

Parciales

1 Parcial 1 2023-05-04

1.1 Array helpers

```

1  /*
2  @file array_helpers.c
3  @brief Array Helpers method implementation
4  */
5  #include <assert.h>
6  #include <stdbool.h>
7  #include <stdio.h>
8  #include <stdlib.h>
9  #include "array_helpers.h"
10
11 static const int EXPECTED_DIM_VALUE = 2;
12
13 static const char* CITY_NAMES[CITIES] = {
14     "Cordoba", "Rosario", "Posadas", "Tilcara", "Bariloche"};
15 static const char* SEASON_NAMES[SEASONS] = {"low", "high"};
16
17 void array_dump(BakeryProductTable a)
18 {
19     for (unsigned int city = 0u; city < CITIES; ++city)
20     {
21         for (unsigned int season = 0u; season < SEASONS; ++season)
22         {
23             fprintf(stdout, "%s_in_%s_season: flour_(%u,%u) Yeast_(%u,%u)
24                 )_Butter_(%u,%u)_Sales_value_%u",
25                     CITY_NAMES[city], SEASON_NAMES[season], a[city][
26                         season].flour_cant,
27                         a[city][season].flour_price, a[city][season].
28                         yeast_cant,
29                         a[city][season].yeast_price, a[city][season].
30                         butter_cant,
31                         a[city][season].butter_price, a[city][season].
32                         sale_value);
33             fprintf(stdout, "\n");
34         }
35     }
36 }

```

```

29     }
30 }
31 }
32
33 unsigned int best_profit(BakeryProductTable a)
34 {
35     unsigned int max_profit = 0u;
36     unsigned int costo = 0u;
37     for (unsigned int ciudad = 0; ciudad < CITIES; ciudad++)
38     {
39         for (season_t temporadas = 0; temporadas < SEASONS; temporadas
40             ++)
41         {
42             costo = ((a[ciudad][temporadas].flour_cant)*(a[ciudad][
43                 temporadas].flour_price)) +
44                 ((a[ciudad][temporadas].yeast_cant)*(a[ciudad][temporadas].
45                     yeast_price)) +
46                 ((a[ciudad][temporadas].butter_cant)*(a[ciudad][temporadas].
47                     butter_price));
48
49             if (a[ciudad][temporadas].sale_value - costo > max_profit)
50             {
51                 max_profit = a[ciudad][temporadas].sale_value - costo;
52             }
53         }
54     }
55     return max_profit;
56 }
57
58 void array_from_file(BakeryProductTable array, const char* filepath)
59 {
60     FILE* file = NULL;
61
62     file = fopen(filepath, "r");
63     if (file == NULL)
64     {
65         fprintf(stderr, "File_does_not_exist.\n");
66         exit(EXIT_FAILURE);
67     }
68 }

```

```

68     int i = 0;
69     while (!feof(file))
70     {
71         unsigned int codciudad;
72         season_t temp;
73         int res = fscanf(file, "##%u??%u", &codciudad, &temp);
74         if (res != EXPECTED_DIM_VALUE)
75         {
76             fprintf(stderr, "Invalid file.\n");
77             exit(EXIT_FAILURE);
78         }
79         BakeryProduct product = bakery_product_from_file(file);
80         array[codciudad][temp] = product;
81         /* COMPLETAR: Leer y guardar product en el array
82            multidimensional*/
83         /* Agregar las validaciones que considere necesarias*/
84         /* Completar*/
85         ++i;
86     }
87     if (i != CITIES*SEASONS)
88     {
89         fprintf(stderr, "File is incomplete or overloaded.\n");
90         exit(EXIT_FAILURE);
91     }
92     fclose(file);
93 }
94
95 /*
96  @file array_helpers.h
97  @brief defines array helpers methods. These methods operates over
98         multidimensional array of prices
99 */
100 #ifndef _ARRAY_HELPERS_H
101 #define _ARRAY_HELPERS_H
102 #include <stdbool.h>
103 #include "bakery_product.h"
104
105 #define CITIES 5
106 #define SEASONS 2
107
108 typedef BakeryProduct BakeryProductTable[CITIES][SEASONS];
109
110 /**
111  * @brief Write the content of the array 'a' into stdout.

```

```

17  * @param[in] a array to dump in stdout
18  */
19 void array_dump(BakeryProductTable a);
20
21 /**
22  * @brief calculates best bakery/season profit
23  * @param[in] a array with data
24  */
25 unsigned int best_profit(BakeryProductTable a);
26
27 /**
28  * @brief reads an array of prices information from file
29  * @details
30  *
31  * Each element is read from the file located at 'filepath'.
32  * The file must exist in disk and must have its contents in a sequence
33  * of
34  * lines, each with the following format:
35  *
36  * ##<city_number>??<season_number> (<f_c>,<f_p>) (<y_c>,<y_p>) (<b_c
37  * >,<b_p>) <s_v>
38  *
39  * Those elements are copied into the multidimensional array 'a'.
40  * The dimensions of the array are given by the macros noted above.
41  *
42  * @param a array which will contain read file
43  * @param filepath file with prices data
44  */
45 void array_from_file(BakeryProductTable a, const char *filepath);
46 #endif

```

1.2 Bakery Product

```

1  /*
2  @file bakery_product.c
3  @brief Implements bakery product structure and methods
4  */
5  #include <stdlib.h>
6  #include "bakery_product.h"
7
8  static const int AMOUNT_OF_PRODUCT_VARS = 7;

```

```

9
10 BakeryProduct bakery_product_from_file(FILE* file)
11 {
12     BakeryProduct product;
13     int res = fscanf(file, EXPECTED_PRODUCT_FILE_FORMAT,
14                     &product.flour_cant, &product.flour_price,
15                     &product.yeast_cant, &product.yeast_price,
16                     &product.butter_cant, &product.butter_price,
17                     &product.sale_value);
18     if (res != AMOUNT_OF_PRODUCT_VARS)
19     {
20         fprintf(stderr, "Invalid product data.\n");
21         exit(EXIT_FAILURE);
22     }
23     return product;
24 }

```

```

1 /*
2  @file bakery_product.h
3  @brief Defines bakery products information
4  */
5
6 #ifndef _BAKERY_PRODUCT_H
7 #define _BAKERY_PRODUCT_H
8 #define EXPECTED_PRODUCT_FILE_FORMAT "(%u,%u) (%u,%u) (%u,%u) %u "
9 typedef enum
10 {
11     low,
12     high
13 } season_t;
14
15 #include <stdio.h>
16
17 /** @brief Type used to represent bakery product data.*/
18 typedef struct _product
19 {
20     unsigned int flour_cant;
21     unsigned int flour_price;
22     unsigned int yeast_cant;
23     unsigned int yeast_price;
24     unsigned int butter_cant;
25     unsigned int butter_price;
26     unsigned int sale_value;

```

```

27 } BakeryProduct;
28
29 /**
30  * @brief reads bakery product data from file line
31  * @details
32  * Bakery product file line must contain:
33  * (<unsigned int>,<unsigned int>) (<unsigned int>,<unsigned int>) (<
34  *   unsigned int>,<unsigned int>) <unsigned int>
35  *
36  * @param[in] file Opened file stream
37  * @return BakeryProduct structure which contains read information from
38  *   file
39  */
40 BakeryProduct bakery_product_from_file(FILE *file);
41
42 #endif // _BAKERY_PRODUCT_H

```

1.3 Main

```

1 /*
2  @file main.c
3  @brief Defines main program function
4  */
5
6 /* First, the standard lib includes, alphabetically ordered */
7 #include <assert.h>
8 #include <stdio.h>
9 #include <stdlib.h>
10
11 /* Then, this project's includes, alphabetically ordered */
12 #include "array_helpers.h"
13
14 /**
15  * @brief print usage help
16  * @param[in] program_name Executable name
17  */
18 void print_help(char *program_name)
19 {
20     /* Print the usage help of this program. */
21     printf(
22         "Usage: %s <input_file_path>\n\n"
23         "Load bakery product data from a given file in disk.\n"
24         "\n"

```

```

25     "The input file must exist in disk and every line in it must
        have the following format:\n\n"
26     "##<uint>\?\?\<uint>\<uint>\<uint>\<uint>\<uint>\<uint>\<
        uint>\<uint>\n\n"
27     "where each value represent:\n\n"
28     "##<city_code>\?\?\<season>\<flour_cant>\<flour_price>\<
        yeast_cant>\<yeast_price>\<butter_cant>\<butter_price>\<
        sales_value>\n\n"
29     "Those elements must be integers and will be copied into the
        multidimensional integer array'a'.\n"
30     "\n\n",
31     program_name);
32 }
33
34 /**
35  * @brief reads file path from command line
36  *
37  * @param[in] argc amount of command line arguments
38  * @param[in] argv command line arguments
39  *
40  * @return An string containing read filepath
41  */
42 char *parse_filepath(int argc, char *argv[])
43 {
44     /* Parse the filepath given by command line argument. */
45     char *result = NULL;
46
47     if (argc < 2)
48     {
49         print_help(argv[0]);
50         exit(EXIT_FAILURE);
51     }
52
53     result = argv[1];
54
55     return (result);
56 }
57
58 /**
59  * @brief Main program function
60  *
61  * @param[in] argc amount of command line arguments
62  * @param[in] argv command line arguments

```

```

63  *
64  * @return EXIT_SUCCESS when programs executes correctly, EXIT_FAILURE
        otherwise
65  */
66 int main(int argc, char *argv[])
67 {
68     char *filepath = NULL;
69
70     /* parse the filepath given in command line arguments */
71     filepath = parse_filepath(argc, argv);
72
73     /* create an array with the type of flight */
74     BakeryProductTable array;
75
76     /* parse the file to fill the array and obtain the actual length */
77     array_from_file(array, filepath);
78
79     /* show the data on the screen */
80     array_dump(array);
81
82     unsigned int res = best_profit(array);
83     fprintf(stdout, "%u", res);
84     return (EXIT_SUCCESS);
85 }

```

2 Parcial 1 2022-05-03, Tema D

2.1 Array helpers

```

1  /*
2  @file array_helpers.c
3  @brief Array Helpers method implementation
4  */
5  #include <assert.h>
6  #include <stdbool.h>
7  #include <stdio.h>
8  #include <stdlib.h>
9  #include "array_helpers.h"
10
11 /**
12  * @brief returns true when reach last line in flight file
13  * @return True when is the last line of the file, False otherwise

```

```

14 */
15 static bool is_last_line(unsigned int hour, unsigned int type) {
16     return hour == HOURS - 1u && type == TYPE - 1u;
17 }
18
19 void array_dump(DelayTable a) {
20     for (unsigned int type = 0u; type < TYPE; ++type) {
21         for (unsigned int hour = 0u; hour < HOURS; ++hour) {
22             Flight f = a[type][hour];
23             fprintf(stdout, "%c: %s flight with %u passengers arrived at %u
24                 :00, with %u delay",
25                 f.code,
26                 f.type == 0 ? "last_mile" : "layover",
27                 f.passengers_amount,
28                 f.hour - 1,
29                 f.delay
30             );
31             if (!is_last_line(hour, type))
32                 fprintf(stdout, "\n");
33         }
34     }
35 }
36
37 unsigned int compensation_cost (DelayTable a) {
38     unsigned int total_cost = 0;
39     for (unsigned j = 0u; j < 18; j++){
40         if (a[0][j].delay > MAX_LM_DELAY_ALLOWED){
41             total_cost = total_cost + ((a[0][j].delay - MAX_LM_DELAY_ALLOWED)
42                 * COMPENSATION_PER_MINUTE);
43         }
44     }
45     for (unsigned j = 0u; j < 18; j++){
46         if (a[1][j].delay > MAX_LAYOVER_DELAY_ALLOWED){
47             total_cost = total_cost + ((a[1][j].delay -
48                 MAX_LAYOVER_DELAY_ALLOWED) * COMPENSATION_PER_MINUTE);
49         }
50     }
51     return total_cost;
52 }
53

```

```

54 void array_from_file(DelayTable array, const char *filepath) {
55     FILE *file = NULL;
56
57     file = fopen(filepath, "r");
58     if (file == NULL) {
59         fprintf(stderr, "File does not exist.\n");
60         exit(EXIT_FAILURE);
61     }
62
63     char code;
64     int i = 0;
65     while (!feof(file) && i < HOURS) {
66         Flight last_mile_info = flight_from_file(file);
67         last_mile_info.type = last_mile;
68         Flight layover_info = flight_from_file(file);
69         layover_info.type = layover;
70         int res = fscanf(file, EXPECTED_FLIGHT_FILE_FORMAT, &code);
71         if (res != 1) {
72             fprintf(stderr, "Invalid file.\n");
73             exit(EXIT_FAILURE);
74         }
75         last_mile_info.code = code;
76         layover_info.code = code;
77         array[0][i] = last_mile_info;
78         array[1][i] = layover_info;
79         i++;
80     }
81     fclose(file);
82 }
83
84 /*
85  * @file array_helpers.h
86  * @brief defines array helpers methods. These methods operates over
87  *        multidimensional array of layover
88  */
89 #ifndef _ARRAY_HELPERS_H
90 #define _ARRAY_HELPERS_H
91
92 #include <stdbool.h>
93 #include "flight.h"
94
95 #define HOURS 24
96 #define TYPE 2
97

```

```

13
14 #define MAX_LM_DELAY_ALLOWED 60
15 #define MAX_LAYOVER_DELAY_ALLOWED 120
16 #define COMPENSATION_PER_MINUTE 0.5
17
18 typedef Flight DelayTable [TYPE] [HOURS];
19
20 /**
21  * @brief Write the content of the array 'a' into stdout.
22  * @param[in] a array to dump in stdout
23  */
24 void array_dump(DelayTable a);
25
26 /**
27  * @brief calculates how much compensation the company has to pay.
28  * @param[in] a array
29  */
30 unsigned int compensation_cost(DelayTable a);
31
32
33 /**
34  * @brief reads an array of delay information from file
35  * @details
36  *
37  * Each element is read from the file located at 'filepath'.
38  * The file must exist in disk and must have its contents in a sequence
39  * of
40  * lines, each with the following format:
41  *
42  * <hour> <delay> <passengers_amount> <hour> <delay> <passengers_amount>
43  * > <flight_code>
44  *
45  * Those elements are copied into the multidimensional array 'a'.
46  * The dimensions of the array are given by the macros noted above.
47  *
48  * @param a array which will contain read file
49  * @param filepath file with layover data
50  */
51 void array_from_file(DelayTable a, const char *filepath);
52
53 #endif

```

2.2 Flight

```

1 /*
2  * @file layover.c
3  * @brief Implements flight structure and methods
4  */
5 #include <stdlib.h>
6 #include "flight.h"
7
8 static const int AMOUNT_OF_FLIGHT_VARS = 3;
9
10 Flight flight_from_file(FILE* file)
11 {
12     Flight flight;
13     int res = fscanf(file, "%u%u%u", &flight.hour, &flight.delay, &
14         flight.passengers_amount);
15     if (res != AMOUNT_OF_FLIGHT_VARS){
16         fprintf(stderr, "Error de lectura");
17         exit(EXIT_FAILURE);
18     }
19     return flight;
20 }
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

```

```

1 /*
2  * @file flight.h
3  * @brief Defines airport flight data
4  */
5
6 #ifndef _FLIGHT_H
7 #define _FLIGHT_H
8 #define EXPECTED_FLIGHT_FILE_FORMAT "%c# "
9
10 typedef enum { last_mile , layover } flight_t;
11
12 #include <stdio.h>
13
14 /** @brief Type used to represent flight data.*/
15 typedef struct _flight
16 {
17     char code;
18     flight_t type;
19     unsigned int hour; // hour
20     unsigned int delay; // minutes

```

```

21     unsigned int passengers_amount;
22 } Flight;
23
24 /**
25  * @brief reads flight data from file line
26  * @details
27  * Flight file line must contain:
28  * <unsigned int> <unsigned int> <unsigned int> #<char>#
29  *
30  * @param[in] file Opened file stream
31  * @return Flight structure which contain read information from file
32  */
33 Flight flight_from_file(FILE* file);
34
35 #endif //_FLIGHT_H

```

2.3 Main

```

1  /*
2   @file main.c
3   @brief Defines main program function
4  */
5
6  /* First, the standard lib includes, alphabetically ordered */
7  #include <assert.h>
8  #include <stdio.h>
9  #include <stdlib.h>
10
11 /* Then, this project's includes, alphabetically ordered */
12 #include "array_helpers.h"
13
14 /**
15  * @brief print usage help
16  * @param[in] program_name Executable name
17  */
18 void print_help(char *program_name) {
19     /* Print the usage help of this program. */
20     printf("Usage: %s <input_file_path>\n\n"
21           "Load flight data from a given file in disk.\n"
22           "\n"
23           "The input file must exist in disk and every line in it must"
24           "have the following format:\n\n"
25           "<code><flight_type> <hour> <passengers> <flight_type> <hour>

```

```

25         > <passengers>\n\n"
26         "Those elements must be integers and will be copied into the"
27         "multidimensional integer array 'a'.\n"
28         "\n\n",
29         program_name);
30 }
31
32 /**
33  * @brief reads file path from command line
34  *
35  * @param[in] argc amount of command line arguments
36  * @param[in] argv command line arguments
37  *
38  * @return An string containing read filepath
39  */
40 char *parse_filepath(int argc, char *argv[]) {
41     /* Parse the filepath given by command line argument. */
42     char *result = NULL;
43
44     if (argc < 2) {
45         print_help(argv[0]);
46         exit(EXIT_FAILURE);
47     }
48
49     result = argv[1];
50
51     return (result);
52 }
53
54 /**
55  * @brief Main program function
56  *
57  * @param[in] argc amount of command line arguments
58  * @param[in] argv command line arguments
59  *
60  * @return EXIT_SUCCESS when programs executes correctly, EXIT_FAILURE
61  * otherwise
62  */
63 int main(int argc, char *argv[]) {
64     char *filepath = NULL;
65
66     /* parse the filepath given in command line arguments */
67     filepath = parse_filepath(argc, argv);

```

```

65
66  /* create an array with the type of flight */
67  DelayTable array;
68
69  /* parse the file to fill the array and obtain the actual length */
70  array_from_file(array, filepath);
71
72  /* show the array in the screen */
73  array_dump(array);
74
75  printf("\nCompensation_cost: %u\n", compensation_cost(array));
76
77  return (EXIT_SUCCESS);
78 }

```

3 Parcial 1 2022-04-28, Tema A

3.1 Array helpers

```

1  /*
2  @file array_helpers.c
3  @brief Array Helpers method implementation
4  */
5  #include <assert.h>
6  #include <stdbool.h>
7  #include <stdio.h>
8  #include <stdlib.h>
9
10 #include "array_helpers.h"
11
12 /**
13  * @brief returns true when reach last entry in flight table
14  * @return True when is the last entry of the flight table, False
15  *         otherwise
16  */
17 static bool is_last_line(unsigned int hour, unsigned int type) {
18     return hour == HOURS - 1u && type == TYPE - 1u;
19 }
20
21 void array_dump(LayoverTable a) {
22     for (unsigned int hour = 0u; hour < HOURS; ++hour) {
23         for (unsigned int type = 0u; type < TYPE; ++type) {

```

```

23         Flight f = a[hour][type];
24         fprintf(stdout, "%c: %s at %u:00 with %u passengers", f.code, f.
25             type == 0 ? "arrives" : "departs", f.hour - 1, f.
26             passengers_amount);
27         if (!is_last_line(hour, type))
28             {
29             fprintf(stdout, "\n");
30             }
31     }
32 }
33
34 unsigned int passengers_amount_in_airport (LayoverTable a, unsigned int
35     h) {
36     unsigned int res;
37     for(unsigned int i=0u; i<HOURS; i++) {
38         if(a[i][1].hour == h) {
39             res = a[i][1].passengers_amount;
40         }
41     }
42     return res;
43 }
44
45 void array_from_file(LayoverTable array, const char *filepath) {
46     FILE *file = NULL;
47
48     file = fopen(filepath, "r");
49     if (file == NULL) {
50         fprintf(stderr, "File does not exist.\n");
51         exit(EXIT_FAILURE);
52     }
53
54     char code;
55     int i=0;
56     while (i < HOURS && !feof(file)) {
57         int res = fscanf(file, EXPECTED_FLIGHT_FILE_FORMAT, &code);
58         if (res != 1) {
59             fprintf(stderr, "Invalid file.\n");
60             exit(EXIT_FAILURE);
61         }
62         // Assign flights for current hour

```



```

63     array[i][arrival] = flight_from_file(file, code);
64     array[i][departure] = flight_from_file(file, code);
65     i++;
66 }
67 fclose(file);
68 }

1  /*
2   @file array_helpers.h
3   @brief defines array helpers methods. These methods operates over
4       multidimensional array of layover
5  */
6  #ifndef _ARRAY_HELPERS_H
7  #define _ARRAY_HELPERS_H
8
9  #include <stdbool.h>
10 #include "flight.h"
11
12 #define HOURS 24
13 #define TYPE 2
14
15 typedef Flight LayoverTable [HOURS][TYPE];
16
17 /**
18  * @brief Write the content of the array 'a' into stdout.
19  * @param[in] a array to dump in stdout
20  */
21 void array_dump(LayoverTable a);
22
23 /**
24  * @brief calculates how many passengers are awaiting for a flight.
25  * @param[in] a array with data
26  * @param[in] hour A value between 0 and 23 that represent the hour to
27  *               compute
28  *               the amount of awaiting passengers
29  */
30 unsigned int passengers_amount_in_airport(LayoverTable a, unsigned int
31     hour);
32
33 /**
34  * @brief reads an array of layover information from file
35  * @details

```

```

34  *
35  * Each element is read from the file located at 'filepath'.
36  * The file must exist in disk and must have its contents in a sequence
37  * of
38  * lines, each with the following format:
39  *
40  * <flight_code> <type> <hour> <passengers> <type> <hour> <passengers>
41  *
42  * Those elements are copied into the multidimensional array 'a'.
43  * The dimensions of the array are given by the macros noted above.
44  *
45  * @param a array which will contain read file
46  * @param filepath file with layover data
47  */
48 void array_from_file(LayoverTable a, const char *filepath);
49 #endif

```

3.2 Flight

```

1  /*
2   @file layover.c
3   @brief Implements flight structure and methods
4  */
5  #include <stdlib.h>
6  #include "flight.h"
7
8  static const int AMOUNT_OF_FLIGHT_VARS = 3 ;
9
10 Flight flight_from_file(FILE* file, char code)
11 {
12     Flight flight;
13     flight.code = code;
14
15     int res = fscanf(file, "%u%u%u", &flight.type, &flight.hour, &
16         flight.passengers_amount);
17
18     if (res != AMOUNT_OF_FLIGHT_VARS){
19         fprintf(stderr, "Error de lectura de datos");
20         exit(EXIT_FAILURE);
21     }
22     return flight;
23 }

```

```

1  /*
2   @file layover.h
3   @brief Defines an airport layover between the arrival and departure of
        a flight
4  */
5
6  #ifndef _FLIGHT_H
7  #define _FLIGHT_H
8  #define EXPECTED_FLIGHT_FILE_FORMAT "%c_"
9
10 typedef enum { arrival, departure } flight_t;
11
12 #include <stdio.h>
13
14 /** @brief Type used to represent flight data.*/
15 typedef struct _flight
16 {
17     char code;
18     flight_t type;
19     unsigned int hour;
20     unsigned int passengers_amount;
21 } Flight;
22
23 /**
24  * @brief reads flight data from file line
25  * @details
26  * Flight file line must contain:
27  * <unsigned int> <unsigned int> <unsigned int>
28  *
29  * @param[in] file Opened file stream
30  * @param[in] code The flight code
31  * @return Flight structure which contain read information from file
32  */
33 Flight flight_from_file(FILE* file, char code);
34
35 #endif //_FLIGHT_H

```

3.3 Main

```

1  /*
2   @file main.c
3   @brief Defines main program function
4  */

```

```

5
6  /* First, the standard lib includes, alphabetically ordered */
7  #include <assert.h>
8  #include <stdio.h>
9  #include <stdlib.h>
10
11 /* Then, this project's includes, alphabetically ordered */
12 #include "array_helpers.h"
13
14 /**
15  * @brief print usage help
16  * @param[in] program_name Executable name
17  */
18 void print_help(char *program_name) {
19     /* Print the usage help of this program. */
20     printf("Usage: %s<input_file_path>\n\n"
21           "Load flight data from a given file in disk.\n"
22           "\n"
23           "The input file must exist in disk and every line in it must"
24           "have the following format:\n\n"
25           "<code> <flight_type> <hour> <passengers> <flight_type> <hour>"
26           " <passengers>\n\n"
27           "Those elements must be integers and will be copied into the"
28           " multidimensional integer array 'a'. \n"
29           "\n\n",
30           program_name);
31 }
32
33 /**
34  * @brief reads file path from command line
35  *
36  * @param[in] argc amount of command line arguments
37  * @param[in] argv command line arguments
38  *
39  * @return An string containing read filepath
40  */
41 char *parse_filepath(int argc, char *argv[]) {
42     /* Parse the filepath given by command line argument. */
43     char *result = NULL;
44
45     if (argc < 2) {
46         print_help(argv[0]);
47         exit(EXIT_FAILURE);
48     }
49 }

```

```

45     }
46
47     result = argv[1];
48
49     return (result);
50 }
51
52 /**
53  * @brief Main program function
54  *
55  * @param[in] argc amount of command line arguments
56  * @param[in] argv command line arguments
57  *
58  * @return EXIT_SUCCESS when programs executes correctly, EXIT_FAILURE
59  *         otherwise
60 */
61 int main(int argc, char *argv[]) {
62     char *filepath = NULL;
63
64     /* parse the filepath given in command line arguments */
65     filepath = parse_filepath(argc, argv);
66
67     /* create an array with the type of flight */
68     LayoverTable array;
69
70     /* parse the file to fill the array and obtain the actual length */
71     array_from_file(array, filepath);
72
73     /* shows the data on the screen */
74     array_dump(array);
75
76     printf("\nAmount of passengers at %u:00: %u\n", 10,
77           passengers_amount_in_airport(array, 10));
78
79     return (EXIT_SUCCESS);
80 }

```

4 Parcial 1 2022-04-28, Tema B

4.1 Array helpers

1 /*

```

2 @file array_helpers.c
3 @brief Array Helpers method implementation
4 */
5 #include <assert.h>
6 #include <stdbool.h>
7 #include <stdio.h>
8 #include <stdlib.h>
9
10 #include "array_helpers.h"
11
12 /**
13  * @brief returns true when reach last entry in flight table
14  * @return True when is the last entre of the flight table, False
15  *         otherwise
16 */
17 static bool is_last_line(unsigned int hour, unsigned int type) {
18     return hour == HOURS - 1u && type == TYPE - 1u;
19 }
20
21 void array_dump(DeliveryTable a) {
22     for (unsigned int type = 0u; type < TYPE; ++type) {
23         for (unsigned int hour = 0u; hour < HOURS; ++hour) {
24             Flight f = a[type][hour];
25             fprintf(stdout, "%c: flight with %u %s arrived at %u:00", f.code,
26                     f.items_amount, f.type == 0 ? "boxes" : "letters", f.hour - 1)
27             ;
28             if (!is_last_line(hour, type))
29                 {
30                     fprintf(stdout, "\n");
31                 }
32         }
33     }
34 }
35
36 unsigned int extra_space_fee_cost (DeliveryTable a) {
37     unsigned int cost;
38     // Costo de boxes
39     for (unsigned int hrs = 1u; hrs <= HOURS; hrs++){
40         unsigned int cant = a[boxes][hrs].items_amount;
41         if(cant > MAX_ALLOWED_BOXES && hrs < 7){
42             int penalt = (cant - MAX_ALLOWED_BOXES) * BOX_PENALTY;
43             cost = cost + penalt;
44         }
45     }
46 }

```

```

42     }
43 }
44 // Costo de letters
45 for (unsigned int hrs = 1u; hrs <= HOURS; hrs++){
46     unsigned int cant = a[letters][hrs].items_amount;
47     if(cant > MAX_ALLOWED_LETTERS && hrs < 7){
48         int penalt = (cant - MAX_ALLOWED_LETTERS) * LETTER_PENALTY;
49         cost = cost + penalt;
50     }
51 }
52 return cost;
53 }

54
55
56 void array_from_file(DeliveryTable array, const char *filepath) {
57     FILE *file = NULL;
58
59     file = fopen(filepath, "r");
60     if (file == NULL) {
61         fprintf(stderr, "File_does_not_exist.\n");
62         exit(EXIT_FAILURE);
63     }
64
65     char code;
66     int i = 0;
67     while (i != HOURS) {
68         // Lectura del codigo de vuelo
69         int res = fscanf(file, EXPECTED_FLIGHT_FILE_FORMAT , &code);
70         if (res != 1) {
71             fprintf(stderr, "Invalid_file.\n");
72             exit(EXIT_FAILURE);
73         }
74         // Generar ambos Flight
75         Flight flight_boxes = flight_from_file(file, code, boxes);
76         Flight flight_letters = flight_from_file(file, code, letters);
77
78         // Guardo los datos en el arreglo multidimensional
79         array[boxes][flight_boxes.hour - 1] = flight_boxes;
80         array[letters][flight_letters.hour - 1] = flight_letters;
81
82         i++;
83     }
84 }

```

```

1  /*
2     @file array_helpers.h
3     @brief defines array helpers methods. These methods operates over
4           multidimensional array of layover
5  */
6  #ifndef _ARRAY_HELPERS_H
7  #define _ARRAY_HELPERS_H
8
9  #include <stdbool.h>
10 #include "flight.h"
11
12 #define HOURS 24
13 #define TYPE 2
14
15 #define MAX_ALLOWED_BOXES 75
16 #define MAX_ALLOWED_LETTERS 150
17 #define BOX_PENALTY 10
18 #define LETTER_PENALTY 2
19 #define FEE_CLOSE_HOUR 18
20
21 typedef Flight DeliveryTable [TYPE] [HOURS];
22
23 /**
24  * @brief Write the content of the array 'a' into stdout.
25  * @param[in] a array to dump in stdout
26  */
27 void array_dump(DeliveryTable a);
28
29 /**
30  * @brief calculates how much extra fee the company has to pay for the
31  *        day.
32  * @param[in] a array
33  * @details
34  *   Counts items arrived until FEE_CLOSE_HOUR (inclusive).
35  *   For each extra box adds BOX_PENALTY to the fee.
36  *   For each extra letter adds LETTER_PENALTY to the fee.
37  */
38
39 unsigned int extra_space_fee_cost(DeliveryTable a);
40
41 /**
42  * @brief reads an array of layover information from file
43  * @details

```

```

42 *
43 * Each element is read from the file located at 'filepath'.
44 * The file must exist in disk and must have its contents in a sequence
    of
45 * lines, each with the following format:
46 *
47 * <flight_code> <hour> <boxes> <hour> <letters>
48 *
49 * Those elements are copied into the multidimensional array 'a'.
50 * The dimensions of the array are given by the macros noted above.
51 *
52 * @param a array which will contain read file
53 * @param filepath file with layover data
54 */
55 void array_from_file(DeliveryTable a, const char *filepath);
56
57 #endif

```

4.2 Flight

```

1 /*
2  @file layover.c
3  @brief Implements flight structure and methods
4  */
5 #include <stdlib.h>
6 #include "flight.h"
7
8 static const int AMOUNT_OF_FLIGHT_VARS = 2;
9
10 Flight flight_from_file(FILE* file, char code, item_t type)
11 {
12     Flight flight;
13     flight.code = code;
14     flight.type = type;
15
16     // Variable de lectura de datos
17     int read = fscanf(file, "%u%u", &flight.hour, &flight.
        items_amount);
18
19     // Verifico la lectura
20     if (read != AMOUNT_OF_FLIGHT_VARS){
21         fprintf(stderr, "Error de lectura");
22         exit(EXIT_FAILURE);

```

```

23     }
24
25     return flight;
26 }
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```

4.3 Main

```

1  /*
2   @file main.c
3   @brief Defines main program function
4  */
5
6  /* First, the standard lib includes, alphabetically ordered */
7  #include <assert.h>
8  #include <stdio.h>
9  #include <stdlib.h>
10
11 /* Then, this project's includes, alphabetically ordered */
12 #include "array_helpers.h"
13
14 /**
15  * @brief print usage help
16  * @param[in] program_name Executable name
17  */
18 void print_help(char *program_name) {
19     /* Print the usage help of this program. */
20     printf("Usage: %s <input_file_path>\n\n"
21           "Load flight data from a given file in disk.\n"
22           "\n"
23           "The input file must exist in disk and every line in it must"
24           "have the following format:\n\n"
25           "<code><arrival-hour><amount-boxes><arrival-hour><amount-"
26           "letters>\n\n"
27           "Those elements will be copied into the multidimensional"
28           "array 'a'.\n"
29           "\n\n",
30           program_name);
31 }
32
33 /**
34  * @brief reads file path from command line
35  *
36  * @param[in] argc amount of command line arguments
37  * @param[in] argv command line arguments
38  *
39  * @return An string containing read filepath
40  */
41 char *parse_filepath(int argc, char *argv[]) {

```

```

39  /* Parse the filepath given by command line argument. */
40  char *result = NULL;
41
42  if (argc < 2) {
43      print_help(argv[0]);
44      exit(EXIT_FAILURE);
45  }
46
47  result = argv[1];
48
49  return (result);
50 }
51
52 /**
53  * @brief Main program function
54  *
55  * @param[in] argc amount of command line arguments
56  * @param[in] argv command line arguments
57  *
58  * @return EXIT_SUCCESS when programs executes correctly, EXIT_FAILURE
59  *         otherwise
60  */
61 int main(int argc, char *argv[]) {
62     char *filepath = NULL;
63
64     /* parse the filepath given in command line arguments */
65     filepath = parse_filepath(argc, argv);
66
67     /* create an array with the type of flight */
68     DeliveryTable array;
69
70     /* parse the file to fill the array and obtain the actual length */
71     array_from_file(array, filepath);
72
73     /* show the ordered array in the screen */
74     array_dump(array);
75
76     printf("\nExtra fee cost: %u\n", extra_space_fee_cost(array));
77
78     return (EXIT_SUCCESS);
79 }

```