**Flight.h**

/\* flight.h - interface to an ADT which implements a flight \*/

#ifndef FLIGHT\_H\_

#define FLIGHT\_H\_

#define NUM\_DEST\_LETTERS 3

**typedef** struct flight\_t **{**

long int flight\_id**;**

FlightType ftype**;**

char dest**[**3**];**

BOOL emergency**;**

**}**FLIGHT**,** **\***PFLIGHT**;**

BOOL isUpperCase**(**const char **\***String**);**

PFLIGHT createFlight**(**long int id **,** FlightType ftype **,** char**\*** dest**,** BOOL emergency**);**

void destroyFlight**(**PFLIGHT pFlight**);**

Result printFlight**(**PFLIGHT pFlight**);**

#endif /\* FLIGHT\_H\_ \*/

**Runway.h**

/\* runway.h - interface to an ADT which implements a runway \*/

#ifndef RUNWAY\_H\_

#define RUNWAY\_H\_

**typedef** struct node\_t **{** // linked list of flights

PFLIGHT pFlight**;**

struct node\_t**\*** next**;**

**}** NODE**;**

**typedef** struct runway\_t **{**

long int runway\_id**;**

FlightType run\_type**;**

NODE**\*** head**;** // head of the linked list //

long int num\_of\_flights**;**

**}**RUNWAY**,** **\***PRUNWAY**;**

PRUNWAY createRunway**(**long int id **,** FlightType run\_type**);**

void destroyRunway**(**PRUNWAY pRunway**);**

BOOL isFlightExists**(**PRUNWAY pRunway**,** long int id**);**

long int getFlightNum**(**PRUNWAY pRunway**);**

Result addFlight**(**PRUNWAY pRunway**,** PFLIGHT pFlight**);**

Result removeFlight**(**PRUNWAY pRunway**,** long int id**);**

Result depart**(**PRUNWAY pRunway**);**

Result printRunway**(**PRUNWAY pRunway**);**

#endif /\* RUNWAY\_H\_ \*/

**Airport.h**

/\* airport.h - interface to an ADT which implements a flight \*/

#ifndef AIRPORT\_H\_

#define AIRPORT\_H\_

**typedef** struct node\_air **{** // linked list of runways

PRUNWAY pRunway**;**

struct node\_air**\*** next**;**

**}** NODEAIR**,** **\***PNODEAIR**;**

PNODEAIR CreateAirport**();**

Result addRunway**(**long int id **,** FlightType run\_type**);**

Result removeRunway**(**long int id**);**

Result addFlightToAirport**(**long int id**,** FlightType flight\_type**,** char**\*** dest**,** BOOL emergency**);**

Result departFromRunway**(**long int id**);**

Result stormAlert**(**char **\***dest**);**

void printAirport**();**

void destroyAirport**();**

#endif /\* AIRPORT\_H\_ \*/

**Flight.c**

/\* flight.c \*/

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "ex2.h"

#include "flight.h"

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

//\* function name: isUpperCase

//\* Description : Gets a string and checks if it contaions only english upper case characters(A...Z)

//\* Parameters : String - a pointer to the string

//\* Return Value : TRUE or FALSE

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

BOOL isUpperCase**(**const char **\***String**)**

**{**

long int i**;**

long int n **=** strlen**(**String**);**

**for** **(**i **=** 0**;** i **<** n**;** i**++)**

**if** **(!(**String**[**i**]** **>=** 'A' **&&** String**[**i**]** **<=** 'Z'**))**

**return** FALSE**;**

**return** TRUE**;**

**}**

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

//\* function name: createFlight

//\* Description : Gets the flight parameters, allocs a FLIGHT struct to the flight and returns a pointer

//\* to the flight

//\* Parameters : id - flight id

//\* flight\_type - flight type(D or I)

//\* dest - destination (three upper case letters. Example: JFK)

//\* emergency - is it an emergency flight or not ( TRUE or FALSE)

//\* Return Value : pFlight - a pointer to the created struct of the flight

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

PFLIGHT createFlight**(**long int id **,** FlightType flight\_type **,** char**\*** dest**,** BOOL emergency**){**

PFLIGHT pFlight **=** **NULL;**

**if** **(**id **<=** 0 **||** id **>** MAX\_ID **||** **(**flight\_type **!=** DOMESTIC **&&** flight\_type **!=** INTERNATIONAL**)**

**||** **(**dest **==** **NULL** **||** strlen**(**dest**)** **!=** NUM\_DEST\_LETTERS **||** **!**isUpperCase**(**dest**))**

|| (emergency != TRUE && emergency != FALSE))

return pFlight; // illegal parameters checks

if ((pFlight = (PFLIGHT) malloc(sizeof(FLIGHT)))){

pFlight->flight\_id = id;

pFlight->ftype = flight\_type;

strcpy(pFlight->dest, dest);

pFlight->emergency = emergency;

}

return pFlight;

}

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

//\* function name: destroyFlight

//\* Description : Gets the pointer to a flight and frees the alloced struct for the flight

//\* Parameters : pFlight - a pointer to a flight

//\* Return Value : None

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

void destroyFlight(PFLIGHT pFlight){

free(pFlight);

}

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

//\* function name: printFlight

//\* Description : Gets the pointer to a flight prints the information on the flight. If the pointer to

//\* the flight is NULL then it will return FAILURE, otherwise SUCCESS

//\* Parameters : pFlight - a pointer to a flight

//\* Return Value : SUCCESS or FAILURE

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

Result printFlight(PFLIGHT pFlight) {

if (pFlight != NULL){

char dest\_type, flight\_urgence;

dest\_type = (pFlight->ftype == DOMESTIC) ? 'D' : 'I';

flight\_urgence =(pFlight->emergency == TRUE) ? 'E' : 'R';

printf("Flight %ld %c %s %c\n", pFlight->flight\_id, dest\_type, pFlight->dest, flight\_urgence);

return SUCCESS;

}

return FAILURE;

}

**Runway.c**

/\* runway.c \*/

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "ex2.h"

#include "flight.h"

#include "runway.h"

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

//\* function name: createRunway

//\* Description : Gets the runway parameters, allocs a RUNWAY struct to the runway and returns a pointer

//\* to the runway

//\* Parameters : id - runway id

//\* run\_type - runway type(D or I)

//\* Return Value : pRunway - a pointer to the created struct of the runway

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

PRUNWAY createRunway**(**long int id **,** FlightType run\_type**){**

PRUNWAY pRunway **=** **NULL;**

**if** **(**id **<=** 0 **||** id **>** MAX\_ID**)** // illegal parameters check

**return** pRunway**;**

**if** **((**pRunway **=** **(**PRUNWAY**)** malloc**(sizeof(**RUNWAY**)))){**

pRunway**->** runway\_id **=** id**;**

pRunway**->** run\_type **=** run\_type**;**

pRunway**->** num\_of\_flights **=** 0**;**

pRunway**->** head **=** **NULL;**

**}**

**return** pRunway**;**

**}**

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

//\* function name: destroyRunway

//\* Description : Gets the pointer to a runway and frees the alloced struct for the runway

//\* Parameters : pRunway - a pointer to a runway

//\* Return Value : None

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

void destroyRunway**(**PRUNWAY pRunway**){**

**if** **(**pRunway **==** **NULL)**

return; // pointer is empty

NODE\* tmp;

tmp = pRunway->head;

while (tmp!=NULL ){ // destroy all COPIES of flights on the runway

NODE\* requested\_node = tmp;

destroyFlight(tmp->pFlight);

tmp = tmp->next;

free(requested\_node);

}

free(pRunway);

}

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

//\* function name: isFlightExists

//\* Description : Gets a pointer to the runway and a flight id, returns if a flight is there (TRUE or FALSE)

//\* Parameters : pRunway - a pointer to a runway

//\* id - flight id

//\* Return Value : TRUE or FALSE

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

BOOL isFlightExists**(**PRUNWAY pRunway**,** long int id**){**

**if** **(**pRunway **==** **NULL** **||** id **<=** 0 **||** id **>** MAX\_ID **||** pRunway**->**head **==** **NULL)**

**return** FALSE**;** // parameters check

NODE**\*** tmp**;**

**for** **(**tmp **=** pRunway**->**head**;**tmp**;**tmp**=**tmp**->**next**)** // check all flights on the runway list

**if** **(**tmp**->**pFlight**->**flight\_id **==** id**)**

**return** TRUE**;** // flight found!

**return** FALSE**;** // flight not found

**}**

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

//\* function name: getFlightNum

//\* Description : Gets a pointer to the runway and returns the amount of flights in the runway

//\* Parameters : pRunway - a pointer to a runway

//\* Return Value : num\_of\_flights - number of flights

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

long int getFlightNum**(**PRUNWAY pRunway**){**

**if** **(**pRunway **==** **NULL)**

**return** **-**1**;** // illegal runway id

**return** pRunway**->**num\_of\_flights**;**

**}**

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

//\* function name: addFlight

//\* Description : Gets a pointer to a runway and a pointer to a flight, COPY the flight and enters it to

//\* the runway by the ruleset given on HW2. The function returns SUCCESS if the flight was

//\* added and FAILURE if it failed.

//\* Parameters : pRunway - runway pointer

//\* pFlight - flight pointer

//\* Return Value : Result - SUCCESS or FAILURE of adding the flight to the runway

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

Result addFlight**(**PRUNWAY pRunway**,** PFLIGHT pFlight**){**

**if** **(**pRunway **==** **NULL** **||** pFlight **==** **NULL** **)**

**return** FAILURE**;** // illegal pointers

**if** **(**pRunway**->**run\_type **!=** pFlight**->**ftype**)**

**return** FAILURE**;** // flight type isn't the same of runway type

NODE**\*** tmp**;**

tmp **=** pRunway**->**head**;**

**if** **(**isFlightExists**(**pRunway**,** pFlight**->**flight\_id**))**// flight already exists on runway

**return** FAILURE**;**

//////////////////////////////////////////

// case it's the first flight on runway //

//////////////////////////////////////////

**if** **(**pRunway**->**num\_of\_flights **==** 0 **){**

tmp **=** **(**NODE**\*)**malloc**(sizeof(**NODE**));**

tmp**->**pFlight **=** createFlight**(**pFlight**->**flight\_id **,** pFlight**->**ftype **,** pFlight**->**dest**,** pFlight**->**emergency**);**

tmp**->**next **=** **NULL;**

pRunway**->**head **=** tmp**;**

**(**pRunway**->**num\_of\_flights**)++;**

**return** SUCCESS**;**

**}**

////////////////////////////////////////////////////////////////

// case the first flight is REGULAR and entering is EMERGENCY //

////////////////////////////////////////////////////////////////

**if** **(**tmp**->**pFlight**->**emergency **==** FALSE **&&** pFlight**->**emergency **==** TRUE**)** **{**

NODE**\*** tmp\_emer**=(**NODE**\*)**malloc**(sizeof(**NODE**));**

tmp\_emer**->**pFlight **=** createFlight**(**pFlight**->**flight\_id**,** pFlight**->**ftype**,** pFlight**->**dest**,** pFlight**->**emergency**);**

tmp\_emer**->**next **=** tmp**;**

pRunway**->**head **=** tmp\_emer**;**

**(**pRunway**->**num\_of\_flights**)++;**

**return** SUCCESS**;**

**}**

////////////////// goes until the first regular flight or until list ends

**while** **(**tmp**->**next **!=** **NULL** **&&** tmp**->**next**->**pFlight**->**emergency **==** TRUE**)**

tmp**=**tmp**->**next**;**

//////////////////////////////////////////////////////////////

// case when emergency flight or no other flights on runway //

//////////////////////////////////////////////////////////////

**if** **(**pFlight**->**emergency **==** TRUE**){**

NODE**\*** current\_next **=** tmp**->**next**;**

tmp**->**next **=** **(**NODE**\*)**malloc**(sizeof(**NODE**));**

tmp**->**next**->**pFlight **=** createFlight**(**pFlight**->**flight\_id**,** pFlight**->**ftype**,** pFlight**->**dest**,** pFlight**->**emergency**);**

tmp**->**next**->**next **=** current\_next**;** // re-links to the next flight

**(**pRunway**->**num\_of\_flights**)++;**

**return** SUCCESS**;**

**}**

/////////////////////////////////////////

// case when there are regular flights //

/////////////////////////////////////////

**while** **(**tmp**->**next **!=** **NULL)**

tmp**=**tmp**->**next**;** // goes to the end of the list

tmp**->**next **=** **(**NODE**\*)**malloc**(sizeof(**NODE**));**

tmp**->**next**->**pFlight **=** createFlight**(**pFlight**->**flight\_id **,** pFlight**->**ftype **,** pFlight**->**dest**,** pFlight**->**emergency**);**

tmp**->**next**->**next **=** **NULL;**

**(**pRunway**->**num\_of\_flights**)++;**

**return** SUCCESS**;**

**}**

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

//\* function name: removeFlight

//\* Description : Gets the pointer to a runway and flight id and removes it if exist on runway

//\* Parameters : pRunway - a pointer to a runway

//\* id - flight id

//\* Return Value : Result - SUCCESSS or FAILURE in removing the flight with the given id

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

Result removeFlight(PRUNWAY pRunway, long int id){

if (pRunway == NULL)

return FAILURE; // illegal pointer

if (isFlightExists(pRunway, id) == FALSE)

return FAILURE; // flight isn't on runway

NODE\* tmp;

tmp = pRunway->head;

if (tmp->pFlight->flight\_id == id){

// case it's the first flight

destroyFlight(tmp->pFlight);

pRunway->head = tmp->next;

free(tmp);

(pRunway->num\_of\_flights)--;

return SUCCESS;

}

while (tmp->next->pFlight->flight\_id != id)

// case elsewise

tmp=tmp->next; // goes to the flight linked before to the requested flight

NODE\* requested\_node = tmp->next;

destroyFlight(requested\_node->pFlight);

tmp->next = tmp->next->next;

free(requested\_node);

(pRunway->num\_of\_flights)--;

return SUCCESS;

}

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

//\* function name: depart

//\* Description : Gets the pointer to a runway and removes the first flight

//\* Parameters : pRunway - a pointer to a runway

//\* Return Value : Result - SUCCESSS or FAILURE in removing the first flight in the runway

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

Result depart(PRUNWAY pRunway){

if (pRunway == NULL)

return FAILURE; // illegal pointer

if (pRunway->num\_of\_flights == 0)

return FAILURE; // no flights on runway

NODE\* tmp;

tmp = pRunway->head;

destroyFlight(tmp->pFlight);

pRunway->head = tmp->next;

free(tmp);

(pRunway->num\_of\_flights)--;

return SUCCESS;

}

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

//\* function name: printRunway

//\* Description : Gets the pointer to a runway and prints the data of it(id, type, num of flights waiting

//\* and flights' data)

//\* Parameters : pRunway - a pointer to a runway

//\* Return Value : Result - SUCCESSS or FAILURE in printing the runway data

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

Result printRunway(PRUNWAY pRunway){

//

if (pRunway != NULL){

printf("Runway %ld ", pRunway->runway\_id);

char run\_type;

run\_type =( pRunway->run\_type == DOMESTIC) ? 'D' : 'I';

if (run\_type == 'D')

printf("domestic\n");

else

printf("international\n");

printf("%ld flights are waiting:\n", pRunway->num\_of\_flights);

NODE\* tmp;

tmp = pRunway->head;

while (tmp!=NULL){

char flight\_urgence;

flight\_urgence =(tmp->pFlight->emergency == TRUE) ? 'E' : 'R';

printf("Flight %ld %c %s %c\n", tmp->pFlight->flight\_id, run\_type, tmp->pFlight->dest, flight\_urgence);

tmp=tmp->next;

}

return SUCCESS;

}

return FAILURE;

}

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

//\* function name: isFlightExists

//\* Description : Gets a pointer to the runway and a flight id, returns if a flight is there (TRUE or FALSE)

//\* Parameters : pRunway - a pointer to a runway

//\* id - flight id

//\* Return Value : TRUE or FALSE

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

BOOL isFlightExists(PRUNWAY pRunway, long int id){

if (pRunway == NULL || id <= 0 || id > MAX\_ID || pRunway->head == NULL)

return FALSE; // parameters check

NODE\* tmp;

for (tmp = pRunway->head;tmp;tmp=tmp->next) // check all flights on the runway list

if (tmp->pFlight->flight\_id == id)

return TRUE; // flight found!

return FALSE; // flight not found

}

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

//\* function name: getFlightNum

//\* Description : Gets a pointer to the runway and returns the amount of flights in the runway

//\* Parameters : pRunway - a pointer to a runway

//\* Return Value : num\_of\_flights - number of flights

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

long int getFlightNum(PRUNWAY pRunway){

if (pRunway == NULL)

return -1; // illegal runway id

return pRunway->num\_of\_flights;

}

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

//\* function name: addFlight

//\* Description : Gets a pointer to a runway and a pointer to a flight, COPY the flight and enters it to

//\* the runway by the ruleset given on HW2. The function returns SUCCESS if the flight was

//\* added and FAILURE if it failed.

//\* Parameters : pRunway - runway pointer

//\* pFlight - flight pointer

//\* Return Value : Result - SUCCESS or FAILURE of adding the flight to the runway

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

Result addFlight(PRUNWAY pRunway, PFLIGHT pFlight){

if (pRunway == NULL || pFlight == NULL )

return FAILURE; // illegal pointers

if (pRunway->run\_type != pFlight->ftype)

return FAILURE; // flight type isn't the same of runway type

NODE\* tmp;

tmp = pRunway->head;

if (isFlightExists(pRunway, pFlight->flight\_id))// flight already exists on runway

return FAILURE;

//////////////////////////////////////////

// case it's the first flight on runway //

//////////////////////////////////////////

if (pRunway->num\_of\_flights == 0 ){

tmp = (NODE\*)malloc(sizeof(NODE));

tmp->pFlight = createFlight(pFlight->flight\_id , pFlight->ftype , pFlight->dest, pFlight->emergency);

tmp->next = NULL;

pRunway->head = tmp;

(pRunway->num\_of\_flights)++;

return SUCCESS;

}

////////////////////////////////////////////////////////////////

// case the first flight is REGULAR and entering is EMERGENCY //

////////////////////////////////////////////////////////////////

if (tmp->pFlight->emergency == FALSE && pFlight->emergency == TRUE) {

NODE\* tmp\_emer=(NODE\*)malloc(sizeof(NODE));

tmp\_emer->pFlight = createFlight(pFlight->flight\_id, pFlight->ftype, pFlight->dest, pFlight->emergency);

tmp\_emer->next = tmp;

pRunway->head = tmp\_emer;

(pRunway->num\_of\_flights)++;

return SUCCESS;

}

////////////////// goes until the first regular flight or until list ends

while (tmp->next != NULL && tmp->next->pFlight->emergency == TRUE)

tmp=tmp->next;

//////////////////////////////////////////////////////////////

// case when emergency flight or no other flights on runway //

//////////////////////////////////////////////////////////////

if (pFlight->emergency == TRUE){

NODE\* current\_next = tmp->next;

tmp->next = (NODE\*)malloc(sizeof(NODE));

tmp->next->pFlight = createFlight(pFlight->flight\_id, pFlight->ftype, pFlight->dest, pFlight->emergency);

tmp->next->next = current\_next; // re-links to the next flight

(pRunway->num\_of\_flights)++;

return SUCCESS;

}

/////////////////////////////////////////

// case when there are regular flights //

/////////////////////////////////////////

while (tmp->next != NULL)

tmp=tmp->next; // goes to the end of the list

tmp->next = (NODE\*)malloc(sizeof(NODE));

tmp->next->pFlight = createFlight(pFlight->flight\_id , pFlight->ftype , pFlight->dest, pFlight->emergency);

tmp->next->next = NULL;

(pRunway->num\_of\_flights)++;

return SUCCESS;

}

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

//\* function name: removeFlight

//\* Description : Gets the pointer to a runway and flight id and removes it if exist on runway

//\* Parameters : pRunway - a pointer to a runway

//\* id - flight id

//\* Return Value : Result - SUCCESSS or FAILURE in removing the flight with the given id

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

Result removeFlight(PRUNWAY pRunway, long int id){

if (pRunway == NULL)

return FAILURE; // illegal pointer

if (isFlightExists(pRunway, id) == FALSE)

return FAILURE; // flight isn't on runway

NODE\* tmp;

tmp = pRunway->head;

if (tmp->pFlight->flight\_id == id){

// case it's the first flight

destroyFlight(tmp->pFlight);

pRunway->head = tmp->next;

free(tmp);

(pRunway->num\_of\_flights)--;

return SUCCESS;

}

while (tmp->next->pFlight->flight\_id != id)

// case elsewise

tmp=tmp->next; // goes to the flight linked before to the requested flight

NODE\* requested\_node = tmp->next;

destroyFlight(requested\_node->pFlight);

tmp->next = tmp->next->next;

free(requested\_node);

(pRunway->num\_of\_flights)--;

return SUCCESS;

}

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

//\* function name: depart

//\* Description : Gets the pointer to a runway and removes the first flight

//\* Parameters : pRunway - a pointer to a runway

//\* Return Value : Result - SUCCESSS or FAILURE in removing the first flight in the runway

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

Result depart(PRUNWAY pRunway){

if (pRunway == NULL)

return FAILURE; // illegal pointer

if (pRunway->num\_of\_flights == 0)

return FAILURE; // no flights on runway

NODE\* tmp;

tmp = pRunway->head;

destroyFlight(tmp->pFlight);

pRunway->head = tmp->next;

free(tmp);

(pRunway->num\_of\_flights)--;

return SUCCESS;

}

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

//\* function name: printRunway

//\* Description : Gets the pointer to a runway and prints the data of it(id, type, num of flights waiting

//\* and flights' data)

//\* Parameters : pRunway - a pointer to a runway

//\* Return Value : Result - SUCCESSS or FAILURE in printing the runway data

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

Result printRunway(PRUNWAY pRunway){

//

if (pRunway != NULL){

printf("Runway %ld ", pRunway->runway\_id);

char run\_type;

run\_type =( pRunway->run\_type == DOMESTIC) ? 'D' : 'I';

if (run\_type == 'D')

printf("domestic\n");

else

printf("international\n");

printf("%ld flights are waiting:\n", pRunway->num\_of\_flights);

NODE\* tmp;

tmp = pRunway->head;

while (tmp!=NULL){

char flight\_urgence;

flight\_urgence =(tmp->pFlight->emergency == TRUE) ? 'E' : 'R';

printf("Flight %ld %c %s %c\n", tmp->pFlight->flight\_id, run\_type, tmp->pFlight->dest, flight\_urgence);

tmp=tmp->next;

}

return SUCCESS;

}

return FAILURE;

}

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

//\* function name: getFlightNum

//\* Description : Gets a pointer to the runway and returns the amount of flights in the runway

//\* Parameters : pRunway - a pointer to a runway

//\* Return Value : num\_of\_flights - number of flights

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

long int getFlightNum**(**PRUNWAY pRunway**){**

**if** **(**pRunway **==** **NULL)**

**return** **-**1**;** // illegal runway id

**return** pRunway**->**num\_of\_flights**;**

**}**

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

//\* function name: addFlight

//\* Description : Gets a pointer to a runway and a pointer to a flight, COPY the flight and enters it to

//\* the runway by the ruleset given on HW2. The function returns SUCCESS if the flight was

//\* added and FAILURE if it failed.

//\* Parameters : pRunway - runway pointer

//\* pFlight - flight pointer

//\* Return Value : Result - SUCCESS or FAILURE of adding the flight to the runway

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

Result addFlight**(**PRUNWAY pRunway**,** PFLIGHT pFlight**){**

**if** **(**pRunway **==** **NULL** **||** pFlight **==** **NULL** **)**

**return** FAILURE**;** // illegal pointers

**if** **(**pRunway**->**run\_type **!=** pFlight**->**ftype**)**

**return** FAILURE**;** // flight type isn't the same of runway type

NODE**\*** tmp**;**

tmp **=** pRunway**->**head**;**

**if** **(**isFlightExists**(**pRunway**,** pFlight**->**flight\_id**))**// flight already exists on runway

**return** FAILURE**;**

//////////////////////////////////////////

// case it's the first flight on runway //

//////////////////////////////////////////

**if** **(**pRunway**->**num\_of\_flights **==** 0 **){**

tmp **=** **(**NODE**\*)**malloc**(sizeof(**NODE**));**

tmp**->**pFlight **=** createFlight**(**pFlight**->**flight\_id **,** pFlight**->**ftype **,** pFlight**->**dest**,** pFlight**->**emergency**);**

tmp**->**next **=** **NULL;**

pRunway**->**head **=** tmp**;**

**(**pRunway**->**num\_of\_flights**)++;**

**return** SUCCESS**;**

**}**

////////////////////////////////////////////////////////////////

// case the first flight is REGULAR and entering is EMERGENCY //

////////////////////////////////////////////////////////////////

**if** **(**tmp**->**pFlight**->**emergency **==** FALSE **&&** pFlight**->**emergency **==** TRUE**)** **{**

NODE**\*** tmp\_emer**=(**NODE**\*)**malloc**(sizeof(**NODE**));**

tmp\_emer**->**pFlight **=** createFlight**(**pFlight**->**flight\_id**,** pFlight**->**ftype**,** pFlight**->**dest**,** pFlight**->**emergency**);**

tmp\_emer**->**next **=** tmp**;**

pRunway**->**head **=** tmp\_emer**;**

**(**pRunway**->**num\_of\_flights**)++;**

**return** SUCCESS**;**

**}**

////////////////// goes until the first regular flight or until list ends

**while** **(**tmp**->**next **!=** **NULL** **&&** tmp**->**next**->**pFlight**->**emergency **==** TRUE**)**

tmp**=**tmp**->**next**;**

//////////////////////////////////////////////////////////////

// case when emergency flight or no other flights on runway //

//////////////////////////////////////////////////////////////

**if** **(**pFlight**->**emergency **==** TRUE**){**

NODE**\*** current\_next **=** tmp**->**next**;**

tmp**->**next **=** **(**NODE**\*)**malloc**(sizeof(**NODE**));**

tmp**->**next**->**pFlight **=** createFlight**(**pFlight**->**flight\_id**,** pFlight**->**ftype**,** pFlight**->**dest**,** pFlight**->**emergency**);**

tmp**->**next**->**next **=** current\_next**;** // re-links to the next flight

**(**pRunway**->**num\_of\_flights**)++;**

**return** SUCCESS**;**

**}**

/////////////////////////////////////////

// case when there are regular flights //

/////////////////////////////////////////

**while** **(**tmp**->**next **!=** **NULL)**

tmp**=**tmp**->**next**;** // goes to the end of the list

tmp**->**next **=** **(**NODE**\*)**malloc**(sizeof(**NODE**));**

tmp**->**next**->**pFlight **=** createFlight**(**pFlight**->**flight\_id **,** pFlight**->**ftype **,** pFlight**->**dest**,** pFlight**->**emergency**);**

tmp**->**next**->**next **=** **NULL;**

**(**pRunway**->**num\_of\_flights**)++;**

**return** SUCCESS**;**

**}**

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

//\* function name: removeFlight

//\* Description : Gets the pointer to a runway and flight id and removes it if exist on runway

//\* Parameters : pRunway - a pointer to a runway

//\* id - flight id

//\* Return Value : Result - SUCCESSS or FAILURE in removing the flight with the given id

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

Result removeFlight(PRUNWAY pRunway, long int id){

if (pRunway == NULL)

return FAILURE; // illegal pointer

if (isFlightExists(pRunway, id) == FALSE)

return FAILURE; // flight isn't on runway

NODE\* tmp;

tmp = pRunway->head;

if (tmp->pFlight->flight\_id == id){

// case it's the first flight

destroyFlight(tmp->pFlight);

pRunway->head = tmp->next;

free(tmp);

(pRunway->num\_of\_flights)--;

return SUCCESS;

}

while (tmp->next->pFlight->flight\_id != id)

// case elsewise

tmp=tmp->next; // goes to the flight linked before to the requested flight

NODE\* requested\_node = tmp->next;

destroyFlight(requested\_node->pFlight);

tmp->next = tmp->next->next;

free(requested\_node);

(pRunway->num\_of\_flights)--;

return SUCCESS;

}

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

//\* function name: depart

//\* Description : Gets the pointer to a runway and removes the first flight

//\* Parameters : pRunway - a pointer to a runway

//\* Return Value : Result - SUCCESSS or FAILURE in removing the first flight in the runway

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

Result depart(PRUNWAY pRunway){

if (pRunway == NULL)

return FAILURE; // illegal pointer

if (pRunway->num\_of\_flights == 0)

return FAILURE; // no flights on runway

NODE\* tmp;

tmp = pRunway->head;

destroyFlight(tmp->pFlight);

pRunway->head = tmp->next;

free(tmp);

(pRunway->num\_of\_flights)--;

return SUCCESS;

}

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

//\* function name: printRunway

//\* Description : Gets the pointer to a runway and prints the data of it(id, type, num of flights waiting

//\* and flights' data)

//\* Parameters : pRunway - a pointer to a runway

//\* Return Value : Result - SUCCESSS or FAILURE in printing the runway data

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

Result printRunway(PRUNWAY pRunway){

//

if (pRunway != NULL){

printf("Runway %ld ", pRunway->runway\_id);

char run\_type;

run\_type =( pRunway->run\_type == DOMESTIC) ? 'D' : 'I';

if (run\_type == 'D')

printf("domestic\n");

else

printf("international\n");

printf("%ld flights are waiting:\n", pRunway->num\_of\_flights);

NODE\* tmp;

tmp = pRunway->head;

while (tmp!=NULL){

char flight\_urgence;

flight\_urgence =(tmp->pFlight->emergency == TRUE) ? 'E' : 'R';

printf("Flight %ld %c %s %c\n", tmp->pFlight->flight\_id, run\_type, tmp->pFlight->dest, flight\_urgence);

tmp=tmp->next;

}

return SUCCESS;

}

return FAILURE;

}

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

//\* function name: removeFlight

//\* Description : Gets the pointer to a runway and flight id and removes it if exist on runway

//\* Parameters : pRunway - a pointer to a runway

//\* id - flight id

//\* Return Value : Result - SUCCESSS or FAILURE in removing the flight with the given id

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

Result removeFlight**(**PRUNWAY pRunway**,** long int id**){**

**if** **(**pRunway **==** **NULL)**

**return** FAILURE**;** // illegal pointer

**if** **(**isFlightExists**(**pRunway**,** id**)** **==** FALSE**)**

**return** FAILURE**;** // flight isn't on runway

NODE**\*** tmp**;**

tmp **=** pRunway**->**head**;**

**if** **(**tmp**->**pFlight**->**flight\_id **==** id**){**

// case it's the first flight

destroyFlight**(**tmp**->**pFlight**);**

pRunway**->**head **=** tmp**->**next**;**

free**(**tmp**);**

**(**pRunway**->**num\_of\_flights**)--;**

**return** SUCCESS**;**

**}**

**while** **(**tmp**->**next**->**pFlight**->**flight\_id **!=** id**)**

// case elsewise

tmp**=**tmp**->**next**;** // goes to the flight linked before to the requested flight

NODE**\*** requested\_node **=** tmp**->**next**;**

destroyFlight**(**requested\_node**->**pFlight**);**

tmp**->**next **=** tmp**->**next**->**next**;**

free**(**requested\_node**);**

**(**pRunway**->**num\_of\_flights**)--;**

**return** SUCCESS**;**

**}**

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

//\* function name: depart

//\* Description : Gets the pointer to a runway and removes the first flight

//\* Parameters : pRunway - a pointer to a runway

//\* Return Value : Result - SUCCESSS or FAILURE in removing the first flight in the runway

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

Result depart**(**PRUNWAY pRunway**){**

**if** **(**pRunway **==** **NULL)**

**return** FAILURE**;** // illegal pointer

**if** **(**pRunway**->**num\_of\_flights **==** 0**)**

**return** FAILURE**;** // no flights on runway

NODE**\*** tmp**;**

tmp **=** pRunway**->**head**;**

destroyFlight**(**tmp**->**pFlight**);**

pRunway**->**head **=** tmp**->**next**;**

free**(**tmp**);**

**(**pRunway**->**num\_of\_flights**)--;**

**return** SUCCESS**;**

**}**

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

//\* function name: printRunway

//\* Description : Gets the pointer to a runway and prints the data of it(id, type, num of flights waiting

//\* and flights' data)

//\* Parameters : pRunway - a pointer to a runway

//\* Return Value : Result - SUCCESSS or FAILURE in printing the runway data

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

Result printRunway**(**PRUNWAY pRunway**){**

//

**if** **(**pRunway **!=** **NULL){**

printf**(**"Runway %ld "**,** pRunway**->**runway\_id**);**

char run\_type**;**

run\_type **=(** pRunway**->**run\_type **==** DOMESTIC**)** **?** 'D' **:** 'I'**;**

**if** **(**run\_type **==** 'D'**)**

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

**else**

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

printf**(**"%ld flights are waiting:\n"**,** pRunway**->**num\_of\_flights**);**

NODE**\*** tmp**;**

tmp **=** pRunway**->**head**;**

**while** **(**tmp**!=NULL){**

char flight\_urgence**;**

flight\_urgence **=(**tmp**->**pFlight**->**emergency **==** TRUE**)** **?** 'E' **:** 'R'**;**

printf**(**"Flight %ld %c %s %c\n"**,** tmp**->**pFlight**->**flight\_id**,** run\_type**,** tmp**->**pFlight**->**dest**,** flight\_urgence**);**

tmp**=**tmp**->**next**;**

**}**

**return** SUCCESS**;**

**}**

**return** FAILURE**;**

**}**

**Airport.c**