GROUP ASSIGNMENT

TECHNOLOGY PARK MALAYSIA

CT077-3-2 DSTR

DATA STRUCTURES

APU2F2106CS(IS) / APD2F2106CS(DF) / APU2F2106CS(DF) / APD2F2106CS(IS)

HAND OUT DATE	**13 DECEMBER 2021**
HAND IN DATE	**25 FEBRUARY 2022**
GROUP MEMBERS	
BEH CHI HAO	**TP064850**
SIA DE LONG	**TP060810**
THIAN SHAN YOU	**TP051932**

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

Introduction

This report documents the outcome of the proposed prototype of the GRANDPLEX Cinema Movie Ticket Inventory Management System developed with C++ to utilize efficient data structures and algorithms. The system is used by the cinema chain's staff to manage the ticket inventory for movies that are currently screening. Hence, it consists of two main components which are Movie and Purchase. Movie handles the ticket inventory of every movie and purchases allow the staff to log the purchases made by customers. It is important to note that the information for the inventory part is only storing the movie information and the ticket quantity available for each movie. The table below shows the features developed for each component of the system

Table 1 Feature List by Component

Movie	Purchase
Add Movie	Add Purchase
Display Movie	Display Purchase
Filter Movie by Genre	Insert Sort Purchase by Price
Search Movie by ID/ Title/Genre	Display Purchase Detail
Merge Sort Movie by Duration/Quantity/Price	
Filter by Genre	

Data Structures

Movie struct

```
struct Movie {

    int id;
    string title;
    string genre;
    string description;
    string voiceLanguage;
    string subtitleLanguage;
    string releaseDate;
    int minuteDuration;
    int ticketQuantity;
    float price;

    Movie* prev = NULL;
    Movie* next = NULL;

    ~Movie() {

        cout << "Movie node deleted succesfully!";
    }

}*newMovie, *currentMovie;
```

Figure 1: Source Code of Movie Struct

Figure above shows the structure of Movie node which include the attributes, pointers and a destructor, the struct is used instead of class is because it does not require any method and private attributes. The attributes of this struct is to store data related to the movie node which including id, title, genre, description, voice language, subtitle language, release data, minute duration, ticket quantity and price to demonstrate the inventory management functions. To create a doubly linked list for the purpose of higher efficiency on adding node and display algorithm, two pointers are required to store both previous node and the following node address which are assigned as NULL value as default. The destructor is used to display delete message when a node is deleted. Lastly, two global pointers are declared which are newMovie and currentMovie for the usage of methods.

MovieLinkedList class

```
class MovieLinkedList {

private:
    Movie* splitMovie(Movie* head) { ... }
    Movie* mergeMovie(Movie* firstHalf, Movie* secondHalf) { ... }

public:
    Movie* head, * tail;
    string sortMode, displayOrder;
    void addMovieToEnd() { ... }
    void displayMovie() { ... }

    void displayMovieBasedOnGenre(int idToDisplay, string titleToDisplay, string genreToDisplay, string descriptionToDisplay,
        string voiceToDisplay, string subtitleToDisplay, string dateToDisplay, int durationToDisplay,
        int quantityToDisplay, float priceToDisplay) { ... }

    void searchMovieById(int targetedId) { ... }

    void searchMovieByTitle(string targetedTitle) { ... }

    // Category filter
    void searchMovieBasedOnGenre(string targetedGenre) { ... }
    // Update
    void updateMovieAttributes(int idToUpdate, string titleToUpdate, string genreToUpdate, string descriptionToUpdate,
        string voiceToUpdate, string subtitleToUpdate, string dateToUpdate, int durationToUpdate,
        int quantityToUpdate, float priceToUpdate) { ... }

    void updateMovieAtIndex(int targetedId) { ... }

    Movie* sortMovie(Movie* firstLinkedListhead) { ... }
    // Delete
    void deleteMovieAtIndex(int targetedId) { ... }
};
MovieLinkedList movieLinkedList;
```

Figure 2: Source Code of MovieLinkedList Class

Figure above shows the MovieLinkesList class which include private methods, public variables and methods. There are two public pointers variable which will holding the address of the linked list first node and last node, so that the linked list is possible to travel in both direction and acive doubly linked list concept. The sortMode variable and two private methods will be only used by the sortMovie method where sortMode can be access anywhere to change the sort mode and private methods can only be called within the class. The displatOrder variable will be used by the displayMovie method to choose the order of the output. Lastly, the movieLinkedList is declared as a global variable of this class instant.

Purchase struct

```
struct Purchase {

    // Purchase attributes
    int id;
    int totalQuantity;

    string buyerName;
    string buyerContact;
    string datePurchased;
    string moviesPurchased;

    float price;

    Purchase* prev = NULL;
    Purchase* next = NULL;

    // destructor
    ~Purchase() {

        cout << "Purchase node deleted succesfully!";

    }

}*newPurchase, * currentPurchase, * nextPurchase; // ne
```

Figure 3: Source Code of Purchase struct

The figure above shows the struct created for a single node in the Purchase linked list. The struct contains the attributes for each purchase. The node also contains the pointers for the previous and next addresses of the adjacent nodes. This is to done to create a doubly linked list structure so that node traversal can go in both directions. Bi-directional traversal will allow for effective implementations of searching and sorting algorithms for the linked list. After the struct is declared, 3 nodes, newPurchase, currentPurchase and nextPurchase are created to allow for access and manipulation of the nodes. The newPurchase is used for inserting new nodes into the list, currentPurchase is in all algorithms that require node traversal and the nextPurchase node is used in the insertion sort algorithm.

Purchase LinkedList class

```
class PurchaseLinkedList {

public:
    Purchase* head, * tail;

    void addPurchaseToFront() { ... }
    void displayPurchase() { ... }
    void sortedInsert() { ... }
    void insertionSortPurchase(Purchase* unsorted) { ... }
    void displayPurchaseDetail(int searchId) { ... }
};
PurchaseLinkedList purchaseLinkedList, sortedPurchaseLinkedList;
```

Figure 4: Source Code of PurchaseLinkedList class

Figure above shows the code for the PurchaseLinkedList class which stores the head and tail pointers as well as functions for linked list operations. The head and tail pointers allow for access of the list from the front as well as the end. This reduces the time complexity for insertion and traversal as it is not required to traverse the whole node from the front to reach the end. The operations allow the users to add new data to the front of the linked list, display the nodes inside, sort it by price through insertion sort as well as view the details of a purchase. Two linked lists were created from the class for the unsorted list and the sorted list. This is done so that users can the data for the unsorted and sorted list are both maintained after the sorting has been performed.

Implementation

Menu

In Grandplex Movie Ticket Inventory management system, all the menus displayed are in structured way using different methods.

```
void printCenter(string text) {

    // Print the text in center
    int textLength = text.length();
    int oddOrEven = text.length() % 2;
    if (oddOrEven == 0) {
        cout << "|" << string(((59 - text.length()) / 2), ' ') << text << string(((58 - text.length()) / 2), ' ') << "|" << endl;
    }
    else {
        cout << "|" << string(((58 - textLength) / 2), ' ') << text << string(((59 - textLength) / 2), ' ') << "|" << endl;
    }
}
```

Figure 5 Source code of showing menu text in the center

To display menu in a structured way, the title of menu will be positioned at the center. To align the text in the center, the length of text will be calculated and then used to position the text.

```
void printMenuOptions(string text) {

    // Print the options of each menu in a structured way
    int textLength = text.length();
    int oddOrEven = text.length() % 2;
    if (oddOrEven == 0) {
        cout << "|" << string(20, ' ') << text << string(((38 - textLength)), ' ') << "|" << endl;
    }
    else {
        cout << "|" << string(20, ' ') << text << string(((38 - textLength)), ' ') << "|" << endl;
    }
}
```

Figure 6 Source code of showing menu options in structured manners

For the options of menus, all of them will be aligned at the same starting point. Similar to the previous way, the length of the text will be calculated to determine the spaces that required to be filled before drawing the box.

Main Menu

```
void mainMenu() {

    // Declare the variables for main menu
    string mainMenu = "Main Menu";
    string optionM1 = "1 --> Movie Transaction";
    string optionM2 = "2 --> Transaction";
    string optionM3 = "3 --> Exit";

    // Display the main menu by calling those functions
    drawLine();
    emptySpace();
    printCenter(mainMenu);
    emptySpace();
    drawLine();
    emptySpace();
    printMenuOptions(optionM1);
    printMenuOptions(optionM2);
    printMenuOptions(optionM3);
    emptySpace();
    drawLine();
    cout << endl;
}
```

Figure 7 Source code of displaying main menu

To print a consistent menu, each lines of the main menu will be printed by calling the functions that explained above. Same method will be applied for movie inventory menu and transaction menu as displayed in Figure 8 and Figure 9.

Movie Inventory Menu

```
void movieInventoryMenu() {

    // Declare the variables for movie inventory menu
    string movieInventoryMenu = "Movie Inventory Menu";
    string optionI1 = "1 --> Add Movie";
    string optionI2 = "2 --> Display Movie";
    string optionI3 = "3 --> Search Movie";
    string optionI4 = "4 --> Category Filter";
    string optionI5 = "5 --> Update Movie";
    string optionI6 = "6 --> Sort Movie";
    string optionI7 = "7 --> Delete Movie";
    string optionI8 = "8 --> Back";
    string optionI9 = "9 --> Exit";

    // Display the movie inventory menu by calling those functions
    drawLine();
    emptySpace();
    printCenter(movieInventoryMenu);
    emptySpace();
    drawLine();
    emptySpace();
    printMenuOptions(optionI1);
    printMenuOptions(optionI2);
    printMenuOptions(optionI3);
    printMenuOptions(optionI4);
    printMenuOptions(optionI5);
    printMenuOptions(optionI6);
    printMenuOptions(optionI7);
    printMenuOptions(optionI8);
    printMenuOptions(optionI9);
    emptySpace();
    drawLine();
    cout << endl;
}
```

Figure 8 Source code of displaying movie inventory menu

Transaction Menu

```
void transactionMenu() {

    // Declare the variables for transaction menu
    string transactionMenu = "Transaction Menu";
    string optionT1 = "1 --> Add Purchase";
    string optionT2 = "2 --> View Purchase";
    string optionT3 = "3 --> Sort Purchase";
    string optionT4 = "4 --> Purchase Details";
    string optionT5 = "5 --> Back";
    string optionT6 = "6 --> Exit";

    // Display the movie inventory menu by calling those functions
    drawLine();
    emptySpace();
    printCenter(transactionMenu);
    emptySpace();
    drawLine();
    emptySpace();
    printMenuOptions(optionT1);
    printMenuOptions(optionT2);
    printMenuOptions(optionT3);
    printMenuOptions(optionT4);
    printMenuOptions(optionT5);
    printMenuOptions(optionT6);
    emptySpace();
    drawLine();
    cout << endl;
}
```

Figure 9 Source code of transaction menu

Movie

Add Movie

```
case 1:

    movieLinkedList.displayMovie();

    do {

        // Read new movie data
        newMovie = new Movie;
        cout << "Enter Movie ID: ";
        cin >> newMovie->id;
        cout << "Enter Movie Title: ";
        cin.ignore();
        getline(cin, newMovie->title);
        cout << "Enter Movie Genre: ";
        getline(cin, newMovie->genre);
        cout << "Enter Movie Description: ";
        getline(cin, newMovie->description);
        cout << "Enter Movie Voice Language: ";
        getline(cin, newMovie->voiceLanguage);
        cout << "Enter Movie Subtitle Language: ";
        getline(cin, newMovie->subtitleLanguage);
        cout << "Enter Movie Release Date: ";
        getline(cin, newMovie->releaseDate);
        cout << "Enter Movie Duration(minutes): ";
        cin >> newMovie->minuteDuration;
        cout << "Enter Movie Ticket Quantity: ";
        cin >> newMovie->ticketQuantity;
        cout << "Enter Movie Price: ";
        cin >> newMovie->price;

        // Add new movie to the end
        movieLinkedList.addMovieToEnd();

        cout << "Do you want to insert another movie? (Y/N): ";
        cin >> decision;
        system("cls");
    } while (decision == "y" || decision == "Y");

    movieInventoryMenu();
    goto movieInventoryMenu;
    break;
```

Figure 10: Source Code of Add Movie Option

After the add movie option is being selected by the user on the Movie Inventory Menu, displayMovie method will be called first to let the user know which movie is already exist in the list to avoid duplication. Then, the user will be going into a do while loop where user will be kept

on asked the decision on whether another movie is needed to be added until the user refuse to. Every iteration of the loop will assign the newMovie variable as a new Movie node then request the user to input the attributes for the node. Lastly, addMovieToEnd method will be called to add the newMovie node to the linked list.

```
void addMovieToEnd() {

    // Scenario 1: At least one node in the linked list
    if (head != NULL) {

        // Assign the address
        tail->next = newMovie;
        newMovie->prev = tail;
        tail = newMovie;
    }
    // Scenario 2: The linked list is empty
    else {

        head = newMovie;
        tail = newMovie;
    }
}
```

Figure 11: Source Code of Add Movie

The addMovieToEnd method will do separate task for two different scenarios where first scenario is there is at least one node existing in the linked list and another scenario is the linked list is empty. The scenario for the linked list will be determined by using the head value as indicator. If the head is having NULL value, then it means the linked is empty while head and tail will be assigned as newMovie, else the newMovie will be the new tail while making sure the new tail and the previous tail have proper linking address for prev and next.

Display Movie

```
case 2:

    cout << "Display Order" << endl;
    cout << "1 --> Ascending" << endl;
    cout << "2 --> Descending" << endl;

    cout << "Enter Display Order: ";
    cin >> option;

    do {
        switch (option) {

        case 1: movieLinkedList.displayOrder = "Asc"; break;
        case 2: movieLinkedList.displayOrder = "Desc"; break;
        default: invalid(); break;
        }
    } while (option < 1 && option > 3);

    // Display all movie in the linked list
    movieLinkedList.displayMovie();

    clearScreen();
    movieInventoryMenu();
    goto movieInventoryMenu;
    break;
```

Figure 12: Source Code of Display Movie Option

After the display movie option is being selected by the user on the Movie Inventory Menu, the display order will be request from the user with validation checking until a valid option is chosen. Then the displayOrder variable will be changed according to the option. Lastly, displayMovie method will be called to display all the node within the linked list.

```
void displayMovie() {

    // Read node by node from head or tail and cout the attributes
    if (displayOrder == "Asc") {

        currentMovie = head;
    }
    else if (displayOrder == "Desc") {

        currentMovie = tail;
    }
    while (currentMovie != NULL) {

        cout << "ID:\t\t\t" << currentMovie->id << endl
            << "Title:\t\t\t" << currentMovie->title << endl
            << "Genre:\t\t\t" << currentMovie->genre << endl
            << "Description:\t\t" << currentMovie->description << endl
            << "Voice Language:\t\t" << currentMovie->voiceLanguage << endl
            << "Subtitle Language:\t" << currentMovie->subtitleLanguage << endl
            << "Release Date:\t\t" << currentMovie->releaseDate << endl
            << "Duration(minutes):\t" << currentMovie->minuteDuration << endl
            << "Ticket Quantity:\t" << currentMovie->ticketQuantity << endl
            << "Price:\t\t\t" << currentMovie->price << endl << endl;

        if (displayOrder == "Asc") {

            currentMovie = currentMovie->next;
        }
        else if (displayOrder == "Desc") {

            currentMovie = currentMovie->prev;
        }
    }
}
```

Figure 13: Source Code of Display Movie

The displayMovie method will be using the displayOrder variable to determine the display order. Firstly, the currentMovie variable will be assigned with head or tail depend on the displayOrder string, then a while loop will be performed by checking if the currentMovie value is NULL or not. If the temp is not NULL, then the currentMovie attributes value will be displayed using organized format and assign currentMovie to be the next node or previous node depend on the displayOrder string after displaying it until there is no node left.

Filter Movie by Genre

```
// Category filter
void searchMovieBasedOnGenre(string targetedGenre) {
    currentMovie = head;
    bool existed = false;

    // If the list is not empty
    while (currentMovie != NULL) {

        // If the genre keyword inputted match with list
        if (currentMovie->genre.find(targetedGenre) != string::npos) {

            displayMovieBasedOnGenre(currentMovie->id,
                currentMovie->title,
                currentMovie->genre,
                currentMovie->description,
                currentMovie->voiceLanguage,
                currentMovie->subtitleLanguage,
                currentMovie->releaseDate,
                currentMovie->minuteDuration,
                currentMovie->ticketQuantity,
                currentMovie->price);
            existed = true;
        }
        currentMovie = currentMovie->next;
    }

    if (existed == false) {
        cout << "Movie with genre (" << targetedGenre << ") is not found!" << endl;
    }
}
```

Figure 14 Source Code of Filter Movie By Genre

One of the function available in this system is filter the movies based on genre. Before that, user will be prompted to enter the keyword of genre. Under the condition when the list is not empty, all the information of the movies with matched genre will be displayed. If there is movie matched, then the variable which named as existed will be assigned with true, else it will remain as false. If existed variable equals to false, it indicates that the movie with the genre keyword inputted are not found in the list. Thus, a movie not found message will be displayed along with genre keyword inputted to inform the user.

```
void displayMovieBasedOnGenre(int idToDisplay, string titleToDisplay, string genreToDisplay, string descriptionToDisplay,
    string voiceToDisplay, string subtitleToDisplay, string dateToDisplay, int durationToDisplay,
    int quantityToDisplay, float priceToDisplay) {
    // Display the searched genre result
    cout << "Genre:\t\t\t" << genreToDisplay << endl
        << "ID:\t\t\t" << idToDisplay << endl
        << "Title:\t\t\t" << titleToDisplay << endl
        << "Description:\t\t" << descriptionToDisplay << endl
        << "Voice Language:\t\t" << voiceToDisplay << endl
        << "Subtitle Language:\t" << subtitleToDisplay << endl
        << "Release Date:\t\t" << dateToDisplay << endl
        << "Duration(minutes):\t" << durationToDisplay << endl
        << "Ticket Quantity:\t" << quantityToDisplay << endl
        << "Price:\t\t\t" << priceToDisplay << endl << endl;

}
```

Figure 15 Source Code of Display Movie By Genre

After the previous function filtered the movie, this function will used to print out all the information of the movie in such sequence which are genre, id, title, description, voice language, subtitle language, duration, seat quantity and price.

<u>Search Movie by ID/ Title/Genre</u>

```
case 3:

    do {

        // Get targeted title
        cout << "Enter Targeted Movie Title: ";
        cin >> targetedTitle;

        // Search from linked list to find matching title and display them
        movieLinkedList.searchMovieByTitle(targetedTitle);

        cout << "Do you want to search for another movie? (Y/N): ";
        cin >> decision;
        system("cls");
    } while (decision == "y" || decision == "Y");

    system("cls");
    movieInventoryMenu();
    goto movieInventoryMenu;
    break;
```

Figure 16: Source code of Search Movie By Title Option

After the search movie option is being selected by the user on the Movie Inventory Menu, a targeted title will be request from the users first, then the targeted title will be used to call the

searchMovieByTitle method. Lastly, the search procedure will keep on repeating until the user choose the decision to stop searching movie in a do while loop.

```
void searchMovieByTitle(string targetedTitle) {

    // Read node by node from head and cout the attributes if the title match the targeted title
    currentMovie = head;
    while (currentMovie != NULL) {

        if (currentMovie->title.find(targetedTitle) != string::npos) {

            cout << "ID:\t\t\t" << currentMovie->id << endl
                << "Title:\t\t\t" << currentMovie->title << endl
                << "Genre:\t\t\t" << currentMovie->genre << endl
                << "Description:\t\t" << currentMovie->description << endl
                << "Voice Language:\t\t" << currentMovie->voiceLanguage << endl
                << "Subtitle Language:\t" << currentMovie->subtitleLanguage << endl
                << "Release Date:\t\t" << currentMovie->releaseDate << endl
                << "Duration(minutes):\t" << currentMovie->minuteDuration << endl
                << "Ticket Quantity:\t" << currentMovie->ticketQuantity << endl
                << "Price:\t\t\t" << currentMovie->price << endl << endl;
        }
        currentMovie = currentMovie->next;
    }
}
```

Figure 17: Source Code of Search Movie By Title

The searchMovieByTitle method starts from assigning currentMovie variable as the current head node. The while loop will be executed if the currentMovie is not NULL, to try to find the targeted title within the title node by node. If the node has the targeted title in the title, then the currentMovie attributes value will be displayed using organized format. Lastly, the while loop continues by assigning next node for currentMovie until the end to return to the main function.

Merge Sort Movie by Duration/Quantity/Price

```
case 6:

    cout << "Sort Movie" << endl;
    cout << "1 --> Sort By Duration" << endl;
    cout << "2 --> Sort By Quantity" << endl;
    cout << "3 --> Sort By Price" << endl;

    cout << "Enter Sort Mode: ";
    cin >> option;

    do {
        switch (option) {

        case 1: movieLinkedList.sortMode = "Duration"; break;
        case 2: movieLinkedList.sortMode = "Quantity"; break;
        case 3: movieLinkedList.sortMode = "Price"; break;
        default: invalid(); break;
        }
    } while (option < 1 && option > 3);

    // Sort movie by ticket quantity
    movieLinkedList.head = movieLinkedList.sortMovie(movieLinkedList.head);
    cout << "Linked List Sorted!" << endl;
    clearScreen();

    movieInventoryMenu();
    goto movieInventoryMenu;
    break;
```

Figure 18: Source Code of Sort Movie Option

After the sort movie option is being selected by the user on the Movie Inventory Menu, the sort mode will be request from the user with validation checking until a valid option is chosen. Then the sortMode variable will be changed according to the option. Lastly, sortMovie method will be called to sort the linked list using the given mode and current linked list head node.

```
Movie* sortMovie(Movie* firstLinkedListhead) {

    if (!firstLinkedListhead || !firstLinkedListhead->next) {
        // if the linked list is too small that no need to sort
        return firstLinkedListhead;
    }

    // Split the linked list to small linked list
    Movie* secondLinkedListhead = splitMovie(firstLinkedListhead);

    // Recursive for first and second linked list
    firstLinkedListhead = sortMovie(firstLinkedListhead);
    secondLinkedListhead = sortMovie(secondLinkedListhead);

    // Merge while sort the two linked list and return back to the called function
    return mergeMovie(firstLinkedListhead, secondLinkedListhead);
}
```

Figure 19: Source Code of sortMovie Method

The algorithm will be used when sorting Movie is merge sort because it has significant less time complexity compared to common sorting algorithm such as insertion sort and bubble sort. Hence recursive function will be widely used because it is easier to assign head node for every split of nodes while it is also an advantage over time complexity. The procedure starts from the main merge sort function, sortMovie called. It will keep on split the node until it is only one single node for every head using splitMovie method, then compare and merge the nodes together using mergeMovie method, this process will be done recursively.

```
Movie* splitMovie(Movie* head) {

    Movie* fastNode = head, * slowNode = head;

    // fast node travel twice as fast as slow node to find the middle node of the linked list
    while (fastNode->next && fastNode->next->next)
    {
        fastNode = fastNode->next->next;
        slowNode = slowNode->next;
    }

    // pass data to another variable to avoid memory error
    Movie* secondHalf = slowNode->next;
    slowNode->next = NULL;

    return secondHalf;
}
```

Figure 20: Source Code of spiltMovie Method

The splitMovie method is used for finding the midpoint of the linked list to split it to two halves. Since there is no midpoint predefined, one node will travel twice as fast as another, by the time it reaches the end or not able to travel forward anymore meaning that the slower node has reached the midpoint.

```
Movie* mergeMovie(Movie* firstHalf, Movie* secondHalf) {

    // If first linked list is empty
    if (!firstHalf)
        return secondHalf;

    // If second linked list is empty
    if (!secondHalf)
        return firstHalf;

    // Smaller value will be put on the front
    if (sortMode == "Duration") {

        if (firstHalf->minuteDuration < secondHalf->minuteDuration)
        {
            firstHalf->next = mergeMovie(firstHalf->next, secondHalf);
            firstHalf->next->prev = firstHalf;
            firstHalf->prev = NULL;
            return firstHalf;
        }
        else
        {
            secondHalf->next = mergeMovie(firstHalf, secondHalf->next);
            secondHalf->next->prev = secondHalf;
            secondHalf->prev = NULL;
            return secondHalf;
        }
    }
    else if (sortMode == "Quantity") {

        if (firstHalf->ticketQuantity < secondHalf->ticketQuantity)
        {
            firstHalf->next = mergeMovie(firstHalf->next, secondHalf);
            firstHalf->next->prev = firstHalf;
            firstHalf->prev = NULL;
            return firstHalf;
        }
        else
        {
            secondHalf->next = mergeMovie(firstHalf, secondHalf->next);
            secondHalf->next->prev = secondHalf;
            secondHalf->prev = NULL;
            return secondHalf;
        }
    }
    else if (sortMode == "Price") {

        if (firstHalf->price < secondHalf->price)
        {
            firstHalf->next = mergeMovie(firstHalf->next, secondHalf);
            firstHalf->next->prev = firstHalf;
            firstHalf->prev = NULL;
            return firstHalf;
        }
        else
        {
            secondHalf->next = mergeMovie(firstHalf, secondHalf->next);
            secondHalf->next->prev = secondHalf;
            secondHalf->prev = NULL;
            return secondHalf;
        }
    }
}
```

Figure 21: Source Code of mergeMovie Method

The mergeMovie method is to find the node that has lower duration, quantity or price based on the sortMode variable and put into the front. However, sometime the node also carries another

node behind, therefore the merge process will be done recursively to sort out the correct ascending arrangement.

Update Movie

```
void updateMovieAtIndex(int targetedId) {
    currentMovie = head;
    bool existed = false;

    // If the list is not empty
    while (currentMovie != NULL) {

        // If the id match with one of the id in the list
        if (currentMovie->id == targetedId) {
            cout << "Update Movie with ID (" << targetedId << "): \n\n" << endl;

            cout << "ID:\t\t\t" << currentMovie->id << endl
                << "Title:\t\t\t" << currentMovie->title << endl
                << "Genre:\t\t\t" << currentMovie->genre << endl
                << "Description:\t\t" << currentMovie->description << endl
                << "Voice Language:\t\t" << currentMovie->voiceLanguage << endl
                << "Subtitle Language:\t" << currentMovie->subtitleLanguage << endl
                << "Release Date:\t\t" << currentMovie->releaseDate << endl
                << "Duration(minutes):\t" << currentMovie->minuteDuration << endl
                << "Ticket Quantity:\t" << currentMovie->ticketQuantity << endl
                << "Price:\t\t\t" << currentMovie->price << endl << endl;
```

Figure 22 Source Code of Update Movie

For the function of updating movie, the existing movie list will be displayed first for the user to make decision on which movie is tended to be updated. Next, user will be asked to enter the id number of the movie. If the movie list is not equal to NULL and the movie id entered is found within the movie list, the information of the particular movie will be displayed again.

```
// Read the new information to update
cout << "Enter New Movie Title: ";
cin >> currentMovie->title;
cout << "Enter New Movie Genre: ";
cin >> currentMovie->genre;
cout << "Enter New Movie Description: ";
cin >> currentMovie->description;
cout << "Enter New Movie Voice Language: ";
cin >> currentMovie->voiceLanguage;
cout << "Enter New Movie Subtitle Language: ";
cin >> currentMovie->subtitleLanguage;
cout << "Enter New Movie Release Date: ";
cin >> currentMovie->releaseDate;
cout << "Enter New Movie Duration(minutes): ";
cin >> currentMovie->minuteDuration;
cout << "Enter New Movie Ticket Quantity: ";
cin >> currentMovie->ticketQuantity;
cout << "Enter New Movie Price: ";
cin >> currentMovie->price;
```

Figure 23 Source Code of Update Movie

Then, the system will prompt the user to enter the new information for the movie displayed. The new information includes, movie title, genre, description, voice language, subtitle language, release date, movie duration, ticket quantity and movie price but not include movie id. Movie id is unchangeable and will remain the same although other information has been updated.

```
            // Update the attributes of movie
            updateMovieAttributes(currentMovie->id,
                currentMovie->title,
                currentMovie->genre,
                currentMovie->description,
                currentMovie->voiceLanguage,
                currentMovie->subtitleLanguage,
                currentMovie->releaseDate,
                currentMovie->minuteDuration,
                currentMovie->ticketQuantity,
                currentMovie->price);

            existed = true;
        }
        currentMovie = currentMovie->next;
    }

    if (existed == false) {
        cout << "No Movie With Id (" << targetedId << ") is found. Please try again! " << endl;
    }
}
```

Figure 24 Source Code of Update Movie

The updateMovieAttributes function as displayed in Figure 21 will be called to manipulate the movie attributes after the new information have been inputted. Similar to the filter movie based on genre function, if the movie is found within the movie list, then the variable named as existed will be assigned with true. Otherwise, a movie not found message will be shown along with the movie id inputted.

```
// Update
void updateMovieAttributes(int idToUpdate, string titleToUpdate, string genreToUpdate, string descriptionToUpdate,
    string voiceToUpdate, string subtitleToUpdate, string dateToUpdate, int durationToUpdate,
    int quantityToUpdate, float priceToUpdate) {
    // Get the user input and replace with the old data
    Movie* temp = head;
    idToUpdate = temp->id;
    titleToUpdate = temp->title;
    genreToUpdate = temp->genre;
    descriptionToUpdate = temp->description;
    voiceToUpdate = temp->voiceLanguage;
    subtitleToUpdate = temp->subtitleLanguage;
    dateToUpdate = temp->releaseDate;
    durationToUpdate = temp->minuteDuration;
    quantityToUpdate = temp->ticketQuantity;
    priceToUpdate = temp->price;
}
```

Figure 25 Source Code of Update Movie Attributes

After this function is being called, it will accept the new information for movie attributes in original sequences.

Delete Movie

```
// Delete
void deleteMovieAtIndex(int targetedId) {

    // Cannot delete as list is empty
    if (head == NULL) {
        cout << "No movie available to be deleted" << endl;
        return;
    }
    else if (head->id == targetedId) {
        // Delete first node when list is not empty
        currentMovie = head;
        head = head->next;
        cout << "Movie with title (" << currentMovie->title << ") is deleted!" << endl;
        delete currentMovie;
    }
    else {
        // Delete the node in other location except first node
        currentMovie = head;
        Movie* prev = NULL;

        // If the list is not empty
        while (currentMovie != NULL) {
            if (currentMovie->id == targetedId) {
                prev->next = currentMovie->next;
                cout << "Movie with title (" << currentMovie->title << ") is deleted!" << endl;
                delete currentMovie;
                return;
            }
            prev = currentMovie;
            currentMovie = currentMovie->next;
        }
        cout << "Movie with id (" << targetedId << ") is not found!" << endl;
    }
}
```

Figure 26 Source Code of Delete Movie

In the deleteMovieAtIndex finction, the user required to specify the movie id that tends to be deleted. There might be three scenario. The first scenario is no movie can be deleted as the movie list is empty. For second scenario, the id inputted is at first location and the list is not empty, therefore, the first object will be deleted. The last scenario is the node is located in other location except the first location. A while loop will be implemented to find the movie which matched with the id inputted and then delete it. If delete successfully, a message regarding to movie deleted successfully will be displayed.

Purchase

Add Purchase

```
case 1:

    do {

        // Input new Purchase data
        newPurchase = new Purchase;
        newPurchase->id = purchaseLinkedList.head->id + 1;
        cin.ignore();
        cout << "Enter Buyer Name: ";
        getline(cin, newPurchase->buyerName);
        cout << "Enter Buyer Contact: ";
        getline(cin, newPurchase->buyerContact);
        cout << "Enter Purchase Date (DD/MM/YYYY): ";
        getline(cin, newPurchase->datePurchased);
```

Figure 27 Source Code of Purchase Input (1)

```
    cout << "Available movies: \n\n " << endl;
    movieLinkedList.displayMovie();
    string movieIds = "";
    string movieId = "";
    int count = 0;
    newPurchase->price = 0;

    // loop to allow user to enter multiple movies
    do {

        cout << "Enter Movie Id : ";
        cin.ignore();
        getline(cin, movieId);

        // convert id from string to integer
        movieIds = movieIds + movieId + ";";
        ss << movieId;
        ss >> intId;

        // traverse movie linked list
        currentMovie = movieLinkedList.head;
        while (currentMovie != NULL) {
            // if found, add price to purchase total price, decrement ticket quantity by 1
            if (currentMovie->id == intId) {

                newPurchase->price += currentMovie->price;
                currentMovie->ticketQuantity -= 1;
                break;
            }
            currentMovie = currentMovie->next;
        }

        // count for total quantity
        count++;

        cout << "Do you want to insert another movie? (Y/N): ";
        cin >> decision;
    } while (decision == "y" || decision == "Y");
    newPurchase->moviesPurchased = movieIds;
    newPurchase->totalQuantity = count;

    // Add new purchase to linked list
    purchaseLinkedList.addPurchaseToFront();

    cout << "Do you want to insert another purchase? (Y/N): ";
    cin >> decision;
    cin.ignore();
    system("cls");
} while (decision == "y" || decision == "Y");
```

Figure 28 Source Code of Purchase Input (2)

The two figures above show the source code for the input of new Purchase data. The user is prompted to enter the movie id to insert 1 ticket quantity for a movie. The id input is used to identify whether the movie is stored in the linked list and the price and quantity data are manipulated and added to the purchase information. Once all the information has been assigned to the variables in the Purchase node, the function addPurchaseToFront is called.

```
void addPurchaseToFront() {

    // set current head address to the next address of the new node
    newPurchase->next = head;
    newPurchase->prev = NULL;
    // update head to new node
    head = newPurchase;
    // if tail is empty
    if (tail == NULL) {
        tail = newPurchase;
    }
    else {
        // link the next address to new node
        newPurchase->next->prev = newPurchase;
    }
}
```

Figure 29 Source Code of addPurchaseToFront()

The snippet above shows how a new Purchase node is added to the system. The new node's next pointer is set to the current head and then reassigned as the new head of the linked list. There is also a condition to assign the tail if it is empty. If it is not empty, the new node will link to the next node.

<u>Display Purchase</u>

```
void displayPurchase() {

    currentPurchase = head;

    // if current is not NULL, then display node attributes
    while (currentPurchase != NULL) {

        cout << "ID:\t\t\t" << currentPurchase->id << endl
            << "Customer Name:\t\t\t" << currentPurchase->buyerName << endl
            << "Customer Contact:\t\t\t" << currentPurchase->buyerContact << endl
            << "Date Purchased:\t\t" << currentPurchase->datePurchased << endl
            << "Total Quantity:\t\t" << currentPurchase->totalQuantity << endl
            << "Total Price:\t" << currentPurchase->price << endl << endl;

        // traverse to next node
        currentPurchase = currentPurchase->next;
    }
};
```

Figure 30 Source Code of displayPurchase()

This function is used to display all the purchase nodes in the console. A while loop is used to traverse through the whole list and each iteration lists out all attributes of a node.

Insertion Sort Purchase by Price

```
void sortedInsert() {

    // sort in ascending order
    // set node of unsorted list to new node
    newPurchase = currentPurchase;

    // if sorted list is empty
    if (head == NULL) {
        head = newPurchase;
    }
    // if price of new node is smaller than head, place new node infront of head
    else if (head->price >= newPurchase->price) {
        newPurchase->next = head;
        newPurchase->next->prev = newPurchase;
        head = newPurchase;
    }
    // if price of new node is more than price of head
    else {
        // set current node to head of sorted list
        currentPurchase = head;

        // while the next node is not NULL and
        // price of new node is greater than price of the node next to current
        while (currentPurchase->next != NULL && currentPurchase->next->price < newPurchase->price) {
            // go to next node
            currentPurchase = currentPurchase->next;
        }

        // after node has traversed to a node that it is larger than
        // set the next address of new node to the next address of the current node
        newPurchase->next = currentPurchase->next;

        // if node reached end of list
        if (currentPurchase->next != NULL) {
            newPurchase->next->prev = newPurchase;
        }

        currentPurchase->next = newPurchase;
        newPurchase->prev = currentPurchase;

    }

};
```

Figure 31 Source Code of sortedInsert()

The figure above shows the sorted insertion function which will be used in a loop of the insertionSortPurchase() function. The newPurchase and currentPurchase in the snippet represent a new node to be inserted in the sorted list and the current node in the unsorted list respectively. There are 3 main conditions to check when performing a sorted insertion. The first is to check if the sorted list is empty, then it will check if the head node's price is less than the current unsorted node. If it is less than it, it will proceed to the next condition, which is to traverse until the unsorted node's price value is greater than the next sorted node's price.

```
void insertionSortPurchase(Purchase* unsorted) {

    currentPurchase = unsorted;
    while (currentPurchase != NULL) {

        nextPurchase = currentPurchase->next;
        currentPurchase->prev = NULL;
        currentPurchase->next = NULL;
        sortedInsert();
        currentPurchase = nextPurchase;

    }

};
```

Figure 32 Source Code of insertionSortPurchase()

Figure above shows the insertionSortPurchase function which takes a Purchase* argument. This is used to insert the head of the unsorted list into the function, which will then be assigned into the currentPurchase node, where it will loop through all the values in the unsorted list and insert it into another linked list in a sorted way. In each iteration, the current node, the current node's next node is stored in the nextPurchase node, and the addresses of the adjacent nodes for the current node is set as NULL. Afterwards the sortedInsert() function is called and the next node is set as the current node.

Display Purchase Detail

```
void displayPurchaseDetail(int searchId) {

    currentPurchase = head;
    int intId;

    while (currentPurchase != NULL) {
        if (currentPurchase->id == searchId) {
            string movieIds = currentPurchase->moviesPurchased;
            string id = "";

            // loop through string until delimiter is found
            for (auto x : movieIds) {
                if (x == ';') {
                    // convert id of type string to int for insertion to node
                    ss << id;
                    ss >> intId;
                    movieLinkedList.searchMovieById(intId);
                }
                else {
                    // if current iteration is not a delimiter, concatenate string
                    id = id + x;
                }
            }
        }
        currentPurchase = currentPurchase->next;
    }
};
```

Figure 33 Source Code of displayPurchaseDetail()

The displayPurchaseDetail() function is used to show the details of the movies purchased in a single transaction. This function will traverse to the node with the ID input by the user. Once the node is found, the moviesPurchased string data is accessed and each character is looped through until a delimiter is found. If a delimiter was not found it will combine the characters, it has looped through to form a single ID. After a delimiter is found, the searchMovieById function is also called from the movieLinkedList object to retrieve the movie data.

```
void searchMovieById(int targetedId) {

    // Read node by node from head and cout the attributes for purchase detail if the id is equal to the targeted id
    currentMovie = head;
    while (currentMovie != NULL) {

        if (currentMovie->id == targetedId) {

            cout << "ID:\t\t\t" << currentMovie->id << endl
                << "Title:\t\t\t" << currentMovie->title << endl
                << "Price:\t" << currentMovie->price << endl << endl;
        }
        currentMovie = currentMovie->next;
    }
}
```

Figure 34 Source Code of searchMovieById

The snippet above shows the source code of the searchMovieById, it is similar to the other search functions found in the MovieLinkedList except for the attributes displayed.

Result

Main Menu

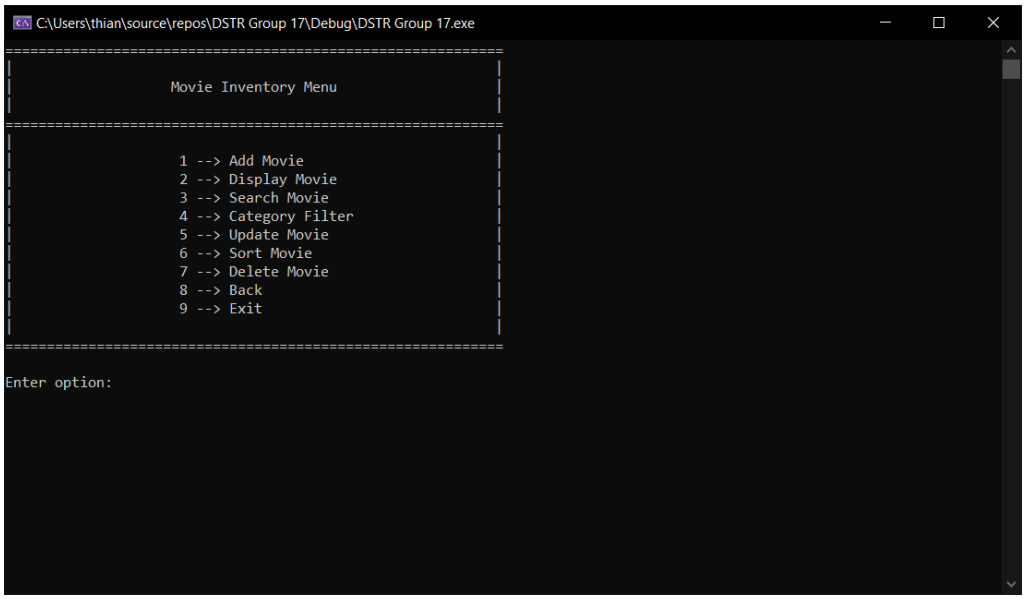
```
C:\Users\thian\source\repos\DSTR Group 17\Debug\DSTR Group 17.exe
=========================================================
|                                                       |
|                       Main Menu                       |
|                                                       |
=========================================================
|                                                       |
|                  1 --> Movie Transaction              |
|                  2 --> Transaction                    |
|                  3 --> Exit                           |
|                                                       |
=========================================================

Enter option:
```

Figure 35 Output of Main Menu

As displayed in Figure 35, a main menu is displayed in a structured alignment. The title of the menu will be shown in the middle of the menu box. Meanwhile, the menu options will located at the same starting points. The options available in this main menu are movie inventory and transaction. When the user select the option based on the number, then he user will be redirect to the relevant page.

Movie Inventory

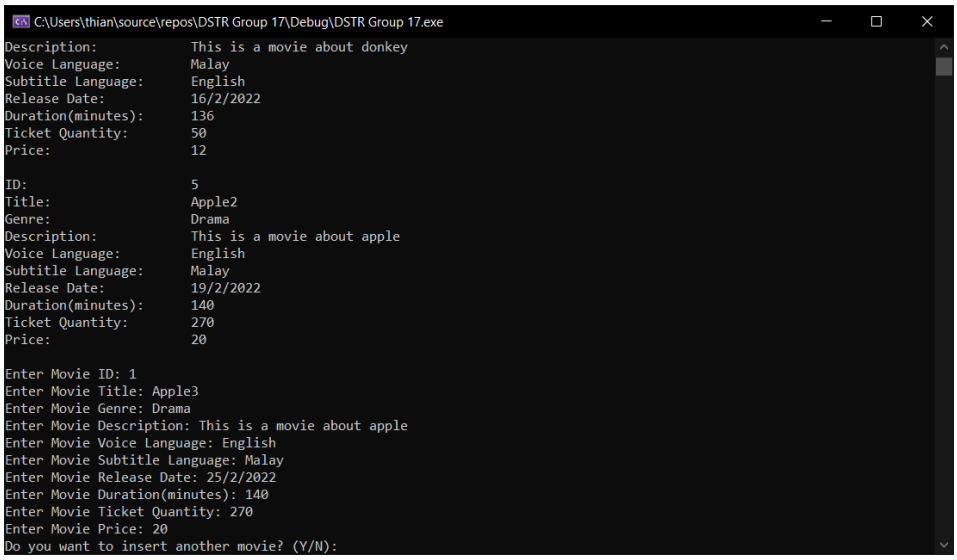


```
C:\Users\thian\source\repos\DSTR Group 17\Debug\DSTR Group 17.exe
=========================================================
|                                                       |
|                  Movie Inventory Menu                 |
|                                                       |
=========================================================
|                                                       |
|                    1 --> Add Movie                    |
|                    2 --> Display Movie                |
|                    3 --> Search Movie                 |
|                    4 --> Category Filter              |
|                    5 --> Update Movie                 |
|                    6 --> Sort Movie                   |
|                    7 --> Delete Movie                 |
|                    8 --> Back                         |
|                    9 --> Exit                         |
|                                                       |
=========================================================

Enter option:
```

Figure 36 Output of Movie Inventory Menu

Similar to the main menu, the movie inventory menu is displayed in a proper and structured manner. In the movie inventory menu, there are nine options to be selected which are add movie, display movie, search movie, category filter, update movie, sort movie, delete movie, back and exit. The system will prompt user to select an option to proceed.



```
C:\Users\thian\source\repos\DSTR Group 17\Debug\DSTR Group 17.exe
Description:            This is a movie about donkey
Voice Language:         Malay
Subtitle Language:      English
Release Date:           16/2/2022
Duration(minutes):      136
Ticket Quantity:        50
Price:                  12

ID:                     5
Title:                  Apple2
Genre:                  Drama
Description:            This is a movie about apple
Voice Language:         English
Subtitle Language:      Malay
Release Date:           19/2/2022
Duration(minutes):      140
Ticket Quantity:        270
Price:                  20

Enter Movie ID: 1
Enter Movie Title: Apple3
Enter Movie Genre: Drama
Enter Movie Description: This is a movie about apple
Enter Movie Voice Language: English
Enter Movie Subtitle Language: Malay
Enter Movie Release Date: 25/2/2022
Enter Movie Duration(minutes): 140
Enter Movie Ticket Quantity: 270
Enter Movie Price: 20
Do you want to insert another movie? (Y/N):
```

Figure 37 Output of Add New Movie

In Figure 37, it shows that the correct ways to add a new movie. First of all, the existed movie list will be displayed. Users will then prompted to input the information for the new movie in such sequence which are id, title, genre, description, voice language, subtitle language, release date, ticket quantity and price. After successfully adding the new movie, the system will ask the user whether like to continue to add new movie or not.

```
Display Order
1 --> Ascending
2 --> Descending
Enter Display Order: 1
ID:                     1
Title:                  Apple
Genre:                  Drama
Description:            This is a movie about apple
Voice Language:         English
Subtitle Language:      Malay
Release Date:           1/2/2022
Duration(minutes):      120
Ticket Quantity:        250
Price:                  15

ID:                     2
Title:                  Boy
Genre:                  Adventure
Description:            This is a movie about boy
Voice Language:         Mandarin
Subtitle Language:      Malay
Release Date:           7/2/2022
Duration(minutes):      124
Ticket Quantity:        150
Price:                  20

ID:                     3
Title:                  Cat
Genre:                  Action
Description:            This is a movie about cat
Voice Language:         Malay
Subtitle Language:      English
Release Date:           5/2/2022
Duration(minutes):      157
Ticket Quantity:        100
Price:                  35

ID:                     4
Title:                  Donkey
Genre:                  Drama
Description:            This is a movie about donkey
Voice Language:         Malay
Subtitle Language:      English
Release Date:           16/2/2022
Duration(minutes):      136
Ticket Quantity:        50
Price:                  12

ID:                     5
Title:                  Apple2
```

Figure 38 Output of Display Movie

After the display movie option is being selected by the user on the Movie Inventory Menu, the display order is chosen to be ascending then the whole linked list will be displayed node by node from head until tail. Lastly, a system pause will be called until user enter a key to back to menu.

```
C:\Users\Sia De Long\Desktop\XiiaoVampireZ\VisualProjects\DSTR Lab Week 1\Debug\DSTR Lab.exe
Enter Targeted Movie Title: Apple
ID:                     1
Title:                  Apple
Genre:                  Drama
Description:            This is a movie about apple
Voice Language:         English
Subtitle Language:      Malay
Release Date:           1/2/2022
Duration(minutes):      120
Ticket Quantity:        250
Price:                  15

ID:                     5
Title:                  Apple2
Genre:                  Drama
Description:            This is a movie about apple
Voice Language:         English
Subtitle Language:      Malay
Release Date:           19/2/2022
Duration(minutes):      140
Ticket Quantity:        270
Price:                  20

Do you want to search for another movie? (Y/N):
```

Figure 39 Output of Search Movie

After the search movie option is being selected by the user on the Movie Inventory Menu, a targeted title “Apple” is given to the system, then all the node which have the title includes the targeted title will be display accordingly. The decision of next search will then be asked until user input ‘N’. Lastly, a system pause will be called until user enter a key to back to menu.

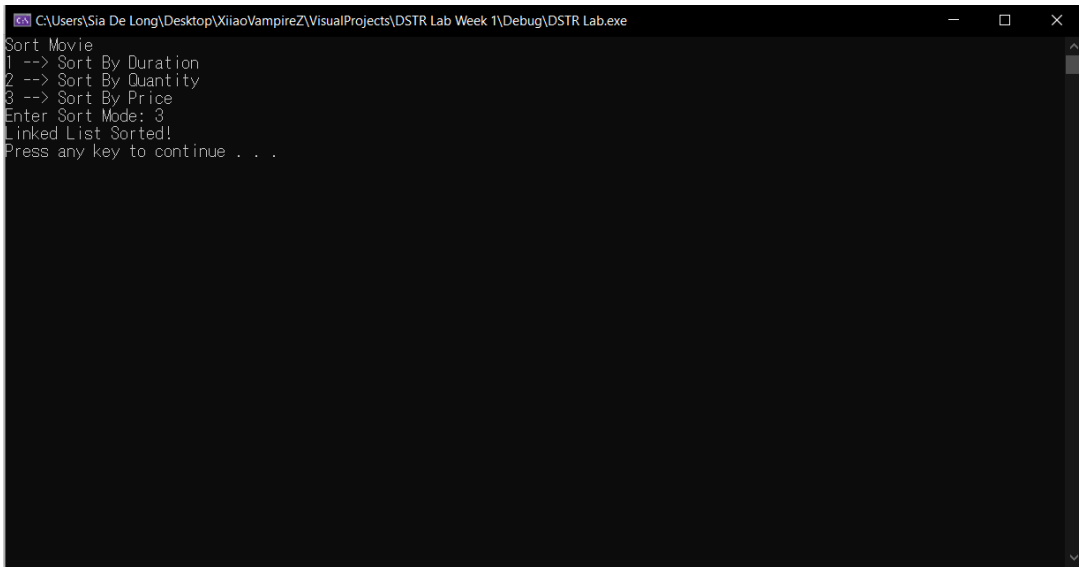


Figure 40 Output of Sort Movie

After the sort movie option is being selected by the user on the Movie Inventory Menu, the sort mode of price is chosen. Then the linked list will be sorted ascendingly by the price attribute from the nodes. Lastly, a system pause will be called until user enter a key to back to menu.

Transaction

```
C:\Users\thian\source\repos\DSTR Group 17\Debug\DSTR Group 17.exe
============================================================
|                                                          |
|                     Transaction Menu                     |
|                                                          |
============================================================
|                                                          |
|                   1 --> Add Purchase                     |
|                   2 --> View Purchase                    |
|                   3 --> Sort Purchase                    |
|                   4 --> Purchase Details                 |
|                   5 --> Back                             |
|                   6 --> Exit                             |
|                                                          |
============================================================

Enter option:
```

Figure 41 Output of Transaction Menu

The transaction menu shows all the functions that the user can perform in purchases. The options available are add purchase, view purchase, sort purchase, purchase detail, back and exit.

```
C:\Users\thian\source\repos\DSTR Group 17\Debug\DSTR Group 17.exe
Enter Buyer Name: John Doe
Enter Buyer Contact: 0123055032
Enter Purchase Date (DD/MM/YYYY): 25/02/2022
Available movies:


ID:                     1
Title:                  Apple
Genre:                  Drama
Description:            This is a movie about apple
Voice Language:         English
Subtitle Language:      Malay
Release Date:           1/2/2022
Duration(minutes):      120
Ticket Quantity:        250
Price:                  15

ID:                     2
Title:                  Boy
Genre:                  Adventure
Description:            This is a movie about boy
Voice Language:         Mandarin
Subtitle Language:      Malay
Release Date:           7/2/2022
Duration(minutes):      124
Ticket Quantity:        150
Price:                  20

ID:                     3
Title:                  Cat
```

Figure 42 Output of Add Purchase

In the beginning, the system will prompt the user to enter their personal information such as name, contact and purchase date. The list of movies will then be displayed for the users to select the movie.

```
C:\Users\thian\source\repos\DSTR_GROUP17\Debug\DSTR_GROUP17.exe

ID:                     4
Customer Name:          Thian Shan You
Customer Contact:       0135094199
Date Purchased:         23/02/2022
Total Quantity:         4
Total Price:    82

ID:                     3
Customer Name:          Sia De Long
Customer Contact:       0122654199
Date Purchased:         22/02/2022
Total Quantity:         3
Total Price:    70

ID:                     2
Customer Name:          Beh Chi Hao
Customer Contact:       0122094239
Date Purchased:         21/02/2022
Total Quantity:         2
Total Price:    35

ID:                     1
Customer Name:          Thian Shan You
Customer Contact:       0122095199
Date Purchased:         20/02/2022
Total Quantity:         5
Total Price:    102

Press any key to continue . . .
```

Figure 43 Output of Display Purchase

The display purchase page shows a list of previous purchases in no sort order.

```
C:\Users\thian\source\repos\DSTR_GROUP17\Debug\DSTR_GROUP17.exe

ID:                     2
Customer Name:          Beh Chi Hao
Customer Contact:       0122094239
Date Purchased:         21/02/2022
Total Quantity:         2
Total Price:    35

ID:                     3
Customer Name:          Sia De Long
Customer Contact:       0122654199
Date Purchased:         22/02/2022
Total Quantity:         3
Total Price:    70

ID:                     4
Customer Name:          Thian Shan You
Customer Contact:       0135094199
Date Purchased:         23/02/2022
Total Quantity:         4
Total Price:    82

ID:                     1
Customer Name:          Thian Shan You
Customer Contact:       0122095199
Date Purchased:         20/02/2022
Total Quantity:         5
Total Price:    102

Press any key to continue . . .
```

Figure 44 Output of Sort by Price

When sort purchase is selected, the list is sorted by price in descending order.

```
C:\Users\thian\source\repos\DSTR_GROUP17\Debug\DSTR_GROUP17.exe
Total Quantity:         2
Total Price:    35

ID:                     3
Customer Name:          Sia De Long
Customer Contact:       0122654199
Date Purchased:         22/02/2022
Total Quantity:         3
Total Price:    70

ID:                     4
Customer Name:          Thian Shan You
Customer Contact:       0135094199
Date Purchased:         23/02/2022
Total Quantity:         4
Total Price:    82

ID:                     1
Customer Name:          Thian Shan You
Customer Contact:       0122095199
Date Purchased:         20/02/2022
Total Quantity:         5
Total Price:    102

Enter Purchase ID: 2
ID:                     1
Title:                  Apple
Price:  15

Press any key to continue . . .
```

Figure 45 Output for Specific Purchase Detail

Conclusion, Future Works and Reflection

In summary, all the proposed functions have been implemented successfully in Grandplex Movie Ticket Inventory management system. For data structure components, doubly linked list has been integrated to allow two-way traversal for more efficient searching and sorting functions. This system consists of two parts which are movie inventory and transaction. In part of movie inventory, some operations can be performed such as add movie, display movie, search movie, filter movie, update movie, sort movie and delete movie. Meanwhile for the transaction, user can add purchase, view purchase, sort purchase and view purchase details.

In terms of the features, the purchase component should be further expanded upon as the current one is a basic implementation. Besides, more comparison can be conducted between different types of data structure to determine the best and most efficient data structure to be implemented. After considering about the factor of time complexities, a study should be conducted on lower complexity algorithms.

Throughout this project, the differences of singly linked list and doubly linked list have been investigated. Due to the efficiency in searching and sorting functions, thus doubly linked list is selected for this project. In this project too, hands-on experience has been gained through implementing several functions such as add, edit, update, delete, sort and more in doubly linked list.

Workload Matrix

	Beh Chi Hao	Sia De Long	Thian Shan You
Implementation (%)			
Movie Inventory Component	43	57	
Purchase Component			100
Menu Component	100		
Documentation (%)			
Introduction		50	50
Implementation	33	33	33
Result	33	33	33
Conclusion, Future Works and Reflection	33	33	33
Signature	BCH	SDL	TSY