Parciales

1 Parcial 1 2023-05-04

1.1 Array helpers

```c
/*
@file array_helpers.c
@brief Array Helpers method implementation
*/
#include <assert.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include "array_helpers.h"

static const int EXPECTED_DIM_VALUE = 2;

static const char* CITY_NAMES[CITIES] = {
    "Cordoba", "Rosario", "Posadas", "Tilcara", "Bariloche"};
static const char* SEASON_NAMES[SEASONS] = {"low", "high"};

void array_dump(BakeryProductTable a)
{
    for (unsigned int city = 0u; city < CITIES; ++city)
    {
        for (unsigned int season = 0u; season < SEASONS; ++season)
        {
            fprintf(stdout, "%s␣in␣%s␣season:␣flour␣(%u,%u)␣Yeast␣(%u,%u)␣Butter␣(%u,%u)␣Sales␣value␣%u",
                    CITY_NAMES[city], SEASON_NAMES[season], a[city][season].flour_cant,
                    a[city][season].flour_price, a[city][season].yeast_cant,
                    a[city][season].yeast_price, a[city][season].butter_cant,
                    a[city][season].butter_price, a[city][season].sale_value);
            fprintf(stdout, "\n");
        }
    }
}

unsigned int best_profit(BakeryProductTable a)
{
    unsigned int max_profit = 0u;
    unsigned int costo = 0u;
    for (unsigned int ciudad = 0; ciudad < CITIES; ciudad++)
    {
        for (season_t temporadas = 0; temporadas < SEASONS; temporadas++)
        {
            costo = ((a[ciudad][temporadas].flour_cant)*(a[ciudad][temporadas].flour_price)) +
            ((a[ciudad][temporadas].yeast_cant)*(a[ciudad][temporadas].yeast_price)) +
            ((a[ciudad][temporadas].butter_cant)*(a[ciudad][temporadas].butter_price));

            if (a[ciudad][temporadas].sale_value - costo > max_profit)
            {
                max_profit = a[ciudad][temporadas].sale_value - costo;
            }

        }

    }

    return max_profit;
}

void array_from_file(BakeryProductTable array, const char* filepath)
{
    FILE* file = NULL;

    file = fopen(filepath, "r");
    if (file == NULL)
    {
        fprintf(stderr, "File␣does␣not␣exist.\n");
        exit(EXIT_FAILURE);
    }

    int i = 0;
    while (!feof(file))
    {   unsigned int codciudad;
        season_t temp;
        int res = fscanf(file,"##%u??%u␣",&codciudad,&temp);
        if (res != EXPECTED_DIM_VALUE)
        {
            fprintf(stderr, "Invalid␣file.\n");
            exit(EXIT_FAILURE);
        }
        BakeryProduct product = bakery_product_from_file(file);
        array[codciudad][temp] = product;
        /* COMPLETAR: Leer y guardar product en el array multidimensional*/
        /* Agregar las validaciones que considere necesarias*/
        /* Completar*/
        ++i;
    }
    if( i != CITIES*SEASONS)
    {
        fprintf(stderr, "File␣is␣incomplete␣or␣overloaded.␣\n");
        exit(EXIT_FAILURE);
    }
    fclose(file);
}
```

```
/*
  @file array_helpers.h
  @brief defines array helpers methods. These methods operates over multidimensional array of prices
*/
#ifndef _ARRAY_HELPERS_H
#define _ARRAY_HELPERS_H
#include <stdbool.h>
#include "bakery_product.h"

#define CITIES 5
#define SEASONS 2

typedef BakeryProduct BakeryProductTable[CITIES][SEASONS];

/**
 * @brief Write the content of the array 'a' into stdout.
 * @param[in] a array to dump in stdout
 */
void array_dump(BakeryProductTable a);

/**
 * @brief calculates best bakery/season profit
 * @param[in] a array with data
 */
unsigned int best_profit(BakeryProductTable a);

/**
 * @brief reads an array of prices information from file
 * @details
 *
 *  Each element is read from the file located at 'filepath'.
 *  The file must exist in disk and must have its contents in a sequence of
 *  lines, each with the following format:
 *
 *   ##<city_number>??<season_number> (<f_c>,<f_p>) (<y_c>,<y_p>) (<b_c>,<b_p>) <s_v>
 *
 *   Those elements are copied into the multidimensional array 'a'.
 *   The dimensions of the array are given by the macros noted above.
 *
 * @param a array which will contain read file
 * @param filepath file with prices data
 */

void array_from_file(BakeryProductTable a, const char *filepath);

#endif
```

1.2 Bakery Product

```
/*
  @file bakery_product.c
  @brief Implements bakery product structure and methods
*/
#include <stdlib.h>
#include "bakery_product.h"

static const int AMOUNT_OF_PRODUCT_VARS = 7;
```

```

BakeryProduct bakery_product_from_file(FILE* file)
{
    BakeryProduct product;
    int res = fscanf(file, EXPECTED_PRODUCT_FILE_FORMAT,
                      &product.flour_cant, &product.flour_price,
                      &product.yeast_cant, &product.yeast_price,
                      &product.butter_cant, &product.butter_price,
                      &product.sale_value);
    if (res != AMOUNT_OF_PRODUCT_VARS)
    {
        fprintf(stderr, "Invalid␣product␣data.\n");
        exit(EXIT_FAILURE);
    }
    return product;
}
```

```
/*
  @file bakery_product.h
  @brief Defines bakery products information
*/

#ifndef _BAKERY_PRODUCT_H
#define _BAKERY_PRODUCT_H
#define EXPECTED_PRODUCT_FILE_FORMAT "(%u,%u) (%u,%u) (%u,%u) %u "
typedef enum
{
    low,
    high
} season_t;

#include <stdio.h>

/** @brief Type used to represent bakery product data.*/
typedef struct _product
{
    unsigned int flour_cant;
    unsigned int flour_price;
    unsigned int yeast_cant;
    unsigned int yeast_price;
    unsigned int butter_cant;
    unsigned int butter_price;
    unsigned int sale_value;
} BakeryProduct;

/**
 * @brief reads bakery product data from file line
 * @details
 * Bakery product file line must contain:
 * (<unsigned int>,<unsigned int>) (<unsigned int>,<unsigned int>) (<unsigned int>,<unsigned int>) <unsigned int>
 *
 * @param[in] file Opened file stream
 * @return BakeryProduct structure which contains read information from file
 */
BakeryProduct bakery_product_from_file(FILE *file);

#endif  //_BAKERY_PRODUCT_H
```

1.3 Main

```
/*
  @file main.c
  @brief Defines main program function
*/

/* First, the standard lib includes, alphabetically ordered */
#include <assert.h>
#include <stdio.h>
#include <stdlib.h>

/* Then, this project's includes, alphabetically ordered */
#include "array_helpers.h"

/**
 * @brief print usage help
 * @param[in] program_name Executable name
 */
void print_help(char *program_name)
{
    /* Print the usage help of this program. */
    printf(
        "Usage:␣%s␣<input␣file␣path>\n\n"
        "Load␣bakery␣product␣data␣from␣a␣given␣file␣in␣disk.\n"
        "\n"
```

```c
        "The␣input␣file␣must␣exist␣in␣disk␣and␣every␣line␣in␣it␣must␣have␣the␣following␣format:\n\n"
        "##<uint>\?\?<uint>␣(<uint>,<uint>)␣(<uint>,<uint>)␣(<uint>,<uint>)␣<uint>␣\n\n"
        "where␣each␣value␣represent:␣\n\n"
        "##<city_code>\?\?<season>␣(<flour_cant>,<flour_price>)␣(<yeast_cant>,<yeast_price>)␣(<butter_cant>,<butter_price>)␣<sales_value>␣\n\n"
        "Those␣elements␣must␣be␣integers␣and␣will␣be␣copied␣into␣the␣multidimensional␣integer␣array␣'a'.\n"
        "\n\n",
        program_name);
}

/**
 * @brief reads file path from command line
 *
 * @param[in] argc amount of command line arguments
 * @param[in] argv command line arguments
 *
 * @return An string containing read filepath
 */
char *parse_filepath(int argc, char *argv[])
{
    /* Parse the filepath given by command line argument. */
    char *result = NULL;

    if (argc < 2)
    {
        print_help(argv[0]);
        exit(EXIT_FAILURE);
    }

    result = argv[1];

    return (result);
}

/**
 * @brief Main program function
 *
 * @param[in] argc amount of command line arguments
 * @param[in] argv command line arguments
 *
 * @return EXIT_SUCCESS when programs executes correctly, EXIT_FAILURE otherwise
 */
int main(int argc, char *argv[])
{
    char *filepath = NULL;

    /* parse the filepath given in command line arguments */
    filepath = parse_filepath(argc, argv);

    /* create an array with the type of flight */
    BakeryProductTable array;

    /* parse the file to fill the array and obtain the actual length */
    array_from_file(array, filepath);

    /* show the data on the screen */
    array_dump(array);

    unsigned int  res = best_profit(array);
    fprintf(stdout, "%u",res);
    return (EXIT_SUCCESS);
}
```

2 Parcial 1 2022-05-03, Tema D

2.1 Array helpers

```c
/*
@file array_helpers.c
@brief Array Helpers method implementation
*/
#include <assert.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include "array_helpers.h"

/**
* @brief returns true when reach last line in flight file
* @return True when is the last line of the file, False otherwise
```

```c
*/
static bool is_last_line(unsigned int hour, unsigned int type) {
  return  hour == HOURS - 1u && type == TYPE - 1u;
}

void array_dump(DelayTable a) {
  for (unsigned int type = 0u; type < TYPE; ++type) {
    for (unsigned int hour = 0u; hour < HOURS; ++hour) {
      Flight f = a[type][hour];
      fprintf(stdout, "%c:␣%s␣flight␣with␣%u␣passengers␣arrived␣at␣%u:00,␣with␣%u␣delay",
        f.code,
        f.type == 0 ? "last_mile" : "layover",
        f.passengers_amount,
        f.hour - 1,
        f.delay
      );
      if (!is_last_line(hour, type))
      {
        fprintf(stdout, "\n");
      }
    }
  }
}

unsigned int compensation_cost (DelayTable a) {
  unsigned int total_cost = 0;
  for (unsigned j = 0u; j < 18; j++){
    if (a[0][j].delay > MAX_LM_DELAY_ALLOWED){
      total_cost = total_cost + ((a[0][j].delay - MAX_LM_DELAY_ALLOWED) * COMPENSATION_PER_MINUTE);
    }
  }
  for (unsigned j = 0u; j < 18; j++){
    if (a[1][j].delay > MAX_LAYOVER_DELAY_ALLOWED){
      total_cost = total_cost + ((a[1][j].delay - MAX_LAYOVER_DELAY_ALLOWED) * COMPENSATION_PER_MINUTE);
    }
  }
  return total_cost;
}


void array_from_file(DelayTable array, const char *filepath) {
  FILE *file = NULL;

  file = fopen(filepath, "r");
  if (file == NULL) {
    fprintf(stderr, "File␣does␣not␣exist.\n");
    exit(EXIT_FAILURE);
  }

  char code;
  int i = 0;
  while (!feof(file) && i < HOURS) {
    Flight last_mile_info = flight_from_file(file);
    last_mile_info.type = last_mile;
    Flight layover_info = flight_from_file(file);
    layover_info.type = layover;
    int res = fscanf(file, EXPECTED_FLIGHT_FILE_FORMAT, &code);
    if (res != 1) {
      fprintf(stderr, "Invalid␣file.\n");
      exit(EXIT_FAILURE);
    }
    last_mile_info.code = code;
    layover_info.code = code;
    array[0][i] = last_mile_info;
    array[1][i] = layover_info;
    i++;
  }
  fclose(file);
}
```

```c
/*
  @file array_helpers.h
  @brief defines array helpers methods. These methods operates over multidimensional array of layover
*/
#ifndef _ARRAY_HELPERS_H
#define _ARRAY_HELPERS_H

#include <stdbool.h>
#include "flight.h"

#define HOURS 24
#define TYPE 2
```

```

#define MAX_LM_DELAY_ALLOWED 60
#define MAX_LAYOVER_DELAY_ALLOWED 120
#define COMPENSATION_PER_MINUTE 0.5

typedef Flight DelayTable [TYPE][HOURS];

/**
 * @brief Write the content of the array 'a' into stdout.
 * @param[in] a array to dump in stdout
 */
void array_dump(DelayTable a);

/**
 * @brief calculates how much compensation the company has to pay.
 * @param[in] a array
 */
unsigned int compensation_cost(DelayTable a);


/**
 * @brief reads an array of delay information from file
 * @details
 *
 *  Each element is read from the file located at 'filepath'.
 *  The file must exist in disk and must have its contents in a sequence
      of
 *  lines, each with the following format:
 *
 *  <hour> <delay> <passengers_amount> <hour> <delay> <passengers_amount
     > <flight_code>
 *
 *   Those elements are copied into the multidimensional array 'a'.
 *   The dimensions of the array are given by the macros noted above.
 *
 * @param a array which will contain read file
 * @param filepath file with layover data
 */
void array_from_file(DelayTable a, const char *filepath);

#endif
```

2.2 Flight

```
/*
  @file layover.c
  @brief Implements flight structure and methods
*/
#include <stdlib.h>
#include "flight.h"

static const int AMOUNT_OF_FLIGHT_VARS = 3;

Flight flight_from_file(FILE* file)
{
    Flight flight;
    int res = fscanf(file, "%u␣%u␣%u␣", &flight.hour, &flight.delay, &
        flight.passengers_amount);
    if (res != AMOUNT_OF_FLIGHT_VARS){
      fprintf(stderr, "Error␣de␣lectura");
      exit(EXIT_FAILURE);
    }
    return flight;
}
```

```
/*
  @file flight.h
  @brief Defines airport flight data
*/

#ifndef _FLIGHT_H
#define _FLIGHT_H
#define EXPECTED_FLIGHT_FILE_FORMAT "#%c# "

typedef enum { last_mile , layover } flight_t;

#include <stdio.h>

/** @brief Type used to represent flight data.*/
typedef struct _flight
{
  char code;
  flight_t type;
  unsigned int hour; // hour
  unsigned int delay; // minutes
```

```c
  unsigned int passengers_amount;
} Flight;

/**
 * @brief reads flight data from file line
 * @details
 * Flight file line must contain:
 * <unsigned int> <unsigned int> <unsigned int> #<char>#
 *
 * @param[in] file Opened file stream
 * @return Flight structure which contain read information from file
 */
Flight flight_from_file(FILE* file);

#endif //_FLIGHT_H
```

2.3 Main

```c
/*
  @file main.c
  @brief Defines main program function
*/

/* First, the standard lib includes, alphabetically ordered */
#include <assert.h>
#include <stdio.h>
#include <stdlib.h>

/* Then, this project's includes, alphabetically ordered */
#include "array_helpers.h"

/**
 * @brief print usage help
 * @param[in] program_name Executable name
 */
void print_help(char *program_name) {
    /* Print the usage help of this program. */
    printf("Usage:␣%s␣<input␣file␣path>\n\n"
           "Load␣flight␣data␣from␣a␣given␣file␣in␣disk.\n"
           "\n"
           "The␣input␣file␣must␣exist␣in␣disk␣and␣every␣line␣in␣it␣must␣have␣the␣following␣format:\n\n"
           "<code>␣<flight␣type>␣<hour>␣<passengers>␣<flight␣type>␣<hour>␣<passengers>\n\n"
           "Those␣elements␣must␣be␣integers␣and␣will␣be␣copied␣into␣the␣multidimensional␣integer␣array␣'a'.\n"
           "\n\n",
           program_name);
}

/**
 * @brief reads file path from command line
 *
 * @param[in] argc amount of command line arguments
 * @param[in] argv command line arguments
 *
 * @return An string containing read filepath
 */
char *parse_filepath(int argc, char *argv[]) {
    /* Parse the filepath given by command line argument. */
    char *result = NULL;

    if (argc < 2) {
        print_help(argv[0]);
        exit(EXIT_FAILURE);
    }

    result = argv[1];

    return (result);
}

/**
 * @brief Main program function
 *
 * @param[in] argc amount of command line arguments
 * @param[in] argv command line arguments
 *
 * @return EXIT_SUCCESS when programs executes correctly, EXIT_FAILURE otherwise
 */
int main(int argc, char *argv[]) {
    char *filepath = NULL;

    /* parse the filepath given in command line arguments */
    filepath = parse_filepath(argc, argv);

    /* create an array with the type of flight */
    DelayTable array;

    /* parse the file to fill the array and obtain the actual length */
    array_from_file(array, filepath);

    /* show the array in the screen */
    array_dump(array);

    printf("\nCompensation␣cost:␣%u\n", compensation_cost(array));

    return (EXIT_SUCCESS);
}
```

3 Parcial 1 2022-04-28, Tema A

3.1 Array helpers

```c
/*
@file array_helpers.c
@brief Array Helpers method implementation
*/
#include <assert.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>

#include "array_helpers.h"

/**
* @brief returns true when reach last entry in flight table
* @return True when is the last entry of the flight table, False otherwise
*/
static bool is_last_line(unsigned int hour, unsigned int type) {
 return  hour == HOURS - 1u && type == TYPE - 1u;
}

void array_dump(LayoverTable a) {
 for (unsigned int hour = 0u; hour < HOURS; ++hour) {
  for (unsigned int type = 0u; type < TYPE; ++type) {
   Flight f = a[hour][type];
   fprintf(stdout, "%c:␣%s␣at␣%u:00␣with␣%u␣passengers", f.code, f.type == 0 ? "arrives" : "departs", f.hour - 1, f.passengers_amount);
   if (!is_last_line(hour, type))
   {
    fprintf(stdout, "\n");
   }
  }
 }
}

unsigned int passengers_amount_in_airport (LayoverTable a, unsigned int h) {
 unsigned int res;
 for(unsigned int i=0u; i<HOURS; i++) {
  if(a[i][1].hour == h) {
   res = a[i][1].passengers_amount;
  }
 }

 return res;
}

void array_from_file(LayoverTable array, const char *filepath) {
 FILE *file = NULL;

 file = fopen(filepath, "r");
 if (file == NULL) {
  fprintf(stderr, "File␣does␣not␣exist.\n");
  exit(EXIT_FAILURE);
 }

 char code;
 int i=0;
 while (i < HOURS && !feof(file)) {
  int res = fscanf(file, EXPECTED_FLIGHT_FILE_FORMAT, &code);
  if (res != 1) {
   fprintf(stderr, "Invalid␣file.\n");
   exit(EXIT_FAILURE);
  }

  // Assign flights for current hour
```

```
    array[i][arrival] = flight_from_file(file, code);
    array[i][departure] = flight_from_file(file, code);
    i++;
  }
  fclose(file);
}
```

```
/*
  @file array_helpers.h
  @brief defines array helpers methods. These methods operates over multidimensional array of layover
*/
#ifndef _ARRAY_HELPERS_H
#define _ARRAY_HELPERS_H

#include <stdbool.h>
#include "flight.h"

#define HOURS 24
#define TYPE 2

typedef Flight LayoverTable [HOURS][TYPE];

/**
 * @brief Write the content of the array 'a' into stdout.
 * @param[in] a array to dump in stdout
 */
void array_dump(LayoverTable a);

/**
 * @brief calculates how many passengers are awaiting for a flight.
 * @param[in] a array with data
 * @param[in] hour A value between 0 and 23 that represent the hour to compute
 *                 the amount of awaiting passengers
 */
unsigned int passengers_amount_in_airport(LayoverTable a, unsigned int hour);


/**
 * @brief reads an array of layover information from file
 * @details
 *
 * Each element is read from the file located at 'filepath'.
 * The file must exist in disk and must have its contents in a sequence of
 * lines, each with the following format:
 *
 *  <flight_code> <type> <hour> <passengers> <type> <hour> <passengers>
 *
 *  Those elements are copied into the multidimensional array 'a'.
 *  The dimensions of the array are given by the macros noted above.
 *
 * @param a array which will contain read file
 * @param filepath file with layover data
 */
void array_from_file(LayoverTable a, const char *filepath);

#endif
```

3.2 Flight

```
/*
  @file layover.c
  @brief Implements flight structure and methods
*/
#include <stdlib.h>
#include "flight.h"

static const int AMOUNT_OF_FLIGHT_VARS = 3 ;

Flight flight_from_file(FILE* file, char code)
{
    Flight flight;
    flight.code = code;

    int res = fscanf(file, "%u␣%u␣%u", &flight.type, &flight.hour, &flight.passengers_amount);

    if (res != AMOUNT_OF_FLIGHT_VARS){
      fprintf(stderr, "Error␣de␣lectura␣de␣datos");
      exit(EXIT_FAILURE);
    }
    return flight;
}
```

```c
/*
  @file layover.h
  @brief Defines an airport layover betwen the arrival and departure of a flight
*/

#ifndef _FLIGHT_H
#define _FLIGHT_H
#define EXPECTED_FLIGHT_FILE_FORMAT "_%c_"

typedef enum { arrival, departure } flight_t;

#include <stdio.h>

/** @brief Type used to represent flight data.*/
typedef struct _flight
{
  char code;
  flight_t type;
  unsigned int hour;
  unsigned int passengers_amount;
} Flight;

/**
 * @brief reads flight data from file line
 * @details
 * Flight file line must contain:
 * <unsigned int> <unsigned int> <unsigned int>
 *
 * @param[in] file Opened file stream
 * @param[in] code The flight code
 * @return Flight structure which contain read information from file
 */
Flight flight_from_file(FILE* file, char code);

#endif //_FLIGHT_H
```

3.3 Main

```c
/*
  @file main.c
  @brief Defines main program function
*/

/* First, the standard lib includes, alphabetically ordered */
#include <assert.h>
#include <stdio.h>
#include <stdlib.h>

/* Then, this project's includes, alphabetically ordered */
#include "array_helpers.h"

/**
 * @brief print usage help
 * @param[in] program_name Executable name
 */
void print_help(char *program_name) {
    /* Print the usage help of this program. */
    printf("Usage:␣%s␣<input␣file␣path>\n\n"
           "Load␣flight␣data␣from␣a␣given␣file␣in␣disk.\n"
           "\n"
           "The␣input␣file␣must␣exist␣in␣disk␣and␣every␣line␣in␣it␣must␣have␣the␣following␣format:\n\n"
           "<code>␣<flight␣type>␣<hour>␣<passengers>␣<flight␣type>␣<hour>␣<passengers>\n\n"
           "Those␣elements␣must␣be␣integers␣and␣will␣be␣copied␣into␣the␣multidimensional␣integer␣array␣'a'.\n"
           "\n\n",
           program_name);
}

/**
 * @brief reads file path from command line
 *
 * @param[in] argc amount of command line arguments
 * @param[in] argv command line arguments
 *
 * @return An string containing read filepath
 */
char *parse_filepath(int argc, char *argv[]) {
    /* Parse the filepath given by command line argument. */
    char *result = NULL;

    if (argc < 2) {
        print_help(argv[0]);
        exit(EXIT_FAILURE);
    }

    result = argv[1];

    return (result);
}

/**
 * @brief Main program function
 *
 * @param[in] argc amount of command line arguments
 * @param[in] argv command line arguments
 *
 * @return EXIT_SUCCESS when programs executes correctly, EXIT_FAILURE otherwise
 */
int main(int argc, char *argv[]) {
    char *filepath = NULL;

    /* parse the filepath given in command line arguments */
    filepath = parse_filepath(argc, argv);

    /* create an array with the type of flight */
    LayoverTable array;

    /* parse the file to fill the array and obtain the actual length */
    array_from_file(array, filepath);

    /* shows the data on the screen */
    array_dump(array);

    printf("\nAmount␣of␣passengers␣at␣%u:00␣:␣%u\n", 10, passengers_amount_in_airport(array, 10));

    return (EXIT_SUCCESS);
}
```

4 Parcial 1 2022-04-28, Tema B

4.1 Array helpers

```
/*
@file array_helpers.c
@brief Array Helpers method implementation
*/
#include <assert.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>

#include "array_helpers.h"

/**
* @brief returns true when reach last entry in flight table
* @return True when is the last entre of the flight table, False otherwise
*/
static bool is_last_line(unsigned int hour, unsigned int type) {
  return  hour == HOURS - 1u && type == TYPE - 1u;
}

void array_dump(DeliveryTable a) {
  for (unsigned int type = 0u; type < TYPE; ++type) {
    for (unsigned int hour = 0u; hour < HOURS; ++hour) {
      Flight f = a[type][hour];
      fprintf(stdout, "%c:␣flight␣with␣%u␣%s␣arrived␣at␣%u:00", f.code, f.items_amount, f.type == 0 ? "boxes" : "letters", f.hour - 1);
      if (!is_last_line(hour, type))
      {
        fprintf(stdout, "\n");
      }
    }
  }
}


unsigned int extra_space_fee_cost (DeliveryTable a) {
  unsigned int cost;
  // Costo de boxes
  for (unsigned int hrs = 1u; hrs <= HOURS; hrs ++){
    unsigned int cant = a[boxes][hrs].items_amount;
    if(cant > MAX_ALLOWED_BOXES && hrs < 7){
      int penalt = (cant - MAX_ALLOWED_BOXES) * BOX_PENALTY;
      cost = cost + penalt;
```

```c
    }
  }
  // Costo de letters
  for (unsigned int hrs = 1u; hrs <= HOURS; hrs ++){
    unsigned int cant = a[letters][hrs].items_amount;
    if(cant > MAX_ALLOWED_LETTERS && hrs < 7){
      int penalt = (cant - MAX_ALLOWED_LETTERS) * LETTER_PENALTY;
      cost = cost + penalt;
    }
  }
  return cost;
}


void array_from_file(DeliveryTable array, const char *filepath) {
  FILE *file = NULL;

  file = fopen(filepath, "r");
  if (file == NULL) {
    fprintf(stderr, "File␣does␣not␣exist.\n");
    exit(EXIT_FAILURE);
  }

  char code;
  int i = 0;
  while (i != HOURS) {
    // Lectura del codigo de vuelo
    int res = fscanf(file, EXPECTED_FLIGHT_FILE_FORMAT , &code);
    if (res != 1) {
      fprintf(stderr, "Invalid␣file.\n");
      exit(EXIT_FAILURE);
    }
    // Generar ambos Flight
    Flight flight_boxes = flight_from_file(file, code, boxes);
    Flight flight_letters = flight_from_file(file, code, letters);

    // Guardo los datos en el arreglo multidimensional
    array[boxes][flight_boxes.hour - 1] = flight_boxes;
    array[letters][flight_letters.hour - 1] = flight_letters;

    i++;
  }
}
```

```c
/*
  @file array_helpers.h
  @brief defines array helpers methods. These methods operates over multidimensional array of layover
*/
#ifndef _ARRAY_HELPERS_H
#define _ARRAY_HELPERS_H

#include <stdbool.h>
#include "flight.h"

#define HOURS 24
#define TYPE 2

#define MAX_ALLOWED_BOXES 75
#define MAX_ALLOWED_LETTERS 150
#define BOX_PENALTY 10
#define LETTER_PENALTY 2
#define FEE_CLOSE_HOUR 18

typedef Flight DeliveryTable [TYPE][HOURS];

/**
 * @brief Write the content of the array 'a' into stdout.
 * @param[in] a array to dump in stdout
 */
void array_dump(DeliveryTable a);

/**
 * @brief calculates how much extra fee the company has to pay for the day.
 * @param[in] a array
 * @details
 *   Counts items arrived until FEE_CLOSE_HOUR (inclusive).
 *   For each extra box adds BOX_PENALTY to the fee.
 *   For each extra letter adds LETTER_PENALTY to the fee.
 */
unsigned int extra_space_fee_cost(DeliveryTable a);


/**
 * @brief reads an array of layover information from file
 * @details
 *
 *  Each element is read from the file located at 'filepath'.
 *  The file must exist in disk and must have its contents in a sequence of
 *  lines, each with the following format:
 *
 *   <flight_code> <hour> <boxes> <hour> <letters>
 *
 *   Those elements are copied into the multidimensional array 'a'.
 *   The dimensions of the array are given by the macros noted above.
 *
 * @param a array which will contain read file
 * @param filepath file with layover data
 */
void array_from_file(DeliveryTable a, const char *filepath);

#endif
```

4.2 Flight

```c
/*
  @file layover.c
  @brief Implements flight structure and methods
*/
#include <stdlib.h>
#include "flight.h"

static const int AMOUNT_OF_FLIGHT_VARS = 2;

Flight flight_from_file(FILE* file, char code, item_t type)
{
    Flight flight;
    flight.code = code;
    flight.type = type;

    // Variable de lectura de datos
    int read = fscanf(file, "␣%u␣%u␣", &flight.hour, &flight.items_amount);

    // Verifico la lectura
    if (read != AMOUNT_OF_FLIGHT_VARS){
      fprintf(stderr, "Error␣de␣lectura");
      exit(EXIT_FAILURE);
    }

    return flight;
}
```

```c
/*
  @file layover.h
  @brief Defines an airport layover betwen the arrival and departure of a flight
*/

#ifndef _FLIGHT_H
#define _FLIGHT_H
#define EXPECTED_FLIGHT_FILE_FORMAT " _%c_ "

typedef enum { boxes, letters } item_t;

#include <stdio.h>

/** @brief Type used to represent flight data.*/
typedef struct _flight
{
  char code;
  item_t type;
  unsigned int hour;
  unsigned int items_amount;
} Flight;

/**
 * @brief reads flight data from file line
 * @details
 * Flight file line must contain:
 * <unsigned int> <unsigned int>
 *
 * @param[in] file Opened file stream
 * @param[in] code The flight code
 * @param[in] type The item-type (boxes / letters)
 * @return Flight structure which contain read information from file
 */
Flight flight_from_file(FILE* file, char code, item_t type);

#endif //_FLIGHT_H
```

4.3 Main

```c
/*
  @file main.c
  @brief Defines main program function
*/

/* First, the standard lib includes, alphabetically ordered */
#include <assert.h>
#include <stdio.h>
#include <stdlib.h>

/* Then, this project's includes, alphabetically ordered */
#include "array_helpers.h"

/**
 * @brief print usage help
 * @param[in] program_name Executable name
 */
void print_help(char *program_name) {
    /* Print the usage help of this program. */
    printf("Usage:␣%s␣<input␣file␣path>\n\n"
           "Load␣flight␣data␣from␣a␣given␣file␣in␣disk.\n"
           "\n"
           "The␣input␣file␣must␣exist␣in␣disk␣and␣every␣line␣in␣it␣must␣have␣the␣following␣format:\n\n"
           "<code>␣<arrival-hour>␣<amount-boxes>␣<arrival-hour>␣<amount-letters>\n\n"
           "Those␣elements␣will␣be␣copied␣into␣the␣multidimensional␣array␣'a'.\n"
           "\n\n",
           program_name);
}

/**
 * @brief reads file path from command line
 *
 * @param[in] argc amount of command line arguments
 * @param[in] argv command line arguments
 *
 * @return An string containing read filepath
 */
char *parse_filepath(int argc, char *argv[]) {
    /* Parse the filepath given by command line argument. */
    char *result = NULL;

    if (argc < 2) {
        print_help(argv[0]);
        exit(EXIT_FAILURE);
    }

    result = argv[1];

    return (result);
}

/**
 * @brief Main program function
 *
 * @param[in] argc amount of command line arguments
 * @param[in] argv command line arguments
 *
 * @return EXIT_SUCCESS when programs executes correctly, EXIT_FAILURE otherwise
 */
int main(int argc, char *argv[]) {
    char *filepath = NULL;

    /* parse the filepath given in command line arguments */
    filepath = parse_filepath(argc, argv);

    /* create an array with the type of flight */
    DeliveryTable array;

    /* parse the file to fill the array and obtain the actual length */
    array_from_file(array, filepath);

    /* show the ordered array in the screen */
    array_dump(array);

    printf("\nExtra␣fee␣cost:␣%u\n", extra_space_fee_cost(array));

    return (EXIT_SUCCESS);
}
```