234124 - מבוא לתכנות מערכות
תרגיל בית 1
סמסטר אביב 2020

תאריך פרסום: 18/11/2020
תאריך הגשה: 20/12/2020

משקל התרגיל: 12% מהציון הסופי.
מתרגלים אחראים: דור הריס

מענה לשאלות בנוגע לתרגיל יינתן אך ורק בפורום התרגיל בפיאצה או בשעות הקבלה. לפני פרסום שאלה בפורום אנא בדקו אם כבר נענתה – מומלץ להיעזר בכלי החיפוש שהוצגו במצגת האדמיניסטרציה בתרגול הראשון.

1 הערות כלליות

- יש להגיש את תרגיל הבית בזוגות בלבד.
- שימו לב: לא תינתנה דחיות במועד הגשת התרגיל פרט למקרים חריגים. **תכננו את הזמן בהתאם**.
- כל חומר נלווה לתרגיל נמצא על השרת בתיקייה ~mtm/public/2021a/ex1.
- קראו את התרגיל עד סופו לפני שאתם מתחילים לממש. חובה להתעדכן בעמוד ה- F.A.Q של התרגיל, הכתוב שם מחייב.
- העתקות קוד בין סטודנטים ובפרט גם העתקות מסמסטרים קודמים תטופלנה. עם זאת – מומלץ ומבורך להתייעץ עם חברים על ארכיטקטורת המימוש.
- מומלץ מאוד לכתוב את הקוד בחלקים קטנים, לקמפל כל חלק בנפרד על השרת, ולבדוק שהוא עובד באמצעות שימוש בטסטים קטנים שתכתבו בעצמכן. לא נדרש מכן בתרגיל להגיש טסטים, אך כידוע, כולנו בני אדם – רצוי לבדוק את התרגיל שלכן היטב כולל מקרי קצה, כי אתן תידרשו לכך.

2 חלק יבש

2.1 זיהוי שגיאות בקוד

2.1.1 סעיף א

מצאו 6 שגיאות תכנות ו-4 שגיאות קונבנציה(code convention [1]) בפונקציה הבאה. מטרת הפונקציה היא לשכפל מספר פעמים את המחרוזת המתקבלת לתוך מחרוזת חדשה. למשל, הקריאה stringDuplicator("Hello", 3) תחזיר את המחרוזת "HelloHelloHello". במקרה של שגיאה בריצת הפונקציה, הפונקציה תחזיר NULL. מותר להניח שהקלט תקין.

```
#include "stdlib.h"
#include "string.h"
#include "assert.h"

char* stringDuplicator(char* s, int times){
      assert(!s);
      assert(times > 0);
      int LEN = strlen(s);
      char* out = malloc(LEN*times);
      assert(out);
      for (int i=0; i<=times; i++){
      out = out + LEN;
      strcpy(out,s);
      }
      return out;
}
```

2.1.2 סעיף ב

כתבו גרסה מתוקנת של הפונקציה.

[1] ראו מסמך "Code Conventions.pdf" באתר הקורס

3 חלק רטוב

3.1 מימוש מבנה נתונים גנרי – תור עדיפויות

בחלק זה נממש ADT גנרי עבור "תור עדיפויות" – תור של איברים שבו לכל איבר יש עדיפות. הממשק של התור נמצא בקובץ priority_queue.h. עליכן לכתוב את הקובץ priority_queue.c אשר ממש את מבנה הנתונים הגנרי המתואר.

מאחר שמבנה הנתונים גנרי יש לאתחל אותו עם מספר מצביעים לפונקציות אשר יגדירו את אופן הטיפול בעצמים המאוחסנים בו ובעדיפויות שלהם.

כדי לאפשר למשתמשים בתור (לא למפתחי התור!) לעבור על איבריו בצורה סדרתית, לכל תור מוגדר איטרטור (מלשון איטרציה, מעבר על איברים) פנימי ויחיד שבעזרתו יוכל המשתמש לעבור על כל איברי תור העדיפויות. האיטרציה על איברי התור צריכה להבטיח למשתמש מעבר על האיברים בסדר יורד לפי העדיפות שלהם (מהאיבר עם העדיפות הגבוהה ביותר, לאיבר עם העדיפות הנמוכה ביותר), נדע לעשות זאת באמצעות פונקציית השוואת העדיפויות (comparePriority) אשר מסופקת העת יצירת התור.

פונקציית compare מחזירה 0 אם שני האיברים שהיא מקבלת שווים, ערך חיובי אם המפתח הראשון גדול מהשני וערך שלילי אם המפתח השני גדול מהראשון (בדומה לstrcmp). במקרה ששישנן שתי עדיפויות זהות, האיטרטור יחזיר את האיברים לפי סדר הכנסתם לתור.

3.1.1 פונקציות שנדרש לממש

להלן תיאור קצר של הפונקציות אותן אתן נדרשות לממש. פירוט נוסף הכולל משמעות הפרמטרים וערכי ההחזרה האפשריים של כל פונקציה נמצאים בקובץ priority_queue.h. על המימוש שתספקו לענות על דרישות התיעוד (המובא כאן ובקובץ h) ויש להניח שכל הדרישות ייבדקו. במידת הצורך לאחר פרסום התרגיל יפורסמו הבהרות על ידי סגל הקורס והן יהיו מחייבות עבור המימוש שתספקו. ההבהרות שיפורסמו יעודכנו בקובץ h וכמו כן בכתב מודגש בצהוב בחלק זה.

3.1.1.1 יצירת תור חדש

```
PriorityQueue pqCreate(CopyPQElement copy_element, FreePQElement free_element,
EqualPQElements equal_elements, CopyPQElementPriority copy_priority,
FreePQElementPriority free_priority, ComparePQElementPriorities compare_priority);
```

יוצר תור עדיפויות חדש ריק.
שימו לב! הפונקצייה equal_elements מחזירה האם שני איברים זהים. לעומתה, הפונקציה compare_priority יוצרת יחס סדר על העדיפויות.

3.1.1.2 הריסת תור קיים

```
void pqDestroy(PriorityQueue queue);
```

הורס תור ואת כל האיברים שהוא מכיל

3.1.1.3 העתקת תור

```
PriorityQueue pqCopy(PriorityQueue queue);
```

יוצר תור חדש בעל אותן פונקציות גנריות כמו queue, ומעתיק לתוכו את כל האיברים מqueue כולל העדיפויות שלהם.

3.1.1.4 החזרת מספר האיברים בתור

```
int pqGetSize(PriorityQueue queue);
```

מחזיר את מספר האיברים בתור העדיפויות.

3.1.1.5 בדיקה האם איבר קיים בתור

```
bool pqContains(PriorityQueue queue, PQElement element);
```

בדיקה האם איבר בעל ערך element נמצא בתור העדיפויות queue.

3.1.1.6 הכנסת איבר חדש לתור

```
PriorityQueueResult pqInsert(PriorityQueue queue, PQElement element,
PQElementPriority priority);
```

הכנסת איבר חדש לתור. ערך האיבר הוא element ועדיפותו היא priorityניתן להכניס לתור את אותו איבר עם אותה העדיפות מספר פעמים.

3.1.1.7 שינוי עדיפות של איברים בתור

```
PriorityQueueResult pqChangePriority(PriorityQueue queue, PQElement element,
PQElementPriority old_priority, PQElementPriority new_priority);
```

שינוי העדיפות של איבר בעל הערך element מעדיפות old_priority לnew_priority. לצורכי האיטרציה, שינוי עדיפות של איבר שקולה להכנסתו מחדש לתור.

3.1.1.8 הוצאת איבר הראשון מהתור

```
PriorityQueueResult pqRemove(PriorityQueue queue);
```

הוצאת האיבר בעל העדיפות הגבוהה ביותר מהתור ומחיקתו, כלומר, לאחר פעולה זו האיבר לא נמצא יותר בתור.

3.1.1.9 הוצאת איבר כללי מהתור

```
PriorityQueueResult pqRemoveElement(PriorityQueue queue, PQElement element);
```

הוצאת האיבר בעל העדיפות הגבוהה ביותר שערכו שווה לelement מהתור ומחיקתו. כלומר, לאחר פעולה זו האיבר לא נמצא יותר בתור.

3.1.1.10 ריקון התור

```
PriorityQueueResult pqClear(PriorityQueue queue);
```

מוחק את כל האיברים מהתור. כלומר, לאחר פעולה זו, התור נמצא במצב זהה למצבו מיד לאחר היצירה הראשונית שלו.

3.1.1.11 הזזת האיטרטור לתחילת התור והחזרת האיבר הראשון

```
PQElement pqGetFirst(PriorityQueue queue);
```

מחזיר את האיטרטור לתחילת התור ומחזיר מצביע לאיבר הראשון בתור.

3.1.1.12 קידום האיטרטור והחזרת המצביע לאיבר המוצבע על ידו

```
PQElement pqGetNext(PriorityQueue queue);
```

הקידום מבוצע לפי הסדר שמוגדר על העדיפויות בתור.

<u>דגשים נוספים ודרישות מימוש:</u>

- **קראו את התיעוד ב-priority_queue.h! הוא מגדיר במפורש כל פעולה שעליכם לממש ויעזור לכם במיוחד!**
- קיימות פונקציות להן מספר ערכי שגיאה אפשריים. בהערה מעל כל פונקציה תוכלו למצוא את כל השגיאות שיכולות להתרחש בעת קריאה אליה בקובץ הממשק שסופק לכם (מתחת למילה return בהערה). במקרה של מספר שגיאות אפשריות, החזירו את השגיאה שהוגדרה ראשונה בקובץ.
  אם מתרחשת שגיאה שאינה ברשימה, יש להחזיר PQ_ERROR.
- אין הגבלה על מספר האיברים בתור.
- במקרה של שגיאה יש לשמור על שלמות מבנה הנתונים ולוודא שאין דליפות זיכרון.
- במידה ואחת הפונקציות מקבלת NULL כאיבר ו/או כעדיפות, החזירו PQ_NULL_ARGUMENT.
- בתיעוד המופיע ב-priority_queue.h עבור חלק מהפונקציות כתוב שהאיטרטור במצב לא מוגדר אחרי הקריאה לפונקציה, המשמעות היא שכאשר איטרטור נמצא במצב זה, אסור למשתמש להניח משהו לגביו, כלומר שאינכם צריכים להבטיח שום דבר בנוגע לערך האיטרטור ולכן עליכם להחזיר NULL כאשר קוראים לpqGetNext.

3.2 מימוש מערכת לניהול אירועים

כידוע, הפקולטה למדעי המחשב הינה הפקולטה הטובה ביותר ללמוד בה בטכניון. ועד הסטודנטים רוצה לארגן לטובת הסטודנטים מספר רב של אירועים בשנה, אך אבוי! קשה לו לנהל רישום מסודר של האירועים והוא רוצה לייעל את תהליך הרישום של האירועים והפצתם לסטודנטים, לכן פנה לחניכי קורס מת"מ בבקשה לעזרה.

לכל אירוע שמארגן ועד מדמ"ח יש שם, תאריך וקבוצת חברי ועד שאחראים על קיומו. ועד הסטודנטים רוצה שמערכת ניהול האירועים תשמור אוסף של אירועים שעתידים לקרות, וכאשר מגיע הזמן לביצוע של אירוע, להחזיר אותו ולהוציא אותו מהמערכת, כאשר לכל אירוע ישמר שמו, התאריך שלו וקבוצת חברי הועד האחראים עליו.

<u>הערות</u>:

מסופק לכם מבנה הנתונים priority_queue שכבר מומשו על ידינו. הוסיפו את הקובץ priority_queue.h לקבצי ה-h הנדרשים ודאגו שהקובץ libpriority_queue.a (שנמצא בתיקיה שסופקה לכם) יימצא בתיקיה הנוכחית. לבסוף קמפלו לפי ההנחיות שבסוף התרגיל – שימו לב לדגלים שנוספו ולהנחיות.

על מנת להוסיף את הקובץ ב-:cmake

1. לפני השורה של add_executable יש להוסיף (שימו לב ל-. בין הסוגריים):

(.)link_directories

2. לאחר השורה של add_executable יש להוסיף:

target_link_libraries(<name_of_executable_file> libpriority_queue.a)

כאשר name_of_executable_file הוא שם קובץ ההרצה שכתבתם בפקודה add_executable.

אתם רשאים להשתמש בmap שבניתם לצורך מימוש חלק זה, אך זו לא חובה.

3.2.1 טיפוס הנתונים DATE

בחלק זה נממש ADT עבור Date – מבנה של תאריך, בו לכל תאריך יש יום, חודש ושנה. הממשק של Date נמצא בקובץ date.h. עליכם לכתוב את הקובץ date.c אשר ממש את מבנה הנתונים הגנרי המתואר.

פונקציות שנדרש לממש

להלן תיאור קצר של הפונקציות אותן אתן נדרשות לממש. פירוט נוסף כולל את משמעות הפרמטרים וערכי ההחזרה האפשריים של כל פונקציה נמצאים בקובץ date.h. על המימוש שתספקו לענות על דרישות התיעוד (המובא כאן ובקובץ h) ויש להניח שכל הדרישות ייבדקו. במידת הצורך לאחר פרסום התרגיל יפורסמו הבהרות על ידי סגל הקורס והן יהיו מחויבות עבור המימוש שתספקו. ההבהרות שיפורסמו יעודכנו בקובץ h וכמו כן בכתב מודגש בצהוב בחלק זה.

3.2.1.1 יצירת תאריך חדש

```
Date dateCreate(int day, int month, int year);
```

יוצר תאריך חדש.

3.2.1.2 הריסת תאריך קיים

```
void dateDestroy(Date date);
```

הורס תאריך ואת כל תוכנו.

3.2.1.3 העתקת תאריך קיים

```
Date dateCopy(Date date);
```

יוצר תאריך חדש בעל אותו תוכן כמו date.

3.2.1.4 קבלת תאריך

```
bool dateGet(Date date, int* day, int* month, int* year);
```

מחזיר את הערכים אשר ניתנו ביצירת התאריך לתוך המצביעים הרלוונטיים.

3.2.1.5 השוואת תאריך

```
int dateCompare(Date first_date, Date second_date);
```

משווה בין שני תאריכים. אם התאריך הראשון מוקדם יותר יחזיר ערך שלילי, אם התאריך השני מוקדם יותר יחזיר ערך חיובי. אם שני התאריכים זהים יחזיר 0.

3.2.1.6 קידום תאריך

```
void dateTick(Date date);
```

מקדם את התאריך ביום אחד. שימו לב! אתם יכולים להניח שבכל חודש יש 30 ימים, ואין שנים מעוברות.

3.2.2 טיפוס נתונים ראשי

המערכת הראשית תהיה המערכת EventManager. המבנה מאגד בתוכו את כל נתונים האירועים שיבואו אלינו לטובה מאת ועד מדמ"ח.

יובהר כי:

- לא יתכן שבמערכת יהיו שני אירועים בעלי אותו שם באותו היום. (כן יתכן שני אירועים בעלי אותו שם בימים שונים)
- לא יתכן שיוכנס למערכת אירוע שכבר היה בעבר (אפשר להכניס רק אירועים עתידיים)

הפונקציות הבאות אינן דורשות כל גנריות ועליכן לכתוב אותן בצורה שהכי נוחה לכן.

3.2.2.1 יצירת מערכת לניהול אירועים

```
EventManager createEventManager(Date date);
```

פונקציה היוצרת את המערכת לניהול אירועים EventManager בתאריך date.

<u>פרמטרים</u>: date - התאריך הראשון בו מתחילה המערכת לעבוד.

<u>ערכי שגיאה</u>: NULL במקרה של שגיאה כלשהי, אחרת נחזיר מערכת ניהול בחירות חדשה.

3.2.2.2 הריסת המערכת

```
void destroyEventManager(EventManager em);
```

הפונקציה תהרוס את מערכת ניהול האירועים ותשחרר את כל המשאבים שהוקצו לה. במידה והתקבל NULL אין לבצע דבר.

<u>פרמטרים</u>: em - המערכת אותה יש להרוס.

<u>ערכי שגיאה</u>: המערכת לא מחזירה ערך ובפרט לא מחזירה ערך שגיאה כלשהו.

3.2.2.3 הוספת אירוע חדש למערכת בתאריך ספציפי

```
EventManagerResult emAddEventByDate(EventManager em, char* event_name, Date date,
int event_id);
```

הפונקציה מוסיפה אירוע חדש למערכת בתאריך date.

<u>פרמטרים</u>:

- em - המערכת אליה נרצה להוסיף אירוע חדש.
- event_name - השם של האירוע.
- date - התאריך בו יתווסף האירוע. התאריך חייב להיות אחרי התאריך הנוכחי בו עובדת המערכת.
- event_id - מספר זיהוי אירוע ייחודי.

<u>ערכי שגיאה</u>:

- *EM_NULL_ARGUMENT* – אחד הארגומנטים שהתקבל הוא NULL.
- *EM_INVALID_DATE* – התאריך שהתקבל לפני התאריך הנוכחי של המערכת.
- *EM_INVALID_EVENT_ID* – מספר הזיהוי של האירוע הוא שלילי.
- *EM_EVENT_ALREADY_EXISTS* – האירוע בעל שם event_name כבר קיים בתאריך date.
- *EM_EVENT_ID_ALREADY_EXISTS* – מספר הזיהוי של האירוע כבר קיים במערכת.
- *EM_OUT_OF_MEMORY* - מעידה על חוסר בזכרון. במקרה כזה על המשתמש לשחרר את כל משאבי המערכת ולסיים את פעולת התוכנית. הניחו שתופעה זו עלולה לקרות לאחר כל הקצאת זכרון.
- *EM_SUCCESS* – במידה והפעולה הצליחה

3.2.2.4 הוספת אירוע חדש למערכת בעוד כמות קבועה של ימים

```
EventManagerResult emAddEventByDiff(EventManager em, char* event_name, int days,
int event_id);
```

הפונקציה מוסיפה אירוע חדש למערכת בעוד days ימים מהתאריך הנוכחי במערכת.

<u>פרמטרים</u>:

- em - המערכת אליה נרצה להוסיף אירוע חדש.
- event_name - השם של האירוע.
- days - מספר הימים מהתאריך הנוכחי בו יתווסף האירוע, מספר הימים חייב להיות אי שלילי.
- event_id - מספר זיהוי אירוע ייחודי.

<u>ערכי שגיאה</u>:

- *EM_NULL_ARGUMENT* – אחד הארגומנטים שהתקבל הוא NULL.
- *EM_INVALID_DATE* – התאריך שהתקבל לפני התאריך הנוכחי של המערכת.
- *EM_INVALID_EVENT_ID* – מספר הזיהוי של האירוע הוא שלילי.
- *EM_EVENT_ALREADY_EXISTS* – האירוע בעל שם event_name כבר קיים בתאריך שאליו מנסים להכניס אותו.
- *EM_EVENT_ID_ALREADY_EXISTS* – מספר הזיהוי של האירוע כבר קיים במערכת
- *EM_OUT_OF_MEMORY* - מעידה על חוסר בזכרון. במקרה כזה על המשתמש לשחרר את כל משאבי המערכת ולסיים את פעולת התוכנית. הניחו שתופעה זו עלולה לקרות לאחר כל הקצאת זכרון.
- *EM_SUCCESS* – במידה והפעולה הצליחה

3.2.2.5 הסרת אירוע מהמערכת

```
EventManagerResult emRemoveEvent(EventManager em, int event_id);
```

הפונקציה מסירה מהמערכת את האירוע `event_id`.

<u>פרמטרים</u>:

- em - המערכת ממנה נרצה להסיר אירוע.
- event_id - מספר זיהוי של האירוע אותו רוצים להסיר מהמערכת.

<u>ערכי שגיאה</u>:

- *EM_NULL_ARGUMENT* – אחד הארגומנטים שהתקבל הוא NULL.
- *EM_INVALID_EVENT_ID* – מספר הזיהוי של האירוע הוא שלילי.
- *EM_EVENT_NOT_EXISTS* – לא קיים אירוע כזה במערכת.
- *EM_OUT_OF_MEMORY* - מעידה על חוסר בזכרון. במקרה כזה על המשתמש לשחרר את כל משאבי המערכת ולסיים את פעולת התוכנית. הניחו שתופעה זו עלולה לקרות לאחר כל הקצאת זכרון.
- *EM_SUCCESS* – במידה והפעולה הצליחה

3.2.2.6 שינוי תאריך של אירוע

```
EventManagerResult emChangeEventDate(EventManager em, int event_id, Date new_date);
```

הפונקציה משנה את התאריך של event_id לnew_date.

<u>פרמטרים</u>:

- em - המערכת בה נרצה לשנות תאריך של אירוע.

- event_id - מספר זיהוי אירוע ייחודי
- new_date - התאריך החדש של האירוע.

ערכי שגיאה:

- *EM_NULL_ARGUMENT* – אחד הארגומנטים שהתקבל הוא NULL
- *EM_INVALID_DATE* – התאריך שהתקבל לפני התאריך הנוכחי של המערכת
- *EM_INVALID_EVENT_ID* – מספר הזיהוי של האירוע הוא שלילי
- *EM_EVENT_ID_NOT_EXISTS* – מספר הזיהוי של האירוע לא קיים במערכת.
- *EM_EVENT_ALREADY_EXISTS* – האירוע בעל שם event_name כבר קיים בתאריך שאליו מנסים להכניס אותו.
- *EM_OUT_OF_MEMORY* - מעידה על חוסר בזכרון. במקרה כזה על המשתמש לשחרר את כל משאבי המערכת ולסיים את פעולת התוכנית. הניחו שתופעה זו עלולה לקרות לאחר כל הקצאת זכרון.
- *EM_SUCCESS* – במידה והפעולה הצליחה

3.2.2.7 הוספת חבר ועד חדש למערכת

```
EventManagerResult emAddMember(EventManager em, char* member_name, int member_id);
```

הפונקציה מוסיפה חבר ועד חדש בשם member_name שלו זיהוי של member_id. שימו לב שכאשר מוסיפים חבר ועד חדש למערכת לא ניתן להסירו והוא תמיד יהיה במערכת.

פרמטרים:

- em - המערכת אליה נרצה להוסיף חבר ועד חדש.
- member_name - השם של חבר הועד החדש.
- member_id - מספר זיהוי ייחודי של חבר הוועד.

ערכי שגיאה:

- *EM_NULL_ARGUMENT* – אחד הארגומנטים שהתקבל הוא NULL
- *EM_INVALID_MEMBER_ID* – מספר הזיהוי של חבר הוועד הוא שלילי
- *EM_MEMBER_ID_ALREADY_EXISTS* – מספר הזיהוי של חבר הועד כבר קיים במערכת
- *EM_OUT_OF_MEMORY* - מעידה על חוסר בזכרון. במקרה כזה על המשתמש לשחרר את כל משאבי המערכת ולסיים את פעולת התוכנית. הניחו שתופעה זו עלולה לקרות לאחר כל הקצאת זכרון.
- *EM_SUCCESS* – במידה והפעולה הצליחה

3.2.2.8 קישור בין חבר ועד לאירוע

```
EventManagerResult emAddMemberToEvent(EventManager em, int member_id, int
event_id);
```

הפונקציה מוסיפה את חבר הועד בעל הזיהוי member_id לקבוצת הסטודנטים שמנהלים את האירוע event_id.

פרמטרים:

- em - המערכת בה נרצה להוסיף קישור בין אירוע לחבר ועד.
- event_id - מספר זיהוי של האירוע.
- member_id - מספר זיהוי ייחודי של חבר הוועד.

ערכי שגיאה:

- *EM_NULL_ARGUMENT* – אחד הארגומנטים שהתקבל הוא NULL
- *EM_INVALID_EVENT_ID* – מספר הזיהוי של האירוע הוא שלילי

- *EM_INVALID_MEMBER_ID* – מספר הזיהוי של חבר הוועד הוא שלילי
- *EM_EVENT_ID_NOT_EXISTS* – מספר הזיהוי של חבר הועד לא קיים במערכת
- *EM_MEMBER_ID_NOT_EXISTS* – מספר הזיהוי של חבר הועד לא קיים במערכת.
- *EM_EVENT_AND_MEMBER_ALREADY_LINKED* – האירוע כבר מנוהל על ידי חבר הועד הרלוונטי
- *EM_OUT_OF_MEMORY* - מעידה על חוסר בזכרון. במקרה כזה על המשתמש לשחרר את כל משאבי המערכת ולסיים את פעולת התוכנית. הניחו שתופעה זו עלולה לקרות לאחר כל הקצאת זכרון.
- *EM_SUCCESS* – במידה והפעולה הצליחה

3.2.2.9 הסרת קישור בין חבר ועד לאירוע

```
EventManagerResult emRemoveMemberFromEvent (EventManager em, int member_id, int
event_id);
```

הפונקציה מסירה את חבר הועד בעל הזיהוי member_id לקבוצת הסטודנטים שמנהלים את האירוע event_id.

<u>פרמטרים</u>:

- em - המערכת בה נרצה להסיר קישור בין חבר ועד לאירוע.
- event_id - מספר זיהוי של האירוע.
- member_id - מספר זיהוי ייחודי של חבר הוועד.

<u>ערכי שגיאה</u>:

- *EM_NULL_ARGUMENT* – אחד הארגומנטים שהתקבל הוא NULL
- *EM_INVALID_EVENT_ID* – מספר הזיהוי של האירוע הוא שלילי
- *EM_INVALID_MEMBER_ID* – מספר הזיהוי של חבר הוועד הוא שלילי
- *EM_EVENT_ID_NOT_EXISTS* – מספר הזיהוי של חבר הועד לא קיים במערכת
- *EM_MEMBER_ID_NOT_EXISTS* – מספר הזיהוי של חבר הועד לא קיים במערכת.
- *EM_EVENT_AND_MEMBER_NOT_LINKED* – האירוע לא מנוהל על ידי חבר הועד הרלוונטי
- *EM_OUT_OF_MEMORY* - מעידה על חוסר בזכרון. במקרה כזה על המשתמש לשחרר את כל משאבי המערכת ולסיים את פעולת התוכנית. הניחו שתופעה זו עלולה לקרות לאחר כל הקצאת זכרון.
- *EM_SUCCESS* – במידה והפעולה הצליחה

3.2.2.10 קידום הזמן המערכת

```
EventManagerResult emTick(EventManager em, int days)
```

הפונקציה מקדמת את הזמן במערכת בdays, כמו כן הפונקציה מזהה את האירועים שצריכים להתרחש בזמן שעבר ומוציאה אותם מהמערכת.

<u>פרמטרים</u>:

- em - המערכת בה נרצה לקדם את הזמן.
- days - כמות הימים שיש לקדם את המערכת.

<u>ערכי שגיאה</u>:

- *EM_NULL_ARGUMENT* – אחד הארגומנטים שהתקבל הוא NULL
- *EM_INVALID_DATE* – כמות הימים שיש לקדם את המערכת שלילי או 0.

- *EM_OUT_OF_MEMORY* - מעידה על חוסר בזכרון. במקרה כזה על המשתמש לשחרר את כל משאבי המערכת ולסיים את פעולת התוכנית. הניחו שתופעה זו עלולה לקרות לאחר כל הקצאת זכרון.
- *EM_SUCCESS* – במידה והפעולה הצליחה

3.2.2.11 החזרת מספר האירועים השמור במערכת

```
int emGetEventsAmount(EventManager em);
```

הפונקציה מחזירה את כמות האירועים העתידיים השמורים במערכת.
פרמטרים:

- em - המערכת אותה נרצה לתשאל.

ערכי שגיאה: אם em הוא NULL יש להחזיר 1-, אחרת יש להחזיר את כמות האירועים במערכת.

3.2.2.12 החזרת האירוע הבא

```
char* emGetNextEvent(EventManager em);
```

הפונקציה מחזירה את האירוע הבא שאמור להתרחש, אם שני אירועים אמורים להתרחש באותו היום, המערכת תחזיר קודם את האירוע שהוכנס קודם למערכת.
פרמטרים:

- em - המערכת אותה נרצה לתשאל.

ערכי שגיאה: אם em הוא NULL יש להחזיר NULL, אחרת יש להחזיר את מצביע לשם האירוע הבא.

3.2.2.13 הדפסת כלל האירועים

```
void emPrintAllEvents(EventManager em, const char* file_name);
```

הפונקציה תדפיס לקובץ בשם file_name את שמות האירועים, התאריך שלהם ושמות הסטודנטים האחראים עליהם מסודרים לפי התאריך שלהם. אם יש שני אירועים באותו תאריך, האירוע שהוכנס קודם למערכת יודפס קודם.
בכל שורה יודפס תחילה שם האירוע, לאחר מכן פסיק ואז תאריך האירוע, ואז חברי הועד המנהלים את האירוע מופרדים בפסיק גם הם, כאשר אם יש כמה חברי ועד שמנהלים את אותו אירוע, הם יודפסו בסדר עולה לפי הid שלהם.

פרמטרים:

- em - המערכת אותה נרצה לתשאל.
- file_name - שם הקובץ אליו נדרש לכתוב את המידע השמור במערכת.

ערכי שגיאה: הפונקציה לא מחזירה ערכים ובפרט לא מחזירה ערכי שגיאה.

נניח ויש שני אירועים event1 בתאריך ה1.1.2021 שמנוהל על ידי member1 וmember2 וevent2 בתאריך ה2.2.2021 שמנוהל על ידי member1 בלבד אז המערכת תדפיס לקובץ את התוכן הבא:

event1,1.1.2021,member1,member2

3.2.2.14 הדפסת חברי הועד event2,2.2.2021,member1

```
void emPrintAllResponsibleMembers(EventManager em, const char* file_name);
```

הפונקציה תדפיס לקובץ בשם file_name את שמות חברי הועד וכמות האירועים עליהם הם אחראים. כאשר חבר הועד אשר אחראי על מספר האירועים הגבוה ביותר יודפס תחילה, אם יש שני חברי ועד שמנהלים את אותה כמות אירועים, חבר הועד בעל מספר הזיהוי הנמוך יותר יודפס קודם.
בכל שורה יודפס תחילה שם חבר הועד, לאחר מכן פסיק ואז כמות האירועים עליהם הוא אחראי. חבר ועד אשר אחראי על אפס אירועים לא יודפס.

פרמטרים:

- em - המערכת אותה נרצה לתשאל.
- file_name - שם הקובץ אליו נדרש לכתוב את המידע השמור במערכת.

ערכי שגיאה: הפונקציה לא מחזירה ערכים ובפרט לא מחזירה ערכי שגיאה.

נניח ויש שני אירועים event1 בתאריך ה1.1.2021- שמנוהל על ידי member1 וmember2- ו-event2 בתאריך ה-2.2.2021 שמנוהל על ידי member1 בלבד אז המערכת תדפיס לקובץ את התוכן הבא:

member1,2

Member2,1

3.3 דגשים נוספים ודרישות מימוש

1. המימוש חייב לציית לכללי כתיבת הקוד המופיעים תחת Code Conventions -> Course Material. **אי עמידה בכללים אלו תגרור הורדת נקודות**
2. על המימוש שלכם לעבור ללא **שגיאות זיכרון** (גישות לא חוקיות וכדומה) וללא **דליפות זיכרון**.
3. במידה ובפונקציה מסוימת קיימות מספר אפשרויות פוטנציאליות לערך שגיאה, אנו נבחר את השגיאה הראשונה על פי סדר השגיאות המופיע תחת הפונקציה הספציפית.
   השגיאה EVENT_MANAGER_OUT_OF_MEMORY
   יכולה להתרחש בכישלון בהקצאת זיכרון בכל שלב ולכן לא משתתפת בעניין הסדר.
4. המערכת צריכה לעבוד על שרת csl3.
5. מימוש כל המערכת צריך להיעשות ע"י חלוקה ל-ADT שונים.
   נצפה לחלוקה נוחה של המערכת כך שניתן יהיה להכניס שינויים בקלות יחסית ולהשתמש בטיפוסי הנתונים השונים עבור תוכנות דומות.
6. מומלץ לתכנן את ארכיטקטורת המערכת (מבני הנתונים המשתתפים, ה-data types וה-ADT-ים הרלוונטיים) ואת עדכון כל החלקים הרלוונטיים בארכיטקטורה בכל אחת מן הפונקציות. זו הסיבה שהתרגיל הזה נחשב לארוך. הדגש צריך להיות התכנון הנכון.

3.4 MAKEFILE

1. עליכם לספק Makefile כמו שנלמד בקורס עבור בניית הקוד של תרגיל זה.
2. הכלל הראשון ב Makefile יקרא event_manager ויבנה קובץ הרצה בשם event_manager – השתמשו בקובץ event_manager_example_tests.c שסופק לכם, ובמימוש שלכם של הטיפוס EventManager כפי שתואר מעלה.
3. הכלל השני במייקפייל יקרא priority_queue ויבנה קובץ הרצה בשם priority_queue - השתמשו בקובץ pq_example_tests.c שסופק לכם ובמימוש שלכם של הטיפוס PriorityQueue כפי שתואר למעלה.
3. אנו מצפים לראות שלכל ADT קיים כלל אשר בונה עבורו קובץ .o כפי שלמדתם בקורס - כדי לחסוך הידור של כל התכנית כאשר משנים רק חלק קטן ממנה.
4. הוסיפו גם כלל clean שמוחק את תוצרי ההידור.
5. על ה-makefile להיות יחיד, ואין להשתמש באלמנטים שלא נלמדו.

3.5 הידור, קישור ובדיקה

ועליו לעבור הידור באופן הבא csl3 התרגיל ייבדק על שרת

עבור ה- priority queue:

- `gcc -std=c99 -o priority_queue -Wall -pedantic-errors -Werror -DNDEBUG tests/pq_example_test.c mtm_pq/*.c`

עבור ה- event manager:

- `gcc -std=c99 -o event_manager -Wall -pedantic-errors -Werror -DNDEBUG *.c tests/eventManagerTests*.c -L. -lpriority_queue`

*.c מציין את כל קבצי c שנמצאים בתיקייה בפרט את הקבצים שמסופקים לכם (אותם אין להגיש).

-lpriority_queue מקשר את הקובץ libpriority_queue.a שסופק לכם (מניח שהקובץ נמצא בתיקייה הנוכחית).

3.6 איתור דליפות זכרון באמצעות valgrind

המערכת חייבת לשחרר את כל הזיכרון שעמד לרשותה בעת ריצתה. כדי לוודא זאת, תוכלו להשתמש בכלי בשם valgrind שמתחקה אחר ריצת התכנית שלכם, ובודק האם ישנם משאבים שלא שוחררו. הדרך להשתמש בכלי על מנת לבדוק האם יש לכם דליפות בתכנית היא באמצעות שתי הפעולות הבאות:

1. קימפול של השורה לעיל עם הדגל g-.
2. הרצת השורה הבאה:

- `valgrind --leak-check=full ./event_manager`

כאשר election זה שם קובץ ההרצה (לא מגישים את קובץ ההרצה לכן תוכלו לתת לו שם כרצונכם).

הפלט ש-valgrind מפיק אמור לתת לכם, במידה ויש לכם דליפות (והידרתם את התוכנית עם דגל -g), את שרשרת הקריאות שהתבצעו שגרמו לדליפה. אתם אמורים באמצעות ניפוי שגיאות (בדומה לתרגיל 0 להבין היכן היה צריך לשחרר את אותו משאב שהוקצה ולתקן את התכנית. בנוסף, valgrind מראה דברים נוספים כמו קריאה לא חוקית (שלא גררה segmentation fault - גם שגיאות אלו עליכם להבין מהיכן מגיעות ולתקן (בכל מקרה בדיקה שיש בו שגיאת זיכרון כלשהי לא מקבל נקודות.

תוכלו למצוא תיעוד של דגלים נוספים שימושיים של הכלי ע"י man valgrind (או לחפש באינטרנט, לדוג' כאן – תחת Memchek Options).

4 אופן ההגשה

לנוחותכן מסופקת לכם תוכנית "בדיקה עצמית" בשם finalCheck, בתיקיית התרגיל. התוכנית בודקת ש-zip ההגשה בנוי נכון ומריצה את הטסטים שסופקו כפי שיורצו ע"י הבודק האוטומטי. הפעלת התוכנית ע"י:

```
~mtm/public/2021a/ex1/finalCheck <submission>.zip
```

הקפידו להריץ את הבדיקה על קובץ (zip) ההגשה ממש, דהיינו – אם אתם משנים אותו לאחר מכן – הקפידו להריץ את הבדיקה שוב!

4.1 הגשה יבשה

את הפתרון לחלק היבש יש להקליד ולהגיש כקובץ pdf בשם dry.pdf ולשים אותו בתיקייה הראשית של התרגיל (שמגישים בהגשה רטובה). **לא יתקבלו פתרונות בכתב יד ו/או מצולמים** (יקבלו 0 ללא בדיקה). אין להגיש בתא הקורס.

4.2 הגשה רטובה

את ההגשה הרטובה יש לבצע דרך אתר הקורס, תחת

Electronic Submit <- HW1 <- Assignments

פורמט ההגשה יהיה כדלקמן:

- הקובץ **Submission.zip** יכיל את הקבצים הבאים:
  - dry.pdf - פתרון התרגיל היבש.
  - event_manager.c - מימוש החלק הרטוב השני.
  - makefile - קובץ המakefile שהכנתן.
  - קבצים נוספים שמימשתן, אם מימשתן.
  - priority_queue.c - מימוש החלק הרטוב הראשון.
  - קבצים נוספים שמימשתן, אם מימשתן.

- **שימו לב: השתמשו אך ורק בzip** (פורמט אחר לא יתקבל). אין חשיבות לשם קובץ ה-zip המוגש (Submission.zip מטה).
- הקבצים הבאים מסופקים לכם במהלך הבדיקה **ואין לצרפם להגשה**:
  - date.h, event_manager.h, priority_queue.h - תחת התיקייה הראשית
  - test_utilities.h, pq_example_test.c, event_manager_example_tests.c - תחת התיקייה tests.
  - libpriority_queue.a – ספרייה שמממשת את תור העדיפויות עבור שימוש בחלק הרטוב השני של התרגיל. תחת התיקייה הראשית

event_manager_example_tests.c - זהו קובץ שלתוכו תוכלו לכתוב טסטים עבור התרגיל, בדומה לטסטים הכתובים בקובץ pq_example_test.c

- הקבצים נמצאים בשרת csl3 תחת התיקייה :

```
~mtm/public/1920b/ex1/
```

- מותר להגיש את התרגיל מספר פעמים, **רק ההגשה האחרונה נחשבת**.
- על מנת לבטח את עצמכן נגד תקלות בהגשה האוטומטית **שימרו את קוד האישור עבור ההגשה**. עדיף לשלוח גם לשותפה. כמו כן שימרו עותק של התרגיל על חשבון ה-csl3 שלכם לפני ההגשה האלקטרונית ואל תשנו אותו לאחריה (שינוי הקובץ יגרור שינוי חתימת העדכון

האחרון).
*** כל אמצעי אחר לא יחשב הוכחה לקיום הקוד לפני ההגשה.