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 1 - DATA STRUCTURE AND OPERATIONS - SORTING AND SEARCHING

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 main objective for developing the courier management system is to ease the courier and customer. The courier management system is designed to make sure delivery services run smoothly and get packages to their destination safely and on time. This system aims to provide parcel tracking and tracing, customer satisfaction, customer self-service essentially helping and improving courier quality. The system also carries out a user-friendly interface for ease of use with a menu-driven system that guides users through the entire process until users choose to exit the program. The system implements a data hiding concept to ensure data integrity during file operation. Data hiding promises exclusive data access to class members only and maintains object integrity by prohibiting intended or unintended changes and disturbances.

There are few features in the courier management system which is

1. View courier available

Displaying all courier information available including name, parcel type, source, destination, status and tracking number.

2. Sort and arrange couriers

Sort and arrange couriers based on what customers want with three possible options which are sort by name, sort by parcel type, and sort by tracking number. Note that all three options are in ascending order and the algorithm used is a quick sort algorithm.

3. Search courier information

Can search any data that exists in the list whether name, parcel type, source, destination, status or tracking number. The algorithm used is a simple linear search.

4. Write the arranged and sorted data

The new sorted and arranged data about couriers will be stored into a new file name "sorted_couriers.txt".

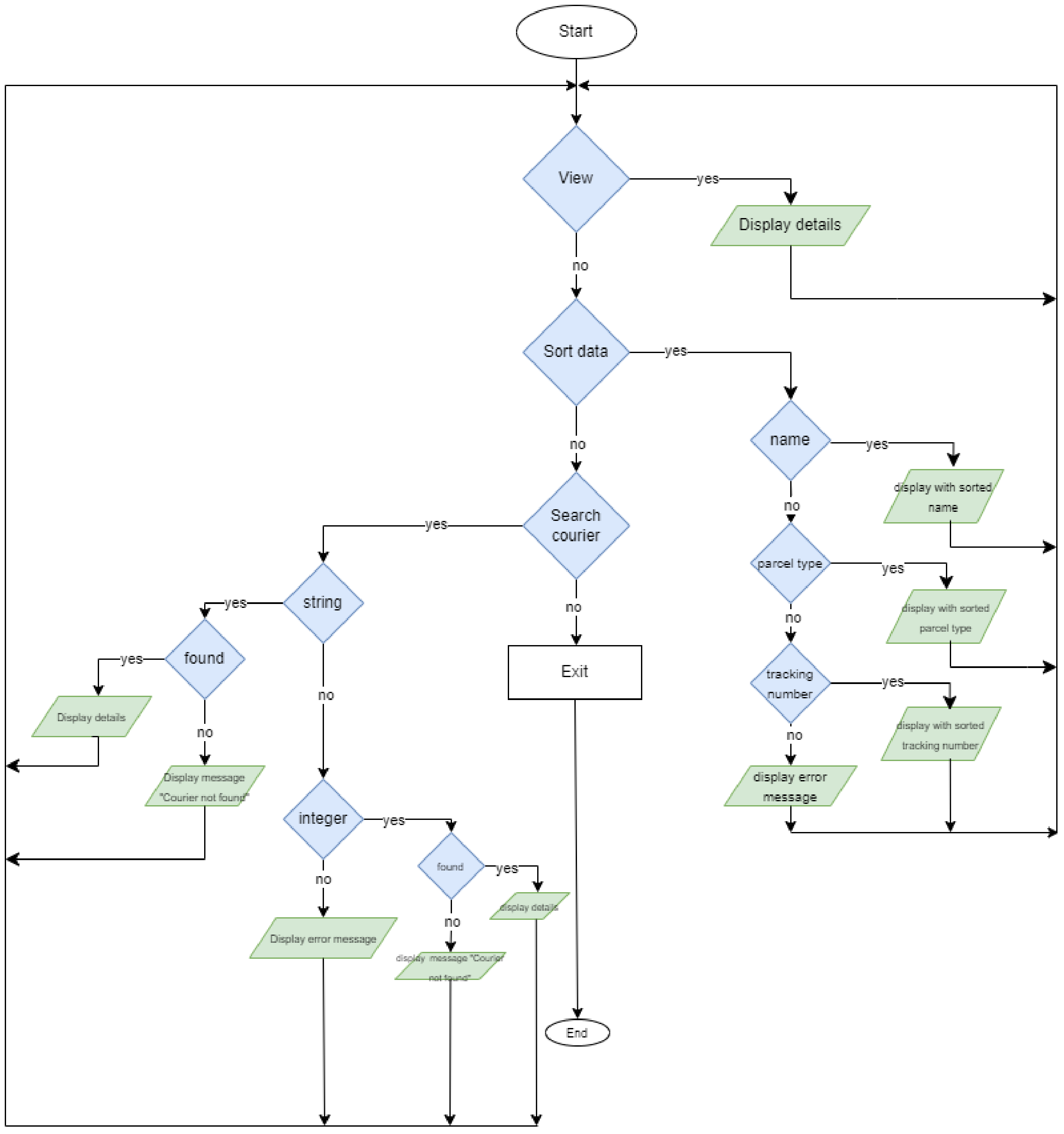
SYNOPSIS

This program is designed for organizing courier management systems efficiently, with features such as sorting and searching for specific courier details. The foundation of the program focuses around a Courier class, which encapsulates each courier's information such as name, parcel type, source, destination, status, and tracking number. This program can read and write data to or from files, display information in an organized form and conduct sorting and searching.

During execution, this program will display a menu with options for viewing all couriers, sort them according to certain criteria, search for specific couriers, and exit the program. The class design ensures encapsulation by offering ways for successfully interacting with the courier data.

DESIGN

Flowchart



DESIGN DESCRIPTION

MAIN FUNCTION

1. Read customer information from an input file (customer.txt).
2. Then, show the menu for view all data of customers option, sorting (ascending) option, searching option and exit option.

   2.1 Option 1: Display all data of customers from input file

   2.2 Option 2: Pass the customers data to Sorting function:

   - 2.2.1 Pass the customers' data to the quickSortStr() function and then pass it to partitionStrByName() function to sort the data by name, and then pass the sorted data to the display() function to display the sorting result.
   - 2.2.2 Pass the customers' data to the quickSortStr() function and then pass it to partitionStrByType() function to sort the data by name, and then pass the sorted data to the display() function to display the sorting result.
   - 2.2.3 Pass the customers' data to the quickSortInt() function and then pass it to partitionInt() function to sort the data by name, and then pass the sorted data to the display() function to display the sorting result.

   2.3 Option 3: Pass the customers data to Searching function:

   - 2.3.1 Pass the customers' data to the SearchString() function to search data by name, type of parcel, source, destination, or status delivery. Then pass the searched data to the display() function to display the search result.
   - 2.3.2 Pass the customers' data to the SearchInt() function to search data by tracking numbers. Then pass the searched data to the display() function to display the searching result.

   2.4 Option 4: Exit the system

   2.5 Ask users to choose the valid option.

3. End

SORTING FUNCTION

This program uses two sorting algorithms which are quick sort for sorting by integer (trackingNum) and quick sort for sorting by string (name or parcelType). It also uses a different partition function for each sorting process.

1. Get the option from the main function.
2. From the option, the data will be sorted by corresponding function either sort by name, parcel type or tracking number.

   2.1 The sorting function uses a quick sort technique to rearrange the customer data ascendingly based on the option that was chosen by the customer.

   2.2 Option 1: quickSortStr() function: sort data only for String type

   2.2.1 partitionStrByName(): sort data based on the name of the customer

   2.2.2 partitionStrByType(): sort data based on the parcel type of customer

   2.3 Option 2: quickSortInt() function: sort data only for Integer type

   2.3.1 partitionInt(): sort data based on the tracking numbers of the customer

SEARCHING FUNCTION

This program also implements two search functions which are SearchString() function for searching all the string attributes (name,parcelType,source,destination or status) and SearchInt() function for searching integer attributes (trackingNum).

1. Get the option from the main function.
2. From the option, the data will be searched by corresponding function either search by name, parcel type or tracking number.

   2.1 The searching function uses a compare search technique to search customers data based on the option that was chosen by the customer.

2.2 Option 1: SearchString() function: search data only for String type (name, source, destination, status, parcel type).
2.3 Option 2: SearchInt() function: search data only for Int type (tracking numbers).

DISPLAY FUNCTION

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

CODING

```
#include <iostream>
#include <fstream>
#include <string>
#include <iomanip>
const int size = 10;
using namespace std;

class Courier {
private:
   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; }

        int readFile(ifstream& file, Courier couriers[]) {
        if (!file.is_open()) {
        cout << "Error: Unable to open the file\n";
        return 0;
              }
              int m = 0;
        while (getline(file, couriers[m].name, ',')){
       getline(file, couriers[m].parcelType, ',') ;
       getline(file, couriers[m].source, ',');
       getline(file, couriers[m].destination, ',') ;
       getline(file, couriers[m].status, ',');
               file >> couriers[m].trackingNum;
         file.ignore();
         ++m;
              }
    return m;
  }

        void writeFile(string filename, Courier arr[]) {
        ofstream outFile(filename);

        if (!outFile.is_open()) {
            cout << "Error: Unable to open the file for writing\n";
            return;
                        }

            for (int i = 0; i < size; ++i) {
            outFile << arr[i].getName() << ',' << arr[i].getType() << ',' << arr[i].getSource() << ','
                        << arr[i].getDest() << ',' << arr[i].getStat() << ',' << arr[i].getTrackNum() << '\n';
            }

                        outFile.close();
}

    void display() const {
        cout << left << setw(25) << name
             << setw(15) << parcelType
             << setw(15) << source
             << setw(18) << destination
             << setw(16) << status
             << setw(15) << trackingNum << endl;
    }
};



int SearchString(string key, Courier arr[], int size) {
    for (int i = 0; i < size; i++) {
            if (arr[i].getName() == key || arr[i].getType() == key || arr[i].getSource() == key ||
                arr[i].getDest() == key || arr[i].getStat() == key) {
                return i;
            }
    }
return -1;
```

```
}

int SearchInt(int key, Courier arr[], int size) {
   for (int i = 0; i < size; ++i) {
          if (arr[i].getTrackNum() == key) {
             return i;
          }
   }
return -1;
}

bool compareStrByName(Courier& a, Courier& b) {
    return a.getName() < b.getName();
}

bool compareStrByType(Courier& a, Courier& b) {
    return a.getType() < b.getType();
}

void swap(Courier& a, Courier& b) {
    Courier temp = a;
    a = b;
    b = temp;
}

int partitionInt(Courier arr[], int first, int last) {
   int pivot = arr[last].getTrackNum();
   int bottom = first - 1, top = last;

   while (true) {
          while (arr[--top].getTrackNum() > pivot);
```

```
            while (arr[++bottom].getTrackNum() < pivot);
            if (bottom < top) {
                        swap(arr[bottom], arr[top]);
            }
            else {
              break;
            }
    }
    swap(arr[bottom], arr[last]);
    return bottom;
}

int partitionSortByName(Courier arr[], int first, int last) {
    Courier pivot = arr[last];
    int bottom = first - 1, top = last;

    while (true) {
        while (arr[--top].getName() > pivot.getName());
        while (arr[++bottom].getName() < pivot.getName());
        if (bottom < top) {
            swap(arr[bottom], arr[top]);
        } else {
            break;
        }
    }
    swap(arr[bottom], arr[last]);

    return bottom;
}

int partitionSortByType(Courier arr[], int first, int last) {
    Courier pivot = arr[last];
    int bottom = first - 1, top = last;

    while (true) {
        while (arr[--top].getType() > pivot.getType());
        while (arr[++bottom].getType() < pivot.getType());
        if (bottom < top) {
            swap(arr[bottom], arr[top]);
        } else {
            break;
        }
    }
    swap(arr[bottom], arr[last]);

    return bottom;
}


void quickSortInt(Courier arr[], int first, int last) {
            if (first < last) {
                        int pivot = partitionInt(arr, first, last);

            quickSortInt(arr, first, pivot - 1);
            quickSortInt(arr, pivot + 1, last);
            }
}

void quickSortStr(Courier arr[], int first, int last, bool sortByType) {
            if (first < last) {
                        int pivotIndex;
                        if (sortByType) {
```

```
                        pivotIndex = partitionSortByType(arr, first, last);
                }
                else {
                        pivotIndex = partitionSortByName(arr, first, last);
                }

        quickSortStr(arr, first, pivotIndex - 1, sortByType);
        quickSortStr(arr, pivotIndex + 1, last, sortByType);
        }
}

int main() {

    Courier couriers[size];
    ifstream file("customer.txt");

    for(int i = 0; i < size; i++){
    couriers[i].readFile(file, couriers);
    }

    int choice, index, intKey;;
    string key;

    do {
        cout << "Menu:\n"
                << "[1] View all couriers\n"
             << "[2] Sort couriers(ascending)\n"
             << "[3] Search courier\n"
             << "[4] Exit\n"
             << "Enter your choice: ";
        cin >> choice;
        switch (choice) {
            case 1:
                        cout << "\n| Name              | Parcel Type   | Source        | Destination    | Status
| Tracking Number |\n"
                     <<
"--------------------------------------------------------------------------------------------------------------
--\n";
                for (int i = 0; i < size; ++i) {
                    couriers[i].display();
                }
                cout << endl;
                break;

            case 2:
                cout << "\nSorting options:\n"
                     << "[1] Sort by Name\n"
                     << "[2] Sort by Parcel Type\n"
                     << "[3] Sort by Tracking Number\n"
                     << "Enter your sorting choice: ";
                int sortingChoice;
                        cin >> sortingChoice;

                        switch (sortingChoice) {
                                    case 1:
                                    quickSortStr(couriers, 0, size - 1, false);
                                    break;

                                    case 2:
                                    quickSortStr(couriers, 0, size - 1, true);
                                    break;
                    case 3:
                    quickSortInt(couriers, 0, size -1);
                    break;

                    default:
                    cout << "Invalid sorting choice.\n";
                    continue;
            }
                cout << "\nCouriers sorted.\n";
            cout << "\n| Name              | Parcel Type   | Source        | Destination    | Status          |
Tracking Number |\n"
                 <<
"------------------------------------------------------------------------------------------------------------------
--\n";
            for (int i = 0; i < size; ++i) {
              couriers[i].display();
            }
            break;

        case 3:
            cout << "\nChoose search type:\n"
                    << "[1] Search by string\n"
                 << "[2] Search by integer\n"
                 << "Enter your choice: ";
            int searchType;
            cin >> searchType;

                switch (searchType) {
                    case 1: {
                        cout << "\nEnter the keyword to search: ";
                        cin >> key;
                        index = SearchString(key, couriers, size);
                        break;
                    }
                    case 2: {
                        cout << "\nEnter the number to search: ";
                        cin >> intKey;
                        index = SearchInt(intKey, couriers, size);
                        break;
                    }
                    default:
                    cout << "Invalid search type choice.\n";
                    index = -1;
                }
                    if (index != -1) {
                    cout << "\nCourier found at list number " << index+1 << ":\n";
                        cout << "\n| Name             | Parcel Type   | Source         |
Destination    | Status          | Tracking Number |\n"
                         <<
"-----------------------------------------------------------------------------------------------------------------
--\n";
                    couriers[index].display();
                        }
                        else {
                    cout << "Courier not found.\n";
                        }
                    break;

            case 4:
              for(int i = 0; i < size; i++)
              couriers[i].writeFile("sorted_couriers.txt", couriers);
```

```
                cout << "\nData has been written to sorted_couriers.txt\n";
                cout << "Exiting program.\n";
                break;

            default:
                cout << "Invalid choice. Please enter a valid option.\n";
        }

    } while (choice != 4);

    return 0;
}
```