'*'␣+␣'␣'␣*␣57␣+␣'*'}\n"
                f"*␣␣-␣Done␣by:␣{namePairs}{'␣'␣*␣(23)}*\n"
                f"*␣␣-␣Class:␣{self.get_class()}{'␣'␣*␣(56␣-␣len(self.
                    ↪ get_class())␣-␣10)}*\n"
                f"{'*'␣+␣'␣'␣*␣57␣+␣'*'}\n"
                f"{'*'␣*␣59}\n")
```

userMenu.py (Done by: Goh Rui Zhuo

```
from user import User
from error import Error

class UserMenu:
    def __int__(self, choices):
        self.choices = choices

    '''
     Set the choices
    '''
    # Set the choices
    def display_choices(self):
        # Looping through to display the different options
        statements = ''
        for index, choice in enumerate(self.choices, start = 1):
            statements += (f'\t{index}.␣{choice}\n')
        return statements[:-1]

    '''
     Get the choice from the user
    '''
    def get_choice(self):
        minimum, maximum = 1, len(self.choices)

        # Getting the user input
        while True:
            user_input = input(f"Please␣select␣your␣choice:␣{str(tuple(f'{i}'␣
                ↪ for␣i␣in␣range(1,len(self.choices)+1))).replace('␣',␣'')␣
                ↪ }:\n" + self.display_choices() + '\nEnter␣choice:␣' ).strip
                ↪ ()

            # Validate selected option
            choice = Error.validate_integer(user_input, minimum, maximum)

            # If the choice is not none here, return the full model
            if choice is not None:
                return choice
```

9.2 Appendix B: References

List the references, papers, and other resources consulted during the project.