

University of Malta

Faculty of ICT

Department of Computer Science

Airline reservations system

by

Lazar Lazarević

Lecturer: Dr. Mark J. Vella

Study unit: Programming principles in C

Study unit code: CPS 1003

# Summary

In this report, it will be discussed about data objects used for the assignment, how program is implemented by explaining how each function works, providing algorithm for each and in which file it can be found if someone wish to take a look at the source code. The testing will be presented in great details, testing all possible user inputs and showing how program takes care of bad inputs.

The last section is source code, but for better understanding it is preferable to read the code from editor such as visual studio, code block or eclipse, because for each line of code a comment is written describing what that code does. Moreover, functions are also documented; comments will be seen before each function explaining function’s purpose.

Note: since the software is developed under 64 bit machine, some unexpected errors might occur under 32 bit machines, but program’s functionality will remain the same. The only problem that occurred when I run the program at 32 bit laptop was printing nothing from a file. To prevent this, new data files should be created on each platform in order to be saved in correct format. In other words, if data is saved on 64 bit PC, and then tried to be read on 32 bit PC, there is a chance for data to be represented in bad format but not necessarily.

[Summary 2](#_Toc375419136)

[1. Data Objects 5](#_Toc375419137)

[1.1 Passenger 5](#_Toc375419138)

[1.2 Flight 6](#_Toc375419139)

[2. Implementation 7](#_Toc375419140)

[2.1 User interface 7](#_Toc375419141)

[2.2 Source file organization 8](#_Toc375419142)

[2.3 Data type definitions 8](#_Toc375419143)

[2.4 Functions 9](#_Toc375419144)

[2.4.1 User input / input validation 9](#_Toc375419145)

[short get\_input(short nMIN, short nMAX) 9](#_Toc375419146)

[unsigned long long int get\_int(short n) 9](#_Toc375419147)

[void get\_Title(char arr[]) 10](#_Toc375419148)

[void get\_date(short arr[], short minYear, short maxYear, short n) 10](#_Toc375419149)

[short get\_Time(short n) 10](#_Toc375419150)

[void normalize\_name(char arr[]) 11](#_Toc375419151)

[void get\_Name(char arr[], short n, short maxN) 11](#_Toc375419152)

[unsigned int checkPass(short chosenP, short count, const struct Passenger \*passenger) 12](#_Toc375419153)

[short checkID(short flight, short count, const struct Flight \*scheduledFlight) 12](#_Toc375419154)

[short openPassengerFile(char \*mode) / 12](#_Toc375419155)

[short openfscFlightFile(char \*mode) 12](#_Toc375419156)

[2.4.2 Data objects manipulation 13](#_Toc375419157)

[short createNewPassenger(struct Passenger \*passenger, short n) / 13](#_Toc375419158)

[void createFlight(struct Flight \*scheduledFlight) 13](#_Toc375419159)

[void modifyPassenger(struct Passenger \*passenger) / 14](#_Toc375419160)

[void modifyFlight(struct Flight \*scheduledFlight) 14](#_Toc375419161)

[void cancelFlight(struct Flight \*scheduledFlight) 14](#_Toc375419162)

[void show\_PassengerInfo(short n) / 15](#_Toc375419163)

[void show\_FlightInfo(short n, const struct Flight \*scheduledFlight) / 15](#_Toc375419164)

[void show\_existingPassengers(const struct Passenger \*passenger) / 15](#_Toc375419165)

[void show\_existingFlights(const struct Flight \*scheduledFlight) / 15](#_Toc375419166)

[short listPassengers(const struct Passenger \*passenger) / 15](#_Toc375419167)

[short listFlights(const struct Flight \*scheduledFlight) 15](#_Toc375419168)

[2.4.3 Reservations 16](#_Toc375419169)

[void iniPassRes(struct Passenger \*passenger, const struct Fligh \*scheduledFlight) 16](#_Toc375419170)

[short findFreeElement(struct Passenger \*passenger, short n) 16](#_Toc375419171)

[void makeReservationExistingP(struct Passenger \*passenger, struct Flight \*scheduledFlight, short n, short newPass) 17](#_Toc375419172)

[void makeReservationNewP(struct Passenger \*passenger, struct Flight \*scheduledFlight) 18](#_Toc375419173)

[void show\_reservations(const struct Passenger \*passenger, const struct Flight \*scheduledFlight) 18](#_Toc375419174)

[3. Store/Load Functionality 19](#_Toc375419175)

[4. Testing 20](#_Toc375419176)

[4.1 Testing scheduled flight 20](#_Toc375419177)

[4.1.1 Scheduled flight creation 20](#_Toc375419178)

[4.1.2 Scheduled flight modification 23](#_Toc375419179)

[4.1.3 Scheduled flight cancelation 26](#_Toc375419180)

[4.1.4 Presenting all scheduled flights 27](#_Toc375419181)

[4.2 Testing passenger 27](#_Toc375419182)

[4.2.1 Passenger creation 28](#_Toc375419183)

[4.2.2 Passenger modification 29](#_Toc375419184)

[4.2.3 Presenting all passengers 31](#_Toc375419185)

[4.3 Testing reservations 32](#_Toc375419186)

[4.3.1 Reservation for new passenger 32](#_Toc375419187)

[4.3.2 Reservation for existing passenger 33](#_Toc375419188)

[4.3.3 Presenting all passenger’s past reservations 34](#_Toc375419189)

[5. Appendix 35](#_Toc375419190)

[5.2 Passenger 35](#_Toc375419191)

[5.2.1 Passenger.h 35](#_Toc375419192)

[static struct Passenger 35](#_Toc375419193)

[5.2.2 Passenger.c 37](#_Toc375419194)

[void get\_Name(char arr[], short n, short maxN) 37](#_Toc375419195)

[void normalize\_name(char arr[]) 39](#_Toc375419196)

[void get\_Title(char arr[]) 39](#_Toc375419197)

[short createNewPassenger(struct Passenger \*passenger, short n) 41](#_Toc375419198)

[void show\_existingPassengers(const struct Passenger \*passenger) 43](#_Toc375419199)

[void show\_PassengerInfo(short n) 43](#_Toc375419200)

[void menu(void) 44](#_Toc375419201)

[void modifyPassenger(struct Passenger \*passenger) 45](#_Toc375419202)

[unsigned int checkPass(short chosenP, short count, const struct Passenger \*passenger) 46](#_Toc375419203)

[short listPassengers(const struct Passenger \*passenger) 46](#_Toc375419204)

[void get\_date(short arr[], short minYear, short maxYear, short n) 47](#_Toc375419205)

[short get\_input(short nMIN, short nMAX) 48](#_Toc375419206)

[5.3 Scheduled flight 49](#_Toc375419207)

[5.3.1 Flight.h 49](#_Toc375419208)

[static struct Flight 49](#_Toc375419209)

[5.3.2 Flight.c 50](#_Toc375419210)

[void createFlight(struct Flight \*scheduledFlight) 50](#_Toc375419211)

[void modifyFlight(struct Flight \*scheduledFlight) 52](#_Toc375419212)

[void cancelFlight(struct Flight \*scheduledFlight) 53](#_Toc375419213)

[void show\_existingFlights(const struct Flight \*scheduledFlight) 54](#_Toc375419214)

[void show\_FlightInfo(short n, const struct Flight \*scheduledFlight) 55](#_Toc375419215)

[short listFlights(const struct Flight \*scheduledFlight) 55](#_Toc375419216)

[short get\_Time(short n) 56](#_Toc375419217)

[short checkID(short flight, short count, const struct Flight \*scheduledFlight) 57](#_Toc375419218)

[short openfscFlightFile(char \*mode) 57](#_Toc375419219)

[5.4 Reservation 58](#_Toc375419220)

[5.4.1 Reservation.h 58](#_Toc375419221)

[5.4.1 Reservatoin.c 58](#_Toc375419222)

[void makeReservationExistingP(struct Passenger \*passenger, struct Flight \*scheduledFlight, short n, short newPass) 58](#_Toc375419223)

[void makeReservationNewP(struct Passenger \*passenger, struct Flight \*scheduledFlight) 60](#_Toc375419224)

[void show\_reservations(const struct Passenger \*passenger, const struct Flight \*scheduledFlight) 60](#_Toc375419225)

[short findFreeElement(struct Passenger \*passenger, short n) 61](#_Toc375419226)

[void iniPassRes(struct Passenger \*passenger, const struct Fligh \*scheduledFlight) 62](#_Toc375419227)

# Data Objects

For this assignment, used data objects are: Passenger and Flight.

## Passenger

Passenger is of type **struct** and it is used to store various passengers’ information.

Code:

static struct Passenger

{

char name[MAXNAME];

char surname[MAXNAME];

char title[MAXTITLE];

char country[MAXNAME];

char city[MAXNAME];

char homeAddress[MAXADDRESS];

short DOB[DATE];

short reservations[MAXPASSENGERS];

unsigned int passportNo;

unsigned long long int mobNo;

}passenger[MAXPASSENGERS];

Character arrays name, surname, title, country, city and home address are used to store passenger’s details just as the names say respectively. Integers, in this case short DOB and reservations are used to store passenger’s date of birth and reservations being made by him/her. Unsigned integer passportNo stores traveller’s passport number while unsigned long long integer mobNo stores mobile number.

## Flight

Flight is of type struct and it contains details about scheduled flights.

static struct Flight

{

short flightID;

short departureDate[DATE];

short timeDeparture;

short timeArrival;

short availableSeats;

char airline[MAXNAME];

char departureCity[MAXNAME];

char arrivalCity[MAXNAME];

char status[STATUS];

}scheduledFlight[MAXPASSENGERS];

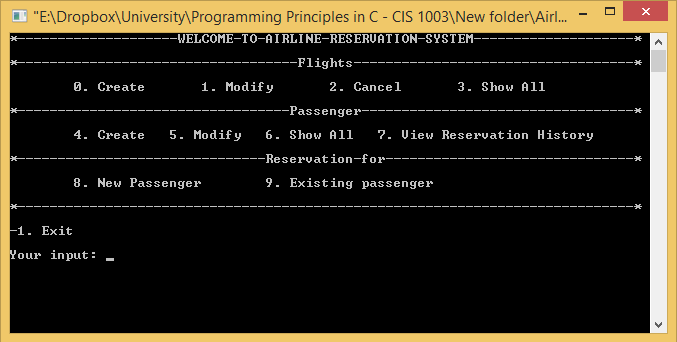
This structure is made of five integer short type and four arrays of character type. FlightID stores identifier for particular scheduled flight and it is unique. DepartureDdate stores information when the flight will take place, timeDeparture and timeArrival at what time and availableSeats represents how many seats for a specific flight has left. Arrays airline, departureCity and arrivalCity stores airline name, in which city the plane will take of and where it will land respectively while status shows whether the scheduled flight is cancelled or not.

# Implementation

This program has been designed to be user-friendly as much as possible while providing maximum level of security and reliability. The software will only accept correct input, while for bad will show corresponding error and request a new one. Input validation and robustness will be presented in greater detail later in this chapter and with chapter 3, Testing.

## 2.1 User interface

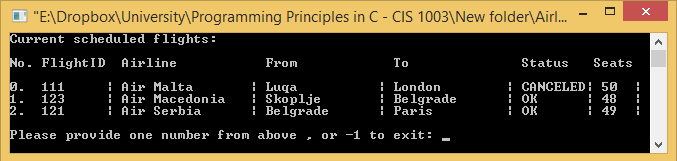
User interface is designed in a way to be more readable by a user, not taking much space and clearing console window when needed. Below is example of main menu, the first screen that will be seen by a user.



Frist line is name of the application, second is name referring to the third line. Option 0 means to create a new scheduled flight, “modify” and “cancel” to edit and to call of scheduled flight respectively, while “show all” will display all created scheduled flights.

Forth line “passenger” refers to fifth line. “Create”, “modify” and “show all” represents the same actions as for flight but this time for passenger, while “view reservation history” shows all reservations made by a particular passenger.

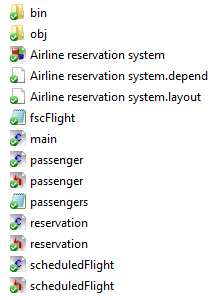
Sixth line refers to seventh. “New passenger” will create new passenger and make reservations for him/her and “existing passenger” will make reservations for passengers that have already been created.



With this example (modify flight), it is presented “screen clear”, and listing of created flights and their details.

## Source file organization

In this project, three header files have been created, passenger, scheduledFlight and reservation as seen below. There are four c source files, passenger, scheduledFlight, reservation and main. First three contain functions declaration which definitions are in corresponding header file, while main contains only one function call which controls all others.



## Data type definitions

The mostly used data types are short and char. At first, only int types were used, because at that moment the size of variables were unknown. When the sizes became known, all int types were converted to short where it was possible and array sizes had been reduced. It was interesting that by converting ints to shorts and reducing array sizes by few elements reduced executable size by almost 50%.

## Functions

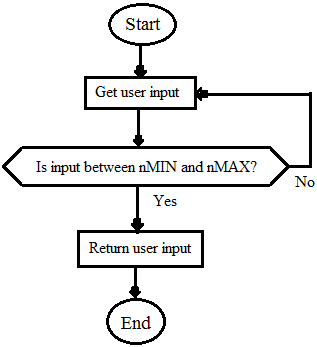
The actual implementation will be discussed here. Explaining how functions work and how data is handled with input validation.

### User input / input validation

Algorithms will be presented for easier understanding of basic function’s procedure.

#### short get\_input(short nMIN, short nMAX)

Found in passenger.c



This function takes two short numbers as arguments. First represents the minimum value that user can enter, while the other one maximum value.

if(scanf("%hd", &option) != 0)

{

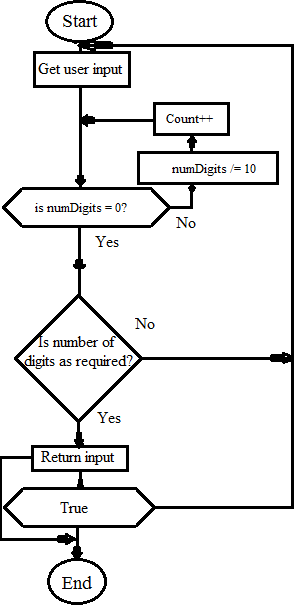
if(option >= nMIN && option <= nMAX)

break;

}

This is basic input validation. If input is not number it will show error, otherwise it will check if the input is between minimum and maximum values, and if yes break the while loop, returning user input.

#### unsigned long long int get\_int(short n)



Found in passenger.c

This function takes a number which represents the number of digits required. Input validation is the same as the foregoing

while(numDigits) {

numDigits /= 10;

count;

}

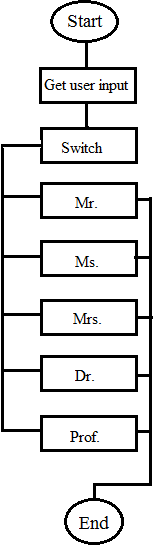
This piece of code will calculate number of digits

if(count == n) return num;

And if it is as required, it will return user input.

#### void get\_Title(char arr[])

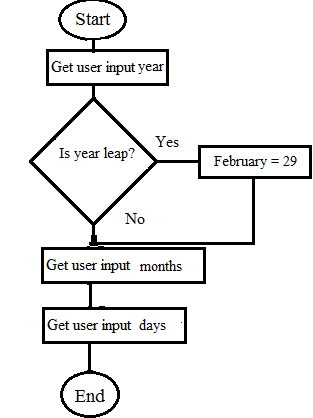
Found in passenger.c



This function is straight forward, it takes user input and depending on number provided, it will store a string into array passed as argument.

For input validation it uses get\_input function.

#### void get\_date(short arr[], short minYear, short maxYear, short n)



Found in passenger.c

Function as arguments takes array where data will be stored, minimum and maximum year that can be inputted, and n which determines what to print.

With code below number of days for each month are initialized.

short mm[] = {31,28,31,30,31,30,31,31,30,31,30,31};

if((arr[2]=get\_input(minYear, maxYear)) % 4 == 0) mm[1]=29

With this code, user is requested to input a year, after that it is calculated whether the year is leap or not, if yes, change number of days of February to 29.

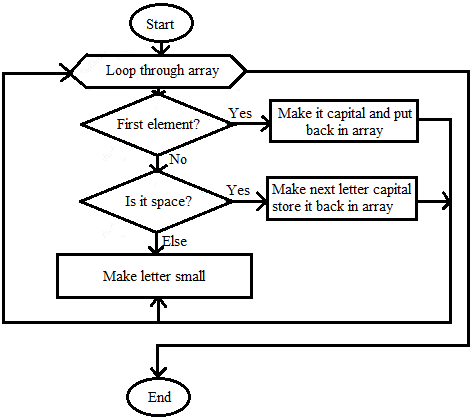
#### short get\_Time(short n)

In: scheduleFlight.c

Very simple function, it uses get\_input to get hours and minutes from user and at the end it return hours and minutes as one number. N only determines what will be printed.

return hour\*100 + min;

#### void normalize\_name(char arr[])

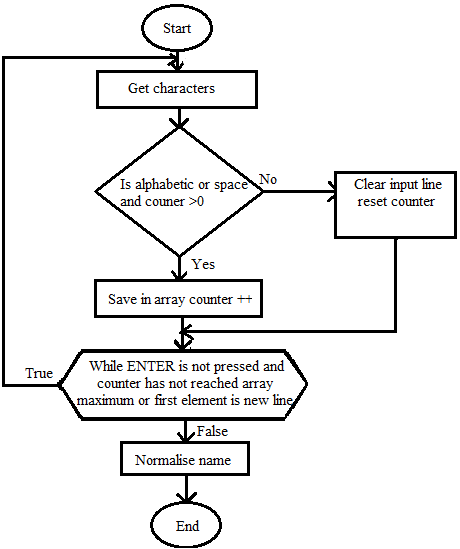


Passenger.c

Function as argument takes an array which holds a name. Usually this name is not in correct format e.g. LaAaR. After passing trough this function, name will be normalized to Lazar. If there are spaces, after each space, letter will be capitalized.

If it is not first letter nor letter after space, letter will be decapitalized.

#### void get\_Name(char arr[], short n, short maxN)

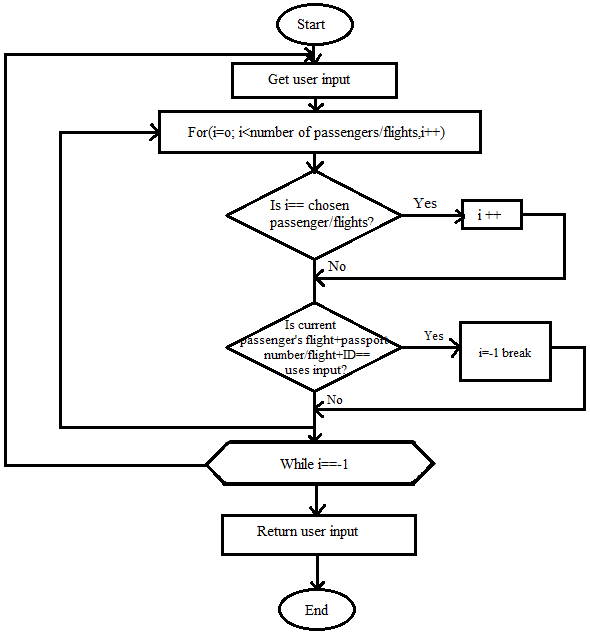


passenger.c

With this function user is asked to enter a name. After enter is being pressed, each letter is checked for validation. If it is number, special character or a space at first array element, error will be shown and request for new name will be given. Otherwise, letter will be stored into array provided as argument. N indicates what will be printer and to make available to press enter for name of passenger and airline company since that is termination character in function for creating passenger and scheduled flight. If everything is in correct format, name will be normalized with function normalize\_name.

unsigned int checkPass(short chosenP, short count, const struct Passenger \*passenger) /

#### short checkID(short flight, short count, const struct Flight \*scheduledFlight)



passenger.c and scheduledFlight.c

As arguments it takes chosen passenger/flight, number of records in a file, and structure.

This function checks whether a passport number of flight ID already exist. If yes, user will be prompted to enter a new number.

if(chosenP == i)

i++;

If this function is called in modify part, it will skip the passenger or scheduled flight being edited because he/she/it can retain old number.

#### short openPassengerFile(char \*mode) /

#### short openfscFlightFile(char \*mode)

This function is used to open a binary file where data is stored.

if ((fscFlight = fopen("fscFlight.dat", mode)) == NULL)

{

fputs("Cannot open fscFlight.dat file.\n",stderr);

return 0;

}

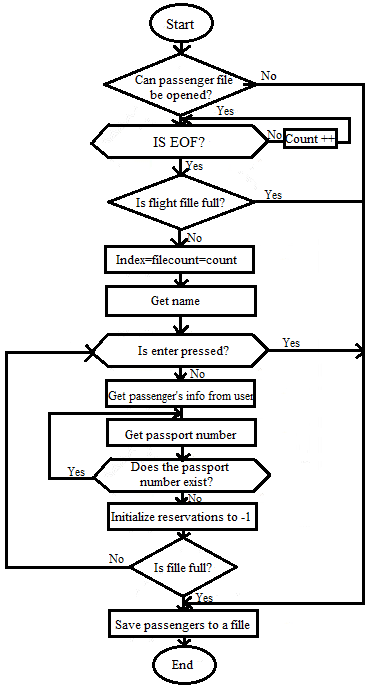
else return 1;

In case that the file cannot be opened, function will return zero, otherwise one will be returned. This is to prevent program from crashing and other errors.

### Data objects manipulation

#### short createNewPassenger(struct Passenger \*passenger, short n) /

#### void createFlight(struct Flight \*scheduledFlight)



passenger.c and scheduledFlight.c

This function/s is responsible for creation of new passenger or schedule flight. It will first check whether a file can be opened or not.

if(!openPassengerFile("a+b")) return;

If file cannot be opened, terminate function.

Then it will read all records into structure, and if the file is full, function will end.

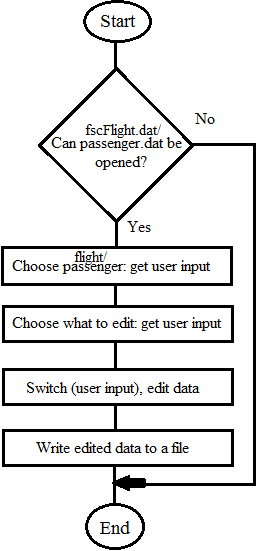
User will be then prompted to enter passenger’s or scheduled flight’s details. If enter is pressed at the name input, function will exit and save all records to a file if any is created.

After one objet is created and if there is still free space in file, user will be prompted again to enter a name.

For createFlight principle is exactly the same as well as alghorithm, therefore no explanation is needed of it.

#### void modifyPassenger(struct Passenger \*passenger) /

#### void modifyFlight(struct Flight \*scheduledFlight)



Found in: passenger.c and scheduledFlight.c respectively

if((count = listPassengers(passenger)) != -1 && (optionP = get\_input(-1, count)) != -1)

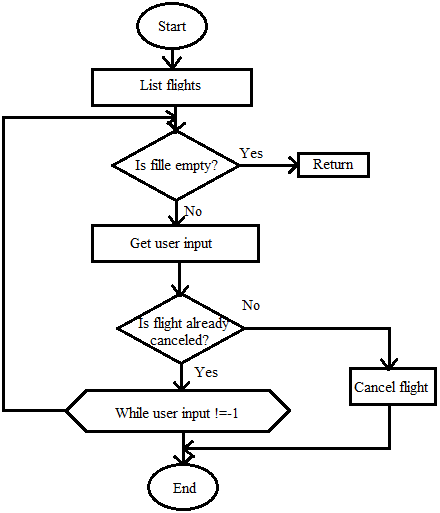
The code above will list all passengers and get user input. If count is not -1, which function listPassengers returns if there is no data, or user input is -1, function will be terminated. Here user input is to choose passenger to be edited.

if((optionE = get\_input(-1, 9)) != -1)

With this code, user choose what part to edit. If -1 is inputed, program will terminate.

The same principles applies for modifyFlight, so it would be unnecessary to explain it separately.

#### void cancelFlight(struct Flight \*scheduledFlight)

scheduledFlight.c

Function to cancel scheduled flight.

if(strcmp(scheduledFlight[optionF].status, "CANCELLED") != 0)

With this code above, it is checked whether a flight has already being cancelled or not. If it is not cancelled, string CANCELLED is copied into a string which holds status information.

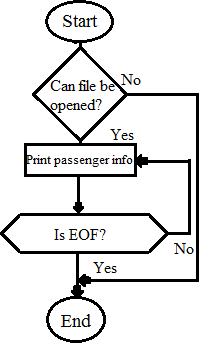
User will be prompted to confirm his decision about flight cancellation, and if he agrees, flight is cancelled.

#### void show\_PassengerInfo(short n) /

#### void show\_FlightInfo(short n, const struct Flight \*scheduledFlight) /

#### void show\_existingPassengers(const struct Passenger \*passenger) /

#### void show\_existingFlights(const struct Flight \*scheduledFlight) /



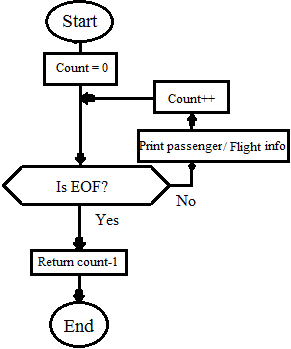
Found in: passenger.c and scheduledFlight.c respectively

The first two functions are very simple, their job is to print all details of corresponding data object (passenger or scheduled flight). Representing details in very nice format and easy readable by user.

The second two functions are used to print all passengers or scheduled flights respectively.

#### short listPassengers(const struct Passenger \*passenger) /

#### short listFlights(const struct Flight \*scheduledFlight)



Found in: passenger.c and scheduledFlight.c respectively

These two functions print details of data objects but in much simpler form used to present to a user which passenger of scheduled flight can be chosen for an action to be taken on them. The another difference of these two and foregoing functions is that these two return a number of records (passengers or scheduled flights). If there is no data in a file, the function will return -1.

### Reservations

#### void iniPassRes(struct Passenger \*passenger, const struct Fligh \*scheduledFlight)

reservation.c

This function will initialize array’s elements to -1,

for(index = 0; index < MAXPASSENGERS; index++)

{

passenger[count].reservations[index] = -1 if(passenger[count].reservations[index + 1] == -1) break

}

This code is looping trough array, setting the values to -1, if array has already been initialized but file was deleted, it would be unnecessarily to go to the end.

#### short findFreeElement(struct Passenger \*passenger, short n)

reservation.c

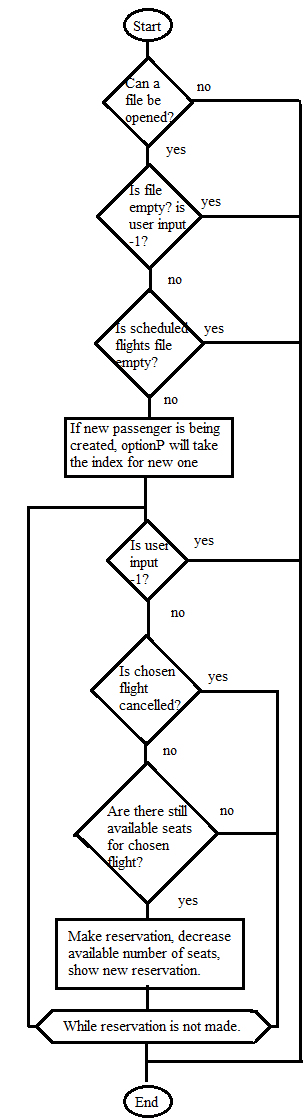
This function will loop trough reservations[] until -1 is found, since all elements are initialized to -1 at the creation of new passenger, therefore -1 represents a free element.

if(passenger[n].reservations[i] == -1)

break;

With this piece of code, loop will be stopped when free element is found. At the end, function will return index where the first -1 is.

#### void makeReservationExistingP(struct Passenger \*passenger, struct Flight \*scheduledFlight, short n, short newPass)



reservation.c

As arguments, function takes both data objects, n represent that new passenger is going to be created

This function will make a reservation for a passenger. At the beginning, it will check whether a files can be opened, and if user input is not -1.

N indicates that a reservation for new passenger is being made and in code:

if( n || ((count = listPassengers(passenger)) != -1 && (optionP = get\_input(-1, count)) != -1))

n is 1, counting passenger is not necessarily because a passenger is created at step before, also user input is not needed.

It will ask then to choose a flight which a passenger wants to book, and if it is not cancelled and if there are still available seats, reservation is done. On the other hand, while loop will loop through these steps until a valid reservation is not made or if user’s input is not -1.

#### void makeReservationNewP(struct Passenger \*passenger, struct Flight \*scheduledFlight)

reservation.c

This function will first create new passenger, and then make a reservation for him.

short passengerIndex = listPassengers(passenger) + 1;

This code will count the number of passenger in a file, and passenger index will hold an index where to save a new passenger.

if(createNewPassenger(passenger, 1))

makeReservationExistingP(passenger, scheduledFlight, 1, passengerIndex);

This code will create new passenger, if file is full or other errors had occurred, no reservations will be made nor new passenger will be created. Otherwise, new passenger is created and a reservation is being made for him/her.

#### void show\_reservations(const struct Passenger \*passenger, const struct Flight \*scheduledFlight)

reservation.c

This function shows a scheduled flight that is being booked by a chosen passenger.

# Store/Load Functionality

All data created or modified will be stored on hard disk or load into memory when needed. After creating new passenger or scheduled flight, .dat files will be created as shown below.



For example, when new passenger is created all data will be saved onto hard disk with this code:

fwrite(&passenger[filecount], size, count - filecount, fpassengers);

fpassenger is name of the pointer to a file where data will be saved.

With code below, data from the hard disk will be loaded into memory, making it available to access data for read or edit purposes.

while (fread(&passenger[count], size, 1, fpassengers) == 1)

{

count++; // increment counter

}

Codes like this or similar are used throughout a project, somewhere will be a little bit different depending on the current needs such as saving data to specific location in a file.

# Testing

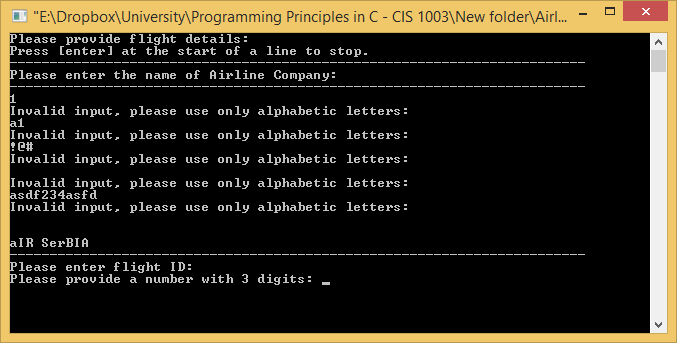
In this section, program functionality will be tested. All possible input will be provided for each part, showing program’s input validation, reliability and robustness.

## Testing scheduled flight

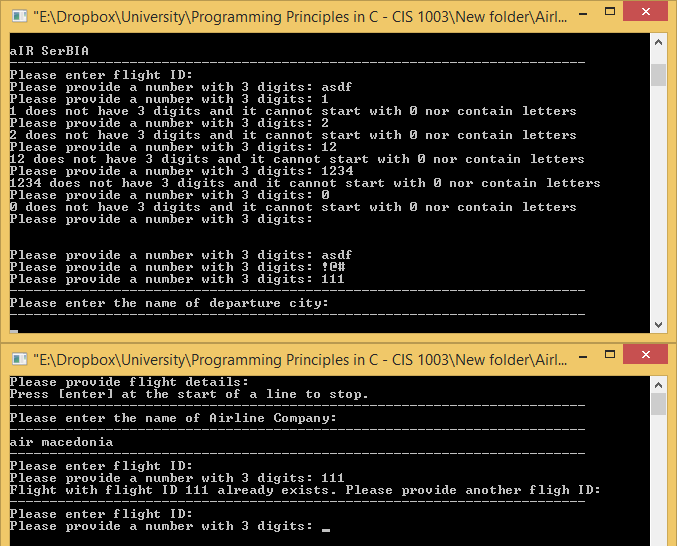
Here will be tested scheduled flight creation, modification and cancelation. At the end, all scheduled flights will be presented. For each input, validity will be checked in order to prevent bad inputs which can lead to incorrect data format or program crash.

### Scheduled flight creation

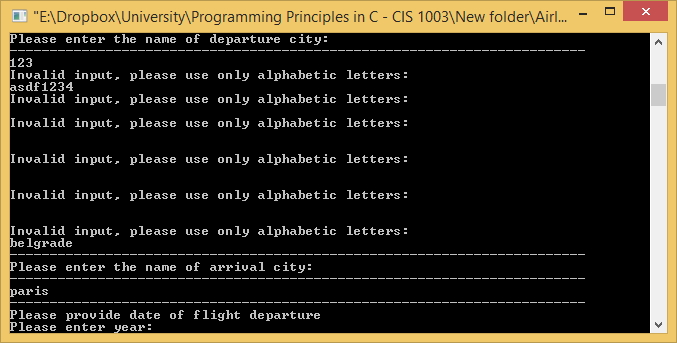
New scheduled flight is created by requesting from user to provide input for airline company name, flight ID, departure city as well as arrival city, date of flight departure and time.



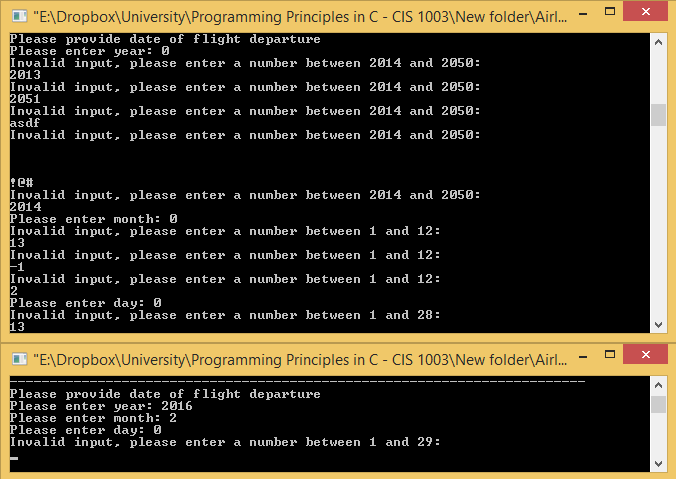
Here is shown that for any name requested, program will accept only alphabetic characters. If user input is correct, but capitalization is not, as above example aIR SerBIA, program will proceed, correcting input by normalizing the name to Air Serbia, as it will be sown at the end.



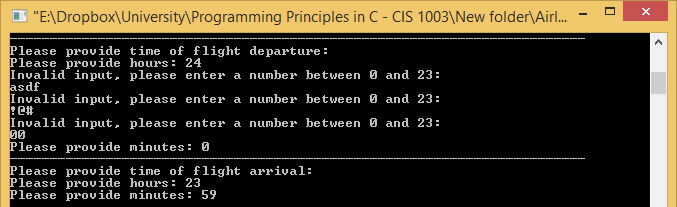
For flight ID only three digits are requested, every other input will be thrown away. If flight with enter ID already exist, program will inform user about that and request another one.



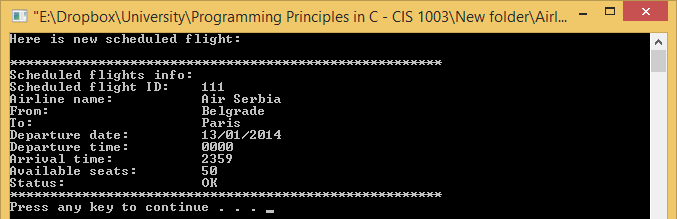
Another example of requesting correct name format from user.



For date of flight departure, first the year must be entered in order to calculate whether the inputted year is leap or not. Flights can be made only after 2013 till 2050. Next requested input is month, in order for next input to ask for correct number of days as shown above. If the year is leap and month is February, program will accept number 29.

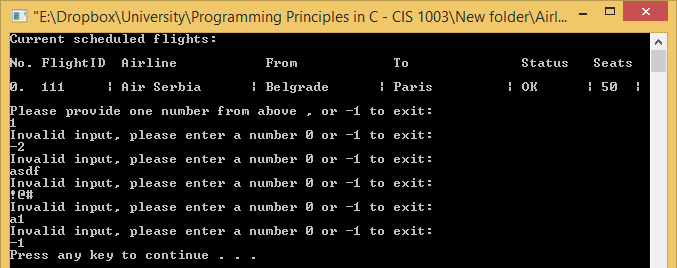


Next, time of departure and arrival is requested. Input can only be from 0 to 23 for hours, and from 0 to 59 to minutes. Bad inputs will be thrown away and new will be requested. After providing all inputs in correct format, new scheduled flight will be shown with all details. As it can be seen, airline name has been normalized.

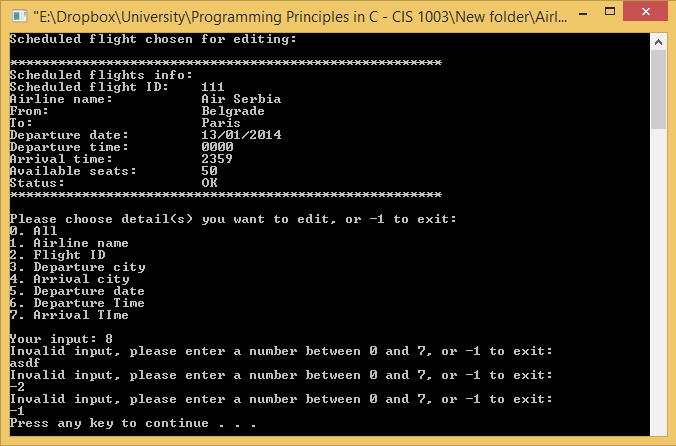


### Scheduled flight modification

With flight modification, any data can be changed separately or all at once.



First, all scheduled flights will be listed with number at the beginning which user can choose. Only shown numbers can be chosen or -1 if it is decided to not modify any flight.



When flight for modification is chosen, list of options will be provided. Again, only listed numbers can be chosen, all others will be ignored with adequate message, and new will be requested.



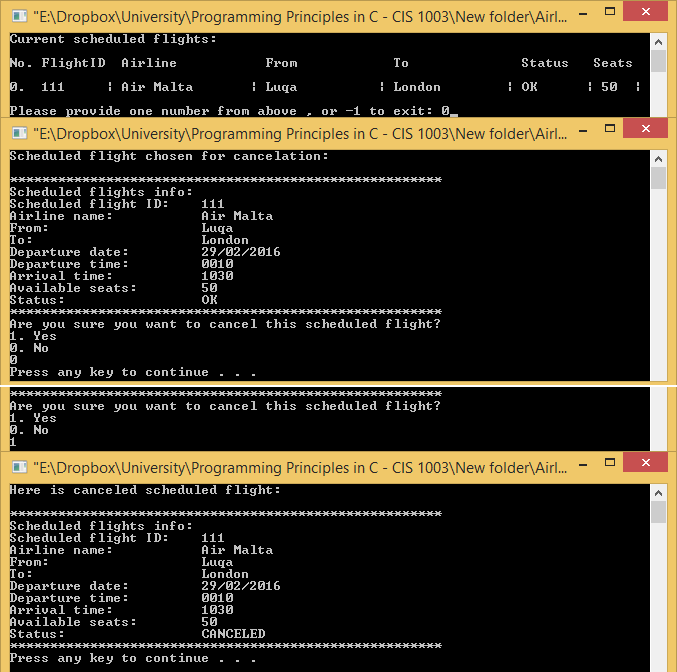
If all is chosen, program will prompt all details to be changed. At the end, edited scheduled flight is shown.



Here is presented that each input correspond to correct field to be edited.

### Scheduled flight cancelation

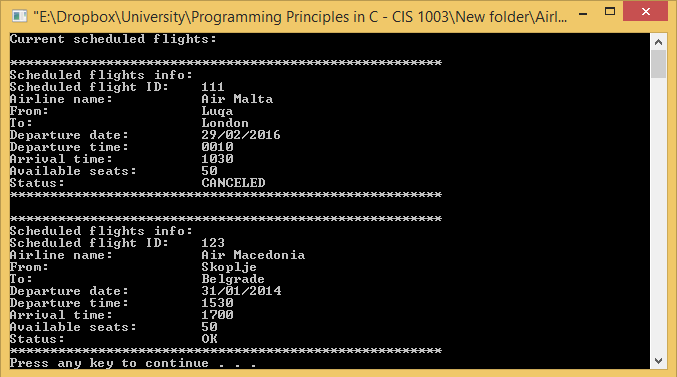
If it is decided that a particular flight cannot be done, cancelation is provided.



As always, the program will take care of bad inputs. If number 1 is chosen, flight will be cancelled, showing to user cancelled flight, otherwise nothing will happen, and main menu will be shown.

### Presenting all scheduled flights

If someone wishes to see all scheduled flights:



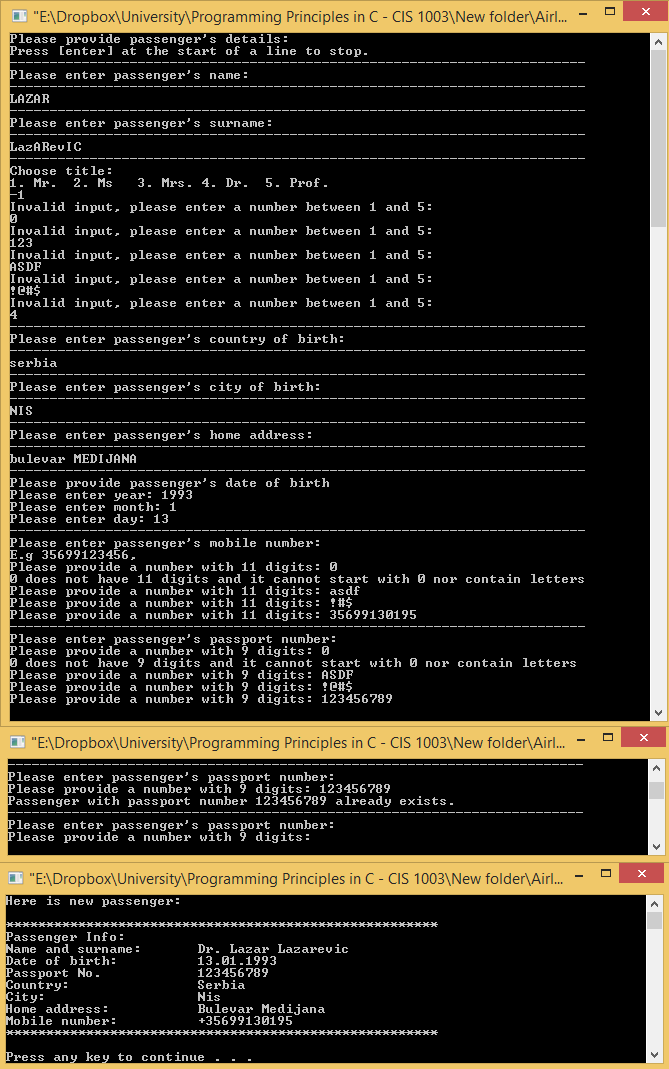
## Testing passenger

Here will be tested passenger creation and modification. At the end, all passengers will be presented. For each input, validity will be checked in order to prevent bad inputs which can lead to incorrect data format or program crash.

Since input validation is on the same principle as for the scheduled flight, many bad inputs will be skipped.

### Passenger creation

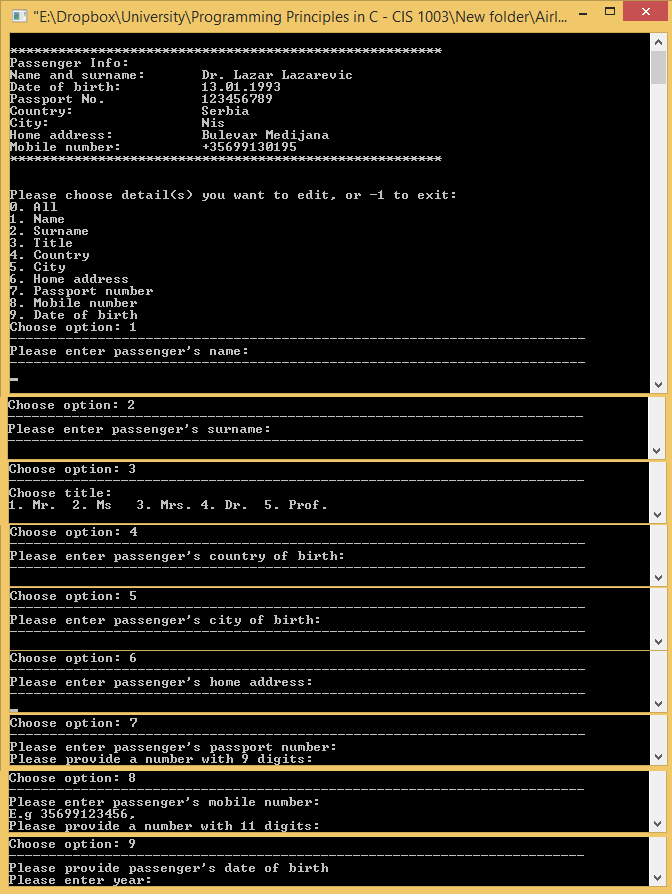
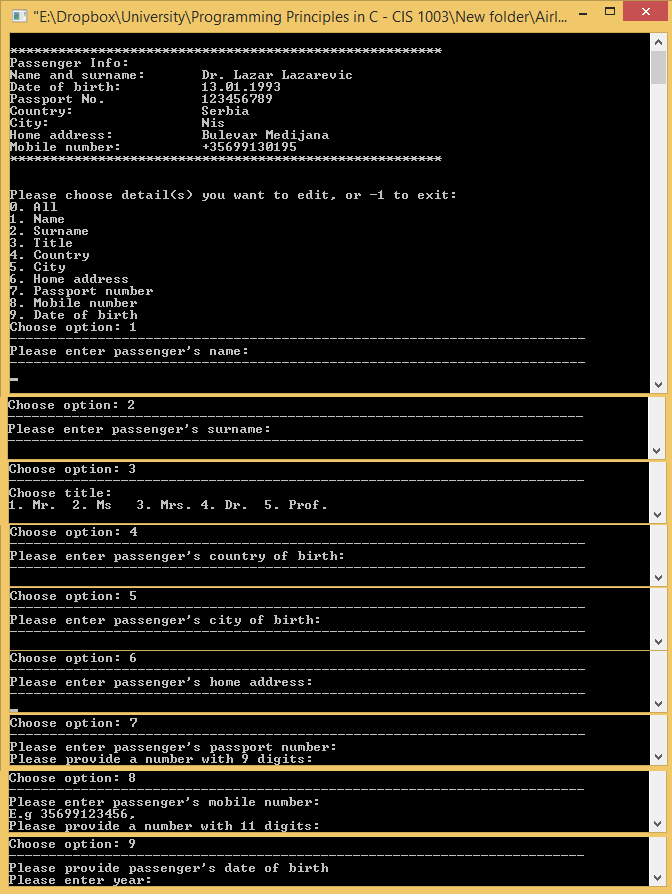
For passenger creation, details that will be requested are: name, surname, title, country of birth, city of birth, home address, date of birth, mobile number and passport.



For passenger’s title, only numbers from 1 to 5 are accepted. Mobile number and passport number cannot contain letters nor can start with 0. If passenger with provided passport number already exists, new passport number will be requested.

### Passenger modification

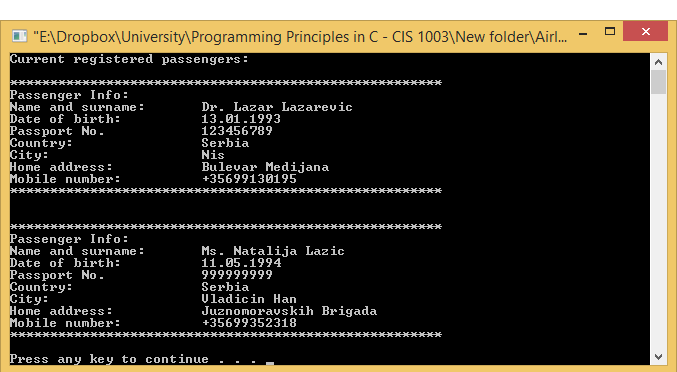
As in flight modification, user will be prompted to choose what to edit. In case below, everything will be edited.



In this case, we are testing that user input correspond to correct part of passenger detail to be edited. If same passport number is entered program will accept it. Otherwise error will be shown and new passport number will be requested.

### Presenting all passengers

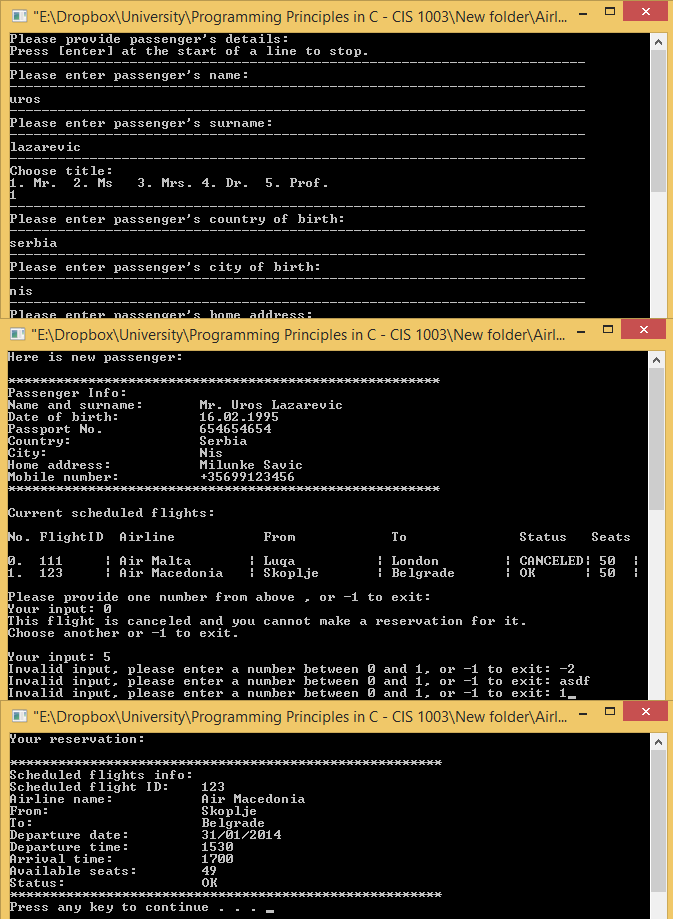
Here, all passengers are being shown, all with correct details that have been entered by a user.



## Testing reservations

In this part of the report, reservations will be tested. This include reservation for new passenger, reservation for existing passengers and show all past passenger’s reservations.

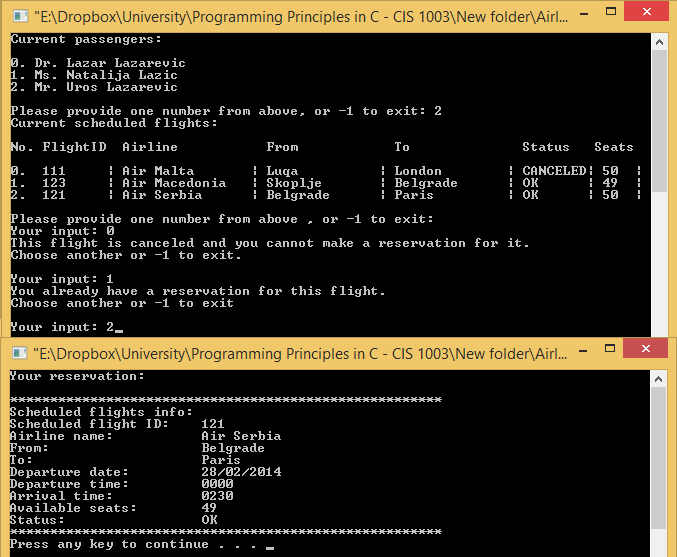
### Reservation for new passenger



This will create new passenger and make reservation for him/her. If cancelled flight is chosen, error will be shown and new input request will be provided. Bad inputs are handled. The number of seats will be reduced with each reservation being made by passenger.

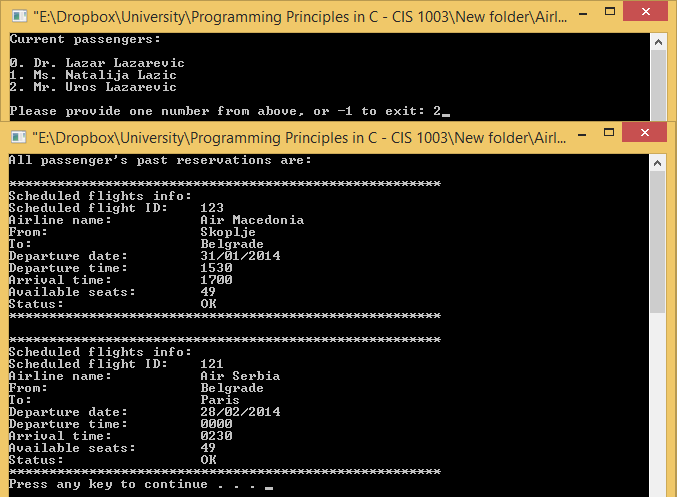
### Reservation for existing passenger

First passengers are listed for user to choose, then scheduled flights are listed. If flight cancelled reservation cannot be made as well as if reservation is already being made for particular flight by particular passenger. As shown below, bad inputs will not be accepted.



### Presenting all passenger’s past reservations

Here for chosen passenger, all reservations are shown. All details are correct and no incorrect data types are being shown as input validations are perfect.



# Appendix

## Passenger

### Passenger.h

#ifndef PASSENGER\_H\_INCLUDED

#define PASSENGER\_H\_INCLUDED

// constants for arrays size

#define MAXNAME 16 /\* maximum lenght for any name(name, surname, country, city...) \*/

#define MAXADDRESS 30 /\* maximum lenght for addresses \*/

#define MAXTITLE 6 /\* maximum lenght for titles \*/

#define MAXPASSENGERS 50 /\* maximum number of passengers \*/

#define DATE 3

// constants for printing details in function get\_name();

#define NAME 1 /\* number for name \*/

#define SURNAME 2 /\* number for surname \*/

#define COUNTRY 3 /\* number for country \*/

#define CITY 4 /\* number for city \*/

#define ADDRESS 5 /\* number for address \*/

// constants for printing details in function get\_Int();

#define PASSDIGITS 9 /\* how many digits a passport number must have \*/

#define MOBDIGITS 11 /\* how many digits a mobile number must have \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Sturcture for passenger informations

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#### static struct Passenger

{

char name[MAXNAME]; // Passenger's name

char surname[MAXNAME]; // Passenger's surname

char title[MAXTITLE]; // Passenger's title

char country[MAXNAME]; // Passenger's country

char city[MAXNAME]; // Passenger's city

char homeAddress[MAXADDRESS]; // Passenger's home address

short DOB[DATE]; // Passenger's date of birth

short reservations[MAXPASSENGERS]; // Link to flights regarding reservations

unsigned int passportNo; // Passenger's passpoert number

unsigned long long int mobNo; // Passenger's mobile number

}passenger[MAXPASSENGERS];

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Functions prototypes

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

unsigned long long int get\_int(short n); // function to get integer for any purpose

short get\_input(short nMIN, short nMAX); // function to get input from user

void get\_Title(char arr[]); // function to choose title(Mr.,Dr., ...)

void get\_date(short arr[], short minYear, short maxYear, short n); // function to get date in format dd/mm/yyyy

void get\_Name(char arr[], short n, short maxN); // function to get name( of passenger, country, city, address...)

void normalize\_name(char arr[]); // function to normalize a name. e.g. lAZaR -> Lazar

short createNewPassenger(struct Passenger \*passenger, short n); // function to create a new passenger

void modifyPassenger(struct Passenger \*passenger); // function to modify an existing passenger

short listPassengers(const struct Passenger \*passenger);

void show\_existingPassengers(const struct Passenger \*passenger); // function to show all existing passenger's informations

void show\_PassengerInfo(short n); // function to show a particular passenger's informations

short openPassengerFile(char \*mode); // function for file opening

unsigned int checkPass(short chosenP, short count, const struct Passenger \*passenger);

void menu(void); // function to print main menu

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#endif

### Passenger.c

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <ctype.h>

#include "passenger.h"

#include "scheduledFlight.h"

#include "reservation.h"

FILE \* fpassengers; // create poiner to a file

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Function gets a name for anything,

\* such as passenger name, surname, country, city, etc.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#### void get\_Name(char arr[], short n, short maxN)

{

// depending on number provided as argument, choose what to print

switch (n)

{

case 0:

case 1: puts("------------------------------------------------------------------------\n"

"Please enter passenger's name:"

"\n------------------------------------------------------------------------");

break;

case 2: puts("------------------------------------------------------------------------\n"

"Please enter passenger's surname:"

"\n------------------------------------------------------------------------");

break;

case 3: puts("------------------------------------------------------------------------\n"

"Please enter passenger's country of birth:"

"\n------------------------------------------------------------------------");

break;

case 4: puts("------------------------------------------------------------------------\n"

"Please enter passenger's city of birth:"

"\n------------------------------------------------------------------------");

break;

case 5: puts("------------------------------------------------------------------------\n"

"Please enter passenger's home address:"

"\n------------------------------------------------------------------------");

break;

case 6: puts("------------------------------------------------------------------------\n"

"Please enter the name of Airline Company:"

"\n------------------------------------------------------------------------");

break;

case 7: puts("------------------------------------------------------------------------\n"

"Please enter the name of departure city:"

"\n------------------------------------------------------------------------");

break;

case 8: puts("------------------------------------------------------------------------\n"

"Please enter the name of arrival city:"

"\n------------------------------------------------------------------------");

break;

}

char ch; // char for user input

short i = 0; // counter for array

fflush(stdin); // just to prevent any possible error

// while user input is not new line and

// current counter is less than number of elements allowed in array

do

{

ch = getchar();

if(isalpha(ch) || (ch == ' ' && i > 0)) // check whether input is an alphabetic letter or space

{

arr[i] = ch; // if yes, put it in array

i++; // increment i for next element in array

}

else if(ch != '\n' || (i == 0 && n != 1 && n != 6)) // if input is not a new line or name of passenger or airline company

{

while (getchar() != '\n') //clear input line

continue;

// print error

puts("Invalid input, please use only alphabetic letters:");

i = 0; // reset counter for array to zero

arr[0]= '\n'; // put new line at first array element to repeat the loop

}

// while enter is not pressed and counter + 1 is less than number of characters allowed, or if first array element is new line

}while((ch != '\n' && (i + 1) < maxN) || arr[0] == '\n');

fflush(stdin); // empty buffer, to prevent any further errors ...

arr[i] = '\0'; // add \0 at the end of string

normalize\_name(arr);// normalize name

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Function to normalize a name,

\* example: lAZAR -> Lazar, or nAtAlIjA -> Natalija

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void normalize\_name(char arr[]) // Function takes array of name as argument

{

short i = 0; // index for array

for( i = 0; i < strlen(arr); i++) // loop trough array

{

if(i == 0) // if first element

{

arr[i] = toupper(arr[i]); // make a letter capital

}

else if (arr[i] == ' ') // if a element is space

{

i++; // increment index

arr[i] = toupper(arr[i]); // make letter next to space capital

}

else arr[i] = tolower(arr[i]); // else, make a letter small

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Function let us choose a title for us

\* example: Mr. , Dr. etc

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void get\_Title(char arr[]) // Function takes array as argument

{

// print options

puts("------------------------------------------------------------------------\n"

"Choose title:\n1. Mr.\t2. Ms\t3. Mrs.\t4. Dr.\t5. Prof.");

short option = get\_input(1, 5);

// depending on number entered, choose wich title to assing to a passenger

// using strncpy to copy a string (title) to array which holds it

switch(option)

{

case 1: strncpy(arr, "Mr.", 4); break;

case 2: strncpy(arr, "Ms.", 4); break;

case 3: strncpy(arr, "Mrs.", 5); break;

case 4: strncpy(arr, "Dr.", 4); break;

case 5: strncpy(arr, "Prof.", 6); break;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Function returns integer

\* depending on how many digits are required example for: mobile number, passport number

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

unsigned long long int get\_int(short n) //function takes a number as argument which says how many numbers function must return

{

//declarations of variables

unsigned long long int num, numDigits; // to hold a integer

short count; // to hold number of digits

// a number that is passed to n is defined in passenger.h or scheduledFlight.h, PASSDIGITS, MOBDIGITS or

switch (n)

{

case 3: puts("------------------------------------------------------------------------\n"

"Please enter flight ID: ");

break;

case 9: puts("------------------------------------------------------------------------\n"

"Please enter passenger's passport number: ");

break;

case 11: puts("------------------------------------------------------------------------\n"

"Please enter passenger's mobile number:\nE.g 35699123456,");

break;

}

// loop trough while requirements is not met

// in other words,

// do this while number entered does not have required number of digits

do

{

printf("Please provide a number with %hd digits: ", n);

count = 0; // count initialized to 0

fflush(stdin);

if(scanf("%llu", &num) != 0) // get input from user, if input is number, proceed

{

numDigits = num; // store number to temporary variable

while(numDigits) // while numDigits is true ( greater than 0)

{

numDigits /= 10; // truncate last digit

count++; // increment counter ( number of digits)

}

// if counter is the same as number provided as argument or if it time, minutes and hours

if(count == n)

return num; // return integer from input

// else print this error

printf("%llu does not have %hd digits and it cannot start with 0 nor contain letters\n", num, n);

}

}while(1); // will loop trough, wile number's digits from input is not same as required

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Function to create new passenger

\* storing them in array, while each passenger has all information

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#### short createNewPassenger(struct Passenger \*passenger, short n)

{

short count = 0, i, noNew = 1; // inicializing counter to 0, and declaring counter i

short index, filecount; // declaring variables

long long temp; // declaring temp variable for user input regarding passport number for checking purpose

short size = sizeof (struct Passenger); // getting size of struct Passenger

if(!openPassengerFile("a+b")) return; // open file of passengers

// while there are elements in file, count them

while (fread(&passenger[count], size, 1, fpassengers) == 1)

{

count++; // increment counter

}

// if count is the same as the allowed number of passengers

if (count == MAXPASSENGERS)

{

fputs("The passenger.dat file is full.\n", stderr); // show error

system("pause"); // pausing screen to be more readable

fclose(fpassengers); // close file

return; // exit function

}

filecount = count; // inicialize filecount to number of counts

index = count; // store count to index for later use

// if there is space for new elements, proceed

puts("Please provide passenger's details:");

puts("Press [enter] at the start of a line to stop.");

// get first name for while loop

get\_Name(passenger[count].name, NAME, MAXNAME);

while (count < MAXPASSENGERS && passenger[count].name != NULL && passenger[count].name[0] != '\0')

{

get\_Name(passenger[count].surname, SURNAME, MAXNAME);

get\_Title(passenger[count].title); //

get\_Name(passenger[count].country, COUNTRY, MAXNAME); ////

get\_Name(passenger[count].city, CITY, MAXNAME); //////

get\_Name(passenger[count].homeAddress, ADDRESS, MAXADDRESS);

get\_date(passenger[count].DOB, 1900, 2012, 1); //////

passenger[count].mobNo = get\_int(MOBDIGITS); ////

passenger[count].passportNo = checkPass(-5, count, passenger); //

for( i = 0; i < MAXPASSENGERS; i++) // using MAXPASSENGERS(100) in order to avoid redundancy

{

passenger[count].reservations[i] = -1;

}

count++;

if (count < MAXPASSENGERS && !n)

{

system("cls");

puts("Enter the next name or press [ENTER] to exit:");

get\_Name(passenger[count].name, NAME, MAXNAME);

}

else

{

system("cls");

puts("\n\n\n---------------------------------------------------------------\n"

"File is full, you cannot add more passengers!");

}

}

if (count - filecount > 0)

{

system("cls");

if((count - filecount) == 1)

puts("Here is new passenger:");

else

puts("Here are new passengers:");

for (index; index < count; index++)

show\_PassengerInfo(index);

fwrite(&passenger[filecount], size, count - filecount, fpassengers);

}

else

{

puts("No new passengers registered.\n");

noNew = 0;

}

fclose(fpassengers);

return noNew;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Function to display all passengers in the file

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#### void show\_existingPassengers(const struct Passenger \*passenger)

{

short count = 0; // inilialize counter to 0

short size = sizeof (struct Passenger); // get seize of structure passenger

if(!openPassengerFile("rb")) return; // open a file

rewind(fpassengers); // go back to the beggining of a file

while (fread(&passenger[count], size, 1, fpassengers)){ // while there is data to be read

if (count == 0) // display message only once

puts("Current registered passengers:");

show\_PassengerInfo(count); // function which shows passenger info with index count

count++; // increment counter

}

if(count == 0) puts("File is empty."); // if file is empty, print message

fclose(fpassengers); // when done, close file

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Function to display a particular passenger info

\* as argument, it takes a number for index of passenger

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#### void show\_PassengerInfo(short n)

{ printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\nPassenger Info:\nName and surname: \t%s %s %s\nDate of birth: \t\t%-0.2hd.%-0.2hd.%hd\nPassport No. \t\t%d\n Country: \t\t%s\nCity: \t\t\t%s\nHome address: \t\t%s\nMobile number: \t\t+%llu\n \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n"

, passenger[n].title, passenger[n].name

, passenger[n].surname, passenger[n].DOB[0]

, passenger[n].DOB[1], passenger[n].DOB[2]

, passenger[n].passportNo, passenger[n].country

, passenger[n].city, passenger[n].homeAddress

, passenger[n].mobNo);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

\* Function to open the file where passangers' informations are stored

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

short openPassengerFile(char \*mode)

{

if ((fpassengers = fopen("passengers.dat", mode)) == NULL) { // if cannot open a file

fputs("Cannot open passengers.dat file.\n",stderr); // display errorr

return 0;

}

else return 1;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

\* Function to show user interface main menu

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#### void menu(void)

{

short option; // variable to store user input

// printing main menu options

do

{

iniPassRes(passenger, scheduledFlight);

printf("\*--------------------WELCOME-TO-AIRLINE-RESERVATION-SYSTEM--------------------\*\n\n");

printf("\*-----------------------------------Flights-----------------------------------\*\n\n");

printf("\t0. Create\t1. Modify\t2. Cancel\t3. Show All\n\n");

printf("\*----------------------------------Passenger----------------------------------\*\n\n");

printf("\t4. Create 5. Modify 6. Show All 7. View Reservation History\n\n");

printf("\*-------------------------------Reservation-for-------------------------------\*\n\n");

printf("\t8. New Passenger\t9. Existing passenger\n\n");

printf("\*-----------------------------------------------------------------------------\*\n\n");

printf("-1. Exit\n\n");

printf("Your input: ");

option = get\_input(-1, 9);

system("cls"); // clear screen

switch(option) // depending on input,

{

case 0: createFlight(scheduledFlight); break; // create new scheduled flight(s)

case 1: modifyFlight(scheduledFlight); break; // show all existing scheduled flights

case 2: cancelFlight(scheduledFlight); break; // cancel scheduled flight

case 3: show\_existingFlights(scheduledFlight); break; // show all scheduled flights

case 4: createNewPassenger(passenger, 0); break; // create new passenger(s)

case 5: modifyPassenger(passenger); break; // show all passengers

case 6: show\_existingPassengers(passenger); break; // modify passenger

case 7: show\_reservations(passenger, scheduledFlight);break; // show passenger's past reservations

case 8: makeReservationNewP(passenger, scheduledFlight); break; // make new passenger/s and make reservations for them

case 9: makeReservationExistingP(passenger, scheduledFlight, 0, 0); break; // make reservation for existing passenger

case -1: puts("Thank you for using our airline ticket reservation system."); break; // exit the program

default: puts("Invalid input, please provide a number:"); break; // invalid input

}

system("pause"); // pause application

system("cls"); // clear screen

}while(option != -1); // while input is not zero

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Function to modify chosen passenger, all information or specific one

\* such as, user can choose only name of passenger to edit or to edit all informations

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#### void modifyPassenger(struct Passenger \*passenger)

{

short optionP, optionE, count; // optionP is to choose passenger which to edit, and optionE is to choose what to edit

if(!openPassengerFile("r+b")) return; // open a file for editing

fflush(stdin); // unespected behaviour prevention

if((count = listPassengers(passenger)) != -1 && (optionP = get\_input(-1, count)) != -1)

{ system("cls"); // clear screen to be more readable

puts("Passenger chosen for editing:");

show\_PassengerInfo(optionP); // show passenger chosen to be edited

printf("\nPlease choose detail(s) you want to edit, or -1 to exit:\n0. All\n1. Name\n2. Surname\n3. Title\n"

"4. Country\n5. City\n6. Home address\n7. Passport number\n8. Mobile number\n9. Date of birth\nChoose option: "); // print options

if((optionE = get\_input(-1, 9)) != -1)

{ // depenting on number entered

switch(optionE)

{

case 0: get\_Name(passenger[optionP].name, 0, MAXNAME); // edit everything

get\_Name(passenger[optionP].surname, SURNAME, MAXNAME);

get\_Title(passenger[optionP].title);

get\_Name(passenger[optionP].country, COUNTRY, MAXNAME);

get\_Name(passenger[optionP].city, CITY, MAXNAME);

get\_Name(passenger[optionP].homeAddress, ADDRESS, MAXADDRESS);

get\_date(passenger[optionP].DOB, 1900, 2012, 1);

passenger[optionP].passportNo = checkPass(optionP, count, passenger);

passenger[optionP].mobNo = get\_int(MOBDIGITS);

break;

case 1: get\_Name(passenger[optionP].name, 0, MAXNAME); break; // edit only

case 2: get\_Name(passenger[optionP].surname, SURNAME, MAXNAME);break; // edit only surname

case 3: get\_Title(passenger[optionP].title); break; // edit only title

case 4: get\_Name(passenger[optionP].country, COUNTRY, MAXNAME); break; // edit only country

case 5: get\_Name(passenger[optionP].city, CITY, MAXNAME); break; // edit only city

case 6: get\_Name(passenger[optionP].homeAddress, ADDRESS, MAXADDRESS); break; // edit only address

case 7: passenger[optionP].passportNo = checkPass(optionP, count, passenger); break; // edit only passport number

case 8: passenger[optionP].mobNo = get\_int(MOBDIGITS); break; // edit only mobile number

case 9: get\_date(passenger[optionP].DOB, 1900, 2012, 1); break; // edit only date of birth

}

short size = sizeof (struct Passenger); // get seize of structure Passenger

fseek(fpassengers, size \* optionP, SEEK\_SET); // find position of record with chosen passenger

fwrite(&passenger[optionP], size, 1, fpassengers); // write there edited data

system("cls"); // clear screen for to be more readable

puts("Here is edited passenger: "); // print message

show\_PassengerInfo(optionP); // print details of edited passenger

}

}

fclose(fpassengers); // when done, close file

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Function to check whether a passport number exists or not

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#### unsigned int checkPass(short chosenP, short count, const struct Passenger \*passenger)

{

unsigned int temp;

short i;

// input valitaion, checking whether passenger with unputed passport number exist or not.

do

{

temp = get\_int(PASSDIGITS); // getting user input

for (i = 0; i < count; i++)

{

if(chosenP == i)

i++;

if(passenger[i].passportNo == temp) // comparting passport number with user input

{

i = -1;

printf("Passenger with passport number %lld already exists.\n", temp);

break; // if exist, exit loop

}

}

}while(i == -1); // while entered passport number already exist

return temp;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Function to go through passengers, list them and count them

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#### short listPassengers(const struct Passenger \*passenger)

{

short count = 0;

short size = sizeof(struct Passenger);

while (fread(&passenger[count], size, 1, fpassengers) == 1) // while there is data to be read

{ // read from file and store in struct

if (count == 0) // display message only once

puts("Current passengers:\n");

printf("%d. %s %s %s\n", count, passenger[count].title,

passenger[count].name, passenger[count].surname); // show passengers

count++; // increment counter

}

if(count == 0) // if file is empty

{

fclose(fpassengers); // if empty, close it and print message

puts("passengers.dat file is empty.");

return -1; // and exit from function

}

printf("\nPlease provide one number from above, or -1 to exit: ");

return --count; // return number of passengers in a file

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Function to get user input, argument indicates the bigest number that user can enter

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#### void get\_date(short arr[], short minYear, short maxYear, short n)

{

short mm[] = {31,28,31,30,31,30,31,31,30,31,30,31};

if(n) puts("------------------------------------------------------------------------\n"

"Please provide passenger's date of birth");

else puts("------------------------------------------------------------------------\n"

"Please provide date of flight departure");

printf("Please enter year: ");

if((arr[2] = get\_input(minYear, maxYear)) % 4 == 0) mm[1] = 29;

printf("Please enter month: ");

arr[1] = get\_input(1, 12);

printf("Please enter day: ");

arr[0] = get\_input(1, (mm[arr[1] - 1]));

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Function to get user input, argument indicates the bigest number that user can enter

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#### short get\_input(short nMIN, short nMAX)

{

short option;

do

{

if(scanf("%hd", &option) != 0) // get input from user

{

if(option >= nMIN && option <= nMAX) //if number is from 1 to 5

break; //exit the loop

}

// if input is invalid, show these errors depending on numbers provided as argument

if( nMAX == 0 ) printf("Invalid input, please enter a number 0 or -1 to exit: "); // if max is 0

else if ( nMIN == -1 ) printf("Invalid input, please enter a number between 0 and %hd, or -1 to exit: ", nMAX); // if min is -1

else printf("Invalid input, please enter a number between %hd and %hd: ", nMIN, nMAX); // if mn is not -1 and max is not 0

fflush(stdin); // empty buffer

}while (1); // while input is not in the range

return option; // return input number

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

## Scheduled flight

### Flight.h

#ifndef SCHEDULEDFLIGHT\_H\_INCLUDED

#define SCHEDULEDFLIGHT\_H\_INCLUDED

// constants for printing details in function get\_Int();

#define AIRNAME 6

#define FROM 7

#define TO 8

#define FLIGHTID 3

#define STATUS 10

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Sturcture for flight schedules informations

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#### static struct Flight

{

short flightID;

short departureDate[DATE];

short timeDeparture;

short timeArrival;

short availableSeats;

char airline[MAXNAME];

char from[MAXNAME];

char to[MAXNAME];

char status[STATUS];

}scheduledFlight[MAXPASSENGERS];

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Functions prototypes

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void createFlight(struct Flight \*scheduledFlight);

void modifyFlight(struct Flight \*scheduledFlight);

void cancelFlight(struct Flight \*scheduledFlight);

void show\_existingFlights(const struct Flight \*scheduledFlight);

void show\_FlightInfo(short n, const struct Flight \*scheduledFlight);

short listFlights(const struct Flight \*scheduledFlight);

short openfscFlightFile(char \*mode);

short checkID(short flight, short count, const struct Flight \*scheduledFlight);

short get\_Time(short n); /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#endif // SCHEDULEDFLIGHT\_H\_INCLUDED

### Flight.c

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "passenger.h"

#include "scheduledFlight.h"

FILE \* fscFlight;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Function creates new scheduled flight

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#### void createFlight(struct Flight \*scheduledFlight)

{

short count = 0, i; // inicializing counter to 0

short index, filecount; // declaring variables

short temp;

short size = sizeof (struct Flight); // getting size of struct scheduledFlight

if(!openfscFlightFile("a+b")) return;// open a file for editing purpose

// while there are elements in file, count them

while (fread(&scheduledFlight[count], size, 1, fscFlight) == 1)

{

count++; // increment counter

}

// if count is the same as the allowed number of passengers

if (count == MAXPASSENGERS)

{

fputs("The fscFlight.dat file is full.\n", stderr); // show error

system("pause");

fclose(fscFlight); // close file

return; // exit function

}

filecount = count; // inicialize filecount to number of counts

index = count; // store count to index for later use

// if there is space for new elements, pro

puts("Please provide flight details:");

puts("Press [enter] at the start of a line to stop.");

// get first name for while loop

get\_Name(scheduledFlight[count].airline, AIRNAME, MAXNAME);

while (count < MAXPASSENGERS && scheduledFlight[count].airline != NULL && scheduledFlight[count].airline[0] != '\0')

{

scheduledFlight[count].flightID = checkID(-5, count, scheduledFlight); // when input is valid, store it in struct member flightID

get\_Name(scheduledFlight[count].from, FROM, MAXNAME); // get name from where a flight is taking place

get\_Name(scheduledFlight[count].to, TO, MAXNAME); // get name to where a flight is taking place

get\_date(scheduledFlight[count].departureDate, 2014, 2050, 0); // get date of flight departure

scheduledFlight[count].timeDeparture = get\_Time(1); // get time whene a plane will take off ( 1 is indicator what print, departure)

scheduledFlight[count].timeArrival = get\_Time(0); // get time when a plane will land ( 0 is indicator what print, arrival)

strncpy(scheduledFlight[count].status, "OK", 4); // set status of flight to OK(still not cancelled).

scheduledFlight[count].availableSeats = 50; // initialize avaliable seats to 50

count++; // increment counter

if (count < MAXPASSENGERS) // if maximum number of allowed flights is not reached

{

system("cls");

puts("Please provide next flight details, or press [ENTER] to finish:");

get\_Name(scheduledFlight[count].airline, AIRNAME, MAXNAME); // get neext name of airline company

}

else

{

system("cls");

puts("\n\n\n---------------------------------------------------------------\n"

"File is full, you cannot add more scheduled flights!");

}

}

if (count - filecount > 0) // if at least 1 new flight is registered

{

system("cls");

if((count - filecount) == 1) // if only one is made

puts("Here is new scheduled flight:");

else // if more than one

puts("Here are new scheduled flights:");

// show new scheduled flight(s)

for (index; index < count; index++) // index is starting positions of (last flight + 1) in record

show\_FlightInfo(index, scheduledFlight);

// write them in a file

fwrite(&scheduledFlight[filecount], size, count - filecount, fscFlight);

}

else

puts("No scheduled flights.\n"); // if no new flights

fclose(fscFlight);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

\* Function to to modify chosen passenger, all information or specific one

\* such as, user can choose only name of passenger to edit or to edit all informations

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#### void modifyFlight(struct Flight \*scheduledFlight)

{

short count; // inilialize counter to 0

short optionP, optionE; // optionP is to choose passenger which to edit, and optionE is to choose what to edit

short size = sizeof (struct Flight); // get seize of structure Passenger

if(!openfscFlightFile("r+b")) return; // open a file for editing

if((count = listFlights(scheduledFlight)) != -1)// list all flights and store their number in count

{

fflush(stdin); // unespected behaviour prevention

if((optionP = get\_input(-1, count)) != -1)

{

system("cls"); // clear screen to be more readable

puts("Scheduled flight chosen for editing:");

show\_FlightInfo(optionP, scheduledFlight); // show passenger chosen to be edited

printf("\nPlease choose detail(s) you want to edit, or -1 to exit:\n0. All\n1. Airline name\n2. Flight ID\n3. Departure city\n"

"4. Arrival city\n5. Departure date\n6. Departure Time\n7. Arrival TIme\nChoose option: "); // print options

if((optionE = get\_input(-1, 7)) != -1)

{

// depenting on number entered

switch(optionE)

{

case 0: get\_Name(scheduledFlight[optionP].airline, AIRNAME, MAXNAME); // edit everything

scheduledFlight[optionP].flightID = checkID(optionP, count, scheduledFlight);

get\_Name(scheduledFlight[optionP].from, FROM, MAXNAME);

get\_Name(scheduledFlight[optionP].to, TO, MAXNAME);

get\_date(scheduledFlight[optionP].departureDate, 2014, 2050, 0);

scheduledFlight[optionP].timeDeparture = get\_Time(1);

scheduledFlight[optionP].timeArrival = get\_Time(0);

break;

case 1: get\_Name(scheduledFlight[optionP].airline, AIRNAME, MAXNAME); break; // edit only airline name

case 2: scheduledFlight[optionP].flightID = checkID(optionP, count, scheduledFlight); break; // edit only flighID

case 3: get\_Name(scheduledFlight[optionP].from, FROM, MAXNAME); break; // edit only departure city

case 4: get\_Name(scheduledFlight[optionP].to, TO, MAXNAME); break; // edit only arrival city

case 5: get\_date(scheduledFlight[optionP].departureDate, 2014, 2050, 0); break; // edit only date of flight departure

case 6: scheduledFlight[optionP].timeDeparture = get\_Time(1); break; // edit only time of departure

case 7: scheduledFlight[optionP].timeArrival = get\_Time(0); break; // edit only time of arrival

}

fseek(fscFlight, size \* optionP, SEEK\_SET); // find position of record with chosen passenger

fwrite(&scheduledFlight[optionP], size, 1, fscFlight); // write there edited data

system("cls"); // clear screen for to be more readable

puts("Here is edited scheduled flight: "); // print message

show\_FlightInfo(optionP, scheduledFlight); // print details of edited passenger

} // end of inner if

} // end of outer if

}

fclose(fscFlight); // when done, close file

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Function to cancel chosen flight

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#### void cancelFlight(struct Flight \*scheduledFlight)

{

short count = 0; // inilialize counter to 0

short optionF, optionE; // optionF is to choose which flight to cancel, opetionE is for yes/no confirmation

if(!openfscFlightFile("r+b")) return;; // open a file for editing

if((count = listFlights(scheduledFlight)) != -1) // list scheduled flights and store their number in count

{

fflush(stdin); // unespected behaviour prevention

printf("\nPlease provide one number from above , or -1 to exit: ");

do

{

if((optionF = get\_input(-1, count)) != -1) // if input is not -1

if(strcmp(scheduledFlight[optionF].status, " CANCELLED ") != 0) // and if chosen flight is not already cancelled

{

system("cls"); // clear screen to be more readable

puts("Scheduled flight chosen for cancelation:");

show\_FlightInfo(optionF, scheduledFlight); // show flight chosen to be cancelled

puts("Are you sure you want to cancel this scheduled flight?\n1. Yes\n0. No");

if((optionE = get\_input(0, 1)) == 1) // if input is 0

{

strncpy(scheduledFlight[optionF].status, "CANCELLED", STATUS); // cancel flight

short size = sizeof (struct Flight); // get seize of structure Passenger

fseek(fscFlight, size \* optionF, SEEK\_SET); // find position of record with chosen passenger

fwrite(&scheduledFlight[optionF], size, 1, fscFlight); // write there edited data

system("cls"); // clear screen for to be more readable

puts("Here is cancelled scheduled flight: "); // print message

show\_FlightInfo(optionF, scheduledFlight); // print details of edited passenger

break;

}

else break;

}

else puts("Flight is already cancelled. Choose another or -1 to exit.");

}while(optionF != -1); // while input is not -1

}

fclose(fscFlight); // when done, close file

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Function to display all scheduled flights in the file

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#### void show\_existingFlights(const struct Flight \*scheduledFlight)

{

short count = 0; // inilialize counter to 0

short size = sizeof (struct Flight); // get seize of structure passenger

if(!openfscFlightFile("rb")) return; // open a file for reading

rewind(fscFlight); // go back to the beggining of a file

while (fread(&scheduledFlight[count], size, 1, fscFlight)) // while there is data to be read

{

if (count == 0) // display message only once

puts("Current scheduled flights:");

show\_FlightInfo(count, scheduledFlight); // function which shows passenger info with index count

count++; // increment counter

}

if(count == 0) // if file is empty, print message

puts("File is empty.");

fclose(fscFlight); // when done, close file

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Function to display a particular scheduled flight info

\* as argument, it takes an number for index of scheduled flight

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#### void show\_FlightInfo(short n, const struct Flight \*scheduledFlight)

{

// print info

printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\nScheduled flights info:"

"\nScheduled flight ID: \t%d\nAirline name: \t\t%s\nFrom: \t\t\t%s\nTo: \t\t\t%s\n"

"Departure date: \t%-0.2hd/%-0.2hd/%hd\nDeparture time: \t%-0.4hd\nArrival time: \t\t%-0.4hd\n"

"Available seats: \t%d\nStatus:\t\t\t%s\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n"

, scheduledFlight[n].flightID, scheduledFlight[n].airline

, scheduledFlight[n].from, scheduledFlight[n].to

, scheduledFlight[n].departureDate[0], scheduledFlight[n].departureDate[1]

, scheduledFlight[n].departureDate[2], scheduledFlight[n].timeDeparture

, scheduledFlight[n].timeArrival, scheduledFlight[n].availableSeats, scheduledFlight[n].status);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Function to display all scheduled flights and to return how many are in the file

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#### short listFlights(const struct Flight \*scheduledFlight)

{

short count = 0;

short size = sizeof(struct Flight);

while (fread(&scheduledFlight[count], size, 1, fscFlight) == 1) // while there is data to be read

{

if (count == 0) // display message only once

puts("Current scheduled flights:\n\nNo. FlightID Airline \t\tFrom\t\tTo \t\tStatus\t Seats\n");

printf("%hd. %-8hd| %-16s|\t%-14s|\t%-14s|\t%-8s| %-4hd|\n", count, scheduledFlight[count].flightID, scheduledFlight[count].airline,

scheduledFlight[count].from, scheduledFlight[count].to

, scheduledFlight[count].status, scheduledFlight[count].availableSeats);// function which shows passenger info with index count

count++; // increment counter

}

if(count == 0) // if file is empty

{

fclose(fscFlight); // if empty, close it and print message

puts("fscFlight.dat file is empty.");

return -1; // and exit from function

}

printf("\nPlease provide one number from above , or -1 to exit: ");

return --count;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Function to get time in 24h format

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#### short get\_Time(short n)

{

short hour, min;

if (n)

puts("------------------------------------------------------------------------\n"

"Please provide time of flight departure:");

else

puts("------------------------------------------------------------------------\n"

"Please provide time of flight arrival:");

printf("Please provide hours: ");

hour = get\_input(0, 23);

printf("Please provide minutes: ");

min = get\_input(0,59);

return hour\*100 + min;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Function to check if flightID already exist

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#### short checkID(short flight, short count, const struct Flight \*scheduledFlight)

{

short temp, i;

// checking whether flight with entered ID already exist or not

do

{

temp = get\_int(FLIGHTID); // getting user input

for (i = 0; i <= count; i++)

{

if(i == flight)

i++;

if(scheduledFlight[i].flightID == temp) // checking

{

i = -1; // for purpose to repeat the loop

printf("Flight with flight ID %d already exists. Please provide another fligh ID:\n", temp);

break; // exit for loop

}

}

}while(i == -1); // if flight ID exist, repeat process

return temp;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Function to open the file where scheduled flights' informations are stored

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#### short openfscFlightFile(char \*mode)

{

if ((fscFlight = fopen("fscFlight.dat", mode)) == NULL) // if cannot open a file

{

fputs("Cannot open fscFlight.dat file.\n",stderr); // display errorr

return 0;

}

else return 1;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

## Reservation

### Reservation.h

#ifndef RESERVATION\_H\_INCLUDED

#define RESERVATION\_H\_INCLUDED

void makeReservationExistingP(struct Passenger \*passenger, struct Flight \*scheduledFlight, short n, short newPass);

void makeReservationNewP(struct Passenger \*passenger, struct Flight \*scheduledFlight);

void show\_reservations(const struct Passenger \*passenger, const struct Flight \*flight);

short findFreeElement(struct Passenger \*passenger, short n);

#endif // RESERVATION\_H\_INCLUDED

### Reservatoin.c

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "passenger.h"

#include "scheduledFlight.h"

#include "reservation.h"

FILE \* fpassengers;

FILE \* fscFlight;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Function to make a reservation

\* by choosing a passenger and scheduled flight

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#### void makeReservationExistingP(struct Passenger \*passenger, struct Flight \*scheduledFlight, short n, short newPass)

{

short count; // declare variable

short optionP, countF, optionR; // opetionP stores user's choise of passenger, countF to store number of records in scheduled flight file, opetionR store user's choise of flight

fflush(stdin); // unespected behaviour prevention

// if passenger and flight file file can be opened, and there is at least one passenger in file and user input is not -1

// there is at least one schedule flight in a file, proceed

if(openPassengerFile("r+b") && openfscFlightFile("r+b"))

if( n || ((count = listPassengers(passenger)) != -1 && (optionP = get\_input(-1, count)) != -1))

if((countF = listFlights(scheduledFlight)) != -1)

{

short ok = 0; // when it is 1, loop will stop

if(n) optionP = newPass; // if n is 1(which indicates that reservation for new passenger is being made) optionP takes value of passengerIndex

do

{

printf("\nYour input: ");

// get user input and proceed if not -1

if((optionR = get\_input(-1, countF)) == -1) break;

else

{

// if choosen flight is cancelled, reservation cannot be made

if(strcmp(scheduledFlight[optionR].status, "CANCELLED" ) == 0)

puts("This flight is cancelled and you cannot make a reservation for it.\n"

"Choose another or -1 to exit.");

else if(scheduledFlight[optionR].availableSeats < 1)

puts("There are no more available seats in this flight.\nChoose another or -1 to exit.");

else

{

// if not, check whether a reservation on that flight is already being made

for(count = 0; count < MAXPASSENGERS; count++)

{

if(passenger[optionP].reservations[count] == optionR)

{

puts("You already have a reservation for this flight.\n"

"Choose another or -1 to exit");

break;

}

else if(passenger[optionP].reservations[count] == -1) // if free element is reached, make reservation

{

ok = 1;

// store chosen flight to a location where is first free element found

passenger[optionP].reservations[findFreeElement(passenger, optionP)] = optionR;

scheduledFlight[optionR].availableSeats--;

system("cls");

//show reservation

puts("Your reservation:");

show\_FlightInfo(optionR, scheduledFlight);

short sizeP = sizeof (struct Passenger); // get seize of structure Passenger

short sizeF = sizeof (struct Flight); // get seize of structure Passenger

fseek(fpassengers, sizeP \* optionP, SEEK\_SET); // find position of record with chosen passenger

fwrite(&passenger[optionP], sizeP, 1, fpassengers); // write there edited data

fseek(fscFlight, sizeF \* optionR, SEEK\_SET); // find position of record with chosen passenger

fwrite(&scheduledFlight[optionR], sizeF, 1, fscFlight); // write there edited data

break; // break the loop

}

}

}

}

}while(ok != 1); // while free element is not reached

} // inner if

fclose(fscFlight); // close file

fclose(fpassengers); // when done, close file

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Function to make a reservation for new passenger

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#### void makeReservationNewP(struct Passenger \*passenger, struct Flight \*scheduledFlight)

{

openPassengerFile("r+b");

short passengerIndex = listPassengers(passenger) + 1;

system("cls");

if(createNewPassenger(passenger, 1))

makeReservationExistingP(passenger, scheduledFlight, 1, passengerIndex);

fclose(fpassengers);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Function to display all passenger's past reservations

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#### void show\_reservations(const struct Passenger \*passenger, const struct Flight \*scheduledFlight)

{

short count; // declare counter

fflush(stdin);// unespected behaviour prevention

short optionP; // get user input, optionP is to choose passenger which history we want

// if files can be opened and there is at least one passenger, and one scheduled flight, proceed

if(openPassengerFile("rb") && openfscFlightFile("rb") && (count = listPassengers(passenger)) != -1 && ((optionP = get\_input(-1, count)) != -1))

{

count = 0; // reset counter

short sizeF = sizeof (struct Flight); // get size of Flight structure

while (fread(&scheduledFlight[count], sizeF, 1, fscFlight) == 1) // while there is data to be read,

count++; // read from file and store in structure

// increment counter

system("cls");

// loop trough passenger's reservations

for( count = 0; count < MAXPASSENGERS; count++)

{

if(passenger[optionP].reservations[count] == -1) // when free element(-1) is read

break; // exit loop

if(count == 0) // show message only once if there are any reservations

puts("All passenger's past reservations are: ");

show\_FlightInfo(passenger[optionP].reservations[count], scheduledFlight); // show reservation

}

if(count == 0) puts("No reservations by this passenger."); // if there are no reservations

}

fclose(fscFlight);

fclose(fpassengers);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Function loop trough array to find element -1

\* -1 indicates that there is no data written (data is reference to a schedule flight)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#### short findFreeElement(struct Passenger \*passenger, short n)

{

short i; // declare counter;

for ( i = 0; i < MAXPASSENGERS; i++) // loop trough loop

{

if(passenger[n].reservations[i] == -1) // if free element is found

break; // exit loop

}

return i; // return index of free element

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Function reinisialize reservations to -1

\* if file or content of the flight file is deleted accidentally or on purpose

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#### void iniPassRes(struct Passenger \*passenger, const struct Fligh \*scheduledFlight)

{

short count, index;

int size = sizeof(struct Passenger);

if(!openfscFlightFile("rb") || (count = listFlights(scheduledFlight) == -1)) // if file or its content is deleted

{

count = 0;

if(openPassengerFile("r+b"))

while (fread(&passenger[count], size, 1, fpassengers) == 1)

{

for(index = 0; index < MAXPASSENGERS; index++)

{

passenger[count].reservations[index] = -1; // reset reservation to -1(which indicates no reservations

if(passenger[count].reservations[index + 1] == -1) break; // if -1 is next element, break the loop

}

count++; // increment counter

}

rewind(fpassengers);

fwrite(&passenger[0], size, count, fpassengers); // write edited data

}

fclose(fscFlight);

fclose(fpassengers);

system("cls");

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/