UTM
UNIVERSITI TEKNOLOGI MALAYSIA

FACULTY OF COMPUTING
UTM Johor Bahru

FACULTY OF COMPUTING

SESSION 2023/2024

SEMESTER 1

SECJ2013-04 DATA STRUCTURE AND ALGORITHM

(STRUKTUR DATA DAN ALGORITMA)

ASSIGNMENT 2 - DATA STRUCTURE AND OPERATIONS - LINKED LIST

LECTURER: DR. LIZAWATI BINTI MI YUSUF

NO	NAME	MATRIC NO
1	SARAH SOFEA BINTI ANUAR	A22EC104
2	FARAH HAZIRAH NISHA BINTI ABD LATIF	A22EC0159
3	MUHAMMAD ABDUH BIN ABDUL BA'ARI	A22EC0199

OBJECTIVE

The aim is to apply a courier management system based on linked lists. The program will read courier data from aTEST.txt file, and builds a linked list of courier objects, provides a menu- driven interface for performing some operations similar as adding a new courier node to the list, deleting a node from the list, searching for a courier node based on provided options, sorting the list based on provided options, and displaying the updated list. In this coding we use sorted- linked list class and node class to ease the courier management system on adding new nodes, deleting, sorting and searching systems. The system also has a user-friendly interface with a menu- driven system that determines users through the whole process until they choose to exit the system.

The following are the code's vital features and aims

Courier Service

A courier is by attributes such as name, parcel type, source, destination, status, and tracking number. There are approaches for setting and retrieving and also a display method for printing the courier details.

Node Class

A node in a linked list, each of which contains a Courier object and a pointer to the succeeding node.

LinkedList Class

Managed a linked list of Courier nodes. Ask and allow the user to insert a new node at the beginning, middle, or end of the list. Have operations for deleting a node from the list's beginning, middle, or end. This class also provides a method for locating a node using defined features. This function implements a sorting algorithm. Implements a sorting algorithm to sort the list based on varied criteria(similar as name, parcel type, and tracking number).

Main Purpose

The linked list is populated by reading courier information from a file(TEST.txt). Implements a menu- driven interface for interacting with the courier list. Have varied options which are adding, deleting, searching, sorting, and displaying courier nodes. The program will continue to run until the user chooses to exit.

Input from a File

The program reads courier information from a file(TEST.txt) and uses it to add the data of the linked list.

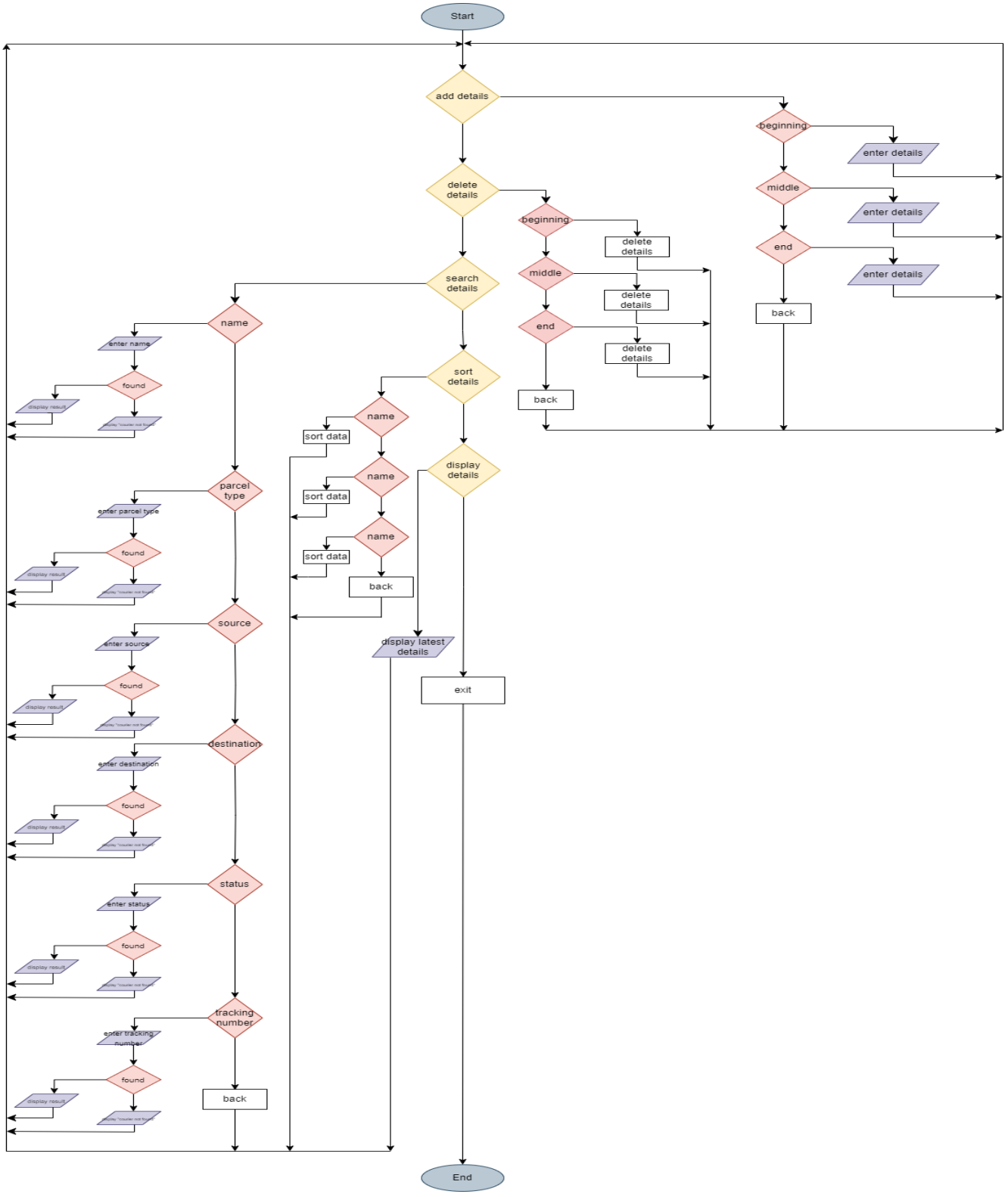
Menu- Based User Interface

The program provides a user-friendly menu to perform varied operations on the courier list. Checking for valid menu options and handling invalid input while searching are included in validation of DataBasic input confirmation . The aim is to produce an adjustable and reciprocal courier operation system based on linked lists which allows the users to make the courier data is easily managed.

SYNOPSIS

The Courier Management System allows workers to store data about parcels in a more organized way. For instance, we can update new data when we receive a parcel order by the customer and delete it a few days after the parcel has been delivered. We also can search the data based on the specified criteria of the parcel. Moreover, this system also can sort the data of parcels in ascending orders (name,parcel type,tracking number) .

DESIGN


Start
add details
delete details
search details
sort details
display details
exit
End
beginning
enter details
middle
enter details
end
enter details
back
beginning
delete details
middle
delete details
end
delete details
back
name
enter name
found
display result
display "courier not found"
parcel type
enter parcel type
found
display result
display "courier not found"
source
enter source
found
display result
display "courier not found"
destination
enter destination
found
display result
display "courier not found"
status
enter status
found
display result
display "courier not found"
tracking number
enter tracking number
found
display result
display "courier not found"
back
name
sort data
name
sort data
name
sort data
back
display latest details

DESIGN DESCRIPTION

ADDING A NEW NODE

1. Get the option from the main function.
2. From the option, the data will be added by corresponding function either insert at the front, middle or end.

   2.1 The insertion function in the Courier Management System allows users to add new couriers to the system

      2.1.1 Option 1: insertFront() function: insert details for the new courier (name, parcel type, source, destination, status, and tracking number) at the front of the linked list

      2.1.2 Option 2: insertMiddle() function: insert details for the new courier (name, parcel type, source, destination, status, and tracking number) at specified location (using position) of the linked list.

      2.1.3 Option 3: insertBack() function: insert details for the new courier (name, parcel type, source, destination, status, and tracking number) at the back of the linked list.

      2.1.4 Option 0: return back to the main menu.

DELETING A NODE

1. Get the option from the main function.

2. From the option, the data will be deleted by corresponding function either delete at the front, middle or end.

   2.1 The deletion function in the Courier Management System allows users to remove couriers from the system

      2.1.1 Option 1: deleteFront() function: the courier at the front of the list is removed.

      2.1.2 Option 2: deleteMiddle() function: the courier at the specific location (using position) of the list is removed.

      2.1.3 Option 3: deleteBack() function: the courier at the back of the list is removed.

      2.1.4 Option 0: return back to the main menu.

FINDING THE NODE

1. Get the option from the main function.
2. From the option, the data will be searched by corresponding function either by name, parcel type, source, destination, status or tracking number.
3. The function sets the value of a boolean variable (found) to false.
4. It iterates through the linked list, comparing the attributes of each courier to the corresponding attributes in the searchData.

   4.1 The findNode function uses searchKey to search the courier data based on the option that was chosen by the user.

      4.1.1 Option 1: name with specific searchKey will be displayed.

      4.1.2 Option 2: parcel type with specific searchKey will be displayed.

      4.1.3 Option 3: source with specific searchKey will be displayed.

      4.1.4 Option 4: destination with specific searchKey will be displayed.

      4.1.5 Option 5: status with specific searchKey will be displayed.

      4.1.6 Option 6: tracking number with specific searchKey will be displayed.

      4.1.7 Option 0: return back to the main menu.

SORTING THE LIST

1. Get the option from the main function.

2. The function determines whether the linked list is not empty and has more than one node.
3. It iteratively traverses the list, comparing and swapping elements based on the sorting criterion selected.

   3.1 The sortList function uses sortCriteria to sort the courier data based on the option that was chosen by the user.

      3.1.1 Option 1: courier data sort by name.

      3.1.2 Option 2: courier data sort by parcel type.

      3.1.3 Option 3: courier data sort by tracking number.

      3.1.4 Option 0: return back to the main menu.

DISPLAYING THE NODE

1. Display the unsorted data from the file (TEST.txt) that has been read or display sorted data and searched data based on users option.

SOURCE CODE

```
#include <iostream>
#include <fstream>
#include <string>
#include <iomanip>
using namespace std;

class Courier {
   string name, parcelType, source, destination, status;
   int trackingNum;

      public:
         Courier(string n = " ", string p = " ", string s = " ", string d = " ", string st = " ", int t =
0)
```

```
    : name(n), parcelType(p), source(s), destination(d), status(st), trackingNum(t) {}

    void setName(string n) { name = n; }
    void setType(string p) { parcelType = p; }
    void setSource(string s) { source = s; }
    void setDest(string d) { destination = d; }
    void setStat(string st) { status = st; }
    void setTrackNum(int t) { trackingNum = t; }

    string getName() const { return name; }
    string getType() const { return parcelType; }
    string getSource() const { return source; }
    string getDest() const { return destination; }
    string getStat() const { return status; }
    int getTrackNum() const { return trackingNum; }

    void display(bool printHeader = true) const {
        if (printHeader) {
            cout << left << setw(20) << "Tracking Number" << setw(20) <<
"Name"
                << setw(20) << "Parcel Type" << setw(20) << "Source"
                << setw(20) << "Destination" << setw(20) << "Status" << endl;
            cout << setfill('-') << setw(120) << " " << setfill(' ') << endl;
    }

        cout << left << setw(20) << trackingNum << setw(20) << name
            << setw(20) << parcelType << setw(20) << source
            << setw(20) << destination << setw(20) << status << endl;
    }
};
```

```
class Node {
	public:
		Courier data;
		Node* next;

		Node(Courier d) {
			data = d;
			next = NULL;
		}
};

class LinkedList {
	Node* head;

	public:
		LinkedList() {
			head = NULL;
		}

		Courier getNewCourier() {
		string n, parcel, sourc, dest, stat;
		int track;

		cout << "\nEnter details for the new Courier:" << endl;

		cout << "Name: ";
		cin.ignore();
		getline(cin, n);

		cout << "Parcel Type: ";
		getline(cin, parcel);
```

```
    cout << "Source: ";
    getline(cin, sourc);

    cout << "Destination: ";
    getline(cin, dest);

    cout << "Status: ";
    getline(cin, stat);

    cout << "Tracking Number: ";
    cin >> track;

    cin.ignore();

    return Courier(n, parcel, sourc, dest, stat, track);
}

    void insertFront(Courier d) {
        Node* newNode = new Node(d);
        newNode->next = head;
        head = newNode;
        cout << "Courier added at the beginning.\n" << endl;
    }

    void insertMiddle(Courier d, int position) {
        Node* newNode = new Node(d);

        if (!head) {
            if (position == 1) {
                head = newNode;
```

```
                return;
            }
            else {
                cout << "Invalid position. List is empty." << endl;
                return;
            }
        }

        int listSize = 1;
    Node* sizeChecker = head;
    while (sizeChecker->next) {
       listSize++;
       sizeChecker = sizeChecker->next;
    }
    if (position == 1) {
       cout << "Invalid position. Use 'Insert at the beginning' option instead.\n" <<
endl;
       delete newNode;
       return;
    } else if (position == listSize) {
       cout << "Invalid position. Use 'Insert at the end' option instead.\n" << endl;
       delete newNode;
       return;
    }

        Node* curr = head;
        int currPosition = 1;

        while (currPosition < position -1 && curr->next != NULL) {
            curr = curr->next;
            currPosition++;
```

```
        }

        if (currPosition < position - 1) {
                cout << "Invalid position." << endl;
                delete newNode;
                return;
        }

        newNode->next = curr->next;
        curr->next = newNode;
        cout << "Courier added at the position " << position << ".\n" << endl;
}

void insertBack(Courier d) {
        Node* newNode = new Node(d);
        if (!head) {
                head = newNode;
                return;
        }
        Node* curr = head;
        while (curr->next) {
                curr = curr->next;
        }
        curr->next = newNode;
}

void deleteFront() {
        if (!head) {
                cout << "List is empty." << endl;
                return;
        }
```

```
            Node* temp = head;
            head = head->next;
            delete temp;
            cout << "Courier deleted at the beginning.\n" << endl;
        }

        void deleteMiddle(int position) {
            if (!head) {
            cout << "List is empty." << endl;
            return;
        }

        int listSize = 1;
           Node* sizeChecker = head;
           while (sizeChecker->next) {
              listSize++;
              sizeChecker = sizeChecker->next;
           }
            if (position == 1) {
              cout << "Invalid position. Use 'Delete at the beginning' option instead.\n" <<
endl;
              return;
           } else if (position == listSize) {
              cout << "Invalid position. Use 'Delete at the end' option instead.\n" << endl;
              return;
           }

        Node* curr = head;
        Node* prev = NULL;
        int currPosition = 1;
```

```
        while (currPosition < position && curr) {
                prev = curr;
                curr = curr->next;
                currPosition++;
                }

                if (currPosition < position) {
                        cout << "Invalid position." << endl;
        return;
                }

                prev->next = curr->next;
                delete curr;
                cout << "Courier deleted at the position " << position << ".\n" << endl;
        }

        void deleteBack() {
                if (!head) {
        cout << "List is empty. Nothing to delete." << endl;
        return;
}
if (!head->next) {
        delete head;
        head = NULL;
        return;
                }
                Node* curr = head;
                while (curr->next->next) {
                        curr = curr->next;
                }
                delete curr->next;
```

```
            curr->next = NULL;
            cout << "Courier deleted at the end.\n" << endl;
        }

        void findNode(const Courier& searchData) {
            Node* curr = head;
            bool found = false;

            while (curr) {
                const Courier& currentData = curr->data;

                if ((searchData.getName().empty() || currentData.getName() ==
searchData.getName()) ||
                    (searchData.getType().empty() || currentData.getType() ==
searchData.getType()) ||
                    (searchData.getSource().empty() || currentData.getSource() ==
searchData.getSource()) ||
                    (searchData.getDest().empty() || currentData.getDest() ==
searchData.getDest()) ||
                    (searchData.getStat().empty() || currentData.getStat() ==
searchData.getStat()) ||
                    (searchData.getTrackNum() != 0 && currentData.getTrackNum() ==
searchData.getTrackNum())) {

                        currentData.display(!found);
                        cout << endl;
                        found = true;
                    }

                curr = curr->next;
            }
```

```
    if (!found)
            cout << "\nCourier not found. Please try again.\n" << endl;
}

void sortList(int sortCriteria) {
    if (!head || !head->next) {
        return;
    }
    bool swapped;
    Node* current;
    Node* lastSorted = NULL;

    do {
        swapped = false;
        current = head;

        while (current->next != lastSorted) {
            const Courier& currentData = current->data;
            const Courier& nextData = current->next->data;

            switch (sortCriteria) {
                case 1:
                            if (currentData.getName() > nextData.getName()) {
                    Courier temp = current->data;
                    current->data = current->next->data;
                    current->next->data = temp;
                    swapped = true;
                }

                    break;
                case 2:
```

```
            if (currentData.getType() > nextData.getType()) {
                Courier temp = current->data;
                current->data = current->next->data;
                current->next->data = temp;
                swapped = true;
            }
            break;
        case 3:
            if (currentData.getTrackNum() > nextData.getTrackNum()) {
                Courier temp = current->data;
                current->data = current->next->data;
                current->next->data = temp;
                swapped = true;
            }
            break;
        default:
            cout << "Invalid sort criteria.\n";
            return;
    }
    current = current->next;
  }

lastSorted = current;
} while (swapped);

switch (sortCriteria) {
  case 1:
    cout << "List sorted by name.\n";
    break;
  case 2:
    cout << "List sorted by parcel type.\n";
```

```
                    break;
                case 3:
                    cout << "List sorted by tracking number.\n";
                    break;
                default:
                    cout << "Invalid sort criteria.\n";
                    break;
            }
        }

        void displayList() {
            Node* current = head;
            int nodeNumber = 1;
                cout << endl;
            cout << left << setw(6) << "No." << setw(20) << "Tracking Number" <<
setw(20) << "Name"
                 << setw(20) << "Parcel Type" << setw(20) << "Source"
                 << setw(20) << "Destination" << setw(20) << "Status" << endl;
            cout << setfill('-') << setw(120) << "" << setfill(' ') << endl;

            while (current) {
                const Courier& currentData = current->data;
                cout << left << setw(6) << "[" + to_string(nodeNumber) + "]" << setw(20)
<< currentData.getTrackNum() << setw(20) << currentData.getName()
                     << setw(20) << currentData.getType() << setw(20) <<
currentData.getSource()
                     << setw(20) << currentData.getDest() << setw(20) <<
currentData.getStat() << endl;
                current = current->next;
                nodeNumber++;
            }
```

```
                }
};

int main() {
    ifstream inputFile("TEST.txt");
    if (!inputFile.is_open()) {
        cout << "Error opening file." << endl;
        return 1;
    }

    LinkedList courierList;

    string line;
    while (getline(inputFile, line)) {
        stringstream ss(line);
        string n, parcel, sourc, dest, stat;
        int track;

        if (getline(ss >> std::ws, n, ',') &&
            getline(ss >> std::ws, parcel, ',') &&
            getline(ss >> std::ws, sourc, ',') &&
            getline(ss >> std::ws, dest, ',') &&
            getline(ss >> std::ws, stat, ',') &&
            (ss >> track)) {

            Courier newCourier(n, parcel, sourc, dest, stat, track);
            courierList.insertBack(newCourier);
        }
    }

    inputFile.close();
```

```
    int choice;
        do {
            cout << "Menu:\n[1] Add a new courier\n[2] Delete a courier\n[3] Find a courier\n[4] Sort the list(ascending)\n[5] Display all couriers\n[0] Exit\n";
            cout << "Enter your choice: ";
            cin >> choice;

            switch (choice) {
                case 1: {
                    int insertChoice;
                    do {
                        cout << "\nAdd Courier Menu:\n[1] Insert at the beginning\n[2] Insert at the middle\n[3] Insert at the end\n[0] Back to Main Menu\nEnter your choice: ";
                        cin >> insertChoice;

                        switch (insertChoice) {
                            case 1: {
                                Courier newCourier = courierList.getNewCourier();
                                courierList.insertFront(newCourier);
                                break;
                            }
                            case 2: {
                                int position;
                                cout << "Enter the position to insert: ";
                                cin >> position;
                                Courier newCourier = courierList.getNewCourier();
                                courierList.insertMiddle(newCourier, position);
                                break;
                            }
                            case 3: {
```

```
                    Courier newCourier = courierList.getNewCourier();
                    courierList.insertBack(newCourier);
                    cout << "Courier added at the back.\n" << endl;
                    break;
                }
                case 0: {
                    cout << "Returning to Main Menu.\n" << endl;
                    break;
                }
                default: {
                    cout << "\nInvalid choice. Please try again.";
                    cout << endl;
                    break;
                }
            }
        } while (insertChoice != 0 && insertChoice != 1 && insertChoice != 2 &&
insertChoice != 3);
            break;
        }
        case 2: {
            int deleteChoice;
            do {
                cout << "\nDelete Courier Menu:\n[1] Delete at the beginning\n[2] Delete at
the middle\n[3] Delete at the end\n[0] Back to Main Menu\nEnter your choice: ";
                cin >> deleteChoice;

                switch (deleteChoice) {
                    case 1:
                        courierList.deleteFront();
                        break;
                    case 2: {
```

```
                int position;
                cout << "Enter the position to delete: ";
                cin >> position;
                courierList.deleteMiddle(position);
                break;
            }
            case 3:
                courierList.deleteBack();
                break;
            case 0:
                cout << "Returning to Main Menu.\n" << endl;
                break;
            default:
                cout << "Invalid choice. Please try again.\n";
                break;
        }
    } while (deleteChoice != 0 && deleteChoice != 1 && deleteChoice != 2 &&
deleteChoice != 3);
    break;
  }
  case 3: {
int searchOption;
    cout << "\nSearch by:\n";
    cout << "[1] Name\n[2] Parcel Type\n[3] Source\n[4] Destination\n[5] Status\n[6]
Tracking Number\n[0] Back to Main Menu\nEnter you choice: ";
    cin >> searchOption;

    if (searchOption == 0) {
        cout << "Returning to Main Menu.\n" << endl;
        break;
    } else if (searchOption < 1 || searchOption > 6) {
```

```
            cout << "Invalid option. Please try again." << endl;
            break;
        }

string searchKey;
Courier searchCourier;

    cout << "Enter the search key: ";
    cin.ignore();
    getline(cin, searchKey);

    switch (searchOption) {
        case 1:
            if (!searchKey.empty()) {
                searchCourier.setName(searchKey);
            }
            break;
        case 2:
            if (!searchKey.empty()) {
                searchCourier.setType(searchKey);
            }
            break;
        case 3:
            if (!searchKey.empty()) {
                searchCourier.setSource(searchKey);
            }
            break;
        case 4:
            if (!searchKey.empty()) {
                searchCourier.setDest(searchKey);
            }
```

```
                    break;
                case 5:
                    if (!searchKey.empty()) {
                        searchCourier.setStat(searchKey);
                    }
                    break;
                case 6:
                    if (!searchKey.empty()) {
                                try {
                            searchCourier.setTrackNum(stoi(searchKey));
                    }
                                catch (const invalid_argument& e) {
                                        cout << "Invalid tracking number. Please enter a valid
number. \n";
                                        break;
                                }
                            }
                    break;
                default:
                    break;
            }

                cout << "\nSearching...\n";
                courierList.findNode(searchCourier);
                break;
        }

            case 4: {
                int sortCriteria;
                bool sorting;
                do {
```

```
                cout << "\n[1] Sort by name\n[2] Sort by parcel type\n[3] Sort by tracking
number\n[0] Back to Main Menu\nEnter your choice: ";
                cin >> sortCriteria;

                if (sortCriteria == 0) {
                    cout << "Returning to Main Menu.\n" << endl;
                    break;
                } else if (sortCriteria < 1 || sortCriteria > 3) {
                    cout << "Invalid option. Please try again.\n";
                    sorting = false;
                    continue;
                }
                courierList.sortList(sortCriteria);
                cout << endl;
                sorting = true;
            } while (!sorting);
            break;
        }
        case 5:
            courierList.displayList();
            cout << endl;
            break;
        case 0:
            cout << "Exiting program.\n";
            break;
        default:
            cout << "Invalid choice. Please try again.\n" << endl;
            break;
    }
} while (choice != 0);
```

```
        return 0;
}
```