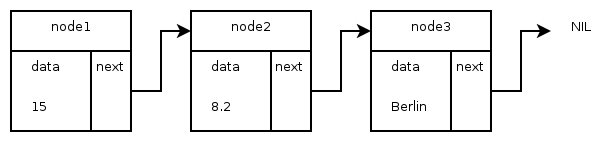
Course work:

Single linked list



Sheyla Ruiz-Gómez Ferreira

Synthesis and Analysis of Algorithms

University of Rousse

Index

Introduction to task statement ------------------------------------------------------------ 2

Used material ------------------------------------------------------------------------------- 2

General description of the classes ------------------------------------------------------- 3

Class Node --------------------------------------------------------------------------------- 3

Class List --------------------------------------------------------------------------------- 6

Main Program ----------------------------------------------------------------------------- 9

Source code ------------------------------------------------------------------------------- 10

Screenshots from the program ---------------------------------------------------------- 18

1. Introduction to the task statement

Define a single linked list, each node of which has the following structure:

* Departure airport
* Destination airport
* Date
* Duration of the flight (in hours)
* Number of stops
* Airline name
* Price
* Link to the next node

The program must perform the following operations:

* Add node in the front of the list
* Remove the end node from the list
* Sort the list in increasing order (using the field “‘Price”
* Display the number of the nodes and the list

1. Used material

The tool used to carry out the task has been the Visual Studio Code with its respective C++ packages for the development of the source code. And in addition, the terminal that the system has for its compilation and execution has also been used.

On the other hand, for references and doubts, the notes of the subject have been used.

1. General description of the classes

To solve this problem, two classes have been used: a ‘**Node**’ class and a ‘**List**’ class.

These files at the same time are broken down into two: a ‘**.h**’ and a ‘**.cc**’, where the first contains the attributes and headers of the defined methods, and the second contains the development of the code for said methods.

In addition, we have a ‘**main.cc**’, that is the main program that contains the data with which the user will relate.

1. Class Node

We are going to describe the methods and functions used in the ‘**Node**’ class.

* **Function**: Node(void)
  + Description: It is the default constructor of the class. It is responsible for creating an empty node to which the pointer of the next node is NULL.
  + Parameters: None
  + Global variables: Node\* next\_
  + Called functions or procedures: None
* **Function**: Node(const std::string& DeparAir, const std::string& DestAir, const std::string& date, const int& duration, const int& stops, const std::string AirlineName, const double& price)
  + Description: Constructor with parameters of the class. It is responsible for creating a node where we initialize with data and to which the pointer of the next node is NULL.
  + Parameters: The same as we can see within the parentheses of the method header.
  + Global variables: Attributes necessary to define the flight reservation, and the pointer of the next node.
    - std::string DepartureAirport\_;
    - std::string DestinationAirport\_;
    - std::string date\_;
    - int duration\_;
    - int stops\_;
    - std::string AirlineName\_;
    - double price\_;
    - Node\* next\_;
  + Called functions or procedures: None
* **Function**: ~Node(void)
  + Description: Class destructor, in this case, does not perform any operations
  + Parameters: None
  + Global variables: None
  + Called functions or procedures: None
* **Function**: void set\_next(Node\* next)
  + Description: It is responsible for giving a new value to the attribute next\_.
  + Parameters: Node\* next
  + Global variables: Node\* next\_
  + Called functions or procedures: None
* **Function**: double getPrice()
  + Description: Is in charge of returning the value of the price attribute for later use.
  + Parameters: None
  + Global variables: double price\_
  + Called functions or procedures: None
* **Function**: void setPrice(const double& price)
  + Description: It is responsible for giving a new value to the attribute price\_.
  + Parameters: const double& price
  + Global variables: double price\_
  + Called functions or procedures: None
* **Function**: Node\* get\_next(void) const
  + Description: Is in charge of returning the pointer next\_ for later use.
  + Parameters: None
  + Global variables: Node\* next\_
  + Called functions or procedures: None
* **Function**: void write(void) const
  + Description: Is in charge of writing different ‘cout’ in which the different attributes will be displayed on the screen.
  + Parameters: None
  + Global variables: Node\* next\_
  + Called functions or procedures: None

1. Class List

Class ‘**List**’ is the main objective of this practice. When we already have the fundamental elements (also called nodes), we will prepare to create the list. For this it is important to know the operation of a simply linked list:

Each node has a single link field. A reference variable contains a reference to the first node (in our case it is called head\_), and each node (except the last one) links to the next node. The last node contains NULL to indicate the end of the list.

When the head\_ is equal to NULL, it means that the list is empty. And by using the head, we could also insert or delete a node, so that's why it is important to know how many nodes there are.

In our class, we can find the follows methods:

* **Function**: List(void)
  + Description: Is responsible for creating a completely empty list where the head pointer is equal to null.
  + Parameters: None
  + Global variables: Node\* head\_
  + Called functions or procedures: None
* **Function**: ~List(void)
  + Description: Is the destructor of the list class. In the case that the list is not empty, we create a node that is equal to the header, we point to the next node and finally, we delete the created node.
  + Parameters: None
  + Global variables: Node\* head\_
  + Called functions or procedures: Node\* get\_next(void) const
* **Function**: bool empty(void) const
  + Description: Boolean method that checks if the head is pointing to null (that is, the list is empty) and returns true if it is.
  + Parameters: None
  + Global variables: Node\* head\_
  + Called functions or procedures: None
* **Function**: void insert\_head(Node\* node)
  + Description: Method that is responsible for inserting new nodes through the head. The node obtained by parameters we make it point to head and we add in one unit the counter for the number of nodes in the list.
  + Parameters: Node\* node
  + Global variables: int numNodes\_, Node\* head\_
  + Called functions or procedures: void set\_next(Node\* next)
* **Function**: void extract\_end()
  + Description: Method that is responsible for extracting the final node. Its operation consists of first comparing that the list is empty or not, and in the case that it is not, we have two cases:
    - If we have a single node, we check that its next node is null, and in that case, we make the head null and delete the node.
    - The other case is that there is more than one node. In that case, we go through the list as long as the node following the next one is not null. As soon as it finds the null node, it leaves the loop and we are placed in the penultimate, and in its next node, the memory is freed and its header points to null.

In both cases, the number of nodes decreases.

* + Parameters: None
  + Global variables: Node\* head\_, int numNodes\_
  + Called functions or procedures: bool empty(void) const, Node\* get\_next(void) const, void set\_next(Node\* next), free()
* **Function**: void sortByPrice()
  + Description: This method takes care of ordering the list by the price attribute (in this case, in increasing case).
  + Parameters: None
  + Global variables: Node\* head\_
  + Called functions or procedures: Node\* get\_next(void) const, double getPrice(), void setPrice(const double& price)
* **Function**: Node\* createPlane ()
  + Description: Asks the user for the reservation data and then turns it into a node and returns it.
  + Parameters: None
  + Global variables: None
  + Called functions or procedures: Constructor with parameters of the node class
* **Function**: void write(void) const
  + Description: A node pointing to the head is created. We check if the node points to null (empty list) and in that case we indicate it. In the case that the list is not empty, we call the write method of the node and print the counter on the screen.
  + Parameters: None
  + Global variables: Node\* head\_
  + Called functions or procedures: void write(void) const, Node\* get\_next(void) const

1. Main program

In the main program, we can find the menu with which the user will work. We could select the option that we most want.

1. Source code

* main.cc:

#include <ostream>

#include <iostream>

#include <stdio.h>

#include "list.h"

int main(void) {

int selection;

List\* list = new List();

do {

std::cout << "This is the menu of the final task:" << std::endl << std::endl;

std::cout << "-------------------------------- Menu --------------------------------" << std::endl << std::endl;

std::cout << "Select '1' to add a node in front of the list" << std::endl;

std::cout << "Select '2' to remove the end node from de list" << std::endl;

std::cout << "Select '3' to sort the list in increasing order by the field price" << std::endl;

std::cout << "Select '4' to display the number of the nodes and the list" << std::endl;

std::cout << "Select '5' to exit the program" << std::endl << std::endl;

std::cout << "Selection number: ";

std::cin >> selection;

std::cout << std::endl;

//Nodes test

Node\* aux = new Node("Barajas", "Tenerife", "10/1/2019", 3, 0, "Ryanair", 320.1);

Node\* aux1 = new Node("Granada", "Sofía", "3/1/2020", 4, 4, "Ryanair", 121);

switch (selection) {

case 1:

std::cout << "----------- Option '1': Adding a node in front of the list -----------" << std::endl;

std::cout << "Creating a new node..." << std::endl;

list -> insert\_head(list -> createPlane());

break;

case 2:

std::cout << "----------- Option '2': Removing a node from the end of the list -----------" << std::endl;

list-> extract\_end();

break;

case 3:

std::cout << "----------- Option '3': Sorting the list -----------" << std::endl;

list -> sortByPrice();

break;

case 4:

std::cout << "----------- Option '4': Display the number of nodes and the list -----------" << std::endl;

list -> write();

break;

default: std::cout << "----------- Exiting the program -----------" << std::endl;

break;

}

} while (selection != 5);

return 0;

}

* node.h:

#include <iostream>

class Node {

private:

std::string DepartureAirport\_;

std::string DestinationAirport\_;

std::string date\_;

int duration\_;

int stops\_;

std::string AirlineName\_;

double price\_;

Node\* next\_;

public:

Node(void);

Node(const std::string& DeparAir, const std::string& DestAir, const std::string& date, const int& duration, const int& stops, const std::string AirlineName, const double& price);

~Node(void);

void set\_next(Node\* next);

double getPrice();

void setPrice(const double& price);

Node\* get\_next(void) const;

void write(void) const;

};

* node.cc

#include "node.h"

Node::Node(void): next\_(NULL) {}

Node::Node(const std::string& DeparAir, const std::string& DestAir, const std::string& date, const int& duration, const int& stops, const std::string AirlineName, const double& price):

next\_(NULL) {

DepartureAirport\_= DeparAir,

DestinationAirport\_= DestAir,

date\_ = date,

duration\_ = duration,

stops\_ = stops,

AirlineName\_ = AirlineName,

price\_ = price;

}

Node::~Node(void) {}

void Node::set\_next(Node\* next) {

next\_ = next;

}

Node\* Node::get\_next(void) const {

return next\_;

}

void Node::write(void) const {

std::cout << "------------------------------------------------" << std::endl;

std::cout << "Printing the dates of the Plane reservation...\n" << std::endl;

std::cout << "Departura Airport: " << DepartureAirport\_ << std::endl;

std::cout << "Destination Airport: " << DestinationAirport\_ << std::endl;

std::cout << "Name of airline: " << AirlineName\_ << std::endl;

std::cout << "Date: " << date\_ << std::endl;

std::cout << "Duration: " << duration\_ << std::endl;

std::cout << "Number of stops: " << stops\_ << std::endl;

std::cout << "Total price: " << price\_ << std::endl << std::endl;

}

double Node::getPrice() {

return price\_;

}

void Node::setPrice(const double& price) {

price\_=price;

}

* list.h:

#include <iostream>

#include <string>

#include <stdlib.h>

#include "node.h"

class List {

private:

Node\* head\_;

int numNodes\_ = 0;

public:

List(void);

~List(void);

bool empty(void) const;

void insert\_head(Node\* node);

void extract\_end();

void sortByPrice();

Node\* createPlane ();

void write(void) const;

};

* list.cc:

#include "list.h"

List::List(void) : head\_(NULL) {}

List::~List(void) {

while (!empty()) {

Node\* aux = head\_;

head\_ = head\_ -> get\_next();

delete aux;

}

}

bool List::empty(void) const {

return (head\_ == NULL);

}

void List::insert\_head(Node\* node) {

node -> set\_next(head\_);

head\_ = node;

numNodes\_++;

}

void List::extract\_end() {

Node\* aux = head\_;

if (empty()) {

std::cout << "\nThe list is empty\n" << std::endl;

}

else {

if (aux -> get\_next() == NULL) {

aux -> set\_next(NULL);

head\_=NULL;

delete aux;

}

else {

while ((aux -> get\_next())->get\_next() != NULL) {

aux -> set\_next(aux -> get\_next());

}

free(aux->get\_next());

aux -> set\_next(NULL);

}

}

numNodes\_--;

}

void List::sortByPrice() {

double aux;

if (head\_ == NULL) {

std::cout << "\nThe list is empty\n" << std::endl;

}

else {

for (Node\* i = head\_; i -> get\_next() != NULL; i = i -> get\_next()) {

for (Node\* j = i -> get\_next(); j != NULL; j = j -> get\_next() ) {

if (i -> getPrice() > j -> getPrice()) {

aux = i -> getPrice();

i -> setPrice(j -> getPrice());

j -> setPrice(aux);

}

}

}

}

}

void List::write(void) const {

Node\* aux = head\_;

if (aux == NULL) {

std::cout << "\nThe list is empty\n" << std::endl;

std::cout << "\nNumber of elements is: 0" << std::endl;

}

else {

std::cout << "-------- Number elements is: " << numNodes\_ << "--------" << std::endl;

while (aux != NULL) {

aux -> write();

aux = aux -> get\_next();

}

}

}

Node\* List::createPlane () {

std::string DepartureA, DestinationA, Date, NomAirline;

double Price;

int Stops, Duration;

std::cout << "\n\nInsert the name of the Departure Airport: ";

std::cin >> DepartureA;

std::cout << "\nInsert the name of the Destination Airport: ";

std::cin >> DestinationA;

std::cout << "\nInsert the date: ";

std::cin >> Date;

std::cout << "\nInsert the duration: ";

std::cin >> Duration;

std::cout << "\nInsert the numbers of stops: ";

std::cin >> Stops;

std::cout << "\nInsert the name of the Airline: ";

std::cin >> NomAirline;

std::cout << "\nInsert the price: ";

std::cin >> Price;

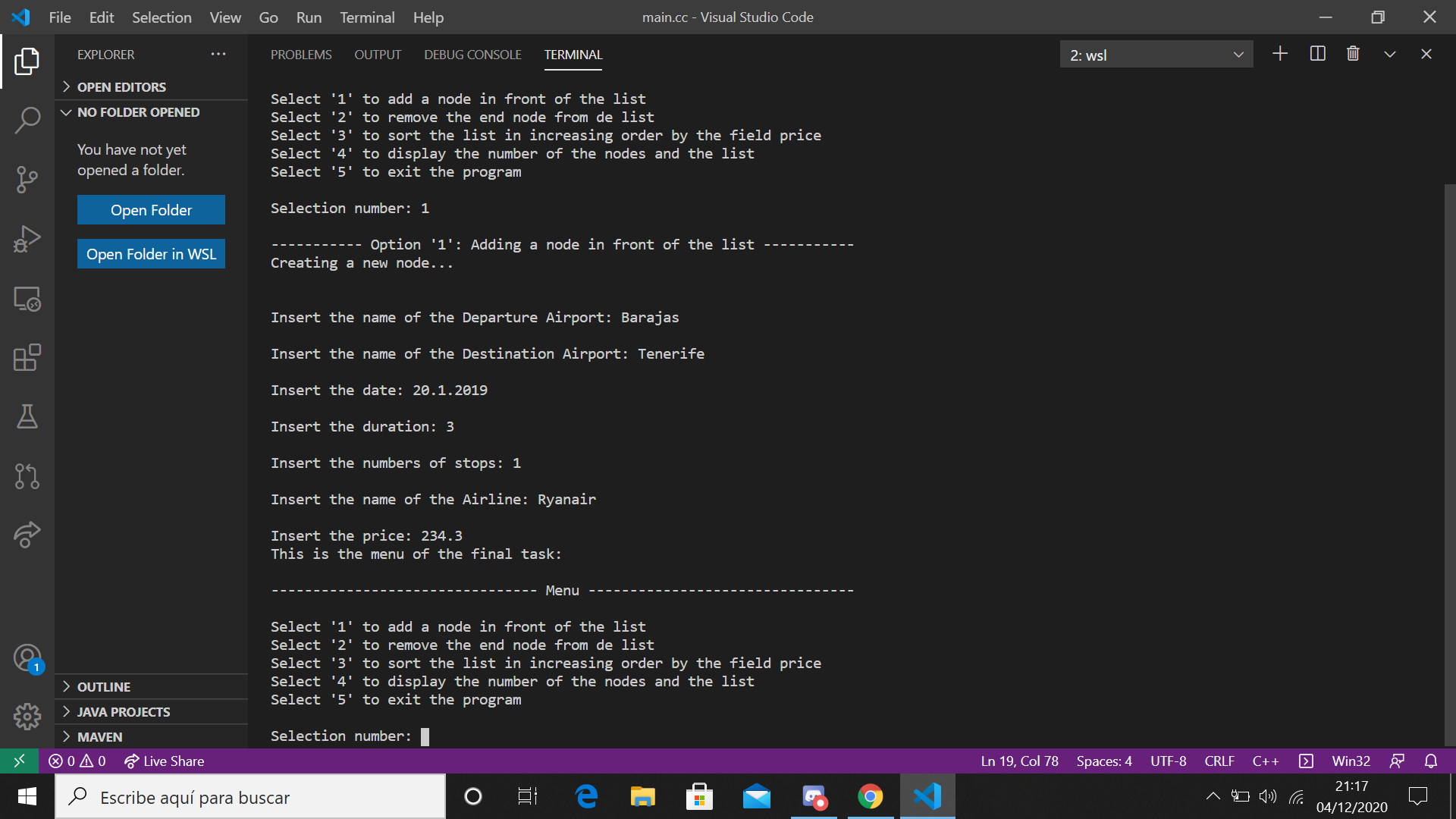
Node\* aux = new Node(DepartureA, DestinationA, Date, Duration, Stops, NomAirline, Price);

return aux;

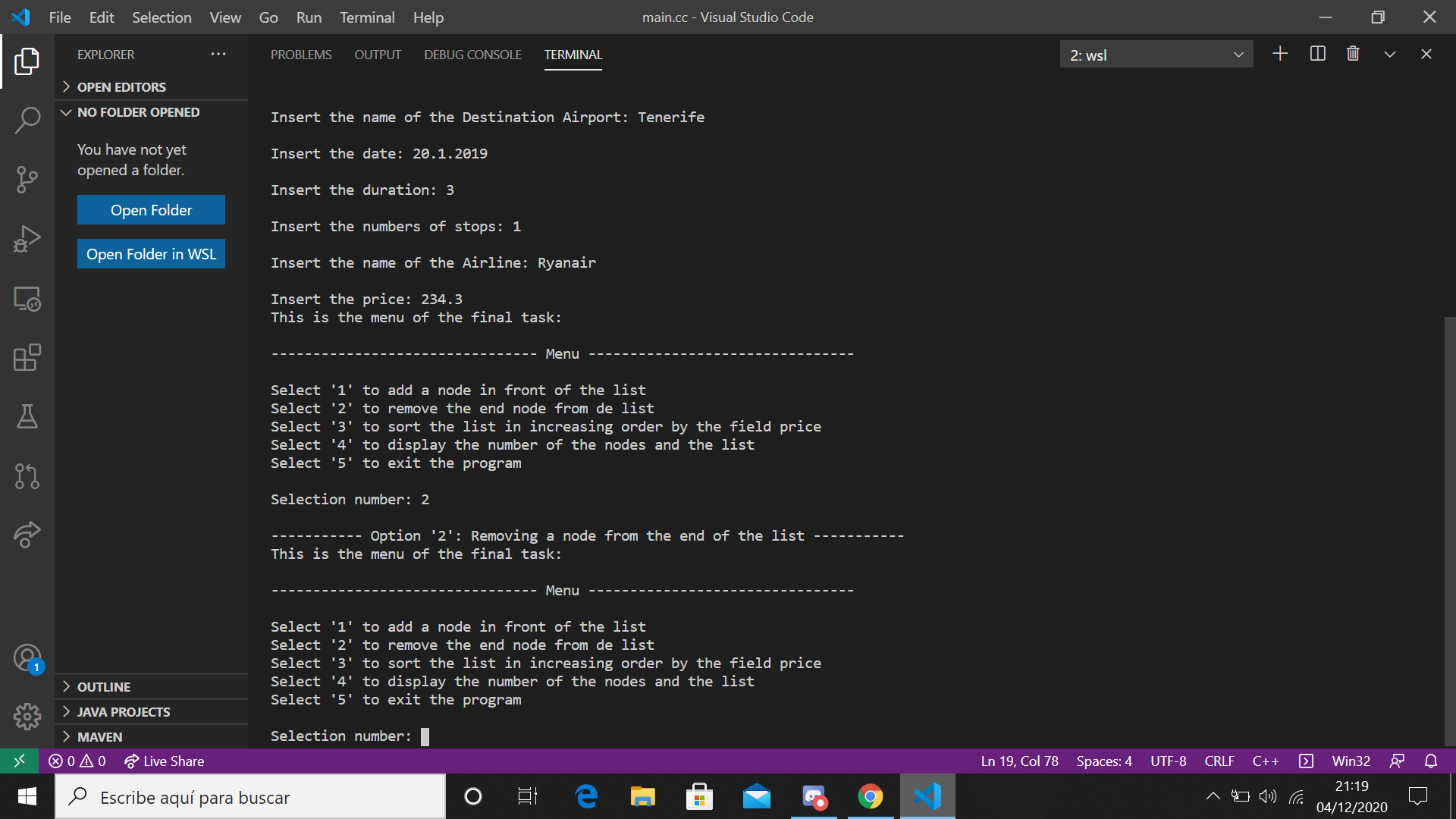
}

1. Screenshots from the program

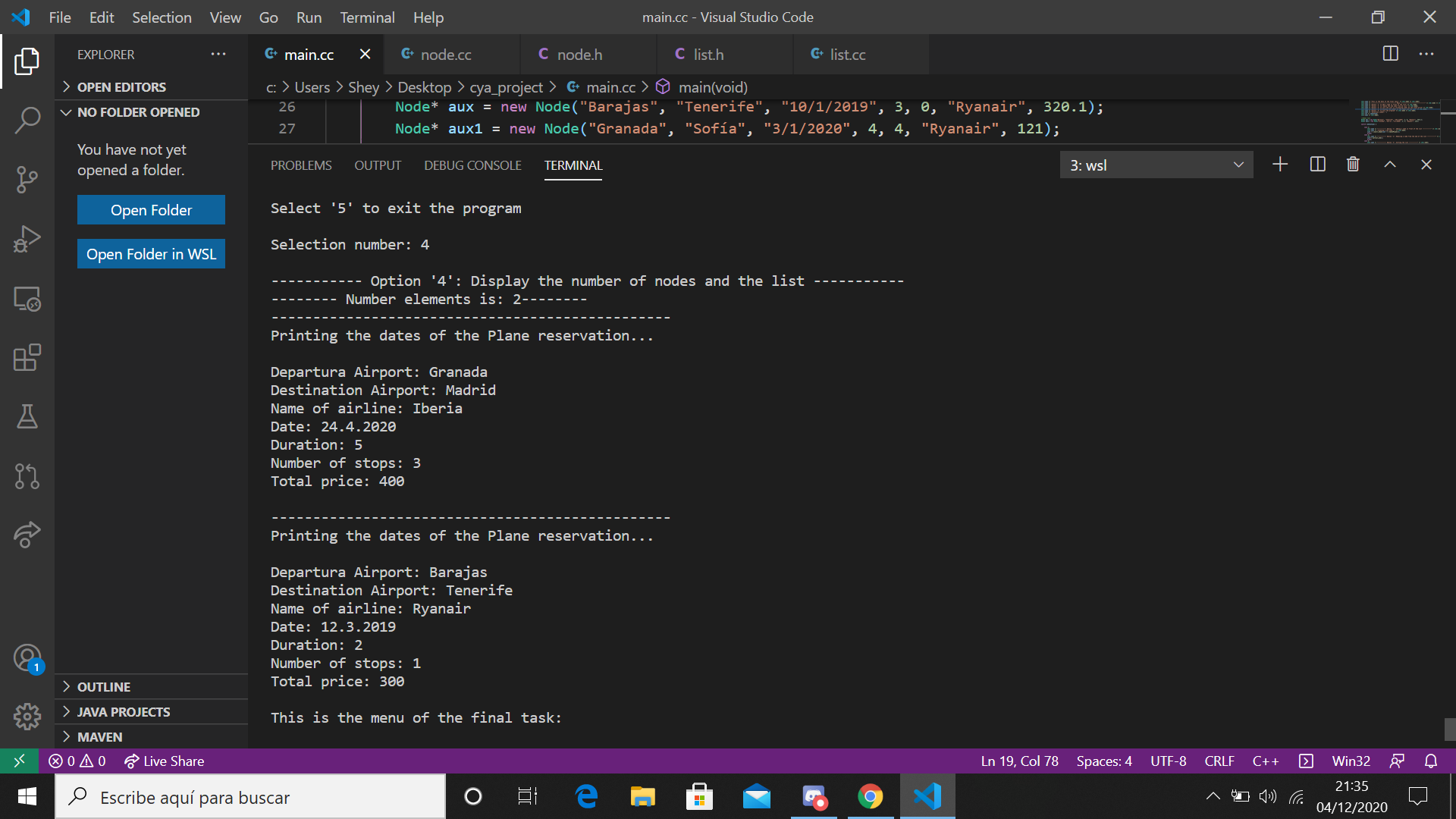
* Insert a node in front of the list:



* Remove the end node:



* Display the number of nodes and the list:



* Sort the list in increasing order by price:

