```cpp
// Header
#include <iostream>
#include <conio.h>
#include <fstream>
#include <iomanip>
#include <string>

using namespace std;

#pragma once
class Information // Базовый абстрактный класс Information
{
public:
    virtual void Edit() = 0;
    virtual void PutData() = 0;
    virtual void Show() = 0;
};

#pragma once
#include "Header.h"
#include "Information.h"
class Client : public Information
{
private:
   string clientCode;
   string FIO;
   string passportId;
   string mail;
   string mobileNumber;
   string login;
   string password;

public:
 Client();
 Client(string _l, string _p);
 ~Client();
 Client(string _cCode, string _fio, string _passId, string _mail, string _mNumber,
string _login, string _pass);
 Client(const Client& obj);

 Client& operator=(const Client& obj);

 friend fstream& operator<<(fstream& f, Client& obj); // перегруженный оператор
вывода для записи данных в файл
 friend fstream& operator>>(fstream& f, Client& obj); // перегруженный оператор ввода
для чтение данных с файла
 // Гетторы
 string getClientCode();
 string getFio();
 string getPassportId();
 string getMail();
 string getMobileNumber();
 // Сетторы
 void setClientCode(string s);
 void setFio(string s);
 void setPasswordId(string s);
 void setMail(string s);
 void setMobile(string s);

 string getLogin() { return login; }
 string getPassword() { return password; }

 // перегруженные методы
 void Edit() override;
 void PutData() override;
 void Show() override;
};


#include "Client.h"

#include "Functions.h"
Client::Client() { }

Client::Client(string _l, string _p) // конструктор с параметрами
{
    password = _p;
    login = _l;
}
Client::~Client() { } // деструктор
Client::Client(string _cCode, string _fio, string _passId, string _mail, string
_mNumber, string _login, string _pass)
{
    clientCode = _cCode;
    passportId = _passId;
    FIO = _fio;
    mail = _mail;
    mobileNumber = _mNumber;
    login = _login;
    password = _pass;
}

Client::Client(const Client& obj) // конмтруктор копирования
{
   clientCode = obj.clientCode;
   passportId = obj.passportId;
   FIO = obj.FIO;
   mail = obj.mail;
   mobileNumber = obj.mobileNumber;
   password = obj.password;
   login = obj.login;
}

Client& Client::operator=(const Client& obj) // перегруженный оператор присваиване
{
   if (this != &obj)
   {
       clientCode = obj.clientCode;
       passportId = obj.passportId;
       FIO = obj.FIO;
       mail = obj.mail;
       mobileNumber = obj.mobileNumber;
       password = obj.password;
       login = obj.login;
   }
   return *this;
}
// Гетторы
string Client::getClientCode(){
  return clientCode;
}
```

```
string Client::getFio(){
  return FIO;
}

string Client::getPassportId(){
   return passportId;
}

string Client::getMail(){
   return mail;
}

string Client::getMobileNumber(){
   return mobileNumber;
}

// Сетторы
void Client::setClientCode(string s){
   clientCode = s;
}

void Client::setFio(string s){
    FIO = s;
}

void Client::setPasswordId(string s){
   passportId = s;
}

void Client::setMail(string s){
  mail = s;
}

void Client::setMobile(string s){
    mobileNumber = s;
}

// перегруженные методы
void Client::Edit() { }

void Client::PutData()
{
   cout << "__ Новый логин: ";
   cin >> login;
   cout << "__ Новый пароль: ";
   cin >> password;
   cout << "__ Придумайте свой ID: ";
   cin >> clientCode;
   cout << "__ Ф.И.О: ";
   FIO = enterCharOnly();
   cout << "__ Номер паспорта: ";
   cin >> passportId;
   cout << "__ Почта: ";
   cin >> mail;
   cout << "__ Мобильный номер: ";
   cin >> mobileNumber;
}

void Client::Show()
{
   cout << setw(15) << login
```

```
    << setw(15) << password
    << setw(8) << clientCode
    << setw(20) << FIO
    << setw(15) << passportId
    << setw(30) << mail
    << setw(15) << mobileNumber;
}

fstream& operator<<(fstream& f, Client& obj)
{
   f << obj.login << " " << obj.password << " "
    << obj.clientCode << " " << obj.FIO << "*"
    << obj.passportId << " "
    << obj.mail << " " << obj.mobileNumber << endl;
   return f;
}


fstream& operator>>(fstream& f, Client& obj)
{
   f >> obj.login >> obj.password;
   f >> obj.clientCode;
   getline(f, obj.FIO, '*');
   f >> obj.passportId >> obj.mail >> obj.mobileNumber;
   return f;
}

#pragma once
#include "Header.h"
#include "Information.h"

class Ticket : public Information
{
 private:
  string ticketCode; //ID билета
  string userCode; // ID клиента
  string transportType; //вид транспорта
  string departurePoint;
  string arrivalPoint;
  string deprtureData;
  string arrivalData;
 public:
   Ticket();
   ~Ticket(); //деструктор
   Ticket(string _tCode, string _uCode, string _tType, string _dPoint, string
_aPoint, string _dData);
   Ticket(const Ticket& obj); //конструктор копирование

   Ticket& operator=(const Ticket& obj); //перегруженный оператор присваивания

   friend fstream& operator<<(fstream& f, Ticket& obj); //перегруженный оператор
вывода для записи данных в файл
   friend fstream& operator>>(fstream& f, Ticket& obj); //перегруженный оператор
ввода для чтение данных с файла

   // гетторы
   string getTicketCode();
   string getUserCode();
   string getTransportType();
   string getDeparturePoint();
   string getArrivalPoint();
```

```
    string getDepartureData();
   // сетторы
    void setTicketCode(string _s);
    void setUserCode(string _s);
    void setTransportType(string _s);
    void setDeparturePoint(string _s);
    void setArrivalPoint(string _s);
    void setDepartureData(string _s);

    void Edit() override;
    void PutData() override;
    void Show() override;

    static void header();
    static void headerLine();
};

#include "Ticket.h"
#include "Functions.h"

Ticket::Ticket() { }

Ticket::~Ticket() { } // деструктор
// конструктор с параметрами
Ticket::Ticket(string _tCode, string _uCode, string _tType, string _dPoint, string
_aPoint, string _dData)
{
    ticketCode = _tCode;
    userCode = _uCode;
    transportType = _tType;
    departurePoint = _dPoint;
    arrivalPoint = _aPoint;
    deprtureData = _dData;
}

// конструктор копирования
Ticket::Ticket(const Ticket& obj)
{
    ticketCode = obj.ticketCode;
    userCode = obj.userCode;
    transportType = obj.transportType;
    departurePoint = obj.departurePoint;
    arrivalPoint = obj.arrivalPoint;
    deprtureData = obj.deprtureData;
    arrivalData = obj.arrivalData;
}

// сетторы
void Ticket::setTicketCode(string _s){
 ticketCode = _s;
}

void Ticket::setUserCode(string _s){
 userCode = _s;
}

void Ticket::setTransportType(string _s){
 transportType = _s;
}

void Ticket::setDeparturePoint(string _s){
 departurePoint = _s;
}

void Ticket::setArrivalPoint(string _s){
 arrivalPoint = _s;
}

void Ticket::setDepartureData(string _s){
 deprtureData = _s;
}

void Ticket::Edit() { } // перегруженный метод

Ticket& Ticket::operator=(const Ticket& obj){
 if (this != &obj)
 {
    ticketCode = obj.ticketCode;
    userCode = obj.userCode;
    transportType = obj.transportType;
    departurePoint = obj.departurePoint;
    arrivalPoint = obj.arrivalPoint;
    deprtureData = obj.deprtureData;
 }
 return *this;
}

void Ticket::PutData()// перегруженный метод
{
   cout << " ___ Введите данные" << endl << endl;

   cout << "1___ Придумайте код для билета: ";
   ticketCode = enterCharOnly();
   cout << "2___ Вид транспорта: ";
   transportType = enterCharOnly();
   cout << "3___ Место отправление: ";
   departurePoint = enterCharOnly();
   cout << "4___ Место прибытие: ";
   arrivalPoint = enterCharOnly();
   cout << "5___ Дата отправление: ";
   cin >> deprtureData;
}

void Ticket::Show()
{
      cout << setw(11) << ticketCode << setw(16) << transportType << setw(16) <<
departurePoint << setw(16) << arrivalPoint << setw(16) << deprtureData;
}

void Ticket::header()
{

 cout << "--- ---------- --------------- --------------- --------------- ------------
---" << endl;
    cout << " №  ID билета     Вид трансп.     Место отпр.     Место приб.      Дата
отпр. " << endl;
    cout << "--- ---------- --------------- --------------- --------------- --------
-------" << endl;
}
```

```
void Ticket::headerLine()
{
   cout << "--- ---------- --------------- --------------- --------------- ---------
------" << endl;
}

// гетторы
string Ticket::getTicketCode(){
 return ticketCode;
}

string Ticket::getUserCode(){
 return userCode;
}

string Ticket::getTransportType(){
 return transportType;
}

string Ticket::getDeparturePoint(){
 return departurePoint;
}

string Ticket::getArrivalPoint(){
 return arrivalPoint;
}

string Ticket::getDepartureData(){
 return deprtureData;
}

// перегруженный оператор вывода для записи данных в файл
fstream& operator<<(fstream& f, Ticket& obj)
{
   f << obj.arrivalPoint << " "
    << obj.departurePoint << " "
    << obj.deprtureData << " "
    << obj.ticketCode << " "
    << obj.userCode << " "
    << obj.transportType << endl;
   return f;
}

// перегруженный оператор ввода для чтение данных с файла
fstream& operator>>(fstream& f, Ticket& obj){
   f >> obj.arrivalPoint
    >> obj.departurePoint
    >> obj.deprtureData
    >> obj.ticketCode
    >> obj.userCode
     >> obj.transportType;
   return f;
}

#pragma once
#include "Header.h"
#include "Information.h"
class Order : public Information
{
private:
 string clientCode;
 string tourCode;

public:
 Order();
 ~Order();
 Order(const Order& obj);

 Order& operator=(const Order& obj);

 friend fstream& operator<<(fstream& f, Order& obj);
 friend fstream& operator>>(fstream& f, Order& obj);

 string getClientCode();
 void setClientCode(string clientCode);

 string getTourCode();
 void setTourCode(string _t);

 void Edit() override;
 void PutData() override;
 void Show() override;
 static void header();
 static void headerLine();
};

#include "Order.h"
#include "Functions.h"

Order::Order() { }
Order::~Order() { }
Order::Order(const Order& obj)
{
 clientCode = obj.clientCode;
 tourCode = obj.tourCode;
}

Order& Order::operator=(const Order& obj)
{
   if (this != &obj)
   {
       clientCode = obj.clientCode;
       tourCode = obj.tourCode;
   }
   return *this;
}

string Order::getClientCode(){
   return clientCode;
}

void Order::setClientCode(string clientCode){
 this->clientCode = clientCode;
}

string Order::getTourCode(){
 return tourCode;
}

void Order::setTourCode(string _t){
 tourCode = _t;
}
```

```
void Order::Edit()
{
   cout << " ___ Выберите какое поле изменить" << endl << endl;

   cout << "1___ Номер клмента\n";
   cout << "2___ Код тура\n";
   cout << "0___ Назад\n";

   int choice;

   choice = enterInt(0, 2);

   switch (choice)
   {
     case 1:
     {
        cout << "1___ Номер клмента: ";
        clientCode = enterCharOnly();
        break;
     }
     case 2:
     {
         cout << "2___ Код тура: ";
         tourCode = enterCharOnly();
         break;
       }
    }
}

void Order::PutData()
{
   cout << " ___ Введите данные" << endl << endl;

   cout << "1___ Введите ваш клиентский код: ";
   clientCode = enterCharOnly();
   cout << "2___ Код тура: ";
   tourCode = enterCharOnly();
}


void Order::Show(){
 cout << setw(16) << clientCode << " " << setw(16) << tourCode;
}

void Order::header()
{
  cout << "--- --------------- ---------------" << endl;
  cout << " №      ID клиента         ID тура " << endl;
  cout << "--- --------------- ---------------" << endl;
}

void Order::headerLine()
{
  cout << "--- --------------- ---------------" << endl;
}

fstream& operator<<(fstream& f, Order& obj){
   f << obj.clientCode << " " << obj.tourCode << endl;
   return f;
}
```

```
fstream& operator>>(fstream& f, Order& obj){
   f >> obj.clientCode >> obj.tourCode;
   return f;
}

#pragma once
#include "Header.h"
#include "Information.h"
class Tour : public Information{
 protected:
  string tourName;
  string tourCode;
  string tourType;
  string tourDate;
  int duration;
  float price;
 public:
  Tour();
  ~Tour();
  Tour(const Tour& obj);
  string getTourCode();
  string getTourName();
  string getTourType();
};

#include "Tour.h"

Tour::Tour(){ }
Tour::~Tour(){ }
Tour::Tour(const Tour& obj)
{
  tourName = obj.tourName;
  tourCode = obj.tourCode;
  tourType = obj.tourType;
  tourDate = obj.tourDate;
  duration = obj.duration;
  price = obj.price;
}

string Tour::getTourCode(){
 return tourCode;
}

string Tour::getTourName(){
 return tourName;
}

string Tour::getTourType(){
 return tourType;
}

#pragma once
#include "Tour.h"

class InternationalTour :public Tour
{
protected:
 string country;
 string city;
public:
 InternationalTour() {};
 ~InternationalTour() {};
 string getCountry();
 string getCity();
 void setCountry(string s);
 void setCity(string s);

};

#include "InternationalTour.h"

string InternationalTour::getCountry(){
 return country;
}

string InternationalTour::getCity(){
 return city;
}

void InternationalTour::setCountry(string s){
 country = s;
}

void InternationalTour::setCity(string s){
 city = s;
}


#pragma once
#include "Tour.h"
class LocalTour :public Tour
{
private:
 string city;
public:
 LocalTour();
 ~LocalTour();
 LocalTour(const LocalTour& obj);

 LocalTour& operator=(const LocalTour& obj);

 friend fstream& operator>>(fstream& f, LocalTour& obj);
 friend fstream& operator<<(fstream& f, LocalTour& obj);

 void Edit() override;
 void PutData() override;
 void Show() override;

 static void header();
 static void headerLine();
};

#include "LocalTour.h"
#include "Functions.h"

LocalTour::LocalTour() { }
LocalTour::~LocalTour() { }
LocalTour::LocalTour(const LocalTour& obj)
{
   tourCode = obj.tourCode;
   tourType = obj.tourType;
   tourDate = obj.tourDate;
```

```
    duration = obj.duration;
    price = obj.price;
    city = obj.city;
    tourName = obj.tourName;
}

LocalTour& LocalTour::operator=(const LocalTour& obj)
{
  if (this != &obj) {
    tourCode = obj.tourCode;
    tourType = obj.tourType;
    tourDate = obj.tourDate;
    duration = obj.duration;
    price = obj.price;
    city = obj.city;
    tourName = obj.tourName;
  }
  return *this;
}

void LocalTour::Edit()
{
    cout << " ___ Выберите какое поле изменить" << endl << endl;

    cout << "1___ Назвния тура\n";
    cout << "2___ Код тура\n";
    cout << "3___ Вид тура\n";
    cout << "4___ Дата\n";
    cout << "5___ Длительность\n";
    cout << "6___ Цена\n";
    cout << "7___ Город\n";
    cout << "0___ Назад\n";

    int choice;
    choice = enterInt(0, 7);
 switch (choice) {
    case 1: {
     cout << "1___ Назвния тура: ";
     tourName = enterCharOnly();
     break;
    }
    case 2: {
     cout << "2___ Придумайте код для тура: ";
     tourCode = enterCharOnly();
     break;
    }
    case 3: {
     cout << "3___ Вид тура: ";
     tourType = enterCharOnly();
     break;
    }
    case 4: {
     cout << "4___ Дата: ";
     cin >> tourDate ;
     break;
    }
    case 5: {
     cout << "5___ Длительность: ";
     duration = enterInt(1,30);
     break;
    }
    case 6: {
     cout << "6___ Цена: ";
     price = enterInt(0, 999999);
     break;
    }
    case 7: {
     cout << "7___ Город: ";
     city = enterCharOnly();
     break;
    }
  }
}

void LocalTour::PutData()
{
   cout << " ___ Введите данные" << endl << endl;

   cout << "1___ Назвния тура: ";
   tourName = enterCharOnly();
   cout << "2___ Придумайте код для тура: ";
   tourCode = enterCharOnly();
   cout << "3___ Вид тура: ";
   tourType = enterCharOnly();
   cout << "4___ Дата: ";
   cin >> tourDate;
   cout << "5___ Длительность: ";
   cin >> duration;
   cout << "6___ Цена: ";
   price = enterInt(0, 999999);
   cout << "7___ Город: ";
   city = enterCharOnly();
}

void LocalTour::Show()
{
   cout << setw(21) << tourName
    << setw(16) << tourCode
    << setw(16) << tourType
    << setw(11) << tourDate
    << setw(11) << duration
    << setw(11) << price
    << setw(16) << city;
}

void LocalTour::header()
{
    cout << "--- -------------------- --------------- --------------- ---------- ---
------- ---------- ---------------" << endl;
    cout << " №        Названия тура         ID тура        Вид тура  Дата тура
Длит.       Цена           Город " << endl;
    cout << "--- -------------------- --------------- --------------- ---------- ---
------- ---------- ---------------" << endl;
}

void LocalTour::headerLine()
{
 cout << "--- -------------------- --------------- --------------- ---------- -------
--- ---------- ---------------" << endl;
}

fstream& operator>>(fstream& f, LocalTour& obj){
  getline(f, obj.tourName, '*');
  f >> obj.tourCode
 >> obj.tourType
 >> obj.tourDate
 >> obj.duration
 >> obj.price;
  getline(f, obj.city, '\n');
  return f;
}
fstream& operator<<(fstream& f, LocalTour& obj)
{
   f << obj.tourName << "*"
    << obj.tourCode << " "
    << obj.tourType << " "
    << obj.tourDate << " "
    << obj.duration<< " "
    << obj.price << " "
  << obj.city << endl;
   return f;
}

#pragma once
#include "InternationalTour.h"

class LandInternational:public InternationalTour
{
  private:
   string carType;
  public:
   LandInternational();
   ~LandInternational();
   LandInternational(const LandInternational& obj);
   LandInternational& operator=(const LandInternational& obj);
   friend fstream& operator<< (fstream& f, LandInternational& obj);
   friend fstream& operator>> (fstream& f, LandInternational& obj);
   string getCarType();
   void setCarTpye(string _t);
   void Edit() override;
   void PutData() override;
   void Show() override;
   static void header();
   static void headerLine();
};

#include "LandInternational.h"
#include "Functions.h"

LandInternational::LandInternational() { }
LandInternational::~LandInternational() { }
LandInternational::LandInternational(const LandInternational& obj){
   tourCode = obj.tourCode;
   tourType = obj.tourType;
   tourDate = obj.tourDate;
   duration = obj.duration;
   tourName = obj.tourName;
   price = obj.price;
   country = obj.country;
   city = obj.city;
   carType = obj.carType;
}
```

```
LandInternational& LandInternational::operator=(const LandInternational& obj)
{
   if (this != &obj)
   {
       tourCode = obj.tourCode;
       tourType = obj.tourType;
       tourDate = obj.tourDate;
       duration = obj.duration;
       price = obj.price;
       country = obj.country;
       city = obj.city;
       tourName = obj.tourName;
       carType = obj.carType;
   }
   return *this;
}

string LandInternational::getCarType(){
 return carType;
}

void LandInternational::setCarTpye(string _t){
 carType = _t;
}

void LandInternational::Edit()
{
   cout << " ___ Выберите какое поле изменить" << endl << endl;

   cout << "1___ Назвния тура\n";
   cout << "2___ Код тура\n";
   cout << "3___ Вид тура\n";
   cout << "4___ Дата\n";
   cout << "5___ Длительность\n";
   cout << "6___ Цена\n";
   cout << "7___ Страна\n";
   cout << "8___ Город\n";
   cout << "9___ Вид транспорта\n";
   cout << "0___ Назад\n";

   int choice;
   choice = enterInt(0, 9);
   switch (choice)
   {
       case 1:
       {
        cout << "1___ Назвния тура: ";
        tourName = enterCharOnly();
        break;
       }
       case 2: {
        cout << "2___ Придумайте код для тура: ";
        tourCode = enterCharOnly();
        break;
       }
       case 3:
       {
        cout << "3___ Вид тура: ";
        tourType  = enterCharOnly();
        break;
       }
```

```
        case 4:
        {
         cout << "4___ Дата: ";
         cin >> tourDate;
         break;
        }
        case 5:
        {
         cout << "5___ Длительность: ";
         duration = enterInt(1,30);
         break;
        }
        case 6:
        {
         cout << "6___ Цена: ";
         price = enterInt(0, 999999);
         break;
        }
        case 7:
        {
         cout << "7___ Страна: ";
         country = enterCharOnly();
         break;
        }
        case 8:
        {
         cout << "8___ Город: ";
         city = enterCharOnly();
         break;
        }
        case 9:
        {
         cout << "9___ Вид транспорта: ";
         carType = enterCharOnly();
         break;
        }
    }
}


void LandInternational::PutData()
{
   cout << " ___ Введите данные" << endl << endl;

   cout << "1___ Назвния тура: ";
   tourName = enterCharOnly();
   cout << "2___ Придумайте код для тура: ";
   tourCode = enterCharOnly();
   cout << "3___ Вид тура: ";
   tourType = enterCharOnly();
   cout << "4___ Дата: ";
   cin >> tourDate;
   cout << "5___ Длительность: ";
   duration = enterInt(1,30);
   cout << "6___ Цена: ";
   price = enterInt(0, 999999);
   cout << "7___ Страна: ";
   country = enterCharOnly();
   cout << "8___ Город: ";
   city = enterCharOnly();
   cout << "9___ Вид транспорта: ";
   carType = enterCharOnly();
}

void LandInternational::Show()
{
   cout << setw(20) << tourName
    << setw(16) << tourCode
    << setw(16) << tourType
    << setw(11) << tourDate
    << setw(11) << duration
    << setw(11) << price
    << setw(16) << country
    << setw(16) << city
    << setw(16) << carType;
}

void LandInternational::header()
{
   cout << "--- -------------------- --------------- --------------- ---------- ----
------ ---------- --------------- --------------- ---------------" << endl;
   cout << " №        Названия тура         ID тура        Вид тура  Дата тура
Длит.       Цена          Страна           Город       Транспорт " << endl;
   cout << "--- -------------------- --------------- --------------- ---------- ----
------ ---------- --------------- --------------- ---------------" << endl;
}


void LandInternational::headerLine()
{
   cout << "--- -------------------- --------------- --------------- ---------- ----
------ ---------- --------------- --------------- ---------------" << endl;
}

fstream& operator<<(fstream& f, LandInternational& obj)
{
   f << obj.tourName << "*" << obj.tourCode << " "
    << obj.tourType << " " << obj.tourDate << " "
    << obj.duration << " " << obj.price << " "
    << obj.country << " " << obj.city << " " << obj.carType << endl;
   return f;
}

fstream& operator>>(fstream& f, LandInternational& obj)
{
   getline(f, obj.tourName, '*');
   f >> obj.tourCode >> obj.tourType >> obj.tourDate
    >> obj.duration >> obj.price >> obj.country >> obj.city;
   getline(f, obj.carType, '\n');
   return f;
}

#pragma once
#include "InternationalTour.h"
class SeaInternational :public InternationalTour
{
  private:
   string cargoName;
   string seaName;
  public:
   SeaInternational();
   ~SeaInternational();
```

```cpp
        SeaInternational(const SeaInternational& obj);
        SeaInternational& operator=(const SeaInternational& obj);

        friend fstream& operator<< (fstream& f, SeaInternational& obj);
        friend fstream& operator>> (fstream& f, SeaInternational& obj);

        void setCargoName(string _c);
        void setSeaName(string _s);
        string getCargoName();
        string getSeaName();

        void Edit() override;
        void PutData() override;
        void Show() override;

        static void header();
        static void headerLine();
    };

    #include "SeaInternational.h"
    #include "Functions.h"
    SeaInternational::SeaInternational() { }
    SeaInternational::~SeaInternational() { }
    SeaInternational::SeaInternational(const SeaInternational& obj)
    {
        tourCode = obj.tourCode;
        tourType = obj.tourType;
        tourDate = obj.tourDate;
        duration = obj.duration;
        price = obj.price;
        country = obj.country;
        city = obj.city;
        cargoName = obj.cargoName;
        seaName = obj.seaName;
        tourName = obj.tourName;
    }

SeaInternational& SeaInternational::operator=(const SeaInternational& obj)
    {
      if (this != &obj) {
         tourCode = obj.tourCode;
         tourType = obj.tourType;
         tourDate = obj.tourDate;
         duration = obj.duration;
         price = obj.price;
         country = obj.country;
         city = obj.city;
         cargoName = obj.cargoName;
         seaName = obj.seaName;
         tourName = obj.tourName;
      }
      return *this;
    }


    void SeaInternational::setCargoName(string _c){
     cargoName = _c;
    }

    void SeaInternational::setSeaName(string _s){
```

```
 seaName = _s;
}

string SeaInternational::getCargoName(){
 return cargoName;
}
string SeaInternational::getSeaName(){
 return seaName;
}
void SeaInternational::Edit()
{
  cout << " ___ Выберите какое поле изменить" << endl << endl;

  cout << "1___ Назвния тура\n";
  cout << "2___ Код тура\n";
  cout << "3___ Вид тура\n";
  cout << "4___ Дата\n";
  cout << "5___ Длительность\n";
  cout << "6___ Цена\n";
  cout << "7___ Страна\n";
  cout << "8___ Город\n";
  cout << "9___ Вид транспорта\n";
  cout << "10__ Названия море\n";
  cout << "0___ Назад\n";

  int choice;

  choice = enterInt(0, 10);

  switch (choice) {
     case 1: {
      cout << "1___ Назвния тура: ";
      tourName = enterCharOnly();
      break;
     }
     case 2: {
      cout << "2___ Придумайте код для тура: ";
      tourCode = enterCharOnly();
      break;
     }
     case 3: {
      cout << "3___ Вид тура: ";
      tourType = enterCharOnly();
      break;
     }
     case 4: {
      cout << "4___ Дата: ";
      cin >> tourDate;
      break;
     }
     case 5: {
      cout << "5___ Длительность: ";
      duration = enterInt(1,30);
      break;
     }


     case 6: {
      cout << "6___ Цена: ";
      price = enterInt(0, 999999);
      break;
```

```
        }
        case 7: {
         cout << "7___ Страна: ";
         country = enterCharOnly();
         break;
        }
        case 8: {
         cout << "8___ Город: ";
         city = enterCharOnly();
         break;
        }
        case 9: {
         cout << "9___ Названия корабля: ";
         cargoName = enterCharOnly();
         break;
        }
        case 10: {
         cout << "10__  Названия море: ";
         seaName = enterCharOnly();
         break;
        }
    }
}

void SeaInternational::PutData()
{
    cout << " ___ Введите данные" << endl << endl;

    cout << "1___ Назвния тура: ";
    cin >> tourName;
    cout << "2___ Придумайте код для тура: ";
    cin >> tourCode;
    cout << "3___ Вид тура: ";
    cin >> tourType;
    cout << "4___ Дата: ";
    cin >> tourDate;
    cout << "5___ Длительность: ";
    cin >> duration;
    cout << "6___ Цена: ";
    price = enterInt(0, 999999);
    cout << "7___ Страна: ";
    cin >> country;
    cout << "8___ Город: ";
    cin >> city;
    cout << "9___ Названия корабля: ";
    cin >> cargoName;
    cout << "10__ Названия море: ";
    cin >> seaName;
}

void SeaInternational::Show()
{
    cout << setw(21) << tourName
     << setw(16) << tourCode
     << setw(16) << tourType
     << setw(11) << tourDate
     << setw(11) << duration
     << setw(11) << price
     << setw(16) << country
     << setw(16) << city
     << setw(16) << cargoName
    << setw(16) << seaName;
}

void SeaInternational::header()
{
   cout << "--- -------------------- --------------- --------------- ---------- ----
------ ---------- --------------- --------------- --------------- ---------------" <<
endl;
   cout << " №        Названия тура         ID тура        Вид тура  Дата тура
Длит.       Цена          Страна           Город         Корабль            Море " <<
endl;
   cout << "--- -------------------- --------------- --------------- ---------- ----
------ ---------- --------------- --------------- --------------- ---------------" <<
endl;
}

void SeaInternational::headerLine()
{

   cout << "--- -------------------- --------------- --------------- ---------- ----
------ ---------- --------------- --------------- --------------- ---------------" <<
endl;
}

fstream& operator<<(fstream& f, SeaInternational& obj)
{
   f << obj.tourName <<"*" << obj.tourCode << " "
    << obj.tourType << " " << obj.tourDate << " "
    << obj.duration << " " << obj.price << " "
    << obj.country << " " << obj.city << " "
  << obj.cargoName << " " << obj.seaName << endl;
   return f;
}

fstream& operator>>(fstream& f, SeaInternational& obj){
   getline(f, obj.tourName, '*');
   f >> obj.tourCode >> obj.tourType >> obj.tourDate
    >> obj.duration >> obj.price >> obj.country
  >> obj.city >> obj.cargoName;
   getline(f, obj.seaName, '\n');
   return f;
}

#pragma once
#include "Header.h"

template <class T>
class File
{
 private:
   fstream filestream;
   char filename[30];
  public:
   File();
   File(char* filename);
   ~File() { filestream.close(); }

   void WriteData(T& obj);
   void ReadData(T& obj);

   bool REndFile();
   void OpenEnd(char* filename);
   void closeFile();
};

template<class T>
inline void File<T>::WriteData(T& obj){
  filestream << obj;
}

template<class T>
inline void File<T>::ReadData(T& obj){
 filestream >> obj;
}

template<class T>
inline bool File<T>::REndFile()
{
  if (filestream.eof())
   return true;
  else
   return false;
}

template<class T>
inline void File<T>::OpenEnd(char* filename){
  strcpy_s(this->filename, filename);
  filestream.open(this->filename, ios::app);
  if (!filestream.is_open())  {
     cout << "___ Ошибка открытие файла" << endl;
     return;
  }
}

template<class T>
inline void File<T>::closeFile(){
 filestream.close();
}

template<class T>
inline File<T>::File(){ }

template<class T>
inline File<T>::File(char* f)
{
  strcpy_s(this->filename, f);
  filestream.open(f, ios::in | ios::out);

  if (!filestream.is_open())
  {
    cout << "___ Ошибка открытие файла" << endl;
    return;
  }
}

#pragma once
#include "Queue.h"
#include "File.h"
#include "Order.h"
#include "Ticket.h"
#include "Functions.h"
#include "Client.h"
```

```
template <class T>
class Interface
{
   Queue<T>* queue;
   Queue<Order>* orders;
   Queue<Ticket>* tickets;
   char filename[30];
  public:
   Interface(char* f);
   ~Interface();
   void add();
   void del();
   void edit();
   void show();
   void start();
   void setFilename(char* f);
};

template<class T>
inline Interface<T>::Interface(char* f)
{
  queue = new Queue<T>;
  orders = new Queue<Order>;
  tickets = new Queue<Ticket>;
  setFilename(f);
}

template<class T>
inline Interface<T>::~Interface()
{
 delete queue;
 delete orders;
 delete tickets;
}

template<class T>
inline void Interface<T>::add()
{
  T object;
  object.PutData();
  queue->enqueue(object);
  cout << "___ Успешно!" << endl;
}

template<class T>
inline void Interface<T>::del(){
 if (queue->is_Empty())
 {
    cout << "___ Пусто" << endl;
    return;
 }

  this->show();
  cout << endl;
  cout << "___ Номер удаляемого элемента: ";
  int k;
  k = enterInt(1, queue->getSize());
  queue->Delete(k);
  cout << "___ Успешно";
}
```

```
template<class T>
inline void Interface<T>::edit()
{
  if (queue->is_Empty()) {
   cout << "___ Пусто" << endl;
   return;
  }

  show();
  cout << "___ Номер редактируемого элемента: ";
  int k;
  k = enterInt(1, queue->getSize());
  (*queue)[k - 1].Edit();
  cout << "___ Успешно" << endl;
 }

 template<class T>
 inline void Interface<T>::show()
 {
   if (queue->is_Empty())
   {
     cout << "___ Пусто" << endl;
     return;
    }
   T::header();
   queue->show();
   T::headerLine();
   cout << endl;
}

template<class T>
inline void Interface<T>::start()
{
  int choice;
  char ti[30] = "tickets.txt";
  char r[30] = "orders.txt";

  ReadDataFromFile(queue, filename);
  ReadDataFromFile(orders, r);
  ReadDataFromFile(tickets, ti);

 do {

    system("cls");
    cout << "___ Меню редактирование" << endl;
    cout << "1__ Добавить" << endl;
    cout << "2__ Удалить" << endl;
    cout << "3__ Изменить" << endl;
    cout << "4__ Сортировать" << endl;
    cout << "5__ Заказы на тур" << endl;
    cout << "6__ Билеты" << endl;
    cout << "7__ Сохранить данные" << endl;
    cout << "8__ Читат с файла" << endl;
    cout << "9__ ПРОСМОТР" << endl;
    cout << "0__ Назад" << endl;
    choice = enterInt(0, 9);

    cout << endl << endl;
    switch (choice)
    {
     case 1:
```

```
{
  add();
  break;
}
case 2:
{
  del();
  break;
}
case 3:
{
  edit();
  break;
}
case 4:
{
  cout << "1__ Сортировать по названию" << endl;
  cout << "2__ Сортировать по виду туров" << endl;
  int o;
  o = enterInt(1, 2);
  if (o == 1)    {
     for (int i = 0; i < queue->getSize(); i++)
     {
         for (int j = i+1; j < queue->getSize(); j++)
         if ((*queue)[i].getTourName() > (*queue)[j].getTourName())
            {
              T temp = (*queue)[i];
              (*queue)[i] = (*queue)[j];
              (*queue)[j] = temp;
           }
        }
     }
     else
  {
        for (int i = 0; i < queue->getSize(); i++)
        {
           for (int j = i + 1; j < queue->getSize(); j++)
          if ((*queue)[i].getTourType() > (*queue)[j].getTourType())
             {
               T temp = (*queue)[i];
               (*queue)[i] = (*queue)[j];
               (*queue)[j] = temp;
             }
        }
    }
    break;
 }

 case 5:
 {
     Queue<Order> temp;
     for (int i = 0; i < orders->getSize(); i++)
     {
       for (int j = 0; j < queue->getSize(); j++)
        {
      if ((*orders)[i].getTourCode() == (*queue)[j].getTourCode()){
             temp.enqueue((*orders)[i]);
           }
        }
     }
      Order::header();
      temp.show();
      Order::headerLine();

      cout << "1__ Подтвердить заказ и создать билет" << endl;
      cout << "0__ Назад" << endl;

      int y;
      y = enterInt(0,1);
      if (y == 1)
      {
        cout << "___ Введите номер заказа: ";
        int ll;
        ll = enterInt(1, temp.getSize());

        Ticket t;
        t.setUserCode(temp[ll].getClientCode());
        t.PutData();
        tickets->enqueue(t);
        cout << "___ Билет успешно создан" << endl;
     }
     break;
}
case 6:
{
  cout << "___ Список билетов" << endl << endl;
  if (tickets->getSize() == 0)
  {
     cout << "___ Пусто" << endl;
     break;
  }
  Ticket::header();
  tickets->show();
  Ticket::headerLine();
  cout << endl;
  break;
}
case 7:
{
  char ti[30] = "tickets.txt";
  char r[30] = "orders.txt";
  WriteDataToFile(queue, filename);
  WriteDataToFile(orders, r);
  WriteDataToFile(tickets, ti);
  break;
}
case 8:
{
  ReadDataFromFile(queue, filename);
  char ti[30] = "tickets.txt";
  char r[30] = "orders.txt";
  ReadDataFromFile(orders, r);
  ReadDataFromFile(tickets, ti);
  break;
}
case 9:
{
  this->show();
  break;
}
default:
 break;
```

```
  }
     cout << endl;
     system("pause");
   } while (choice != 0);
}

template<class T>
inline void Interface<T>::setFilename(char* f){
 strcpy_s(filename, f);
}

#pragma once
#include "Queue.h"
#include "Order.h"
#include "Client.h"
#include "Ticket.h"
#include "File.h"
#include "Functions.h"
#include "LandInternational.h"
#include "SeaInternational.h"
#include "LocalTour.h"

class InterfaceClient
{
   private:
    Client client;
    Queue<Order>* orders;
    Queue<Ticket>* tickets;
   public:
    InterfaceClient(Client obj);
    ~InterfaceClient();
    void start();
    template<class T>
    void makeOrder(Queue<T> *q);
    void loadTicket();
    void loadOrder();
    void writeOrder();
};


inline InterfaceClient::InterfaceClient(Client obj)
{
  orders = new Queue<Order>;
  tickets = new Queue<Ticket>;
  client = obj;
}

inline InterfaceClient::~InterfaceClient()
{
  delete orders;
  delete tickets;
}

inline void InterfaceClient::start()
{
   int vybor;
   do {
    Queue<LandInternational>* lands = new Queue<LandInternational>;
    char filename1[30] = "landtours.txt";
    ReadDataFromFile(lands, filename1);
```

```
char filename2[30] = "seatours.txt";
Queue<SeaInternational>* sea = new Queue<SeaInternational>;
ReadDataFromFile(sea, filename2);

char filename3[30] = "localtours.txt";
Queue<LocalTour>* localtours = new Queue<LocalTour>;
ReadDataFromFile(localtours, filename3);

char filename4[30] = "orders.txt";
ReadDataFromFile(orders, filename4);

char filename5[30] = "tickets.txt"; //билет
ReadDataFromFile(tickets, filename5);

system("cls");
cout << "___ Меню клиента" << endl << endl;
cout << "___ Список" << endl;
cout << "1__ Сухопутные туры" << endl;
cout << "2__ Морские туры" << endl;
cout << "3__ Местные туров" << endl << endl;

cout << "___ Раздел заказы" << endl;

cout << "4__ Бронировать" << endl;
cout << "5__ Мои заказы" << endl;
cout << "6__ Мои билеты" << endl;
cout << "0__ Выход" << endl;

vybor = enterInt(0,6);

switch (vybor)
{
    case 1:
    {
       LandInternational::header();
       lands->show();
       LandInternational::headerLine();
       break;
    }
    case 2:
    {
       SeaInternational::header();
       sea->show();
       SeaInternational::headerLine();
       break;
    }
    case 3:
    {
        LocalTour::header();
        localtours->show();
        LocalTour::headerLine();
        break;
    }
    case 4:
    {
      cout << "1__ Сухопутные туры" << endl;
      cout << "2__ Морские туры" << endl;
      cout << "3__ Местные туров" << endl << endl;
      cout << "___ Выберите тип тура: ";

      int chj;
```

```
  chj = enterInt(1,3);
  switch (chj)
  {
     case 1:     {
         LandInternational::header();
         lands->show();
         LandInternational::headerLine();
         makeOrder(lands);
         break;
     }
     case 2:     {
        SeaInternational::header();
        sea->show();
        SeaInternational::headerLine();
        makeOrder(sea);
        break;
     }
     case 3:
     {
        LocalTour::header();
        localtours->show();
        LocalTour::headerLine();
        makeOrder(localtours);
        break;
     }
     default: break;
  }
  break;
}
case 5:
{
 cout << "___ Мои заказы" << endl << endl;
 int l = 0;
 cout << "1__ Сухопутные туры" << endl << endl;

 LandInternational::header();

 for (int i = 0; i < orders->getSize(); i++)
 {
    if ((*orders)[i].getClientCode() == client.getClientCode())
    {
       for (int j = 0; j < lands->getSize(); j++)
       {
    if ((*orders)[i].getTourCode() == (*lands)[j].getTourCode())
           {
              (*lands)[j].Show();
              cout << endl;
              l++;
           }
      }
    }
 }
 LandInternational::headerLine();
 if (l == 0) cout << "___ Пусто" << endl;
 cout << "\n\n";
 l = 0;
 cout << "2__ Морские туры" << endl << endl;
 SeaInternational::header();
 for (int i = 0; i < orders->getSize(); i++)
 {
   if ((*orders)[i].getClientCode() == client.getClientCode())
```

```
           {
             for (int j = 0; j < sea->getSize(); j++)
             {
               if ((*orders)[i].getTourCode() == (*sea)[j].getTourCode())
               {
                  (*sea)[j].Show();
                  cout << endl;
                  l++;
               }
             }
           }
         }

         SeaInternational::headerLine();

         if (l == 0) cout << "___ Пусто" << endl;
         cout << "\n\n";
         l = 0;
         cout << "3__ Местные туров" << endl << endl;
         LocalTour::header();
         for (int i = 0; i < orders->getSize(); i++)
         {
            if ((*orders)[i].getClientCode() == client.getClientCode())
            {
               for (int j = 0; j < localtours->getSize(); j++) {
     if ((*orders)[i].getTourCode() == (*localtours)[j].getTourCode())
                {
                     (*localtours)[j].Show();
                     cout << endl;
                     l++;
                  }
               }
            }
         }
          LocalTour::headerLine();
          if (l == 0) cout << "___ Пусто" << endl;
          break;
        }
        case 6:
        {
           cout << "___ Мои билеты" << endl << endl;

           Ticket::header();
           for (int i = 0; i < tickets->getSize(); i++)
           {
             if ((*tickets)[i].getUserCode() == client.getClientCode())
             {
                 (*tickets)[i].Show();
                 cout << endl;
              }
           }
           Ticket::headerLine();
           break;
        }
    }
     cout << endl;
     system("pause");
     writeOrder();
   } while (vybor != 0);
}
```

```
inline void InterfaceClient::loadTicket()
{
   char filename[30] = "tickets.txt";
   File<Ticket> f(filename);
   Ticket t;
   while (1)
  {
       f.ReadData(t);
       if (f.REndFile())
        break;
       tickets->enqueue(t);
   }
}

inline void InterfaceClient::loadOrder()
{
   char filename[30] = "orders.txt";
}


inline void InterfaceClient::writeOrder()
{
   char filename[30] = "orders.txt";
   File<Order> f(filename);
   while (orders->getSize()!=0)
   {
     Order t = orders->dequeue();
     f.WriteData(t);
   }
}

template<class T>
inline void InterfaceClient::makeOrder(Queue<T>* q)
{
   cout << "___ Введите номер тура" << endl;
   int k;
   cin >> k;
   k--;
   Order r;
   r.setClientCode(client.getClientCode());
   r.setTourCode((*q)[k].getTourCode());
   orders->enqueue(r);
}

#pragma once
#include <iostream>
#include <iomanip>
using namespace std;

template<typename T>
struct Node {
   T data;
   Node<T>* next = NULL;
   Node<T>* prev = NULL;
};

template<typename T>
class Queue
{
private:
   int Size;
   Node<T>* head;
   Node<T>* tail;
public:
   Queue() : head(NULL), tail(NULL), Size(0) {}
   ~Queue();
   Queue(const Queue<T>& obj);
   void Delete(int index);
   void show();
   T& operator[](int index);
   void enqueue(T data);
   bool is_Empty();
   int getSize();
   T dequeue();
   void Clear();
};

template<typename T>
inline Queue<T>::~Queue(){
 Clear();
}

template<typename T>
inline Queue<T>::Queue(const Queue<T>& obj){
   this->head = obj.head;
   this->Size = obj.Size;
   this->tail = obj.tail;
}
template<typename T>
inline void Queue<T>::Delete(int index)
{
   Node<T>* temp = head;
   if (index < 0 || index > Size)
     return;
   if (index == 1)
   {
     if (!(head)) return;
     Node<T>* temp = head;
     head = head->next;
     if (head) head->prev = nullptr;
     delete temp;
     Size--;
     return;
   }
 else if (index == Size)  {
      dequeue();
      return;
    }
   else
   {
     for (int i = 1; i < index; i++)   {
        temp = temp->next;
      }
       temp->prev->next = temp->next;
       temp->next->prev = temp->prev;
   }
   delete temp;
   Size--;
}
```

```
template<typename T>
inline void Queue<T>::show()
{
   Node<T>* temp = head;
   for (int i = 0; i < Size; i++) {
       cout << setw(2) << i + 1;
       temp->data.Show();
       cout << endl;
       temp = temp->next;
   }
}

template<typename T>
inline T& Queue<T>::operator[](int index)
{
   Node<T>* curr = head;
   if (index >= Size || index < 0)
    return curr->data;

   for (int i = 0; i < index; i++)
    {
      curr = curr->next;
   }
   return curr->data;
}

template<typename T>
inline void Queue<T>::enqueue(T data)
{
    if (Size == 0)
    {
        head = new Node<T>;
        head->prev = NULL;
        head->next = NULL;
        head->data = data;
        tail = head;
        Size++;
        return;
    }
    Node<T>* temp = new Node<T>;
    temp->data = data;
    temp->prev = nullptr;
    temp->next = head;
    head->prev = temp;
    head = temp;
    Size++;
}

template<typename T>
inline bool Queue<T>::is_Empty(){
  return Size == 0;
}

template<typename T>
inline int Queue<T>::getSize(){
 return Size;
}

template<typename T>
inline T Queue<T>::dequeue(){
   if (!(tail)) return T();
   T data = tail->data;
   Node<T>* temp = tail;
   tail = tail->prev;
   if (tail)
    tail->next = nullptr;
   delete temp;
   Size--;
   return data;
}
template<typename T>
inline void Queue<T>::Clear(){
 while (Size != 0) {
   dequeue();
 }
}

//Functions.h
#pragma once
#include "Header.h"
#include "Queue.h"
#include "File.h"

constexpr auto min = 3;
constexpr auto max = 20;

template <class T>
void ReadDataFromFile(Queue<T>*& obj, char* filename);

template <class T>
void WriteDataToFile(Queue<T>*& obj, char* filename);

template<class T>
inline void ReadDataFromFile(Queue<T>*& obj, char* filename)
{
  File<T> file(filename);
  T data;
  obj->Clear();
  while (1)
  {
    file.ReadData(data);
    if (file.REndFile()) break;
    obj->enqueue(data);
  }
}

template<class T>
inline void WriteDataToFile(Queue<T>*& obj, char* filename)
{
  File<T> file(filename);
  while (obj->getSize() != 0)
  {
    T d = obj->dequeue();
    file.WriteData(d);
  }
}

template<typename T>
inline T enterInt(T mini, T maxi)
{
  T i;
  bool flag = true;
  do {
     flag = true;
     cin >> i;
     if (!cin.good() || cin.peek() != '\n')
     {
      cout << "___ Введено не число" << endl;
      flag = false;
      rewind(stdin);
      cout << endl;
      cin.clear();
    }
    else if (i < mini || i > maxi) {
      flag = false;
cout <<"___ Введите число в интервале от "<<mini <<" до "<<maxi<<endl;
      rewind(stdin);
      cout << endl;
      cin.clear();
    }
  } while (flag!= true);
  return i;
}

char* enterCharOnly();


#include "Functions.h"

char* enterCharOnly()
{
 char* tmpo;
 int flag;
 do {

   tmpo = new char[81];
   flag = 0;
   rewind(stdin);
   cin.getline(tmpo, 80);
   int k = strlen(tmpo);

   if (tmpo[0] >= '0' && tmpo[0] <= '9')
   {
      flag = 1;
      _flushall();
      cout << "___ Введено не число" << endl;
      delete[] tmpo;
      cout << endl;
     }
  } while (flag);
  return tmpo;
}

#include "Header.h"
#include "Interface.h"
#include "Client.h"
#include "File.h"
#include "Order.h"
#include "SeaInternational.h"
#include "LandInternational.h"
#include "LocalTour.h"
#include "Ticket.h"
```

```
#include "InterfaceClient.h"
#include "Functions.h"

void AdminMenu();

int main()
{
 system("chcp 1251");
 setlocale(LC_ALL, "rus");
 system("cls");

 int vybor;
 char f[30] = "clients.txt";

 Queue<Client>* clients = new Queue<Client>;

 ReadDataFromFile(clients, f);

 do {
  system("cls");
  cout << "___ Добро пожаловать в систему туристического агентство!" << endl << endl;
  cout << "1__ Войти в систему" << endl;
  cout << "2__ Регистрация" << endl;
  cout << "3__ Выйти из программы" << endl << endl;

  vybor = enterInt(1, 3);
  cout << endl;

  if (vybor == 1)
  {
   string _log, _pas;
   cout << "___ Логин: "; cin >> _log;
   cout << "___ Пароль: "; cin >> _pas;

   if (_log == "admin" && _pas == "admin")
   {
    AdminMenu();
   }
   else
   {
    bool found = false;
    int k;
    for (int i = 0; i < clients->getSize(); i++)
    {
     if ((*clients)[i].getLogin() == _log && (*clients)[i].getPassword() == _pas)
     {
      found = true;
      k = i;
      break;
     }
    }

    if (found)
    {
     cout << "___ Успешно" << endl << endl;
     system("pause");
     InterfaceClient interfaceCC((*clients)[k]);
     interfaceCC.start();
    }
    else
    {
     cout << "___ Неправильные данные" << endl;
    }
   }
  }
  else if (vybor == 2)
  {
   Client obj;
   obj.PutData();
   clients->enqueue(obj);
  }
  system("pause");
 } while (vybor != 3);

 WriteDataToFile(clients, f);
 return 0;
}

void AdminMenu()
{
 int vybor;
 do {
  system("cls");
  cout << "___ Меню администратора" << endl << endl;
  cout << "1__ Раздел сухопутные туры" << endl;
  cout << "2__ Раздел морские туры" << endl;
  cout << "3__ Раздел местные туры" << endl;
  cout << "0__ Выйти из программы" << endl;

  vybor = enterInt(0, 3);
  switch (vybor)
  {
   case 1:
   {
    char filename[30] = "landtours.txt";
    Interface<LandInternational> LAND_TOUR_INTERFACE(filename);
    LAND_TOUR_INTERFACE.start();
    break;
   }
   case 2:
   {
    char filename[30] = "seatours.txt";
    Interface<SeaInternational> SEA_TOUR_INTERFACE(filename);
    SEA_TOUR_INTERFACE.start();
    break;
   }
   case 3:
   {
    char filename[30] = "localtours.txt";
    Interface<LocalTour> LOCAL_TOUR_INTERFACE(filename);
    LOCAL_TOUR_INTERFACE.start();
    break;
   }
   default: break;
  }
 } while (vybor != 0);
}
```