ТЕХНИЧЕСКИ УНИВЕРСИТЕТ – ВАРНА

Факултет по изчислителна техника и автоматизация

Катедра по компютърни науки и технологии

Компютърни системи и технологии

КУРСОВ ПРОЕКТ

по

Обектно-Ориентирано Програмиране – Част 1

Изготвила:

Селин

Фак. №:

Група:

Летен Семестър 2020/2021

Курсов Проект №2

Програма за обработка на самолети

I. Да се дефинира клас CPlane, съхраняващ информация за марка на самолети и летателни часове с необходимите конструктори, методи и оператори.

II. Да се дефинира клас CAirtravel, съхраняващ информация за име на авиокомпанията, националност, брой дестинации и самолети, като последните са съхранени в контейнер map<CPlane, 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 авиокомпании;
- Връща списък от самолети (list<CPlane>), с летателни часове по-малко от 2000 и брой дестинации повече от 10;

III. 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 марка 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.

Обяснение:

1) `class Plane { … };` :

- Клас който съхранява информация за марка на самолет и неговите летателни часове.
- Функциите за четене на информация са **`getType()`** и **`getHours()`.** Те се използват предимно когато искаме да вземем определена информация за самолета.
- `operator <` сравнява летателните часове на lhs с на rhs и ако lhs е по голямо от rhs връща false, ако не true.
- `operator ==` сравнява марката на самолета.
- `operator >>` входен поток който ще ни трябва в втория клас.

2) `class Airline { … };` :

Ако примерно имаме авиокомпания “Turkish Airlines”, с кодово име “THY”, и данните на тази авиокомпания са:

```
1   THY
2   Turkish 5
3   A330 1253 SOF 11
4   B777-300ER 9614 VIE 35
5   B787 4759 JFK 25
6   A350 1843 ADB 12
7   A350 1243 VIE 16
```

- Класът има член променливи които съхраняват:

`string s_Airline;` - **Името** на авиокомпанията

`string s_Nationality;` - **Националността** на авиокомпанията

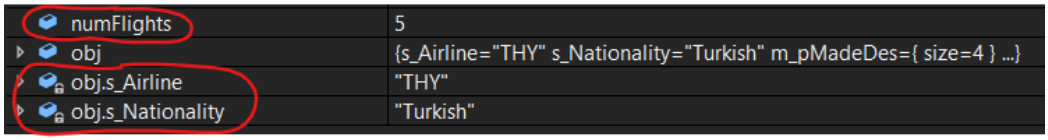
`map<Plane, unsigned> m_pMadeDes;` - Map който съхранява в key value самолет и в mapped value брой дестинациите на този самолет.
(Map **не** може да има еднакви key value-та.)

`map<Plane, string> m_pDest;` - Map който съхранява в key value самолет и в mapped value името на дестинацията на този самолет.

`multimap<string, unsigned> mm_airlinesDest;` - Multimap който съхранява името на дестинацията и колко брой полети до нея са направени.
(Multimap може да има еднакви key value-та като в един град може да има няколко летища и например да има дестинации с имена NYC – JFK и NYC - LGA)

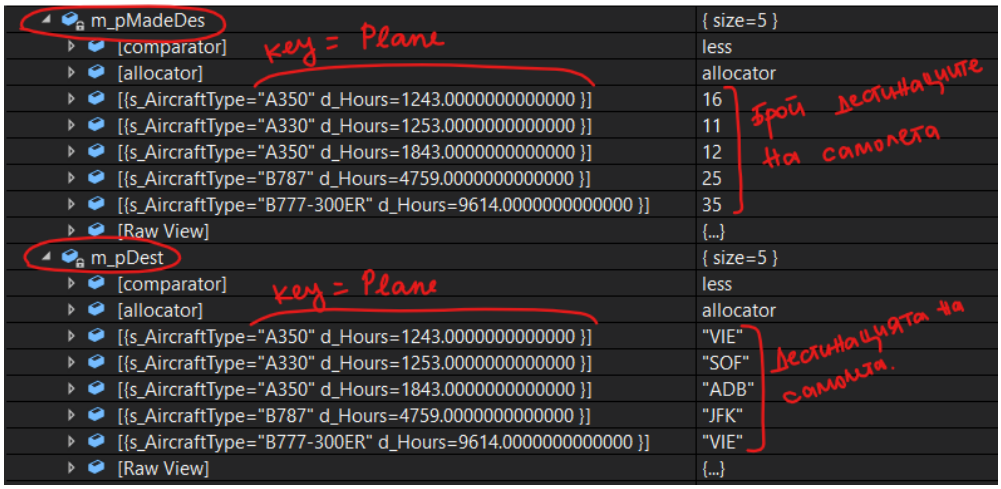
- Функцията `friend istream& operator >>` чете информацията на авиокомпанията.

a) Функцията първо по ред чете името на авиокомпанията, националността ѝ и в момента, колко броя самолета има – това число е сложено само за да стопира цикъла `for`.

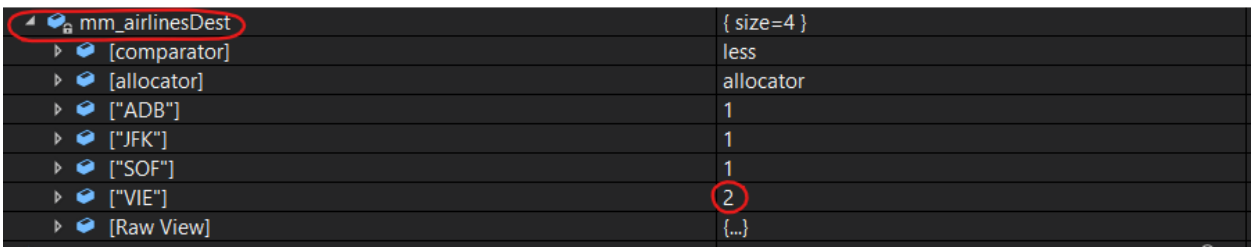


b) С `for` четем на авиокомпанията самолетите, в подаденото летище. Първо четем данните на първия самолет – първи клас, дестинацията му и досега на колко дестинации е бил. Тези данни се съхраняват временно в променливите създадени локално в функцията.

След като сме съхранили данните, почваме да слагаме в необходимите контейнери.



За контейнера **`mm_airlinesDest`** който съдържа колко полета до определена дестинация има, трябва първо да се създаде такава дестинация но ако вече я има, трябва в mapped value-то ѝ да се увеличи броя, защото вече до тази дестинация е отишъл един самолет.



- Функцията `friend ostream& operator <<` извежда информацията на конзолата. Тази функция предимно е за визуализация. Показва:

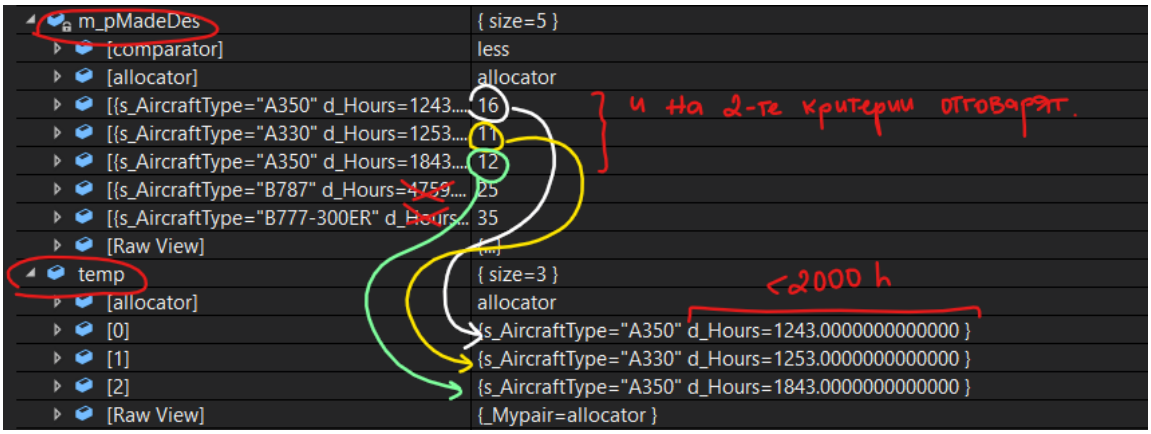
```
+--------------+-----------------------------------------------------+
| Airline      | THY                                                 |
+--------------+-----------------------------------------------------+
| Nationality  | Turkish                                             |
+--------------+-----------------------------------------------------+
| Destinations | 4                                                   |
+--------------+---------------+--------------+---------------------+
| Destination  | Aircraft Type | Flight Hours | Made Destinations   |
+--------------+---------------+--------------+---------------------+
| VIE          | A350          | 1243         | 16                  |
+--------------+---------------+--------------+---------------------+
| SOF          | A330          | 1253         | 11                  |
+--------------+---------------+--------------+---------------------+
| ADB          | A350          | 1843         | 12                  |
+--------------+---------------+--------------+---------------------+
| JFK          | B787          | 4759         | 25                  |
+--------------+---------------+--------------+---------------------+
| VIE          | B777-300ER    | 9614         | 35                  |
+--------------+---------------+--------------+---------------------+
| Tot Flights  | 5                                                   |
+--------------+-----------------------------------------------------+
```

Тука

- Функциите **getAirline()**, **getNation()**, **GetPlaneMD()**, **GetPlaneMap()**, **GetMM**() предимно са за третия клас за да може да достъпи необходимата информация.

- Функцията `list<Plane>` **`listPlanes()`** `const` връща списък от самолети които са с летателни часове по-малко от 2000 часа и са посещавали повече дестинации от 10.

a) Създаваме list (списък) който ще съхранява самолетите и ще действа за връщането на резултат.
b) Търсим и намираме от контейнера **`m_pMadeDes`** самолетите. Ако има самолет с тези критерии, той се записва в списъка.
c) Създадения списък се връща като резултат.



- Функцията `unsigned averageFrench() const` връща френските авиокомпании. Тази функция ще се използва в клас `Airport` за изчисляване на средния брой дестинации на френските авиокомпании.

3) `class Airport { … };` :

- Класът има член променливи които съхраняват:

`string s_Airport;` - Името на летището

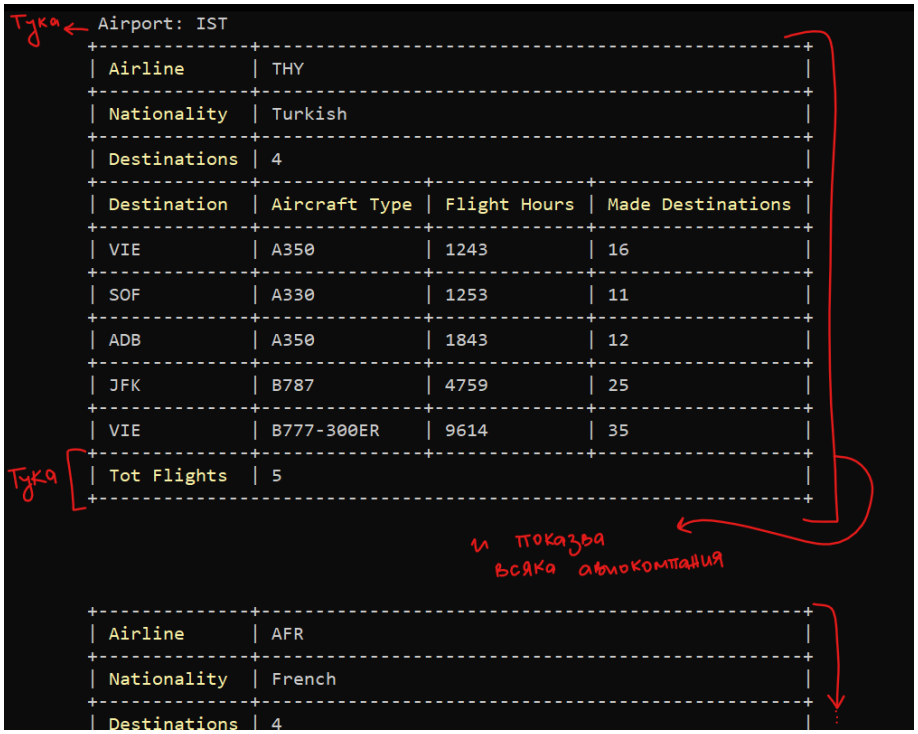
`unsigned u_numFlights;` - Брой полети в летището

`multimap<Airline, unsigned> airline_data;` - Multimap който съхранява авиокомпанията и колко броя полети осигурява в момента.

- Експлицитния конструктор `Airport(const string& AirportName, const string& fileName)` чете файл който се задава в main.

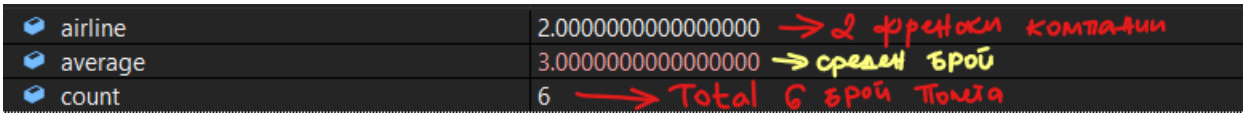
Първо от подадения параметър взимаме името на летището и го съхраняваме в член променливата `s_Airport`.
После ако файла се отвори, четем от него всяка авиокомпания и те се записват в multimap-а **airline_data**.
За брой полети в летището, функцията взима брой полетите на всяка авиокомпания и ги събира.

- Функцията `friend ostream& operator <<` извежда всяка авиокомпания и колко броя полета всяка авиокомпания има. Тази функция предимно е за визуализация на данните. Показва:



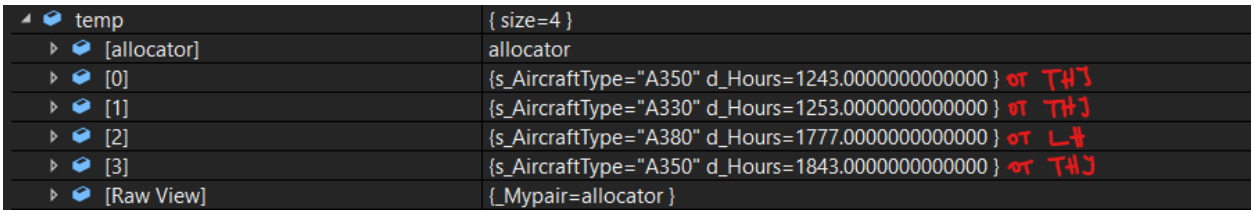
- Функцията `double franceBased()` изчислява и връща средния брой дестинации на френските авиокомпании.

Функцията взима от всичките френски авиокомпании (от клас `Airline`, функция `unsigned averageFrench()`) дестинациите им и ако функцията `unsigned averageFrench()` връща стойност, авиокомпанията се брои като френска и броя на френските авиокомпании се увеличава.
След като вече имаме брой дестинации и брой френски авиокомпании, ги делим за да намерим средния брой на дестинации.



- Функцията `void less2000more10()` взима списъка с самолети, които са с летателни часове по малко от 2000 и брой дестинации повече от 10, от втори клас и ги извежда на конзолата.

Създаваме списък който ще съхранява самолетите от всичките авиокомпании и ги извеждаме.



- Функцията `void flightsAirline(const string& aircraftType)` има параметър марка на самолет, връща броя полети и авиокомпанията.

a) Създаваме самолет с марката еквивалентен на подадения параметър, за да сравняваме.
b) С `for` взимаме едно по едно информацията на всяка авиокомпания.
c) На всяка авиокомпания, взимаме контейнера map<Plane, string> който съдържа името на самолета и дестинацията му – дестинацията в тази функция не е важна.
d) Ако типа на подадения параметър с на авиокомпанията самолета съвпадне, правим count++.
e) Извеждаме авиокомпаниите които имат такъв тип самолет.

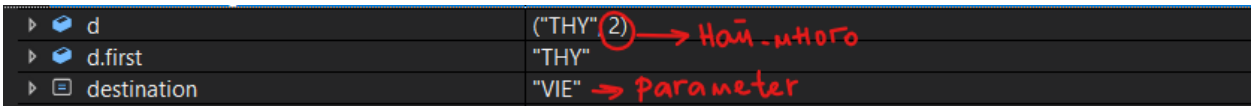
```
Search flight by aircraft type: a350

There are 2 flights made with A350 by THY
There is 1 flight made with A350 by AFR
There is 1 flight made with A350 by LH
```

(За функция `void flightsAirline`)

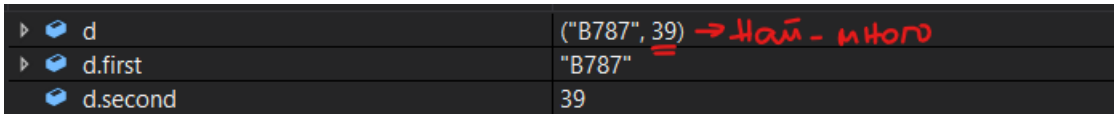
- Функцията `void destAirline(const string& destination)` има параметър дестинация, връща името на авиокомпанията, обезпечила най-много полети до нея;

1) Създаваме една двойка pair<string, unsigned> която е default-на, за да съхранява авиокомпанията която най-много е обезпечила полети до дестинацията в момента.
2) С `for` взимаме едно по едно информацията на всяка авиокомпания.
3) На всяка авиокомпания, взимаме контейнера (multimap<string, unsigned>), който съдържа името на дестинацията и колко пъти тя е посещавана.
4) Ако на текущата авиокомпания дестинацията **съвпада** с параметъра, и на тази авиокомпания **посещенията са повече от на двойката**, pair<string, unsigned> вече ще съдържа тази авиокомпания и нейните посещавания – защото до сега тя е посещавала дестинацията най много.
5) Когато свърши цикълът, вече имаме авиокомпанията с „най-много посещавания".
6) Извеждаме авиокомпанията.



- Функцията `void typeDest_self()` връща марката самолет с най-много брой дестинации:

1) Създаваме една двойка pair<string, unsigned> която е default-на, за да съхранява марката на самолета и неговия брой дестинации.
2) С `for` взимаме едно по едно информацията на всяка авиокомпания.
3) На всяка компания, взимаме контейнера (map<Plane, unsigned>), който съдържа информация за самолетите и броя на дестинациите им.
4) Ако на **текущата авиокомпания самолета** е посещавал **повече места** от на двойката, pair<string, unsigned> вече ще съдържа този тип самолет и броя на дестинациите му. Ако не е повече, самолета в двойката си остава.
5) Извеждаме марката на самолета с най-много брой дестинации.



- Функцията `void typeDest_general()` връща коя марка самолет е използвана най-много в летището.

1) Създаваме:
   - **map<Plane, string>** за да съхраним на авиокомпанията самолетите и техните дестинации,
   - **map<string, unsigned>** за да съхраним типа на самолета и колко пъти той е ползван в летището и
   - **multimap<unsigned, string>** за да намерим най-много ползвания самолет.
2) С `for` взимаме едно по едно информацията на всяка авиокомпания.
3) В контейнера **map<Plane, string>** слагаме самолетите и дестинациите им на текущата авиокомпания.
4) Ако в **map<string, unsigned>** нямаме самолет с типа на текущия, създаваме нов самолет който е ползван един път, а ако вече го има типа на самолета, увеличаваме броя на използването му.
5) За да намерим най-много използвания самолет трябва да знаем че:

- В map<**string**, unsigned> винаги има различни типове самолета защото **key-value** е **unique**, а в mapped-value може да има еднакви. Един map **сортира key value-то** но не и mapped value.
- За да намерим най използвания самолет (може да има няколко самолета), създаваме един **multimap**<unsigned, string>, който може да има няколко еднакви key value-та. Дефиниран е обратно на **map**<string, unsigned>.
- В момента, имаме multimap който на първи ред съхранява самолета който е използван най-малко. Използваме `reverse_iterator` и `rbegin()` за да намерим най-много използвания.

Например:

```
1  THY
2  Turkish 5
3  A330 1253 SOF 11
4  B777-300ER 9614 VIE 35
5  B787 4759 JFK 25
6  A350 1843 ADB 12
7  A350 1243 VIE 16
8
9  LH
10 German 3
11 A350 2134 MUC 12
12 B777 9614 VIE 35
13 A380 1777 VIE 23
14
15 DAL
16 French 3
17 A380 12134 JFK 22
18 A380 2462 JFK 21
19 B787 10614 MIA 39
20
21 AFR
22 French 5
23 A320 61334 SOF 19
24 A380 93852 VIE 27
25 A320 64 IST 2
26 A310 613 CDG 9
27 A350 6133 DOH 12
```

Имаме: 4 → A350 и Имаме 4 → A380

- Взимаме първия на ред самолет (най-много използвания), и с `equal_range(it->first)` гледаме дали има друг тип самолет използван колкото него.

- В примера имаме два типа самолета които се използвали най-много (четири пъти) и с `for (auto it = range.first; …)` открием втория тип, и ще го изведем на конзолата.

- С `for (const auto& i : total)` ще изведем колко пъти един самолет е ползван.

```
The most frequently used aircraft type is: A350 A380

List of how many times an aircraft type has been used:
A310 -> 1
A320 -> 2
A330 -> 1
A350 -> 4
A380 -> 4
B777 -> 1
B777-300ER -> 1
B787 -> 2
```

(за функция `void typeDest_general()`)

- Функцията `string airlineDest(const string& airlineName)` има параметър име на авиокомпания и връща дестинацията с най-много полети.

1) Създаваме авиокомпания с името еквивалентно на подадения параметър, за да сравняваме.
2) С `for` взимаме едно по едно информацията на всяка авиокомпания.
3) Ако на търсената авиокомпания името съвпадне с на текущата авиокомпания, броим кои места е посещавала най-много – sorting.
4) Връщаме дестинацията.

```
Most visited destination of a given airline:

Airline : thy

The most visited destination is: VIE
```

- Функцията `map<Plane, string> planesLAST()` връща контейнер от самолети, съдържащ летателните часове > 10000 на всички самолети от авиокомпаниите, ползващи (отиващи) към летище София.

1) Създаваме контейнер map<Plane, string>**m** за да съхраним самолетите и за да ги върнем като резултат.
2) С `for` взимаме едно по едно информацията на всяка авиокомпания.
3) В контейнера map<Plane, string>**result** слагаме самолетите които са с летателни часове > 10000 и дестинацията им която е до София, на текущата авиокомпания.
4) Връщаме **result**.

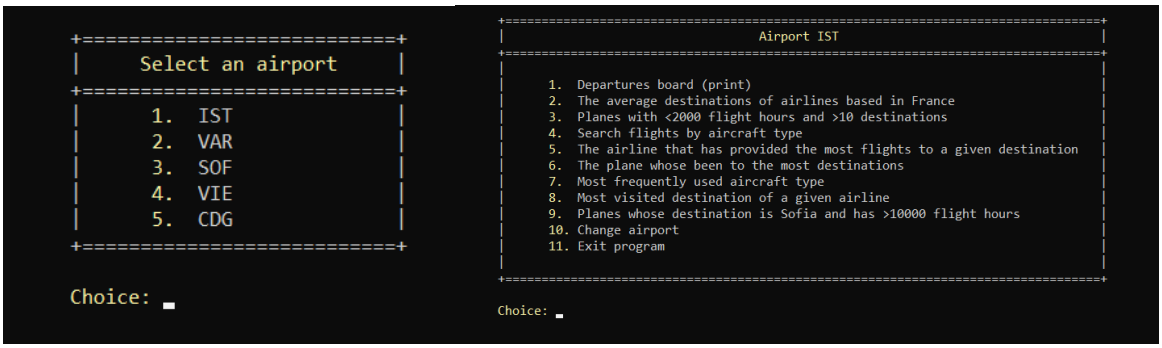
```
Planes whose destination is Sofia and has >10000 flight hours:

A320 61334 SOF
```

4) `#include "color.hpp":`

Това е header-only library което действа за оцветяване на текста в конзолата. Само е ползван за „по-красива“ визуализация на конзолата. Header-ния файл е взет от :

- https://github.com/imfl/color-console
- https://github.com/imfl/color-console/blob/master/include/color.hpp



(dye::light_yellow(“”))

5) Глобалния контейнер **map<string, string>** `city_to_Code`;

Този контейнер съдържа в key value-то си името на дестинация, а в mapped value-то си кодовото име на дестинацията.

```
map<string, string> city_to_Code = {
    {"VIENNA", "VIE"},
    {"ISTANBUL", "IST"},
    {"VARNA", "VAR"},
    {"SOFIA", "SOF"},
    {"DOHA", "DOH"},
    {"NEW YORK", "JFK"},
    {"MUNICH", "MUC"},
    {"MIAMI", "MIA"},
    {"PARIS", "CDG"}
};
```

6) `int main():`

A. В първото меню избираме на кое летище информацията бихме искали да видим. Тази информация е качена предварително с .txt файл. Ако файловете по някаква причина не се отворят, системата ни хвърля “FILE ERROR” и програмата спира да работи.

B. След като сме избрали на кое летище бихме искали да видим информацията, ни излиза второ меню, където има функции.

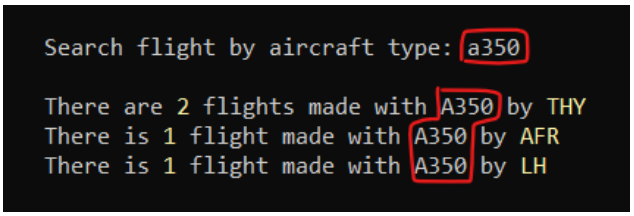
C. В началото на функцията int main( ), създаваме един флаг за изход, който ще има стойност, false. Двете менюта са в един while loop и този цикъл ще работи докато флага не стани true т.е. програмата ще работи докато не изберем “11. Exit program”.

```
case 11:
    exit = true;  // end while
    break;
```

D. Цикъла е създаден за да можем да променяме летищата до когато си искаме.

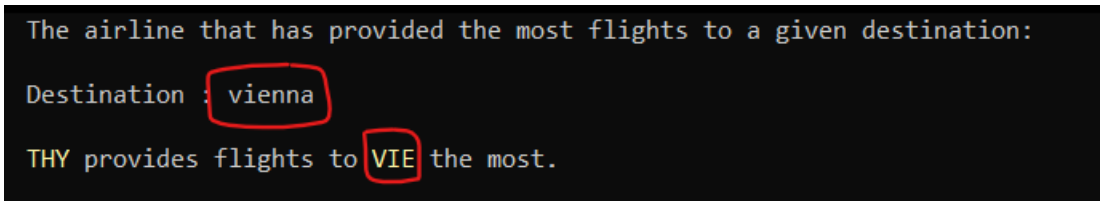
За „**Търсене на полети по подадена марка на самолет:**“ и „**Търсене на подадената авиокомпания, най-много полетите**“:

- Въведената ключова дума се превръща в големи букви, за по малко потребителски грешки.



За функцията която „**Връща името на авиокомпанията, обезпечила най-много полети до нея**“:

- Изисква се потребителя да въведе дестинация, и за по-малко грешки, дестинацията се превръща в дума с големи букви.
- В глобалния контейнер map<string, string> който съдържа в key value-то името на дестинация, а в mapped value-то кодовото име на дестинацията, се търси въведената от потребителя дестинация.
- Ако тази дестинация е намерена, параметъра за функцията става, на тази дестинация кодовото име, ако не, потребителя се уведомява че не е намерена такава дестинация.



Листинг на програмата:

Примерен текстови файл:

```
THY
Turkish 5
A330 1253 SOF 11
B777-300ER 9614 VIE 35
B787 4759 JFK 25
A350 1843 ADB 12
A350 1243 VIE 16

LH
German 3
A350 2134 MUC 12
B777 9614 VIE 35
A380 1777 VIE 23
```

```
DAL
French 3
A380 12134 JFK 22
A380 2462 JFK 21
B787 10614 MIA 39

AFR
French 5
A320 61334 SOF 19
A380 93852 VIE 27
A320 64 IST 2
A310 613 CDG 9
A350 6133 DOH 12
```

.cpp:

```cpp
#include <iostream>
#include <fstream>
#include <map>
#include <string>
#include <iterator>
#include <algorithm>
#include <list>
#include <iomanip>

#include "color.hpp"  // https://github.com/imfl/color-console

#define FILE_ERROR 0
#define DATA_ERROR 2

using namespace std;

int airportMenu();
int funcMenu(string);
map<string, string> city_to_Code = {
        {"VIENNA", "VIE"},
        {"ISTANBUL", "IST"},
        {"VARNA", "VAR"},
        {"SOFIA", "SOF"},
        {"DOHA", "DOH"},
        {"NEW YORK", "JFK"},
        {"MUNICH", "MUC"},
        {"MIAMI", "MIA"},
        {"PARIS", "CDG"}
};


class Plane {
private:
        string s_AircraftType;
        double d_Hours;

public:
        Plane() : d_Hours(0.0) {}
        Plane(const string& s_AircraftType, const double& d_Hours) {
                this->s_AircraftType = s_AircraftType;
                this->d_Hours = d_Hours;
        }

        const string& getType() const { return s_AircraftType; }
        const double& getHours() const { return d_Hours; }

        friend bool operator < (const Plane& lhs, const Plane& rhs) {
                return lhs.d_Hours < rhs.d_Hours;
        }
```

```
        friend bool operator == (const Plane& lhs, const Plane& rhs) {
                return lhs.s_AircraftType == rhs.s_AircraftType;
        }

        friend ostream& operator << (ostream& os, const Plane& obj) {
                os << std::left << setw(13) << obj.s_AircraftType << " | "
                        << std::left << setw(12) << obj.d_Hours;
                return os;
        }
        friend istream& operator >> (istream& is, Plane& obj) {
                is >> obj.s_AircraftType >> obj.d_Hours;
                return is;
        }
};


class Airline : public Plane {
private:
        string s_Airline;
        string s_Nationality;
        map<Plane, unsigned> m_pMadeDes;
        map<Plane, string> m_pDest;
        multimap<string, unsigned> mm_airlinesDest;

public:
        Airline() : s_Airline(""), s_Nationality("") {}
        Airline(const string& s_Airline, const string& s_Nationality) {
                this->s_Airline = s_Airline;
                this->s_Nationality = s_Nationality;
        }

        friend ostream& operator << (ostream& os, const Airline& obj) {
                os << "\t+-------------+-----------------------------------------------------+\n";
                os << "\t| " << std::left << setw(12) << dye::light_yellow("Airline") << " | " <<
std::left << setw(49) << obj.s_Airline << "|\n";
                os << "\t+-------------+-----------------------------------------------------+\n";
                os << "\t| " << std::left << setw(12) << dye::light_yellow("Nationality") << " | " <<
std::left << setw(49) << obj.s_Nationality << "|\n";
                os << "\t+-------------+-----------------------------------------------------+\n";
                os << "\t| " << std::left << setw(12) << dye::light_yellow("Destinations") << " | " <<
std::left << setw(49) << obj.mm_airlinesDest.size() << "|\n";
                os << "\t+-------------+--------------+-------------+------------------+\n";
                os << "\t| " << std::left << setw(12) << dye::light_yellow("Destination") << " | "
                        << std::left << setw(13) << dye::light_yellow("Aircraft Type") << " | "
                        << std::left << setw(12) << dye::light_yellow("Flight Hours") << " | "
                        << std::left << setw(17) << dye::light_yellow("Made Destinations") << " |\n";
                os << "\t+-------------+--------------+-------------+------------------+\n";

                for (auto it = obj.m_pMadeDes.begin(); it != obj.m_pMadeDes.end(); it++) {
                        os << "\t| " << std::left << setw(12) << obj.m_pDest.find(it->first)->second << "
| "
                                << setw(28) << it->first << " | "
                                << std::left << setw(17) <<it->second << " |\n";
                        os << "\t+-------------+--------------+-------------+------------------+\n";
                }
                return os;
        }

        friend istream& operator >> (istream& is, Airline& obj) {
                Plane temp;
                string destination;
                unsigned madeDes;
                int numFlights;

                is >> obj.s_Airline >> obj.s_Nationality >> numFlights;

                for (int i = 0; i < numFlights; i++) {
                        is >> temp;
                        is >> destination >> madeDes;
```

```
                        obj.m_pMadeDes.insert(make_pair(temp, madeDes));
                        obj.m_pDest.insert(make_pair(temp, destination));

                        auto a = obj.mm_airlinesDest.find(destination);

                        if (a == obj.mm_airlinesDest.end())    // not found
                                obj.mm_airlinesDest.insert(make_pair(destination, 1));
                        else {
                                a->second += 1;
                        }
                }
                return is;
        }


        const string& getAirline() const { return s_Airline; }
        const string& getNation() const { return s_Nationality; }

        auto GetPlaneMD() const {
                return m_pMadeDes;
        }
        auto GetPlaneMap() const {
                return m_pDest;
        }
        auto GetMM() const {
                return mm_airlinesDest;
        }

        unsigned getTotalPlanes() const { return m_pMadeDes.size(); }
        unsigned getTotalDiffDest() const { return mm_airlinesDest.size(); }   // returns total number of
destinations

        friend bool operator < (const Airline& lhs, const Airline& rhs) {
                return lhs.mm_airlinesDest < rhs.mm_airlinesDest;
        }
        friend bool operator == (const Airline& lhs, const Airline& rhs) {
                return lhs.m_pDest == rhs.m_pDest;
        }

        // Връща списък от самолети, с летателни часове по малко от 2000 и брой дестинации повече от 10
        // Returns list of planes, with <2000h and 10+ destinations
        list<Plane> listPlanes() const {
                list<Plane> temp;

                for (auto it = m_pMadeDes.begin(); it != m_pMadeDes.end(); it++) {
                        if (((*it).first.getHours() < 2000) && ((*it).second > 10))
                                temp.push_back(it->first);
                }
                return temp;
        }

        // Връща френските авиокомпании за 3-ти клас
        // Returns french airlines
        unsigned averageFrench() const {
                unsigned count = 0;
                if (s_Nationality == "French") {
                        count = getTotalDiffDest();
                }
                return count;
        }
};

class Airport {
        string s_Airport;
        unsigned u_numFlights;
        multimap<Airline, unsigned> airline_data;

public:
        Airport() : s_Airport(""), u_numFlights(0) {}
```

```
        const string& getAirport() const { return s_Airport; }

        Airport(const string& AirportName, const string& fileName) {
                s_Airport = AirportName;
                fstream ifile(fileName, ios_base::in);
                if (ifile.good()) {
                        while (!ifile.eof()) {
                                Airline temp;
                                ifile >> temp;
                                airline_data.insert(make_pair(temp, temp.getTotalPlanes()));
                                u_numFlights += temp.getTotalPlanes();
                        }
                }
                else
                        throw FILE_ERROR;
        }

        friend ostream& operator << (ostream& os, const Airport& obj) {
                os << "\t Airport: " << obj.s_Airport << endl;
                for (auto it = obj.airline_data.begin(); it != obj.airline_data.end(); it++) {
                        os << it->first
                                << "\t| " << std::left << setw(12) << dye::light_yellow("Tot Flights")
<< " | " << std::left << setw(49) << it->second << "|\n";
                        os << "\t+-----------------------------------------------------------------+" <<
"\n\n\n\n\n";
                }
                os << "\tThis airport has " << dye::light_yellow(obj.u_numFlights) << " flights!\n\n";
                return os;
        }

        // Изчислява средния брой дестинации на френските авиокомпании
        double franceBased() {
                unsigned count = 0;
                double airline = 0.0;
                double average = 0.0;

                for (auto it = airline_data.begin(); it != airline_data.end(); it++) {
                        count += it->first.averageFrench();
                        if (it->first.averageFrench() > 0)
                                airline++;
                }
                average = count / airline;
                return average;
        }

    // Показване на конзолата самолетите с летателни часове по-малко от 2000 и брой дестинации повече от
10
        void less2000more10() {
                list<Plane> temp;
                for (auto it = airline_data.begin(); it != airline_data.end(); it++) {
                        temp.merge((*it).first.listPlanes());
                }
                for (const auto& i : temp) {
                        cout << "\t" << dye::light_yellow(i.getType()) << " with " <<
dye::light_yellow(i.getHours()) << " hours" << endl;
                }
        }


        // При подаден аргумент - марка на самолет, връща броя полети и авиокомпанията
        // Returns the number of flights made with A330 by "Airline Name"
        void flightsAirline(const string& aircraftType) {
                Plane temp(aircraftType, 0.0);
                map<Plane, string> mp;

                for (auto it = airline_data.begin(); it != airline_data.end(); it++) {
                        mp = it->first.GetPlaneMap();
                        int count = 0;
```

```
                        for (auto i = mp.begin(); i != mp.end(); i++) {
                                if (i->first.getType() == temp.getType()) {
                                        count++;
                                }
                        }

                        if (count > 0) {
                                if (count == 1)
                                        cout << "\n\tThere is " << dye::light_yellow(count) << " flight
made with " << aircraftType << " by " << dye::light_yellow(it->first.getAirline());
                                else
                                        cout << "\n\tThere are " << dye::light_yellow(count) << " flights
made with " << aircraftType << " by " << dye::light_yellow(it->first.getAirline());
                        }
                }
        }


        // При подаден аргумент име на дестинацията, връща името на авиокомпанията, обезпечила най-много
полети
        // Returns the airline which has made the most flights to the given destination
        void destAirline(const string& destination) {
                multimap<string, unsigned> mm;
                pair<string, unsigned> d = make_pair("", 0);

                for (auto it = airline_data.begin(); it != airline_data.end(); it++) {
                        mm = it->first.GetMM();

                        for (auto i = mm.begin(); i != mm.end(); i++) {

                                if ((i->first == destination) && (i->second > d.second)) {
                                        d = make_pair(it->first.getAirline(), i->second);
                                }
                        }
                }

                if (!d.first.empty())
                        cout << "\n\t" << dye::light_yellow(d.first) << " provides flights to " <<
dye::light_yellow(destination) << " the most.\n\n";
        }


        // Връща марка самолет, с най-много собствен брой дестинации
        // variant 1:  - returns the plane type with the most "self" destinations.
        void typeDest_self() {
                map<Plane, unsigned> m;
                pair<string, unsigned> d = make_pair("", 0);

                for (auto it = airline_data.begin(); it != airline_data.end(); it++) {
                        m = it->first.GetPlaneMD();

                        for (auto i = m.begin(); i != m.end(); i++) {
                                if (i->second > d.second)
                                        d = make_pair(i->first.getType(), i->second);
                        }
                }
                cout << "\n\tType: " << dye::light_yellow(d.first) << " Destinations: " <<
dye::light_yellow(d.second) << "\n\n";
        }

        // Връща коя марка самолет е използвана най-много в летището
        // variant 2:  - returns the plane type with the most destinations (in the airport) -> "B777 is
used the most"
        void typeDest_general() {
                map<Plane, string> m;
                map<string, unsigned> total;
                multimap<unsigned, string> rev;

                for (auto it = airline_data.begin(); it != airline_data.end(); it++) {
```

```
                        m = it->first.GetPlaneMap();

                        for (auto i = m.begin(); i != m.end(); i++) {
                                if (total.find(i->first.getType()) == total.end()) {          // key not
found
                                        total.insert(make_pair(i->first.getType(), 1));
                                }
                                else {
                                        // found
                                        unsigned n = total.find(i->first.getType())->second;
                                        total.at(i->first.getType()) = n + 1;
                                }
                        }
                }

                for (const auto& it : total)
                        rev.insert({ it.second, it.first });
                multimap<unsigned, string>::reverse_iterator it = rev.rbegin();
                auto range = rev.equal_range(it->first);                          // if there's more than 1
"max used" plane
                for (auto it = range.first; it != range.second; it++)
                        cout << dye::light_yellow(it->second) << " ";


                cout << "\n\n\tList of how many times an aircraft type has been used:";
                for (const auto& i : total) {
                        cout << "\n\t" << i.first << " -> " << dye::light_yellow(i.second);
                }
        }


        // При подаден аргумент авиокомпания, връща дестинацията с най-много полети
        // Returns destinaion with the most flights.
        string airlineDest(const string& airlineName) {
                multimap<string, unsigned> mm;
                Airline temp(airlineName, "");

                for (auto it = airline_data.begin(); it != airline_data.end(); it++) {
                        if (it->first.getAirline() == temp.getAirline()) {
                                mm = it->first.GetMM();

                                unsigned currentMax = 0;
                                string arg_Max;

                                for (auto i = mm.begin(); i != mm.end(); i++) {
                                        if (i->second > currentMax) {
                                                arg_Max = i->first;
                                                currentMax = i->second;
                                        }
                                }
                                return arg_Max;
                        }
                        else cout << "\n\tAll the destinations are unique\n";
                }
        }

        // Връща контейнер от самолети, съдържащ летателните часове >10000 на всички самолети от
авиокомпаниите, ползващи летище София.
        // Returns container of planes, >10000h and dest = SOF
        map<Plane, string> planesLAST() {
                map<Plane, string> m, result;

                for (auto it = airline_data.begin(); it != airline_data.end(); it++) {
                        m = it->first.GetPlaneMap();                                          // Plane,
string -> [A350, 1257h | SOF] ...
                        for (auto i = m.begin(); i != m.end(); i++) {
                                if ((i->first.getHours() > 10000) && (i->second == "SOF")) {
                                        result.insert(make_pair(i->first, i->second));
                                }
```

```
                        }
                }
                return result;
        }
};


int main() {
        int airportCh;
        int funcCh;
        bool exit = false;
        multimap<string, string> code_to_City;

        try {
                Airport ist("IST", "IST.txt");
                Airport var("VAR", "IST.txt");
                Airport sof("SOF", "IST.txt");
                Airport vie("VIE", "IST.txt");
                Airport cdg("CDG", "IST.txt");
                Airport temp;

                while (!exit) {
                        do {
                                airportCh = airportMenu();
                                switch (airportCh) {
                                case 1:
                                        temp = ist;
                                        break;
                                case 2:
                                        temp = var;
                                        break;
                                case 3:
                                        temp = sof;
                                        break;
                                case 4:
                                        temp = vie;
                                        break;
                                case 5:
                                        temp = cdg;
                                        break;
                                default:
                                        cout << "\n\t Please select an airport! (1 - 5)\n";
                                        system("pause");
                                }
                        } while (airportCh < 1 || airportCh > 5);

                        string s = temp.getAirport();

                        do {
                                string type, dest, air;
                                funcCh = funcMenu(s);
                                switch (funcCh) {
                                case 1:                                                                 //
Departures board (print)
                                        system("cls");
                                        cout << temp;
                                        system("pause");
                                        break;

                                case 2:                                                                 // The
average destinations of airlines based in France
                                        system("cls");
                                        cout << "\n\tThe average destinations of airlines based in France
is: " << dye::light_yellow(temp.franceBased()) << "\n\n";
                                        system("pause");
                                        break;

                                case 3:                                                                 // Planes
with <2000 flight hours and >10 destinations
```

```
                                        system("cls");
                                        cout << "\n\tThe planes with <2000 flight hours and <10
destinations are: \n\n";
                                        temp.less2000more10();
                                        cout << "\n\n";
                                        system("pause");
                                        break;

                                case 4:                                                 // Search
flights by aircraft type
                                        system("cls");
                                        cout << "\n\tSearch flight by aircraft type: "; cin >> type;
                                        transform(type.begin(), type.end(), type.begin(), toupper);
                                        temp.flightsAirline(type);
                                        cout << "\n\n";
                                        system("pause");
                                        break;

                                case 5:                                                 // The
airline that has provided the most flights to a given destination
                                        system("cls");
                                        cout << "\tThe airline that has provided the most flights to a
given destination:\n\n";
                                        cout << "\tDestination : "; cin >> dest;
                                        transform(dest.begin(), dest.end(), dest.begin(), toupper);   //
algorithm - destinations are with upper case

                                        if (city_to_Code.count(dest))                           //
count returns 1 if key exists
                                                dest = (city_to_Code.find(dest)->second);
                                        else
                                                cout << "\n\tNo city named " << dest << " exists!\n";
                                        temp.destAirline(dest);
                                        system("pause");
                                        break;

                                case 6:                                                 // The plane
whose been to the most destinations
                                        system("cls");
                                        cout << "\n\tThe plane whose been to the most destinations is:
\n";
                                        temp.typeDest_self();
                                        system("pause");
                                        break;

                                case 7:                                                 // Most
frequently used aircraft type
                                        system("cls");
                                        cout << "\n\tThe most frequently used aircraft type is: ";
                                        temp.typeDest_general();
                                        cout << "\n\n";
                                        system("pause");
                                        break;

                                case 8:                                                 // Most
visited destination of a given airline
                                        system("cls");
                                        cout << "\n\tMost visited destination of a given airline:\n\n";
                                        cout << "\tAirline : "; cin >> air;
                                        transform(air.begin(), air.end(), air.begin(), toupper);
                                        cout << "\n\tThe most visited destination is: " <<
dye::light_yellow(temp.airlineDest(air)) << "\n\n";
                                        system("pause");
                                        break;

                                case 9:                                                 // Planes
which destination is Sofia and has >10000 flight hours
                                        system("cls");
```

```
                                        cout << "\n\tPlanes whose destination is Sofia and has >10000
flight hours:\n";
                                        if (s == "SOF")
                                                cout << "\n\tThis is airport SOF, no need to search!";
                                        else {
                                                for (const auto& i : temp.planesLAST()) {
                                                        cout << "\n\t" << i.first.getType() << " " <<
i.first.getHours() << " " << i.second;
                                                }
                                                cout << "\n\n";
                                        }
                                        system("pause");
                                        break;
                                case 10:                                                        // Change
airport
                                        break;
                                case 11:                                                        // Exit
program
                                        exit = true;  // end while
                                        break;
                                default:
                                        cout << "\n\t Please select something between 1 and 11!\n";
                                }
                        } while (funcCh != 11 && !(funcCh < 1 || funcCh > 9));    // if 10 is pressed - it
will restart the while loop
                        // if 11 is pressed - it will end the while loop
                }
        }
        catch (int error) {
                switch (error) {
                case FILE_ERROR:
                        cout << "File error! " << endl;
                        break;
                case DATA_ERROR:
                        cout << "Corrupt data! " << endl;
                        break;
                }
        }
        return 0;
}


int airportMenu() {
        int ch;
        char prev = cout.fill(' ');
        system("cls");
        cout << "\n\t +=========================+";
        cout << "\n\t |" << setw(22) << dye::light_yellow("Select an airport") << setw(6) << "|";
        cout << "\n\t +=========================+";
        cout << "\n\t |\t" << dye::light_yellow("1.  ") << "IST" << "              |";
        cout << "\n\t |\t" << dye::light_yellow("2.  ") << "VAR" << "              |";
        cout << "\n\t |\t" << dye::light_yellow("3.  ") << "SOF" << "              |";
        cout << "\n\t |\t" << dye::light_yellow("4.  ") << "VIE" << "              |";
        cout << "\n\t |\t" << dye::light_yellow("5.  ") << "CDG" << "              |";
        cout << "\n\t +=========================+";
        cout << "\n\n\t" << dye::light_yellow(" Choice: ");
        cin >> ch;
        return ch;
}


int funcMenu(string airportName) {
        system("cls");
        int ch;
        int l = 41 + ((7 + airportName.size()) / 2);      // something like align:center
        int r = 83 - l;
        cout << "\n\t
+=============================================================================+";
        cout << "\n\t |" << setw(l) << dye::light_yellow("Airport " + airportName) << setw(r) << "|";
```

```
        cout << "\n\t
+=============================================================================+";
        cout << "\n\t |
|";
        cout << "\n\t |\t" << dye::light_yellow("1.  ") << "Departures board (print)" << "
|";
        cout << "\n\t |\t" << dye::light_yellow("2.  ") << "The average destinations of airlines based in
France" << "                    |";
        cout << "\n\t |\t" << dye::light_yellow("3.  ") << "Planes with <2000 flight hours and >10
destinations" << "                      |";
        cout << "\n\t |\t" << dye::light_yellow("4.  ") << "Search flights by aircraft type" << "
|";
        cout << "\n\t |\t" << dye::light_yellow("5.  ") << "The airline that has provided the most flights
to a given destination" << "   |";
        cout << "\n\t |\t" << dye::light_yellow("6.  ") << "The plane whose been to the most destinations"
<< "                          |";
        cout << "\n\t |\t" << dye::light_yellow("7.  ") << "Most frequently used aircraft type" << "
|";
        cout << "\n\t |\t" << dye::light_yellow("8.  ") << "Most visited destination of a given airline"
<< "                             |";
        cout << "\n\t |\t" << dye::light_yellow("9.  ") << "Planes whose destination is Sofia and has
>10000 flight hours" << "           |";
        cout << "\n\t |\t" << dye::light_yellow("10. ") << "Change airport" << "
|";
        cout << "\n\t |\t" << dye::light_yellow("11. ") << "Exit program" << "
|";
        cout << "\n\t |
|";
        cout << "\n\t
+=============================================================================+";
        cout << "\n\n\t" << dye::light_yellow(" Choice: ");
        cin >> ch;
        return ch;
}
```