National University of Sciences and Technology (NUST)
School of Electrical Engineering and Computer Science

Department of Computing

School of Electrical Engineering and Computer Science

CS250 – Data Structures and Algorithms

Assignment 1: Linked Lists

Submission Details

Name	**CMS ID**
Muhammad Umer	345834
Group	GP – 1
Instructor	Bostan Khan
Class	BEE12
Date	27/02/2024

1 Table of Contents

2 Linked Lists

2.1 Objective

The objective of this assignment is to explore various applications of linked lists and gain hands-on experience with implementing and manipulating linked list data structures.

Please read the instructions for each task with great care and be sure to implement the requirements stated against each task.

Remember to document your work well in the documentation report. Explain the code, its design and test results with details for obtaining max marks for the whole assignment.

2.2 Tools/Software Requirement

- Visual Studio C++

2.3 Deliverables

You will have to submit a separate .cpp source file for each task. Present the documentation report including the test results in a well-prepared PDF.

The report PDF and the sources files should be compressed in a single zip file and submitted on LMS with the name in following format: <Student_name>_<Reg_no>.zip

2.4 Documentation Report PDF

Provide a report PDF for all your linked list implementations, including explanations of the data structures and algorithms used, as well as any design decisions made for each task. Document the usage of your linked list in each of the application scenarios, including how the linked list data structure is utilized to solve the problem. **The marks of your coding tasks will be directly influenced by the documentation report.** Include clear instructions on how to compile/run your code and how to interact with the implemented applications.

2.5 Additional Notes:

- You are encouraged to be creative and innovative in your implementations. Consider additional features or optimizations that could enhance the functionality or performance of your applications.
- Collaboration with classmates is allowed for discussing concepts and problem-solving strategies, but each student must submit their own individual solution.
- **Plagiarism or copying of code from other students will result in zero marks.**

3 Tasks

3.1 Implementation of Linked List [Marks: 5]

Implement a linked list using classes in C++ for data type int. Your linked list should support the following operations:

- Insertion of a new node at the beginning, middle, and end of the list.
- Deletion of a node from the list.
- Searching for a node based on a given value.
- Traversing and printing the contents of the list.

Documentation & Explanation

The `LinkedList` class is implemented using a struct `Node` to represent each node in the list. The class has a head and tail pointer to keep track of the first and last nodes, and a length variable to keep track of the number of nodes in the list.

insert_beginning(int data)	Inserts a new node at the beginning of the list.
insert_middle(int data)	Inserts a new node in the middle of the list.
insert_end(int data)	Inserts a new node at the end of the list.
delete_node(int data)	Deletes a node from the list based on the given data.
search_node(int data)	Searches for a node in the list based on the given data.
print_list()	Traverses and prints the contents of the list.

- Implementation uses separate functions for insertion at the beginning, middle, and end to handle different scenarios efficiently.
- Maintaining a length variable avoids recalculating the list size in case of insertion of a node in the middle of the list.
- Using both head and tail pointers allows for efficient ($O(1)$) insertion at the end, in lieu of traversing the entire list to find the last node.

Implementation

```
#include <iostream>
using namespace std;

class LinkedList {
   private:
    struct Node {
        int data;
        Node *next;
    };
    Node *head, *tail;
    int length;  // length of the list
```

```
public:
    // Constructor and destructor
    LinkedList() {
        head = nullptr;
        tail = nullptr;
        length = 0;
    }
    ~LinkedList() {}

    // Method definitions
    void insert_beginning(int data) {
        Node *new_node = new Node;
        new_node->data = data;
        new_node->next = head;
        head = new_node;

        // if this is the first node, it's also the tail
        if (length == 0) {
            tail = new_node;
        }

        length++;
    }

    void insert_middle(int data) {
        Node *new_node = new Node;
        Node *temp = head;
        new_node->data = data;

        // if the list is empty, insert at the beginning
        if (length == 0) {
            new_node->next = head;
            head = new_node;
            tail = new_node;
            length++;
            return;
        }

        int mid = (length % 2 == 0) ? length / 2 : (length / 2) + 1;

        while (--mid)  // runs the loop mid - 1 times
        {
            temp = temp->next;
        }  // temp is now pointing to the node before the middle node

        new_node->next = temp->next;
        temp->next = new_node;

        // if the new node is inserted at the end, update the tail
        if (new_node->next == nullptr) {
            tail = new_node;
        }
        length++;
    }

    void insert_end(int data) {
        Node *new_node = new Node;
        new_node->data = data;
        new_node->next = nullptr;

        // if the list is empty
        if (head == nullptr) {
            head = new_node;
        } else {
            tail->next = new_node;
        }

        // the new node is now the tail
        tail = new_node;
        length++;
    }

    void delete_node(int data) {
        Node *target = head;
        Node *prev = nullptr;

        // if the list is empty
        if (head == nullptr) {
            cout << "- List is empty, nothing to delete" << endl;
            return;
        }

        // if the node to be deleted is the head
        if (head->data == data) {
            head = head->next;
            delete target;
            length--;
            return;
        }

        while (target != nullptr && target->data != data) {
            prev = target;
            target = target->next;
        }

        // if the node is not found
        if (target == nullptr) {
            cout << "- No node with the given data found" << endl;
            return;
        }

        prev->next = target->next;
        // if the node to be deleted is the tail
        if (target == tail) {
            tail = prev;
        }

        delete target;
        length--;
    }

    void search_node(int data) {
        Node *temp = head;
        int index = 0;

        while (temp != nullptr) {
            if (temp->data == data) {
                cout << "- Node with data (" << data << ") found at position "
                     << index << endl;
                return;
            }
            temp = temp->next;
            index++;
        }

        cout << "- Node with data (" << data << ") not found" << endl;
    }

    void print_list() {
        Node *temp = head;

        if (length == 0) {
            cout << "- List is empty" << endl;
            return;
        }

        while (temp != nullptr) {
            cout << temp->data << " ";
            temp = temp->next;
        }
        cout << endl;
    }
};
```

The following `main` function demonstrates the usage of the `LinkedList` class by inserting nodes at the beginning, middle, and end of the list, deleting nodes, and searching for nodes based on their data. The output of the program shows the results of each operation.

Testing

```
int main() {
    LinkedList list;

    cout << "Insertion at the beginning: " << endl;
    list.insert_beginning(5);
    list.insert_beginning(10);
    list.print_list();

    cout << "\nInsertion in the middle: " << endl;
    list.insert_middle(25);
    list.print_list();

    cout << "\nInsertion at the end: " << endl;
    list.insert_end(40);
    list.print_list();

    cout << "\nDeletion of nodes: " << endl;
    list.delete_node(25);
    list.delete_node(40);
    list.print_list();

    cout << "\nSearch for nodes: " << endl;
    list.search_node(5);
    list.search_node(50);

    return 0;
}
```

Output

```
Insertion at the beginning:
10 5

Insertion in the middle:
10 25 5

Insertion at the end:
10 25 5 40

Deletion of nodes:
10 5

Search for nodes:
- Node with data (5) found at position 1
- Node with data (50) not found
```

Instructions to Reproduce Output

```
                         Linux (Ubuntu 22.04)
- Compiler Version
gcc version 11.4.0 (Ubuntu 11.4.0-1ubuntu1~22.04)

- Steps
cd to extracted .zip directory
user@hostname:/dsa/assignment_1# g++ -o linked_list linked_list.cpp
user@hostname:/dsa/assignment_1# ./linked_list
```

```
                                   Windows

- Compiler Version
gcc version 13.2.0 (Rev3, Built by MSYS2 project)

- Steps
cd to extracted .zip directory
C:\user\dsa\assignment_1> g++ -Wall -g linked_list.cpp -o linked_list.exe
C:\user\dsa\assignment_1> linked_list.exe
```

3.2 Application Scenarios [Marks: 40]

Please write code for the following application scenarios where linked lists are commonly used. For each scenario, implement the necessary functionality using linked lists. You can use singly, doubly, and circular linked lists. Please provide a separate .cpp file for each of the 4 tasks below.

3.2.1 Employee Management System [10]

Implement an employee management system where each employee is represented as a node in the linked list. Each employee must have the following fields: employee ID, name, contact_number, age, salary, marital_status. The system should support operations such as adding new employees, removing employees based on **searching for employee id**, updating employee information based on **searching for employee id**, and displaying the list of employees.

Documentation & Explanation

The `EmployeeManagementSystem` class is implemented using a singly linked list. Each employee is represented as a node in the linked list. The `Employee` struct contains the fields for an employee: employee_id, name, contact_number, age, salary, marital_status. The `Employee` struct also contains a pointer to the next node in the linked list.

`add_employee``(int employee_id, string name, string contact_number, int age, double salary, string marital_status)`	Adds a new employee to the linked list. The new employee is created as a new node and inserted at the beginning of the list.
`remove_employee``(int employee_id)`	Removes an employee from the linked list based on the employee_id. If the employee_id is found, the corresponding node is removed from the list.
`update_employee``(int employee_id)`	Updates the details of an employee based on the employee_id. If the employee_id is found, the user is prompted to input new details for the employee.
`display_employees``()`	Displays the details of all employees in the linked list.

- The system uses a singly linked list for simplicity and ease of implementation.
- Each employee node stores all relevant information, simplifying access and retrieval of individual employee data.
- The input_details function allows skipping `('skip')` input fields to keep existing data, improving user experience.

Implementation

```
void input_details(string &name, string &contact_number, int &age,
                   double &salary, string &marital_status) {
    string input;
    cout << "Enter name (or 'skip' to keep current): ";
    getline(cin, input);
    if (input != "skip") {
        name = input;
    }

    cout << "Enter contact number (or 'skip' to keep current): ";
    getline(cin, input);
    if (input != "skip") {
        contact_number = input;
    }

    cout << "Enter age (or 'skip' to keep current): ";
    getline(cin, input);
    if (input != "skip") {
        age = stoi(input);
    }

    cout << "Enter salary (or 'skip' to keep current): ";
    getline(cin, input);
    if (input != "skip") {
        salary = stod(input);
    }

    cout << "Enter marital status (or 'skip' to keep current): ";
    getline(cin, input);
    if (input != "skip") {
        marital_status = input;
    }
}

class EmployeeManagementSystem {
   private:
    struct Employee {
        int employee_id;
        string name;
        string contact_number;
        int age;
        double salary;
        string marital_status;

        Employee *next;
    };

    Employee *head;

   public:
    EmployeeManagementSystem() { head = nullptr; }
    ~EmployeeManagementSystem() {}

    void add_employee(int employee_id, string name, string contact_number,
                      int age, double salary, string marital_status) {
```

```
        Employee *new_employee = new Employee;

        new_employee->employee_id = employee_id;
        new_employee->name = name;
        new_employee->contact_number = contact_number;
        new_employee->age = age;
        new_employee->salary = salary;
        new_employee->marital_status = marital_status;

        new_employee->next = head;
        head = new_employee;
    }

    void remove_employee(int employee_id) {
        Employee *current = head;
        Employee *previous = nullptr;

        if (head == nullptr)  // no employees
        {
            cout << "No employees to remove." << endl;
            return;
        }

        while (current != nullptr) {
            if (current->employee_id == employee_id) {
                if (previous == nullptr) {
                    head = current->next;
                } else {
                    previous->next = current->next;
                }

                delete current;
                return;
            }

            // move to the next node
            previous = current;
            current = current->next;
        }

        cout << "Employee with ID " << employee_id << " not found." << endl;
    }

    void update_employee(int employee_id) {
        Employee *current = head;

        while (current != nullptr) {
            if (current->employee_id == employee_id) {
                cout << "Enter new details for employee with ID "
                     << employee_id << endl;

                // input new details, skip to keep current
                input_details(current->name, current->contact_number,
                              current->age, current->salary,
                              current->marital_status);

                return;
            }

            current = current->next;
        }

        cout << "Employee with ID " << employee_id << " not found." << endl;
    }

    void display_employees() {
        Employee *current = head;

        while (current != nullptr) {
            cout << "- Employee ID: " << current->employee_id << endl;
            cout << "- Name: " << current->name << endl;
            cout << "- Contact Number: " << current->contact_number << endl;
            cout << "- Age: " << current->age << endl;
            cout << "- Salary: " << current->salary << endl;
            cout << "- Marital Status: " << current->marital_status << endl;
            cout << endl;

            current = current->next;
        }
    }
};
```

The following main function demonstrates the usage of the `EmployeeManagementSystem` class by adding initial employees, removing an employee, updating an employee, and displaying the list of employees.

Testing

```
int main() {
    EmployeeManagementSystem system;

    system.add_employee(345834, "Umer", "923350526986", 21, 50000, "single");
    system.add_employee(345835, "Ali", "923350526987", 22, 60000, "married");
    system.add_employee(345836, "Ahmed", "923350526988", 23, 70000, "single");

    cout << "Initial Employees: " << endl;
    system.display_employees();

    system.remove_employee(345835);
    cout << "After removing employee with ID 345835: " << endl;
    system.display_employees();

    system.update_employee(345836);
    cout << "After updating employee with ID 345836: " << endl;
    system.display_employees();

    return 0;
}
```

Output

```
Initial Employees:
- Employee ID: 345836
- Name: Ahmed
- Contact Number: 923350526988
- Age: 23
- Salary: 70000
- Marital Status: single

- Employee ID: 345835
- Name: Ali
- Contact Number: 923350526987
- Age: 22
- Salary: 60000
- Marital Status: married

- Employee ID: 345834
- Name: Umer
- Contact Number: 923350526986
- Age: 21
- Salary: 50000
- Marital Status: single

After removing employee with ID 345835:
- Employee ID: 345836
- Name: Ahmed
- Contact Number: 923350526988
- Age: 23
- Salary: 70000
- Marital Status: single

- Employee ID: 345834
- Name: Umer
- Contact Number: 923350526986
- Age: 21
- Salary: 50000
- Marital Status: single

Enter new details for employee with ID 345836
Enter name (or 'skip' to keep current): skip
Enter contact number (or 'skip' to keep current): skip
Enter age (or 'skip' to keep current): skip
Enter salary (or 'skip' to keep current): 75000
Enter marital status (or 'skip' to keep current): skip
After updating employee with ID 345836:
- Employee ID: 345836
- Name: Ahmed
- Contact Number: 923350526988
- Age: 23
- Salary: 75000
- Marital Status: single

- Employee ID: 345834
- Name: Umer
- Contact Number: 923350526986
- Age: 21
- Salary: 50000
- Marital Status: single
```

Instructions to Reproduce Output

```
                              Linux (Ubuntu 22.04)
- Compiler Version
gcc version 11.4.0 (Ubuntu 11.4.0-1ubuntu1~22.04)

- Steps
cd to extracted .zip directory
user@hostname:/dsa/assignment_1# g++ -o employee_system employee_system.cpp
user@hostname:/dsa/assignment_1# ./employee_system

                                    Windows

- Compiler Version
gcc version 13.2.0 (Rev3, Built by MSYS2 project)

- Steps
cd to extracted .zip directory
C:\user\dsa\assignment_1> g++ -Wall -g employee_system.cpp -o
                          employee_system.exe
C:\user\dsa\assignment_1> employee_system.exe
```

3.2.2 **Library Catalog System** [15]

Implement a library catalogue system that manages books in a library. Each book can be represented as a node in the linked list. The system should support functionalities such as adding new books, removing books, searching for books by title or author. There can be multiple books by the same author and same name. The system should also have the following functions:

1. `books_by_author()`: Get an authors name as input and display number of books by that Author in the catalog.
2. `display_catalogue()`: On execution, display all the books and number of copies of each book present in the list. This means that one book should be displayed only once with the number of copies in the list.

Implement the system and display its working in the documentation report.

Documentation & Explanation

The `LibraryCatalogSystem` class is a linked list implementation that manages books in a library. The class has methods to add books, remove books, search for books by title or author, display the entire catalogue, and get the number of books by a specific author. The `Book` struct contains the fields for a book: title, author, copies. It also contains a pointer to the next node in the linked list.

`add_book``(string title, string author, int copies = 1)`	Adds a new book to the list with the given number of copies. If the book already exists in the list, the number of copies is increased.
`remove_book``(string title, string author, int copies = 1)`	Removes the given number of copies of the book from the list. If the number of copies is reduced to zero, the book is removed from the list.
`search_by_title``(string title)`	Searches for a book by title and displays all the details of the book, including the number of copies.
`search_by_author``(string author)`	Searches for books by author and displays all the books by the author.
`books_by_author``(string author)`	Gets an author's name as input and displays the number of books by that author in the catalog.
`display_catalogue``()`	Displays all the books in the catalogue along with the number of copies of each book.

- The system employs a singly linked list to store book information.
- A tail pointer is maintained to make extensions to the system easier.
- Each `Book` struct maintains a field of "`copies`" to account for books by the same author AND the same title. This field allows for deletion of the specific book from the catalogue if copies reduces to zero.

Implementation

```cpp
class LibraryCatalogSystem {
   private:
    struct Book {
        string title;
        string author;
        int copies;
        Book *next;
    };
    Book *head, *tail;

   public:
    // Constructor and destructor
    LibraryCatalogSystem() {
        head = nullptr;
        tail = nullptr;
    }
    ~LibraryCatalogSystem() {}

    // Method definitions
    void add_book(string title, string author, int copies = 1) {
        // if book already exists, increase the number of copies else add a new
        // book to the list with the given number of copies
        Book *current = head;
        while (current != nullptr) {
            if (current->title == title && current->author == author) {
                current->copies += copies;
                return;
            }
            current = current->next;
        }

        Book *new_book = new Book;
        new_book->title = title;
        new_book->author = author;
        new_book->copies = copies;
        new_book->next = nullptr;

        if (head == nullptr) {
            head = new_book;
            tail = new_book;
        } else {
            tail->next = new_book;
            tail = new_book;
        }
    }

    void remove_book(string title, string author, int copies = 1) {
        // remove the given number of copies of the book from the list
        Book *current = head, *prev = head;

        while (current != nullptr) {
            if (current->title == title && current->author == author) {
                if (current->copies > copies) {
                    current->copies -= copies;
                } else {
                    // remove the book from the list
                    if (current == head) {
                        head = head->next;
                    } else {
                        while (prev->next != current) {
                            prev = prev->next;
                        }
                        prev->next = current->next;
                    }
                    if (current == tail) {
                        tail = prev;
                    }
                    delete current;
                }
                return;
            }
            current = current->next;
        }

        cout << "- Book not found!" << endl;
    }

    void search_by_title(string title) {
        // search for books by title and display the number of copies
        Book *current = head;
        while (current != nullptr) {
            if (current->title == title) {
                cout << "- Title: " << current->title
                     << ", Author: " << current->author
                     << ", Copies: " << current->copies << endl;

                return;
            }

            current = current->next;
        }

        cout << "- Book not found!" << endl;
    }

    void search_by_author(string author) {
        // search for books by author and display all the books by the author
        Book *current = head;
        int valid = 0;
        while (current != nullptr) {
            if (current->author == author) {
                cout << "- Title: " << current->title
                     << ", Copies: " << current->copies << endl;
                valid = 1;
            }

            current = current->next;
        }

        if (valid == 0) {
            cout << "- No books found by " << author << endl;
        }
    }

    void books_by_author(string author) {
        // get an authors name as input and display number of books
        int count = 0;
        Book *current = head;
        while (current != nullptr) {
            if (current->author == author) {
                count += current->copies;
            }

            current = current->next;
        }

        if (count > 0) {
            cout << "- Number of books by " << author << ": " << count << endl;
        } else {
            cout << "- No books found by " << author << endl;
        }
    }

    void display_catalogue() {
        Book *current = head;
        while (current != nullptr) {
            cout << "- Title: " << current->title
                 << ", Author: " << current->author
                 << ", Copies: " << current->copies << endl;

            current = current->next;
        }
    }
};
```

The following main function demonstrates the usage of the `LibraryCatalogSystem` class by adding books, adding more copies of the same book, removing book copies, searching for books by title or author, and getting the number of books by an author.

Testing

```
int main() {
    LibraryCatalogSystem library;

    cout << "Adding books to the library:" << endl;
    library.add_book("The Great Gatsby", "F. Scott Fitzgerald");
    library.add_book("To Kill a Mockingbird", "Harper Lee");
    library.add_book("This Side of Paradise", "F. Scott Fitzgerald");
    library.display_catalogue();

    cout << "\nAdding more copies of the same book:" << endl;
    library.add_book("The Great Gatsby", "F. Scott Fitzgerald", 2);
    library.display_catalogue();

    cout << "\nRemoving book copies:" << endl;
    library.remove_book("The Great Gatsby", "F. Scott Fitzgerald");
    library.display_catalogue();

    cout << "\nSearching for books by title:" << endl;
    library.search_by_title("To Kill a Mockingbird");

    cout << "\nSearching for books by author:" << endl;
    library.search_by_author("Harper Lee");
    library.search_by_author("M. Twain");

    cout << "\nGetting number of books by author:" << endl;
    library.books_by_author("F. Scott Fitzgerald");

    return 0;
}
```

Output

```
Adding books to the library:
- Title: The Great Gatsby, Author: F. Scott Fitzgerald, Copies: 1
- Title: To Kill a Mockingbird, Author: Harper Lee, Copies: 1
- Title: This Side of Paradise, Author: F. Scott Fitzgerald, Copies: 1

Adding more copies of the same book:
- Title: The Great Gatsby, Author: F. Scott Fitzgerald, Copies: 3
- Title: To Kill a Mockingbird, Author: Harper Lee, Copies: 1
- Title: This Side of Paradise, Author: F. Scott Fitzgerald, Copies: 1

Removing book copies:
- Title: The Great Gatsby, Author: F. Scott Fitzgerald, Copies: 2
- Title: To Kill a Mockingbird, Author: Harper Lee, Copies: 1
- Title: This Side of Paradise, Author: F. Scott Fitzgerald, Copies: 1

Searching for books by title:
- Title: To Kill a Mockingbird, Author: Harper Lee, Copies: 1

Searching for books by author:
- Title: To Kill a Mockingbird, Copies: 1
- No books found by M. Twain

Getting number of books by author:
- Number of books by F. Scott Fitzgerald: 3
```

Instructions to Reproduce Output

```
                          Linux (Ubuntu 22.04)
- Compiler Version
gcc version 11.4.0 (Ubuntu 11.4.0-1ubuntu1~22.04)

- Steps
cd to extracted .zip directory
user@hostname:/dsa/assignment_1# g++ -o library_system library_system.cpp
user@hostname:/dsa/assignment_1# ./library_system

                                Windows

- Compiler Version
gcc version 13.2.0 (Rev3, Built by MSYS2 project)

- Steps
cd to extracted .zip directory
C:\user\dsa\assignment_1> g++ -Wall -g library_system.cpp -o
                          library_system.exe
C:\user\dsa\assignment_1> library_system.exe
```

3.2.3 **Event Scheduler** [10]

Implement an event scheduler that manages upcoming events or appointments. Each event can be represented as a node in the linked list. Each event node should have a name, date, and event details. The system should support operations such as adding new events, removing events, updating event details, and displaying the list of upcoming events. Choose an appropriate format of your choice for the date. The events in the linked list should always be sorted based on date. Whenever you add a new event, it should be added to the position with respect to its date.

Documentation & Explanation

The `EventScheduler` class is implemented using a singly linked list. The class has methods to add, remove, update, and display events. The `Event` struct represents each event as a node in the linked list, with fields for name, date, details, and a pointer to the next node.

add_event(string name, string date, string details)	Adds a new event to the list with the given date. The method traverses the list to find the correct position for the new event based on its date, and then adds the event to the list.
remove_event(string date)	Removes the event with the given date from the list.
update_event(string date, string new_name, string new_details)	Updates the event with the given date with the new name and details.
display_events()	Displays all the events in the list.

- The system utilizes a singly linked list to store event information.
- Implementation complexity for self-organizing list overshadows the cost incurred through addition of an event through just comparing.
- C++ allows for lexicographical comparison of strings; dates can be validly compared so as long as the format is "`yyyy/mm/dd`".

Implementation

```
class EventScheduler {
   private:
    struct Event {
        string name;
        string date;
        string details;
        Event *next;
    };
    Event *head, *tail;

   public:
    // Constructor and destructor
    EventScheduler() {
        head = nullptr;
        tail = nullptr;
    }
    ~EventScheduler() {}

    // Method definitions
    void add_event(string name, string date, string details) {
        // add a new event to the list with the given date
        Event *new_event = new Event;
        new_event->name = name;
        new_event->date = date;
        new_event->details = details;
        new_event->next = nullptr;

        if (head == nullptr) {
            head = new_event;
            tail = new_event;
        } else {
            Event *current = head;
            Event *prev = nullptr;
            while (current != nullptr && current->date < date) {
                prev = current;
                current = current->next;
            }
            if (prev == nullptr) {
                new_event->next = head;
                head = new_event;
            } else if (current == nullptr) {
                tail->next = new_event;
                tail = new_event;
            } else {
                prev->next = new_event;
                new_event->next = current;
            }
        }
    }

    void remove_event(string date) {
        // remove the event with the given date from the list
        Event *current = head;
        Event *prev = nullptr;
        while (current != nullptr) {
            if (current->date == date) {
                if (prev == nullptr) {
                    head = current->next;
                } else {
                    prev->next = current->next;
                }
                delete current;
                return;
            }
            prev = current;
            current = current->next;
        }
        cout << "Event with date " << date << " not found." << endl;
    }

    void update_event(string date, string new_name, string new_details) {
        // update the event with the given date with the new name and details
        Event *current = head;
        while (current != nullptr) {
            if (current->date == date) {
                current->name = new_name;
                current->details = new_details;
                return;
            }
            current = current->next;
        }

        cout << "Event with date " << date << " not found." << endl;
    }

    void display_events() {
        // display all the events in the list
        Event *current = head;
        while (current != nullptr) {
            cout << "- Name" << current->name << ", Date: " << current->date
                 << ", Details: " << current->details << endl;

            current = current->next;
        }
    }
};
```

The following `main` function demonstrates the use of the `EventScheduler` class by adding events, removing an event, updating an event, and displaying the list of events.

Testing

```
int main() {
    EventScheduler scheduler;

    cout << "Adding events" << endl;
    scheduler.add_event("Meeting", "2021/10/15", "Discuss project progress");
    scheduler.add_event("Lunch", "2021/10/16", "Meet with friends");
    scheduler.add_event("Dinner", "2021/10/17", "Family gathering");
    scheduler.add_event("Conference", "2021/10/18", "Attend tech conference");
    scheduler.add_event("Birthday", "2019/10/19", "Celebrate with friends");
    scheduler.display_events();

    cout << "\nRemoving event on 2021/10/17" << endl;
    scheduler.remove_event("2021/10/17");
    scheduler.display_events();

    cout << "\nUpdating event on 2021/10/16" << endl;
    scheduler.update_event("2021/10/16",
                           "Brunch", "Meet with friends and family");
    scheduler.display_events();

    return 0;
}
```

Output

```
Adding events
- NameBirthday, Date: 2019/10/19, Details: Celebrate with friends
- NameMeeting, Date: 2021/10/15, Details: Discuss project progress
- NameLunch, Date: 2021/10/16, Details: Meet with friends
- NameDinner, Date: 2021/10/17, Details: Family gathering
- NameConference, Date: 2021/10/18, Details: Attend tech conference

Removing event on 2021/10/17
- NameBirthday, Date: 2019/10/19, Details: Celebrate with friends
- NameMeeting, Date: 2021/10/15, Details: Discuss project progress
- NameLunch, Date: 2021/10/16, Details: Meet with friends
- NameConference, Date: 2021/10/18, Details: Attend tech conference

Updating event on 2021/10/16
- NameBirthday, Date: 2019/10/19, Details: Celebrate with friends
- NameMeeting, Date: 2021/10/15, Details: Discuss project progress
- NameBrunch, Date: 2021/10/16, Details: Meet with friends and family
- NameConference, Date: 2021/10/18, Details: Attend tech conference
```

Instructions to Reproduce Output

```
                              Linux (Ubuntu 22.04)
- Compiler Version
gcc version 11.4.0 (Ubuntu 11.4.0-1ubuntu1~22.04)

- Steps
cd to extracted .zip directory
user@hostname:/dsa/assignment_1# g++ -o event_scheduler event_scheduler.cpp
user@hostname:/dsa/assignment_1# ./event_scheduler

                                    Windows

- Compiler Version
gcc version 13.2.0 (Rev3, Built by MSYS2 project)

- Steps
cd to extracted .zip directory
C:\user\dsa\assignment_1> g++ -Wall -g event_scheduler.cpp -o
                          event_scheduler.exe
C:\user\dsa\assignment_1> event_scheduler.exe
```

3.2.4 **Shopping Cart** [10]

Implement a shopping cart system for an online store. Each item in the shopping cart can be represented as a node in the linked list. Each node should have an item ID, item name and price. The system should support functionalities such as adding items to the cart, removing items, updating quantities for each item, calculating total price for the shopping cart (list), and displaying the contents of the cart.

Documentation & Explanation

The ShoppingCart class is a linked list implementation that represents a shopping cart system for an online store. Each item in the shopping cart is represented as a node in the linked list. Each node has an item ID, item name, price, and quantity. The system supports functionalities such as adding items to the cart, removing items, updating quantities for each item, calculating the total price for the shopping cart, and displaying the contents of the cart.

The `EventScheduler` class is implemented using a singly linked list. The class has methods to add, remove, update, and display events. The `Event` struct represents each event as a node in the linked list, with fields for name, date, details, and a pointer to the next node.

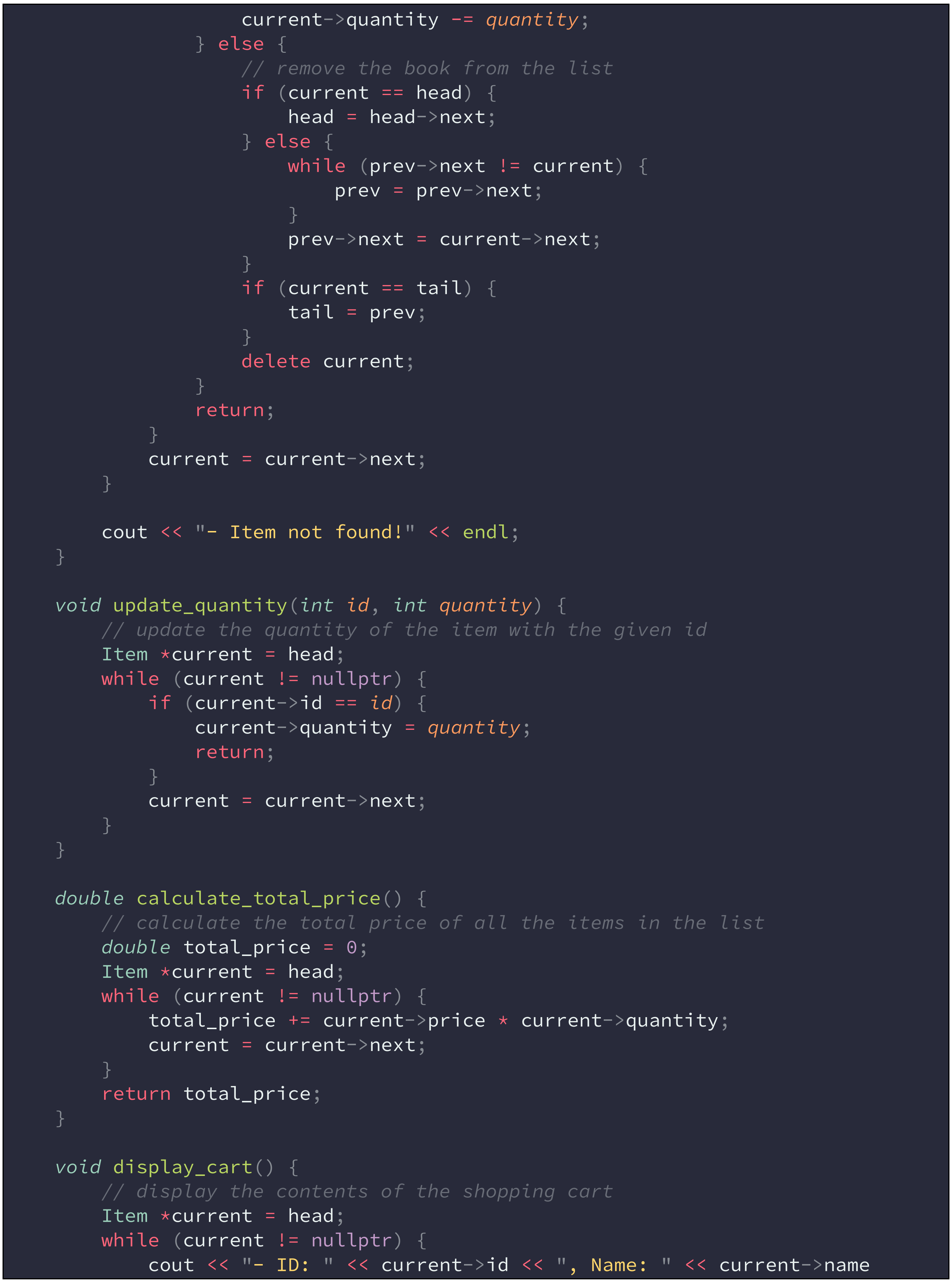
`add_item``(int id, string name, double price, int quantity = 1)`	Adds an item to the shopping cart. If the item already exists in the cart, the quantity is increased. If the item does not exist, a new item is added to the list with the given quantity.
`remove_item``(int id, int quantity = 1)`	Removes the given number of items from the shopping cart. If the quantity to be removed is greater than the quantity of the item in the cart, the item is completely removed from the cart.
`update_quantity``(int id, int quantity)`	Updates the quantity of the item with the given ID in the shopping cart.
`calculate_total_price``()`	Calculates the total price of all the items in the shopping cart.
`display_cart``()`	Displays the contents of the shopping cart.

- Design choices are more or less the same as that made for `LibraryCatalogSystem`.
- A field of "`quantity`" is maintained to allow for existence of duplicate items, as is common in shopping carts. When removing items, the quantity decreases by the argument specified unless it becomes zero, in which case the item is deleted.

Implementation

```
class ShoppingCart {
  private:
   struct Item {
       int id;
```

```
        string name;
        double price;
        int quantity;
        Item *next;
    };
    Item *head, *tail;

public:
    // Constructor and destructor
    ShoppingCart() {
        head = nullptr;
        tail = nullptr;
    }
    ~ShoppingCart() {}

    // Method definitions
    void add_item(int id, string name, double price, int quantity = 1) {
        // if item already exists, increase the quantity else add a new
        // item to the list with the given quantity
        Item *current = head;
        while (current != nullptr) {
            if (current->id == id) {
                current->quantity += quantity;
                return;
            }
            current = current->next;
        }

        Item *new_item = new Item;
        new_item->id = id;
        new_item->name = name;
        new_item->price = price;
        new_item->quantity = quantity;
        new_item->next = nullptr;

        if (head == nullptr) {
            head = new_item;
            tail = new_item;
        } else {
            tail->next = new_item;
            tail = new_item;
        }
    }

    void remove_item(int id, int quantity = 1) {
        // remove the given number of items from the list
        Item *current = head;
        Item *prev = nullptr;

        while (current != nullptr) {
            if (current->id == id) {
                if (current->quantity > quantity) {
```



```
                    current->quantity -= quantity;
                } else {
                    // remove the book from the list
                    if (current == head) {
                        head = head->next;
                    } else {
                        while (prev->next != current) {
                            prev = prev->next;
                        }
                        prev->next = current->next;
                    }
                    if (current == tail) {
                        tail = prev;
                    }
                    delete current;
                }
                return;
            }
            current = current->next;
        }

        cout << "- Item not found!" << endl;
    }

    void update_quantity(int id, int quantity) {
        // update the quantity of the item with the given id
        Item *current = head;
        while (current != nullptr) {
            if (current->id == id) {
                current->quantity = quantity;
                return;
            }
            current = current->next;
        }
    }

    double calculate_total_price() {
        // calculate the total price of all the items in the list
        double total_price = 0;
        Item *current = head;
        while (current != nullptr) {
            total_price += current->price * current->quantity;
            current = current->next;
        }
        return total_price;
    }

    void display_cart() {
        // display the contents of the shopping cart
        Item *current = head;
        while (current != nullptr) {
            cout << "- ID: " << current->id << ", Name: " << current->name
```

```
                  << ", Price: " << current->price
                  << ", Quantity: " << current->quantity << endl;
             current = current->next;
         }
     }
};
```

The `main` function demonstrates the usage of the `ShoppingCart` class by adding items to the cart, removing items, updating quantities, and calculating the total price of the cart.

Testing

```
int main() {
    ShoppingCart cart;

    cout << "Adding items to the cart:" << endl;
    cart.add_item(1, "KitKat", 250, 3);
    cart.add_item(2, "Mouse", 5000, 1);
    cart.add_item(3, "Face Mask", 100, 4);
    cart.display_cart();
    cout << "\nTotal Price: " << cart.calculate_total_price() << endl;

    cout << "\nRemoving items from the cart:" << endl;
    cart.remove_item(3);
    cart.display_cart();

    cout << "\nUpdating quantities in the cart:" << endl;
    cart.update_quantity(1, 5);
    cart.display_cart();

    cout << "\nTotal Price: " << cart.calculate_total_price() << endl;

    return 0;
}
```

Output

```
Adding items to the cart:
- ID: 1, Name: KitKat, Price: 250, Quantity: 3
- ID: 2, Name: Mouse, Price: 5000, Quantity: 1
- ID: 3, Name: Face Mask, Price: 100, Quantity: 4

Total Price: 6150

Removing items from the cart:
- ID: 1, Name: KitKat, Price: 250, Quantity: 3
- ID: 2, Name: Mouse, Price: 5000, Quantity: 1
- ID: 3, Name: Face Mask, Price: 100, Quantity: 3

Updating quantities in the cart:
- ID: 1, Name: KitKat, Price: 250, Quantity: 5
- ID: 2, Name: Mouse, Price: 5000, Quantity: 1
- ID: 3, Name: Face Mask, Price: 100, Quantity: 3
```

```
Total Price: 6550
```

Instructions to Reproduce Output

```
                    Linux (Ubuntu 22.04)
- Compiler Version
gcc version 11.4.0 (Ubuntu 11.4.0-1ubuntu1~22.04)

- Steps
cd to extracted .zip directory
user@hostname:/dsa/assignment_1# g++ -o shopping_cart shopping_cart.cpp
user@hostname:/dsa/assignment_1# ./shopping_cart

                          Windows

- Compiler Version
gcc version 13.2.0 (Rev3, Built by MSYS2 project)

- Steps
cd to extracted .zip directory
C:\user\dsa\assignment_1> g++ -Wall -g shopping_cart.cpp -o
                          shopping_cart.exe
C:\user\dsa\assignment_1> shopping_cart.exe
```