Contents

1 Basic Test Results 2

2 README 4

3 STATEMENT.pdf 7

4 SpreaderDetectorBackend.c 8

1 Basic Test Results

```
*******************************************
*                                         *
*    Hello dear C&C++ Workshop Student,   *
*    We wish you good luck on your exam!  *
*                                         *
*******************************************
Running...

Opening tar file
OK
Tar extracted O.K.

Checking files...
OK
Making sure files are not empty...
OK
Compilation check...

Compiling...
OK

*********************************
*                               *
*     Compilation seems OK!     *
*   Check if you got warnings!  *
*                               *
*********************************

=====================
 Public test cases
=====================

=====================
Running SpreaderDetector - Test # 1
OK
Running diff
OK
Test passed.
=====================

=====================
Running SpreaderDetector - Test # 2
OK
Running diff
OK
Test passed.
=====================

=====================
Running SpreaderDetector - Test # 3
OK
Running diff
OK
Test passed.
=====================

**********************************
*                                *
*   presubmission script passed  *
*        3/3 tests passed         *
*                                 *
***********************************

=========================
= Checking coding style =
=========================
 ** Total Violated Rules      : 0
 ** Total Errors Occurs       : 0
 ** Total Violated Files Count: 0
```

2 README

```
Name:Nitzan Rosen
ID:208564641
CS-USER:nitzanrosen

SpreaderDetectorBackend
-----------------------------
Please explain how you dealt with the following parts of the exam.

Input processing
----------------

--arguments and reading--
first of all, I checked that I had received a suitable number of inputs.
Then, during the reading attempt I tried to open them, and if the file did not exist-
I exited the program in orderly way.
I assumed that the content of the files is correct, as written in the exercise instructions.
I saved the information from the files in a dynamic array of two types that I defined:
People, Meeting.
The memory is released by a special function, which is always called when exiting the program.
Each file was read line-by-line until the end, when I converted each line to the appropriate format.
(separate the different fields by whitespace)

--Error handling hierarchy--
Transfer success / failure status of functions to above level functions.
In cases of functions that cannot return value because they return something else,
I have set a global FLAG variable that will be updated if an error occurs.

--global variables-
I used a global variable for frequently called numeric fields.
It's more convenient than transferring their copies at every functions calling.

--pointers--
I used pointers a lot, to pass references to fields (such as the arrays that store the information).
It helps me access the memory I have allocated from all sorts of places in the code

Data storing
------------

-dynamic array-
I saved the information from the files in a dynamic array of two types that I defined:
People, Meeting.
the size of the array changing dynamically, using realloc when needed.

-Meeting type-
include data from the meeting input file.
it's fields are: spreaderId,InfectedId,distance,time. (data from single meeting)

-People type-
include data from the people input file.
it's include fields of: name, age, id in the array, and of course- "probability to get sick".

each people and meeting is initialize during the reading process.
special function updates the probability from the meeting, for every people.

--string in my code--
I set the field of the person's name to be a char*.
I didn't set a predetermined memory size for it.
During the reading I malloc a memory to storage the name.
the size was the amount of chars I need +1  (for the "\0" character at the end of the string).
```

```
so, that I use exactly the amount of memory I need. and save memory.
Of course at exit the program, I release this memory.

after importing the data I need from the files and storage it, I need to calculate
 the updated probabilities.
first I sort the People array by their id.
after that I iterate the meetings array, and calculate by order the updated probability.
In order to approach the right person and update it's probability,
I use Binary Search on the id field.

When this process done, I have updated probability for every person.

-runtime-
At first, I implemented an algorithm that executes this part in O(m*p^2)
But I was able to reduce the run time by using binary search :)

the runtime of this part: O(m*log(p) + p*log(p))
binary search takes log(n), then update the probability for every person will take: m*log(p)
also, the merge sort will take: p*log(p)

Results sorting
---------------

To return the people according to the chances of infection in descending order,
 I sorted the list of people according to the field of chances of infection.
 (I use the same sorting algorithm as before. the only change is the "merge" part.
   so, I create spacial merge function for every sorting, and call the suit one by enum)

I used a Merge-sort version that sorts in-place.

The sorting is based on the sorting I submitted in the first exercise in this course,
 with adjustments to the current exercise.

The general idea of mergeSort -
a recursive call to sort on sub-arrays of the array that is sorted on it,
until reaching a single cell.
Then connect any two sorted sub-arrays to one array that will be sorted as well.
The depth of the recursion tree is O(logn) and at each stage n work is done,
so in total, the run time of the sorting is O(nlogn)

                                               ./oyyyyyyyyo:`
                                            `/yy+-``   ```-+yy:
                       `-//++++++++/-      -hs.             `-yh-
                     .+o:.`       `./s+.  /m: `:+so+-         `+m:
                   `oo.              `:y+-N: /mNMMMMNh-         om.
                  `y/`                 `ymh -NMMMMMMMMm         `No`
                  yo            ```     `ms :MMMMMMMMMm          dy`        .:.
                 .N`         `+hmmmds-   oh  oNMMMMMMm:         `Nh:------:ddNy`
                 :N         `dMMMMMMMN/  /N:  -+yhhs/.          oNyhdmmmmdhhsms...`
                 -M-        +MMMMMMMMMm  shm:                 `+Nyoooosssoooymhhhhdo.
                  hh`       -NMMMMMMMMs :doyms.             `-ymsooooooooooooosooooyhd/
             `:.` -mh.       :hNMMMNmo`/dsooshdy/-``    `.-+hdyooooooooooooooooooosyNm:
             yhysyyyyh+.       .:/:-.-shoooooooyhdhhyyyyhddhsoooooooooooooooooooooooymd/-.
          `..ddysooooosys+-.````..:oyysoooooooooooooossssoooooooooooooooooooooooooooooohmdm
        ./ymhdysooooooooosyyyysyyyysoooooooooooooooooooooooooooooooooooooooooooooysoosdmNN
      `+mmhsoooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooymsoooshm
      smmdmsoooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooosyhhhsooooos
`-.`.oysyysooooooooooooooooooooooooooooooooooooooooooooosoooooooooooooooooooydmdyoooooooooos
oNmmhyyoooooooooooooooooooooooooooooooooooooooohNNNdhsoohmmddysooooymmmmmmMdsoooooooooooooh
.ddysoooooooooooooooooooooooooooooooooooyhoosmNMMMMMMMNNNMMMMMMNmmmmNMMMMMMmoooooooooooooo
 :yNNhoooooooooooooooooooooooooooooosydMmdmMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMhooooooooooooooo
  oMhoosNdoooooooooooooosdoooohNhyyyhdNMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMyoooooooooooooooh
 omNsoohMNhysoooooooooymMdhdmMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMsoooooooooooos
 sMsoooysshhdddhhhmNNMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMmooooooooooooooo
 oMsoooooooooshmMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMsoooooooooooo
 sMyyooooooooooooymMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMdoooooooooooooooo
 `ymNsooooooooooooohNMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMsooooooooooooohym
  .+MdooooooooooooooyNMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMhoooooooooooosMNy
```

~~~
   `+MhoooooooooooooosNMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMdoooooooooooooosmM+`
    `sMyoooooooooooooooyNMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMhooooooooooooooosMy.
     .hNooooooooooooooosmMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMNyoooooooooooooooooMo`
      :MsossoooooooooooooohNMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMNhsooooooooooooooyssM/
     `yNsyNhooooooooooooooooshNMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMNhsooooooooooooooooooddhMo`
     `dmmNhNooooooooooooooooosymNMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMNdysoooooooooooooooooosdmddh/`
      .:/:.mhoooooooooooooooooooshmNMMMMMMMMMMMMMMMMMMMMMMMMNmdysoooooooooooooooooooydd:```
           .ydsoooooooooooooooooooosyhdmNNNMMMMMMMMMNNNmdhysoooooooooooooooooooyysymh:
            .yNdyyhooooooooooooooooooooosssyyyyyyyysssooooooooooooooooooooooymmmdy/`
             `+hdNNyoooooooooooooooooooooooooooooooooooooooooooooooooooooyNh:-.`
               `./NMmsoooooooooooooooooooooooooooooooooooooooooooooosssoshdd+.
                  -yNmysyyssoooosooooooooooooooooooooooooooooooooshmNdyho:`
                   `:hmNNdmmdhssmmhsooooooooooooooooooooooosoosssdmy/:-``
                      .::.`/ohNhNmmNdhyssyhhsoooooooossydmmdddyss:`
                             `:/:.`-+hNddmhddhhhhddddhs/-....`
                                       ``````````....``
good luck :)
~~~

האוניברסיטה העברית בירושלים
THE HEBREW UNIVERSITY OF JERUSALEM

הצהרה על מקוריות עבודה

(נא לסמן V בכל סעיף)

☑ אני מצהיר/ה בזאת שפתרון המבחן/מטלה בכתב בקורס 67315 – "PROGRAMMING WORKSHOP IN C & C++" שהתקיים בתאריך 02.08 - 04.08 הינו פרי עבודתי המקורית ולא הועתק מתלמיד אחר. כמו כן, לא נעזרתי בחומר שאינו מאושר על-ידי מורי הקורס.

☑ אני מצהיר/ה בזאת שלא עזרתי לאף תלמיד/ה בפתרון הבחינה/המטלה.

☑ אני מצהיר/ה בזאת כי ידוע לי שאם יתגלה כי עברתי עבירת העתקה מסוג זה, תוגש נגדי תלונה על כך לוועדת המשמעת של האוניברסיטה העברית.

☑ אני מבין שלמסמך זה תוקף משפטי על כל ההשלכות האמורות מכך.

03.08	ניצן	רצן	[illegible]
תאריך	שם פרטי	שם משפחה	חתימה

4 SpreaderDetectorBackend.c

```c
//
// Created by nitzan Rosen on 01/08/2020.
//

// ------------------------------ includes ------------------------------
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>
#include <stdbool.h>
#include "SpreaderDetectorParams.h"

// -------------------------- const definitions -------------------------
#define INPUT_ERR "Usage: ./SpreaderDetectorBackend <Path to People.in> <Path to Meetings.in>\n"
#define INPUT_ERR1 "Error in input files.\n"
#define OUTPUT_ERR "Error in output file.\n"

#define INPUT_LINE_LENGTH 1024
#define NULL_SPREADER_ID 42
#define NULL_PROBABILITY_TO_SICK 0
#define NULL_NAME " "
#define EOS '\0'
#define ZERO 0
#define ONE 1
#define ALLOC_SIZE 20
#define NOT_IN_ARRAY -1

// -------------------------- headers ---------------------------------
/**
 * type for People struct
 */
typedef struct People
{
    char *name; //people name
    unsigned long int id;
    float age;
    float probabilityToSick; //probability of sickness. in range [0,1]
} People;

/**
 * type for Meeting struct
 */
typedef struct Meeting
{
    unsigned long int spreaderId;
    unsigned long int InfectedId;
    float distance;
    float time;
} Meeting;

/**
 * enum for the type of sorting.
 */
typedef enum
{
    ID,
    Probability
} SortingParam;

```

```c
void freeAllocations(People **PeoplesArray, Meeting **MeetingsArray);

int readMeetingFile(char meetingFile[], Meeting **MeetingsArray);

int readPeopleFile(char peopleFile[], People **PeopleArray);

Meeting transformMeetingLines(char *line);

People transformPeopleLines(char *line);

float transformStringToFloat(char *line);

float crna(float dist, float time);

int writeToOutput(People **PeopleArray);

int updateProbabilityForEverPeople(People **PeopleArray, Meeting **MeetingsArray);

int calculateTotalProbability(People **PeopleArray, Meeting **MeetingsArray);

void mergeSort(People **pPeopleArray, int left, int right, SortingParam param);

int binarySearch(People ***Array, int left, int right, unsigned long int id);

unsigned long int transformStringToInt(char *line);

float findPreviousProbability(People ***PeopleArray, unsigned long int previousId);

int writeToEmptyOutput();

int handleEmptyFiles(People **PeoplesArray);

int writeToEmptyOutput1(People ***PeopleArray);

void mergeById(People ***Array, int left, int mid, int right);

// -------------------------- global --------------------------------
unsigned long int sickId = 0; // global variable of sick man ID
int gPeoplesCounter; // global variable of number of proples
int gMeetingsCounter; // global variable of number of meetings
int errorHappened = 0; //flag for error
bool emptyMeetingFlag = false; //flag for error
bool emptyPeopleFlag = false; //flag for error
bool elementNotInArray = false; //flag for error

// -------------------------- code --------------------------------

/**
 * main funtion of the program
 * @param argc
 * @param argv
 * @return exit code
 */
int main(int argc, char *argv[])
{
    //check program input correctness (number of arguments, valid files, etc.)
    if (argc != 3)
    {
        fprintf(stderr, INPUT_ERR);
        return EXIT_FAILURE; //1
    }
    //if we arrive here inputs files are ok :)

    //alloc my DS for storage data from input files
    People *PeoplesArray = NULL;
    Meeting *MeetingsArray = NULL;

    //now read files.
    if (readMeetingFile(argv[2], &MeetingsArray) == EXIT_FAILURE ||
        readPeopleFile(argv[1], &PeoplesArray) == EXIT_FAILURE) //reading went well?
    {
        freeAllocations(&PeoplesArray, &MeetingsArray);
        return EXIT_FAILURE;
    }
    if (emptyPeopleFlag || emptyMeetingFlag) //empty files?
    {
        int stat = handleEmptyFiles(&PeoplesArray);
        freeAllocations(&PeoplesArray, &MeetingsArray);
        return stat;
    }
    //update probability
    mergeSort(&PeoplesArray, 0, gPeoplesCounter - 1, ID);
    updateProbabilityForEverPeople(&PeoplesArray, &MeetingsArray);
    calculateTotalProbability(&PeoplesArray, &MeetingsArray);

    mergeSort(&PeoplesArray, 0, gPeoplesCounter - 1, Probability);

    if (writeToOutput(&PeoplesArray) == EXIT_FAILURE)
    {
        freeAllocations(&PeoplesArray, &MeetingsArray);
        return EXIT_FAILURE;
    }
    freeAllocations(&PeoplesArray, &MeetingsArray);
    return EXIT_SUCCESS;
}

/**
 * call the suitable action by the empty file status
 * @param PeoplesArray
 * @return status of success
 */
int handleEmptyFiles(People **PeoplesArray)
{
    if (emptyMeetingFlag && emptyPeopleFlag)
    {
        return (writeToEmptyOutput());
    }
    else if (emptyMeetingFlag == true && emptyPeopleFlag == false)
    {
        return (writeToEmptyOutput1(&PeoplesArray));
    }
    return EXIT_SUCCESS;
}

/**
 * write yo output in case both files are empty
 * @return status of success
 */
int writeToEmptyOutput()
{
    FILE *fp = fopen(OUTPUT_FILE, "w");
    if (fp == NULL)
    {
        fprintf(stderr, OUTPUT_ERR);
        return EXIT_FAILURE;
    }
    fclose(fp);
    return EXIT_SUCCESS;
}

/**
 * write to output in case the was not sick people at all
 * @param PeopleArray
 * @return status of success
 */
int writeToEmptyOutput1(People ***PeopleArray)
```

```c
{
    FILE *fp = fopen(OUTPUT_FILE, "w");
    if (fp == NULL)
    {
        fprintf(stderr, OUTPUT_ERR);
        return EXIT_FAILURE;
    }
    for (int p = 0; p < gPeoplesCounter; ++p)
    {
        fprintf(fp, CLEAN_MSG, (*(*PeopleArray))[p].name, (*(*PeopleArray))[p].id);
    }
    fclose(fp);
    return EXIT_SUCCESS;
}

/**
 * allocate memory for the meeting file
 * open file to reading
 * read the meetings file into tje program
 * @param meetingFile
 * @param MeetingsArray
 * @return status of success
 */
int readMeetingFile(char meetingFile[], Meeting **MeetingsArray)
{
    /* dynamic allocation settings */
    int meetingsCapacity = ALLOC_SIZE;
    *MeetingsArray = (Meeting *)calloc(meetingsCapacity, sizeof(Meeting));
    if (*MeetingsArray == NULL)
    {
        fprintf(stderr, STANDARD_LIB_ERR_MSG);
        return EXIT_FAILURE;
    }

    /* open file to reading */
    FILE *fp = fopen(meetingFile, "r"); //open file for reading
    if (fp == NULL) //fail opening file
    {
        fprintf(stderr, INPUT_ERR1);
        return EXIT_FAILURE;
    }

    char line[INPUT_LINE_LENGTH] = {EOS};
    int meetingsCounter = -1; //counter of the meetings number
    while ( fgets(line, sizeof(line), fp) != NULL)
    {
        /* change the array size if needed */
        if (meetingsCounter == meetingsCapacity)
        {
            meetingsCapacity += ALLOC_SIZE;
            *MeetingsArray = (Meeting *)realloc(*MeetingsArray, sizeof(Meeting) * meetingsCapacity);
            if (*MeetingsArray == NULL)
            {
                fprintf(stderr, STANDARD_LIB_ERR_MSG);
                fclose(fp);
                return EXIT_FAILURE;
            }
        }

        if (meetingsCounter == -1)
        {
            unsigned long int i = transformStringToInt(line);
            sickId = i;
        }
        else
        {
            /* check the line and add it to the array */
            Meeting myMeeting;
            myMeeting = transformMeetingLines(line);
            (*MeetingsArray)[meetingsCounter].time = myMeeting.time;
            (*MeetingsArray)[meetingsCounter].distance = myMeeting.distance;
            (*MeetingsArray)[meetingsCounter].InfectedId = myMeeting.InfectedId;
            (*MeetingsArray)[meetingsCounter].spreaderId = myMeeting.spreaderId;
        }
        meetingsCounter++; //raise the counter by 1
    }
    if (meetingsCounter == -1 && (feof(fp) == true)) //empty file case
    {
        emptyMeetingFlag = true;
    }
    gMeetingsCounter = meetingsCounter;
    fclose(fp);
    return EXIT_SUCCESS;
}

/**
 * allocate memory for the people file
 * open file to reading
 * read the people file into the program
 * @param peopleFile
 * @param PeopleArray
 * @return
 */
int readPeopleFile(char peopleFile[], People **PeopleArray)
{
    /* dynamic allocation settings */
    int peoplesCapacity = ALLOC_SIZE;
    *PeopleArray = (People *)calloc(peoplesCapacity, sizeof(People));
    if (*PeopleArray == NULL)
    {
        fprintf(stderr, STANDARD_LIB_ERR_MSG);
        return EXIT_FAILURE;
    }

    /* open file to reading */
    FILE *fp = fopen(peopleFile, "r"); //open file for reading
    if (fp == NULL) //fail opening file
    {
        fprintf(stderr, INPUT_ERR1);
        return EXIT_FAILURE;
    }

    char line[INPUT_LINE_LENGTH] = {EOS};
    int peopleCounter = 0; //counter of the meetings number
    while ( fgets(line, sizeof(line), fp) != NULL)
    {
        /* change the array size if needed */
        if (peopleCounter == peoplesCapacity)
        {
            peoplesCapacity += ALLOC_SIZE;
            *PeopleArray = (People *)realloc(*PeopleArray, sizeof(People) * peoplesCapacity);
            if (*PeopleArray == NULL)
            {
                fprintf(stderr, STANDARD_LIB_ERR_MSG);
                fclose(fp);
                return EXIT_FAILURE;
            }
        }

        /* check the line and add it to the array */
        People myPeople;
        myPeople = transformPeopleLines(line);
        if (errorHappened == ONE) //handle with fail of strncpy inside transformPeopleLines
        {
            fprintf(stderr, STANDARD_LIB_ERR_MSG);
            fclose(fp);
            return EXIT_FAILURE;
        }
        (*PeopleArray)[peopleCounter].probabilityToSick = myPeople.probabilityToSick;
        (*PeopleArray)[peopleCounter].id = myPeople.id;
        (*PeopleArray)[peopleCounter].name = myPeople.name;
        (*PeopleArray)[peopleCounter].age = myPeople.age;

        peopleCounter++; //raise the counter by 1
    }
    if (peopleCounter == 0 && (feof(fp) == true)) //empty file case
    {
        emptyPeopleFlag = true;
    }

    gPeoplesCounter = peopleCounter;
    fclose(fp);
    return EXIT_SUCCESS;
}

/**
 * free the dynamic allocation of the given pointers
 * @param PeoplesArray
 * @param MeetingsArray
 */
void freeAllocations(People **PeoplesArray, Meeting **MeetingsArray)
{
    if (*PeoplesArray != NULL)
    {
        for (int p = 0; p < gPeoplesCounter; p++)
        {
            free((*PeoplesArray)[p].name);
        }
        free(*PeoplesArray);
    }
    if (*MeetingsArray != NULL)
    {
        free(*MeetingsArray);
    }
    *PeoplesArray = NULL;
    *MeetingsArray = NULL;
}

/**
 * transform string to int
 * @param line
 * @return int val of the given string
 */
unsigned long int transformStringToInt(char *line)
{
    char *end;
    unsigned long int i = strtoul(line, &end, 10);
    return i;
}

/**
 * transform string to int
 * @param line
 * @return float val of the given string
 */
float transformStringToFloat(char *line)
{
    char *end;
    float f = strtof(line, &end);
    return f;
}

/**
 * create Meeting struct from the given details
 * @param line
 * @return Meeting struct
 */
Meeting transformMeetingLines(char *line)
{
    char *tok;
    tok = strtok(line, " ,-");
    int counter = 0;

    Meeting myMeeting = {NULL_SPREADER_ID, NULL_SPREADER_ID, ZERO, ZERO}; //default vals
    while ( tok != NULL)
    {
        if (counter == 0) //<infector_id>
        {
            unsigned long int i = transformStringToInt(tok);
            myMeeting.spreaderId = i;
        }
        else if (counter == 1) //<infected_id>
        {
            unsigned long int i = transformStringToInt(tok);
            myMeeting.InfectedId = i;
        }
        else if (counter == 2) //<distance>
        {
            float f = transformStringToFloat(tok);
            myMeeting.distance = f;
        }
        else if (counter == 3) //<time>
        {
            float f = transformStringToFloat(tok);
            myMeeting.time = f;
        }
        counter++;
        tok = strtok(NULL, " ,-");
    }
    return myMeeting;
}

/**
 * create People struct from the given details
 * @param line
 * @param counterVal
 * @return People struct
 */
People transformPeopleLines(char *line)
{
    //<Person Name> <Person ID> <Person age>\n

    char *tok;
    tok = strtok(line, " ,-");
    int counter = 0;
    People myPeople;
    while ( tok != NULL)
    {
        if (counter == 0) //<Person Name>
        {
            myPeople.name = (char *)malloc((strlen(tok) + 1) * sizeof(char));
            if (myPeople.name == NULL)
            {
                errorHappened = ONE;
                myPeople.name = NULL_NAME;
            }
            strcpy(myPeople.name, tok);
            if (myPeople.name == NULL)
            {
                errorHappened = ONE;
                myPeople.name = NULL_NAME;
            }
        }
        else if (counter == 1) //<Person ID>
        {
            unsigned long int i = transformStringToInt(tok);
            myPeople.id = i;
        }
        else if (counter == 2) //<Person age>
        {
            float f = transformStringToFloat(tok);
            myPeople.age = f;
        }
        counter++;
        tok = strtok(NULL, " ,-");
    }
    myPeople.probabilityToSick = NULL_PROBABILITY_TO_SICK;

    return myPeople;
}

/**
 * Gets information about a meeting between two people,
 * and calculates the chance that one of them will infect the other in Corona
 * @param dist
 * @param time
 * @return float of the probability
 */
float crna(float dist, float time)
{
    if (dist == ZERO)
    {
        return 1;
    }
    return (time * MIN_DISTANCE) / (dist * MAX_TIME);
}

/**
 * update the probability field by iteration on the meetings array:
 * @param PeopleArray
 * @param MeetingsArray
 */
int updateProbabilityForEverPeople(People **PeopleArray, Meeting **MeetingsArray)
{
    for (int m = 0; m < gMeetingsCounter; ++m)
    {
        //calc prob by crna
        float currentProbability = crna((*MeetingsArray)[m].distance, (*MeetingsArray)[m].time);
        int index = binarySearch(&PeopleArray, 0, gPeoplesCounter - 1,
                                 (*MeetingsArray)[m].InfectedId);
        if (index == NOT_IN_ARRAY)
        {
            //if we arrive here, we had id in meeting that is not appear in people.
            //this dosen't sppose to happen, by the Guidelines
            // but I decide to prevent option of index-out-of-range error In my tests. so It's here.
            continue;
        }
        //update the probability for the infected in this meeting
        (*PeopleArray)[index].probabilityToSick = currentProbability;
    }
    // update sick people probability:
    int index = binarySearch(&PeopleArray, 0, gPeoplesCounter - 1, sickId);
    if (index == NOT_IN_ARRAY)
    {
        //if we arrive here, the spreader id not appear in people input file.
        //this dosen't suppose to happen, by the Guidelines. but its here for any case..
        return EXIT_FAILURE;
    }
    (*PeopleArray)[index].probabilityToSick = ONE;
    return EXIT_SUCCESS;
```

```c
}

/**
 * update the final probability for every person, by the infection order (in meeting file)
 * @param PeopleArray
 * @param MeetingsArray
 */
int calculateTotalProbability(People **PeopleArray, Meeting **MeetingsArray)
{
    //updateProbabilityForEverPeople run before
    //we assume the meeting are ordered by they real hierarchy.
    for (int m = 0; m < gMeetingsCounter; ++m) //for every meeting
    {
        int index = binarySearch(&PeopleArray, 0, gPeoplesCounter - 1,
                                 (*MeetingsArray)[m].InfectedId);
        if (index == NOT_IN_ARRAY)
        {
            //if we arrive here, we had id in meeting that is not appear in people.
            //this dosen't suppose to happen, by the Guidelines
            // but I decide to prevent option of index-out-of-range error In my tests. so It's here.
            continue;
        }
        float currentProbability = (*PeopleArray)[index].probabilityToSick;
        //update its val by its previous
        float previousProb = findPreviousProbability(&PeopleArray, (*MeetingsArray)[m].spreaderId);
        if (elementNotInArray == true)
        {
            //same as above.
            continue;
        }
        (*PeopleArray)[index].probabilityToSick = currentProbability * previousProb;
    }
    return EXIT_SUCCESS;
}

/**
 * open output file to write, and write inside it the given params
 * @param PeopleArray
 * @return EXIT_SUCCESS if the process ends successfully, and EXIT_FAILURE else
 */
int writeToOutput(People **PeopleArray)
{
    FILE *fp = fopen(OUTPUT_FILE, "w");
    if (fp == NULL)
    {
        fprintf(stderr, OUTPUT_ERR);
        return EXIT_FAILURE;
    }
    for (int p = gPeoplesCounter - 1; p >= 0; p--)
    {
        float probability = (*PeopleArray)[p].probabilityToSick;
        if (probability >= MEDICAL_SUPERVISION_THRESHOLD)
        {
            fprintf(fp, MEDICAL_SUPERVISION_THRESHOLD_MSG, (*PeopleArray)[p].name,
                    (*PeopleArray)[p].id);
        }
        else if (probability >= REGULAR_QUARANTINE_THRESHOLD)
        {
            fprintf(fp, REGULAR_QUARANTINE_MSG, (*PeopleArray)[p].name, (*PeopleArray)[p].id);
        }
        else
        {
            fprintf(fp, CLEAN_MSG, (*PeopleArray)[p].name, (*PeopleArray)[p].id);
        }
    }
    fclose(fp);
    return EXIT_SUCCESS;
}
```

```c

/**
 * mergeByProbability two subarrays- Array[l..m] and Array[m+1..r] , compare by probability
 * @param Array
 * @param left
 * @param mid
 * @param right
 */
void mergeByProbability(People ***Array, int left, int mid, int right)
{
    int startIndex = mid + 1; // the first element after mid index

    if ((*(*Array))[mid].probabilityToSick <=
        (*(*Array))[startIndex].probabilityToSick) //array is already sorted :)
    {
        return;
    }

    while ( left <= mid && startIndex <= right )
    {
        if ((*(*Array))[left].probabilityToSick <=
            (*(*Array))[startIndex].probabilityToSick) // element is in the right place
        {
            left++;
        }
        else // If element not in right place. replace in-place
        {
            People temporaryPeople =
                    {
                            (*(*Array))[startIndex].name,
                            (*(*Array))[startIndex].id, (*(*Array))[startIndex].age,
                            (*(*Array))[startIndex].probabilityToSick
                    };
            int index = startIndex;

            while ( index != left ) //move elements between the current indexes
            {
                (*(*Array))[index].name = (*(*Array))[index - 1].name;
                (*(*Array))[index].id = (*(*Array))[index - 1].id;
                (*(*Array))[index].age = (*(*Array))[index - 1].age;
                (*(*Array))[index].probabilityToSick = (*(*Array))[index - 1].probabilityToSick;
                index--;
            }
            //return our value to array
            (*(*Array))[left].name = temporaryPeople.name;
            (*(*Array))[left].id = temporaryPeople.id;
            (*(*Array))[left].age = temporaryPeople.age;
            (*(*Array))[left].probabilityToSick = temporaryPeople.probabilityToSick;

            // Update the pointers
            left++;
            mid++;
            startIndex++;
        }
    }
}

/**
 * sorting the given array with mergeByProbability-sort, in place
 * @param Array
 * @param left
 * @param right
 */
void mergeSort(People **pPeopleArray, int left, int right, SortingParam param)
{
    if (left < right)
    {
        int mid = left + (right - left) / 2;
        mergeSort(&*pPeopleArray, left, mid, param); //sort subArray- Array[left....mid]
        mergeSort(&*pPeopleArray, mid + 1, right, param); //sort subArray- Array[mid....right]
        if (param == Probability)
        {
            mergeByProbability(&pPeopleArray, left, mid,
                               right); //mergeByProbability two sorted subArrays
        }
        else if (param == ID)
        {
            mergeById(&pPeopleArray, left, mid, right);
        }
    }
}

/**
 * return the probability of the given "previous id"
 * @param PeopleArray
 * @param previousId
 * @return the probability(0,1) in sucecess, 0 in fail, 1 if we got the id of the spreader.
 */
float findPreviousProbability(People ***PeopleArray, unsigned long int previousId)
{
    //if prev is the spreader, return 1
    if (previousId == sickId)
    {
        return ONE;
    }
    else
    {
        int index = binarySearch(&*PeopleArray, 0, gPeoplesCounter - 1, previousId);
        if (index == NOT_IN_ARRAY)
        {
            //if we arrive here, we had id in meeting that is not appear in people.
            //this dosen't sppose to happen, by the Guidelines
            // but I decide to prevent option of index-out-of-range error In my tests. so It's here.
            elementNotInArray = true;
            return NOT_IN_ARRAY;
        }
        return (*(*PeopleArray))[index].probabilityToSick; //return its probability
    }
}

int binarySearch(People ***Array, int left, int right, unsigned long int id)
{
    if (right < left)//x is not in array
    {
        return NOT_IN_ARRAY;
    }
    int mid = left + (right - left) / 2;
    if ((*(*Array))[mid].id == id) //x is present at the middle
    {
        return mid;
    }
    if ((*(*Array))[mid].id > id) // If x is smaller than mid
    {
        return binarySearch(Array, left, mid - 1, id);
    }
    return binarySearch(Array, mid + 1, right, id); //else x is grater than mid
}

/**
 * mergeById two subarrays- Array[l..m] and Array[m+1..r], compartion by id
 * @param Array
 * @param left
 * @param mid
 * @param right
 */
void mergeById(People ***Array, int left, int mid, int right)
```

```
{
    int startIndex = mid + 1; // the first element after mid index

    if ((*(*Array))[mid].id <=
        (*(*Array))[startIndex].id) //array is already sorted :)
    {
        return;
    }

    while ( left <= mid && startIndex <= right )
    {
        if ((*(*Array))[left].id <=
            (*(*Array))[startIndex].id) // element is in the right place
        {
            left++;
        }
        else // If element not in right place. replace in-place
        {
            People temporaryPeople =
                    {
                            (*(*Array))[startIndex].name,
                            (*(*Array))[startIndex].id, (*(*Array))[startIndex].age,
                            (*(*Array))[startIndex].probabilityToSick
                    };
            int index = startIndex;

            while ( index != left ) //move elements between the current indexes
            {
                (*(*Array))[index].name = (*(*Array))[index - 1].name;
                (*(*Array))[index].id = (*(*Array))[index - 1].id;
                (*(*Array))[index].age = (*(*Array))[index - 1].age;
                (*(*Array))[index].probabilityToSick = (*(*Array))[index - 1].probabilityToSick;
                index--;
            }
            //return our value to array
            (*(*Array))[left].name = temporaryPeople.name;
            (*(*Array))[left].id = temporaryPeople.id;
            (*(*Array))[left].age = temporaryPeople.age;
            (*(*Array))[left].probabilityToSick = temporaryPeople.probabilityToSick;

            // Update the pointers
            left++;
            mid++;
            startIndex++;
        }
    }
}
```

Index of comments

5.1 nice :)

11.1 -3/-3 realloc can fail, you should assign the result to a different pointer object (code='realloc_to_self')

12.1 -2/-2 The function is too long. Rule of thumb - "a function should not be bigger than your head" (code='err_code_too_long_function')