main.C

```
/*
Authors:            Oron Eliza     032544264
                    Mor Hadar      302676838

Project:            HW assignment 3

Description:  This file contains the main function of the program.
*/

#define CRT_SECURE_NO_WARNINGS

#include "Baggage_Tracker.h"
#include "Argument_Check.h"
#include "Prepaing_The_Output.h"
#include "Closing_Program.h"


int main ( int argc , char* argv[])
{
	int retval = 0;
	airplane *head_list = NULL ;
	FILE *input_file = NULL ;

	retval = Checks (argv , &input_file);
	if (retval == 1 )
		return (1);

	retval = BaggageTracker (input_file , atoi(argv[2]) , atoi( argv[3])
,&head_list );
	if (retval == 1 )
		return (1);

	retval = Create_Output_File ( argv[4] , head_list );
	if (retval == 1 )
		return (1);

	return(0);
}
```

Basic_Types.h

```
/*
Authors:            Oron Eliza     032544264
                    Mor Hadar      302676838

Project:            HW assignment 3

Description:  This file contains the declarations of new types and constants.
*/

#ifndef BASIC_TYPES
#define BASIC_TYPES

#include <Windows.h>
#include <stdio.h>
#include <stdlib.h>

#define DESTINATION_LENGTH 3
#define LINE_LENGTH 128

/*
* This struct represents a single airplane. The fields are as follows:
* 1. destination - An array of char that contains the destination of the single
airplane.
* 2. num_of_pieces_mutex - A pointer to a mutex handle controlling the access to
the num of baggages of the airplaine.
* 3. num_of_pieces - An integer of the num of the baggages that must be loaded on
the single airplane.
* 4. next - A pointer to airplane struct. Will point to the next node in the liked
list.
*/
typedef struct airplane
{
       char destination[DESTINATION_LENGTH+1];
       HANDLE *num_of_pieces_mutex;
       int num_of_pieces ;
       struct airplane *next;
}airplane;

/*
*Load_Args - A struct of arguments. The fields are as follows:
*1. input_file - A pointer to input file
*2. array_mutex - A pointer to the mutexes array.
*                          Every single mutex allows / does not allow access to the
i slot in the check in array.
*3. semaphores - A pointer to the semaphores array.
*                         semaphore[0] - used to implement the ready - set - go
logic.
*                  semaphore[1] - used by the load thread to announce to the other
threads that he has finished load all the baggages in the input file.
*4. array_check_in - A pointer to array of char.
*                                Every trio of chars is a single slot.
*5. num_of_slots - An integer of the number of slots.
*6. num_of_sorters - An integer of the number of the sorters.
*/
typedef struct Load_Args
```

```
{
	FILE *input_file;
	HANDLE *array_mutex , *semaphores ;
	char *array_check_in ;
	int num_of_slots , num_of_sorters;
}Load_Args;

/*
*Sort_Args - A struct of arguments. The fields are as follows:
*1. array_mutex - A pointer to the mutexes array.
*                          Every single mutex allows / does not allow access to the
i slot in the check in array.
*2. semaphores - A pointer to the semaphores array.
*                           semaphore[0] - used to implement the ready - set - go
logic.
*                           semaphore[1] - used by the load thread to announce to
the other threads that he has finished load all the baggages in the input file
*3. array_check_in - A pointer to array of char.
*                                  Every trio of char is a single slot.
*4. num_of_slots - An integer of the number of slots.
*5. ptr_head_list - A pointer to pointer to the main airplanes list.
*6. new_airplane_mutex - A pointer to a mutex controlling the access for creating
a
*                                     a new node at the airplanes lits's
*/
typedef struct Sort_Args
{
	HANDLE *array_mutex , *semaphores ;
	char *array_check_in;
	int num_of_slots;
	airplane **ptr_head_list;
	HANDLE *new_airplane_mutex;

}Sort_Args;

#endif
```

Arguments_Check.h

```
/*
Authors:            Oron Eliza     032544264
                    Mor Hadar      302676838

Project:            HW assignment 3

Description:  This file contains the declarations of the function that performs
the arguments check.
*/


#ifndef ARGUMENT_CHECK
#define ARGUMENT_CHECK


#include <Windows.h>
#include <stdio.h>

/*
* The function checks if the input file can be opened.
*
* Input:
* -----
* 1. argv - An array of 4 strings contaning the user input arguments.
*           The function uses only the first argument - The name of the input file
* 2. ptr_input_file - A pointer to the input file (used as an additional output).
*
* Output:
* -----
* 1. integer - Returns 1 if a FATAL ERROR occurred (If the file can't be opened)
And 0 otherwise .
*/
int Checks (  char* argv[] , FILE** ptr_input_file );

#endif
```

Arguments_Check.c

```c
/*
Authors:            Oron Eliza     032544264
                    Mor Hadar      302676838

Project:            HW assignment 3

Description:  This file contains the implementation of the function that performs
the arguments check.
*/

#include "Argument_Check.h"

int Checks (  char* argv[] , FILE** ptr_input_file )
{
       *ptr_input_file = fopen (argv[1] , "r" );
       if ( *ptr_input_file == NULL)
       {
              printf("FATAL ERROR: Failed opening file\n" );
              return(1);
       }
       return (0);
}
```

Baggage_tracker.h

```
/*
Authors:            Oron Eliza     032544264
                    Mor Hadar      302676838

Project:            HW assignment 3

Description:  This file contains the declarations of the function that execute the
load and sort function.
*/

#ifndef BAGGAGE_TRACKER
#define BAGGAGE_TRACKER


#include "Basic_Types.h"
#include "Load_And_Sort_Logic.h"

/*
* The function execute the load and sort function by creating and intializing
needed arguments, creating the threads with these arguments
* and implements the head of the airplanes linked list to the main function.
*
* Input:
* -----
* 1. input_file - A pointer to the input file.
* 2. num_of_slots - An integer of the num of slots requested by the user.
* 3. num_of_sorters - An integer of the num of sorters requested by the user.
* 4. ptr_head_list - A pointer to pointer of the main airplanes list.
*
* Output:
* -----
* 1. Integer - Returns 1 if a FATAL ERROR occurred And 0 otherwise .
*                      ***** In case of FATAL ERROR the function closes all
reasources.
*                      ***** In case of proper run the function closes all
resources except the main airplanes list.
*/
int BaggageTracker ( FILE *input_file , int num_of_slots , int num_of_sorters ,
airplane **ptr_head_list);

#endif
```

Baggage_Tracker.C

```
/*
Authors:            Oron Eliza      032544264
                    Mor Hadar       302676838

Project:            HW assignment 3

Description:  This file contains the implemention of the function that execute the
load and sort function.
*/

#include "Baggage_Tracker.h"
#include "Closing_Program.h"


// Creates a check in array of char and initialize all the slots to be empty.
int Create_Array_Check_In (int num_of_slots , char **ptr_array_check_in )
{
	int i=0 ;

	*ptr_array_check_in = (char*)malloc( num_of_slots * DESTINATION_LENGTH *
sizeof(char)  );
	if( *ptr_array_check_in == NULL )
	{
		printf("FATAL ERROR: memory allocation failed");
		return (1);
	}

	for ( i = 0   ;   i < num_of_slots   ;   i++   )
	{
		(*ptr_array_check_in) [ i * ( DESTINATION_LENGTH  ) ] = '\0' ;
	}
	return (0);
}

// Creates a mutex array of handles and initialize all the mutex handles not to be
owned by any thread.
int Create_Array_Mutex ( int num_of_slots , HANDLE **ptr_array_mutex  )
{
	int i=0 ;

	*ptr_array_mutex = (HANDLE*)malloc( num_of_slots * sizeof(HANDLE)  );
	if( *ptr_array_mutex == NULL )
	{
		printf("FATAL ERROR: Memory allocation failed\n");
		return (1);
	}

	for (  i=0  ;  i < num_of_slots  ;  i++  )
	{
		(*ptr_array_mutex) [i] = CreateMutex (NULL , 0 ,NULL );
		if ((*ptr_array_mutex) [i] == NULL )
		{
			printf("FATAL ERROR : Creating mutex failed\n ");
			return(1);
		}
```

```
	}
	return (0);
}

// Creates a single mutex of new_airplane_mutex and initialize it not to be owned
by any thread.
int Create_New_Airplane_Mutex (HANDLE **new_airplane_mutex)
{
	*new_airplane_mutex = (HANDLE*) malloc ( sizeof(HANDLE) );
	if ( *new_airplane_mutex == NULL)
	{
		printf("FATAL ERROR: Memory allocation failed\n");
		return (1);
	}
	*(*new_airplane_mutex) = CreateMutex (NULL , 0 , NULL);
	if (*new_airplane_mutex == NULL )
	{
		printf("FATAL ERROR : Creating mutex failed\n ");
		return(1);
	}
	return (0);
}

// Creates two semaphores and intialize them to be with count = 0.
int Create_Semaphore (int num_of_sorters , HANDLE *semaphores)
{

	semaphores[0] = CreateSemaphore (NULL , 0 , num_of_sorters + 1 , NULL );
	if ( semaphores[0] == NULL )
	{
		printf("FATAL ERROR: Creating semaphore failed\n");
		return(1);
	}
	semaphores[1] = CreateSemaphore (NULL , 0 , num_of_sorters * 4 , NULL );
	if ( semaphores[1] == NULL )
	{
		printf("FATAL ERROR: Creating semaphore failed\n");
		return(1);
	}
	return(0);
}

// Intialize the argument struct for the load func
void Create_Load_Args ( FILE *input_file , HANDLE *array_mutex , HANDLE
*semaphores ,  char* array_check_in ,
                                        int num_of_slots , int num_of_sorters ,
Load_Args *ptr_load_args )
{
	ptr_load_args -> input_file = input_file ;
	ptr_load_args -> array_mutex = array_mutex ;
	ptr_load_args -> semaphores = semaphores ;
	ptr_load_args -> array_check_in = array_check_in ;
	ptr_load_args -> num_of_slots = num_of_slots ;
	ptr_load_args -> num_of_sorters = num_of_sorters ;
}

// Intialize the argument struct for the sort func
```

```
void Create_Sort_Args ( HANDLE *array_mutex , HANDLE *semaphores ,  char*
array_check_in , int num_of_slots ,
                                        airplane **ptr_head_list , HANDLE*
new_airplane_mutex , Sort_Args *ptr_sort_args )
{
       ptr_sort_args -> array_mutex = array_mutex ;
       ptr_sort_args -> semaphores = semaphores ;
       ptr_sort_args -> array_check_in = array_check_in ;
       ptr_sort_args -> num_of_slots = num_of_slots;
       ptr_sort_args -> ptr_head_list = ptr_head_list ;
       ptr_sort_args -> new_airplane_mutex = new_airplane_mutex ;
 }

// Creates an handle for a single thread running the load function.
HANDLE Run_Load_Single_Thread ( int (*func)(Load_Args*) ,Load_Args *ptr_arg)
{
       return CreateThread ( NULL  ,  0  ,  (LPTHREAD_START_ROUTINE)func  ,
ptr_arg  ,  0  ,  NULL  );
}

// Creates an handle for a single thread running the sort function.
HANDLE Run_Sort_Single_Thread ( int (*func)(Sort_Args*) ,Sort_Args *ptr_arg)
{
       return CreateThread ( NULL  , 0  , (LPTHREAD_START_ROUTINE)func , ptr_arg
,  0  ,  NULL  );
}

// Creates all the thread needed (a single load thread and a num_of_sorters
threads).
int Run_Threads ( HANDLE **ptr_threads , int num_of_sorters , Load_Args*
ptr_load_args , Sort_Args* ptr_sort_args)
{
       int i = 0;

       *ptr_threads = (HANDLE*)malloc(  (num_of_sorters +1 ) * sizeof(HANDLE)  );
       if( *ptr_threads == NULL )
       {
              printf("FATAL ERROR: Memory allocation failed\n");
              return (1);
       }

       (*ptr_threads) [0] = Run_Load_Single_Thread ( Load_Func, ptr_load_args ) ;
       if ( (*ptr_threads) [0] == NULL )
       {
              printf("FATAL ERROR: Last error 0x%x , Ending
program\n",GetLastError()  );
              return (1);
       }
       for ( i = 1 ;   i < num_of_sorters + 1   ;  i++ )
       {
              (*ptr_threads) [i] = Run_Sort_Single_Thread (Sort_Func,
ptr_sort_args ) ;
              if ( (*ptr_threads) [i] == NULL )
              {
                     printf("FATAL ERROR: Last error 0x%x , Ending program\n",
GetLastError()  );
                     return (1);
              }
```

```
	}
	return (0);
}

// Checks the returned values of all treads - 1 means a FATAL ERROR accurred and 0
means the thread ran properly.
int Check_For_Failed_Threads ( HANDLE* threads , int num_of_sorters )
{
	int i = 0 ;
	DWORD   exit_code = 0 ,  *Ip_exit_code = &exit_code ;
	BOOL success = 0 ;

	for ( i = 0  ; i < num_of_sorters + 1  ;  i++)
	{
		success = GetExitCodeThread ( threads[i] , Ip_exit_code);
		if ( *Ip_exit_code != 0 )
		{
			printf("FATAL ERROR: Last error 0x%x , Ending program\n",
GetLastError()  );
			return (1);
		}
	}
	return(0);
}

int BaggageTracker ( FILE *input_file , int num_of_slots , int num_of_sorters ,
airplane **ptr_head_list)
{
	HANDLE semaphores[2] = {NULL , NULL} , *threads = NULL , *array_mutex =
NULL , *new_airplane_mutex = NULL ;
	int retval = 0;
	DWORD error_flag = 0 ;
	Load_Args load_args ;
	Sort_Args sort_args ;
	char *array_check_in = NULL ;
	BOOL success = 0 ;


	retval = Create_Array_Check_In ( num_of_slots , &array_check_in );
	if (retval == 1 )
	{
		Closing_Program_Full (&load_args , &sort_args , threads );
		return (1);
	}

	retval = Create_Array_Mutex ( num_of_slots , &array_mutex );
	if (retval == 1 )
	{
		Closing_Program_Full (&load_args , &sort_args , threads );
		return (1);
	}

	retval = Create_New_Airplane_Mutex ( &new_airplane_mutex );
	if (retval == 1 )
	{
		Closing_Program_Full (&load_args , &sort_args , threads );
		return (1);
	}
```

```
	retval = Create_Semaphore ( num_of_sorters , semaphores ) ;
	if (retval == 1 )
	{
		Closing_Program_Full (&load_args , &sort_args , threads );
		return (1);
	}

	Create_Load_Args ( input_file , array_mutex ,semaphores, array_check_in ,
num_of_slots  ,num_of_sorters ,  &load_args );
	Create_Sort_Args ( array_mutex , semaphores , array_check_in , num_of_slots
, ptr_head_list , new_airplane_mutex , &sort_args);

	retval = Run_Threads( &threads , num_of_sorters , &load_args , &sort_args
);
	if (retval == 1 )
	{
		Closing_Program_Full (&load_args , &sort_args , threads );
		return (1);
	}
	Sleep (10000);
	success = ReleaseSemaphore ( semaphores[0] , num_of_sorters + 1 , NULL
);//Ready - Set - Goooooooo
	if ( success == 0 )
	{
		printf("FATAL ERROR: Releasing semaphore failed\n");
		Closing_Program_Full (&load_args , &sort_args , threads );
		return (1);
	}

	error_flag = WaitForMultipleObjects( num_of_sorters + 1 , threads , 1 ,
INFINITE );
	if (error_flag == WAIT_FAILED)
	{
		printf("FATAL ERROR: Last error 0x%x , Ending program\n",
GetLastError()  );
		Closing_Program_Full (&load_args , &sort_args , threads );
		return (1);
	}

	retval = Check_For_Failed_Threads (threads , num_of_sorters);
	if (retval == 1)
	{
		Closing_Program_Full (&load_args , &sort_args , threads );
		return (1);
	}

	Closing_Program_Partial (&load_args , &sort_args , threads );
	return (0);
}
```

Load And Sort Logic.h

```
/*
Authors:            Oron Eliza     032544264
                    Mor Hadar      302676838

Project:            HW assignment 3

Description:  This file contains the declarations of the functions that performs
the main logic of the program -
                            The loading assingment and the sorting assingment.
*/

#ifndef LOAD_AND_SORT_LOGIC
#define LOAD_AND_SORT_LOGIC


#include <string.h>
#include <tchar.h>
#include "Basic_Types.h"
#include "Closing_Program.h"

/*
* The fuction performs the loading assingment - reads the various destinations
from the input file and placing
* them in the available slots in the arrary check in.
*
* Input:
* -----
* 1. Load_Args - A pointer to a struct defined by us in the Basic_Typed module.
*
* Output:
* -----
* 1. Integer - Returns 1 if a FATAL ERROR occurred And 0 otherwise .
*/
int Load_Func ( Load_Args *load_args);


/*
* The fuction performs the sorting assingment - scaning the check in array
constantly and creates a temporary linked list of airplanes
* to every destination mentioned in the input file with the total number of
baggages that must to be loaded on the airplane, then merge the temporary linked
list
* with the main airplanes list.
*
* Input:
* -----
* 1. Sort_Args - A pointer to a struct defined by us in the Basic_Typed module.
*
* Output:
* -----
* 1. Integer - Returns 1 if a FATAL ERROR occurred And 0 otherwise .
*/
int Sort_Func ( Sort_Args *sort_args);

#endif
```

Load And Sort Logic.c

```
/*
Authors:            Oron Eliza      032544264
                    Mor Hadar       302676838

Project:            HW assignment 3

Description:  This file contains the implementation of the functions that performs
the main logic of the program -
                            The loading assingment and the sorting assinment.
*/

#include "Load_And_Sort_Logic.h"


///////// The load logic /////////

// Arranges the line that was read from the input file to be NULL terminated and
counts the number of
// destinations in the line.
int Arrange_Line (FILE* input_file , char* line)
{
       int line_len = 0;

       if ( feof(input_file) == 0 )
       {
              line_len = strlen (line);
              line[ line_len - 1 ] = '\0';
       }
       return (   (strlen(line) + 1 ) / 4   );
}

//Copies the i destination from the line to the 'destinations' string (and fixing
it to be NULL terminated).
void Copy_The_i_Destination_From_Line (char *line , char *destination, int i )
{
       int j = 0 ;
       for ( j = 0  ;  j < DESTINATION_LENGTH  ;  j++ )
       {
              destination[j] = toupper( line[ (DESTINATION_LENGTH + 1)*i + j ]
);
       }
       destination[j] = '\0' ;
       return;
}

//Copies the 'destination' string to the i slot in the check in array.
void Put_The_Baggage_At_The_i_Slot ( char* array_check_in , char* destination, int
i)
{
       int j=0;

       for (j =0 ;  j < DESTINATION_LENGTH  ; j++)
       {
              array_check_in [   (DESTINATION_LENGTH * i) + j  ] = destination[j]
;
```

```
	}
	return;
}

//Finds an available slot in the check in array and place the baggage inside.
//If all slots are taken - waits for 10ms and then performs the function again
until finding an available slot.
int Place_Baggage_In_An_Available_Slot ( char *destination , char *array_check_in
, HANDLE *array_mutex , int num_of_slots)
{
	int i = 0 , retval = 0 , all_slots_are_taken = 0 ;
	DWORD slot_is_locked = 0 ;
	BOOL error_flag = 0 ;

	for ( i = 0  ;  i < num_of_slots  ;  i++ )
	{
		slot_is_locked = WaitForSingleObject ( array_mutex[i] , 0 );
		if (slot_is_locked == WAIT_FAILED )
		{
			printf("FATAL ERROR: Last error = 0x%x , Ending program\n"  ,
GetLastError );
			return(1);
		}

		if (slot_is_locked == WAIT_TIMEOUT )
		{
			retval = WaitForSingleObject ( array_mutex[i] , INFINITE );
			if ( retval == 1)
			{
				printf("FATAL ERROR: Last error = 0x%x , Ending
program\n"  ,  GetLastError );
				return(1);
			}
			Put_The_Baggage_At_The_i_Slot ( array_check_in , destination ,
i);
			error_flag = ReleaseMutex ( array_mutex[i] ) ;
			if ( error_flag == 0 )
			{
				printf("FATAL ERROR: Releasing mutex failed\n");
				return (1);
			}
			return(0);


		}
		else
		{
			if ( array_check_in[ (DESTINATION_LENGTH ) * i ] == '\0' )
			{
				Put_The_Baggage_At_The_i_Slot ( array_check_in ,
destination , i);
				error_flag = ReleaseMutex ( array_mutex[i] ) ;
				if ( error_flag == 0 )
				{
					printf("FATAL ERROR: Releasing mutex failed\n");
					return (1);
				}
				return(0);
```

```
                        }
                        else
                        {
                                all_slots_are_taken++ ;
                                error_flag = ReleaseMutex ( array_mutex[i] ) ;
                                if ( error_flag == 0 )
                                {
                                        printf("FATAL ERROR: Releasing mutex failed\n");
                                        return (1);
                                }
                        }
                }

                if ( all_slots_are_taken == num_of_slots )
                {
                        all_slots_are_taken = 0 ;
                        i = -1 ;
                        Sleep (10) ;
                }
        }
}

int Load_Func ( Load_Args *load_args)
{
        char destination [ DESTINATION_LENGTH + 1 ] , line [LINE_LENGTH] ;
        int retval = 0 , num_of_destinations_in_line = 0 , i = 0;
        BOOL success = 0 ;
        DWORD error_flag;

        error_flag = WaitForSingleObject ( load_args -> semaphores[0] , INFINITE );
        if (error_flag == WAIT_FAILED)
        {
                printf("FATAL ERROR: Last error 0x%x , Ending program\n",
GetLastError()  );
                return (1);
        }

        while ( feof (load_args -> input_file) == 0 )
        {
                fgets( line  ,  LINE_LENGTH  ,  load_args -> input_file );
                num_of_destinations_in_line = Arrange_Line ( load_args -> input_file
, line );

                for ( i = 0   ;  i < num_of_destinations_in_line  ;  i++  )
                {
                        Copy_The_i_Destination_From_Line ( line  , destination , i );
                        retval = Place_Baggage_In_An_Available_Slot (   destination ,
load_args -> array_check_in   ,
                                                                        load_args -
> array_mutex  ,  load_args -> num_of_slots   );
                        if (retval == 1 )
                                return(1);
                }
        }
        success = ReleaseSemaphore( load_args -> semaphores[1] , ( load_args ->
num_of_sorters ) * 4 , NULL );
        if ( success == 0 )
        {
```

```
			printf("FATAL ERROR: Last error 0x%x , Ending program\n",
GetLastError()  );
			return (1);
		}
		return(0);
}



/////////  The sort logic /////////

//Evacuates the i slot in the check in array and remembers the baggage destination
in the string 'str'.
void Take_The_Baggage_From_The_i_Slot ( char* str ,  char *array_check_in  ,  int
i )
{
		int j = 0 ;

		for ( j = 0  ;  j < DESTINATION_LENGTH   ;  j++ )
		{
			str[j] = array_check_in [  ( (DESTINATION_LENGTH  ) * i ) + j  ];
		}
		str[j] = '\0' ;
		array_check_in [  ( DESTINATION_LENGTH  ) * i  ] = '\0' ;
		return;
}

// The function addes a single baggage to a temp liked_list of baggages.
// Every sort thread ownes is own temporary list of baggages until finishing
scaning the check in array once.
int Add_Baggage(  airplane **ptr_head_list  ,  char *str  )
{
		if (*ptr_head_list == NULL )
		{
			*ptr_head_list = (airplane*)malloc(  sizeof(airplane)  );
			if (*ptr_head_list == NULL )
			{
				printf("FATAL ERROR: Memory allocation failed\n");
				return (1);
			}
			(*ptr_head_list) -> num_of_pieces_mutex = NULL ;
			(*ptr_head_list) -> num_of_pieces = 1 ;
			(*ptr_head_list) -> next = NULL ;
			strcpy( (*ptr_head_list) -> destination , str);
			return(0);
		}
		else if (  strcmp( (*ptr_head_list) -> destination , str) == 0 )
		{
			(*ptr_head_list)->num_of_pieces ++ ;
			return (0);
		}
		else
			return ( Add_Baggage ( &( (*ptr_head_list)->next ) , str) );
}

//Merges the temporary baggages list of a single sort thread with the main
airplanes list.
```

```
int Add_Temp_List_To_Airplane_List ( airplane *ptr_head_temp_list , airplane
**ptr_head_list , HANDLE *new_airplane_mutex)
{
	airplane *temp_p = ptr_head_temp_list , **ptr = ptr_head_list ,
*new_plane_p = NULL ;
	DWORD error_flag = 0 ;
	BOOL success = 0 ;
	int update_flag = 0;

	while( temp_p  !=  NULL)
	{
		while ( *ptr != NULL )
		{
			if (   strcmp( (*ptr) -> destination , temp_p -> destination )
== 0  )
			{
				error_flag = WaitForSingleObject(  *( (*ptr)->
num_of_pieces_mutex ) , INFINITE ) ;
				if( error_flag == WAIT_FAILED)
				{
					printf("FATAL ERROR: Last error 0x%x , Ending
program\n",   GetLastError()  ) ;
					return (1);
				}
				(*ptr)-> num_of_pieces  =  (*ptr)-> num_of_pieces   +
temp_p-> num_of_pieces ;
				success = ReleaseMutex(  *( (*ptr)->
num_of_pieces_mutex ) );
				if (success == 0 )
				{
					printf("FATAL ERROR: Last error 0x%x , Ending
program\n",   GetLastError()  ) ;
					return (1) ;
				}
				temp_p = temp_p-> next ;
				ptr = ptr_head_list ;
				update_flag = 1;
				break;
			}
			else
			{
				ptr = &( (*ptr)-> next ) ;
			}
		}
		if ( update_flag == 1 )
		{
			update_flag = 0;
			continue;
		}
		error_flag = WaitForSingleObject(   *new_airplane_mutex  ,  0  );
		if (error_flag == WAIT_FAILED)
		{
			printf("FATAL ERROR: Last error 0x%x , Ending program\n",
GetLastError()  );
			return (1);
		}

		if (error_flag == WAIT_TIMEOUT)
```

```
		{
			Sleep (10) ;
		}
		else
		{
			new_plane_p = (airplane*)malloc(   sizeof(airplane)  ) ;
			if ( new_plane_p == NULL )
			{
				printf("FATAL ERROR: Memory allocation failed\n") ;
				success = ReleaseMutex(  *new_airplane_mutex );
				if (success == 0 )
				{
					printf("FATAL ERROR: Last error 0x%x , Ending
program\n",   GetLastError()  ) ;
					return (1);
				}
				return (1) ;
			}
			strcpy ( new_plane_p->destination , temp_p-> destination ) ;
			new_plane_p-> num_of_pieces = temp_p-> num_of_pieces ;
			new_plane_p-> next = NULL ;
			new_plane_p ->num_of_pieces_mutex = (HANDLE*)malloc(
sizeof(HANDLE)  ) ;
			if ( new_plane_p ->num_of_pieces_mutex == NULL )
			{
				printf("FATAL ERROR: Memory allocation failed\n") ;
				success = ReleaseMutex(  new_airplane_mutex );
				if (success == 0 )
				{
					printf("FATAL ERROR: Last error 0x%x , Ending
program\n",   GetLastError()  ) ;
					return (1);
				}
				return (1) ;
			}
			*( new_plane_p-> num_of_pieces_mutex ) = CreateMutex(  NULL  ,
0  ,  NULL ) ;
			if ( *(new_plane_p-> num_of_pieces_mutex) == NULL )
			{
				printf("FATAL ERROR: Creating mutex failed\n") ;
				free(new_plane_p) ;
				success = ReleaseMutex(  *new_airplane_mutex );
				if (success == 0 )
				{
					printf("FATAL ERROR: Last error 0x%x , Ending
program\n",   GetLastError()  ) ;
					return (1);
				}
				return (1) ;
			}
			*ptr = new_plane_p ;
			success = ReleaseMutex(  *new_airplane_mutex );
			if (success == 0 )
			{
				printf("FATAL ERROR: Last error 0x%x , Ending
program\n",   GetLastError()  ) ;
				return (1);
			}
```

```
                        temp_p = temp_p-> next ;
                        ptr = ptr_head_list ;
                }
        }
        return (0);
}

int Sort_Func ( Sort_Args *sort_args)
{

        airplane *temp_ptr_list = NULL ;
        int i = 0 , j = 0 , work = 0 , ceased_work = 0 , retval = 0 , baggage_added
= 0 ;
        char str [DESTINATION_LENGTH + 1] ;
        DWORD ready_set_go = 0 , mutex_flag = 0 , load_flag = 0 ;
        BOOL success = 0 ;

        ready_set_go = WaitForSingleObject ( sort_args -> semaphores[0] , INFINITE
);
        if (ready_set_go == WAIT_FAILED)
        {
                printf("FATAL ERROR: Last error 0x%x , Ending program\n",
GetLastError()  );
                return (1);
        }

        load_flag = WaitForSingleObject( sort_args -> semaphores[1] , 0);
        if (load_flag == WAIT_FAILED)
        {
                printf("FATAL ERROR: Last error 0x%x , Ending program\n",
GetLastError()  );
                return (1);
        }

        while ( (load_flag != WAIT_OBJECT_0)  ||   ( ceased_work )   )
        {
                work = 0;
                for ( i = 0  ;  i < (sort_args -> num_of_slots)  ;  i++ )
                {
                        mutex_flag = WaitForSingleObject ( sort_args ->array_mutex[i]
, 0 );
                        if (mutex_flag == WAIT_FAILED)
                        {
                                printf("FATAL ERROR: Last error 0x%x , Ending
program\n",   GetLastError()  );
                                return (1);
                        }
                        if ( mutex_flag == WAIT_OBJECT_0 )
                        {
                                if (sort_args ->  array_check_in [ (DESTINATION_LENGTH
) * i ]   !=   '\0'   )
                                {
                                        Take_The_Baggage_From_The_i_Slot ( str ,
sort_args ->array_check_in , i);
                                        baggage_added = 1;
                                        work = 1;
                                }
                                success = ReleaseMutex( sort_args -> array_mutex[i]  );
```

```
                        if ( success == 0 )
                        {
                                printf("FATAL ERROR: Last error 0x%x , Ending
program\n",   GetLastError()  );
                                return (1);
                        }
                        if (baggage_added == 1)
                        {
                                baggage_added = 0 ;
                                retval =  Add_Baggage ( &temp_ptr_list , str );
                                if (retval == 1)
                                {
                                        Release_P_List ( temp_ptr_list );
                                        return (1);
                                }
                        }
                }
        }
        if ( temp_ptr_list != NULL )
        {
                retval = Add_Temp_List_To_Airplane_List ( temp_ptr_list ,
sort_args-> ptr_head_list , sort_args-> new_airplane_mutex );
                Release_P_List (temp_ptr_list);
                temp_ptr_list = NULL;
        }

        if (retval == 1)
                return(1);

        if ( ceased_work == 1 )
                break;

        load_flag = WaitForSingleObject( sort_args -> semaphores[1] , 0);
        if (load_flag == WAIT_FAILED)
        {
                printf("FATAL ERROR: Last error 0x%x , Ending program\n",
GetLastError()  );
                return (1);
        }
        if ( (load_flag == WAIT_OBJECT_0) )
                ceased_work = 1 ;
        if ( work == 0 )
        {
                Sleep(10);
        }


    }
    return(0);
}
```

Preparing The Output.h

```
/*
Authors:            Oron Eliza     032544264
                    Mor Hadar      302676838

Project:            HW assignment 3

Description:  This file contains the declarations of the function that prepare the
output file.
*/
#ifndef PREPARING_THE_OUTPUT
#define PREPARING_THE_OUTPUT


#include "Basic_Types.h"
#include "Closing_Program.h"

/*
* The function prepare the output file and releases the main airplanes list.
*
* Input:
* -----
* 1. out_file_name - A string contains the output file name requested by the user.
* 2. head_list - A pointer to the head of the main airplane list.
*
* Output:
* -----
* 1. Integer - Returns 1 if a FATAL ERROR occurred and 0 otherwise .
*/
int Create_Output_File ( char* out_file_name , airplane *head_list);

#endif
```

Preparing The Output.c

```
/*
Authors:            Oron Eliza      032544264
                    Mor Hadar       302676838

Project:            HW assignment 3

Description:  This file contains the implemention of the function that prepare the
output file.
*/

#include "Prepaing_The_Output.h"

// The function finds the node which containes the lowest alphabetic destination
in an airplanes list and update the pointer to pointer of the previous node of min
node.
airplane* Find_Min (airplane *head_list , airplane **ptr_prev)
{
	airplane *prev_p_list = head_list , *prev = head_list , *min = head_list ;
	int test = 0;

	head_list = head_list->next ;
	while( head_list != NULL )
	{
		test = strcmp ( min ->destination , head_list -> destination );
		if (test > 0)
		{
			prev = prev_p_list ;
			min = head_list ;
			prev_p_list = head_list;
			head_list = head_list ->next ;
		}
		else
		{
			prev_p_list = head_list ;
			head_list = head_list -> next ;
		}
	}
	*ptr_prev = prev ;
	return (min);
}

// The function arranges the airplanes list by alphabetic order and update the
pointer to pointer of the head list.
airplane* Arrange_List_By_Alphabetic_Order(airplane **ptr_head_list )
{
	airplane  *min = NULL, *prev = *ptr_head_list ;

	if ( (*ptr_head_list) -> next == NULL )
		return ( *ptr_head_list);
	else
	{
		min = Find_Min (*ptr_head_list , &prev);
		if ( min == *ptr_head_list)
		{
```

```
                        (*ptr_head_list)->next = Arrange_List_By_Alphabetic_Order ( &(
(*ptr_head_list) -> next)  );
                        return ( *ptr_head_list);
                }
                else
                {
                        prev-> next = min->next;
                        min-> next = (*ptr_head_list);
                        *ptr_head_list = min;
                        (*ptr_head_list) -> next = Arrange_List_By_Alphabetic_Order (
&(    (*ptr_head_list) -> next)  );
                        return ( *ptr_head_list );
                }
        }
}

int Create_Output_File ( char* out_file_name , airplane *head_list)
{

        FILE *output_file = NULL ;
        int sum = 0 ;
        airplane *ptr = NULL;

        output_file = fopen (out_file_name , "w");
        if (output_file == NULL )
        {
                printf("FATAL ERROR: Failed opening output file\n");
                Release_P_List (head_list);
                return (1);
        }

        head_list = Arrange_List_By_Alphabetic_Order ( &head_list);

        ptr = head_list;
        while( ptr != NULL )
        {
                fprintf( output_file , "%d pieces of baggage were sent to %s.\n" ,
ptr -> num_of_pieces , ptr -> destination  );
                sum = sum + ptr -> num_of_pieces ;
                ptr = ptr -> next ;
        }
        fprintf ( output_file , "In total, %d pieces of baggage were handled." ,
sum);
        fclose (output_file);
        Release_P_List (head_list);
        return (0);
}
```

Closing_Program.h

```
/*
Authors:            Oron Eliza     032544264
                    Mor Hadar      302676838

Project:            HW assignment 3

Description:  This file contains the declarations of the functions that close the
different resources.
*/

#ifndef CLOSING_PROGRAM
#define CLOSING_PROGRAM

#include "Basic_Types.h"

/*
* The function closes all resources allocated to the program - used in case of
FATAL ERROR.
*
* Input:
* -----
* 1. Load_Args - A pointer to a struct defined by us in the Basic_Typed module.
* 2. Sort_Args - A pointer to a struct defined by us in the Basic_Typed module.
* 3. threads - A pointer to an array of the handles to all the threads that have
been created.
*
* Output:
* -----
* NONE
*/
void Closing_Program_Full ( Load_Args *ptr_load_args , Sort_Args *ptr_sort_args ,
HANDLE *threads );


/*
* The function releases an airplane lits.
*
* Input:
* -----
* 1. head_list - A pointer to the head of an airplane list.
*
* Output:
* -----
* NONE
*/
void Release_P_List ( airplane *head_list);

/*
* The function closes all resources allocate to the program except the airplanes
list - used in case of proper run.
*
* Input:
* -----
* 1. Load_Args - A pointer to a struct defined by us in the Basic_Typed module.
* 2. Sort_Args - A pointer to a struct defined by us in the Basic_Typed module.
```

```
* 3. threads - A pointer to an array of the handles to all the threads that have
been created.
*
* Output:
* -----
* NONE
*/
void Closing_Program_Partial ( Load_Args *ptr_load_args , Sort_Args *ptr_sort_args
, HANDLE *threads );

#endif
```

Closing_Program.c

```
/*
Authors:            Oron Eliza      032544264
                    Mor Hadar       302676838

Project:            HW assignment 3

Description:  This file contains the implemention of the functions that close the
different resources.
*/
#include "Closing_Program.h"


void Closing_Program_Partial ( Load_Args *ptr_load_args , Sort_Args *ptr_sort_args
, HANDLE *threads )
{
	int i = 0 ;


	fclose (ptr_load_args -> input_file );

	for ( i = 0  ;  i < ptr_load_args -> num_of_slots ; i++ )
	{
		if (ptr_load_args -> array_mutex[i] != NULL )//release??
			CloseHandle (ptr_load_args -> array_mutex[i] ) ;
	}

	for ( i = 0   ;  i < ptr_load_args ->num_of_sorters + 1   ;   i++ )
	{
		if (threads[i] != NULL )
			CloseHandle( threads[i] );
	}

	for (i=0  ; i<2 ; i++ )
	{
		if (ptr_load_args->semaphores[i] != NULL)//release??
			CloseHandle ( ptr_load_args->semaphores[i]);
	}

	if ( *(ptr_sort_args -> new_airplane_mutex) != NULL )//release??
	{
		CloseHandle( *(ptr_sort_args -> new_airplane_mutex) );
		free ( ptr_sort_args ->new_airplane_mutex);
	}

	if (ptr_load_args -> array_mutex != NULL )
		free ( ptr_load_args -> array_mutex );

	if ( ptr_load_args ->array_check_in != NULL )
		free ( ptr_load_args ->array_check_in );

	if ( threads != NULL )
		free( threads );
}

void Release_P_List ( airplane *head_list)
```

```
{
	if ( head_list == NULL)
		return;
	if ( head_list -> next != NULL )
	{
		Release_P_List (head_list ->next );
		if (head_list -> num_of_pieces_mutex != NULL )
			CloseHandle( *( head_list -> num_of_pieces_mutex ) );
		free( head_list );
	}
	else
	{
		if (head_list -> num_of_pieces_mutex != NULL )
			CloseHandle( *( head_list -> num_of_pieces_mutex ) );
		free( head_list );
	}
	return;
}

void Closing_Program_Full ( Load_Args *ptr_load_args , Sort_Args *ptr_sort_args ,
HANDLE *threads )
{
	Closing_Program_Partial (  ptr_load_args  ,  ptr_sort_args  ,  threads  );
	Release_P_List (  *ptr_sort_args -> ptr_head_list   );
	return;
}
```