FINAL PROJECT- DATA STRUCTURES

ALI HASSAN AHMAD

ROLL NO.19F0324 SECTION: 3C

PROJECT DISCRIPTION REPORT

MARQUEE BOOKING PROGRAM

LINES OF CODE : 1050

PROGRAMMING LANGUAGE : C++

GUI USED : MS-DOS CONSOLE

WORKING HOURS : 48 hrs.

DATA STRUCTURES USED:

1. QUEUE -TAKING ORDERS
2. GRAPHS -FINDING SHORTEST PATH
3. LINKED LIST-MAINITING FILTERED DATA OF MARQUEES
4. STACK -BILLING,PROMOCODE
5. ARRAY -HASHING-SORETING

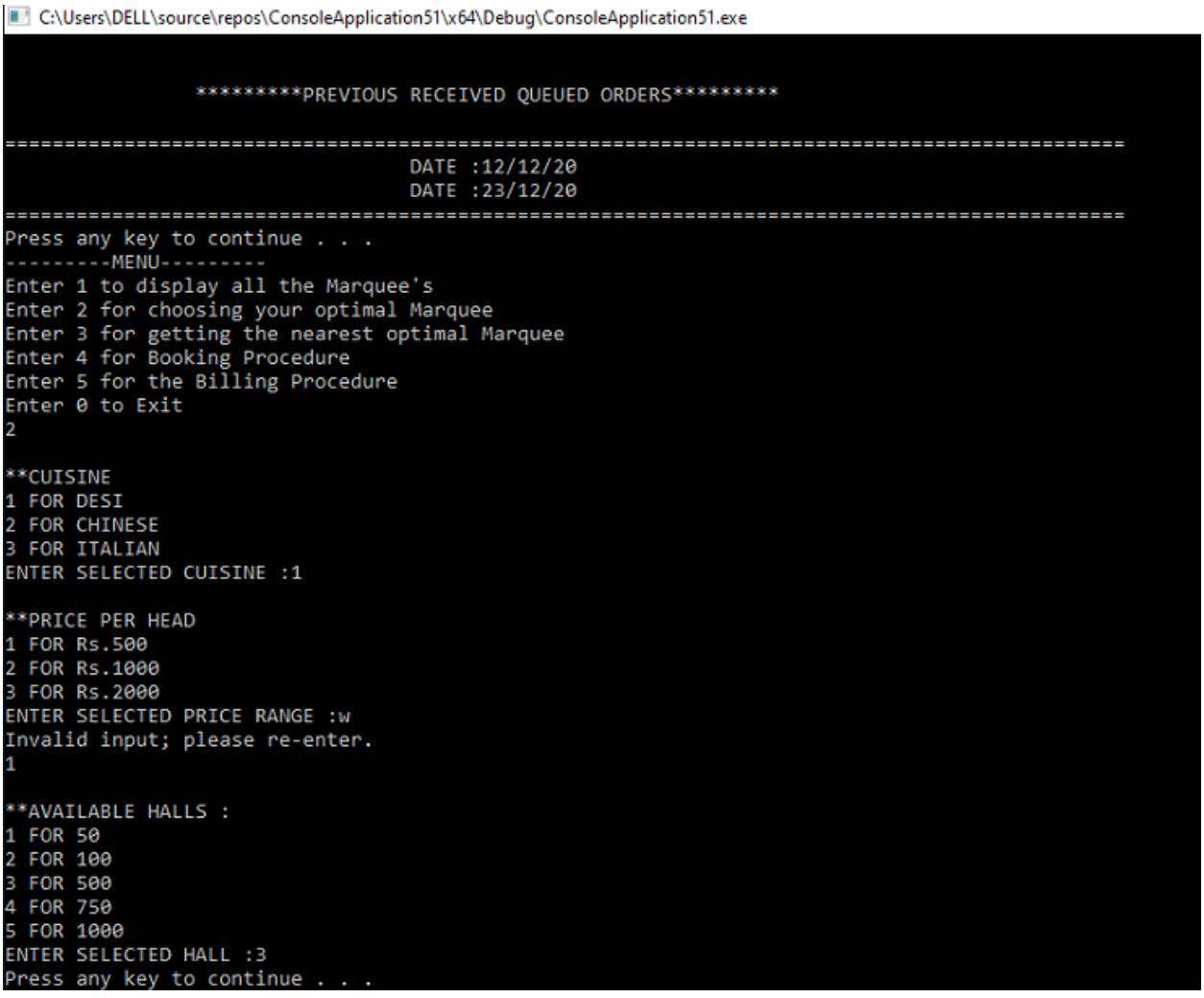
DATA IS STORED IN TEXT FILES

1. SPAM.TXT
2. LOCATION.TXT
3. ORDERS.TXT
4. PROMOCODE.TXT

CPP & TEXT FILES IS ATTACHED WITH THE DOCUMENT

SCREENSHOTS

#1



#2

C:\Users\DELL\source\repos\ConsoleApplication51\x64\Debug\ConsoleApplication51.exe

```
**********LIST EMPTY**********


******The most nearer Marquees from your location having selected attributes******* :



Aflatoon Marquee only 0 km away :

FOOL Marquee only 4 km away :

Balloon Marquee only 5 km away :

COOL Marquee only 6 km away :

AliSakoon Marquee only 6 km away :

MinahilMoon Marquee only 7 km away :


---------MENU---------
Enter 1 to display all the Marquee's
Enter 2 for choosing your optimal Marquee
Enter 3 for getting the nearest optimal Marquee
Enter 4 for Booking Procedure
Enter 5 for the Billing Procedure
Enter 0 to Exit
5
ENTER PROMOCODE IF U HAVE ANY,ELSE PRESS x : LWQ510
CONGRATULATIONS! You got a discount of 30%.
ENTER THE NUMBER OF PEOPLE : 276
---------------BILL-CALCULATION------------------
ACTUAL BILL : 138000
PROMOCODE DISCOUNT : 30
BILL AFTER PROMOCODE : 96600
Press any key to continue . . .
---------MENU---------
Enter 1 to display all the Marquee's
Enter 2 for choosing your optimal Marquee
Enter 3 for getting the nearest optimal Marquee
Enter 4 for Booking Procedure
Enter 5 for the Billing Procedure
Enter 0 to Exit
```

CODE

```
#include<iostream>

#include<fstream>

#include<string>

#include<iomanip>
```

```
using namespace std;
int NoOfMarquee = 0;
int c;
int read_input()
{
   double input = -1;
   bool valid = false;
   do
   {

      cin >> input;
      if (cin.good())
      {
         //everything went well, we'll get out of the loop and return the value
         valid = true;
      }
      else
      {
         //something went wrong, we reset the buffer's state to good
         cin.clear();
         //and empty it
         cin.ignore(numeric_limits<streamsize>::max(), '\n');
         cout << "Invalid input; please re-enter." << endl;
      }
   } while (!valid);

   return (input);
}
bool contactValidation(string a)
{
```

```
    int j = 0;
    for (int i = 0; i != a.length(); i++)
    {
        if ((a[i] >= '0' && a[i] <= '9') || (a[i] == '-'))
        {
            j++;
        }
    }

    if (a.length() != 12)
    {
        return 0;
    }
    if (j == a.length())
    {
        return 1;
    }




}
bool ValidString(string s)
{
    int j = 0;
    for (int i = 0; i <= s.length(); i++)
    {
        if ((s[i] >= 'A' && s[i] <= 'Z') || (s[i] >= 'a' && s[i] <= 'z') || s[i] == ' ')
        {
            j++;
        }
```

```
    }
    if (j == s.length())
    {
        return 1;
    }
    else
    {
        return 0;
    }



}
struct Entity
{
    string name;
    string contactNo;
    string date;
};
struct QUEUE
{
    QUEUE* link;
    Entity data;
};
class SinglyQueue
{
public:
    SinglyQueue()
    {
```

```
    this->queFront = NULL;
    this->queRear = NULL;
    read.open("orders.txt", ios::app);
    write.open("orders.txt", ios::app);
    if (!read.is_open())
        cout << "ORDERS FILE MISSING" << endl;
    else
        this->LoadOrderfromFileinQueue();
}
void LoadOrderfromFileinQueue()
{
    string name;
    string date;
    long long int number;
    while (read.good())
    {
        getline(read, name);

        read >> number;
        read.ignore();
        getline(read, date);
        read.ignore();
        if (name == "")
        {
            return;
        }

        if (this->queFront == NULL)
        {
            QUEUE* newnode = new QUEUE;
```

```
            newnode->link = NULL;
            this->queFront = newnode;
            this->queRear = newnode;
            //input
            newnode->data.name = name;
            newnode->data.contactNo = number;
            newnode->data.date = date;
        }
        else
        {
            if (!this->isEmptyQueue())
            {
                QUEUE* newnode = new QUEUE;
                this->queRear->link = newnode;
                newnode->link = NULL;
                this->queRear = newnode;
                //input
                newnode->data.name = name;
                newnode->data.contactNo = number;
                newnode->data.date = date;
            }
        }
    }
}
void addQue(string name, string number, string date)
{
    if (!this->checkAvailability(date))
    {
        cout << endl << endl << "\t\t********** SORRY! THIS DATE IS UNAVAILABLE
**********" << endl;
```

```
    return;
}
if (this->queFront == NULL)
{
    QUEUE* newnode = new QUEUE;
    newnode->link = NULL;
    this->queFront = newnode;
    this->queRear = newnode;
    //input
    newnode->data.name = name;
    newnode->data.contactNo = number;
    newnode->data.date = date;
    //in file
    write << name << endl << number << endl << date << endl;
}
else
{
    if (!this->isEmptyQueue())
    {
        QUEUE* newnode = new QUEUE;
        this->queRear->link = newnode;
        newnode->link = NULL;
        this->queRear = newnode;
        //input
        newnode->data.name = name;
        newnode->data.contactNo = number;
        newnode->data.date = date;
        //in file
        write << name << endl << number << endl << date << endl;
    }
```

```
    }
}
void deleteQue()
{
    if (this->isEmptyQueue())
    {
        cout << "QUE EMPTY" << endl;
        cout << "TERMINATED" << endl;
        exit(0);
    }
    string item = 0;
    item = this->queFront->data.name;
    //delete first
    QUEUE* p = this->queFront;
    this->queFront = queFront->link;
    free(p);
    cout << endl << "DELETED " << item << "'s ORDER FROM THE QUEUE" << endl;

}
void seeQueue()
{
    QUEUE* temp = this->queFront;
    while (temp != NULL)
    {
        cout << setw(40) << "DATE :" << temp->data.date << endl;
        temp = temp->link;
        if (temp == NULL)
            break;
    }
}
```

```
    bool checkAvailability(string date)
    {
        QUEUE* temp = this->queFront;
        while (temp != NULL)
        {
            if (temp->data.date == date)
            {
                return false;
            }
            temp = temp->link;
        }
        return true;
    }
private:
    QUEUE* queFront;
    QUEUE* queRear;
    bool isEmptyQueue()
    {
        if (this->queFront == NULL)
        {
            return true;
        }
        return false;
    }
    bool isFullQue()
    {
        //dynamically allocated
        //never full
        return true;
    }
```

```
    ifstream read;
    ofstream write;
};
class Marquee
{
public:
    Marquee(int i)
    {

    }
    Marquee()
    {
        this->totalPriceRanges = 3;
        this->totalCuisines = 3;
        this->totalHalls = 5;

        this->Pricerange = new bool[this->totalPriceRanges];
        this->cuisine = new bool[this->totalCuisines];
        this->NoOfPeopleHalls = new bool[this->totalHalls];

        for (int i = 0; i < this->totalPriceRanges; i++)
            this->Pricerange[i] = false;
        for (int i = 0; i < this->totalCuisines; i++)
            this->cuisine[i] = false;
        for (int i = 0; i < this->totalHalls; i++)
            this->NoOfPeopleHalls[i] = false;

    }
    int MarqueeNo;
```

```
int size;
string name;
string Address;
string city;
int Parking;
string Facilities[3];
bool* Pricerange;
bool* cuisine;
bool* NoOfPeopleHalls;
int totalPriceRanges;
int totalCuisines;
int totalHalls;
string Location;

void operator=(Marquee& obj)
{
   this->totalCuisines = obj.totalCuisines;
   this->totalHalls = obj.totalHalls;
   this->totalPriceRanges = obj.totalPriceRanges;
   //
   this->name = obj.name;
   this->Address = obj.Address;
   this->city = obj.city;
   this->Parking = obj.Parking;
   this->size = obj.size;
   for (int i = 0; i < obj.totalCuisines; i++)
   {
      this->Facilities[i] = obj.Facilities[i];
      this->cuisine[i] = obj.cuisine[i];
      this->Pricerange[i] = obj.Pricerange[i];
```

```
        }
        for (int i = 0; i < obj.totalHalls; i++)
        {
            this->NoOfPeopleHalls[i] = obj.NoOfPeopleHalls[i];
        }
        this->MarqueeNo = obj.MarqueeNo;
        this->Location = obj.Location;
    }
};
struct LinkedList
{
    Marquee M;
    LinkedList* next;
};
class Hash
{
    Marquee* array;
    int key;
public:
    Hash(int s)
    {
        array = new Marquee[s];
    }
    void hashFuntion(int MarqueeNo, Marquee M)
    {
        array[MarqueeNo - 1] = M;
    }
    Marquee returnHash(int MarqueeNo)
    {
        return array[MarqueeNo - 1];
```

```
    }
};
struct edge {
    string src;
    string dest;
    int weight;
    static edge* sort(int vertices, edge* e)
    {
        for (int i = 0; i < vertices; i++)
        {
            for (int j = 0; j < vertices; j++)
            {
                if (e[i].weight < e[j].weight)
                {
                    swap(e[i], e[j]);
                }
            }
        }
        return e;
    }
};
class Graph
{
public:
    Graph(int s)
    {

        this->vertices = s;
        graph = new int* [s];
```

```
    for (int i = 0; i < s; i++)
    {
        graph[i] = new int[s];

    }

}
void reading()
{
    ifstream myfile;
    myfile.open("location.txt");



    for (int i = 0; i < vertices; i++)
    {
        for (int j = 0; j < vertices; j++)
        {
            myfile >> graph[i][j];

        }


    }



}
int mindistance(int* distance, bool* visited, int vertices)
{
```

```
    int min = 999;
    int minindex = 0;
    for (int i = 0; i < vertices; i++)
    {
        if (!visited[i] && distance[i] < min)
        {
            min = distance[i];
            minindex = i;
        }


    }
    return minindex;
}
edge* Dijkstra(int vertices, int src, Hash h)
{
    Marquee M = h.returnHash(src + 1);


    bool* visited;
    visited = new bool[vertices];

    int* distance;
    distance = new int[vertices];
    for (int i = 0; i < vertices; i++)
    {
        visited[i] = 0;


        distance[i] = 999;
```

```
}




distance[src] = 0;
for (int i = 0; i < vertices - 1; i++)
{
   int min = mindistance(distance, visited, vertices);
   visited[min] = true;
   for (int k = 0; k < vertices; k++)
   {
      if (graph[min][k] != 0 && !visited[k])
      {
         if (distance[min] + graph[min][k] < distance[k])
         {
            distance[k] = distance[min] + graph[min][k];
         }


      }


   }


}
edge* e = new edge[vertices];
for (int i = 0; i < vertices; i++)
{
   e[i].src = M.Location;


   Marquee M2 = h.returnHash(i + 1);
```

```
            e[i].dest = M2.name;
            e[i].weight = distance[i];

        }

        for (int i = 0; i < vertices; i++)
        {
            e[i].src = M.Location;

            Marquee M2 = h.returnHash(i + 1);
            e[i].dest = M2.name;
            e[i].weight = distance[i];

        }
        e = edge::sort(vertices, e);

        return e;


    }
private:
    int** graph;
    int vertices;
};
class SLL
{

public:
    SLL()
    {
```

```
    this->head = NULL;
    this->tail = NULL;
}
void loadDataFromFile(string fileName)
{

    ifstream myfile;
    myfile.open(fileName);
    if (!myfile)
    {
        cout << "File doesnot exists :\n";
    }
    Marquee M;
    while (!myfile.eof())
    {
        if (myfile.eof())
            break;
        getline(myfile, M.name, '.');

        getline(myfile, M.Address, '.');

        myfile >> M.city;
        myfile >> M.Parking;
        for (int i = 0; i < 3; i++)
        {
            myfile >> M.Facilities[i];
        }
        for (int i = 0; i < 3; i++)
        {
```

```
            myfile >> M.Pricerange[i];
        }
        for (int i = 0; i < M.totalCuisines; i++)
        {
            myfile >> M.cuisine[i];
            //cout << M.cuisine[i] << " - ";
        }
        for (int i = 0; i < M.totalHalls; i++)
        {
            myfile >> M.NoOfPeopleHalls[i];
        }

        myfile >> M.Location;
        NoOfMarquee++;
        M.MarqueeNo = NoOfMarquee;

        AddEnd(M);


    }

}
Hash traverse()

{
    Hash h(this->size);

    //TEMP FUNTION FOR TESTING -
    LinkedList* temp = this->head;
    while (temp != NULL && temp->next != NULL)
```

```
    {
        h.hashFuntion(temp->M.MarqueeNo, temp->M);



        temp = temp->next;



    }
    return h;
}
SLL& filterMatch()
{
    //return another linked list having selected attributes
    SLL obj1 = this->setCuisine();
    SLL obj2 = this->setPriceRanges();
    SLL obj3 = this->setHall();
    SLL common;
    LinkedList* temp1 = obj1.head;
    LinkedList* temp2 = obj2.head;
    LinkedList* temp3 = obj3.head;
    while (temp1 != NULL)
    {
        while (temp2 != NULL)
        {
            while (temp3 != NULL)
            {
                if (temp1->M.name == temp2->M.name && temp2->M.name == temp3->M.name)
                {
                    common.AddEnd(temp1->M);
                }
```

```
            temp3 = temp3->next;
        }
        temp3 = obj3.head;
        temp2 = temp2->next;
    }
    temp2 = obj2.head;
    temp1 = temp1->next;
  }

  return common;
}
SLL& Dis(SLL& l, Hash h)
{
  SLL selected;
  Graph g(l.size - 1);
  int choice;
  g.reading();
  cout << "Please Select your location :\n :";
  cout << "Enter 1 for Abdullah Pur :\n";
  cout << "Enter 2 for Canal Road :\n";
  cout << "Enter 3 for Eden Valley :\n";
  cout << "Enter 4 for D Ground :\n";
  cout << "Enter 5 for Kohinoor :\n";
  cout << "Enter 6 for Allah Hu Chownk :\n";
  cout << "Enter 7 for ChenOne Road :\n";
  cout << "Enter 8 for Nishatabad :\n";
  cout << "Enter 9 for CanalRoad :\n";
  cout << "Enter 10 for Kashmir Bridge :\n";
  choice = read_input();
  while (choice > 10 || choice <= 0)
```

```
    {
        cout << "Invalid Input ! :";
        choice = read_input();
    }
    edge* e2 = g.Dijkstra(l.size - 1, choice - 1, h);
    if (!this->Empty())
    {
        LinkedList* temp = this->head;
        LinkedList* temp1 = this->head;
        cout << "\n\n******The most nearer Marquees from your location having selected
attributes******* :\n";
        cout << endl << endl;
        for (int i = 0; i < l.size; i++)
        {
            while (temp1 != NULL)
            {
                if (e2[i].dest == temp1->M.name)
                {
                    selected.AddEnd(temp1->M);
                    cout << e2[i].dest << " only " << e2[i].weight << " km " << "away :" << endl;
                }
                temp1 = temp1->next;
            }
            temp1 = this->head;
        }
        cout << endl << endl;
    }
    return selected;//FINAL MARQUEES WILL BE IN THIS OBJECT
}
void DisplayMatches()
```

```
{
   if (!this->Empty())
   {
      LinkedList* temp = this->head;
      while (temp != NULL && temp->next != NULL)
      {
         cout << "\t\t================================================\n\n";
         cout << "NAME : " << temp->M.name << endl;
         cout << "ADDRESS : " << temp->M.Address << endl;
         cout << "CITY : " << temp->M.city << endl;
         cout << "CAR PARKING : " << temp->M.Parking << " CARS" << endl;
         cout << "OTHER FACILITIES : " << endl;
         for (int i = 0; i < 3; i++)
         {
            cout << temp->M.Facilities[i] << endl;
         }
         cout << "\t\t================================================\n";
         temp = temp->next;
      }
   }
}
int selectionMenu()
{
M5:
   int choice = 0;
   cout << "PRESS 1 TO BOOK A MARQUEES" <<
      "PRESS 2 TO EXIT" << endl <<
      "PLEASE ENTER :";
   cin >> choice;
   if (choice == 1)
```

```
        return choice;
    else if (choice == 2)
        exit(0);
    else
        goto M5;
}
void setOrder()
{
    SinglyQueue orders;
    if (this->selectMarqee()) //true
    {
        string name;
        string number = 0;
        string date;
        cout << "ENTER YOUR NAME :";
        getline(cin, name);
        bool b = ValidString(name);
        while (b == 0)
        {
            cout << "Invalid Input please enter again" << endl;
            getline(cin, name); b = ValidString(name);




        }
        cout << "ENTER YOUR CONTACT No in the following format 03xx-xxxxxxx:";
        cin >> number;
        bool c = contactValidation(number);
        while (c == 0)
        {
```

```
            cout << "Invalid Input please enter again" << endl;
            getline(cin, number); c = ValidString(number);




        }
        cout << "ENTER YOUR -  Date/Month/Year. :";
        cin >> date;


        orders.addQue(name, number, date);
        cout << endl << "*****************PLEASE NOTE*****************" << endl;
        cout << "YOUR ORDER HAS BEEN ADDED IN THE QUEUE - CURRENT QUEUE" << endl;
        this->seePreveousOrder();
        cout << endl << "WE WILL FORWARDED YOUR ORDER TO THE SELECTED MARQUEE ON YOUR
TURN" << endl <<
            "WILL BE NOTIFIYING YOU ABOUT THE CONFORMATION ON YOUR CONTACT NUMBER SOON"
<< endl;
        cout << endl << endl << "\t\tKEEP VISITING US" << endl;
        exit(0);
      }
      else
      {
        cout << "***************THANK YOU FOR USING OUR SOFTWARE*********************"
<< endl;
        cout << "***************COME AGAIN FOR OUR BEST SERVICES*********************"
<< endl;
        exit(0);
      }
   }
   void seePreveousOrder()
   {
      SinglyQueue orders;
```

```
      cout << endl << endl << "\t\t*********PREVIOUS RECEIVED QUEUED ORDERS*********" << endl
<< endl;
      cout <<
"================================================================================
==========" << endl;
      orders.seeQueue();
      cout <<
"================================================================================
==========" << endl;
   }
private:
   LinkedList* head;
   LinkedList* tail;
   int size = 0;
public:
   int menuCuisine()
   {
      int choice = 0;
      cout << "\n**CUISINE " << endl <<
         "1 FOR DESI" << endl <<
         "2 FOR CHINESE" << endl <<
         "3 FOR ITALIAN" << endl <<
         "ENTER SELECTED CUISINE :";
      choice = read_input();
      while (choice <= 0 && choice >= 4)
      {
         cout << "Invalid input Please enter again";
         choice = read_input();
      }

      return choice;
   }
```

```
int menuPriceRange()
{
   int choice = 0;
   cout << "\n**PRICE PER HEAD " << endl <<
      "1 FOR Rs.500 " << endl <<
      "2 FOR Rs.1000" << endl <<
      "3 FOR Rs.2000" << endl <<
      "ENTER SELECTED PRICE RANGE :";

   choice = read_input();
   while (choice <= 0 && choice >= 4)
   {
      cout << "Invalid input Please enter again";
      choice = read_input();
   }

   c = choice;
   return choice;
}
int menuHallRange()
{
   int choice = 0;
   cout << "\n**AVAILABLE HALLS : " << endl <<
      "1 FOR 50 " << endl <<
      "2 FOR 100" << endl <<
      "3 FOR 500" << endl <<
      "4 FOR 750" << endl <<
      "5 FOR 1000" << endl <<
      "ENTER SELECTED HALL :";
   choice = read_input();
```

```
    while (choice <= 0 && choice >= 4)
    {
        cout << "Invalid input Please enter again";
        choice = read_input();
    }

    return choice;
}
void AddEnd(Marquee& M)
{
    LinkedList* new_node;
    new_node = new LinkedList;
    if (this->head == NULL)
    {



        this->size++;

        new_node->M = M; //i guess - their are pointers -operator overloading is required
        new_node->next = NULL;
        this->head = new_node;
        this->tail = new_node;
    }
    else
    {
        this->size++;
```

```
        LinkedList* temp;
        new_node->M = M; // Link the data part
        new_node->next = NULL;
        temp = head;
        // Traverse to the last node
        while (temp != NULL && temp->next != NULL)
            temp = temp->next;
        temp->next = new_node;
        this->tail = new_node;
    }

}
SLL& setCuisine()
{//FUNTION : Returns : object of linked List -
 //this object's linked list will have the selected cuisine marqees only in its
    SLL cuisineList;
    LinkedList* temp = this->head;
    int choice = menuCuisine();
    while (temp != NULL)
    {
        if (temp->M.cuisine[choice - 1] == true)
        {
            cuisineList.AddEnd(temp->M);
        }
        temp = temp->next;
    }

    return cuisineList;
}
```

```
SLL& setPriceRanges()
{//FUNTION : Returns : object of linked List -
 //this object's linked list will have the selected Price Range marqees only in its
   SLL PriceRangeList;
   LinkedList* temp = this->head;



   int choice = menuPriceRange();


   while (temp != NULL)
   {
      if (temp->M.Pricerange[choice - 1] == true)
      {
         PriceRangeList.AddEnd(temp->M);
      }
      temp = temp->next;
   }
   return PriceRangeList;
}
SLL& setHall()
{
   SLL hallList;
   LinkedList* temp = this->head;


   int choice = menuHallRange();


   while (temp != NULL)
   {
      if (temp->M.NoOfPeopleHalls[choice - 1] == true)
      {
```

```
            hallList.AddEnd(temp->M);
        }
        temp = temp->next;
    }
    return hallList;
}
int Menu()
{
    int opt = 0;
    cout << "---------MENU---------" << endl;
    cout << "Enter 1 to display all the Marquee's" << endl;
    cout << "Enter 2 for choosing your optimal Marquee" << endl;
    cout << "Enter 3 for getting the nearest optimal Marquee" << endl;
    cout << "Enter 4 for Booking Procedure" << endl;
    cout << "Enter 5 for the Billing Procedure" << endl;
    cout << "Enter 0 to Exit " << endl;
    cin >> opt;
    return opt;
}
bool Empty()
{
    if (this->head == NULL)
        return true;
    cout << "**********LIST EMPTY**********" << endl;
    return false;
}
bool selectMarqee()
{
    int choice = 0;
    string sMarquee;
```

```
        this->selectionMenu();
        do {

            cout << "Enter Name Of Marquee :";
            cin.ignore();
            getline(cin, sMarquee);
            sMarquee = "\n" + sMarquee;
            LinkedList* temp = this->head;
            while (temp != NULL)
            {
                if (temp->M.name == sMarquee)
                {
                    return true;
                }
                temp = temp->next;
            }
            cout << "PRESS 1 TO ENTER NAME AGAIN " << endl << "PRESS 2 TO EXIT";
            cin >> choice;
        } while (choice == 1);
        exit(0);
    }//return true if selected
};
class PromoandBill
{
    int dis = 1;
    int noofpeople;
    double bill;
    int packageopt;
    int prange;
    int  pstack[5];
```

```
    string prostack[5];
    int length;
    int top;
    double fbill;
public:

    PromoandBill(int a)
    {
        packageopt = a;
        length = 5;
        top = -1;
    }



    void readpromo()
    {

        string r;
        ifstream myfile;
        myfile.open("promo.txt");
        if (!myfile)
        {
            cout << "File doesnot exists :\n";
        }
        while (!myfile.eof())
        {
            int d = 0;
            if (myfile.eof())
                break;
            myfile >> r;
```

```
        myfile >> d;

        Push(d, r);
    }



}
void getbillandpromo()
{
    if (packageopt == 1)
        prange = 500;
    if (packageopt == 2)
        prange = 1000;
    if (packageopt == 3)
        prange = 2000;

}

void Push(int x, string s)
{

    if (top == length - 1)
    {
        cout << "OVERFLOW!" << endl;
    }
    else
    {
        top++;
```

```
        pstack[top] = x;
        prostack[top] = s;

    }
}
void enterpromo()
{
    readpromo();
    string str;
    cout << "ENTER PROMOCODE IF U HAVE ANY,ELSE PRESS x : ";
    cin >> str;

    if (str == "x")
        return;
    else
        for (int i = 0; i < 5; i++)
        {
            Pop(str);

        }
    if (dis == 1)
    {
        cout << "YOU ENTERED AN INVALID PROMOCODE" << endl;
    }
    cout << "ENTER THE NUMBER OF PEOPLE : ";
    noofpeople = read_input();
    getbillandpromo();
    bill = noofpeople * prange;
    cout << "----------BILL-CALCULATION---------- " << endl;
    cout << "ACTUAL BILL : " << bill << endl;
```

```
    cout << "PROMOCODE DISCOUNT : " << dis << endl;
    fbill = bill * dis / 100;
    bill = bill - fbill;
    cout << "BILL AFTER PROMOCODE : " << bill << endl;
}

void Pop(string d)
{
    int z; string s;
    if (top == -1)
    {
        cout << "UNDERFLOW !" << endl;
    }
    else
    {
        z = pstack[top];
        s = prostack[top];

        top--;
        if (d == s)
        {
            dis = z;
            cout << "CONGRATULATIONS! You got a discount of " << dis << "%. " << endl;

        }

    }

}
```

```
};
int main()
{
   int opt = 1;
   SLL List;

   SLL finalM;
   SLL common;
   List.seePreveousOrder();
   List.loadDataFromFile("spam.txt");
   SLL common3;
   system("pause");
   Hash h = List.traverse();

   while (opt != 0) {
      opt = List.Menu();
      if (opt == 1)
      {
         List.DisplayMatches();
      }
      if (opt == 2)
      {
         common = List.filterMatch();
         system("pause");
         common.DisplayMatches();
         system("pause");
      }
      if (opt == 3)
      {
         finalM = common.Dis(List, h);
```

```
        }
        if (opt == 4)
        {
            finalM.setOrder();
        }



        if (opt == 5)
        {



            PromoandBill p(c);



            p.enterpromo();
            system("pause");

        }
        if (opt == 0)
        {
            return 0;
        }
    }
}
```