Картина, която съдържа текст, графична колекция

Описанието е генерирано автоматичноТЕХНИЧЕСКИ УНИВЕРСИТЕТ – ВАРНА

Факултет по изчислителна техника и автоматизация

Катедра „КНТ“

**КУРСОВ ПРОЕКТ**

по дисциплината „ООП – 1ва част”

на тема: „Програма за обработка на самолети ”

Вариант 2

|  |  |
| --- | --- |
| Изготвил: Ивайло Илиев | Проверил: |
| Специалност: КСТ |  |
| Група: 3б |  |
| Факултетен номер: 22621518  Курс: 2ри |  |

2024

**I. Задание**

**Курсов проект N 2**

**Програма за обработка на самолети**

1. Да се дефинира клас CPlanе, съхраняващ информация за марка на самолета и летателни часове с необходимите конструктори, методи и оператори.
2. Да се дефинира клас CAirtravel, съхраняващ информация за име на авиокомпанията, националност, брой дестинации и самолети, като последните са съхранени в контейнер map<CPlanе, unsigned>. Име на дестинацията и броя полети по нея са съхранени в multimap <string,unsigned>. Освен необходимите методи, да се реализират и следните член функции:
   * изчислява и връща средния брой дестинации на френските авиолкомпании;
   * връща списък от самолети (list<CPlain>), с летателни часове по-малко от 2000 и брой дестинации повече от 10;

1. Да се дефинира клас CAirport, съхраняващ информация за името на летището, номер на полет и контейнер съдържащ CAirtravel, съхраняващ данните за различните авиокомпании и брой полети. Например:

class CAirport{

string name;

unsigned n\_polets;

multimap<CAirtravel,unsigned> airtravel;

….

}

Да се дефинира конструктор с параметри за летището и име на файл, съдържащ необходимата информация за едно летище.

Да се добавят следните методи:

* при подаден аргумент – марка самолет, връща броя полети и авиокомпанията, собственик на самолета;
* при подаден аргумент име на дестинацията, връща името на авиокомпанията, обезпечила най-много полети;
* връща марката самолет, с най много дестинации;
* при подаден аргумент авиокомпания, връща дестинацията с най-много полети;
* връща контейнер от самолети, съдържащ летателните часове >10000 на всички самолети от авиокомпаниите, ползващи летище София.

IV. Да се демонстрира работата на класа CAirport, като се дефинира обект от този клас и се предостави възможност за различни справки за съответното летище. Документиране на курсовия проект: .doc файл със заглавна страница, условие на задачата, кратко описание на класовете и функциите и листинг на програмата с коментари.

**II. Кратко описание на програмата**

**Предназначение**

Програмата може да обработва не-големи данни . Има възможност за добавяне на данни като самолети, авиолинии и летища. Има възможност за извеждане на различни справки

**Данни, с които работи програмата**

Данните, които се обработват с програмата са текст (string) и числа (int ,double). Данните, върху които се извършва обработка се зареждат във експлицитният конструктор от файлове.

**Възможни грешки и защита включена в програмата**

Тъй като някои от данните, с които програмата работи се зареждат от файл със специален формат са предвидени проверки:

- За несъществуващ файл

- За грешен формат на данните във файла ( съществуват и частни случаи, в които програмата може да приеме файл с близък, но все пак невалиден формат) Поради особеностите на технологиите използвани за този проект, съществува опасност да се получат грешки и при въвеждане на различен от изисквания тип данни.

**III. Обща структура на програмата**

1. **Клас CPlane**

* Съхранява информация за марка на самолета и летателни часове.
* Съдържа необходимите конструктори, методи и оператори.

**Член - променливи:**

* String brand - Марка на самолета (напр. "Boeing", "Airbus").
* Unsigned int flightHours - Летателни часове на самолета.
* double altitude – служебна член-променлива

**Конструктори:**

* CPlane() - Конструктор по подразбиране.
* CPlane(string brand, unsigned flightHours) - Конструктор с параметри.
* CPlane(const CPlane &other)- Копиращ конструктор.

**Properties:**

* string getBrand() const - Връща марката на самолета.
* unsigned getFlightHours() const - Връща летателните часове на самолета.
* void SetBrand(string br) -задава марката на самолета
* void SetFlightHours(int flH) – задава летателните часове на самолета.

**Оператори:**

* CPlane& operator=(const CPlane& other) – оператор за присвояване
* bool operator<(const CPlane& other) const -служебен оператор

**Методи:**

* static vector<CPlane> LoadFromFile(const string& filename) – зарежда данните от файл

1. **Клас CAirtravel**

* Съхранява информация за име на авиокомпанията, националност, брой дестинации и самолети (в контейнер map<CPlane, unsigned>).
* Съдържа дестинации и брой полети по тях (в контейнер multimap<string, unsigned>).
* Включва методи за:
* Изчисляване и връщане на средния брой дестинации на френските авиокомпании.
* Връщане на списък от самолети с летателни часове под 2000 и брой дестинации над 10.

**Член - променливи:**

* string nameOfAirline - Име на авиокомпанията.
* string nationality - Националност на авиокомпанията.
* map<CPlane, unsigned int> \*planeAndDestinationsCount - брой дестинации и самолети (в контейнер map<CPlane, unsigned>)
* multimap<string, unsigned int> \*destinationAndFlightsCount - дестинации и брой полети по тях (в контейнер multimap<string, unsigned>).
* double altitude – служебна член-променлива

**Конструктори:**

* CAirtravel() - Конструктор по подразбиране.
* CAirtravel(string a, string n) - Конструктор с параметри.
* CAirtravel(const CAirtravel& other)- Копиращ конструктор.

**Properties:**

* string GetNameOfAirline() const- Връща името на авиокомпанията
* void SetNameOfAirline(string n) – Задава името на авиокомпанията
* string GetNationality() const - Връща националността на авиокомпанията
* void SetNationality(string n) – Задава националността на авиокомпанията
* map<CPlane, unsigned int> GetPlaneAndDestinationsCount()const – Връща самолетите и броя полети.
* multimap<string, unsigned int> GetDestinationAndFlightsCount()const - Връща мултимапа на дестинации.

**Оператори:**

* bool operator<(const CPlane& other) const -служебен оператор

**Методи:**

* void АddPlane(const CPlane& plane, unsigned int destinations) - Добавя самолет в map.
* void AddDestination(const string& destination, unsigned int flights) - Добавя дестинация в мултимапа.
* int AvgDestinationCountOfAllFrenchAirlines()const - Изчислява и връща средния брой дестинации на френските авиокомпании.
* list<CPlane> PlanesWithCriteria2000And10() const - Връща списък от самолети с летателни часове по-малко от 2000 и брой дестинации повече от 10.
* static vector<CAirtravel> LoadFromFile(const string& filename) - зарежда данните от файл

1. **Клас CAirport**

 Съхранява информация за името на летището, номер на полет и контейнер, съдържащ CAirtravel.

 Включва методи за:

* Връщане на броя полети и авиокомпанията, собственик на определена марка самолет.
* Връщане на авиокомпанията с най-много полети към дадена дестинация.
* Връщане на марката самолет с най-много дестинации.
* Връщане на дестинацията с най-много полети за дадена авиокомпания.
* Връщане на контейнер от самолети с летателни часове над 10000 от всички авиокомпании, ползващи летище София.

**Член - променливи:**

* string airportName - Име на летището.
* unsigned int n\_flights – Номер на полет
* multimap<CAirtravel,unsigned int> \*airlinesAndFlightsCount – map с обекти CAirtravel и броя полети

**Конструктори:**

* CAirport() - Конструктор по подразбиране.
* CAirport(string a, unsigned int n) - Конструктор с параметри.
* CAirport(const CAirport& other)- Копиращ конструктор.

**Properties:**

* string GetAirportName() const – връща името на летището
* unsigned int GetN\_flightsconst - връща номера на полет
* void setAirportName(string n) -задава име на летището
* void setN\_flights(unsigned int n ) – задава номер на полет
* multimap<CAirtravel, unsigned int> GetAirlinesAndFlightsCount()const – връща мултимап с авиолиниите и броя полети

**Методи:**

* void AddAirline(const CAirtravel& airline, unsigned int flights)const - Добавя авиолиния в мултимап.
* void ByPlaneBrandGetFlightsCountAndAirline(string brand)const - при подаден аргумент – марка самолет, връща броя полети и авиокомпанията, собственик на самолета
* void ByDestinationGetAirlineWithMostFlights(string destination)const - при подаден аргумент име на дестинацията, връща името на авиокомпанията, обезпечила най-много полети
* void PlaneBrandWithMaxDestination()const - връща марката самолет, с най много дестинации;
* void ByAirlineGetDestinationWithMostFlights(string airline)const - при подаден аргумент авиокомпания, връща дестинацията с най-много полети;
* list<CPlane> PlanesWithCriteria10000AndSofia() const – връща контейнер от самолети, съдържащ летателните часове >10000 на всички самолети от авиокомпаниите, ползващи летище София.
* void PlanesWithCriteria10000AndSofia2()const – връща контейнер от самолети, съдържащ летателните часове >10000 на всички самолети от авиокомпаниите, ползващи летище София.
* static vector<CAirport> LoadFromFile(const string& filename) - зарежда данните от файл

**Общи функции:**

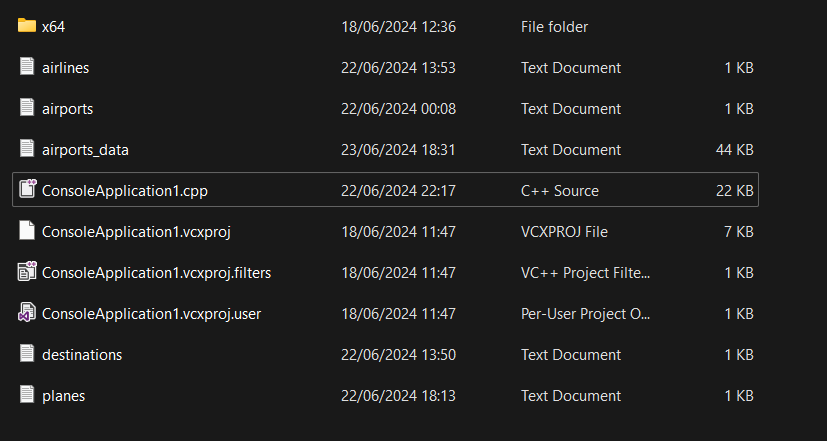
* unsigned generateRandomNumber(unsigned int min, unsigned int max) – генерира произволно число в определен диапазон
* vector<string> LoadFromFile(const string& filename) – зарежда данните от файл
* void WriteAirportsDataToFile(const vector<CAirport>& airports, const string& filename) – записва цялостна справка за всички обекти във файл

**В main:**

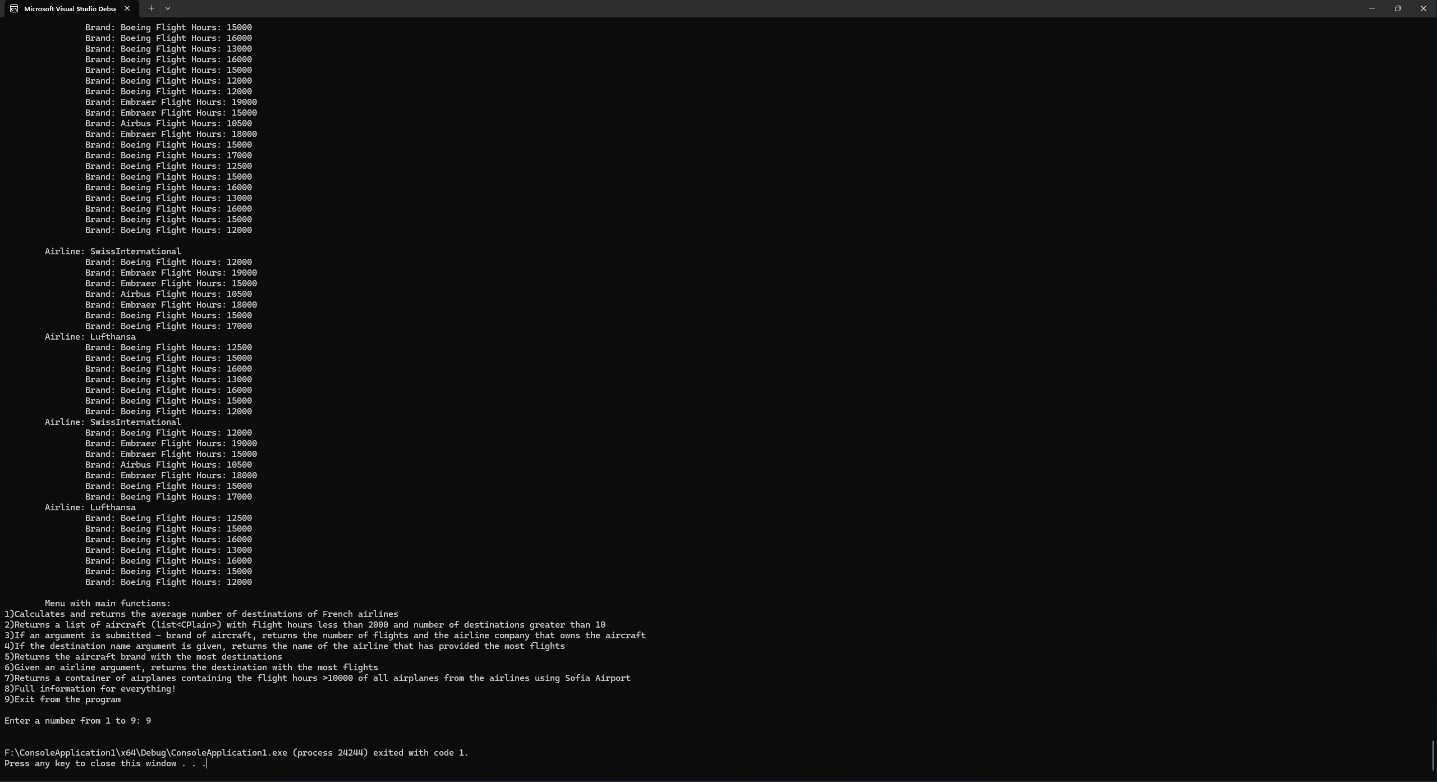
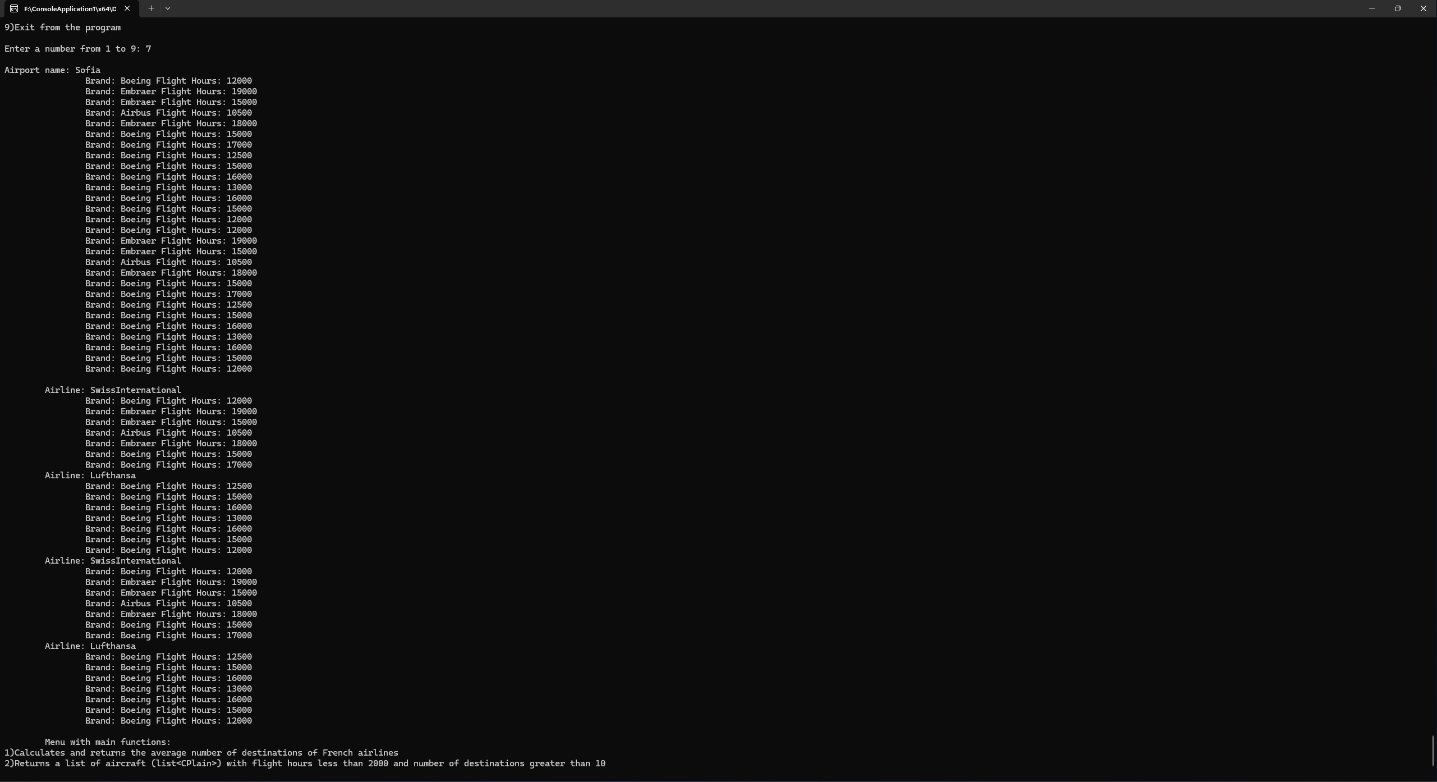
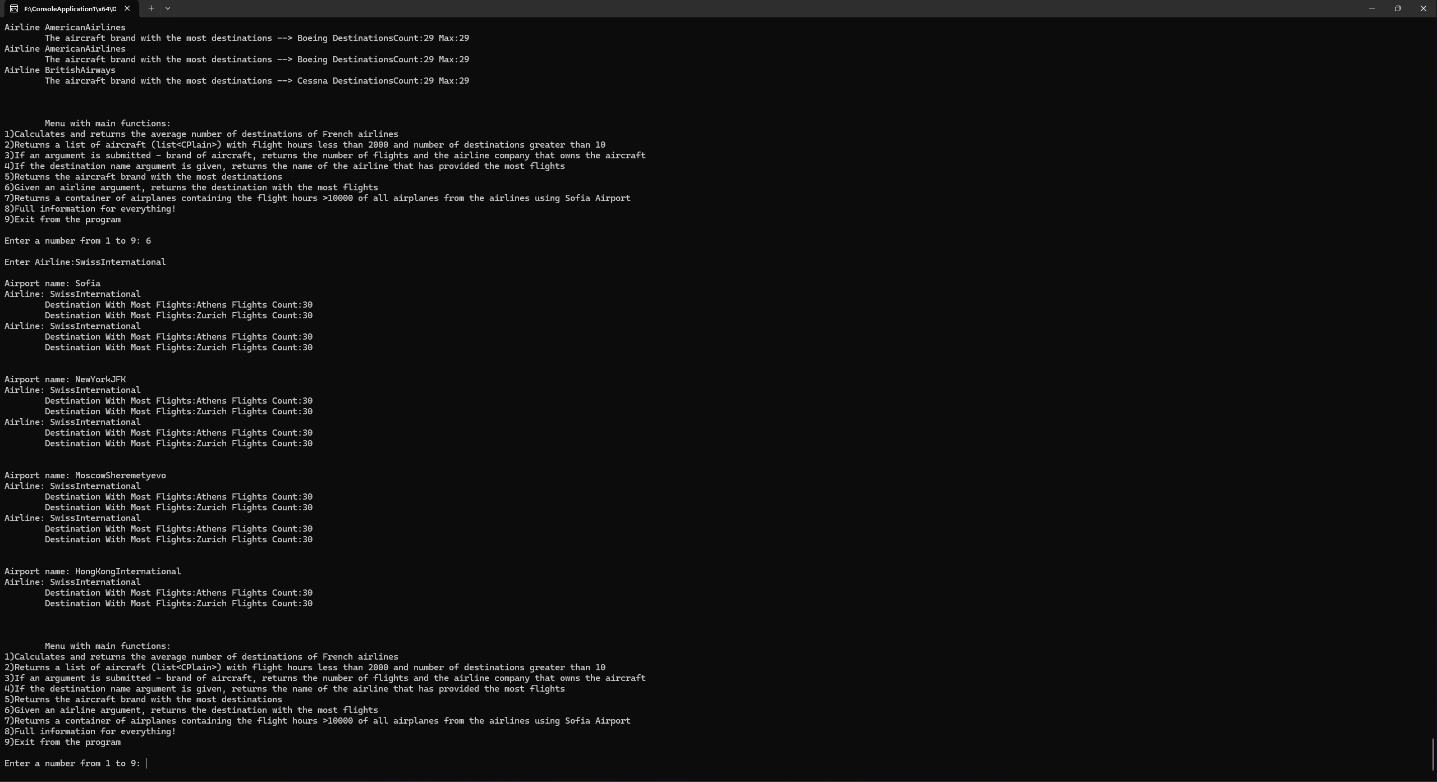
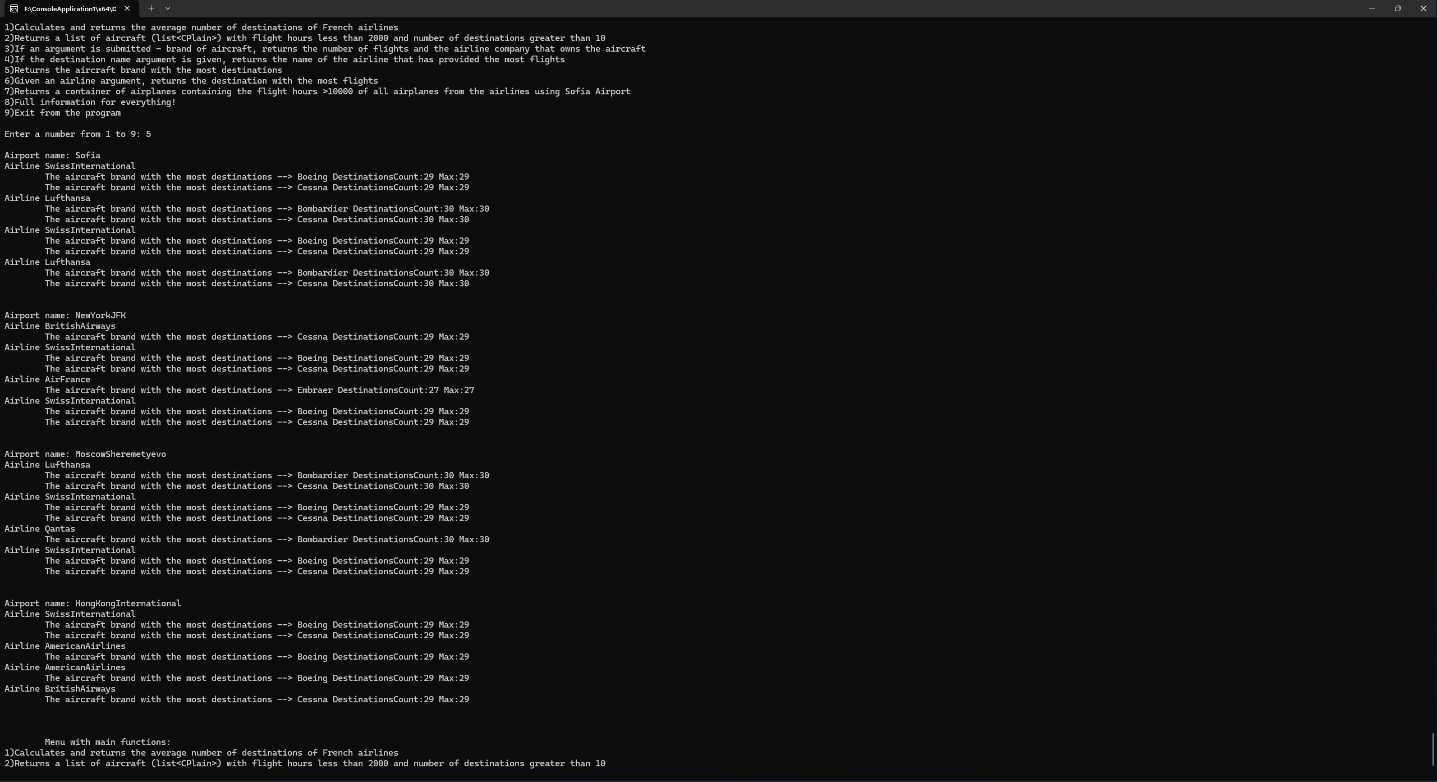
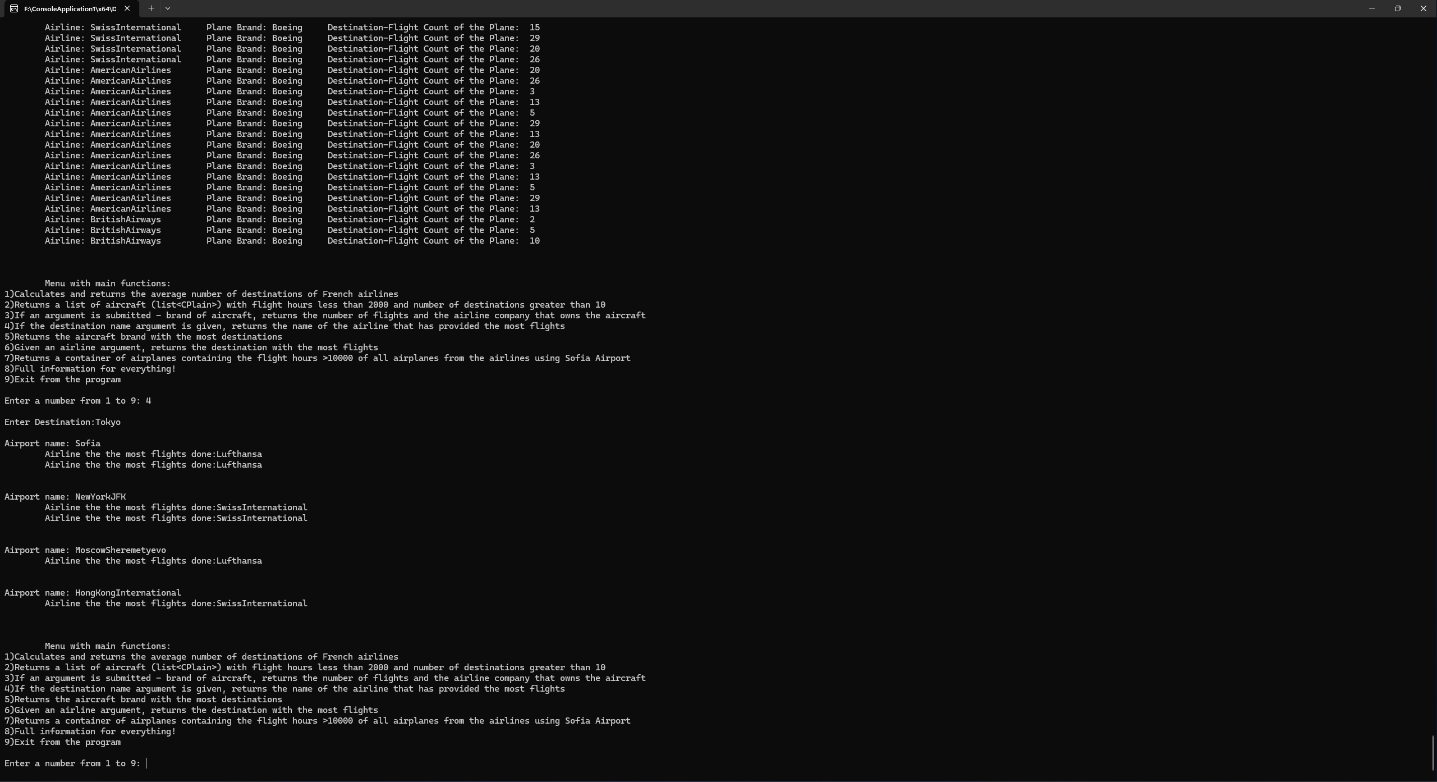
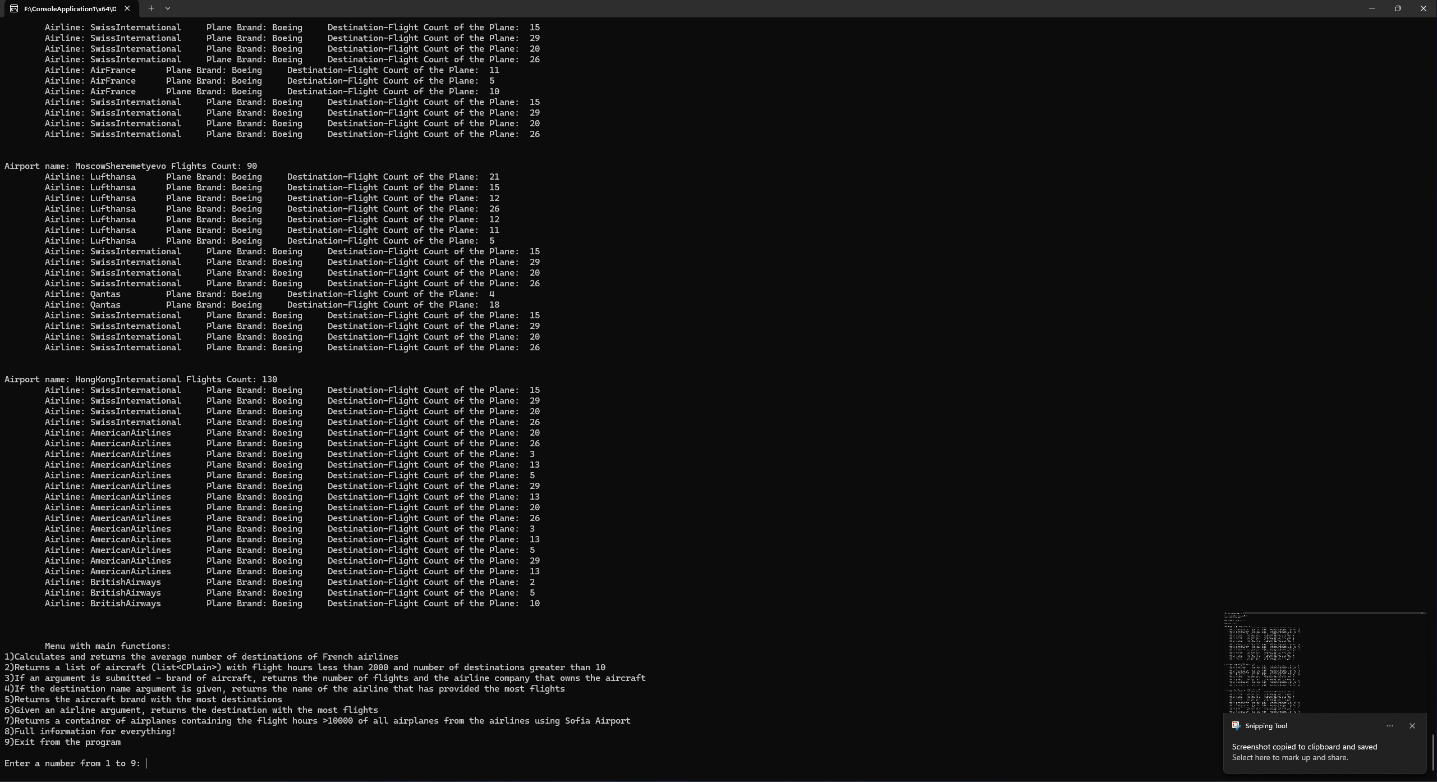
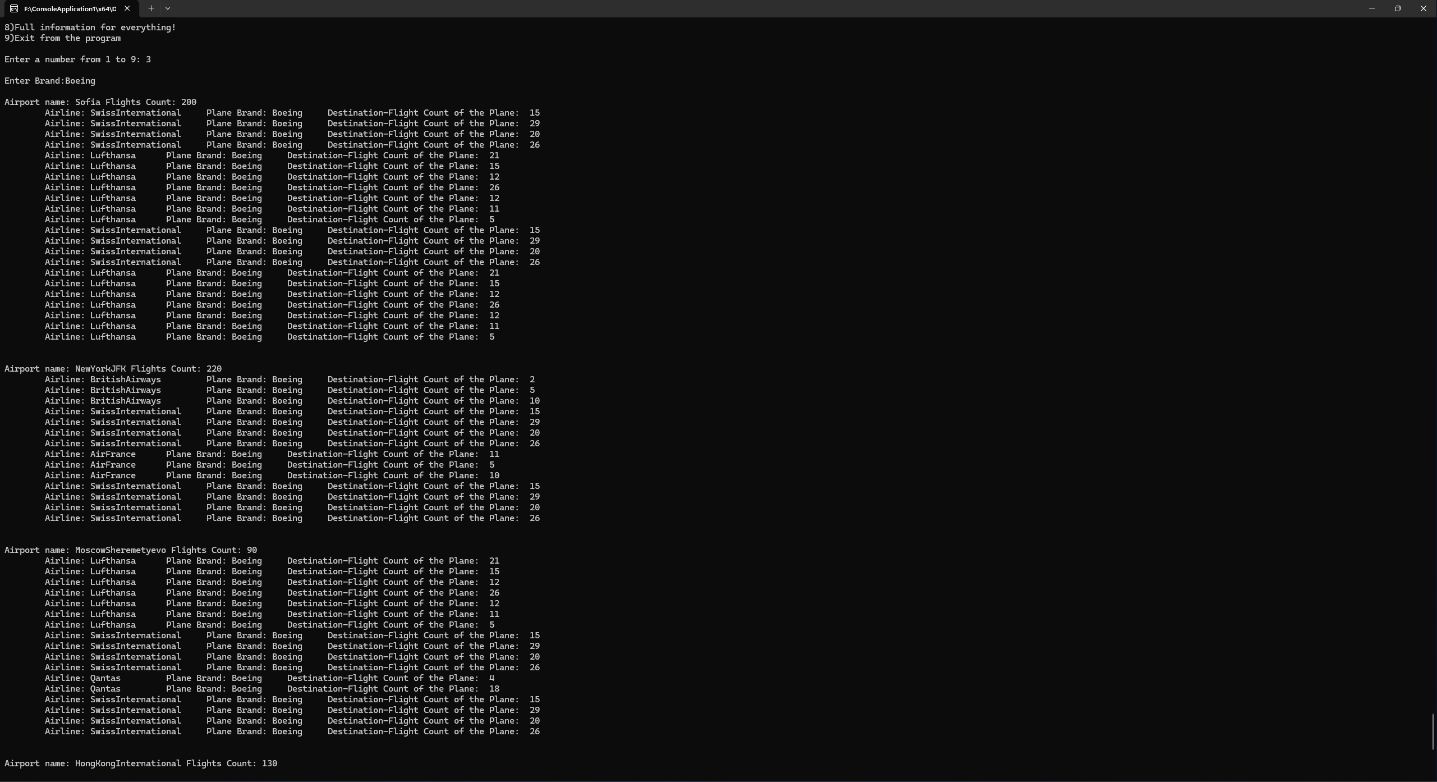
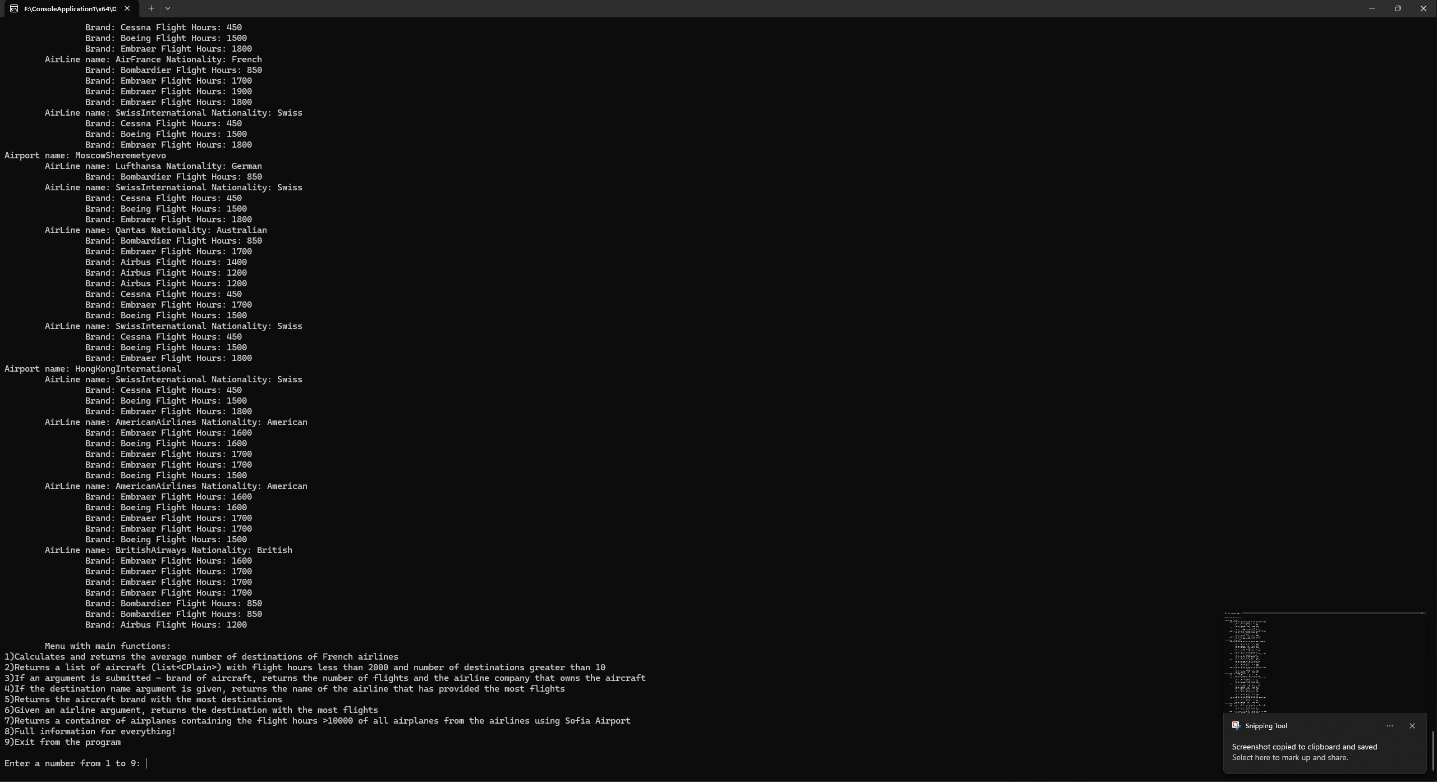
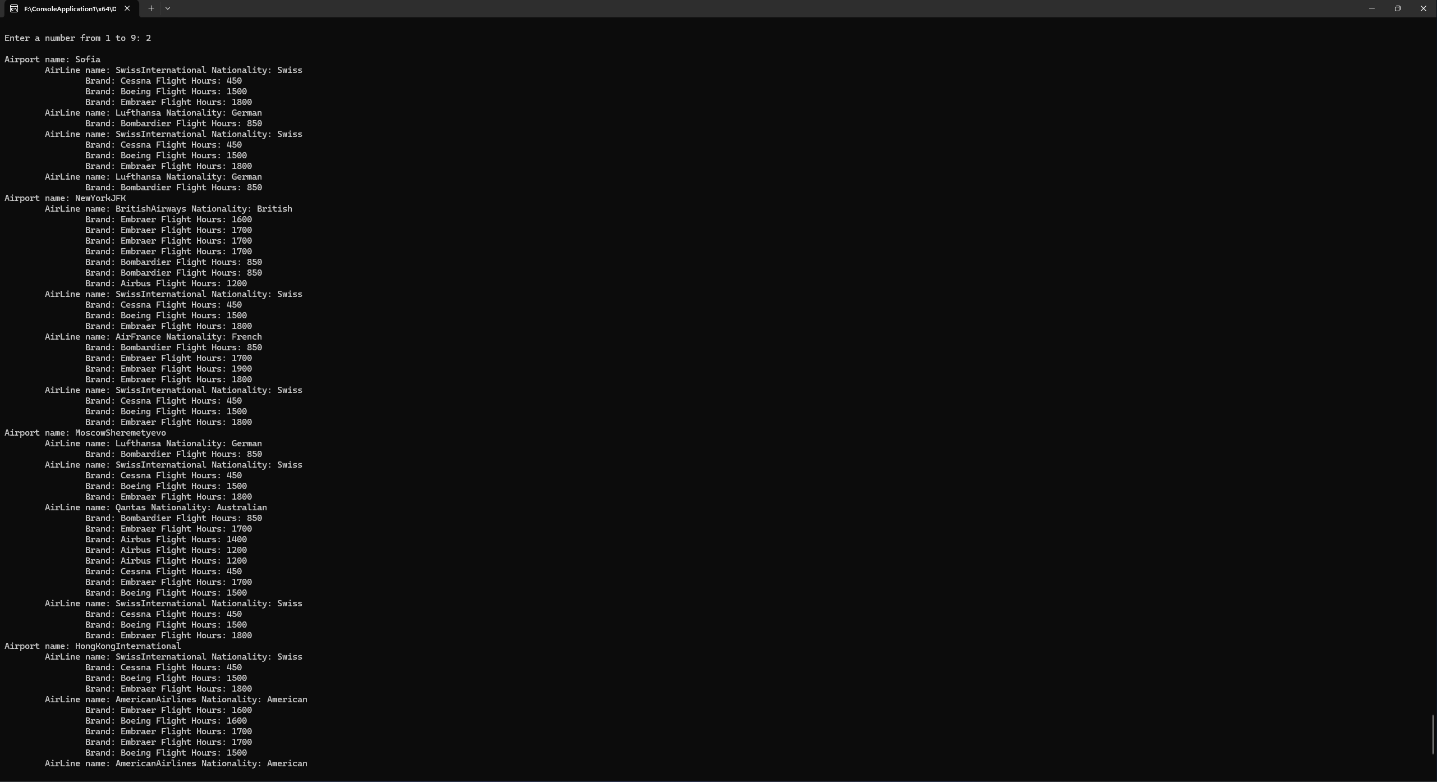
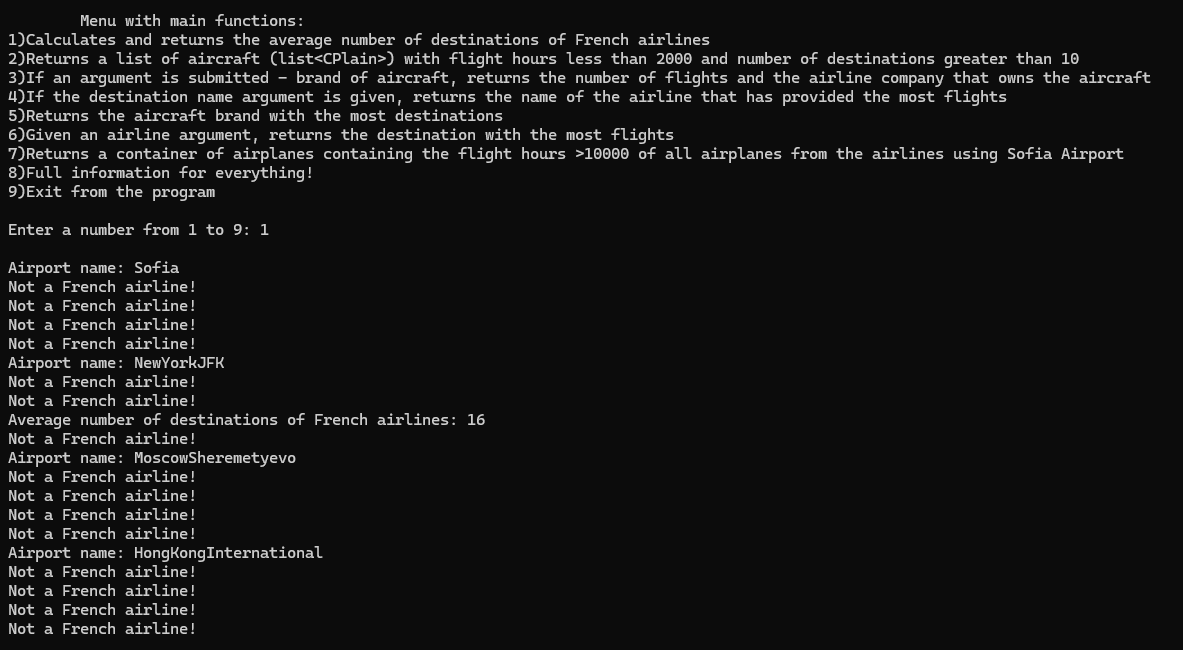
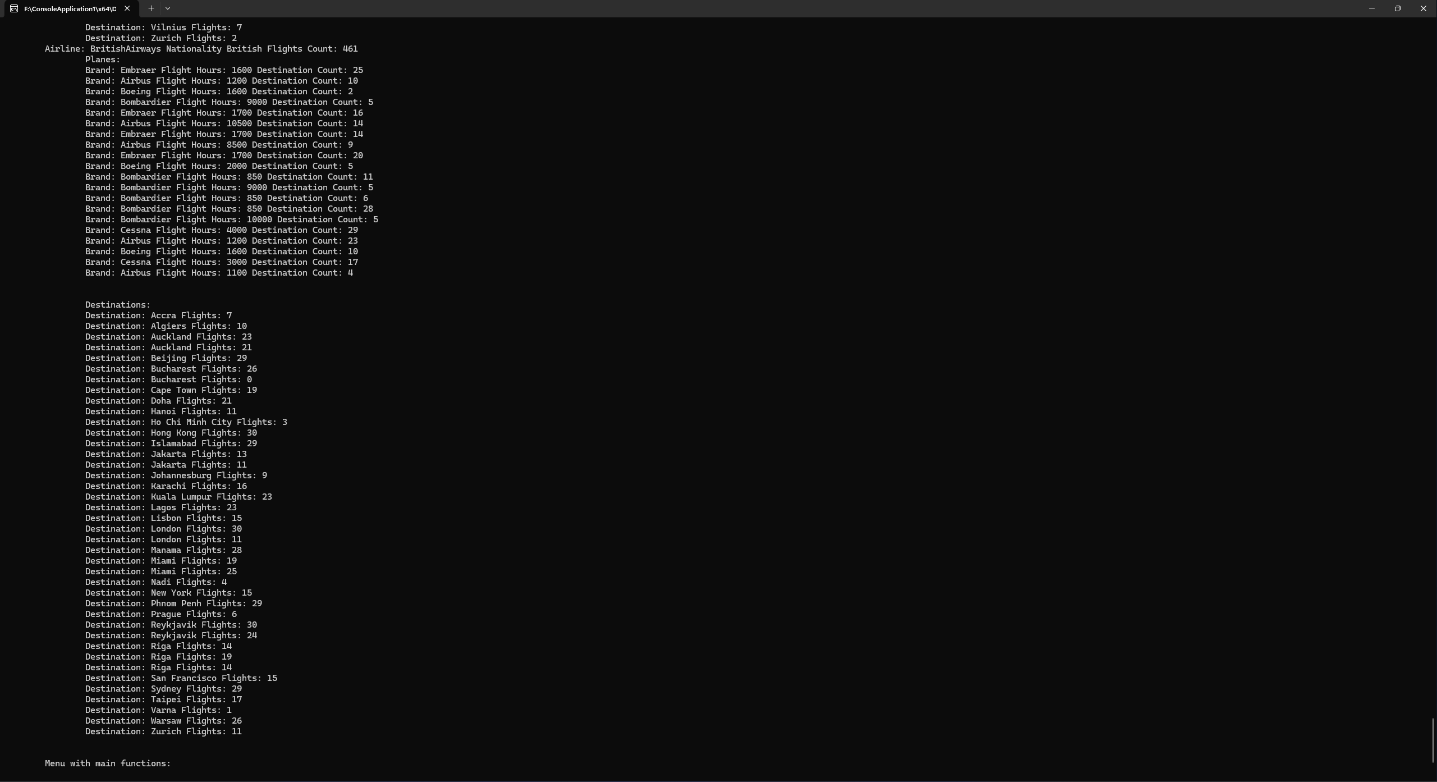
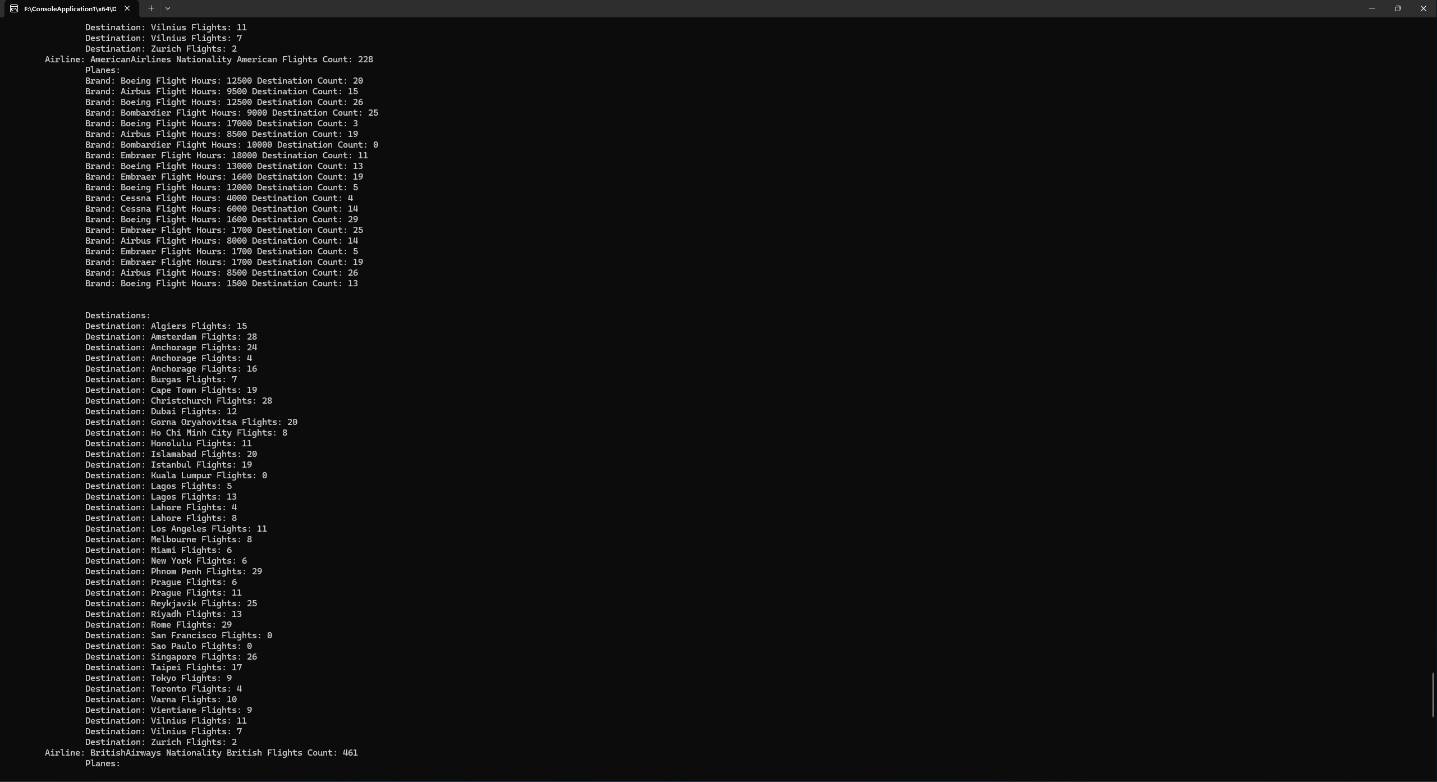
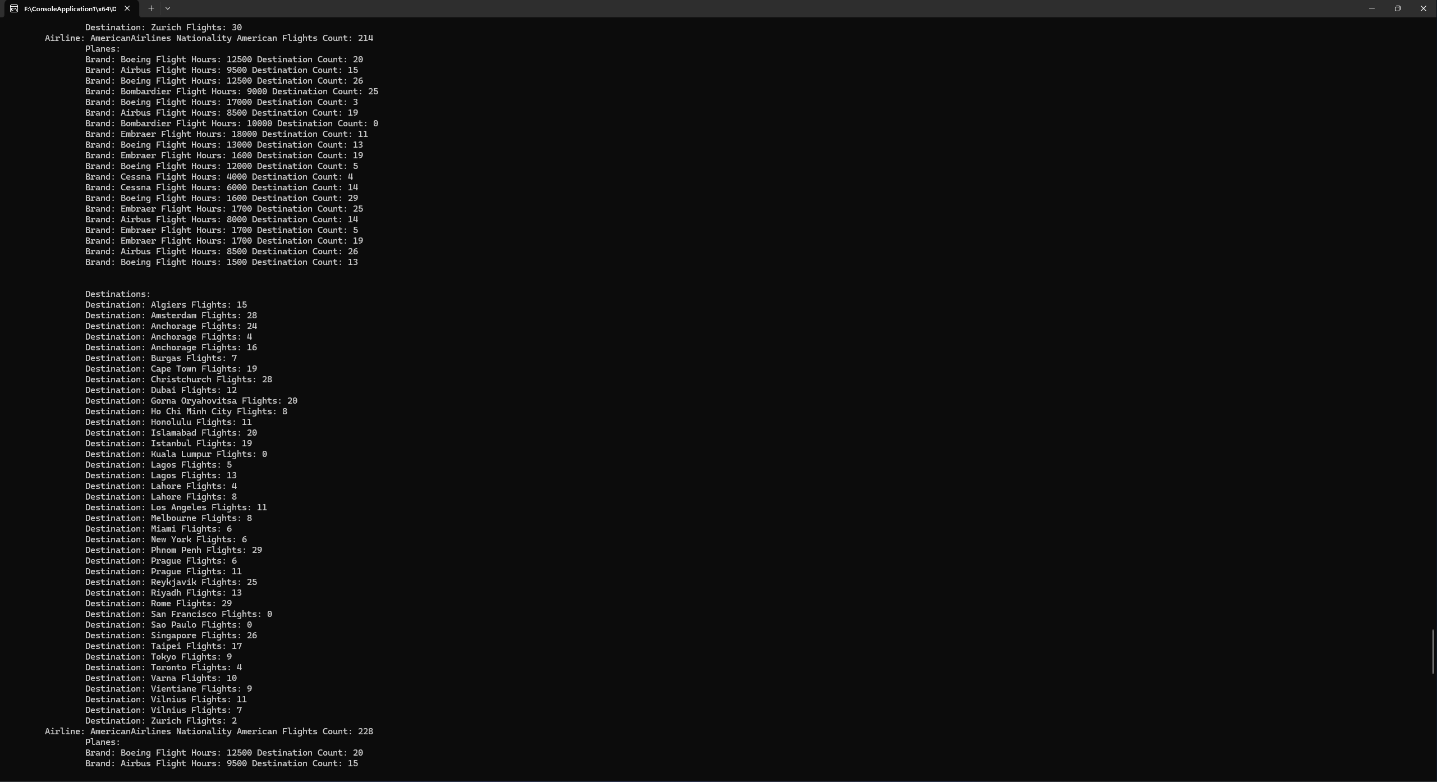
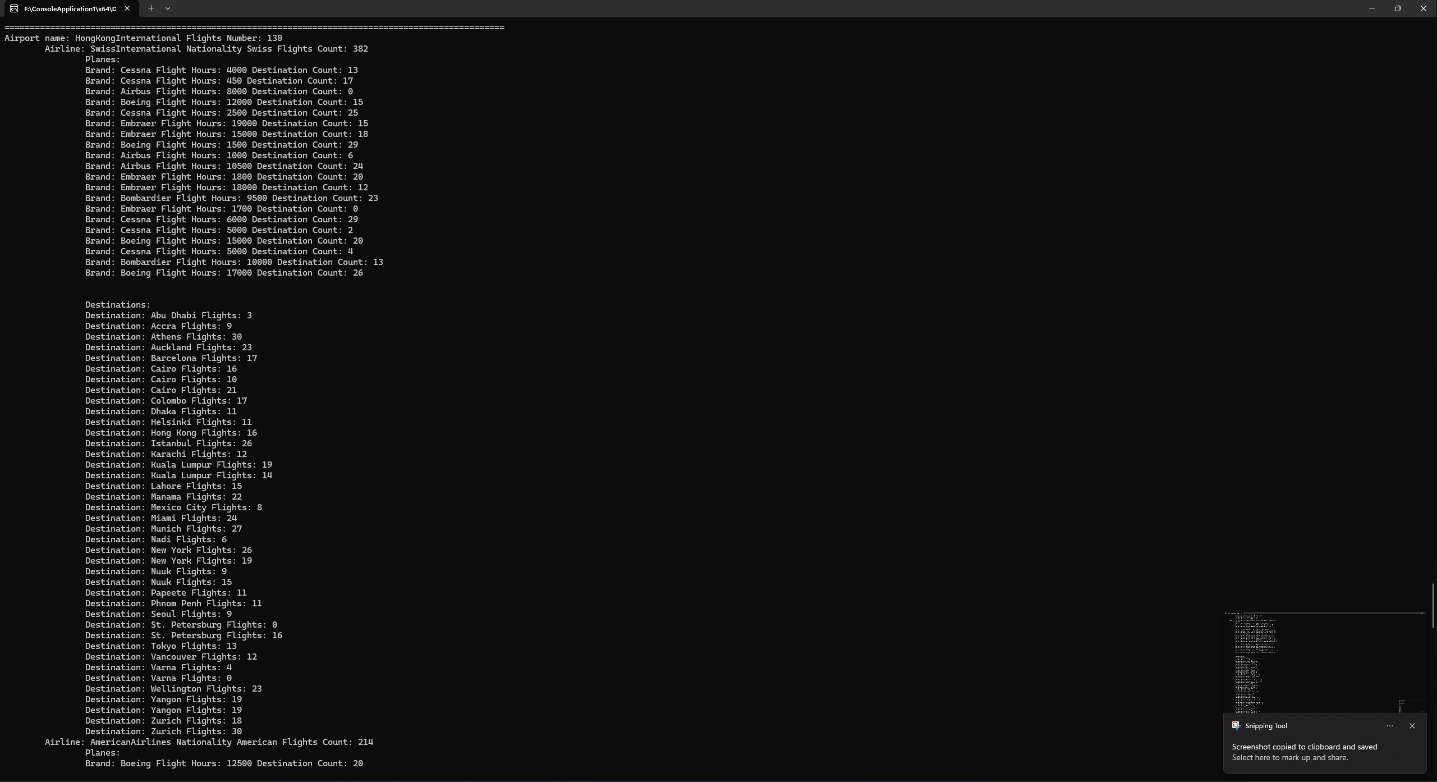
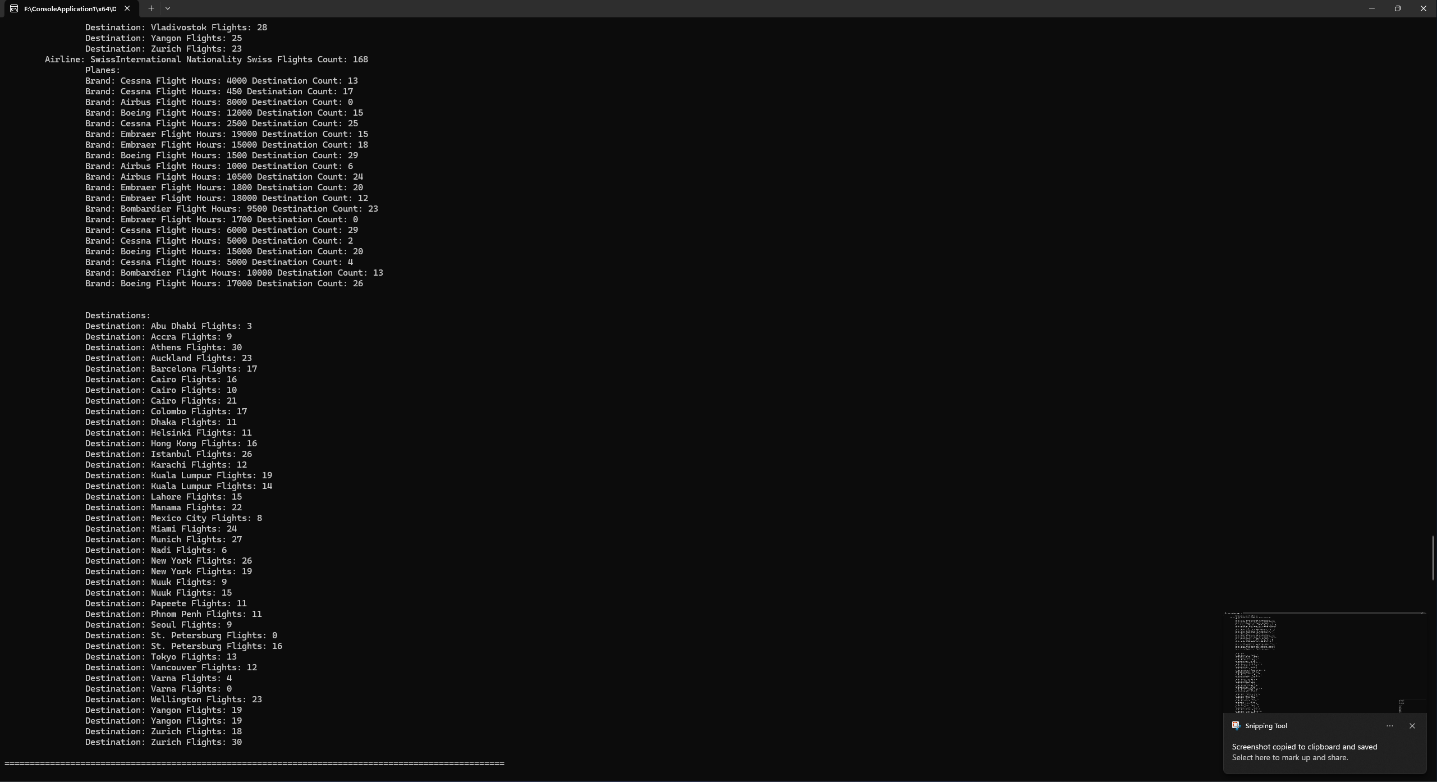
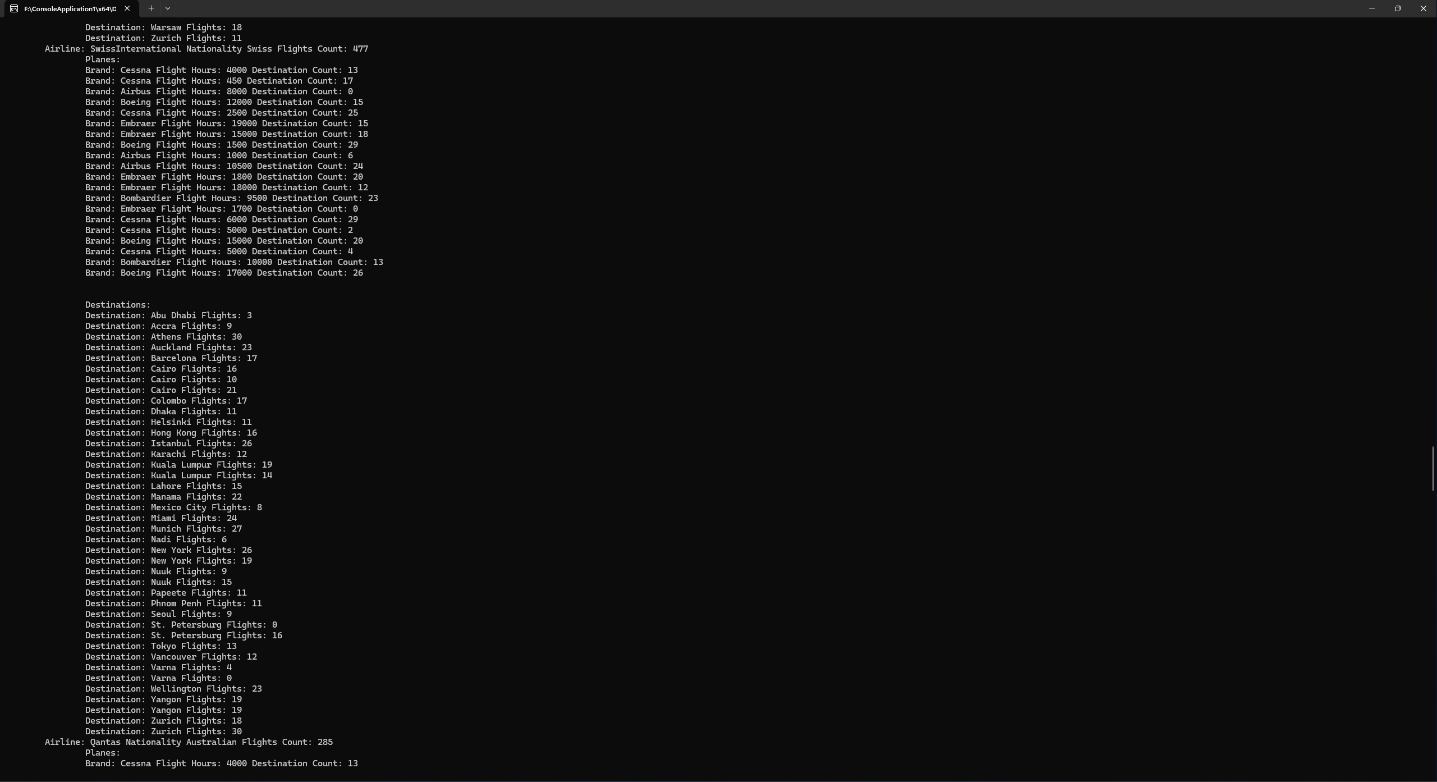
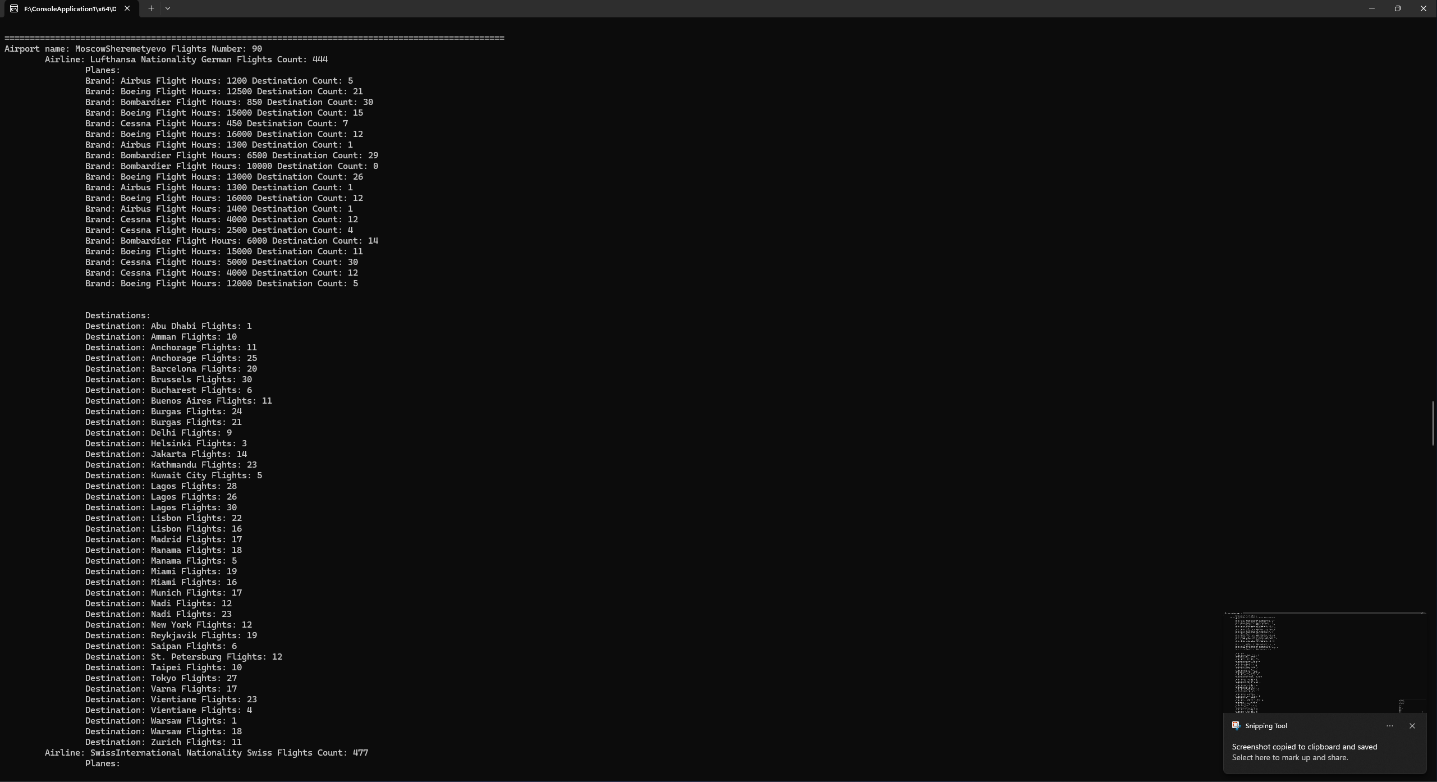
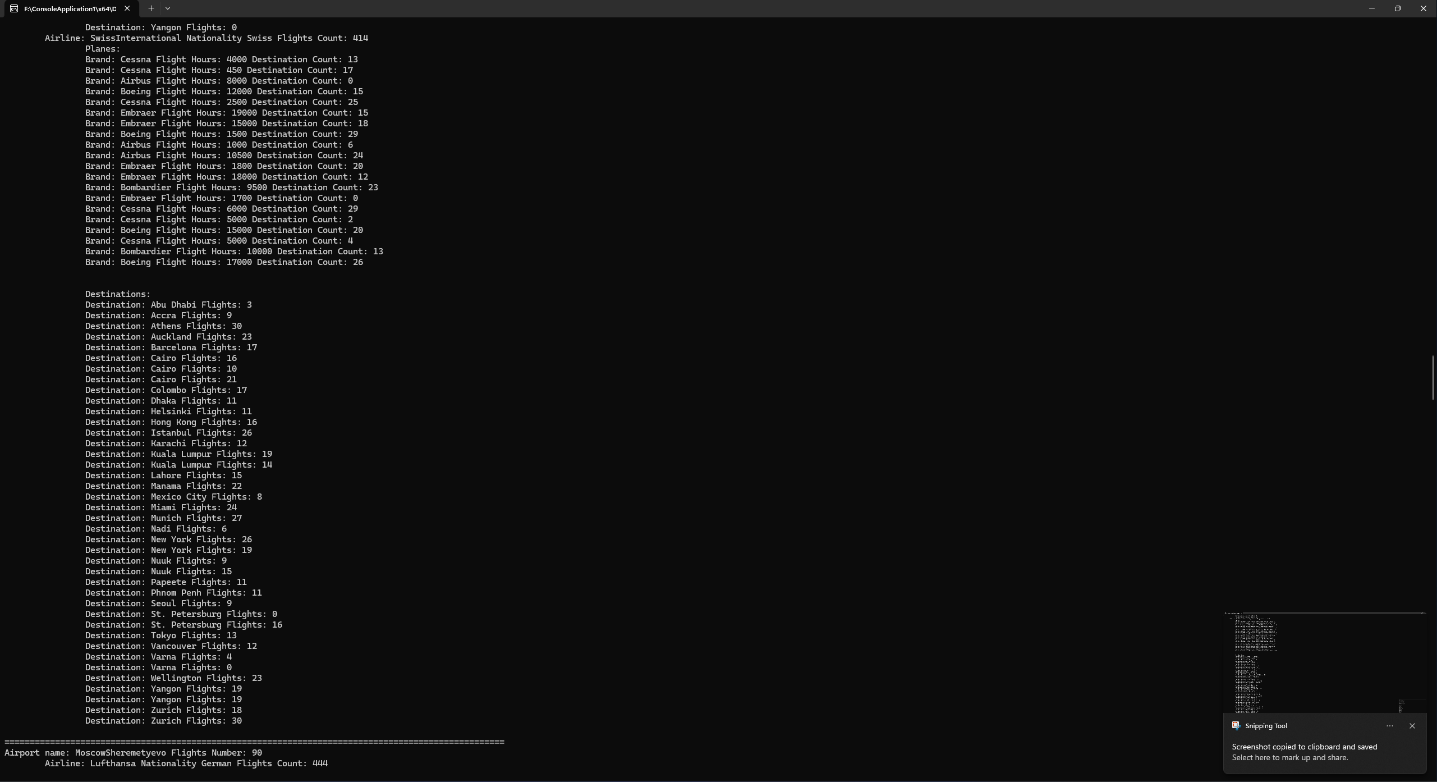
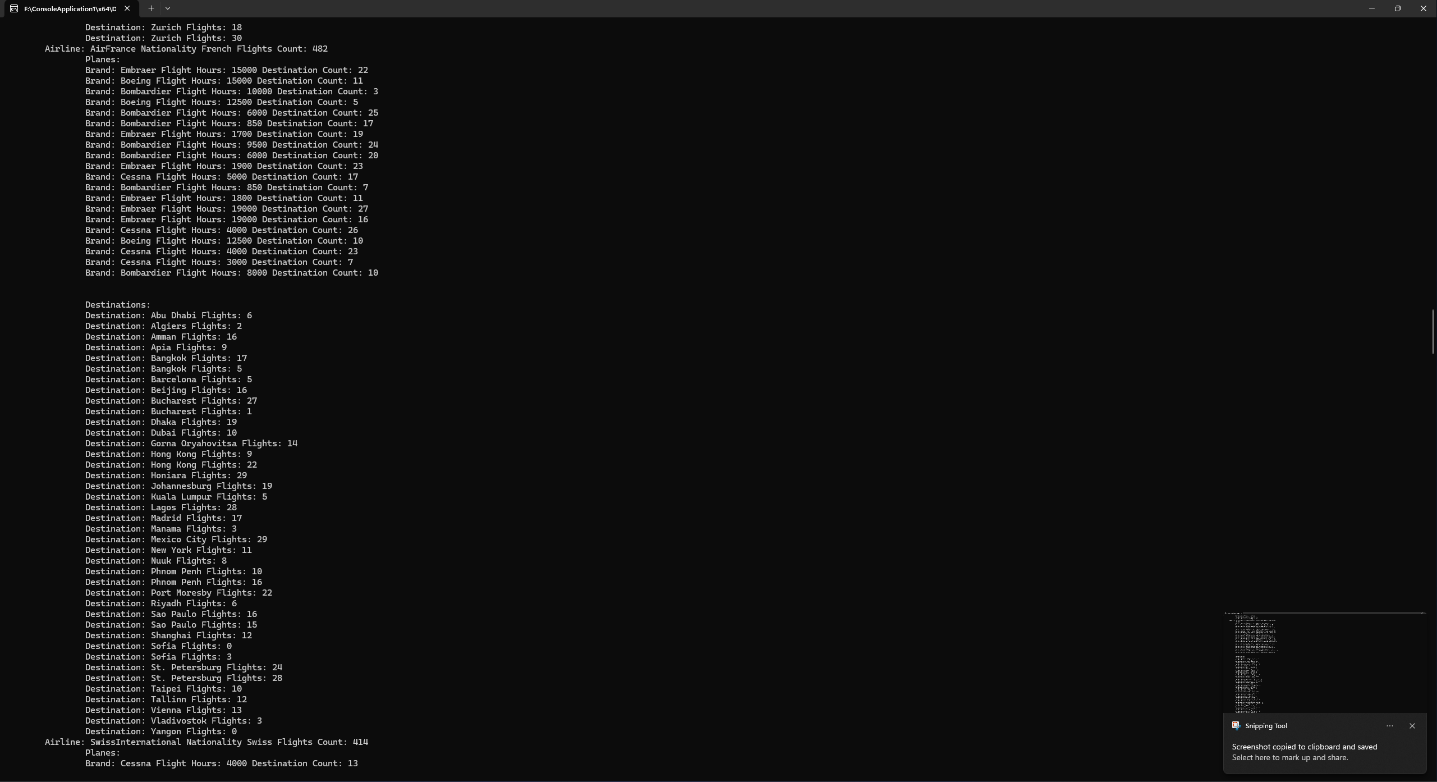
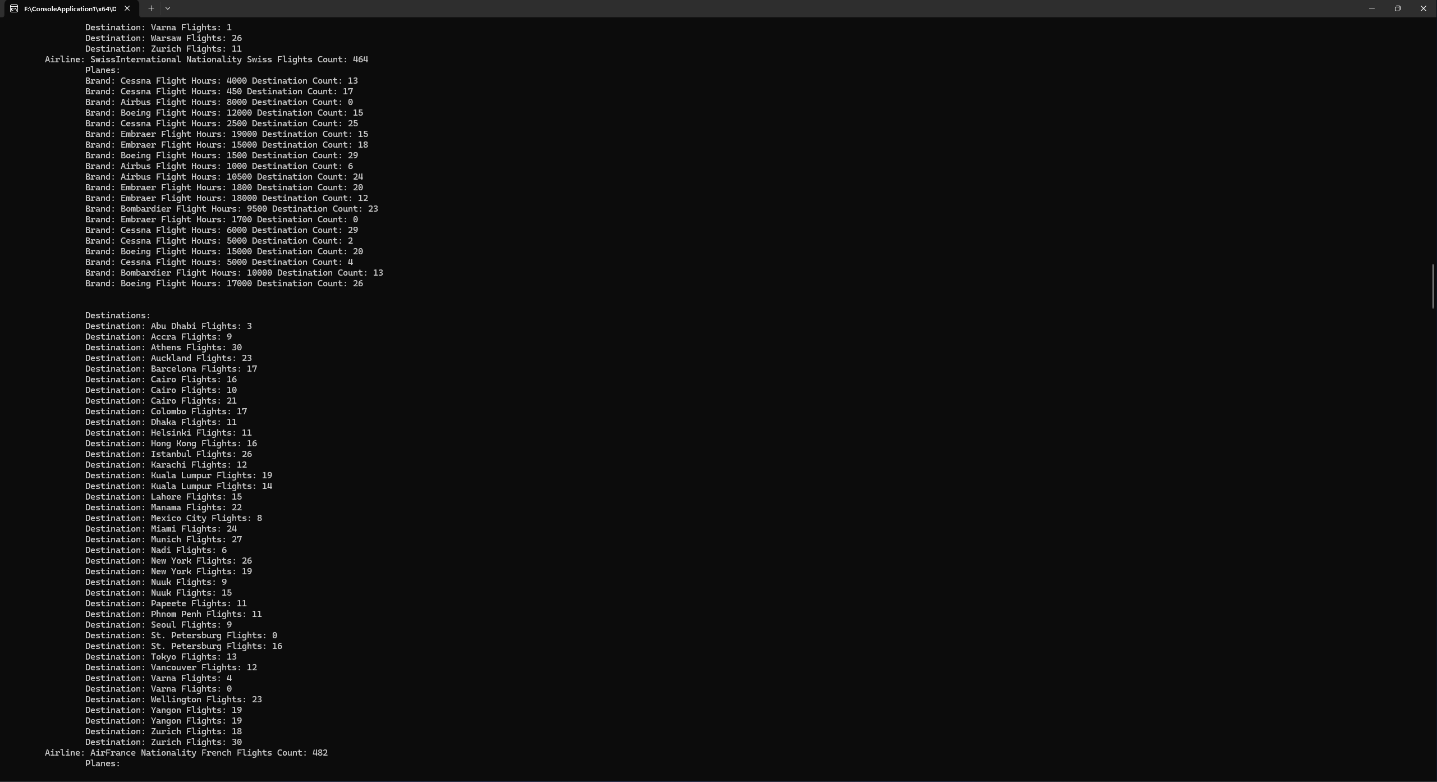
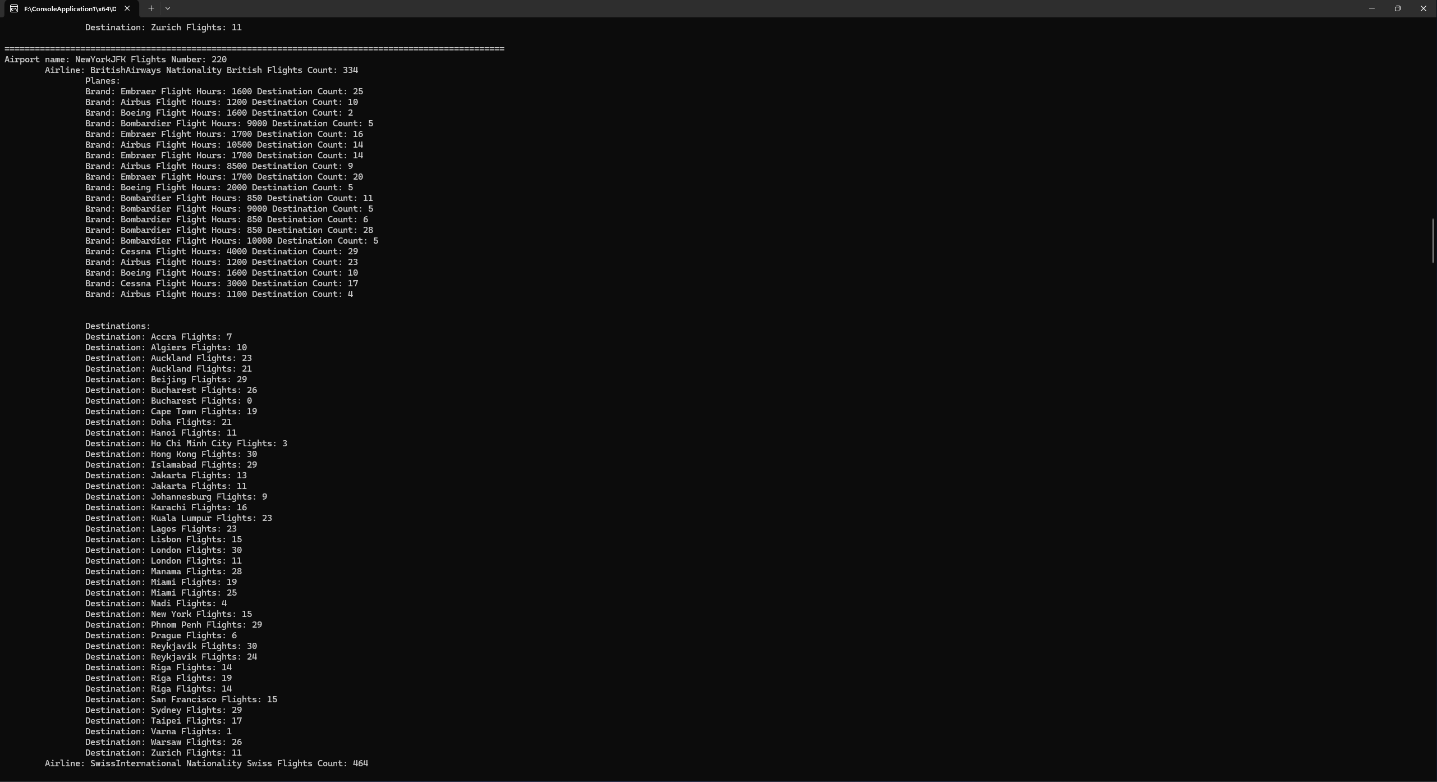
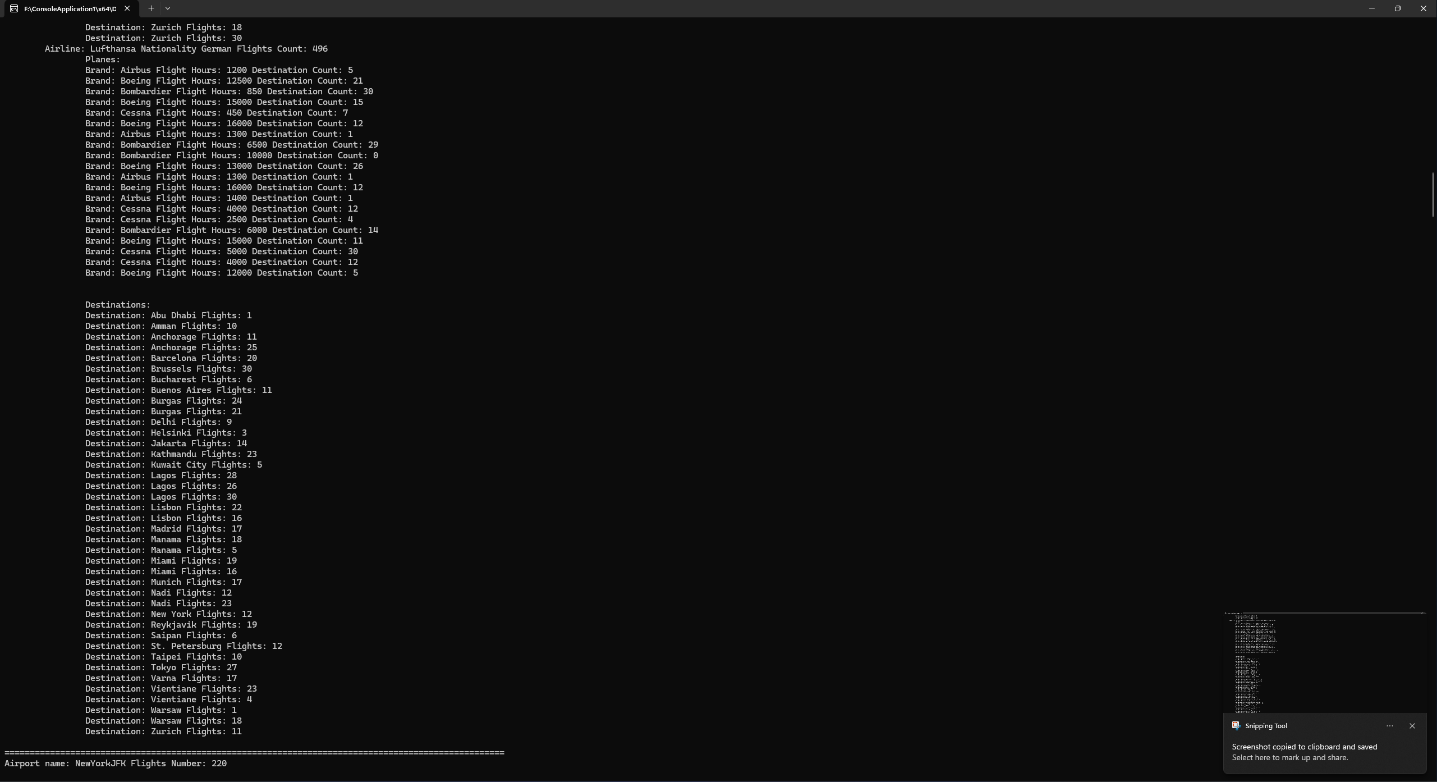
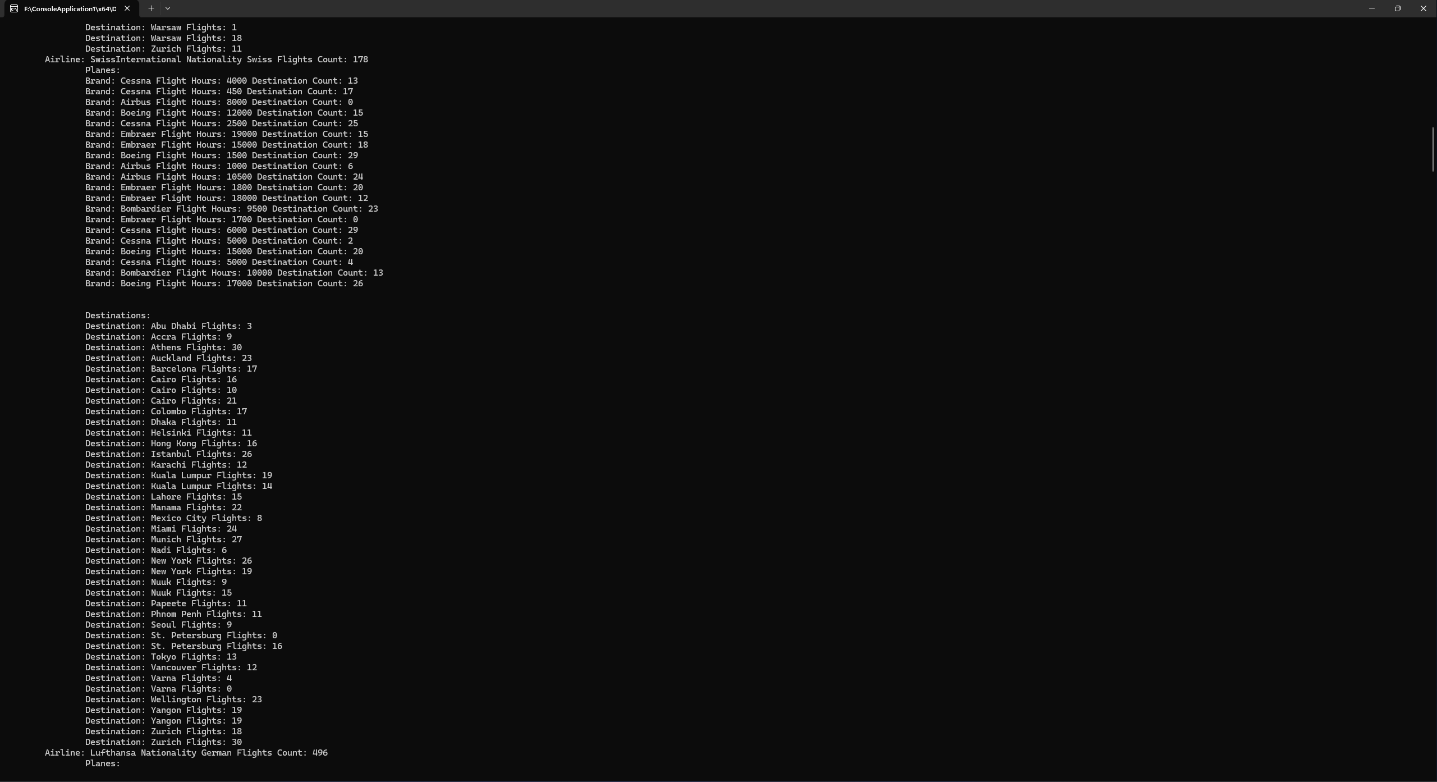
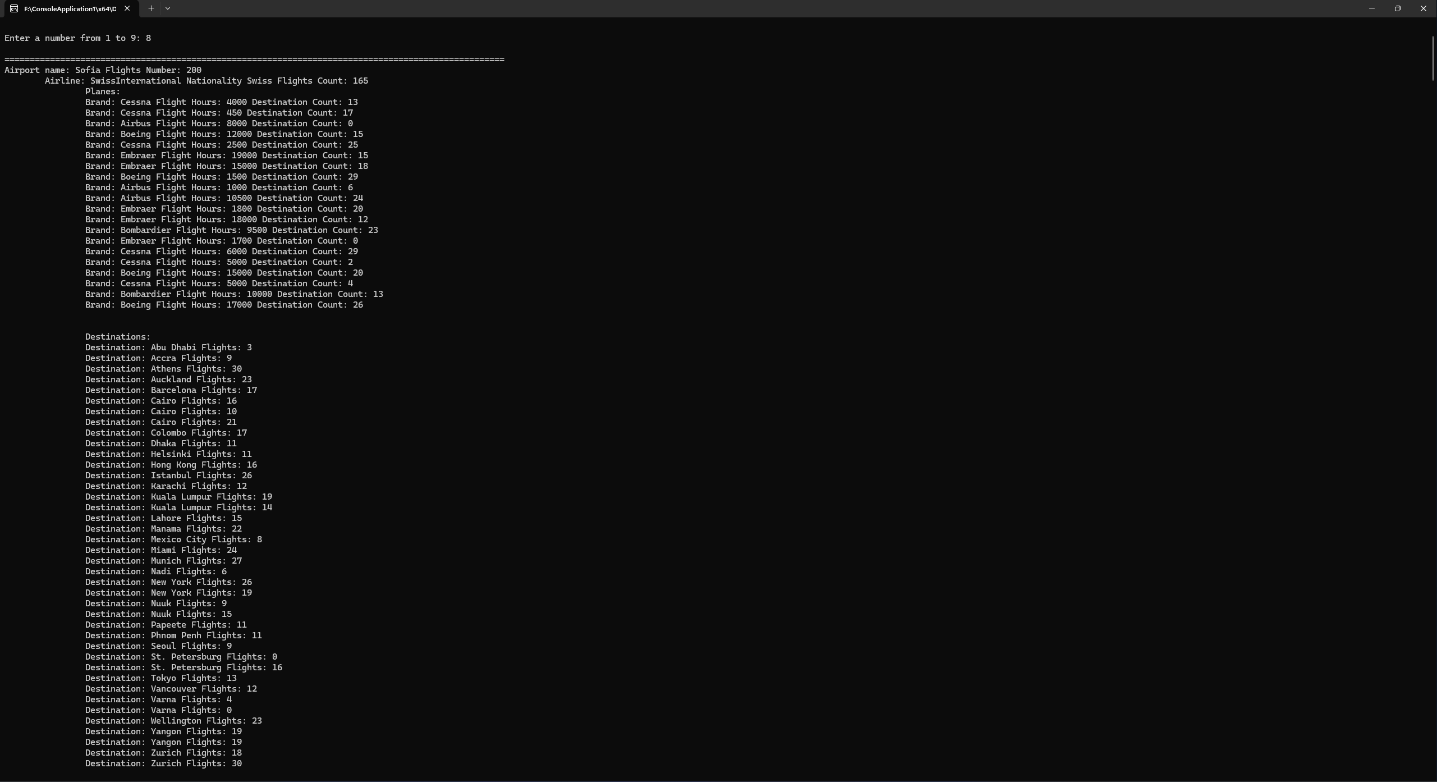
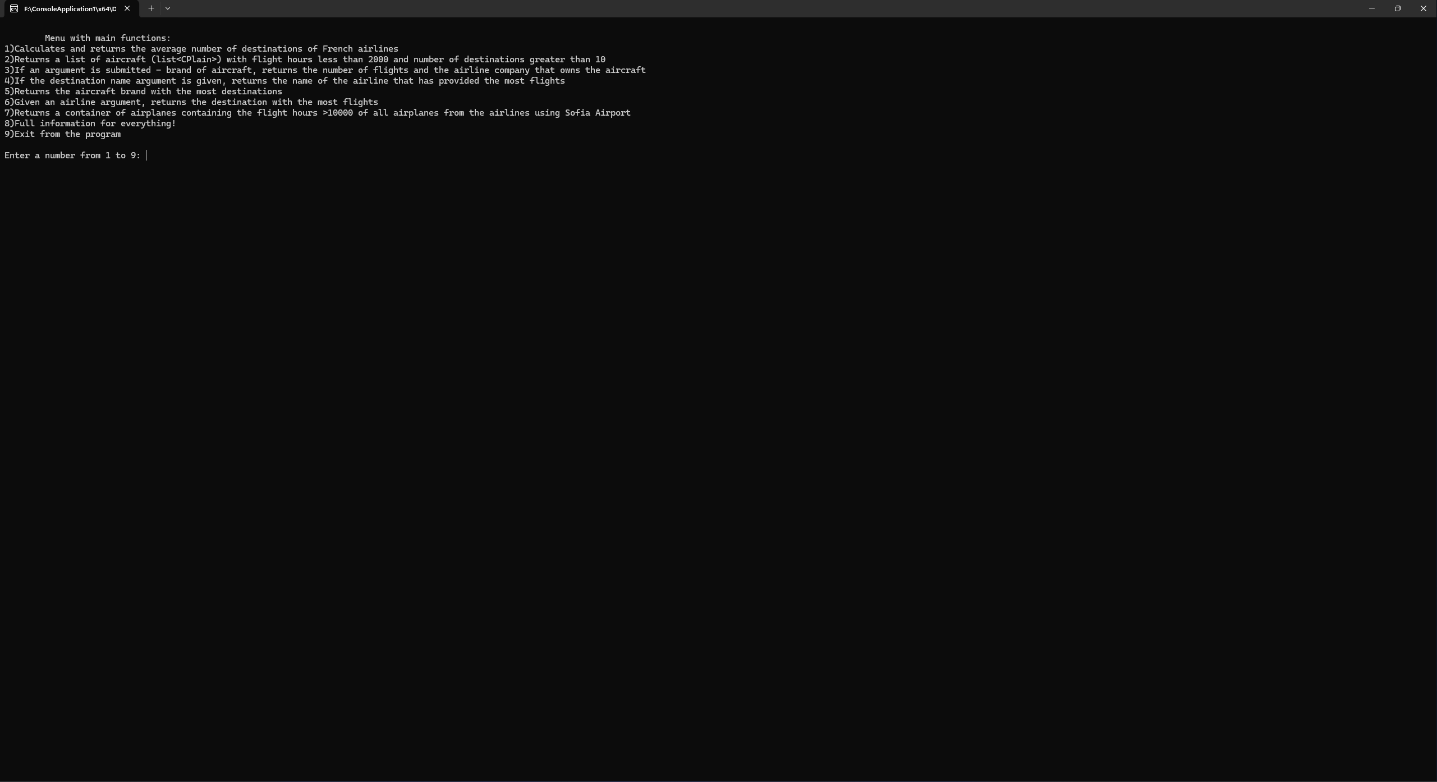
* **Зареждане на данните за работа**
* **Меню за работа с данните и извеждане на различни справки**

**IV. Тестови резултати**

**Нужните файлове за работа с програмата:**

****

Примерна работа на програмата:



**IV.Source код**

**#include <iostream>**

**#include <string>**

**#include <vector>**

**#include <iterator>**

**#include <algorithm>**

**#include <list>**

**#include <map>**

**#include <sstream>**

**#include <fstream>**

**using namespace std;**

**class CPlane {**

**private:**

**string brand;**

**int flightHours;**

**double altitude;**

**public:**

**CPlane() : brand(""), flightHours(0) {}**

**CPlane(string bbrand, int fFlightHours) {**

**brand = bbrand;**

**flightHours = fFlightHours;**

**}**

**CPlane(const CPlane &other)**

**{**

**brand = other.brand;**

**flightHours = other.flightHours;**

**}**

**//=======================================================**

**string GetBrand() const {**

**return brand;**

**}**

**int GetFlightHours() const {**

**return flightHours;**

**}**

**void SetBrand(string br)**

**{**

**this->brand = br;**

**}**

**void SetFlightHours(int flH)**

**{**

**this->flightHours = flH;**

**}**

**//=======================================================**

**CPlane& operator=(const CPlane& other)**

**{**

**brand = other.brand;**

**flightHours = other.flightHours;**

**return \*this;**

**}**

**bool operator<(const CPlane& other) const {**

**return this->altitude < other.altitude;**

**}**

**//=======================================================**

**static vector<CPlane> LoadFromFile(const string& filename) {**

**vector<CPlane> planes;**

**ifstream file(filename);**

**if (file.is\_open()) {**

**while (!file.eof()) {**

**string brand;**

**int flightHours;**

**file >> brand >> flightHours;**

**CPlane plane(brand, flightHours);**

**planes.push\_back(plane);**

**}**

**file.close();**

**}**

**else {**

**cerr << "Unable to open file: " << filename << endl;**

**}**

**return planes;**

**}**

**};**

**class CAirtravel**

**{**

**private:**

**string nameOfAirline;**

**string nationality;**

**map<CPlane, unsigned int> \*planeAndDestinationsCount=nullptr;**

**multimap<string, unsigned int> \*destinationAndFlightsCount=nullptr;**

**double altitude;**

**public:**

**CAirtravel()**

**{**

**nameOfAirline = "";**

**nationality = "";**

**}**

**CAirtravel(const CAirtravel& other)**

**{**

**nameOfAirline = other.nameOfAirline;**

**nationality = other.nationality;**

**planeAndDestinationsCount = other.planeAndDestinationsCount;**

**destinationAndFlightsCount = other.destinationAndFlightsCount;**

**}**

**CAirtravel(string a, string n)**

**{**

**nameOfAirline = a;**

**nationality = n;**

**planeAndDestinationsCount = new map<CPlane, unsigned int>();**

**destinationAndFlightsCount = new multimap<string, unsigned int>();**

**}**

**//=======================================================**

**string GetNameOfAirline() const {**

**return nameOfAirline;**

**}**

**void SetNameOfAirline(string n)**

**{**

**this->nameOfAirline = n;**

**}**

**string GetNationality() const {**

**return nationality;**

**}**

**void SetNationality(string n)**

**{**

**this->nationality = n;**

**}**

**map<CPlane, unsigned int> GetPlaneAndDestinationsCount()const**

**{**

**return \*planeAndDestinationsCount;**

**}**

**multimap<string, unsigned int> GetDestinationAndFlightsCount()const**

**{**

**return \*destinationAndFlightsCount;**

**}**

**//=======================================================**

**void АddPlane(const CPlane& plane, unsigned int destinations)**

**{**

**planeAndDestinationsCount->insert({ plane,destinations });**

**}**

**void AddDestination(const string& destination, unsigned int flights) {**

**\*destinationAndFlightsCount->insert({ destination, flights });**

**}**

**//=======================================================**

**bool operator<(const CAirtravel& other) const {**

**return this->altitude < other.altitude;**

**}**

**//=======================================================**

**int AvgDestinationCountOfAllFrenchAirlines()const**

**{**

**int sum = 0;**

**if (nationality == "French")**

**{**

**for (const auto& m : \*planeAndDestinationsCount)**

**{**

**sum += m.second;**

**}**

**}**

**return round(sum / static\_cast<double>(planeAndDestinationsCount->size()));**

**}**

**list<CPlane> PlanesWithCriteria2000And10() const {**

**list<CPlane> result;**

**for (const auto& plane : \*planeAndDestinationsCount) {**

**if (plane.first.GetFlightHours() < 2000 && plane.second > 10) {**

**result.push\_back(plane.first);**

**}**

**}**

**return result;**

**}**

**static vector<CAirtravel> LoadFromFile(const string& filename) {**

**vector<CAirtravel> airlines;**

**ifstream file(filename);**

**if (file.is\_open()) {**

**while (!file.eof()) {**

**string nameOfAirline;**

**string nationality;**

**file >> nameOfAirline >> nationality;**

**CAirtravel airline(nameOfAirline, nationality);**

**airlines.push\_back(airline);**

**}**

**file.close();**

**}**

**else {**

**cerr << "Unable to open file: " << filename << endl;**

**}**

**return airlines;**

**}**

**};**

**class CAirport**

**{**

**private:**

**string airportName;**

**unsigned int n\_flights;**

**multimap<CAirtravel,unsigned int> \*airlinesAndFlightsCount = nullptr;**

**public:**

**CAirport()**

**{**

**airportName = "";**

**n\_flights = 0;**

**}**

**CAirport(const CAirport& other)**

**{**

**airportName = other.airportName;**

**n\_flights = other.n\_flights;**

**airlinesAndFlightsCount = other.airlinesAndFlightsCount;**

**}**

**CAirport(string a, unsigned int n)**

**{**

**airportName = a;**

**n\_flights = n;**

**airlinesAndFlightsCount = new multimap<CAirtravel, unsigned int>();**

**}**

**//=======================================================**

**string GetAirportName() const {**

**return airportName;**

**}**

**unsigned int GetFlightsCount() const {**

**return n\_flights;**

**}**

**void setAirportName(string n)**

**{**

**this->airportName = n;**

**}**

**void setN\_flights(unsigned int n )**

**{**

**this->n\_flights = n;**

**}**

**multimap<CAirtravel, unsigned int> GetAirlinesAndFlightsCount()const**

**{**

**return \*airlinesAndFlightsCount;**

**}**

**//=======================================================**

**void AddAirline(const CAirtravel& airline, unsigned int flights)const {**

**airlinesAndFlightsCount->insert({ airline, flights });**

**}**

**//=======================================================**

**void ByPlaneBrandGetFlightsCountAndAirline(string brand)const**

**{**

**for (const auto& entry : \*airlinesAndFlightsCount)**

**{**

**for (const auto& plane : entry.first.GetPlaneAndDestinationsCount())**

**{**

**if (plane.first.GetBrand() == brand)**

**{**

**cout <<"\tAirline: " << entry.first.GetNameOfAirline() << " \tPlane Brand: "**

**<< plane.first.GetBrand() << " \tDestination-Flight Count of the Plane: " << plane.second<<endl;**

**}**

**}**

**}**

**}**

**void ByDestinationGetAirlineWithMostFlights(string destination)const**

**{**

**unsigned int max = 0;**

**for (const auto& entry : \*airlinesAndFlightsCount)**

**{**

**for (const auto& dest : entry.first.GetDestinationAndFlightsCount())**

**{**

**if (dest.second > max && dest.first == destination)**

**max = dest.second;**

**}**

**}**

**for (const auto& entry : \*airlinesAndFlightsCount)**

**{**

**//cout << "Airline: " << entry.first.GetNameOfAirline() << endl;**

**for (const auto& dest : entry.first.GetDestinationAndFlightsCount())**

**{**

**if (dest.first == destination && dest.second == max)**

**{**

**cout << "\tAirline the the most flights done:" << entry.first.GetNameOfAirline()<<endl;**

**}**

**}**

**}**

**}**

**void PlaneBrandWithMaxDestination()const**

**{**

**for (const auto& entry : \*airlinesAndFlightsCount)**

**{**

**unsigned int max = 0;**

**for (const auto& plane : entry.first.GetPlaneAndDestinationsCount())**

**{**

**if (plane.second > max) max = plane.second;**

**}**

**cout << "Airline " << entry.first.GetNameOfAirline() << endl;**

**for (const auto& plane : entry.first.GetPlaneAndDestinationsCount())**

**{**

**if (plane.second==max)**

**{**

**cout <<"\tThe aircraft brand with the most destinations --> " << plane.first.GetBrand()**

**<<" DestinationsCount:"<<plane.second << " Max:" << max << endl;**

**}**

**}**

**}**

**}**

**void ByAirlineGetDestinationWithMostFlights(string airline)const**

**{**

**for (const auto& entry : \*airlinesAndFlightsCount)**

**{**

**if (entry.first.GetNameOfAirline() == airline)**

**{**

**unsigned int max = 0;**

**for (const auto& dest : entry.first.GetDestinationAndFlightsCount())**

**{**

**if (dest.second > max) max = dest.second;**

**}**

**cout << "Airline: " << entry.first.GetNameOfAirline() << endl;**

**for (const auto& dest : entry.first.GetDestinationAndFlightsCount())**

**{**

**if (entry.first.GetNameOfAirline() == airline && dest.second == max)**

**{**

**cout<<"\tDestination With Most Flights:" << dest.first<<" Flights Count:"<<dest.second<<endl;**

**}**

**}**

**}**

**}**

**}**

**list<CPlane> PlanesWithCriteria10000AndSofia() const {**

**list<CPlane> result;**

**for (const auto& airline : \*airlinesAndFlightsCount)**

**{**

**for (const auto& plane : airline.first.GetPlaneAndDestinationsCount())**

**{**

**if (plane.first.GetFlightHours() > 10000 && airportName == "Sofia")**

**{**

**result.push\_back(plane.first);**

**}**

**}**

**}**

**return result;**

**}**

**void PlanesWithCriteria10000AndSofia2()const**

**{**

**for (const auto& airline : \*airlinesAndFlightsCount)**

**{**

**cout << "\tAirline: " << airline.first.GetNameOfAirline() << endl;**

**for (const auto& plane : airline.first.GetPlaneAndDestinationsCount())**

**{**

**if (plane.first.GetFlightHours() > 10000 && airportName == "Sofia")**

**{**

**cout << "\t\tBrand: " << plane.first.GetBrand() << " Flight Hours: " << plane.first.GetFlightHours() << endl;**

**}**

**}**

**}**

**}**

**static vector<CAirport> LoadFromFile(const string& filename) {**

**vector<CAirport> airports;**

**ifstream file(filename);**

**if (file.is\_open()) {**

**while (!file.eof()) {**

**string airportName;**

**unsigned int n\_flights;**

**file >> airportName >> n\_flights;**

**CAirport airport(airportName, n\_flights);**

**airports.push\_back(airport);**

**}**

**file.close();**

**}**

**else {**

**cerr << "Unable to open file: " << filename << endl;**

**}**

**return airports;**

**}**

**};**

**unsigned generateRandomNumber(unsigned int min, unsigned int max)**

**{**

**return min + (rand() % (max - min + 1));**

**}**

**vector<string> LoadFromFile(const string& filename) {**

**vector<string> destinations;**

**ifstream file(filename);**

**if (file.is\_open()) {**

**string destination;**

**while (getline(file, destination)) {**

**destinations.push\_back(destination);**

**}**

**file.close();**

**}**

**else {**

**cerr << "Unable to open file: " << filename << endl;**

**}**

**return destinations;**

**}**

**void WriteAirportsDataToFile(const vector<CAirport>& airports, const string& filename) {**

**ofstream file(filename);**

**if (!file.is\_open()) {**

**cerr << "Failed to open file: " << filename << endl;**

**return;**

**}**

**for (const auto& airport : airports) {**

**file << "===================================================================================================" << endl;**

**file << "Airport name: " << airport.GetAirportName() << " Flights Number: " << airport.GetFlightsCount() << endl;**

**for (const auto& airline : airport.GetAirlinesAndFlightsCount()) {**

**file << "\tAirline: " << airline.first.GetNameOfAirline() << " Nationality " << airline.first.GetNationality()**

**<< " Flights Count: " << airline.second << endl;**

**file << "\t\tPlanes:" << endl;**

**for (const auto& plane : airline.first.GetPlaneAndDestinationsCount()) {**

**file << "\t\tBrand: " << plane.first.GetBrand() << " Flight Hours: " << plane.first.GetFlightHours()**

**<< " Destination Count: " << plane.second << endl;**

**}**

**file << endl << endl;**

**file << "\t\tDestinations:" << endl;**

**for (const auto& dest : airline.first.GetDestinationAndFlightsCount()) {**

**file << "\t\tDestination: " << dest.first << " Flights: " << dest.second << endl;**

**}**

**}**

**file << endl;**

**}**

**file.close();**

**}**

**int main()**

**{**

**srand(time(0));**

**vector<CPlane> planes = CPlane::LoadFromFile("planes.txt");**

**vector<CAirtravel> airlines = CAirtravel::LoadFromFile("airlines.txt");**

**vector<CAirport> airports = CAirport::LoadFromFile("airports.txt");**

**vector<string> destinations = LoadFromFile("destinations.txt");**

**for (auto& airline : airlines)**

**{**

**for (int i = 0; i < 20; i++)**

**{**

**int n = generateRandomNumber(0, 49);**

**airline.АddPlane(planes[n], generateRandomNumber(0, 30));**

**airline.AddDestination(destinations[generateRandomNumber(0, 99)], generateRandomNumber(0, 30));**

**airline.AddDestination(destinations[generateRandomNumber(0, 99)], generateRandomNumber(0, 30));**

**}**

**}**

**for (const auto& airport : airports)**

**{**

**for (int i = 0; i < 4; i++)**

**{**

**int n = generateRandomNumber(0, 9);**

**airport.AddAirline(airlines[n], generateRandomNumber(100, 500));**

**}**

**}**

**//==========================================================================================================**

**int menu;**

**string input;**

**do**

**{**

**cout << endl;**

**cout << "\tMenu with main functions:" << endl;**

**cout << "1)Calculates and returns the average number of destinations of French airlines" << endl;**

**cout << "2)Returns a list of aircraft (list<CPlain>) with flight hours less than 2000 and number of destinations greater than 10" << endl;**

**cout << "3)If an argument is submitted - brand of aircraft, returns the number of flights and the airline company that owns the aircraft" << endl;**

**cout << "4)If the destination name argument is given, returns the name of the airline that has provided the most flights" << endl;**

**cout << "5)Returns the aircraft brand with the most destinations" << endl;**

**cout << "6)Given an airline argument, returns the destination with the most flights" << endl;**

**cout << "7)Returns a container of airplanes containing the flight hours >10000 of all airplanes from the airlines using Sofia Airport" << endl;**

**cout << "8)Full information for everything!" << endl;**

**cout << "9)Exit from the program" << endl;**

**cout << endl << "Enter a number from 1 to 9: ";**

**do**

**{**

**cin >> menu;**

**} while (menu < 1 || menu>9);**

**cout << endl;**

**switch (menu)**

**{**

**case 1:**

**for (const auto& airport : airports)**

**{**

**cout << "Airport name: " + airport.GetAirportName() << endl;**

**for (const auto& airline : airport.GetAirlinesAndFlightsCount())**

**{**

**if (airline.first.AvgDestinationCountOfAllFrenchAirlines() != 0)**

**cout << "Average number of destinations of French airlines: " << airline.first.AvgDestinationCountOfAllFrenchAirlines() << endl;**

**else cout << "Not a French airline!" << endl;**

**}**

**}**

**break;**

**case 2:**

**for (const auto& airport : airports)**

**{**

**cout << "Airport name: " + airport.GetAirportName() << endl;**

**for (const auto& airline : airport.GetAirlinesAndFlightsCount())**

**{**

**cout << "\tAirLine name: " << airline.first.GetNameOfAirline() << " Nationality: " << airline.first.GetNationality() << endl;**

**for (auto& plane : airline.first.PlanesWithCriteria2000And10())**

**{**

**cout << "\t\tBrand: " << plane.GetBrand() << " Flight Hours: " << plane.GetFlightHours() << endl;**

**}**

**}**

**}**

**break;**

**case 3:**

**cout << "Enter Brand:";**

**getline(cin.ignore(1000, '\n'), input);**

**cout << endl;**

**for (const auto& airport : airports)**

**{**

**cout << "Airport name: " + airport.GetAirportName() << " Flights Count: " << airport.GetFlightsCount() << endl;**

**airport.ByPlaneBrandGetFlightsCountAndAirline(input);**

**cout << endl << endl;**

**}**

**break;**

**case 4:**

**cout << "Enter Destination:";**

**getline(cin.ignore(1000, '\n'), input);**

**cout << endl;**

**for (const auto& airport : airports)**

**{**

**cout << "Airport name: " + airport.GetAirportName() << endl;**

**airport.ByDestinationGetAirlineWithMostFlights(input);**

**cout << endl << endl;**

**}**

**break;**

**case 5:**

**for (const auto& airport : airports)**

**{**

**cout << "Airport name: " + airport.GetAirportName() << endl;**

**airport.PlaneBrandWithMaxDestination();**

**cout << endl << endl;**

**}**

**break;**

**case 6:**

**cout << "Enter Airline:";**

**getline(cin.ignore(1000, '\n'), input);**

**cout << endl;**

**for (const auto& airport : airports)**

**{**

**cout << "Airport name: " + airport.GetAirportName() << endl;**

**airport.ByAirlineGetDestinationWithMostFlights(input);**

**cout << endl << endl;**

**}**

**break;**

**case 7:**

**for (const auto& airport : airports)**

**{**

**if (airport.GetAirportName() == "Sofia") {**

**cout << "Airport name: " + airport.GetAirportName() << endl;**

**for (const auto& plane : airport.PlanesWithCriteria10000AndSofia())**

**{**

**cout << "\t\tBrand: " << plane.GetBrand() << " Flight Hours: " << plane.GetFlightHours() << endl;**

**}**

**}**

**if (airport.GetAirportName() == "Sofia")**

**{**

**cout << endl;**

**airport.PlanesWithCriteria10000AndSofia2();**

**}**

**}**

**break;**

**case 8:**

**for (const auto& airport : airports)**

**{**

**cout << "===================================================================================================" << endl;**

**cout << "Airport name: " << airport.GetAirportName() << " Flights Number: " << airport.GetFlightsCount() << endl;**

**for (const auto& airline : airport.GetAirlinesAndFlightsCount())**

**{**

**cout << "\tAirline: " << airline.first.GetNameOfAirline() << " Nationality " << airline.first.GetNationality()**

**<< " Flights Count: " << airline.second << endl;**

**cout << "\t\tPlanes:" << endl;**

**for (const auto& plane : airline.first.GetPlaneAndDestinationsCount())**

**{**

**cout << "\t\tBrand: " << plane.first.GetBrand() << " Flight Hours: " << plane.first.GetFlightHours()**

**<< " Destination Count: " << plane.second << endl;**

**}**

**cout << endl << endl;**

**cout << "\t\tDestinations:" << endl;**

**for (const auto& plane : airline.first.GetDestinationAndFlightsCount())**

**{**

**cout << "\t\tDestination: " << plane.first << " Flights: " << plane.second << endl;**

**}**

**}**

**cout << endl;**

**}**

**WriteAirportsDataToFile(airports, "airports\_data.txt");**

**break;**

**case 9: exit(1); break;**

**default: cout << "ERROR!"; system("PAUSE"); exit(1);**

**}**

**} while (menu != 9);**

**return 0;**

**}**