Logo

Description automatically generated

GROUP ASSIGNMENT

TECHNOLOGY PARK MALAYSIA

CT077-3-2 DSTR

DATA STRUCTURES

APU2F2109CS(DA)

HAND OUT DATE: 28 – MARCH – 2022

HAND IN DATE: 30 – MAY – 2022

WEIGHTAGE: 50%

|  |  |
| --- | --- |
| GROUP MEMBER | TP NUMBER |
| MARCELL AGUNG WAHYUDI | TP058650 |
| ROS AMISYA FATIHA BINTI MOHD RIZAL | TP062386 |
| MARSHA BINTI KHAIRIL MUBIN | TP062399 |

INSTRUCTIONS TO CANDIDATES:

1. Submit your assignment at the Moodle System.
2. Students are advised to underpin their answers with the use of references (cited using the APA Style System of Referencing).
3. Late submission will be awarded zero (0) unless Extenuating Circumstances (EC) are upheld.
4. Cases of plagiarism will be penalized.
5. The assignment should be submitted in softcopy, where the softcopy of the written assignment and source code (where appropriate) should be on Moodle System.
6. You must obtain 50% overall to pass this module.

**Table of Contents**

[1.0 Introduction 3](#_Toc102207208)

[1.1 System Introduction 3](#_Toc102207209)

[1.2 Data Structures and Classes 4](#_Toc102207210)

[1.3 Justification 6](#_Toc102207211)

[2.0 System Design and Workflows 7](#_Toc102207212)

[2.1 Pseudocode 7](#_Toc102207213)

[2.1.1 Add a Tutor Record 7](#_Toc102207214)

[2.1.2 Display All Records 11](#_Toc102207215)

[2.1.3 Search a Tutor by Tutor ID 13](#_Toc102207216)

[2.1.4 Search Tutors by overall performance (Rating) 15](#_Toc102207217)

[2.1.5 Sort and display by Tutors ID in ascending order 19](#_Toc102207218)

[2.1.6 Sort and display by Tutors Hourly Pay Rate in ascending order 21](#_Toc102207219)

[2.1.7 Sort and display by Tutors Overall Performance in ascending order 23](#_Toc102207220)

[2.1.8 Modify a Tutor Record 25](#_Toc102207221)

[2.1.9 Delete a Tutor Record 29](#_Toc102207222)

[2.2 Explanation of Algorithms 33](#_Toc102207223)

[2.3 Justification on Algorithms 33](#_Toc102207224)

[3.0 Task Distribution 34](#_Toc102207225)

[4. 0 References 35](#_Toc102207226)

# 1.0 Introduction

## 1.1 System Introduction

This Data Structures (DSTR) Assignment aimed to develop a computerized tutor management system to replace the manual filling system that eXcel Tuition Centre used to implement in the past. Hence, this proposal will explain the team’s proposed data structure along with the related algorithms that will be applied in developing the Tutors Management System using C++ program. The proposal will include the data members that will be included in the structure, workflows of the system as well as the implemented algorithms.

The Tutors Management System will have few interesting features that not only allow the Tuition Centre Head Quarter Human Resources Department to add, display, modify and delete tutor records but also enables them to search for a tutor based on their unique Tutor ID and overall performance (rating). In addition to that, the system will also be able to ascendingly sort and display the tutors’ records by Tutors ID, Tutors Hourly Pay Rate and Tutors Overall Performance.

One of the assumptions that the team made while developing the system is that eXcel Tuition Centre has 3 branches, each located in Bukit Jalil, Ampang and Cheras respectively. The system will store the record of each tutor by their Tutor ID, Name, Date Joined, Date Terminated, Hourly Pay Rate, Phone, Address, Tuition Centre Code, Tuition Centre Name, Subject Code, Subject Name and Rating. The subjects that are available in the tuition centre are Math, English, Science, Art and Sejarah. For the tutor’s rating, it will be rank from 1 to 5, with 1 indicates “very poor performance” and 5 indicates “excellent performance”. A tutor can only teach one subject to ensure that the tutor can give their best teaching in the selected subject.

Though the system allows the record of a tutor to be modified, only tutor’s phone number and address attributes can be modified. Another restriction concerns the deletion of tutor’s records, in which can only be done if the tutor has left the Tuition Centre for more than 6 months. To ensure better record keeping practice, a report will be generated at the end of each week where it will list down all tutors that are available in the tuition centre. This report helps to check whether a new vacancy spot is available should a new tutor wishes to register themselves at the tuition centre.

## 1.2 Data Structures and Classes

The system is proposed to have one data structure which is the struct (structure), it is constructed using one struct only to maximize efficiency of memory whilst also lessens complexity of the code. structure will be for Tutor as tutor more than one variable type data.

Text

Description automatically generated

*Figure 1 Tutor Structure in Linked List Data Structure System*

Figure 1 represents the tutor data structure for linked list, the structure is a singly linked list which is mainly beneficial in terms of memory conservation better than doubly linked list. The tutor structure consists of various variables such as integer variable id, double variables hourlyRate and rating, string variables name, dateJoined, dateTerminated, phone, address, tcCode, tcName, subjectCode and subjectName, and Tutor pointer variable next which will be pointing to the next position in the linked list.

Text

Description automatically generated

*Figure 2 Tutor Structure in Array List Data Structure System*

Figure 2 displays the same tutor structure as the linked list version with the only difference is with the array list system, the structure doesn’t have the tutor structure pointer variable, this is due to the nature concept of array list which stores its members contiguously, as in when traversing the array list, the next member is located next to the present member, so on and so forth for the size of the array of structure, which in this case is 100.

What differentiates linked list and array list system is that linked list system is a little bit more complex since the members/elements stored in the linked list are scattered around throughout the memory, and to access them, the program needs access of the previous member, which has the pointer value for the member that is sought for. But the benefit of linked list is that the memory is dynamically allocated, meaning it can add an infinite number of members, since, when an element is added, it can be inserted into any location in the memory, whereas in array list, since it is statically allocated, the size of the array list has to be initially stated first, and later can’t be changed.

## 1.3 Justification

In terms of data structures, the 2 types that are to be implemented in different systems for the assignment is array and linked list. Both are linear in nature, where each element of both data structures is connected to the previous and next one (GeeksforGeeks, 2022).

Arrays and linked lists are efficient in their own ways. In the case of array data structures, its size is fixed and static. Having said that, arrays are easily accessible in comparison to linked lists. Hence, when the number of elements is known, arrays are more suitable. Even more so, the data structure takes up less memory per element, since it has no need for pointers and does not store references to other nodes (Chan, 2020). Other than that, arrays are more ideal for usage when a lot of random accessing will be done, using the index number of elements. In addition, searching processes are more easily applied on arrays. Nonetheless, arrays do have the disadvantage of having fixed memory allocation, which cannot be altered, increased or decreased in size (GeeksforGeeks, 2020). To counter the issue, linked lists will be implemented in the other version of the tuition centre’s system.

The selected type of linked list is singly linked list, which is the simplest of the variety. In comparison to arrays, the linked list data structure is more efficient when dynamic expansion and addition of the elements is needed. For example, to delete an element at any given position. The only requirement being, each node in the stack must be linked to the next, to prevent spoilage of linkage. Aside from that, linked lists are faster, in terms of deleting and inserting elements, due to its dynamic quality where allocating and deallocating memory is done during execution (Krohn, 2019).

There are several reasons why singly linked list was chosen. One of these being less memory is occupied in comparison to doubly linked lists. Although traversal of data can only be done one way, each node must only store two fields, data and link. The link being the address of the next node in the sequence. Moreover, as previously mentioned, the dynamic arrangement of a singly linked list produces no memory wastage. Efficient memory utilization is achieved and there is no need to pre-allocate memory (GeeksforGeeks, 2022). Similarly, no shifting of the elements is needed during modification, only that the address present in the next pointer is updated. As such, the complexity of a singly linked list is less than a doubly linked list, with the big-O notation of o(n) (Vaghani, 2022).

# 2.0 System Design and Workflows

## 2.1 Pseudocode

### 2.1.1 Add a Tutor Record

*Using Linked List*

FUNCTION inputTutor()

START

DECLARE Integer id

DECLARE String name

DECLARE String dateJoined

DECLARE String dateTerminated

DECLARE Double hourlyRate

DECLARE String phone

DECLARE String address

DECLARE String tcCode

DECLARE String tcName

DECLARE String subjectCode

DECLARE String subjectName

DECLARE Double rating

DISPLAY “Add Tutor Record”

DISPLAY “==================”

DISPLAY “Tutor ID: ”

READ id

DISPLAY “Name: ”

READ name

DISPLAY “Date Joined: ”

READ dateJoined

DISPLAY “Date Terminated: ”

READ dateTerminated

DISPLAY “Hourly Rate: ”

READ hourlyRate

DISPLAY “Phone: ”

READ phone

DISPLAY “Address: ”

READ address

DISPLAY “Tuition Centre Code: ”

READ tcCode

DISPLAY “Tuition Centre Name: ”

READ tcName

DISPLAY “Subject Code: ”

READ subjectCode

DISPLAY “Subject Name: ”

READ subjectName

DISPLAY “Rating: ”

READ rating

TUTOR POINTER VARIABLE tutor SET TO TUTOR POINTER VARIABLE addTutor (id, name, dateJoined, dateTerminated, hourlyRate, phone, address, tcCode, tcName, subjectCode, subjectName, rating)

CALL FUNCTION insertTutor PASS TUTOR POINTER VARIABLE tutor

DISPLAY “Returning to Main Menu”

CALL FUNCTION mainMenu()

END

START

DECLARE TUTOR POINTER VARIABLE addTutor (Integer id, String name, String dateJoined, String dateTerminated, Double hourlyRate, String phone, String address, String tcCode, String tcName, String subjectCode, String subjectName, Double rating)

DECLARE TUTOR POINTER VARIABLE newTutor SET TO new Tutor

SET newTutor->id TO id

SET newTutor->name TO name

SET newTutor->dateJoined TO dateJoined

SET newTutor->dateTerminated TO dateTerminated

SET newTutor->hourlyRate TO hourlyRate

SET newTutor->phone TO phone

SET newTutor->address TO address

SET newTutor->tcCode TO tcCode

SET newTutor->tcName TO tcName

SET newTutor->subjectCode TO subjectCode

SET newTutor->subjectName TO subjectName

SET newTutor->rating TO rating

SET newTutor->next TO NULL

RETURN newTutor

END

FUNCTION insertTutor()

START

DECLARE TUTOR POINTER VARIABLE tutor

IF (head == NULL)

SET head TO tutor

ELSE

TUTOR POINTER VARIABLE current SET TO head

WHILE (current->next != NULL)

SET current TO current->next

END WHILE

SET current->next TO tutor

END IF

END

*Using Array List*

FUNCTION inputTutor()

START

DECLARE Integer id

DECLARE String name

DECLARE String dateJoined

DECLARE String dateTerminated

DECLARE Double hourlyRate

DECLARE String phone

DECLARE String address

DECLARE String tcCode

DECLARE String tcName

DECLARE String subjectCode

DECLARE String subjectName

DECLARE Double rating

DISPLAY “Add Tutor Record”

DISPLAY “==================”

DISPLAY “Tutor ID: ”

READ id

DISPLAY “Name: ”

READ name

DISPLAY “Date Joined: ”

READ dateJoined

DISPLAY “Date Terminated: ”

READ dateTerminated

DISPLAY “Hourly Rate: ”

READ hourlyRate

DISPLAY “Phone: ”

READ phone

DISPLAY “Address: ”

READ address

DISPLAY “Tuition Centre Code: ”

READ tcCode

DISPLAY “Tuition Centre Name: ”

READ tcName

DISPLAY “Subject Code: ”

READ subjectCode

DISPLAY “Subject Name: ”

READ subjectName

DISPLAY “Rating: ”

READ rating

DECLARE STRUCTURE TUTOR VARIABLE tutor = {id, name, dateJoined, dateTerminated, hourlyRate, phone, address, tcCode, tcName, subjectCode, subjectName, rating]

CALL FUNCTION insertTutor PASS STRUCTURE TUTOR VARIABLE tutor

DISPLAY “Returning to Main Menu”

CALL FUNCTION mainMenu()

END

FUNCTION insertTutor()

START

PASS STRUCTURE TUTOR VARIABLE tutor

::allTutor[::insertingIndex] = tutor

::insertingIndex++

END

### 2.1.2 Display All Records

*Using Linked List*

START

DECLARE TUTOR POINTER VARIABLE current, SET to head

DISPLAY “Displaying All Tutors”

WHILE(current != NULL)

Display “Tutor ID : ” SET current->id

Display “Name : ” SET current->name

Display “Date Joined : ” SET current->dateJoined

Display “Date Terminated : ” SET current->dateTerminated

Display “Hourly Rate : ” SET current->hourlyRate

Display “Phone : ” SET current->phone

Display “Address : ” SET current->address

Display “Tuition Centre Code : ” SET current->tcCode

Display “Tuition Centre Name : ” SET current->tcName

Display “Subject Code : ” SET current->subjectCode

Display “Subject Name : ” SET current->subjectName

Display “Rating : ” SET current->rating

Set current TO current->next

END WHILE

DISPLAY “============”

DISPLAY “End of List”

DISPLAY “============”

DISPLAY “Returning to Main Menu”

CALL FUNCTION mainMenu()

END

*Using Array List*

START

DISPLAY “Displaying All Tutors”

FOR(int i = 0; ::allTutor[i].id != 0 ; i++)

Display “Tutor ID : ” SET ::allTutor[i].id

Display “Name : ” SET ::allTutor[i].name

Display “Date Joined : ” SET ::allTutor[i].dateJoined

Display “Date Terminated : ” SET ::allTutor[i].dateTerminated

Display “Hourly Rate : ” SET ::allTutor[i].hourlyRate

Display “Phone : ” SET ::allTutor[i].phone

Display “Address : ” SET ::allTutor[i].address

Display “Tuition Centre Code : ” SET ::allTutor[i].tcCode

Display “Tuition Centre Name : ” SET ::allTutor[i].tcName

Display “Subject Code : ” SET ::allTutor[i].subjectCode

Display “Subject Name : ” SET ::allTutor[i].subjectName

Display “Rating : ” SET ::allTutor[i].rating

END FOR LOOP

DISPLAY “============”

DISPLAY “End of List”

DISPLAY “============”

DISPLAY “Returning to Main Menu”

CALL FUNCTION mainMenu()

END

### 2.1.3 Search a Tutor by Tutor ID

*Using Linked List*

FUNCTION searchTutorID()

START

DECLARE TUTOR POINTER VARIABLE current TO head

DECLARE Integer id

DECLARE Boolean found SET TO false

DECLARE Integer input

DISPLAY “Search a Tutor by Tutor ID”

DISPLAY “============================”

DISPLAY “Enter Tutor ID : ”

READ id

WHILE (current != NULL)

IF (current->id == id)

DISPLAY “Tutor ID: ” SET current->id

DISPLAY “Name: ” SET current->name

DISPLAY “Date Joined: ” SET current->dateJoined

DISPLAY “Date Terminated: ” SET current->dateTerminated

DISPLAY “Hourly Rate: ” SET current->hourlyRate

DISPLAY “Phone: ” SET current->phone

DISPLAY “Address: ” SET current->address

DISPLAY “Tuition Code Centre: ” SET current->tcCode

DISPLAY “Tuition Centre Name: ” SET current->tcName

DISPLAY “Subject Code: ” SET current->subjectCode

DISPLAY “Subject Name: ” SET current->subjectName

DISPLAY “Rating: ” SET current->rating

SET found TO true

BREAK

ELSE IF (current->id != id)

SET current TO current->next

END IF

END WHILE

IF (found == false)

DISPLAY “Unable to find tutor ID, press 1 to try again, or any to go back : ”

READ input

IF (input == 1)

CALL FUNCTION searchTutorID()

ELSE

RETURN

END IF

END IF

END

*Using Array List*

FUNCTION searchTutorID()

START

DECLARE Integer id

DECLARE Boolean found SET TO false

DECLARE Integer input

DSIPLAY “Search a Tutor by Tutor ID”

DISPLAY “=============================”

DISPLAY “Enter Tutor ID : ”

READ id

FOR (int i = 0; ::allTutor[i].id != 0l i++)

IF (::allTutor[i].id == id)

DISPLAY “Tutor ID: ” SET allTutor[i].id

DISPLAY “Name: ” SET allTutor[i].name

DISPLAY “Date Joined: ” SET allTutor[i].dateJoined

DISPLAY “Date Terminated: ” SET allTutor[i].dateTerminated

DISPLAY “Hourly Rate: ” SET allTutor[i].hourlyRate

DISPLAY “Phone: ” SET allTutor[i].phone

DISPLAY “Address: ” SET allTutor[i].address

DISPLAY “Tuition Code Centre: ” SET allTutor[i].tcCode

DISPLAY “Tuition Centre Name: ” SET allTutor[i].tcName

DISPLAY “Subject Code: ” SET allTutor[i].subjectCode

DISPLAY “Subject Name: ” SET allTutor[i].subjectName

DISPLAY “Rating: ” SET allTutor[i].rating

SET found TO true

BREAK

END IF

END FOR

IF (found == false)

DISPLAY “Unable to find tutor ID, press 1 to try again, or any to go back : ”

READ input

IF (input == 1)

CALL FUNCTION searchTutorID()

ELSE

RETURN

END IF

ELSE IF (found == true)

DISPLAY “Tutor Found, Returning to Main Menu”

RETURN

END IF

END

### 2.1.4 Search Tutors by overall performance (Rating)

*Using Linked List*

START

DECLARE TUTOR POINTER VARIABLE current, SET to head

DECLARE Double rating

DECLARE Boolean found SET to false

DECLARE Integer input

DISPLAY “Search a Tutor by Overall Performance (Rating)

DISPLAY “Enter Tutor Rating: ”

READ rating

WHILE(current != NULL)

IF(current->rating = = rating)

Display “Tutor ID : ” SET current->id

Display “Name : ” SET current->name

Display “Date Joined : ” SET current->dateJoined

Display “Date Terminated : ” SET current->dateTerminated

Display “Hourly Rate : ” SET current->hourlyRate

Display “Phone : ” SET current->phone

Display “Address : ” SET current->address

Display “Tuition Centre Code : ” SET current->tcCode

Display “Tuition Centre Name : ” SET current->tcName

Display “Subject Code : ” SET current->subjectCode

Display “Subject Name : ” SET current->subjectName

Display “Rating : ” SET current->rating

SET found TO true

SET current TO current->next

ELSE IF(current->rating != rating)

SET current TO current->next

END IF

END WHILE

IF (found = = false)

DISPLAY “Unable to find Tutor ID, press 1 to try again, or any to go back: ”

READ input

IF (input = = 1)

CALL FUNCTION searchTutorRating()

ELSE

RETURN

END IF

END IF

RETURN

END

*Using Array List*

START

DECLARE Boolean found SET TO false

DECLARE Integer rating

DECLARE Integer input

DISPLAY “Search Tutors by Tutor ID”

DISPLAY “Enter Tutor Rating: ”

READ rating

FOR (int i = 0; ::allTutor[i].id != 0; i++)

IF(::allTutor[i].rating == rating)

Display “Tutor ID : ” SET ::allTutor[i].id

Display “Name : ” SET ::allTutor[i].name

Display “Date Joined : ” SET ::allTutor[i].dateJoined

Display “Date Terminated : ” SET ::allTutor[i].dateTerminated

Display “Hourly Rate : ” SET ::allTutor[i].hourlyRate

Display “Phone : ” SET ::allTutor[i].phone

Display “Address : ” SET ::allTutor[i].address

Display “Tuition Centre Code : ” SET ::allTutor[i].tcCode

Display “Tuition Centre Name : ” SET ::allTutor[i].tcName

Display “Subject Code : ” SET ::allTutor[i].subjectCode

Display “Subject Name : ” SET ::allTutor[i].subjectName

Display “Rating : ” SET ::allTutor[i].rating

SET found TO true

END IF

END FOR LOOP

IF(found == false)

DISPLAY “Unable to find Tutor ID, press 1 to try again, or any to go back: ”

READ input

IF(input = = 1)

CALL FUNCTION searchTutorRating()

ELSE

RETURN

END IF

ELSE IF (found = = true)

DISPLAY “Tutor Found, Returning to Main Menu”

CALL FUNCTION mainMenu()

END IF

END

### 2.1.5 Sort and display by Tutors ID in ascending order

*Using Linked List*

START

DECLARE TUTOR POINTER VARIABLE current, SET to head

DECLARE TUTOR POINTER VARIABLE index, SET to NULL

DISPLAY "Sort and Display by Tutors ID in ascending Order"

IF(head == NULL)

RETURN

ELSE

WHILE(current != NULL):

SET index TO current->next

WHILE(index != NULL):

IF(current->id > index->id):

SET temp TO current->id

SET current TO index->id

SET index->id TO temp

END IF

SET index TO index->next

END WHIILE

SET current TO current->next

END WHILE

END IF

CALL displayTutor()

DISPLAY "Returning to Main Menu"

RETURN

END

*Using Array List*

START

DECLARE STRUCTURE TUTOR VARIABLE temp

DISPLAY "Sort and display by Tutors ID in ascending order"

DISPLAY "Sorting Tutors by Tutor ID[Ascending]"

FOR(INT i = 0; i < ::insertingIndex; i++)

IF(::allTutor[i].id > ::allTutor[i+1].id):

SET temp TO ::allTutor[i]

SET ::allTutor[i] TO ::allTutor[i+1]

SET ::allTutor[i] TO temp

END IF

END FOR

CALL FUNCTION displayTutor()

DISPLAY "Returning to Main Menu"

RETURN

END

### 2.1.6 Sort and display by Tutors Hourly Pay Rate in ascending order

*Using Linked List*

START

DECLARE TUTOR POINTER VARIABLE current, SET to head

DECLARE TUTOR POINTER VARIABLE index, SET to NULL

DISPLAY "Sort and Display by Tutors ID in ascending Order"

IF(head == NULL)

RETURN

ELSE

WHILE(current != NULL)

SET index TO current->next

WHILE(index != NULL)

IF(current->hourlyRate > index->hourlyRate):

SET temp TO current->hourlyRate

SET current TO index->hourlyRate

SET index->hourlyRate TO temp

END IF

SET index TO index->next

END WHIILE

SET current TO current->next

END WHILE

END IF

CALL displayTutor()

DISPLAY "Returning to Main Menu"

RETURN

END

*Using Array List*

START

DECLARE STRUCTURE TUTOR VARIABLE temp

DISPLAY "Sort and display by Tutors Hourly Pay Rate in ascending order"

DISPLAY "Sorting Tutors by Tutor Hourly Pay Rate[Ascending]"

FOR(INT i = 0; i < ::insertingIndex; i++)

IF(::allTutor[i].hourlyRate > ::allTutor[i+1].hourlyRate)

SET temp TO ::allTutor[i]

SET ::allTutor[i] TO ::allTutor[i+1]

SET ::allTutor[i] TO temp

END IF

END FOR

CALL FUNCTION displayTutor()

DISPLAY "Returning to Main Menu"

RETURN

END

### 2.1.7 Sort and display by Tutors Overall Performance in ascending order

*Using Linked List*

START

DECLARE TUTOR POINTER VARIABLE current, SET to head

DECLARE TUTOR POINTER VARIABLE index, SET to NULL

DISPLAY "Sort and Display by Tutors ID in ascending Order"

IF(head == NULL)

RETURN

ELSE :

WHILE(current != NULL)

SET index TO current->next

WHILE(index != NULL):

IF(current->rating > index->rating):

SET temp TO current->rating

SET current TO index->rating

SET index->rating TO temp

END IF

SET index TO index->next

END WHIILE

SET current TO current->next

END WHILE

END IF

CALL displayTutor()

DISPLAY "Returning to Main Menu"

RETURN

END

*Using Array List*

START

DECLARE STRUCTURE TUTOR VARIABLE temp

DISPLAY "Sort and display by Tutors Rating in ascending order"

DISPLAY "Sorting Tutors by Tutor Rating[Ascending]"

FOR(INT i = 0; i < ::insertingIndex; i++)

IF(::allTutor[i].hourlyRate > ::allTutor[i+1].hourlyRate)

SET temp TO ::allTutor[i]

SET ::allTutor[i] TO ::allTutor[i+1]

SET ::allTutor[i] TO temp

END IF

END FOR LOOP

CALL FUNCTION displayTutor()

DISPLAY "Returning to Main Menu"

RETURN

END

### 2.1.8 Modify a Tutor Record

*Using Linked List*

FUNCTION modifyTutor()

START

DECLARE TUTOR POINTER VARIABLE current TO head

DECLARE Integer inpID

DECLARE Boolean found SET TO false

DECLARE Integer input

DECLARE Integer id

DECLARE String name

DECLARE String dateJoined

DECLARE String dateTerminated

DECLARE Double hourlyRate

DECLARE String phone

DECLARE String address

DECLARE String tcCode

DECLARE String tcName

DECLARE String subjectCode

DECLARE String subjectName

DECLARE Double rating

DISPLAY “Modify a Tutor Record”

DISPLAY “========================”

DISPLAY “Enter Tutor ID : ”

READ inpID

WHILE (current != NULL)

IF (current->id == inpID)

DISPLAY “Current Tutor Information”

DISPLAY “--------------------------”

DISPLAY “Tutor ID: ” SET current->id

DISPLAY “Name: ” SET current->name

DISPLAY “Date Joined: ” SET current->dateJoined

DISPLAY “Date Terminated: ” SET current->dateTerminated

DISPLAY “Hourly Rate: ” SET current->hourlyRate

DISPLAY “Phone: ” SET current->phone

DISPLAY “Address: ” SET current->address

DISPLAY “Tuition Code Centre: ” SET current->tcCode

DISPLAY “Tuition Centre Name: ” SET current->tcName

DISPLAY “Subject Code: ” SET current->subjectCode

DISPLAY “Subject Name: ” SET current->subjectName

DISPLAY “Rating: ” SET current->rating

SET found TO true

BREAK

ELSE IF (current->id != inpID)

SET current TO current->next

END IF

END WHILE

IF (found)

DISPLAY “New Tutor Information”

DISPLAY “----------------------”

DISPLAY “Tutor ID: ”

READ current->id

DISPLAY “Name: ”

READ current->name

DISPLAY “Date Joined: ”

READ current->dateJoined

DISPLAY “Date Terminated: ”

READ current->dateTerminated

DISPLAY “Hourly Rate: ”

READ current->hourlyRate

DISPLAY “Phone: ”

READ current->phone

DISPLAY “Address: ”

READ current->address

DISPLAY “Tuition Centre Code: ”

READ current->tcCode

DISPLAY “Tuition Centre Name: ”

READ current-> tcName

DISPLAY “Subject Code: ”

READ current->subjectCode

DISPLAY “Subject Name: ”

READ current->subjectName

DISPLAY “Rating: ”

READ current->rating

DISLAY “Successfully updated”

ELSE

DISPLAY “Unable to find tutor ID, press 1 to try again, or any to go back : ”

READ input

IF (input == 1)

CALL FUNCTION modifyTutor()

ELSE

DISPLAY “Returning to Main Menu”

RETURN

END IF

END IF

DISPLAY “Returning to Main Menu”

RETURN

END

*Using Array List*

FUNCTION modifyTutor()

START

DECLARE Integer inpID, input i

DECLARE Boolean found SET TO false

DISPLAY “Modify a Tutor Record”

DISPLAY “========================”

DISPLAY “Enter Tutor ID : ”

READ inpID

FOR (i = 0; i < ::insertingIndex; i++)

IF (::allTutor[i].id == inpID)

DISPLAY “Current Tutor Information”

DISPLAY “--------------------------”

DISPLAY “Tutor ID: ” SET allTutor[i].id

DISPLAY “Name: ” SET allTutor[i].name

DISPLAY “Date Joined: ” SET allTutor[i].dateJoined

DISPLAY “Date Terminated: ” SET allTutor[i].dateTerminated

DISPLAY “Hourly Rate: ” SET allTutor[i].hourlyRate

DISPLAY “Phone: ” SET allTutor[i].phone

DISPLAY “Address: ” SET allTutor[i].address

DISPLAY “Tuition Code Centre: ” SET allTutor[i].tcCode

DISPLAY “Tuition Centre Name: ” SET allTutor[i].tcName

DISPLAY “Subject Code: ” SET allTutor[i].subjectCode

DISPLAY “Subject Name: ” SET allTutor[i].subjectName

DISPLAY “Rating: ” SET allTutor[i].rating

SET found TO true

BREAK

END IF

END FOR

IF (found)

DISPLAY “New Tutor Information”

DISPLAY “----------------------”

DISPLAY “Tutor ID: ”

READ allTutor[i].id

DISPLAY “Name: ”

READ allTutor[i].name

DISPLAY “Date Joined: ”

READ allTutor[i].dateJoined

DISPLAY “Date Terminated: ”

READ allTutor[i].dateTerminated

DISPLAY “Hourly Rate: ”

READ allTutor[i].hourlyRate

DISPLAY “Phone: ”

READ allTutor[i].phone

DISPLAY “Address: ”

READ allTutor[i].address

DISPLAY “Tuition Centre Code: ”

READ allTutor[i].tcCode

DISPLAY “Tuiton Center Name: ”

READ allTutor[i].tcName

DISPLAY “Subject Code: ”

READ allTutor[i].subjectCode

DISPLAY “Subject Name: ”

READ allTutor[i].subjectName

DISPLAY “Rating: ”

READ allTutor[i].rating

DISPLAY “Successfully Updated”

ELSE

DISPLAY “Unable to find tutor ID, press 1 to try again, or any to go back : ”

READ input

IF (input == 1)

CALL FUNCTION modifyTutor()

ELSE

DISPLAY “Returning to Main Menu”

RETURN

END IF

END IF

DISPLAY “Returning to Main Menu”

RETURN

END

### 2.1.9 Delete a Tutor Record

*Using Linked List*

START

DECLARE TUTOR POINTER VARIABLE current, SET to head

DECLARE TUTOR POINTER VARIABLE previous, SET to NULL

DECLARE Boolean found SET to false

DECLARE Integer inpID

DECLARE Integer Input

DECLARE Integer id

DECLARE String name

DECLARE String dateJoined

DECLARE String dateTerminated

DECLARE Double hourlyRate

DECLARE String phone

DECLARE String address

DECLARE String tcCode

DECLARE String tcName

DECLARE String subjectCode

DECLARE String subjectName

DECLARE Double rating

DISPLAY “Modify a Tutor Record”

DISPLAY “Enter Tutor ID: ”

WHILE (current != NULL)

IF(current->id = = inpID)

DISPLAY “Tutor Information”

Display “Tutor ID : ” SET current->id

Display “Name : ” SET current->name

Display “Date Joined : ” SET current->dateJoined

Display “Date Terminated : ” SET current->dateTerminated

Display “Hourly Rate : ” SET current->hourlyRate

Display “Phone : ” SET current->phone

Display “Address : ” SET current->address

Display “Tuition Centre Code : ” SET current->tcCode

Display “Tuition Centre Name : ” SET current->tcName

Display “Subject Code : ” SET current->subjectCode

Display “Subject Name : ” SET current->subjectName

Display “Rating : ” SET current->rating

SET found TO true

DISPLAY “Confirm deletion (1 for Yes, any for No): ”

READ input

IF(input = = 1)

IF (previous = = NULL):

SET head TO current->next

DELETE current

DISPLAY “Returning to Main Menu”

RETURN

END IF

SET previous->next TO current->next

DELETE current

DISPLAY “Returning to Main Menu”

RETURN

ELSE

DISPLAY “Deletion Cancelled, Returning to Main Menu”

END IF

BREAK

ELSE IF(current->id != inpID)

SET previous TO current

SET current TO current->next

END IF

END WHILE

DISPLAY “Returning to Main Menu”

RETURN

END

*Using Array List*

START

DECLARE Integer inpID

DECLARE Integer input

DECLARE Integer i

DECLARE Boolean found SET TO false

DECLARE STRUCTURE TUTOR VARIABLE temp

DISPLAY “Delete a Tutor Record”

DISPLAY “Enter Tutor ID: ”

READ inpID

FOR (i = 0; i < ::insertingIndex; i++)

IF (::allTutor[i].id == inpID)

Display “Tutor ID : ” SET ::allTutor[i].id

Display “Name : ” SET ::allTutor[i].name

Display “Date Joined : ” SET ::allTutor[i].dateJoined

Display “Date Terminated : ” SET ::allTutor[i].dateTerminated

Display “Hourly Rate : ” SET ::allTutor[i].hourlyRate

Display “Phone : ” SET ::allTutor[i].phone

Display “Address : ” SET ::allTutor[i].address

Display “Tuition Centre Code : ” SET ::allTutor[i].tcCode

Display “Tuition Centre Name : ” SET ::allTutor[i].tcName

Display “Subject Code : ” SET ::allTutor[i].subjectCode

Display “Subject Name : ” SET ::allTutor[i].subjectName

Display “Rating : ” SET ::allTutor[i].rating

SET found TO true

DISPLAY “Confirm deletion (1 for Yes, any for No): ”

READ input

IF(input = = 1)

WHILE (::allTutor[i + 1].id != 0)

SET ::allTutor[i] TO ::allTutor[i + 1]

INCREMENT i by 1

END WHILE

DISPLAY “Returning to Main Menu”

RETURN

ELSE

DISPLAY “Deletion Cancelled, Returning to Main Menu”

END IF

BREAK

END IF

END FOR LOOP

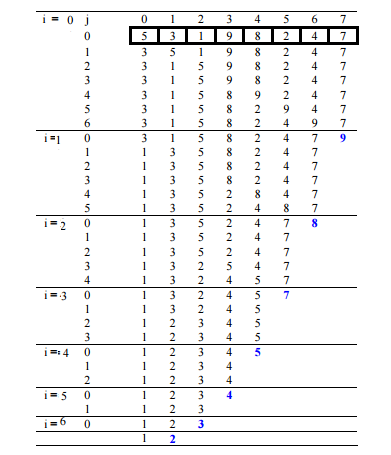
DISPLAY “Returning to Main Menu”

RETURN

END

## 2.2 Explanation of Algorithms

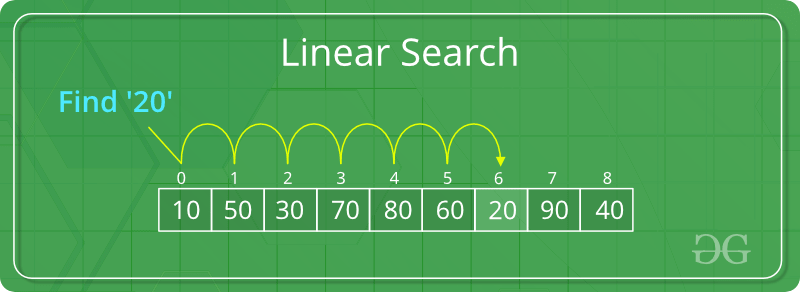
One of the functionalities that the Tutor Management System provides is the option to sort and display the tutor records based on specific criteria. In order to do this, some sorting algorithms need to be applied to allow the records to behave in the correct order before displaying to the user. In this assignment, the chosen sorting algorithm is bubble sort. Commonly known as the simplest sorting algorithm, it works by swapping adjacent elements repeatedly should the order is still incorrect (GeeksforGeeks, 2014).



*Figure 3 Bubble Sort Example*

The sorting of the elements in the linked list or array list is done in iteration, so by rearranging the elements ascendingly, the largest element will be placed at the very end position.

As for the search algorithm, linear search has been selected to be implemented for the systems. Linear search is a very simple sequential search algorithm. The process involves going over each and every element one by one. Starting from the leftmost element, in concept, the item to be searched, variable x for example, will be compared with. If a match is found, the index of the element is returned. Otherwise, the linear search will continue till the end of the data collection and return -1 (GeeksforGeeks, 2021).



*Figure 4 Linear Searching Example*

## 2.3 Justification on Algorithms

Bubble sort was used in this assignment since it’s the most known algorithm by all members, as well as it is easy to understand and implement in the code. One weakness of the bubble sort is that it’s inefficient for huge samples of data since it runs with O(n2), meaning, the time it takes to finish sorting will exponentially grow according to the size of the array, which luckily, the proposed system for now will only be able to store 100 structures of tutor in the structure of array.

For searching algorithm, the proposed system will utilize the linear search, since it is also considered the easiest algorithm to understand and to implement. One disadvantage of the linear search is that it’s iterates the dataset one by one, which, depending on the size of the dataset, can be very inefficient, but, vice versa, in the proposed system, the linear searching was deliberately utilized since it performs relatively fast for small datasets, which in this case, has the array size of 100, this algorithm can also be initiated without needing to sort the dataset first, which most O(log n) algorithms requires, it is also not affected by insertions and deletions since it doesn’t need to be sorted first.

# 3.0 Task Distribution

|  |  |  |
| --- | --- | --- |
| Group Member | Task | Signature |
| Marcell Agung Wahyudi (TP058650) | 1.2 Data Structures and Classes  2.0 System Design and Workflows  2.1 Pseudocode:   * Sort and display by Tutors ID in ascending order * Sort and display by Tutors Hourly Pay Rate in ascending order * Sort and display by Tutors Overall Performance in ascending order |  |
| Ros Amisya Fatiha binti Mohd Rizal (TP062386) | 1.1 System Introduction  2.0 System Design and Workflows  2.1 Pseudocode:   * Display All Records * Search Tutors by overall performance (Rating) * Delete a Tutor Record |  |
| Marsha binti Khairil Mubin (TP062399) | 1.3 Justification  2.0 System Design and Workflows  2.1 Pseudocode:   * Add a Tutor Record * Search a Tutor by Tutor ID * Modify a Tutor Record | Text, letter  Description automatically generated |

# 4. 0 References

Bitesize. (n.d.). *Searching algorithms*. Retrieved from Bitesize: https://www.bbc.co.uk/bitesize/guides/zts8v9q/revision/4

Chan, M. (2020, March 7). *Array vs Linked List Data Structures*. Retrieved from Level Up Coding: https://levelup.gitconnected.com/array-vs-linked-list-data-structure-c5c0ff405f16

GeeksforGeeks. (2014, February 2). *Bubble Sort* . Retrieved from GeeksforGeeks: https://www.geeksforgeeks.org/bubble-sort/

GeeksforGeeks. (2020, October 29). *Advantages and Disadvantages of Array in C*. Retrieved from GeeksforGeeks: https://www.geeksforgeeks.org/advantages-and-disadvantages-of-array-in-c/?ref=rp

GeeksforGeeks. (2021, September 15). *Linear Search*. Retrieved from GeeksforGeeks: https://www.geeksforgeeks.org/linear-search/

GeeksforGeeks. (2022, March 28). *Advantages and Disadvantages of Linked List*. Retrieved from GeeksforGeeks: https://www.geeksforgeeks.org/advantages-and-disadvantages-of-linked-list/?ref=rp

GeeksforGeeks. (2022, April 13). *Bubble Sort*. Retrieved from GeeksforGeeks: https://www.geeksforgeeks.org/bubble-sort/

GeeksforGeeks. (2022, March 24). *Difference between Linear and Non-linear Data Structures*. Retrieved from GeeksforGeeks: https://www.geeksforgeeks.org/difference-between-linear-and-non-linear-data-structures/#:~:text=Data%20structure%20where%20data%20elements,elements%20in%20single%20run%20only.

Krohn, H. (2019, July 2). *Linked Lists vs. Arrays*. Retrieved from Medium: https://towardsdatascience.com/linked-lists-vs-arrays-78746f983267

Vaghani, R. (2022, April 12). *Difference between Singly linked list and Doubly linked list*. Retrieved from GeeksforGeeks: https://www.geeksforgeeks.org/difference-between-singly-linked-list-and-doubly-linked-list/?ref=rp

Wandy, J. (2017, June 27). *The Advantages & Disadvantages of Sorting Algorithms*. Retrieved from sciencing: https://sciencing.com/the-advantages-disadvantages-of-sorting-algorithms-12749529.html