

Contents

1 Basic Test Results	2
2 README	4
3 STATEMENT.pdf	7
4 SpreaderDetectorBackend.c	8

1 Basic Test Results

```
1 *****
2 *
3 *   Hello dear C&C++ Workshop Student,   *
4 *   We wish you good luck on your exam!  *
5 *
6 *****
7 Running...
8
9 Opening tar file
10 OK
11 Tar extracted O.K.
12
13 Checking files...
14 OK
15 Making sure files are not empty...
16 OK
17 Compilation check...
18
19 Compiling...
20 OK
21
22 *****
23 *
24 *   Compilation seems OK!   *
25 *   Check if you got warnings! *
26 *
27 *****
28
29 =====
30 Public test cases
31 =====
32
33 =====
34 Running SpreaderDetector - Test # 1
35 OK
36 Running diff
37 OK
38 Test passed.
39 =====
40
41 =====
42 Running SpreaderDetector - Test # 2
43 OK
44 Running diff
45 OK
46 Test passed.
47 =====
48
49 =====
50 Running SpreaderDetector - Test # 3
51 OK
52 Running diff
53 OK
54 Test passed.
55 =====
56
57 *****
58 *
59 *   presubmission script passed   *
```

```
60 *          3/3 tests passed          *
61 *                                     *
62 *****
63
64 =====
65 = Checking coding style =
66 =====
67 ** Total Violated Rules      : 0
68 ** Total Errors Occurs      : 0
69 ** Total Violated Files Count: 0
```

2 README

```
1 Name:Nitzan Rosen
2 ID:208564641
3 CS-USER:nitzanrosen
4
5 SpreaderDetectorBackend
6 -----
7 Please explain how you dealt with the following parts of the exam.
8
9 Input processing
10 -----
11
12 --arguments and reading--
13 first of all, I checked that I had received a suitable number of inputs.
14 Then, during the reading attempt I tried to open them, and if the file did not exist-
15 I exited the program in orderly way.
16 I assumed that the content of the files is correct, as written in the exercise instructions.
17 I saved the information from the files in a dynamic array of two types that I defined:
18 People, Meeting.
19 The memory is released by a special function, which is always called when exiting the program.
20 Each file was read line-by-line until the end, when I converted each line to the appropriate format.
21 (separate the different fields by whitespace)
22
23 --Error handling hierarchy--
24 Transfer success / failure status of functions to above level functions.
25 In cases of functions that cannot return value because they return something else,
26 I have set a global FLAG variable that will be updated if an error occurs.
27
28 --global variables-
29 I used a global variable for frequently called numeric fields.
30 It's more convenient than transferring their copies at every functions calling.
31
32 --pointers--
33 I used pointers a lot, to pass references to fields (such as the arrays that store the information).
34 It helps me access the memory I have allocated from all sorts of places in the code
35
36 Data storing
37 -----
38
39 -dynamic array-
40 I saved the information from the files in a dynamic array of two types that I defined:
41 People, Meeting.
42 the size of the array changing dynamically, using realloc when needed.
43
44 -Meeting type-
45 include data from the meeting input file.
46 it's fields are: spreaderId,InfectedId,distance,time. (data from single meeting)
47
48 -People type-
49 include data from the people input file.
50 it's include fields of: name, age, id in the array, and of course- "probability to get sick".
51
52 each people and meeting is initialize during the reading process.
53 special function updates the probability from the meeting, for every people.
54
55 --string in my code--
56 I set the field of the person's name to be a char*.
57 I didn't set a predetermined memory size for it.
58 During the reading I malloc a memory to storage the name.
59 the size was the amount of chars I need +1 (for the "\0" character at the end of the string).
```

```

60 so, that I use exactly the amount of memory I need. and save memory.
61 Of course at exit the program, I release this memory.
62
63 after importing the data I need from the files and storage it, I need to calculate
64 the updated probabilities.
65 first I sort the People array by their id.
66 after that I iterate the meetings array, and calculate by order the updated probability.
67 In order to approach the right person and update it's probability,
68 I use Binary Search on the id field.
69
70 When this process done, I have updated probability for every person.
71
72 -runtime-
73 At first, I implemented an algorithm that executes this part in  $O(m \cdot p^2)$ 
74 But I was able to reduce the run time by using binary search :)
75
76 the runtime of this part:  $O(m \cdot \log(p) + p \cdot \log(p))$ 
77 binary search takes  $\log(n)$ , then update the probability for every person will take:  $m \cdot \log(p)$ 
78 also, the merge sort will take:  $p \cdot \log(p)$ 
79
80 Results sorting
81 -----
82
83 To return the people according to the chances of infection in descending order,
84 I sorted the list of people according to the field of chances of infection.
85 (I use the same sorting algorithm as before. the only change is the "merge" part.
86 so, I create spacial merge function for every sorting, and call the suit one by enum)
87
88 I used a Merge-sort version that sorts in-place.
89
90 The sorting is based on the sorting I submitted in the first exercise in this course,
91 with adjustments to the current exercise.
92
93 The general idea of mergeSort -
94 a recursive call to sort on sub-arrays of the array that is sorted on it,
95 until reaching a single cell.
96 Then connect any two sorted sub-arrays to one array that will be sorted as well.
97 The depth of the recursion tree is  $O(\log n)$  and at each stage  $n$  work is done,
98 so in total, the run time of the sorting is  $O(n \log n)$ 
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
9
```




הצהרה על מקורות 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.

חתימה

שם משפחה

שם פרטי

תאריך

4 SpreaderDetectorBackend.c

```
1  //
2  // Created by nitzan Rosen on 01/08/2020.
3  //
4
5  // ----- includes -----
6  #include <stdio.h>
7  #include <string.h>
8  #include <stdlib.h>
9  #include <ctype.h>
10 #include <stdbool.h>
11 #include "SpreaderDetectorParams.h"
12
13 // ----- const definitions -----
14 #define INPUT_ERR "Usage: ./SpreaderDetectorBackend <Path to People.in> <Path to Meetings.in>\n"
15 #define INPUT_ERR1 "Error in input files.\n"
16 #define OUTPUT_ERR "Error in output file.\n"
17
18 #define INPUT_LINE_LENGTH 1024
19 #define NULL_SPREADER_ID 42
20 #define NULL_PROBABILITY_TO_SICK 0
21 #define NULL_NAME " "
22 #define EOS '\0'
23 #define ZERO 0
24 #define ONE 1
25 #define ALLOC_SIZE 20
26 #define NOT_IN_ARRAY -1
27
28 // ----- headers -----
29 /**
30  * type for People struct
31  */
32 typedef struct People
33 {
34     char *name; //people name
35     unsigned long int id;
36     float age;
37     float probabilityToSick; //probability of sickness. in range [0,1]
38 } People;
39
40 /**
41  * type for Meeting struct
42  */
43 typedef struct Meeting
44 {
45     unsigned long int spreaderId;
46     unsigned long int InfectedId;
47     float distance;
48     float time;
49 } Meeting;
50
51 /**
52  * enum for the type of sorting.
53  */
54 typedef enum
55 {
56     ID,
57     Probability
58 } SortingParam;
```



```

60 void freeAllocations(People **PeoplesArray, Meeting **MeetingsArray);
61
62 int readMeetingFile(char meetingFile[], Meeting **MeetingsArray);
63
64 int readPeopleFile(char peopleFile[], People **PeopleArray);
65
66 Meeting transformMeetingLines(char *line);
67
68 People transformPeopleLines(char *line);
69
70 float transformStringToFloat(char *line);
71
72 float crna(float dist, float time);
73
74 int writeToOutput(People **PeopleArray);
75
76 int updateProbabilityForEverPeople(People **PeopleArray, Meeting **MeetingsArray);
77
78 int calculateTotalProbability(People **PeopleArray, Meeting **MeetingsArray);
79
80 void mergeSort(People **pPeopleArray, int left, int right, SortingParam param);
81
82 int binarySearch(People ***Array, int left, int right, unsigned long int id);
83
84 unsigned long int transformStringToInt(char *line);
85
86 float findPreviousProbability(People ***PeopleArray, unsigned long int previousId);
87
88 int writeToEmptyOutput();
89
90 int handleEmptyFiles(People **PeoplesArray);
91
92 int writeToEmptyOutput1(People ***PeopleArray);
93
94 void mergeById(People ***Array, int left, int mid, int right);
95
96 // ----- global -----
97 unsigned long int sickId = 0; // global variable of sick man ID
98 int gPeoplesCounter; // global variable of number of peoples
99 int gMeetingsCounter; // global variable of number of meetings
100 int errorHappened = 0; //flag for error
101 bool emptyMeetingFlag = false; //flag for error
102 bool emptyPeopleFlag = false; //flag for error
103 bool elementNotInArray = false; //flag for error
104
105 // ----- code -----
106
107 /**
108  * main function of the program
109  * @param argc
110  * @param argv
111  * @return exit code
112  */
113 int main(int argc, char *argv[])
114 {
115     //check program input correctness (number of arguments, valid files, etc.)
116     if (argc != 3)
117     {
118         fprintf(stderr, INPUT_ERR);
119         return EXIT_FAILURE; //1
120     }
121     //if we arrive here inputs files are ok :)
122
123     //alloc my DS for storage data from input files
124     People *PeoplesArray = NULL;
125     Meeting *MeetingsArray = NULL;
126
127     //now read files.

```

```

128     if (readMeetingFile(argv[2], &MeetingsArray) == EXIT_FAILURE ||
129         readPeopleFile(argv[1], &PeoplesArray) == EXIT_FAILURE) //reading went well?
130     {
131         freeAllocations(&PeoplesArray, &MeetingsArray);
132         return EXIT_FAILURE;
133     }
134     if (emptyPeopleFlag || emptyMeetingFlag) //empty files?
135     {
136         int stat = handleEmptyFiles(&PeoplesArray);
137         freeAllocations(&PeoplesArray, &MeetingsArray);
138         return stat;
139     }
140     //update probability
141     mergeSort(&PeoplesArray, 0, gPeoplesCounter - 1, ID);
142     updateProbabilityForEverPeople(&PeoplesArray, &MeetingsArray);
143     calculateTotalProbability(&PeoplesArray, &MeetingsArray);
144
145     mergeSort(&PeoplesArray, 0, gPeoplesCounter - 1, Probability);
146
147     if (writeToOutput(&PeoplesArray) == EXIT_FAILURE)
148     {
149         freeAllocations(&PeoplesArray, &MeetingsArray);
150         return EXIT_FAILURE;
151     }
152     freeAllocations(&PeoplesArray, &MeetingsArray);
153     return EXIT_SUCCESS;
154 }
155
156 /**
157  * call the suitable action by the empty file status
158  * @param PeoplesArray
159  * @return status of success
160  */
161 int handleEmptyFiles(People **PeoplesArray)
162 {
163     if (emptyMeetingFlag && emptyPeopleFlag)
164     {
165         return (writeToEmptyOutput());
166     }
167     else if (emptyMeetingFlag == true && emptyPeopleFlag == false)
168     {
169         return (writeToEmptyOutput1(&PeoplesArray));
170     }
171     return EXIT_SUCCESS;
172 }
173
174 /**
175  * write yo output in case both files are empty
176  * @return status of success
177  */
178 int writeToEmptyOutput()
179 {
180     FILE *fp = fopen(OUTPUT_FILE, "w");
181     if (fp == NULL)
182     {
183         fprintf(stderr, OUTPUT_ERR);
184         return EXIT_FAILURE;
185     }
186     fclose(fp);
187     return EXIT_SUCCESS;
188 }
189
190 /**
191  * write to output in case the was not sick people at all
192  * @param PeopleArray
193  * @return status of success
194  */
195 int writeToEmptyOutput1(People ***PeopleArray)

```

```

196 {
197     FILE *fp = fopen(OUTPUT_FILE, "w");
198     if (fp == NULL)
199     {
200         fprintf(stderr, OUTPUT_ERR);
201         return EXIT_FAILURE;
202     }
203     for (int p = 0; p < gPeoplesCounter; ++p)
204     {
205         fprintf(fp, CLEAN_MSG, ((*PeopleArray))[p].name, ((*PeopleArray))[p].id);
206     }
207     fclose(fp);
208     return EXIT_SUCCESS;
209 }
210
211 /**
212  * allocate memory for the meeting file
213  * open file to reading
214  * read the meetings file into the program
215  * @param meetingFile
216  * @param MeetingsArray
217  * @return status of success
218  */
219 int readMeetingFile(char meetingFile[], Meeting **MeetingsArray)
220 {
221     /* dynamic allocation settings */
222     int meetingsCapacity = ALLOC_SIZE;
223     *MeetingsArray = (Meeting *)calloc(meetingsCapacity, sizeof(Meeting));
224     if (*MeetingsArray == NULL)
225     {
226         fprintf(stderr, STANDARD_LIB_ERR_MSG);
227         return EXIT_FAILURE;
228     }
229
230     /* open file to reading */
231     FILE *fp = fopen(meetingFile, "r"); //open file for reading
232     if (fp == NULL) //fail opening file
233     {
234         fprintf(stderr, INPUT_ERR1);
235         return EXIT_FAILURE;
236     }
237
238     char line[INPUT_LINE_LENGTH] = {EOS};
239     int meetingsCounter = -1; //counter of the meetings number
240     while (fgets(line, sizeof(line), fp) != NULL)
241     {
242         /* change the array size if needed */
243         if (meetingsCounter == meetingsCapacity)
244         {
245             meetingsCapacity += ALLOC_SIZE;
246             *MeetingsArray = (Meeting *)realloc(*MeetingsArray, sizeof(Meeting) * meetingsCapacity);
247             if (*MeetingsArray == NULL)
248             {
249                 fprintf(stderr, STANDARD_LIB_ERR_MSG);
250                 fclose(fp);
251                 return EXIT_FAILURE;
252             }
253         }
254
255         if (meetingsCounter == -1)
256         {
257             unsigned long int i = transformStringToInt(line);
258             sickId = i;
259         }
260         else
261         {
262             /* check the line and add it to the array */
263             Meeting myMeeting;

```

11.1

```

264         myMeeting = transformMeetingLines(line);
265         (*MeetingsArray)[meetingsCounter].time = myMeeting.time;
266         (*MeetingsArray)[meetingsCounter].distance = myMeeting.distance;
267         (*MeetingsArray)[meetingsCounter].InfectedId = myMeeting.InfectedId;
268         (*MeetingsArray)[meetingsCounter].spreaderId = myMeeting.spreaderId;
269     }
270     meetingsCounter++; //raise the counter by 1
271 }
272 if (meetingsCounter == -1 && (feof(fp) == true)) //empty file case
273 {
274     emptyMeetingFlag = true;
275 }
276 gMeetingsCounter = meetingsCounter;
277 fclose(fp);
278 return EXIT_SUCCESS;
279 }
280
281 /**
282  * allocate memory for the people file
283  * open file to reading
284  * read the people file into the program
285  * @param peopleFile
286  * @param PeopleArray
287  * @return
288  */
289 int readPeopleFile(char peopleFile[], People **PeopleArray)
290 {
291     /* dynamic allocation settings */
292     int peoplesCapacity = ALLOC_SIZE;
293     *PeopleArray = (People *)calloc(peoplesCapacity, sizeof(People));
294     if (*PeopleArray == NULL)
295     {
296         fprintf(stderr, STANDARD_LIB_ERR_MSG);
297         return EXIT_FAILURE;
298     }
299
300     /* open file to reading */
301     FILE *fp = fopen(peopleFile, "r"); //open file for reading
302     if (fp == NULL) //fail opening file
303     {
304         fprintf(stderr, INPUT_ERR1);
305         return EXIT_FAILURE;
306     }
307
308     char line[INPUT_LINE_LENGTH] = {EOS};
309     int peopleCounter = 0; //counter of the meetings number
310     while (fgets(line, sizeof(line), fp) != NULL)
311     {
312         /* change the array size if needed */
313         if (peopleCounter == peoplesCapacity)
314         {
315             peoplesCapacity += ALLOC_SIZE;
316             *PeopleArray = (People *)realloc(*PeopleArray, sizeof(People) * peoplesCapacity);
317             if (*PeopleArray == NULL)
318             {
319                 fprintf(stderr, STANDARD_LIB_ERR_MSG);
320                 fclose(fp);
321                 return EXIT_FAILURE;
322             }
323         }
324
325         /* check the line and add it to the array */
326         People myPeople;
327         myPeople = transformPeopleLines(line);
328         if (errorHappened == ONE) //handle with fail of strncpy inside transformPeopleLines
329         {
330             fprintf(stderr, STANDARD_LIB_ERR_MSG);
331             fclose(fp);

```

12.1

```

332         return EXIT_FAILURE;
333     }
334     (*PeopleArray)[peopleCounter].probabilityToSick = myPeople.probabilityToSick;
335     (*PeopleArray)[peopleCounter].id = myPeople.id;
336     (*PeopleArray)[peopleCounter].name = myPeople.name;
337     (*PeopleArray)[peopleCounter].age = myPeople.age;
338
339     peopleCounter++; //raise the counter by 1
340 }
341 if (peopleCounter == 0 && (feof(fp) == true)) //empty file case
342 {
343     emptyPeopleFlag = true;
344 }
345
346 gPeoplesCounter = peopleCounter;
347 fclose(fp);
348 return EXIT_SUCCESS;
349 }
350
351 /**
352  * free the dynamic allocation of the given pointers
353  * @param PeoplesArray
354  * @param MeetingsArray
355  */
356 void freeAllocations(People **PeoplesArray, Meeting **MeetingsArray)
357 {
358     if (*PeoplesArray != NULL)
359     {
360         for (int p = 0; p < gPeoplesCounter; p++)
361         {
362             free((*PeoplesArray)[p].name);
363         }
364         free(*PeoplesArray);
365     }
366     if (*MeetingsArray != NULL)
367     {
368         free(*MeetingsArray);
369     }
370     *PeoplesArray = NULL;
371     *MeetingsArray = NULL;
372 }
373
374 /**
375  * transform string to int
376  * @param line
377  * @return int val of the given string
378  */
379 unsigned long int transformStringToInt(char *line)
380 {
381     char *end;
382     unsigned long int i = strtoul(line, &end, 10);
383     return i;
384 }
385
386 /**
387  * transform string to int
388  * @param line
389  * @return float val of the given string
390  */
391 float transformStringToFloat(char *line)
392 {
393     char *end;
394     float f = strtod(line, &end);
395     return f;
396 }
397
398 /**
399  * create Meeting struct from the given details

```

```

400  * @param line
401  * @return Meeting struct
402  */
403  Meeting transformMeetingLines(char *line)
404  {
405      char *tok;
406      tok = strtok(line, " , -");
407      int counter = 0;
408
409      Meeting myMeeting = {NULL_SPREADER_ID, NULL_SPREADER_ID, ZERO, ZERO}; //default vals
410      while ( tok != NULL)
411      {
412          if (counter == 0) //<infector_id>
413          {
414              unsigned long int i = transformStringToInt(tok);
415              myMeeting.spreaderId = i;
416          }
417          else if (counter == 1) //<infected_id>
418          {
419              unsigned long int i = transformStringToInt(tok);
420              myMeeting.infectedId = i;
421          }
422          else if (counter == 2) //<distance>
423          {
424              float f = transformStringToFloat(tok);
425              myMeeting.distance = f;
426          }
427          else if (counter == 3) //<time>
428          {
429              float f = transformStringToFloat(tok);
430              myMeeting.time = f;
431          }
432          counter++;
433          tok = strtok(NULL, " , -");
434      }
435      return myMeeting;
436  }
437
438  /**
439   * create People struct from the given details
440   * @param line
441   * @param counterVal
442   * @return People struct
443   */
444  People transformPeopleLines(char *line)
445  {
446      //<Person Name> <Person ID> <Person age>\n
447
448      char *tok;
449      tok = strtok(line, " , -");
450      int counter = 0;
451      People myPeople;
452      while ( tok != NULL)
453      {
454          if (counter == 0) //<Person Name>
455          {
456              myPeople.name = (char *)malloc((strlen(tok) + 1) * sizeof(char));
457              if (myPeople.name == NULL)
458              {
459                  errorHappened = ONE;
460                  myPeople.name = NULL_NAME;
461              }
462              strcpy(myPeople.name, tok);
463              if (myPeople.name == NULL)
464              {
465                  errorHappened = ONE;
466                  myPeople.name = NULL_NAME;
467              }

```

```

468     }
469     else if (counter == 1) //<Person ID>
470     {
471         unsigned long int i = transformStringToInt(tok);
472         myPeople.id = i;
473     }
474     else if (counter == 2) //<Person age>
475     {
476         float f = transformStringToFloat(tok);
477         myPeople.age = f;
478     }
479     counter++;
480     tok = strtok(NULL, " , -");
481 }
482 myPeople.probabilityToSick = NULL_PROBABILITY_TO_SICK;
483
484 return myPeople;
485 }
486
487 /**
488  * Gets information about a meeting between two people,
489  * and calculates the chance that one of them will infect the other in Corona
490  * @param dist
491  * @param time
492  * @return float of the probability
493  */
494 float crna(float dist, float time)
495 {
496     if (dist == ZERO)
497     {
498         return 1;
499     }
500     return (time * MIN_DISTANCE) / (dist * MAX_TIME);
501 }
502
503 /**
504  * update the probability field by iteration on the meetings array:
505  * @param PeopleArray
506  * @param MeetingsArray
507  */
508 int updateProbabilityForEverPeople(People **PeopleArray, Meeting **MeetingsArray)
509 {
510     for (int m = 0; m < gMeetingsCounter; ++m)
511     {
512         //calc prob by crna
513         float currentProbability = crna((*MeetingsArray)[m].distance, (*MeetingsArray)[m].time);
514         int index = binarySearch(&PeopleArray, 0, gPeoplesCounter - 1,
515                                 (*MeetingsArray)[m].InfectedId);
516         if (index == NOT_IN_ARRAY)
517         {
518             //if we arrive here, we had id in meeting that is not appear in people.
519             //this dosen't sponse to happen, by the Guidelines
520             // but I decide to prevent option of index-out-of-range error In my tests. so It's here.
521             continue;
522         }
523         //update the probability for the infected in this meeting
524         (*PeopleArray)[index].probabilityToSick = currentProbability;
525     }
526     // update sick people probability:
527     int index = binarySearch(&PeopleArray, 0, gPeoplesCounter - 1, sickId);
528     if (index == NOT_IN_ARRAY)
529     {
530         //if we arrive here, the spreader id not appear in people input file.
531         //this dosen't suppose to happen, by the Guidelines. but its here for any case..
532         return EXIT_FAILURE;
533     }
534     (*PeopleArray)[index].probabilityToSick = ONE;
535     return EXIT_SUCCESS;

```

```

536 }
537
538 /**
539  * update the final probability for every person, by the infection order (in meeting file)
540  * @param PeopleArray
541  * @param MeetingsArray
542  */
543 int calculateTotalProbability(People **PeopleArray, Meeting **MeetingsArray)
544 {
545     //updateProbabilityForEverPeople run before
546     //we assume the meeting are ordered by they real hierarchy.
547     for (int m = 0; m < gMeetingsCounter; ++m) //for every meeting
548     {
549         int index = binarySearch(&PeopleArray, 0, gPeoplesCounter - 1,
550                                 (*MeetingsArray)[m].InfectedId);
551         if (index == NOT_IN_ARRAY)
552         {
553             //if we arrive here, we had id in meeting that is not appear in people.
554             //this dosen't suppose to happen, by the Guidelines
555             // but I decide to prevent option of index-out-of-range error In my tests. so It's here.
556             continue;
557         }
558         float currentProbability = (*PeopleArray)[index].probabilityToSick;
559         //update its val by its previous
560         float previousProb = findPreviousProbability(&PeopleArray, (*MeetingsArray)[m].spreaderId);
561         if (elementNotInArray == true)
562         {
563             //same as above.
564             continue;
565         }
566         (*PeopleArray)[index].probabilityToSick = currentProbability * previousProb;
567     }
568     return EXIT_SUCCESS;
569 }
570
571 /**
572  * open output file to write, and write inside it the given params
573  * @param PeopleArray
574  * @return EXIT_SUCCESS if the process ends successfully, and EXIT_FAILURE else
575  */
576 int writeToOutput(People **PeopleArray)
577 {
578     FILE *fp = fopen(OUTPUT_FILE, "w");
579     if (fp == NULL)
580     {
581         fprintf(stderr, OUTPUT_ERR);
582         return EXIT_FAILURE;
583     }
584     for (int p = gPeoplesCounter - 1; p >= 0; p--)
585     {
586         float probability = (*PeopleArray)[p].probabilityToSick;
587         if (probability >= MEDICAL_SUPERVISION_THRESHOLD)
588         {
589             fprintf(fp, MEDICAL_SUPERVISION_THRESHOLD_MSG, (*PeopleArray)[p].name,
590                     (*PeopleArray)[p].id);
591         }
592         else if (probability >= REGULAR_QUARANTINE_THRESHOLD)
593         {
594             fprintf(fp, REGULAR_QUARANTINE_MSG, (*PeopleArray)[p].name, (*PeopleArray)[p].id);
595         }
596         else
597         {
598             fprintf(fp, CLEAN_MSG, (*PeopleArray)[p].name, (*PeopleArray)[p].id);
599         }
600     }
601     fclose(fp);
602     return EXIT_SUCCESS;
603 }

```



```

604
605 /**
606  * mergeByProbability two subarrays- Array[l..m] and Array[m+1..r] , compare by probability
607  * @param Array
608  * @param left
609  * @param mid
610  * @param right
611  */
612 void mergeByProbability(People ***Array, int left, int mid, int right)
613 {
614     int startIndex = mid + 1; // the first element after mid index
615
616     if ((*Array)[mid].probabilityToSick <=
617         (*Array)[startIndex].probabilityToSick) //array is already sorted :)
618     {
619         return;
620     }
621
622     while ( left <= mid && startIndex <= right )
623     {
624         if ((*Array)[left].probabilityToSick <=
625             (*Array)[startIndex].probabilityToSick) // element is in the right place
626         {
627             left++;
628         }
629         else // If element not in right place. replace in-place
630         {
631             People temporaryPeople =
632             {
633                 (*Array)[startIndex].name,
634                 (*Array)[startIndex].id, (*Array)[startIndex].age,
635                 (*Array)[startIndex].probabilityToSick
636             };
637             int index = startIndex;
638
639             while ( index != left ) //move elements between the current indexes
640             {
641                 (*Array)[index].name = (*Array)[index - 1].name;
642                 (*Array)[index].id = (*Array)[index - 1].id;
643                 (*Array)[index].age = (*Array)[index - 1].age;
644                 (*Array)[index].probabilityToSick = (*Array)[index - 1].probabilityToSick;
645                 index--;
646             }
647             //return our value to array
648             (*Array)[left].name = temporaryPeople.name;
649             (*Array)[left].id = temporaryPeople.id;
650             (*Array)[left].age = temporaryPeople.age;
651             (*Array)[left].probabilityToSick = temporaryPeople.probabilityToSick;
652
653             // Update the pointers
654             left++;
655             mid++;
656             startIndex++;
657         }
658     }
659 }
660
661 /**
662  * sorting the given array with mergeByProbability-sort, in place
663  * @param Array
664  * @param left
665  * @param right
666  */
667 void mergeSort(People **pPeopleArray, int left, int right, SortingParam param)
668 {
669     if (left < right)
670     {
671         int mid = left + (right - left) / 2;

```

```

672     mergeSort(&pPeopleArray, left, mid, param); //sort subArray- Array[left...mid]
673     mergeSort(&pPeopleArray, mid + 1, right, param); //sort subArray- Array[mid...right]
674     if (param == Probability)
675     {
676         mergeByProbability(&pPeopleArray, left, mid,
677                             right); //mergeByProbability two sorted subArrays
678     }
679     else if (param == ID)
680     {
681         mergeById(&pPeopleArray, left, mid, right);
682     }
683 }
684 }
685
686 /**
687  * return the probability of the given "previous id"
688  * @param PeopleArray
689  * @param previousId
690  * @return the probability(0,1) in sucecess, 0 in fail, 1 if we got the id of the spreader.
691  */
692 float findPreviousProbability(People ***PeopleArray, unsigned long int previousId)
693 {
694     //if prev is the spreader, return 1
695     if (previousId == sickId)
696     {
697         return ONE;
698     }
699     else
700     {
701         int index = binarySearch(&PeopleArray, 0, gPeoplesCounter - 1, previousId);
702         if (index == NOT_IN_ARRAY)
703         {
704             //if we arrive here, we had id in meeting that is not appear in people.
705             //this dosen't sponse to happen, by the Guidelines
706             // but I decide to prevent option of index-out-of-range error In my tests. so It's here.
707             elementNotInArray = true;
708             return NOT_IN_ARRAY;
709         }
710         return ((*PeopleArray)[index].probabilityToSick); //return its probability
711     }
712 }
713
714 int binarySearch(People ***Array, int left, int right, unsigned long int id)
715 {
716     if (right < left) //x is not in array
717     {
718         return NOT_IN_ARRAY;
719     }
720     int mid = left + (right - left) / 2;
721     if ((*Array)[mid].id == id) //x is present at the middle
722     {
723         return mid;
724     }
725     if ((*Array)[mid].id > id) // If x is smaller than mid
726     {
727         return binarySearch(Array, left, mid - 1, id);
728     }
729     return binarySearch(Array, mid + 1, right, id); //else x is grater than mid
730 }
731
732 /**
733  * mergeById two subarrays- Array[l..m] and Array[m+1..r], compartion by id
734  * @param Array
735  * @param left
736  * @param mid
737  * @param right
738  */
739 void mergeById(People ***Array, int left, int mid, int right)

```

```

740 {
741     int startIndex = mid + 1; // the first element after mid index
742
743     if ((*Array)[mid].id <=
744         (*Array)[startIndex].id) //array is already sorted :
745     {
746         return;
747     }
748
749     while ( left <= mid && startIndex <= right )
750     {
751         if ((*Array)[left].id <=
752             (*Array)[startIndex].id) // element is in the right place
753         {
754             left++;
755         }
756         else // If element not in right place. replace in-place
757         {
758             People temporaryPeople =
759                 {
760                     (*Array)[startIndex].name,
761                     (*Array)[startIndex].id, (*Array)[startIndex].age,
762                     (*Array)[startIndex].probabilityToSick
763                 };
764             int index = startIndex;
765
766             while ( index != left ) //move elements between the current indexes
767             {
768                 (*Array)[index].name = (*Array)[index - 1].name;
769                 (*Array)[index].id = (*Array)[index - 1].id;
770                 (*Array)[index].age = (*Array)[index - 1].age;
771                 (*Array)[index].probabilityToSick = (*Array)[index - 1].probabilityToSick;
772                 index--;
773             }
774             //return our value to array
775             (*Array)[left].name = temporaryPeople.name;
776             (*Array)[left].id = temporaryPeople.id;
777             (*Array)[left].age = temporaryPeople.age;
778             (*Array)[left].probabilityToSick = temporaryPeople.probabilityToSick;
779
780             // Update the pointers
781             left++;
782             mid++;
783             startIndex++;
784         }
785     }
786 }

```

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')