ASSIGNMENT

TECHNOLOGY PARK MALAYSIA

CT077-3-2 DSTR

DATA STRUCTURES

HAND OUT DATE: 9 – MARCH - 2020

HAND IN DATE: 18 – MAY - 2020

WEIGHTAGE: 50%

INSTRUCTIONS TO CANDIDATES:

1. **Submit your assignment at the administrative counter**
2. **Students are advised to underpin their answers with the use of references (cited using the Harvard Name 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 bound in an appropriate style (comb bound or stapled).**
6. **Where the assignment should be submitted in softcopy for both written assignment (documentation) and source code (where appropriate) through Moodle.**
7. **You must obtain 50% overall to pass this module.**

Version 1.2 UCTI AssCv 2007-10-23

Group Member : **Chan Jia Le** **(TP049952)**

Lee Jin Heng **(TP053710)**

Chen Chee Kin **(TP053224)**

Intake Code : **UC2F1908CS(DA)**

Module Code : **CT077-3-2-DSTR**

Assignment Title : **Data Structure**

Hand-out Date : **9th March 2020**

Hand-in Date : **18th May 2020**

Lecturer : **Mr Au Yit Wah**

Table of Contents

1.0 | Introduction

The group is consisted of three members, each one of them are from different modules. The teammates are Chen Chee Kin (TP053224) from Intelligence System, Chan Jia Le (TP049952) from CS(DA) which is a specialism course in Data Analytic, Lee Jin Heng (TP053710) which is taking Computer Science. The aim of this report is to explain about a tuition centre management system to provide an overall view of the system to have a smooth construction soon. The tutor management system will be able to add, display records of Tutors, Search, Sort & Display Tutor details by Tutor_ID, Hourly Pay Rate or even Overall Performance. Thus, modifying and deleting tutor records. All in all, the system will be constructed for the purpose for eXcel Tuition Centre for a more efficient and effective way of retrieving and managing tutor record.

2.0 | Data Structure and Classes

It is decided that the system will be constructed with 4 data structures as linked list or array. The data structures include Tuition Centre, Subject, Date and Tutor. Each Tutor will be included with the 3 other data structure. The reason for separating Tuition Centre, Subject and Date as individual data structure is because each Subject, Tuition Centre and date has their own additional information like name and unique code of the centre. Other than that, the purpose of this is to better sort, manage, categorize the tuition centre and subject codes, in order to meet the objective of this system which is effectively and efficiently achieve Tutor Details.

```
struct Tutor {
    int tutorID;
    string tutorFirstName;
    string tutorLastName;
    struct Date datejoined, dateTerminated;
    double hourlyPayRate;
    struct TuitionCenter center;
    struct Subject teachSub;
    string phoneNumber;
    string tutorAddress;
    int rating;
};
struct Tutor tutorDetails[100];
```

Figure 2.1 shows the data structure of Tutor, which includes vital details like tutor ID,

Figure 2.1 Data Structure of Tutor in Array of Structure

first and last name, data structure for date with member variable dateJoined and dateTerminated, hourly pay rate, phone number, address, 2 data structure of tuition center and subject and lastly rating or performance. All the details are defined as the data member of Tutor to satisfy the

requirement requested by the eXcel tuition center's HR department. The structure will then be initialized as an array (array of structure) in the size of 100 with a member variable name as tutorDetails. This initialization means this array of structure can contain 100 tutor and each of them contain the details in the structure.

```
struct Tutor {
    struct Tutor *previous;
    int tutorID;
    string tutorFirstName;
    string tutorLastName;
    struct Date datejoined, dateTerminated;
    double hourlyPayRate;
    struct TuitionCenter center;
    struct Subject teachSub;
    string phoneNumber;
    string tutorAddress;
    int rating;
    struct Tutor* next;
} *start, *newnode, *previous, *current, *last;
```

Figure 2.2 Data Structure of Tutor in Doubly Link

Figure 2.2 shows the data structure in a link list form. The team plans to implement doubly link list. The details of the structure are like the one in array of structure but with an additional previous and next node. There will also be 5 pointers which include start (which is the pointer of the start node), new node (pointer for creating a new node), previous and temp (pointer used for inserting and deleting in specific location in a sorted link list) and lastly last (pointer to the last node).

```
struct Date {
    int dd;
    int mm;
    int yyyy;
};
```

Figure 2.3 Data

Figure 2.3 shows data structure of Date, this data structure is included inside Data structure of Tutor as shown in Figure 1, it includes the day, month, and year. The extra data structure constructed and declared as two-member variable dateJoined and dateTerminated because the team plans to use the value to calculate the day that Tutor has joined as the Tutor details only can be deleted once the day tutor had left is over 60 days.

```
struct Subject {
    int subjectCode;
    string subjectName;
};
```

Figure 2.4 Data Structure of Subject

Figure 2.4 shows data structure of Subject, this data structure is included inside Data structure of Tutor as shown in Figure 1, it includes the subject code and subject name. The extra data structure constructed for Subject is as mentioned to better differentiate each unique subject with subject code and its name.

```
struct TuitionCenter {
    int tuitionCenterCode;
    string tutorCenterName;
    string tutorCenterLocation;
};
```

Figure 2.5 Data Structure of Tuition Center

Figure 2.5 shows the data structure of Tuition, as shown in Figure 1, this data structure is included in the data structure of Tutor, it includes details like tuition canter code, name, and its location. Thus, as for mentioned, it is constructed for easier differentiation of each unique tuition center for each tutor.

2.1 | Flow Chart Diagram

2.1.1 | Main Menu

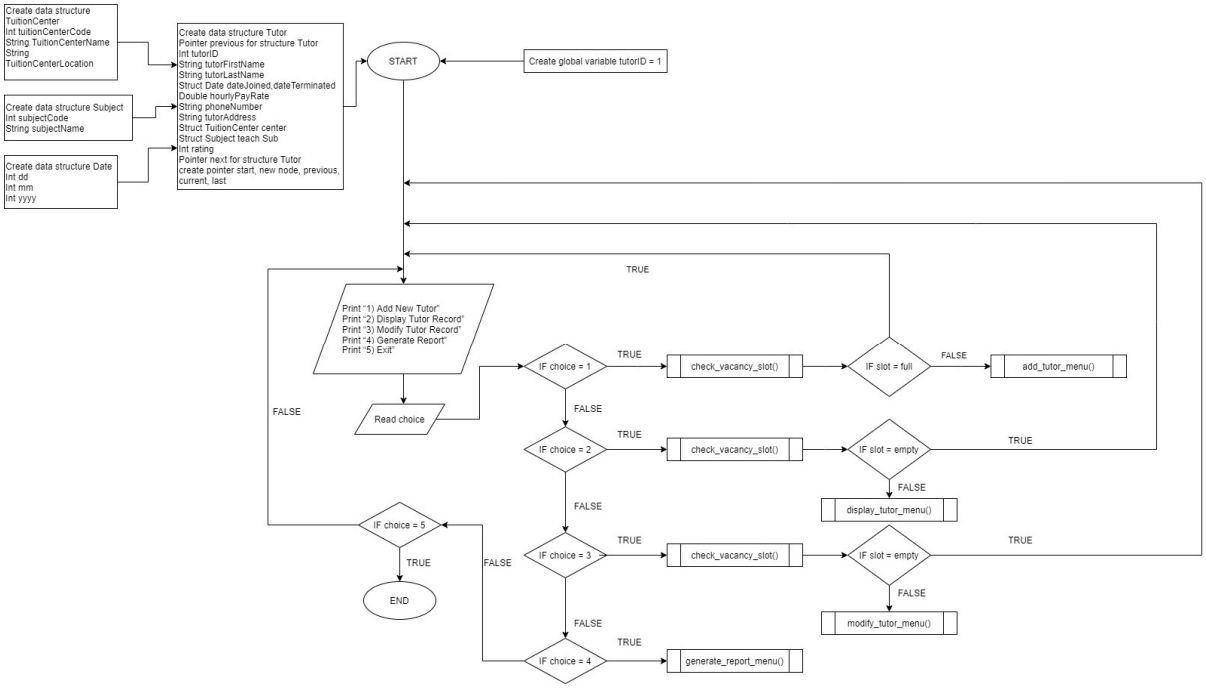


Figure 2.1.1.1 Main Menu for Link List

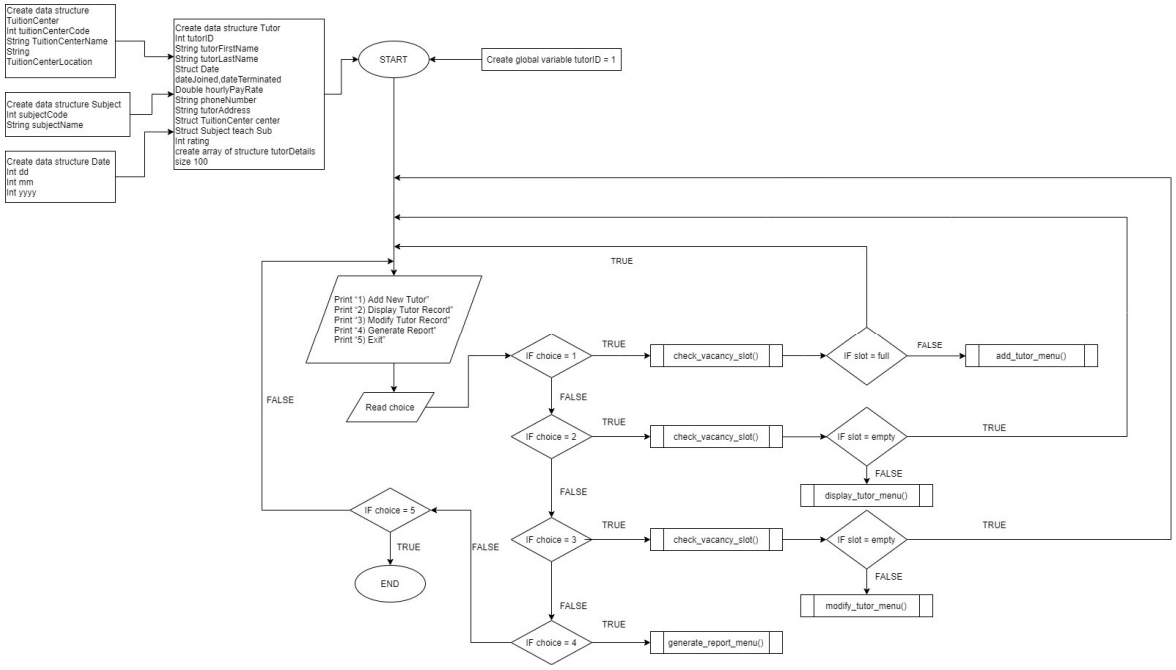


Figure 2.1.1.2 Main Menu for Array of Structure

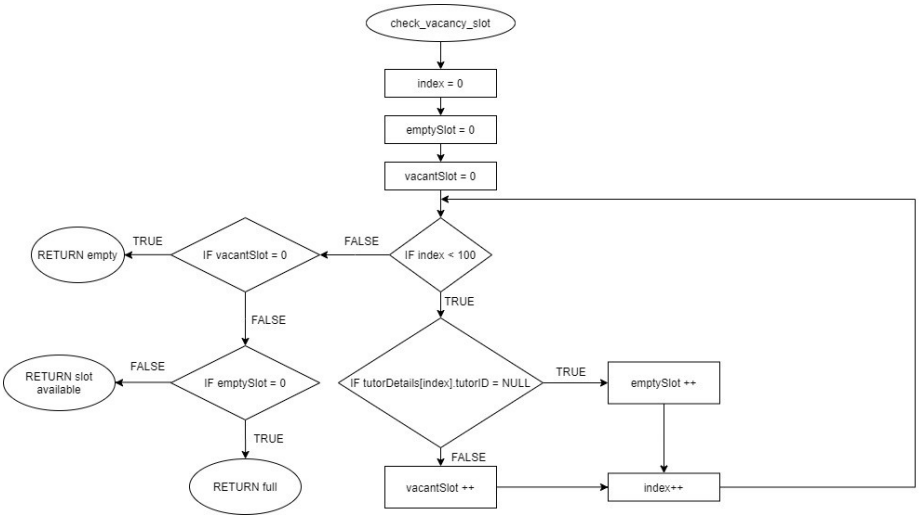


Figure 2.1.1.3 check vacancy slot at Main Menu for Array of Structure

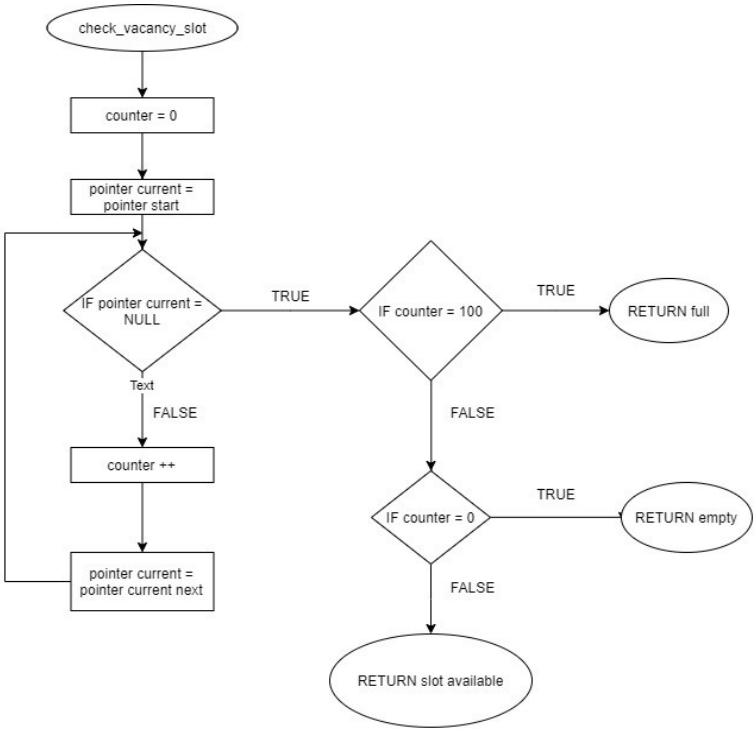


Figure 2.1.1.4 check vacancy slot at Main Menu for Linked List

Figure 2.1.1.1 and Figure 2.1.1.2 shows the main menu of the system for array of structure and linked list respectively. The program will first ask user input for each specific functionality which includes add tutor, display tutor records, modify tutor records (included with delete record), generate report or exit. Before entering add tutor menu, the program will contain a check vacancy slot function to ensure it is not full. Furthermore, if user wishes to enter display tutor record or modify tutor record, the check vacancy slot function will be implemented in order to ensure the tutor records is not empty. Figure 2.1.1.3 and Figure 2.1.1.4 shows the function check vacancy slot for both array of structure and linked list.

2.1.2 | Add Tutor Menu

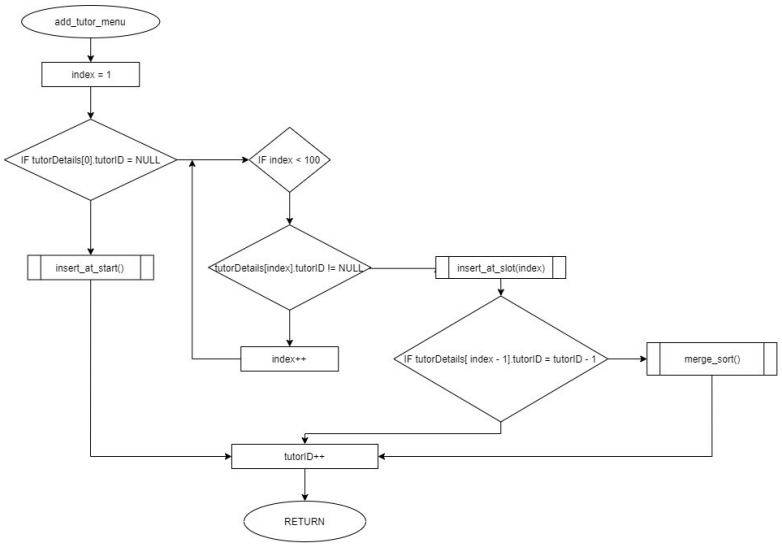


Figure 2.1.2.1 Add Tutor Menu for Array of Structure

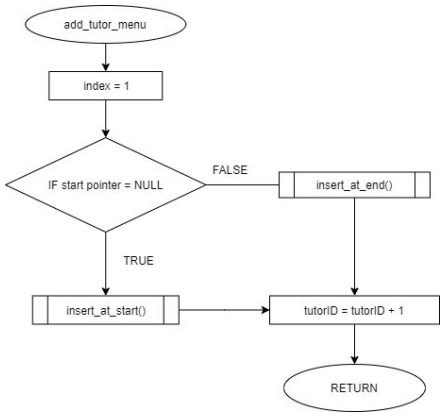


Figure 2.1.2.2 Add Tutor for Linked List

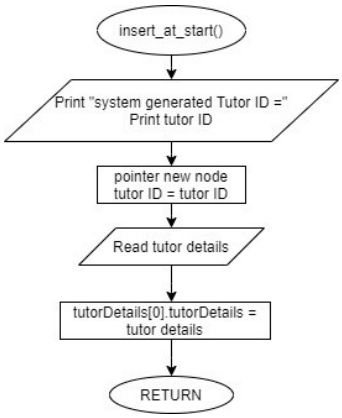


Figure 2.1.2.3 insert at start function for add tutor menu (array of structure)

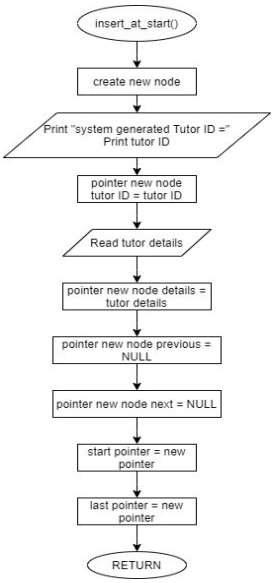


Figure 2.1.2.4 insert at start function for add tutor menu (linked list)

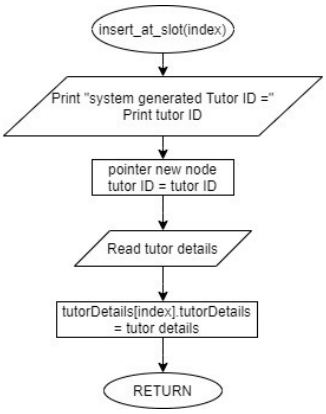


Figure 2.1.2.5 insert at slot function for add tutor menu (array of structure)

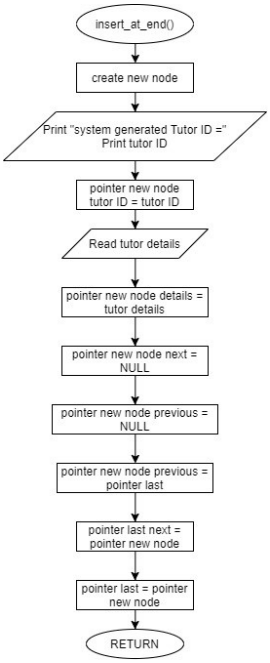


Figure 2.1.2.6 insert at end function for add tutor menu (linked list)

Add tutor menu is constructed as shown in Figure 2.1.2.1 and Figure 2.1.2.2. Add tutor menu allows user to add new tutor details to the system. The program will first identify if the first element of array or linked list is empty. If it is empty the program will insert the user input tutor details to the first element of array or linked list. If it is not, the program will insert by searching for empty slot in array of structure and into the last element if its linked list. The other figures show how the input will be processed and insert into the program.

2.1.3 | Display Tutor Menu

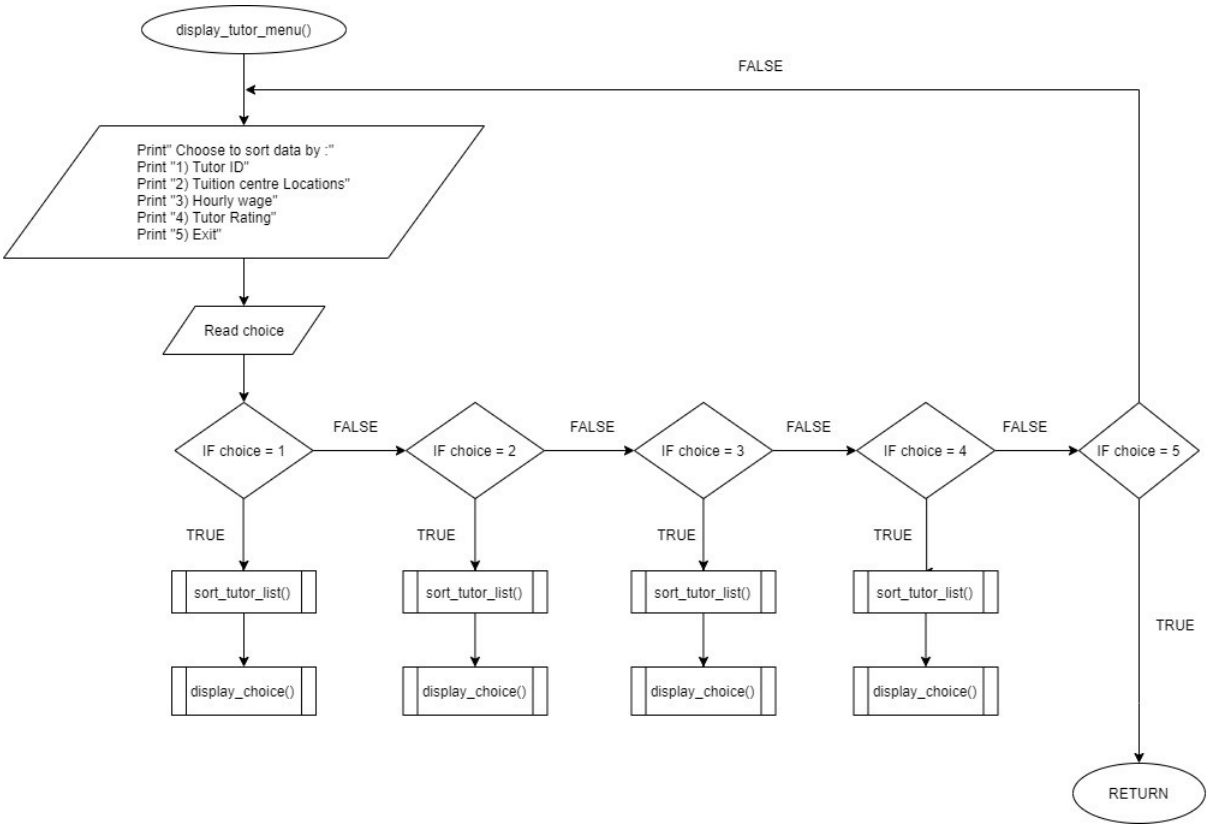


Figure 2.1.3.1 Display Tutor Menu

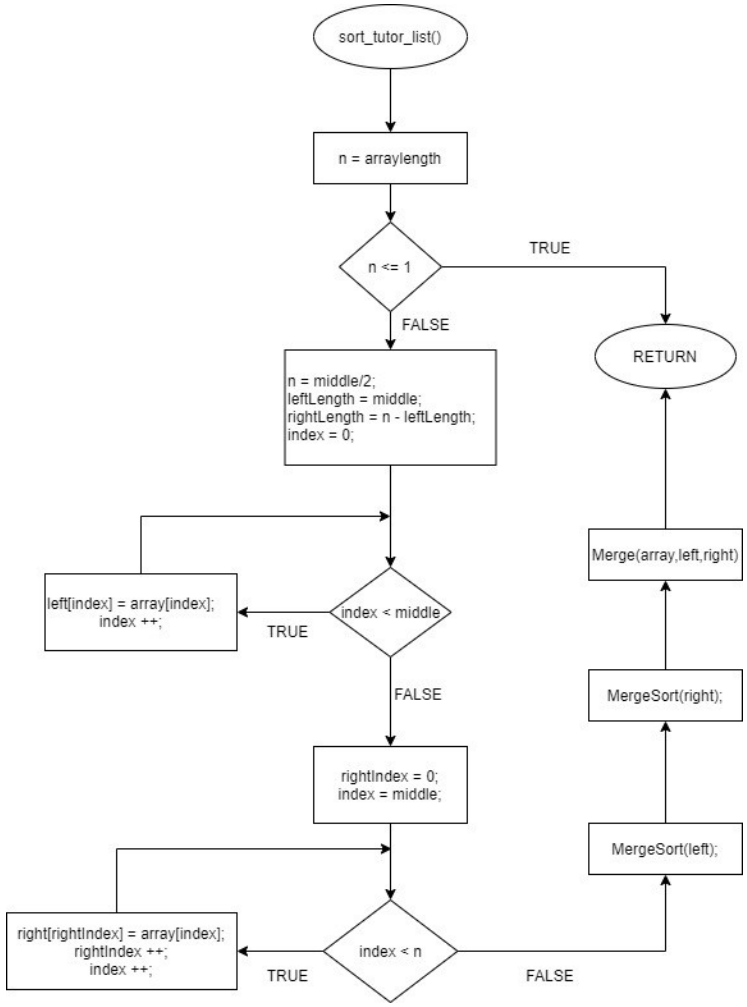


Figure 2.1.3.2 Sort Tutor List for Display Tutor Menu (Array of Structure)

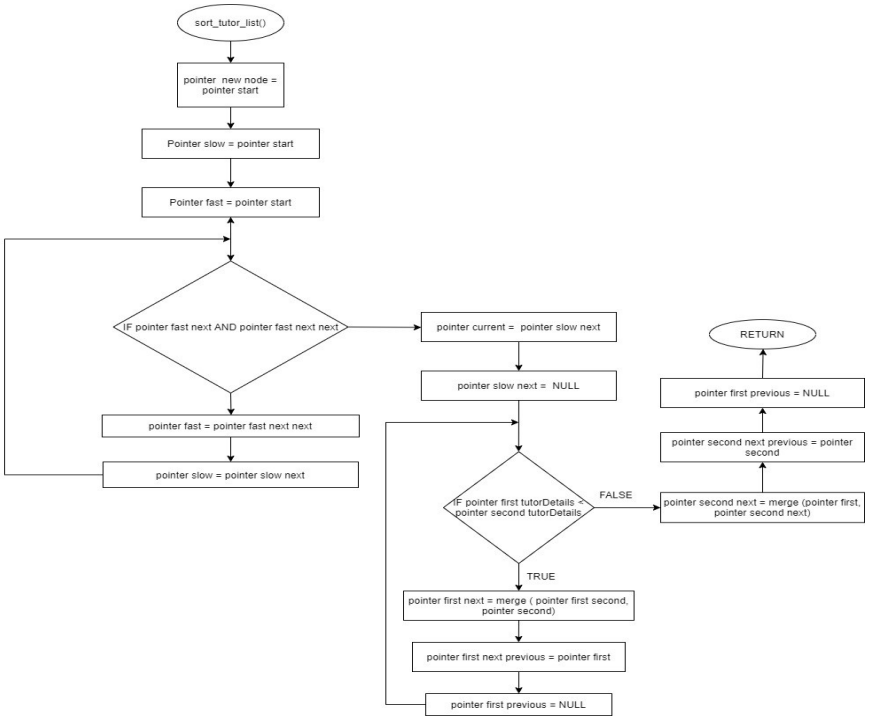


Figure 2.1.3.3 Sort Tutor List for Display Tutor Menu (Linked List)

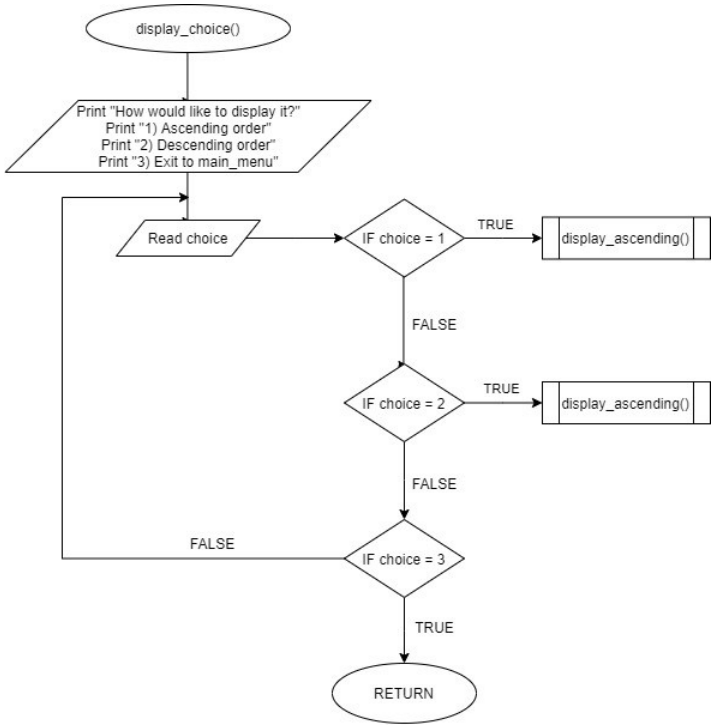


Figure 2.1.3.4 Display Choice for Display Tutor Menu (Linked List and Array of Structure)

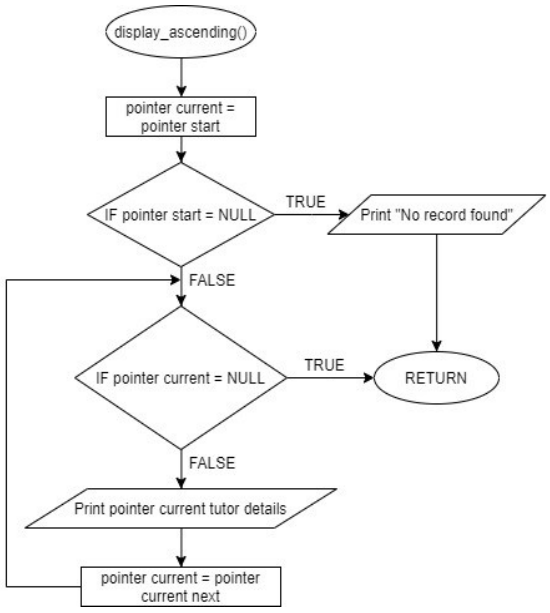


Figure 2.1.3.5 Display Ascending function for display choice (Linked List)

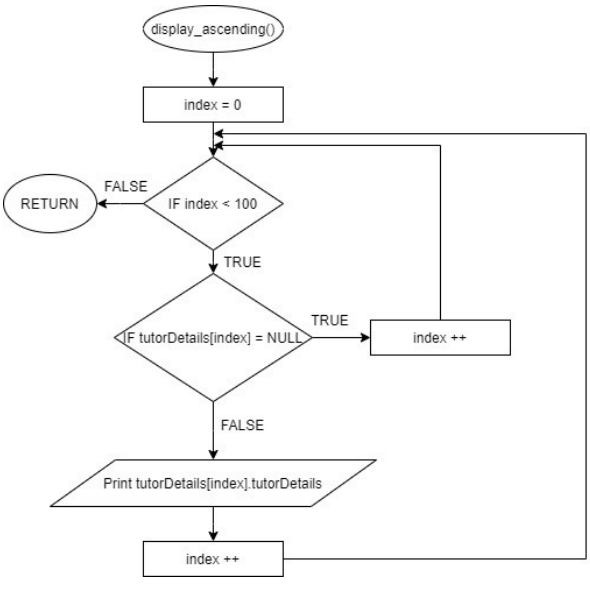


Figure 2.1.3.6 Display Ascending function for display choice (Array of Structure)

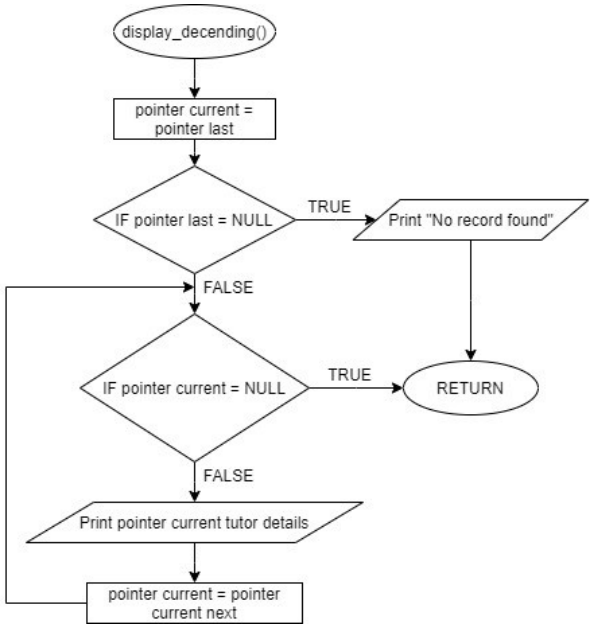


Figure 2.1.3.6 Display Descending function for display choice (Linked List)

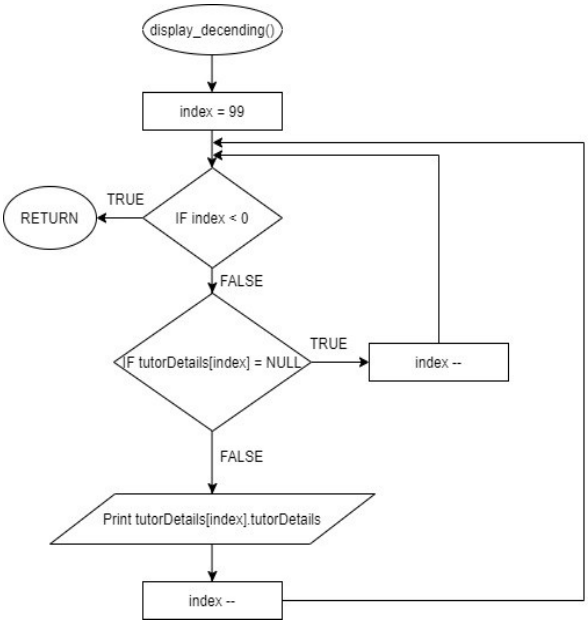


Figure 2.1.3.7 Display Descending function for display choice (Array of Structure)

Figure 2.1.3.1 is the menu for Display Tutor Details. The user will be asked to input the details that they wish to be sorted by either by tutor id, rating, hourly pay rate and so on. The system will then implement a merge sort function based on the system, array of structure or linked list. After the details are sorted based on user's desire. It will then execute the display choice function, shown in figure 2.1.3.4. The system will then request user to display either in ascending or descending order. Once user inputs the desired outcome, the system will then execute the function display either by ascending or descending as shown in figure 2.1.3.5 to figure 2.1.3.7. after finishing displaying all the tutor details. It will then return to the main menu.

2.1.4 | Modify Tutor Menu

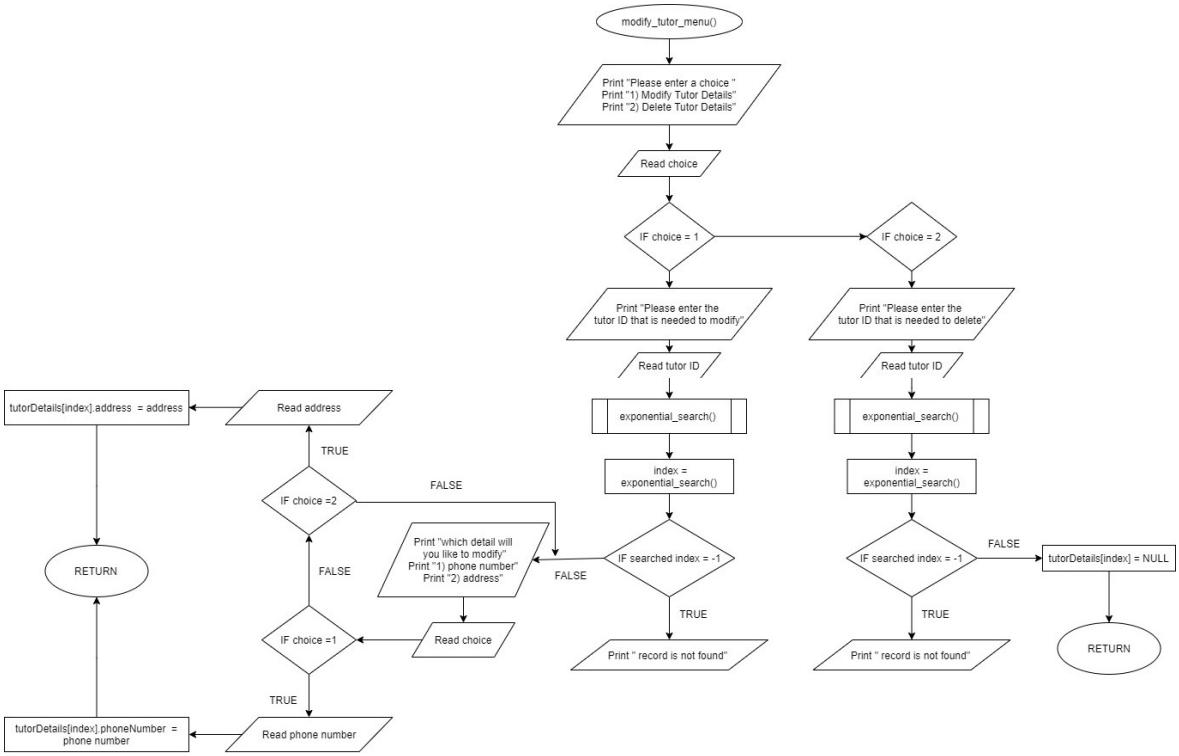


Figure 2.1.4.1 Modify Tutor Menu for Array of Structure

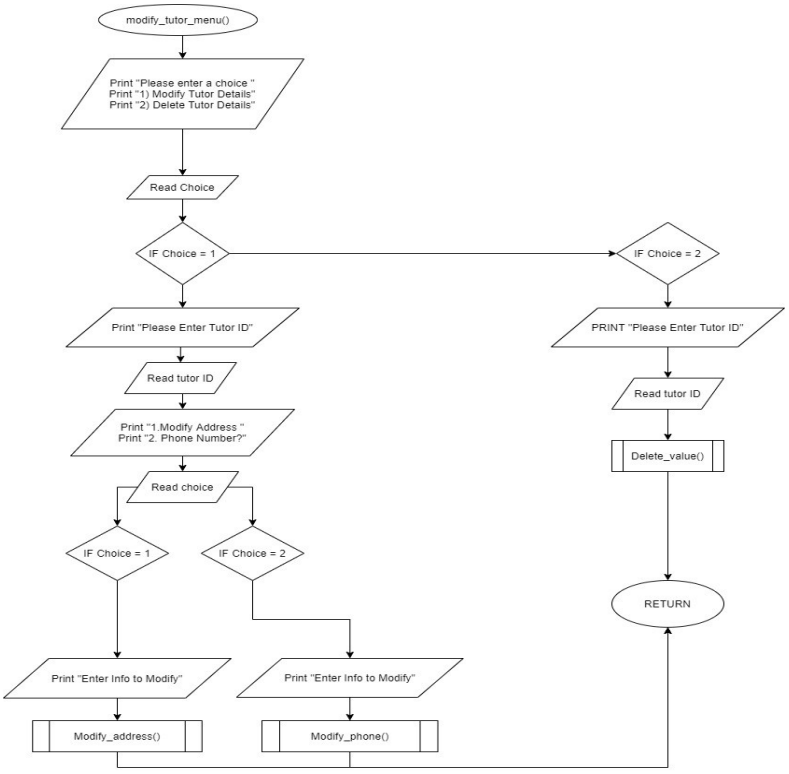


Figure 2.1.4.2 Modify Tutor Menu for Linked List

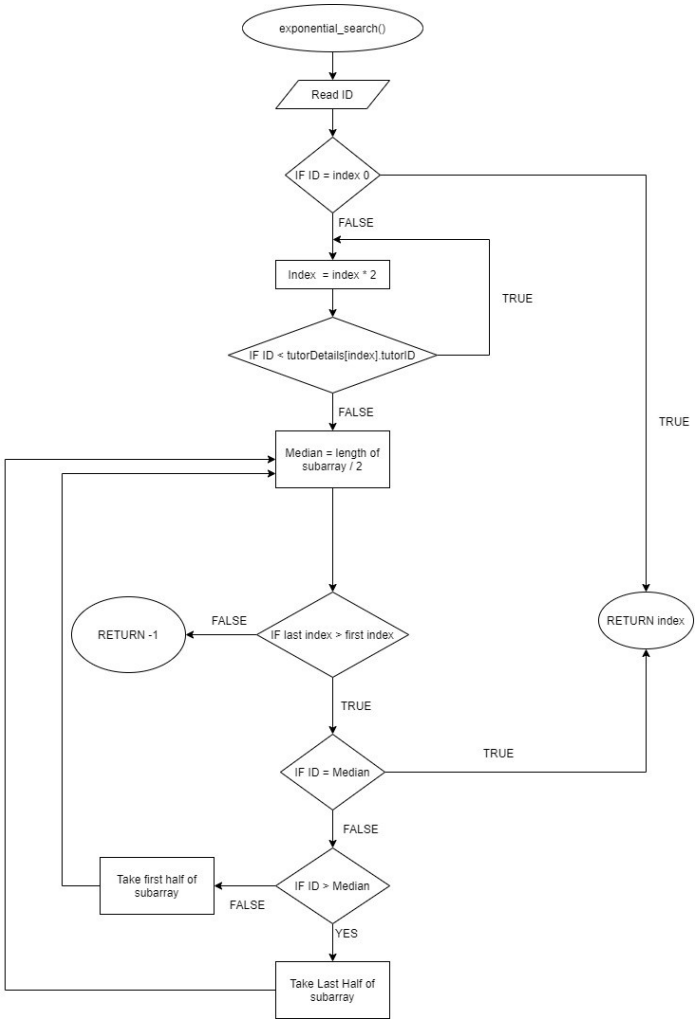


Figure 2.1.4.3 Exponential Search for Modify Tutor Menu (Array of Structure)

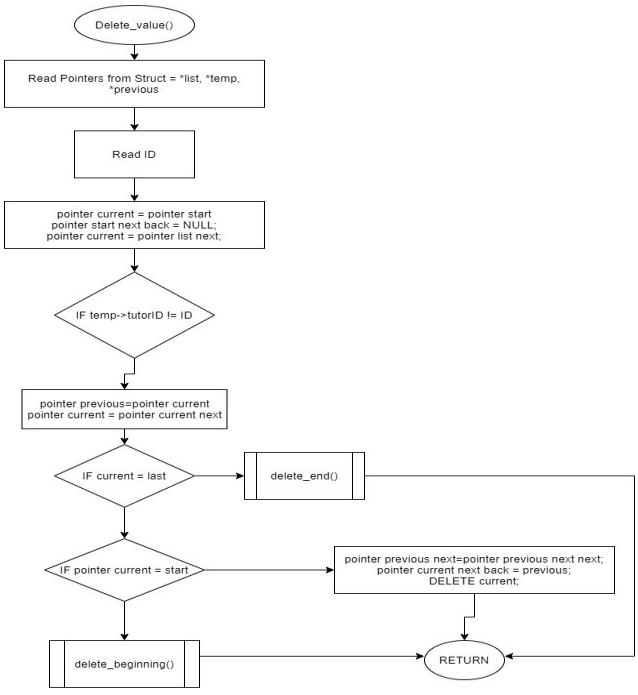


Figure 2.1.4.4 Delete value for Modify Tutor Menu (Linked List)

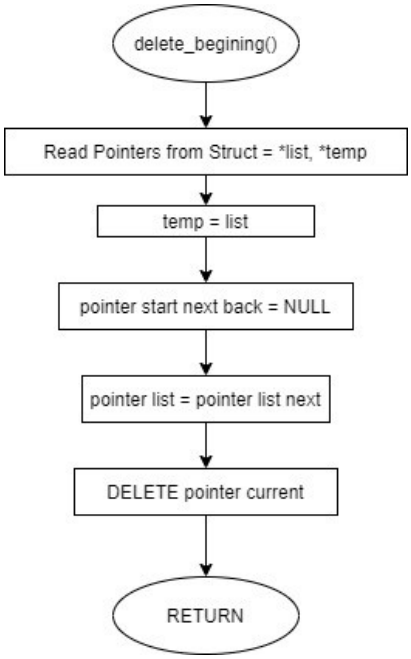


Figure 2.1.4.5 Delete beginning function for Delete Value (Linked List)

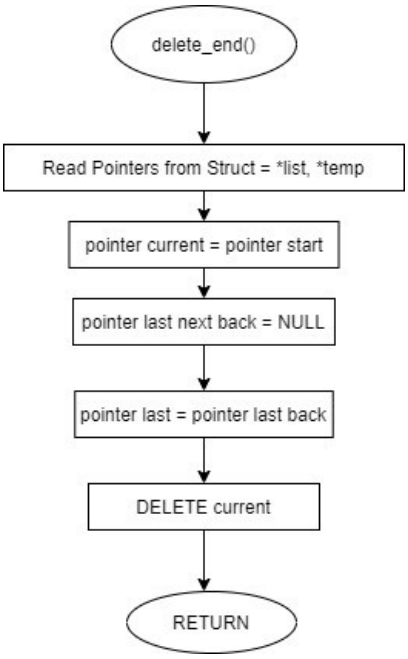


Figure 2.1.4.6 Delete end function for Delete Value (Linked List)

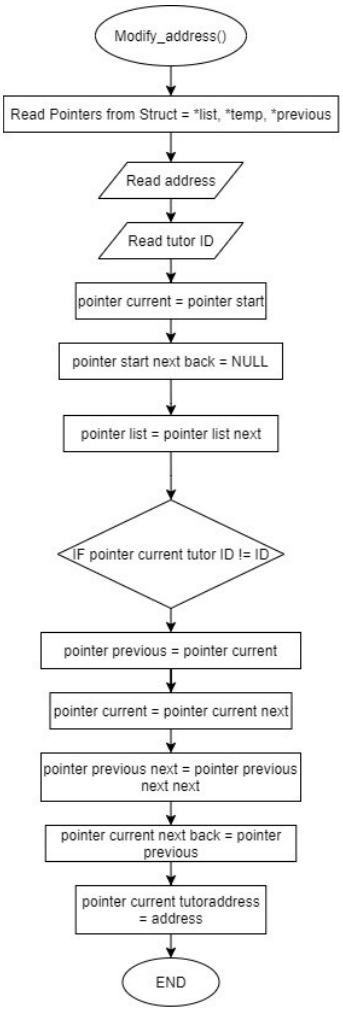


Figure 2.1.4.7 Modify address for Modify Tutor Menu (Linked List)

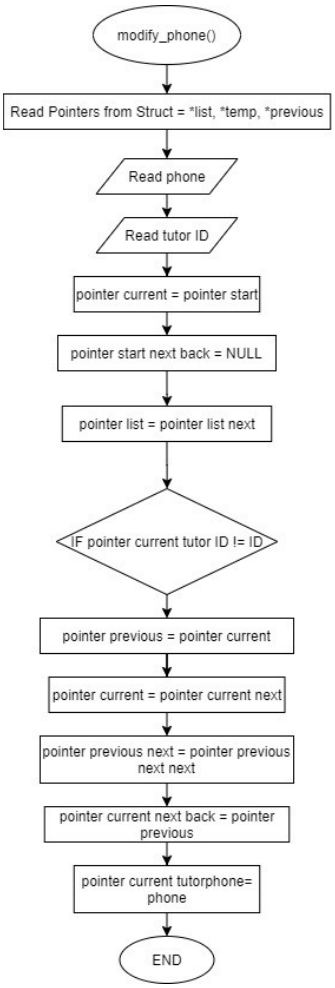


Figure 2.1.4.8 Modify address for Modify Tutor Menu (Linked List)

Figure 2.1.4.1 and Figure 2.1.4.2 are the flow chart for Modifying Tutor Menu. Modifying Tutor Menu is the menu for user to modify Tutor's phone or address or Deleting a tutor detail. For array of structure it will perform an exponential search to search for the desired Tutor by using Tutor ID and the program will then ask user to input the desired phone number or address or delete the tutor details. The program will then return to the main menu. For Linked list, the program will ask the same user input and the program will transverse through the linked list to identify the position of desired tutor details and modify it or delete accordingly. The other figures show the function that is embedded inside the modify tutor menu.

2.1.5 | Generate Report Menu

Figure 2.1.5.1 generate report menu for Array of Structure

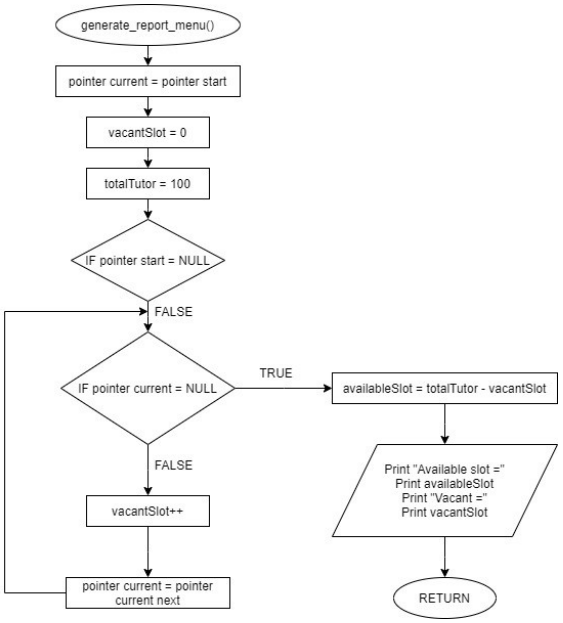


Figure 2.1.5.2 generate report menu for Linked List

Figure 2.1.5.1 and Figure 2.1.5.2 are both flow charts for generate report menu. One is constructed with the array of structure and one with Linked List. Generate report menu is constructed for the sake of easy report generation, it will go through the array of structure or linked list to calculate the available slot for vacancy and the occupied number and print it out for user's information.

3.0 | Algorithms Implemented in the System

3.1 | Array

3.1.1 | Add Tutor

```
int addTutorDetails() {
    int i = 1;
    /*check whether the first element in array is empty*/
    if (tutorDetails[0].tutorID == NULL) {
        tutorDetails[0].tutorID = tutorID;
        enterDetails(0); //call enter details with the first index passed to the enter details function
        tutorID = tutorID + 1;
        return 0;
    }
    else {
        /*find the index of the empty element*/
        for (i < 100; i; i++) {
            /*check whether the element in index is empty*/
            if (tutorDetails[i].tutorID != NULL) {
                continue;
            }
            else {
                tutorDetails[i].tutorID = tutorID;
                enterDetails(i);
                tutorID = tutorID + 1;
                break;
            }
        }
    }
    return 1;
}
```

Figure 3.1.1.1 Add Tutor Details Function (Array)

Add tutor details function is a function that allows the users to add tutor details such as tutor name into an array. The function will check the array list and validate if its empty. If so, the function will generate a tutor ID for the first index which is 1, if not the function will check and find the index of the empty element and input the newly registered tutor details there. Each ID generated is unique and will not be repeated.

3.1.2 | Display Ascending and Descending

```
void display_ascending(int lastIndex) {
    system("CLS");
    /*formating for the title for details of tutor*/
    cout << "\t| TUTOR DETAILS |" << endl << endl;
    cout << "\t" << setw(4); cout << "|TUTOR ID|"; cout << setw(13) << "|FIRST NAME|"; cout << setw(12) << "|LAST NAME|"; cout << setw(14) << "|DATE JOINED|";
    cout << setw(18) << "|DATE TERMINATED|"; cout << setw(6) << "|PAY|" ; cout << setw(14) << "|CENTRE CODE|"; cout << setw(18) << "|CENTRE LOCATION|"; cout << setw(14) << "|CENTRE NAME|";
    cout << setw(11) << "|SUB CODE|"; cout << setw(11) << "|SUB NAME|"; cout << setw(15) << "|PHONE NUM|"; cout << setw(21) << "|ADDRESS|"; cout << setw(23) << "|RATINGS|" << endl;
    int i = 0;
    /*while index is lesser or equals to last index display tutor details*/
    while (i <= lastIndex) {
        /*check if tutor ID in the index is not null*/
        if (tutorDetails[i].tutorID != NULL) {
            cout << "\t" << setw(6) << tutorDetails[i].tutorID << setw(14) << tutorDetails[i].tutorFirstName << setw(13) << tutorDetails[i].tutorLastName << setw(7);
            cout << tutorDetails[i].datejoined.dd << "/" << tutorDetails[i].datejoined.mm << "/" << tutorDetails[i].datejoined.yyyy << setw(10);
            cout << tutorDetails[i].dateTerminated.dd << "/" << tutorDetails[i].dateTerminated.mm << "/" << tutorDetails[i].dateTerminated.yyyy << setw(11);
            cout << tutorDetails[i].hourlyPayRate << setw(10) << tutorDetails[i].center.tuitionCenterCode << setw(21) << tutorDetails[i].center.tuitionCenterLocation;
            cout << setw(11) << tutorDetails[i].center.tuitionCenterName;
            cout << setw(12) << tutorDetails[i].teachSub.subjectCode << setw(13) << tutorDetails[i].teachSub.subjectName;
            cout << setw(17) << tutorDetails[i].phoneNumber << setw(30) << tutorDetails[i].tutorAddress;
            cou                                              etails[i].rating;
            cou
        }
        i++;
    }
    /*press enter to leave the current function*/
    cout << "   PRESS ENTER TO LEAVE ......" << endl;
    cout << "   "; cin.get();
    if (cin.get() == '\n') {
        cout << "   EXITING BACK TO MAIN MENU   " << endl;
        Sleep(2000);
        system("CLS");
    }
}
```

std::ostream std::cout
Search Online

Figure 3.1.2.1 Display Tutor Details Ascending and Descending

This function will print the index in the array in ascending or descending order. The display ascending function will initialize i = 0 and loop the print if the index is lesser or equal to the last index variable. And for the display descending function, i is initialized as the last index and the process will be looped if the variable is less than or equal to zero, which 0 is the first index

3.1.3 | Merge Sort Function

```
void merge(int l, int m, int r, int choice) {
    int i, j, k;
    int n1 = m - l + 1;
    int n2 = r - m;

    struct Tutor L[50], R[50];
    for (i = 0; i < n1; i++) {
        L[i] = tutorDetails[l + i];
    }
    for (j = 0; j < n2; j++) {
        R[j] = tutorDetails[m + 1 + j];
    }

    /* Merge the temp arrays back into arr[l..r]*/
    i = 0; // Initial index of first subarray
    j = 0; // Initial index of second subarray
    k = l; // Initial index of merged subarray
    switch (choice) {
    case 1:
        while (i < n1 && j < n2)
        {
            if (L[i].tutorID <= R[j].tutorID)
            {
                tutorDetails[k] = L[i];
                i++;
            }
            else
            {
                tutorDetails[k] = R[j];
                j++;
            }
            k++;
        }
        break;
```

Figure 3.1.3.1 Merge sort (split array)Function

The merge sort function in array is split into 2 parts which are the merge sort splitting array function and the merge sort merging sorted list function. The first function will split the array list into single elements and the second will sort and merge the elements into a sorted array list. The split array function will find the middle of the array and split it into 2 equal parts and the process is repeated until all the array is split down to single elements.

```
void mergeSort(int l, int r, int choice)

{   /*check if left l is greater than right r*/
    if (l < r)
    {
        // Same as (l+r)/2, but avoids overflow for
        // large l and h
        int m = l + (r - l) / 2;

        // Sort first and second halves
        mergeSort(l, m,choice); //call merge sort function
        mergeSort(m + 1, r, choice); //call merge sort function

        merge(l, m, r, choice); //call merge function
    }
}
```

Figure 3.1.3.2 Merge Sort (sorting array) function

The sorting function which is the second part of the merge sort will sort the elements in an ascending order and pass the values back to first function to be finalized and put together the sorted array list. The switch is implemented for the users to choose which tutor details to be sorted from such as the tutor ratings.

3.1.4 | Check vacant slot

```
/*function that checks whether the list is empty or full*/
int check_vacant_slot() {
    int i = 0; int emptySlot = 0; int vacantSlot = 0;
    /*perform 100 times counting*/
    while (i < 100) {
        /*check if the tutor id in index is NULL*/
        if (tutorDetails[i].tutorID != NULL) {
            vacantSlot++; //increment vacant slot
        }
        else {
            emptySlot++; //increment empty slot
        }
        i++;
    }
    if (vacantSlot == 0) {
        return 1; //if vacantSlot is empty, it will return 1 meaning the array is empty
    }
    else if (emptySlot == 0) {
        return 2; //if emptySlot is empty, it will return 1 meaning the array is full
    }
    else {
        return 3;
    }
}
```

Figure 3.1.4.1 Check vacant function

The method above will check if there is any free slot in the array, if there is the user is free to proceed to add tutor. If the array has already been filled with tutors, the system will not accept any more tutor to be added and the request to add more tutor will be rejected. This method will be call during the process of adding tutor to check if there's any more vacant slot to add and also be call when the system needs to check if the array is empty since the other function of this method is to check if the array is empty.

3.1.5 | Modify Tutor Details

```
/*modify tutor details*/
void modify_tutor_details(int index) {
    int choice;
    string address, phoneNumber;
    char confirm; char choiceConfirm;
    system("CLS"); //clear the screen
    /*display the details of the tutor that to be modified*/
    cout << "\t| MODIFY TUTOR MENU |" << endl;
    cout << "\t" << setw(4); cout << "|TUTOR ID|"; cout << setw(13) << "|FIRST NAME|"; cout << setw(12) << "|LAST NAME|";
    cout << setw(6) << "|PAY|"; cout << setw(14) << "|CENTRE CODE|"; cout << setw(18) << "|CENTRE LOCATION|"; cout << setw(14) << "|CENTRE NAME|";
    cout << setw(11) << "|SUB CODE|"; cout << setw(11) << "|SUB NAME|"; cout << setw(15) << "|PHONE NUM|"; cout << setw(21) << "|ADDRESS|"; cout << setw(23) << "|RATINGS|" << endl;
    cout << "\t"<<setw(6) << tutorDetails[index].tutorID << setw(13) << tutorDetails[index].tutorFirstName << setw(13) << tutorDetails[index].tutorLastName << setw(8);
    cout << tutorDetails[index].hourlyPayRate << setw(10) << tutorDetails[index].center.tuitionCenterCode << setw(21) << tutorDetails[index].center.tuitionCenterLocation;
    cout << setw(12) << tutorDetails[index].center.tuitionCenterName;
    cout << setw(12) << tutorDetails[index].teachSub.subjectCode << setw(13) << tutorDetails[index].teachSub.subjectName;
    cout << setw(16) << tutorDetails[index].phoneNumber << setw(30) << tutorDetails[index].tutorAddress;
    cout << setw(10) << tutorDetails[index].rating;
    cout << endl << endl;
```

Figure 3.1.5.1 Modify Tutor Details Function

The diagram above is a part of the modify tutor details method, which the purpose of this method is to handle the modification of the tutor details. This part of method will display the details of the tutor chosen. Before, displaying the tutor details, the system will first clear the command prompt to prevent the user from looking at the wrong section.

```
error:
    cout << "\tARE YOU SURE THIS IS THE RECORD YOU WANT TO MODIFY? (Y/N) : ";
    cin >> choiceConfirm;
    /*check if user is to modify this detail*/
    if (toupper(choiceConfirm) == 'Y') {
        cout << "\tWHICH DETAILS WOULD YOU LIKE TO CHANGE?" << endl;
        cout << "\t1. PHONE NUMBER" << endl;
        cout << "\t2. ADDRESS     " << endl;
        cout << "\t3. EXIT BACK TO MAIN MENU" << endl << "\t";
    retry:
        cin >> choice;
        /*check the choice of the user*/
        switch (choice) {
        case 1:
        phoneConfirmation:
            cout << "\tPLEASE INSERT NEW PHONE NUMBER : ";
            cin >> phoneNumber;
            cout << "\tYOU ENTERED THE NUMBER " << phoneNumber << endl;
        wrongInput:
            cout << "\tIS THIS THE PHONE NUMBER YOU WANTED? (Y/N)" << endl << "\t";
            cin >> confirm;
            /*validate if customer had key in the right number*/
            if (toupper(confirm) == 'Y') {
                tutorDetails[index].phoneNumber = phoneNumber;
                cout << "\tPHONE NUMBER HAD SUCCESSFULLY CHANGED" << endl;
                Sleep(3000); //wait 3 second
                cout << "   *****      RETURNING TO MAIN MENU      *****";
                Sleep(3000); //wait 3 second
                system("CLS"); //clear the screen
                return;
            }
            else if (toupper(confirm) == 'N') {
                cout << "\tINSERT AGAIN " << endl;
                goto phoneConfirmation; //go to phoneConfirmation location
            }
            else {
                cout << "\tWRONG INPUT, PLEASE TRY AGAIN" << endl;
                goto wrongInput; //go to wrongInput location
            }
            break;
```

Figure 3.1.5.2 Modify tutor Details (Switch) Function

The diagram above shows the second part of the modify tutor details, in this section, user can choose what time of element to check from address to phone number. User will then be asked to input the new phone number or address after choosing their desired option. A verification is done at the end of each option to prevent user from input the wrong data.

```
case 2:
addressConfirmation:
    cout << "\tPLEASE INSERT ADDRESS : ";
    cin.ignore();
    getline(cin, address);
    cout << "\tYOU ENTERED THE ADDRESS " << address << endl;
wrongInputA:
    cout << "\tIS THIS THE ADDRESS YOU WANTED? (Y/N)" << endl << "\t";
    cin >> confirm;
    /*validate if customer had key in the right address*/
    if (toupper(confirm) == 'Y') {
        tutorDetails[index].tutorAddress = address;
        cout << "\tADDRESS HAD SUCCESSFULLY CHANGED" << endl;
        Sleep(3000); //wait for 3 seconds
        cout << "   *****      RETURNING TO MAIN MENU      *****";
        Sleep(3000); //wait for 3 seconds
        system("CLS"); //clear the screen
        return;
    }
    else if (toupper(confirm) == 'N') {
        cout << "\tINSERT AGAIN " << endl;
        goto addressConfirmation; //go to addressConfirmation location
    }
    else {
        cout << "\tWRONG INPUT, PLEASE TRY AGAIN" << endl;
        goto wrongInputA; //go to wronginputA location
    }
    break;
case 3:
    cout << "   *****      RETURNING TO MAIN MENU      *****";
    Sleep(3000); //wait for 3 seconds
    system("CLS");
    break;
default:
    cout << "\tTHE INPUT IS NOT VALID, PLEASE TRY AGAIN !" << endl;
    goto retry; //go to retry location
}
```

Figure 3.1.5.3 Modify Tutor (Verification) Function

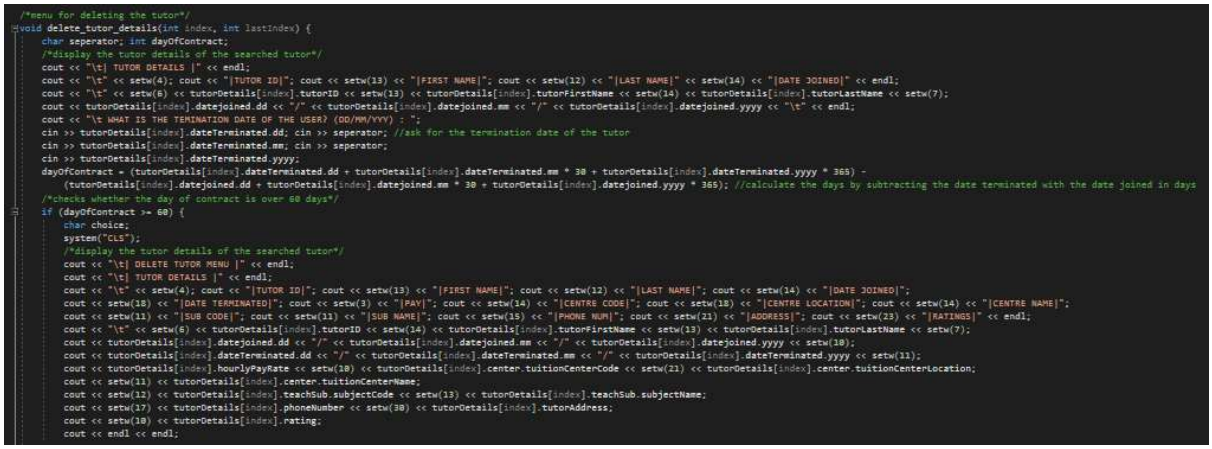
This section of code will be called when the user wanted to modify the tutor's details, same as the code section for modifying phone number, a verification is created to prevent logical error.

```
    }
    else if (toupper(choiceConfirm) == 'N') {
        cout << "\tPLEASE TRY AGAIN !" << endl<<endl;
        cout << "   *****      RETURNING TO MAIN MENU      *****";
        Sleep(3000); //wait for 3 seconds
        system("CLS"); //clear the screen
    }
    else {
        cout << "\tPLEASE TRY AGAIN WITH VALID INPUT!" << endl;
        Sleep(3000); //wait for 3 seconds
        goto error; //clear the screen
    }
}
```

Figure 3.1.5.4 Modify Tutor (Incorrect input) Function

This is the last part of modify tutor details. The section of code will only be called when the user input the wrong input, or the user want to go back to the main menu.

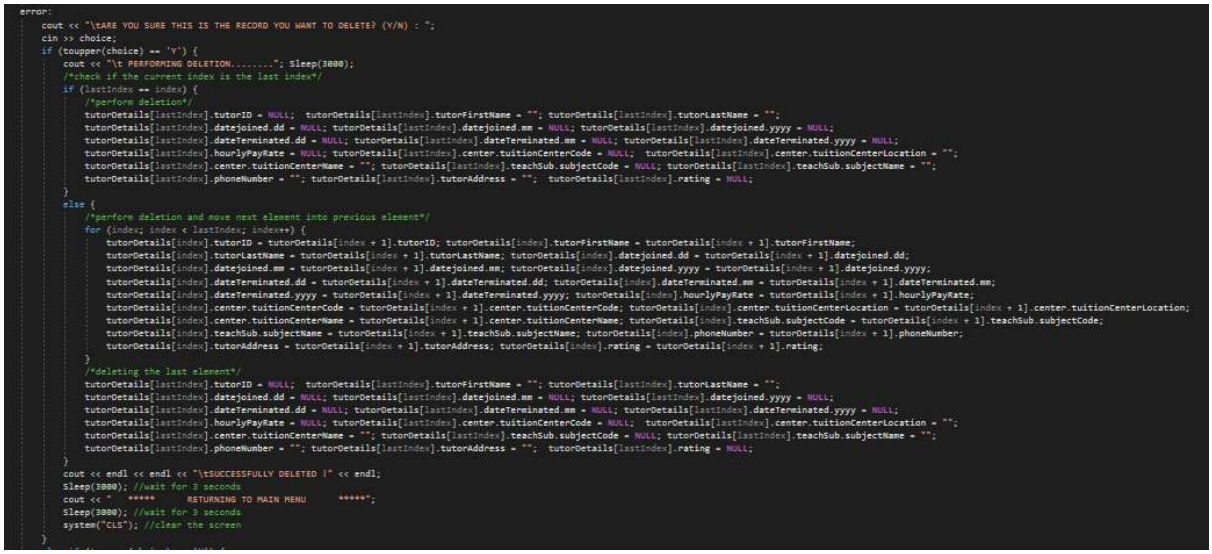
3.1.6 | Delete Tutor Details



```
/*menu for deleting the tutor*/
void delete_tutor_details(int index, int lastIndex) {
    char seperator; int dayOfContract;
    /*display the tutor details of the searched tutor*/
    cout << "\t| TUTOR DETAILS |" << endl;
    cout << "\t" << setw(4); cout << "|TUTOR ID|"; cout << setw(13) << "|FIRST NAME|"; cout << setw(12) << "|LAST NAME|" << setw(14) << "|DATE JOINED|" << endl;
    cout << "\t" << setw(6) << tutorDetails[index].tutorID << setw(13) << tutorDetails[index].tutorFirstName << setw(14) << tutorDetails[index].tutorLastName << setw(7);
    cout << tutorDetails[index].datejoined.dd << "/" << tutorDetails[index].datejoined.mm << "/" << tutorDetails[index].datejoined.yyyy << "\t" << endl;
    cout << "\t WHAT IS THE TERMINATION DATE OF THE USER? (DD/MM/YYY) : ";
    cin >> tutorDetails[index].dateTerminated.dd; cin >> seperator; //ask for the termination date of the tutor
    cin >> tutorDetails[index].dateTerminated.mm; cin >> seperator;
    cin >> tutorDetails[index].dateTerminated.yyyy;
    dayOfContract = (tutorDetails[index].dateTerminated.dd + tutorDetails[index].dateTerminated.mm * 30 + tutorDetails[index].dateTerminated.yyyy * 365) -
        (tutorDetails[index].datejoined.dd + tutorDetails[index].datejoined.mm * 30 + tutorDetails[index].datejoined.yyyy * 365); //calculate the days by subtracting the date terminated with the date joined in days
    /*checks whether the day of contract is over 60 days*/
    if (dayOfContract >= 60) {
        char choice;
        system("CLS");
        /*display the tutor details of the searched tutor*/
        cout << "\t| DELETE TUTOR MENU |" << endl;
        cout << "\t| TUTOR DETAILS |" << endl;
        cout << "\t" << setw(4); cout << "|TUTOR ID|"; cout << setw(13) << "|FIRST NAME|"; cout << setw(12) << "|LAST NAME|"; cout << setw(14) << "|DATE JOINED|";
        cout << setw(18) << "|DATE TERMINATED|"; cout << setw(3) << "|PAY|"; cout << setw(14) << "|CENTRE CODE|"; cout << setw(18) << "|CENTRE LOCATION|"; cout << setw(14) << "|CENTRE NAME|";
        cout << setw(11) << "|SUB CODE|"; cout << setw(11) << "|SUB NAME|"; cout << setw(15) << "|PHONE NUM|"; cout << setw(21) << "|ADDRESS|"; cout << setw(23) << "|RATINGS|" << endl;
        cout << "\t" << setw(6) << tutorDetails[index].tutorID << setw(14) << tutorDetails[index].tutorFirstName << setw(13) << tutorDetails[index].tutorLastName << setw(7);
        cout << tutorDetails[index].datejoined.dd << "/" << tutorDetails[index].datejoined.mm << "/" << tutorDetails[index].datejoined.yyyy << setw(10);
        cout << tutorDetails[index].dateTerminated.dd << "/" << tutorDetails[index].dateTerminated.mm << "/" << tutorDetails[index].dateTerminated.yyyy << setw(11);
        cout << tutorDetails[index].hourlyPayRate << setw(10) << tutorDetails[index].center.tuitionCenterCode << setw(21) << tutorDetails[index].center.tuitionCenterLocation;
        cout << setw(11) << tutorDetails[index].center.tuitionCenterName;
        cout << setw(12) << tutorDetails[index].teachSub.subjectCode << setw(13) << tutorDetails[index].teachSub.subjectName;
        cout << setw(17) << tutorDetails[index].phoneNumber << setw(30) << tutorDetails[index].tutorAddress;
        cout << setw(10) << tutorDetails[index].rating;
        cout << endl << endl;
```

Figure 3.1.6.1 Deletion Function

This method is like modify tutor details but instead of modifying the details, this method deletes it. The diagram above shows similar code compare to modify tutor details by showing the tutor details that are chosen by the user.



```
error:
    cout << "\tARE YOU SURE THIS IS THE RECORD YOU WANT TO DELETE? (Y/N) : ";
    cin >> choice;
    if (toupper(choice) == 'Y') {
        cout << "\t PERFORMING DELETION......"; Sleep(3000);
        /*check if the current index is the last index*/
        if (lastIndex == index) {
            /*perform deletion*/
            tutorDetails[lastIndex].tutorID = NULL;  tutorDetails[lastIndex].tutorFirstName = ""; tutorDetails[lastIndex].tutorLastName = "";
            tutorDetails[lastIndex].datejoined.dd = NULL; tutorDetails[lastIndex].datejoined.mm = NULL; tutorDetails[lastIndex].datejoined.yyyy = NULL;
            tutorDetails[lastIndex].dateTerminated.dd = NULL; tutorDetails[lastIndex].dateTerminated.mm = NULL; tutorDetails[lastIndex].dateTerminated.yyyy = NULL;
            tutorDetails[lastIndex].hourlyPayRate = NULL; tutorDetails[lastIndex].center.tuitionCenterCode = NULL;  tutorDetails[lastIndex].center.tuitionCenterLocation = "";
            tutorDetails[lastIndex].center.tuitionCenterName = ""; tutorDetails[lastIndex].teachSub.subjectCode = NULL; tutorDetails[lastIndex].teachSub.subjectName = "";
            tutorDetails[lastIndex].phoneNumber = ""; tutorDetails[lastIndex].tutorAddress = "";  tutorDetails[lastIndex].rating = NULL;
        }
        else {
            /*perform deletion and move next element into previous element*/
            for (index; index < lastIndex; index++) {
                tutorDetails[index].tutorID = tutorDetails[index + 1].tutorID; tutorDetails[index].tutorFirstName = tutorDetails[index + 1].tutorFirstName;
                tutorDetails[index].tutorLastName = tutorDetails[index + 1].tutorLastName; tutorDetails[index].datejoined.dd = tutorDetails[index + 1].datejoined.dd;
                tutorDetails[index].datejoined.mm = tutorDetails[index + 1].datejoined.mm; tutorDetails[index].datejoined.yyyy = tutorDetails[index + 1].datejoined.yyyy;
                tutorDetails[index].dateTerminated.dd = tutorDetails[index + 1].dateTerminated.dd; tutorDetails[index].dateTerminated.mm = tutorDetails[index + 1].dateTerminated.mm;
                tutorDetails[index].dateTerminated.yyyy = tutorDetails[index + 1].dateTerminated.yyyy; tutorDetails[index].hourlyPayRate = tutorDetails[index + 1].hourlyPayRate;
                tutorDetails[index].center.tuitionCenterCode = tutorDetails[index + 1].center.tuitionCenterCode; tutorDetails[index].center.tuitionCenterLocation = tutorDetails[index + 1].center.tuitionCenterLocation;
                tutorDetails[index].center.tuitionCenterName = tutorDetails[index + 1].center.tuitionCenterName; tutorDetails[index].teachSub.subjectCode = tutorDetails[index + 1].teachSub.subjectCode;
                tutorDetails[index].teachSub.subjectName = tutorDetails[index + 1].teachSub.subjectName; tutorDetails[index].phoneNumber = tutorDetails[index + 1].phoneNumber;
                tutorDetails[index].tutorAddress = tutorDetails[index + 1].tutorAddress; tutorDetails[index].rating = tutorDetails[index + 1].rating;
            }
            /*deleting the last element*/
            tutorDetails[lastIndex].tutorID = NULL;  tutorDetails[lastIndex].tutorFirstName = ""; tutorDetails[lastIndex].tutorLastName = "";
            tutorDetails[lastIndex].datejoined.dd = NULL; tutorDetails[lastIndex].datejoined.mm = NULL; tutorDetails[lastIndex].datejoined.yyyy = NULL;
            tutorDetails[lastIndex].dateTerminated.dd = NULL; tutorDetails[lastIndex].dateTerminated.mm = NULL; tutorDetails[lastIndex].dateTerminated.yyyy = NULL;
            tutorDetails[lastIndex].hourlyPayRate = NULL; tutorDetails[lastIndex].center.tuitionCenterCode = NULL;  tutorDetails[lastIndex].center.tuitionCenterLocation = "";
            tutorDetails[lastIndex].center.tuitionCenterName = ""; tutorDetails[lastIndex].teachSub.subjectCode = NULL; tutorDetails[lastIndex].teachSub.subjectName = "";
            tutorDetails[lastIndex].phoneNumber = ""; tutorDetails[lastIndex].tutorAddress = "";  tutorDetails[lastIndex].rating = NULL;
        }
        cout << endl << endl << "\tSUCCESSFULLY DELETED !" << endl;
        Sleep(3000); //wait for 3 seconds
        cout << "  *****     RETURNING TO MAIN MENU      *****";
        Sleep(3000); //wait for 3 seconds
        system("CLS"); //clear the screen
    }
```

Figure 3.1.6.2 Deletion Function (Process)

This section of code from the delete tutor method will initiate the deletion process. The code will first check if the tutor details chosen is the last element. If yes it will delete the last element. If the tutor details are not the last element, a for loop will be used to loop the element until the chosen element is push to the last element. After that deletion will be done which deletes the last element.

```
    else if (toupper(choice) == 'N') {
        cout << "\tPLEASE TRY AGAIN !!!" << endl << endl;
        cout << "    *****      RETURNING TO MAIN MENU      *****";
        Sleep(3000); //wait for 3 seconds
        system("CLS"); //clear the screen
        return;
    }
    else {
        cout << "\tPLEASE INSERT A VALID INPUT!";
        goto error; //go to error location
    }
}
else {
    cout << "\tTERMINATION DATE IS NOT OVER 60 DAYS!" << endl << endl;
    cout << "\tDELETE EXECUTION DENIED" << endl;
    cout << "    *****      RETURNING TO MAIN MENU      *****";
    Sleep(3000); //wait for 3 seconds
    system("CLS"); //clear the screen
    return;
}
```

Figure 3.1.6.3 Deletion Function (Verification)

The rest of the code will be handling the wrong input by users or if the user regrets it decision and wants to get back to main menu.

3.1.7 | Generate Report

```
/*menu for generating the number of vacant and empty slots of the system*/
int generate_report_menu() {
    int i = 0; int emptySlot = 0; int vacantSlot = 0; int random;
    /*perform 100 times of calculation*/
    while (i < 100) {
        /*check if tutor id in selected index is NULL*/
        if (tutorDetails[i].tutorID != NULL) {
            vacantSlot++; //increment vacantSlot
        }
        else {
            emptySlot++; //increment emptySlot
        }
        i++;
    }
    cout << "   AVAILABLE SLOT = " << emptySlot << endl;
    cout << "   VACANT SLOT    = " << vacantSlot << endl;
    cout << "   PRESS ENTER TO LEAVE ......" << endl;
    cout << "   "; cin.get();
    /*check enter to leave*/
    if (cin.get() == '\n') {
        cout << "   EXITING BACK TO MAIN MENU   " << endl;
        Sleep(2000); //wait for 2 seconds
        system("CLS"); //clear the screen
    }
    return vacantSlot;
}
```

Figure 3.1.7.1 Display Report Function

The method allow use to generate a report for viewing the available slot for adding tutor. The number of free slot and vacant slot will be shown after the report has been generated.

3.1.8 | Exponential Search

```
/*exponential search algorithm to search tutor ID*/
int exponential_search(int n, int tutID) {

    /*check whether given id is at first location*/
    if (tutorDetails[0].tutorID == tutID) {
        return 0;
    }
    int i = 1;
    /*find the range for binary search by doubling the i*/
    while (i < n && tutorDetails[i].tutorID <= tutID) {
        i = i * 2;
    }

    return binarySearch(i / 2, min(i, n), tutID); //call binary search for the found range
}

/*binary search algorithm to search tutor ID*/
int binarySearch(int left, int right, int tutID) {
    if (right >= left)
    {
        int mid = left + (right - left) / 2; //calculation for middle
        if (tutorDetails[mid].tutorID == tutID)  //if the tutor id in the middle index then return
            return mid;
        else if (tutorDetails[mid].tutorID > tutID) //if the tutor id in the middle index is greater than the given tutor ID
            return binarySearch(left, mid - 1, tutID); //put middle - 1 as the int r in the binary search
        else
            return binarySearch(mid + 1, right, tutID); //put middle + 1 as the int l in the binary search
    }
    return -1; //reached element is not present in array
}
```

Figure 3.1.8.1 Exponent Search Function

The exponential search will be used when there is a need to search for data in the array. As mention in our proposal, the exponential search will require binary search to assist will the execution. The searching starts by checking if the data is in the first location, if not the it will find the range of where the data might be and call binary search to search for the data which the binary search will proceed the searching by doing linear search by keep diving the array in two to find the range closest to the wanted data. The exponential search works as an enhancement for binary search to lower the calculation power as well as calculation time.

3.2 | Linked List

3.2.1 | Add Tutor

```
/* add tutor details in the end of linked list */
void addTutorDetails() {
    newnode = new Tutor; //creating a new node for the linked list
    newnode->datejoined = new Date;
    newnode->dateTerminated = new Date;
    newnode->center = new TuitionCenter;
    newnode->teachSub = new Subject;
    newnode->tutorID = tutorID;
    newnode->next = NULL;
    newnode->previous = NULL;
    /* check whether start of linked list is empty */
        if (start == NULL) {
            enterDetails(); //enter details into new node
            tutorID = tutorID + 1; //increment the generated tutor ID
            start = last = newnode; //set the newnode to start node and last node
        }
        else {
            enterDetails(); //enter details into new node
            tutorID = tutorID + 1; //increment the generated tutor ID
            newnode->previous = last; // newnode previous pointer is the previous last node
            last->next = newnode; // previous last node next pointer is now new node
            last = newnode; //set the newnode to the last node
        }
}
```

Figure 3.2.1.1 Add Tutor Details Function (Linked List)

This function allows users to input tutor details from tutor name, date joined, date terminated, location of teaching to which subject tutor. If the linked list is empty, the details entered will be added at the top of the list as the first node which both the start node and last node will be set to, else the details will be added to the list as the next node, which will set the new node previous pointer as the previous last node and the last node next pointer to the current node, which is the new node. And lastly the new node will be set as the last node as such the new tutor detail added to the linked list will always be at the bottom. The tutor ID will be automatically generated and as stated in the assumption, unique to only the user and will have no reoccurring IDs.

3.2.2 | Display Ascending and Descending

```
void display_ascending() {
    system("CLS");
    /*formating for the title for details of tutor*/
    cout << "\t| TUTOR DETAILS |" << endl << endl;
    cout << "\t" << setw(4); cout << "|TUTOR ID|"; cout << setw(13) << "|FIRST NAME|"; cout << setw(12) << "|LAST NAME|"; cout << setw(14) << "|DATE JOINED|";
    cout << setw(18) << "|DATE TERMINATED|"; cout << setw(6) << "|PAY|"; cout << setw(14) << "|CENTRE CODE|"; cout << setw(18) << "|CENTRE LOCATION|"; cout << setw(14) << "|CENTRE NAME|";
    cout << setw(11) << "|SUB CODE|"; cout << setw(11) << "|SUB NAME|"; cout << setw(15) << "|PHONE NUM|"; cout << setw(21) << "|ADDRESS|"; cout << setw(23) << "|RATINGS|" << endl;
    int i = 0;
    current = start; // set current node = start node
    /*while the current node is not null, print all the details of the tutor of the current node*/
    while (current != NULL) {
        cout << "\t" << setw(6) << current->tutorID << setw(14) << current->tutorFirstName << setw(13) << current->tutorLastName << setw(7);
        cout << current->datejoined->dd << "/" << current->datejoined->mm << "/" << current->datejoined->yyyy << setw(10);
        cout << current->dateTerminated->dd << "/" << current->dateTerminated->mm << "/" << current->dateTerminated->yyyy << setw(11);
        cout << current->hourlyPayRate << setw(10) << current->center->tuitionCenterCode << setw(21) << current->center->tuitionCenterLocation;
        cout << setw(11) << current->center->tuitionCenterName;
        cout << setw(12) << current->teachSub->subjectCode << setw(13) << current->teachSub->subjectName;
        cout << setw(17) << current->phoneNumber << setw(30) << current->tutorAddress;
        cout << setw(10) << current->rating;
        cout << endl;
        current = current->next; // set the current node to next node
    }
    /*press enter to leave the current function*/
    cout << "    PRESS ENTER TO LEAVE ......" << endl;
    cout << "    "; cin.get();
    if (cin.get() == '\n') {
        cout << "    EXITING BACK TO MAIN MENU    " << endl;
        Sleep(2000); //wait for 2 seconds
        system("CLS"); //clear the screen
    }
```

Figure 3.2.2.1 Display Ascending Linked list (Linked List)

This function displays the tutor details from the linked list either from top to bottom (ascending) or bottom to top (descending) and will print tutor details line by line. The current pointer will be set to start and the pointer and will print the details if current pointer does not equal null. When the details are printed the current pointer will be set to the next node and will print the node it is pointing. The descending display function is the same as the ascending, but the current pointer is set to the last node instead

3.2.3 | Checking Linked List

```
int check_vacant_slot(Tutor *temp) {
    int emptySlot = 0; int vacantSlot = 0;
    if (start == NULL) {
        return 1; //if the start is empty, it will return 1 meaning the list is empty
    }
    else {
        while (temp != NULL) { // while the pointer is not null continue to tranverse the next pointer
            vacantSlot++; //will increment the vacantSlot
            temp = temp->next;
        }
        emptySlot = 100 - vacantSlot; // calculate empty slots in system
        if (emptySlot == 0) {
            return 2; //if the empty slot is 0, it will return 2 meaning the list is full
        }
        else {
            return 3;
        }
    }
}
```

Figure 3.2.3.1 Check Vacant Slot function (Linked List)

The following function will count the number of nodes using the temp pointer going through the list. The number will be subtracted by 100, which is the maximum number users registered and will return the value of remaining slots.

3.2.4 | Bubble Sort Algorithm

```
void bubbleSort(struct Tutor *start, int choice)
{
    int swapped, i;
    struct Tutor *ptr1;
    struct Tutor *lptr = NULL;

    /* Checking for empty list */
    if (start == NULL)
        return;
    switch (choice) {
    case 1:
        do {
            swapped = 0;
            ptr1 = start;
            while (ptr1->next != lptr) // while pointer ptr1 is not NULL continue to transverse
            {
                /*if the given choice is 1 means is sorting by tutorID
                  thus, if current tutor id is greater than the next tutor id
                  then it will continue to swap the positions*/
                if (ptr1->tutorID > ptr1->next->tutorID)
                {
                    swap(ptr1->tutorID, ptr1->next->tutorID); swap(ptr1->tutorFirstName, ptr1->next->tutorFirstName);
                    swap(ptr1->tutorLastName, ptr1->next->tutorLastName);
                    swap(ptr1->datejoined->dd, ptr1->next->datejoined->dd); swap(ptr1->datejoined->mm, ptr1->next->datejoined->mm);
                    swap(ptr1->datejoined->yyyy, ptr1->next->datejoined->yyyy);
                    swap(ptr1->dateTerminated->dd, ptr1->next->dateTerminated->dd); swap(ptr1->dateTerminated->mm, ptr1->next->dateTerminated->mm);
                    swap(ptr1->dateTerminated->yyyy, ptr1->next->dateTerminated->yyyy);
                    swap(ptr1->hourlyPayRate, ptr1->next->hourlyPayRate);
                    swap(ptr1->center->tuitionCenterCode, ptr1->next->center->tuitionCenterCode);
                    swap(ptr1->center->tuitionCenterName, ptr1->next->center->tuitionCenterName);
                    swap(ptr1->center->tuitionCenterLocation, ptr1->next->center->tuitionCenterLocation);
                    swap(ptr1->teachSub->subjectCode, ptr1->next->teachSub->subjectCode);
                    swap(ptr1->teachSub->subjectName, ptr1->next->teachSub->subjectName);
                    swap(ptr1->phoneNumber, ptr1->next->phoneNumber); swap(ptr1->tutorAddress, ptr1->next->tutorAddress);
                    swap(ptr1->rating, ptr1->next->rating);
                    swapped = 1; //set swap = 1 meaning it needs to continue to perform sorting
                }
                ptr1 = ptr1->next;
            }
            lptr = ptr1;
        } while (swapped); // if the swap is 0 it will then stop the sort
        break;
```

Figure 3.2.4.1 Bubble Sort Function (Linked List)

The algorithm will compare and swap the nodes with the next node if the first node if the element in it is bigger and will do it in a loop endlessly until no swap has been made between the nodes. The function will do it for all the tutor details. Switch is also implemented for users to choose which tutor details to sort from, such as sorting by tutor ID or tutor ratings.

3.2.5 | Delete tutor details

```
/*menu for deleting the tutor*/
void delete_tutor_details(Tutor *temp) {
    char seperator; int dayOfContract;
    /*display the tutor details of the searched tutor*/
    cout << "\t| TUTOR DETAILS |" << endl;
    cout << "\t" << setw(4); cout << "|TUTOR ID|"; cout << setw(13) << "|FIRST NAME|"; cout << setw(12) << "|LAST NAME|" << setw(14) << "|DATE JOINED|" << endl;
    cout << "\t" << setw(6) << temp->tutorID << setw(13) << temp->tutorFirstName << setw(14) << temp->tutorLastName << setw(7);
    cout << temp->datejoined->dd << "/" << temp->datejoined->mm << "/" << temp->datejoined->yyyy << "\t" << endl;
    cout << "\t WHAT IS THE TEMINATION DATE OF THE USER? (DD/MM/YYY) : ";
    cin >> temp->dateTerminated->dd; cin >> seperator; //ask for the termination date of the tutor
    cin >> temp->dateTerminated->mm; cin >> seperator;
    cin >> temp->dateTerminated->yyyy;
    dayOfContract = ((temp->dateTerminated->dd + temp->dateTerminated->mm * 30 + temp->dateTerminated->yyyy * 365) -
        (temp->datejoined->dd + temp->datejoined->mm * 30 + temp->datejoined->yyyy * 365); //calculate the days by subtracting the date terminated with the date joined in days
    /*checks whether the day of contract is over 60 days*/
    if (dayOfContract >= 60) {
        char choice;
        system("CLS");
        /*display the tutor details of the searched tutor*/
        cout << "\t| DELETE TUTOR MENU |" << endl;
        cout << "\t| TUTOR DETAILS |" << endl;
        cout << "\t" << setw(4); cout << "|TUTOR ID|"; cout << setw(13) << "|FIRST NAME|"; cout << setw(12) << "|LAST NAME|"; cout << setw(14) << "|DATE JOINED|";
        cout << setw(18) << "|DATE TERMINATED|"; cout << setw(3) << "|PAY|"; cout << setw(14) << "|CENTRE CODE|"; cout << setw(18) << "|CENTRE LOCATION|"; cout << setw(14) << "|CENTRE NAME|";
        cout << setw(11) << "|SUB CODE|"; cout << setw(11) << "|SUB NAME|"; cout << setw(15) << "|PHONE NUM|"; cout << setw(21) << "|ADDRESS|"; cout << setw(23) << "|RATINGS|" << endl;
        cout << "\t" << setw(6) << temp->tutorID << setw(14) << temp->tutorFirstName << setw(13) << temp->tutorLastName << setw(7);
        cout << temp->datejoined->dd << "/" << temp->datejoined->mm << "/" << temp->datejoined->yyyy << setw(10);
        cout << temp->dateTerminated->dd << "/" << temp->dateTerminated->mm << "/" << temp->dateTerminated->yyyy << setw(11);
        cout << temp->hourlyPayRate << setw(10) << temp->center->tuitionCenterCode << setw(21) << temp->center->tuitionCenterLocation;
        cout << setw(11) << temp->center->tuitionCenterName;
        cout << setw(12) << temp->teachSub->subjectCode << setw(13) << temp->teachSub->subjectName;
        cout << setw(17) << temp->phoneNumber << setw(30) << temp->tutorAddress;
        cout << setw(10) << temp->rating;
        cout << endl << endl;
```

Figure 3.2.5.1 Delete Tutor Details Function

This method above is called when the user wants to delete tutor details, there are several parts of this method because the method itself is quite lengthy, this part of the method will display the tutor details chosen and check if it has passed the date of termination. The system will retrieve the chosen tutor details from the nodes in linked list and show in the output stream.

```
error:
    cout << "\tARE YOU SURE THIS IS THE RECORD YOU WANT TO DELETE? (Y/N) : "; //confirmation for deletion
    cin >> choice;
    /*check whether choice is Y or N*/
    if (toupper(choice) == 'Y') {
        cout << "\t PERFORMING DELETION........"; Sleep(3000);
        /*check whether current node is the start or last node*/
        if (temp == start && temp == last) { //current node is the start and last node
            start = NULL; //set start node = NULL
        }
        else if (temp == start) { //current node is the start node
            delete_first_tutor(temp); //call delete first tutor function
        }
        else if (temp == last) { //start node is the start node
            delete_last_tutor(temp); //call delete last tutor function
        }
        else {
            back = temp->previous; //back node is now the previous node of the current node;
            back->next = back->next->next; // the next pointer of the back node is not the next and next pointer of the back node
            temp->next->previous = back; // the pointer next previous of the current node is now the back node
            delete temp; //delete the current node
        }
        cout << endl << endl << "\tSUCCESSFULLY DELETED !" << endl;
        Sleep(3000); //wait for 3 seconds
        cout << "   *****      RETURNING TO MAIN MENU      *****";
        Sleep(3000); //wait for 3 seconds
        system("CLS"); //clear the screen
    }
    else if (toupper(choice) == 'N') {
        cout << "\tPLEASE TRY AGAIN !!!" << endl << endl;
        cout << "   *****      RETURNING TO MAIN MENU      *****";
        Sleep(3000); //wait for 3 seconds
        system("CLS"); //clear the screen
        return;
    }
    else {
        cout << "\tPLEASE INSERT A VALID INPUT!";
        goto error; //go to the error position
    }
}
else {
    cout << "\tTERMINATION DATE IS NOT OVER 60 DAYS!" << endl << endl;
    cout << "\tDELETE EXECUTION DENIED" << endl;
    cout << "   *****      RETURNING TO MAIN MENU      *****";
    Sleep(3000); //wait for 3 seconds
    system("CLS");  //clear the screen
    return;
}
```

Figure 3.2.5.2 Delete Tutor Details (Verification) Function

The second part of the delete tutor details will be a confirmation with the user if he or she really wants to delete the tutor details, if yes, the system will proceed with the deletion process and return the user to main menu after a certain period of time. During the deletion process, the system will first check which node to be delete, if the node is the start node, the function to delete the start node will be call and vice versa for deleting last node in the linked list as well as the node in the middle.

3.2.6 | Delete first and last tutor

```
/*Deleting the last node/tutor*/
void delete_last_tutor(Tutor *temp) {
    temp = last; //set the current node = last node
    last->previous->next = NULL; //the next pointer for the previous node of the last node will be set to NULL
    last = last->previous; //the last node is now the previous node of the current last node
    delete temp; //delete the current node
}

/*Deleting the first node/tutor*/
void delete_first_tutor(Tutor *temp) {
    temp = start; //set the current node = start node
    start->next->previous = NULL; //the next pointer for the previous node of the start node will be set to NULL
    start = start->next; //the start node is now the next node of the current start node
    delete temp; //delete the current node
}
```

Figure 3.2.6.1 Delete Tutor Details (Fist and Last node) Function

Both these methods are call depends on which node to delete, if the first node need to be deleted the delete_first_tutor method will be called and vice versa. The system will set the next pointer from the previous node user wants to delete and set the pointer to null which then proceed to delete last node. For the second method, the system will set the next node of the start node to the new start node of the linked list and clear its previous node pointer and deletes the original start node.

3.2.7 | Modify Tutor Details

```
/*modify tutor details*/
void modify_tutor_details(Tutor *temp) {
    int choice;
    string address, phoneNumber;
    char confirm; char choiceConfirm;
    system("CLS"); //clear the screen
    /*display the details of the tutor that to be modified*/
    cout << "\t| MODIFY TUTOR MENU |" << endl;
    cout << "\t" << setw(4); cout << "|TUTOR ID|"; cout << setw(13) << "|FIRST NAME|"; cout << setw(12) << "|LAST NAME|";
    cout << setw(6) << "|PAY|"; cout << setw(14) << "|CENTRE CODE|"; cout << setw(18) << "|CENTRE LOCATION|"; cout << setw(14) << "|CENTRE NAME|";
    cout << setw(11) << "|SUB CODE|"; cout << setw(11) << "|SUB NAME|"; cout << setw(15) << "|PHONE NUM|"; cout << setw(21) << "|ADDRESS|"; cout << setw(23) << "|RATINGS|" << endl;
    cout << "\t" << setw(6) << temp->tutorID << setw(13) << temp->tutorFirstName << setw(13) << temp->tutorLastName << setw(8);
    cout << temp->hourlyPayRate << setw(10) << temp->center->tuitionCenterCode << setw(21) << temp->center->tuitionCenterLocation;
    cout << setw(12) << temp->center->tuitionCenterName;
    cout << setw(12) << temp->teachSub->subjectCode << setw(13) << temp->teachSub->subjectName;
    cout << setw(16) << temp->phoneNumber << setw(30) << temp->tutorAddress;
    cout << setw(10) << temp->rating;
    cout << endl << endl;
```

Figure 3.2.7.1 Modify Tutor (linked list)

This method will be like modify tutor details for array, at the first section of the method, the system will retrieve data from the node in linked list and output the chosen tutor details to the command prompt.

```
error:
    cout << "\tARE YOU SURE THIS IS THE RECORD YOU WANT TO MODIFY? (Y/N) : ";
    cin >> choiceConfirm;
    /*check if user is to modify this detail*/
    if (toupper(choiceConfirm) == 'Y') {
        cout << "\tWHICH DETAILS WOULD YOU LIKE TO CHANGE?" << endl;
        cout << "\t1. PHONE NUMBER" << endl;
        cout << "\t2. ADDRESS     " << endl;
        cout << "\t3. EXIT BACK TO MAIN MENU" << endl << "\t";
    retry:
        cin >> choice;
        /*check the choice of the user*/
        switch (choice) {
        case 1:
        phoneConfirmation:
            cout << "\tPLEASE INSERT NEW PHONE NUMBER : ";
            cin >> phoneNumber;
            cout << "\tYOU ENTERED THE NUMBER " << phoneNumber << endl;
        wrongInput:
            cout << "\tIS THIS THE PHONE NUMBER YOU WANTED? (Y/N)" << endl << "\t";
            cin >> confirm;
            /*validate if customer had key in the right number*/
            if (toupper(confirm) == 'Y') {
                temp->phoneNumber = phoneNumber; //set the current node phone number to the new number given
                cout << "\tPHONE NUMBER HAD SUCCESSFULLY CHANGED" << endl;
                Sleep(3000); //wait 3 second
                cout << "   *****      RETURNING TO MAIN MENU      *****";
                Sleep(3000); //wait 3 second
                system("CLS"); //clear the screen
                return;
            }
            else if (toupper(confirm) == 'N') {
                cout << "\tINSERT AGAIN " << endl;
                goto phoneConfirmation; //go to phoneConfirmation location
            }
            else {
                cout << "\tWRONG INPUT, PLEASE TRY AGAIN" << endl;
                goto wrongInput; //go to wrongInput location
            }
            break;
```

Figure 3.2.7.2 Modify Tutor (linked list verification)

The second part will be confirming with the user if he or she really wants to do it, if yes, the user can choose if to modify phone number or address. After that, the system will proceed with the modification, the data in the nodes will be modified using a temporary pointer.

```
    case 2:
    addressConfirmation:
        cout << "\tPLEASE INSERT ADDRESS : ";
        cin.ignore();
        getline(cin, address);
        cout << "\tYOU ENTERED THE ADDRESS " << address << endl;
    wrongInputA:
        cout << "\tIS THIS THE ADDRESS YOU WANTED? (Y/N)" << endl << "\t";
        cin >> confirm;
        /*validate if customer had key in the right address*/
        if (toupper(confirm) == 'Y') {
            temp->tutorAddress = address;
            cout << "\tADDRESS HAD SUCCESSFULLY CHANGED" << endl;
            Sleep(3000); //wait for 3 seconds
            cout << "   *****      RETURNING TO MAIN MENU      *****";
            Sleep(3000); //wait for 3 seconds
            system("CLS"); //clear the screen
            return;
        }
        else if (toupper(confirm) == 'N') {
            cout << "\tINSERT AGAIN " << endl;
            goto addressConfirmation; //go to addressConfirmation location
        }
        else {
            cout << "\tWRONG INPUT, PLEASE TRY AGAIN" << endl;
            goto wrongInputA; //go to wrongInputA location
        }
        break;
    case 3:
        cout << "   *****      RETURNING TO MAIN MENU      *****";
        Sleep(3000); //wait for 3 seconds
        system("CLS");
        break;
    default:
        cout << "\tTHE INPUT IS NOT VALID, PLEASE TRY AGAIN !" << endl;
        goto retry; //go to retry location
    }
}
else if (toupper(choiceConfirm) == 'N') {
    cout << "\tPLEASE TRY AGAIN !" << endl << endl;
    cout << "   *****      RETURNING TO MAIN MENU      *****";
    Sleep(3000); //wait for 3 seconds
    system("CLS"); //clear the screen
}
else {
    cout << "\tPLEASE TRY AGAIN WITH VALID INPUT!" << endl;
    Sleep(3000); //wait for 3 seconds
    goto error; //clear the screen
}
```

Figure 3.2.7.3 Modify Tutor (linked list Incorrect Input)

The last part of the modify tutor details for linked list will be the second option of the modification which is the address and the cases which if the user input wrong data and the system need to redirect the user. The modification process works the same as the phone number modification, using a temporary pointer to go into the linked list nodes and modify the address.

3.2.8 | Recursive linear search algorithm

```
/*recursive linear search algorithm searching the pointer that given tutor id is at*/
Tutor *search(Tutor *temp, int tutorID) {
    /*checking the pointer given*/
    if (temp == NULL) // if the given pointer is NULL, return.
    {
        return temp;
    }
    else
    {
        /*checking the whether the tutor id in current link list*/
        if (temp->tutorID == tutorID) //if the current linked list tutor id is the same as the one given by user, return the pointer;
        {
            return temp;
        }
        else {
            search(temp->next, tutorID); // if the tutor ID is not found in the current pointer, perform recursive searching passing the next pointer
        }
    }
}
```

Figure 3.2.8 Recursive linear search algorithm function

This method will be called whenever a searching in the linked list is needed to be done. The code will first check if the given pointer is null, if yes, the system will terminate the method. If no, the system will proceed to check through the linked list until it find the desired data, if not the method will keep calling itself which where the recursive will be as it keep calling itself to rerun the code.

4.0 | Screenshots of input and output

Most of the input and output for both array system and linked list system are the same. So, to save time, only 1 output for each function will be shown.

4.1 | Main Menu

```
|   WELCOME TO TUTOR MANAGEMENT SYSTEM!   |

|  1. ADD A TUTOR RECORD                  |
|  2. DISPLAY TUTOR RECORD                |
|  3. MODIFY TUTOR RECORD                 |
|  4. GENERATE REPORT                     |
|  5. EXIT                                |
|          PLEASE ENTER A CHOICE !        |
```

Figure 4.1.1 This section will be the main menu

Display the options users will be able to choose from in numbers from 1 to 5. If users enter an incorrect input or numbers more then 5, they will be brought back to this menu.

4.2 | Add Tutor

```
|      ENTER TUTOR DETAILS MENU      |
Generated Tutor ID                   : 1
Please enter Tutor Name              : Daniel
Please enter The Last Name of Tutor  : Lee
Please enter Tutor Date Joined       :
Format (DD/MM/YYYY)                  : 12/01/2020
Please enter Hourly Pay Rate (40-80): 45
Please select Tuition Center         :
1. Bukit Bintang
2. Bukit Jalil
3. Puchong
4. Cheras
5. Kepong
Please insert choice (1-5) : 1
Please select Subject Taught         :
1. Chinese
2. Malay
3. English
4. Mathematics
5. Science
Please insert choice (1-5) : 1
Please enter Phone Number            : 012883331
Please enter Tutor Address           : Bukit Jalil
Please enter Tutor Rating     (1-5) : 4
```

Figure 4.1.2: User can add tutor details from the add tutor function

Display and gets input from users about the tutor details that they want to input into the system. Tutor ID is generated automatically and is unique to each and every tutor in the system and will not be repeated.

4.3 | Display Tutor

```
| TUTOR DETAILS |
|TUTOR ID| |FIRST NAME| |LAST NAME| |DATE JOINED| |DATE TERMINATED| |PAY| |CENTRE CODE| |CENTRE LOCATION| |CENTRE NAME| |SUB CODE| |SUB NAME|    |PHONE NUM|          |ADDRESS|              |RATINGS|
     6            F          Lao     23/4/1999         0/0/0        40      1003        Bukit Bintang      Alpha       2001      Chinese      0124020303       22, Medan Kincha, 23000        5
     7            B          Lao     23/4/1999         0/0/0        40      1001        Bukit Bintang      Alpha       2001      Chinese     60134020100        32, Medan  3, 54200        5
     1            A      CHANGJI     23/4/1999         0/0/0        40      1002        Bukit Bintang      Alpha       2001          Mal     60134020303     99,Kuala Lumpur 2, 43200        4
     4            G          Yao     23/4/1999         0/0/0        40      1005        Bukit Bintang      Alpha       2001      Chinese      0134020304         2, Cheras 2, 14000        2
     3            C         Chen     23/4/1999         0/0/0        40      1003        Bukit Bintang      Alpha       2001      Chinese     60134070303     42, Medan Kurau 2, 14000        2
     5            E          Lee     23/4/1999         0/0/0        40      1004        Bukit Bintang      Alpha       2001      Chinese     60174020303     21, Tmn Mahkota 2, 43200        1
     2            D         Yang     23/4/1999         0/0/0        40      1005        Bukit Bintang      Alpha       2001      Chinese     60124020303     32, Tmn Kincala 2, 53200        1
PRESS ENTER TO LEAVE ......
```

Figure 4.1.3 Displaying tutor details based on the chosen sorting way (The abv is sort by rating in descending way)

This display is the descending order of tutor ratings. This how it will visually look for users that want to sort any kind of tutor details such as tutor ID or hourly wages.

4.4 | Display Option

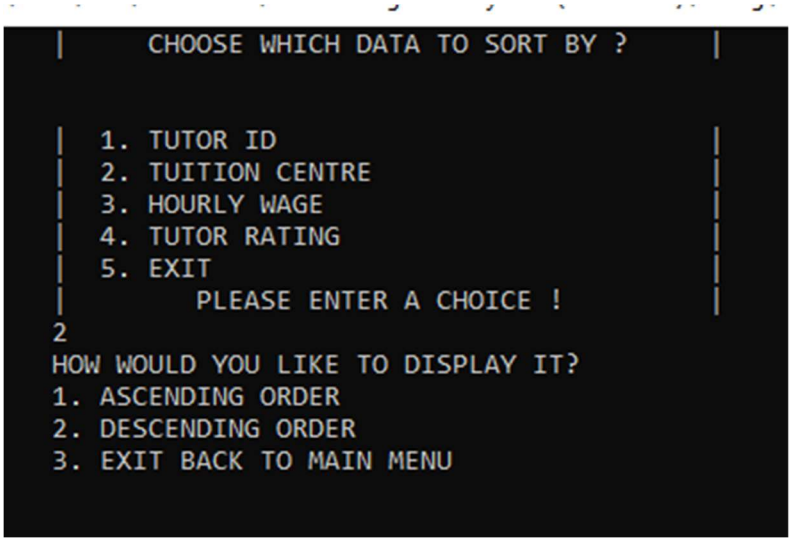


Figure 4.4.1 Sorting Option before displaying data

This display shows users what kind of data they want their tutor details to be sorted from. That includes tutor ID, tuition centre location, hourly wages, and tutor rating. This also allow users to choose how the display order is, either ascending or descending.

4.5 | Modify Menu

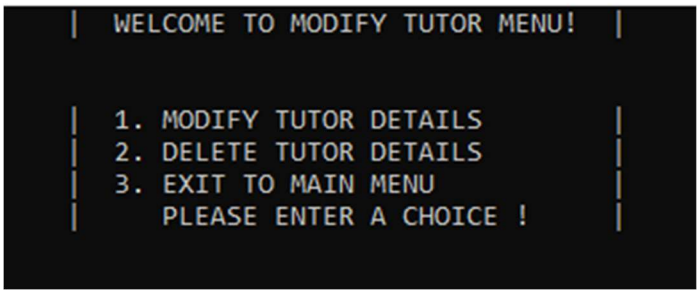


Figure 4.5.1 Modify Menu

This display the choice of what users can modify in the tutor details or delete a tutor. Users may not input numbers that are not in the option or not the function will prompt the users with a massage and bring them back to the main menu.

4.6 | Modify Option

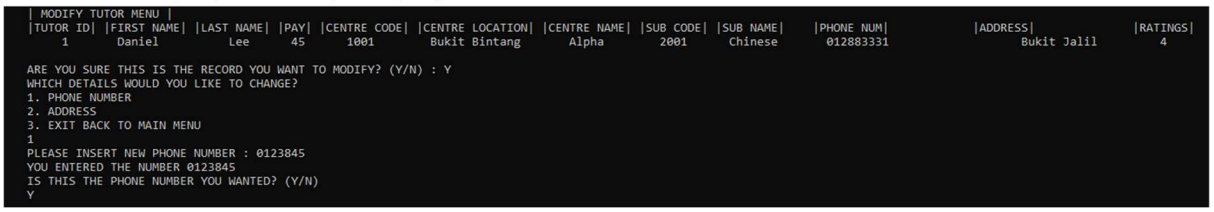


Figure 4.6.1 User can choose what to modify

A tutor is chosen, and users can choose from two options in this display when modifying tutor details. User can only choose to modify the address or phone number of the following tutor.

4.7 | Deletion

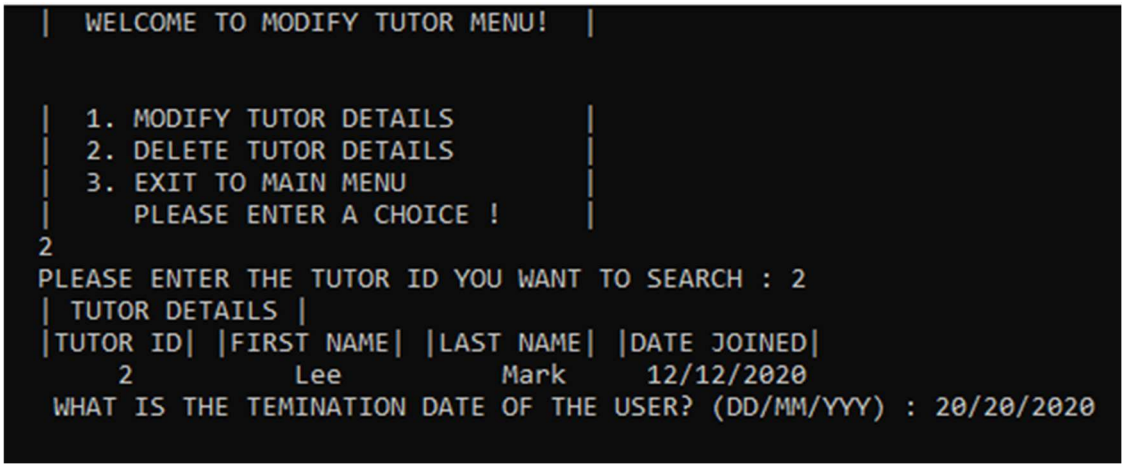


Figure 4.7.1 Deletion of tutor details

The deletion displays allows users to search the ID of the tutor they want to delete and will display the tutor's name and date joined this will prompt the user to enter a termination date and will then go to delete the user from the list.

4.8 | Deletion Confirmation

Figure 4.8.1 System display tutor details that is chosen before being delete, there is a confirmation to make sure user is right

This is a confirmation massage displayed to users to ensure if they intend to delete the tutor in case of incorrect input. If yes then the tutor details will be deleted and if no then this will return the user to the main menu.

4.9 | Generate Report

Figure 4.9.1 A report to check if there are still places left for new tutor

This display will tell the users about the vacant slots in the tuition centre which at the start is set to a hundred. Users can then press enter to leave the report and back to the main menu.

4.10 | Exit

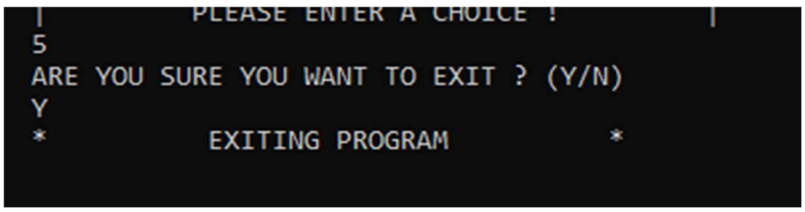


Figure 4.10.1 Exit process

Lastly is the exit display which will allow the user to safely exit the program without any corruption of data. This will be displayed to users to confirm if they intend to exit the program if they have chosen the exit function from the main menu.

5.0 | Array and linked list comparison

Array and linked list have a similar function yet used quite differently from each other. And as such there is advantages and disadvantages in using each of them.

Type	Advantages	Disadvantages
Array	- One of the easiest and better way of saving data with same type and size (Kakria, 2020) - User can save data in any type of dimensional array (Kakria, 2020) - No memory overflow as the system will prepare some memory in contiguous memory locations for the data (Tutorials Cup, 2020)	- Array is required to input fixed number of elements into it that need to be declared in the beginning. Hence more planning is needed in memory allocation. (Kakria, 2020) - During compilation of the array, indexes are not verified and will produce runtime errors, when any indexes pointed which are specified more than the dimension. (Tutorials Cup, 2020)
Linked List	- The space for a linked list can be dynamic and change during runtime which can reduce memory wastage. (Mishra, 2016) - Modifying, adding or even deletion of nodes in a linked list is easy because it do not require the system to make changes to the whole	- Reverse traversing is difficult, which requires a memory space for an extra back pointer in doubly linked list. (Shah, 2018) - Cannot randomly access any element and has to transverse all nodes to get to a specific node. (Shah, 2018)

	list, just changing address will do. (Mishra, 2016)	- On the other hand, linked list which needs pointers in each node will require some memory for the pointer itself (Shah, 2018)

6.0 | Conclusion and Critical Evaluation

6.1 | Conclusion

To conclude everything we have done, we are proud to say that we have successfully build two system which can fulfil the requirement given by our client. The system works perfectly and is able to show what the user desired. Functions provided within the system can be executed successfully. This is not just a chance for us to improve our coding skill but also let us obtain more knowledge on the field of data structure.

6.2 | Critical Evaluation

For critical evaluation, we are confident to say that the system can and will work smoothly, the system has a clean and smart design with can help users to navigate the system. A sleep function is added into the system to make the system more realistic as a loading feature has been added. All necessary data are shown when the users want to see the data and the data can also be arranged in the order they like. Overall, the system has fulfilled all requirements.

On the other hand, we have found out that our verification when inserting data is not very complete. For example, during the insertion of date there are no verification if the data is valid, and the phone number can be random numbers as there are no verification. The way the output is shown when displaying tutor details are a little bit messy as the output does not fit to the command prompt size. Overall, the problems found are minor and will not affect the system by a lot by will reduce the satisfaction of our users.

7.0 | References

[1] Geeks for Geeks, 2019. *Exponential Search.* [Online]
Available at: https://www.geeksforgeeks.org/exponential-search/
[Accessed 8 April 2020].

[2] Holamyself, 2019. *Merge sort, advantages and disadvantages.* [Online]
Available at: https://getrevising.co.uk/grids/merge-sort-advantages-and-disadvantages
[Accessed 11 April 2020].

[3]kakria, R., Unkwnon. *What are Advantages and Disadvantages of arrays?.* [Online]
Available at: http://www.xpode.com/ShowArticle.aspx?Articleid=502
[Accessed 10 5 2020].

[4]Mishra, N., 2016. *dvantages and Disadvantages of Linked List.* [Online]
Available at: https://www.thecrazyprogrammer.com/2016/11/advantages-disadvantages-linked-list.html
[Accessed 10 5 2020].

[5] Sehgal, K., 2018. *A Simplified Explanation of Merge Sort.* [Online]
Available at: https://medium.com/karuna-sehgal/a-simplified-explanation-of-merge-sort-77089fe03bb2
[Accessed 10 April 2020].

[6]Shah, B., 2018. *What are the advantages and disadvantages of linked list?.* [Online]
Available at: https://www.quora.com/What-are-the-advantages-and-disadvantages-of-linked-list?share=1
[Accessed 10 5 2020].

[7] Tech Delight , 2019. *Exponential Search.* [Online]
Available at: https://www.techiedelight.com/exponential-search/
[Accessed 7 April 2020].

[8] Tech Differences. 2020. Difference Between Array And Linked List (With Comparison Chart) - Tech Differences. [online] Available at: <https://techdifferences.com/difference-between-array-and-linked-list.html> [Accessed 14 May 2020].

[9] Tutorials point, 2018. *Exponential Search.* [Online]
Available at: https://www.tutorialspoint.com/Exponential-Search
[Accessed 1 April 2020].

[10] TutorialCup. 2020. Advantages and Disadvantages of Array In C Programming. [online] Available at: <https://www.tutorialcup.com/cprogramming/array-advantages-disadvantages.htm> [Accessed 14 May 2020].

[11]Unknwon, 2020. *Advantages and Disadvantages of Array in C Programming.* [Online]
Available at: https://www.tutorialcup.com/cprogramming/array-advantages-disadvantages.htm
[Accessed 10 5 2020].

8.0 | Task Distribution

	Chan Jia Le TP049952	**Chen Chee Kin** TP053324	**Lee Jin Heng** TP053710
1. Introduction	34%	33%	33%
2. Assumption	34%	33%	33%
3. Data Structure	34%	33%	33%
4. Flow Chart	34%	33%	33%
5. Searching Algorithm	34%	33%	33%
6. Sorting Algorithm	34%	33%	33%
7. Documentation	34%	33%	33%
8. Code Debugging	34%	33%	33%
9. Code compiling	34%	33%	33%