מבוא לתכנות מערכות
תרגיל בית מספר 3 (ADT)
סמסטר אביב 2018

תאריך פרסום: 7 במאי 2018

תאריך הגשה: 28 במאי 2018

משקל התרגיל: 9% מהציון הסופי (תקף)

מתרגלת אחראית: **סיפן נוה**

1 הערות כלליות

- **שימו לב: לא יינתנו דחיות במועד התרגיל. תכננו את הזמן בהתאם.**
- לשאלות בנוגע להבנת התרגיל יש לפנות לסדנאות של אחד מהמתרגלים, או לשאול בפורום של הקורס במודל. לשאלה לגבי הניסוח אפשר לפנות לסיפן במייל. לפני שליחת שאלה - נא וודאו שהיא לא נענתה כבר ב-F.A.Q או במודל, ושהתשובה אינה ברורה ממסמך זה, מהדוגמא ומהבדיקות שפורסמו עם התרגיל.
- **קראו מסמך זה עד סופו לפני שאתם מתחילים לממש. יתכן שתצטרכו להתאים את המימוש שלכם לחלק עתידי בתרגיל. תכננו את המימוש שלכם לפני שאתם ניגשים לעבוד.** רצוי לעבור על הדוגמא שפורסמה **לפני** תחילת הפתרון.
- כל חומר נלווה לתרגיל נמצא על השרת בתיקייה ~mtmchk/public/1718b/ex3 .
- **חובה** להתעדכן בעמוד ה- F.A.Q של התרגיל, הכתוב שם מחייב.
- **העתקות קוד בין סטודנטים תטופלנה בחומרה!**
- **מומלץ מאוד מאוד לכתוב את הקוד בחלקים קטנים, לקמפל כל חלק בנפרד על השרת, ולבדוק שהוא עובד באמצעות שימוש בטסטים. קראו את סעיף 5 –דגשים ורמזים – לפני תחילת העבודה על הקוד!**

לנוחיותכם, השינויים שבוצעו לאחר פרסום הגיליון מסומנים בצהוב

2 חלק יבש

2.1 בחירת מבני נתונים

לפניכם מספר דוגמאות אשר דורשות שימוש במבני נתונים. עליכם להחליט איזה מבנה נתונים מתוך מבני הנתונים הבסיסיים שנלמדו בקורס (מילון – ראו הסבר בסעיף 3.1, קבוצה, מחסנית ורשימה) הוא המתאים ביותר לכל דוגמה. הסבירו מדוע הוא מתאים יותר מן האחרים.

1. המתרגלת בקומבי (בעוד כמה תרגולים של קומבי תבינו את הרפרנס) מעוניינת במבנה נתונים שיאפשר לה לבדוק האם הסוגריים (מכל סוג שהוא: {[()]}) מאוזנים בכל ביטוי שתקבל. באיזה מבנה נתונים תציעו לה להשתמש?
2. המתכנתים של מערכת הרישום לקורסים בטכניון מעוניינים במבנה נתונים שיאפשר לשמור בעבור כל סטודנט את כמות הנקודות שלו, כדי לדעת באיזה תאריך לפתוח לו את האפשרות להרשם לקורסים. איזה מבנה נתונים יכול לשמש עבור מטרה זו?
3. הציעו מבנה נתונים שמאפשר להפוך רשימה נתונה עם חזרות לרשימה בלי חזרות בדרך פשוטה.

2.2 תכנות גנרי

ברצוננו לממש אלגוריתם merge sort המקבל מערך של איברים, וממיין אותו.

1. כתבו את הפונקציה mergeSort, המקבלת מערך של עצמים מטיפוס לא ידוע, וממיינת אותו. על הפונקציות להתאים לעצמים מכל סוג שהוא. השתמשו בפונקציות עזר אם צריך. שימו לב כי הפונקציה לא תחזיר מערך חדש, אלא היא תמיין את המערך שהתקבל כקלט. כלומר, לפונקציה לא יהיה ערך החזרה.
2. הסבירו בקצרה מה התפקיד והטיפוס של כל פרמטר שיש לשלוח לפונקציה הראשית ולפונקציות העזר שכתבתם.

2.3 רשימות מקושרות

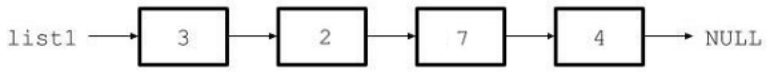
נתון מבנה פשוט של רשימה מקושרת של מספרים שלמים (int):

```
typedef struct node_t* Node;
struct node_t {
        int n;
        Node next;
};
```

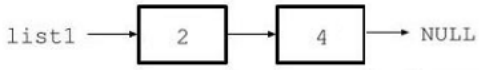
1. כתבו את הפונקציה listFilter.

<u>קלט:</u> רשימה מקושרת ותנאי בוליאני על מספרים שלמים (מקבל int ומחזיר true/false)

<u>פלט:</u> הפונקציה תחזיר רשימה חדשה שמכילה רק את האיברים שבעבורם התנאי מחזיר true. לדוגמא, אם הקלט הוא התנאי "האיבר זוגי" והרשימה הבאה:



הפלט יהיה:

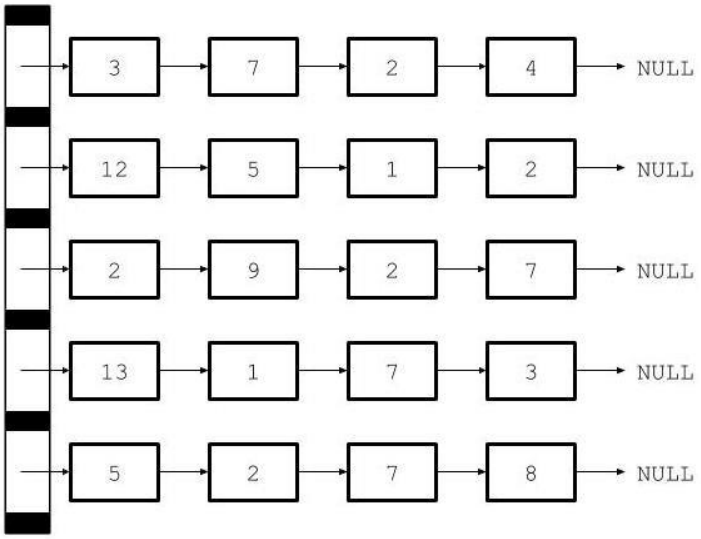


2. כתבו את הפונקציה coolElements שמשתמשת בפונקציה listFilter.

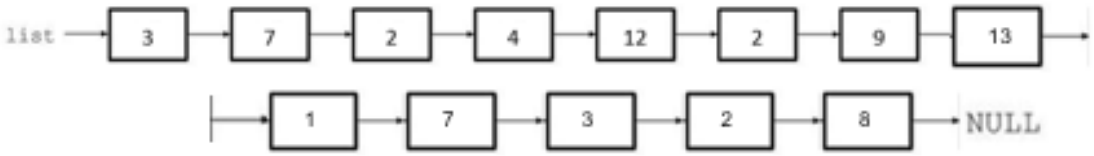
<u>קלט:</u> מערך של רשימות מקושרות, וגודל המערך.

<u>פלט:</u> רשימה מקושרת חדשה, אשר מכילה רק את האיברים ברשימות שמתחלקים במספרו הסידורי של התא במערך שמכיל את הרשימה, מודולו 3, ועוד 1. כלומר, מהרשימה הראשונה נחזיר את כל האיברים שמתחלקים ב-1, מהרשימה השניה את כל אלו שמתחלקים ב-2, מהרשימה החמישית (זאת שנמצאת בתא הרביעי במערך) נחזיר את כל אלו שמתחלקים ב2 וכך הלאה.

לדוגמא, אם הקלט הוא כנ"ל:



הפלט יהיה:



הערות:

- ניתן להשתמש בפונקציות עזר על מנת לפשט את הפתרון.
- הניחו כי קיימת הפונקציה void destroyList(Node) אשר משחררת את כל הזיכרון עבור רשימה מקושרת נתונה מלבד פונקציה זו יש לממש את כל הפונקציות אשר בהן אתם משתמשים.
- התייחסו למצביע ל-NULL כאל רשימה ריקה.

3 חלק רטוב

3.1 מימוש מילון ממוין *Generic Data Type*

בחלק זה נממש תחילה ADT גנרי עבור מילון ממוין. מילון הוא מבנה נתונים שמאפשר שמירה של זוגות <מפתח, ערך> וחיפוש הערך לפי המפתח הצמוד אליו. לדוגמה, מילון של סטודנטים בטכניון עשוי להכיל את תעודת הזהות של הסטודנט בתור מפתח, structו של סטודנט בתור ערך.

רמז למימוש: חישבו על הרשימות שראינו בתרגול, כאשר הפעם במקום להכיל ערך אחד, כל איבר ברשימה מכיל מפתח וערך.

קובץ המנשק map_mtm.h מסופק לכם ונמצא בתיקיית התרגיל על השרת. עליכם לכתוב את הקובץ map_mtm.c המממש את מבנה הנתונים המתואר.

מאחר והמילון הממוין (מעתה נגיד רק 'מילון') גנרי, יש לאתחל אותו עם מספר מצביעים לפונקציות אשר יגדירו את אופן הטיפול בעצמים המאוחסנים בו.

על מנת לאפשר למשתמשים במילון לעבור על איבריו, לכל מילון מוגדר איטרטור פנימי (יחיד) אשר בעזרתו יכול המשתמש לעבור על כל איברי המילון. האיטרציה על איברי המילון צריכה להבטיח למשתמש מעבר על האיברים בסדר עולה מהמפתח הקטן למפתח הגדול - לפי פונקצית השוואת המפתחות המסופקת בעת יצירת מילון חדש. פונקצית ההשוואה אמורה להחזיר 0 אם שני המפתחות שהיא מקבלת שווים, 1 אם המפתח הראשון גדול מהשני, ו-1- אחרת (בדומה ל-strcmp).

שימו לב! בחלק זה איננכם יכולים להשתמש במבני הנתונים שסופקו לכם ועליכם לממש את המילון בעצמכם

3.1.1 פעולות

1. mapCreate– יצירת מילון ממוין חדש, בפעולה זו מוגדרות לרשימה הפעולות בעזרתן ניתן להעתיק, לשחרר מפתחות וערכים, ולהשוות מפתחות.
2. mapDestroy– מחיקת מילון קיים תוך שחרור מסודר של כל הזיכרון שבשימוש.
3. mapCopy– העתקת מילון קיים ל**עותק חדש**, כולל העתקת האיברים (מפתחות וערכים).
4. mapGetSize– החזרת מספר המפתחות במילון.
5. mapContains– תחזיר true אם המפתח הנתון קיים במילון וfalse אחרת.
6. mapPut– שינוי ערך של מפתח קיים/ הוספת זוג מפתח-ערך חדש למילון.
7. mapGet– החזרת הערך הממופה למפתח הנתון.
8. mapRemove– מחיקת מפתח מהמילון.
9. mapGetFirst– החזרת האיטרטור הפנימי לתחילת המילון והחזרת המפתח הראשון.
10. mapGetNext– קידום האיטרטור הפנימי והחזרת המפתח המוצבע על ידו.
11. mapClear– ריקון המילון מאיברים.

מהו איטרטור? איטרטור הוא מצביע לאיבר במפה. הוא אמור לאפשר למשתמש לבצע איטרציות (בעזרת לולאת while לדוגמה) על מילון, באופן שקול לאיטרציה על מערך באמצעות מצביע. הפונקציה GetFirst שקולה לאתחול הפוינטר (לדוגמה ptr) לתחילת המערך, והפונקציה getNext שקולה ל++ptr, כלומר מקדמת את המצביע לאיבר הבא. שתי הפונקציות המדוברות, מחזירות למשתמש את ה**מפתח** שכן הוא לא יודע מה המימוש הפנימי שלכם, והוא מכיר רק את המפתח. כדי לקבל את הערך הצמוד אליו, הוא ישתמש בפונקציה Get.

לכל פקודה ייתכנו שגיאות שונות. ניתן למצוא את השגיאות האפשריות לכל פקודה בתיעוד בקובץ map_mtm.h.
במקרה של כמה שגיאות אפשריות יש להחזיר את ערך השגיאה שהוגדר ראשון בקובץ המנשק הנתון.

3.1.2 הגבלות על המימוש

על המימוש שלכם לעמוד במגבלות הבאות:

- אין הגבלה על מספר האיברים במילון.
- במקרה של שגיאה יש לשמור על שלמות מבנה הנתונים ולוודא שאין דליפות זיכרון.

3.2 כתיבת מערכת לצפיה בסדרות טלויזיה

כפי שידוע לכם, קיים הרבה עומס לימודי בטכניון וכתוצאה מכך הרבה סטודנטים מבלים את זמנם בנעימים בצפיה בסדרות, בין המבחנים ושיעורי הבית.

אנשי הפיתוח של נטפליקס רצו להתאים את המערכת לצרכי הסטודנטים ולשם כך הם פונים אליכם.

למזלכם, קיימים מימושים מצוינים עבור מבני הנתונים הבסיסיים שנלמדו בקורס ומסופקים לכם.

עליכם לממש את MtmFlix ADT.

שימו לב שלצורך מימוש המערכת מסופקים לכם מבני הנתונים הסטנדרטיים שכבר ממומשים על ידינו. בשביל להשתמש בהם, עשו #include לקובץ הh, דאגו שהקובץ libmtm.a (שנמצא בתיקיה שסופקה לכם) יהיה בתיקיה הנוכחית שלכם, וקמפלו לפי ההנחיות שבסוף התרגיל – שימו לב לדגלים שנוספו להנחיות.

3.2.1 טיפול בשגיאות

במקרה שיש כמה שגיאות אפשריות יש לדווח על השגיאה העליונה ביותר בPDF הנוכחי.

מכיוון שברצוננו להשתמש בADT הראשי בעתיד, חובה לוודא כי גם במקרה של שגיאה כלשהי, המערכת נשמרת במצב תקין.

כלומר לאחר גילוי שגיאה, מצב המערכת יהיה כאילו לא בוצעה פקודה כלשהי.

ניתן להניח כי לא קיימות שגיאות נוספות מלבד אלו המפורטות לכל פקודה (ושגיאות *MTMFLIX_OUT_OF_MEMORY* שאותן יש להחזיר במקרה של בעיית זיכרון, מכל הפונקציות בתוכנית), היכולות להתרחש בכל נקודה בתוכנית כתלות במימוש.

3.2.2 הפקודות

על המערכת לתמוך בפונקציות הללו:

<u>יצירת מערכת מתמפליקס חדשה</u>

```
MtmFlix mtmFlixCreate();
```

תיאור הפעולה: פקודה זו יוצרת מערכת חדשה לניהול סדרות.

- **פרמטרים:** אין.
- **מחזירה:** הפונקציה תחזיר NULL במקרה של שגיאה. אחרת, תחזיר מערכת מתמפליקס חדשה.

<u>הריסת מערכת מתמפליקס</u>

```
void mtmFlixDestroy(MtmFlix mtmflix);
```

תיאור הפעולה: פקודה זו הורסת את המערכת ואת כל משאביה.

- **פרמטרים:**

mtmflix – המערכת אותה נרצה להרוס.

- **מחזירה:**

אין לפונקציה ערך חזרה.

<u>הוספת משתמש למערכת</u>

```
MtmFlixResult mtmFlixAddUser(MtmFlix mtmflix, const char* username, int age);
```

תיאור הפעולה: פקודה זו מוסיפה משתמש חדש למערכת.

- **פרמטרים:**

mtmflix – המערכת שאיתה אנו עובדים.

username - השם של המשתמש החדש. לא יתכנו שני משתמשים בעלי אותו שם משתמש.

age - הגיל של המשתמש החדש.

- **מחזירה:**

MTMFLIX_SUCCESS – הפעולה בוצעה בהצלחה.

MTMFLIX_NULL_ARGUMENT – הפוינטר שהועבר מצביע לNULL.

MTMFLIX_ILLEGAL_USERNAME – שם המשתמש לא חוקי, כלומר לא מכיל תו אחד לפחות, או לא מכיל רק אותיות אנגליות, ומספרים.

MTMFLIX_USERNAME_ALREADY_USED – שם המשתמש כבר נמצא בשימוש על ידי משתמש במערכת.

MTMFLIX_ILLEGAL_AGE – הגיל לא חוקי, כלומר לא גדול או שווה ל-MTM_MIN_AGE וקטן או שווה ל-*MTM_MAX_AGE*.

הסרת משתמש מהמערכת

```
MtmFlixResult mtmFlixRemoveUser(MtmFlix mtmflix, const char* username);
```

תיאור הפעולה: פקודה זו מסירה משתמש מהמערכת. כל משתמש שהוסיף את המשתמש הזה כחבר, יפסיק לראות אותו.

- **פרמטרים:**

mtmflix – המערכת שאיתה אנו עובדים.

username – שם המשתמש של המשתמש.

- **מחזירה:**

MTMFLIX_SUCCESS – הפעולה בוצעה בהצלחה.

MTMFLIX_NULL_ARGUMENT – הפוינטר שהועבר מצביע לNULL.

MTMFLIX_USER_DOES_NOT_EXIST – שם המשתמש לא קיים במערכת.

הוספת סדרה למערכת

```
MtmFlixResult mtmFlixAddSeries(MtmFlix mtmflix, const char* name, int episodesNum, Genre genre, int* ages, int episodeDuration);
```

תיאור הפעולה: פקודה זו מוסיפה סדרה חדשה למערכת.

- **פרמטרים:**

mtmflix – המערכת שאיתה אנו עובדים.

name - השם של הסדרה החדשה.

episodesNum – מספר הפרקים בסדרה.

genre – הז'אנר של הסדרה.

ages – אם הפרמטר הוא NULL, אין הגבלת גיל והסדרה מתאימה לכל המשפחה. אחרת, מערך בגודל 2 שמכיל בתא הראשון הגבלת גיל מינימלי, ובתא השני מכיל גיל מקסימלי שהסדרה מתאימה לו. אם הגיל המינימלי קטן מ-MTM_MIN_AGE, הגיל המינימלי יהיה MTM_MIN_AGE, ואם הגיל המקסימלי גדול מ- MTM_MAX_AGE הוא יהיה MTM_MAX_AGE.

episodeDuration – האורך הממוצע של פרקי הסדרה.

- **מחזירה:**

MTMFLIX_SUCCESS – הפעולה בוצעה בהצלחה.
MTMFLIX_NULL_ARGUMENT – הפוינטר שהועבר מצביע לNULL.
MTMFLIX_ILLEGAL_SERIES_NAME – שם הסדרה לא חוקי, כלומר לא מכיל תו אחד לפחות, או לא מכיל רק אותיות אנגליות, ומספרים.
MTMFLIX_SERIES_ALREADY_EXISTS – כבר קיימת סדרה עם השם במערכת.
MTMFLIX_ILLEGAL_EPISODES_NUM – מספר הפרקים לא גדול ממש מאפס.
MTMFLIX_ILLEGAL_EPISODES_DURATION – אורך פרק ממוצע לא גדול ממש מאפס.

הסרת סדרה מהמערכת

```
MtmFlixResult mtmFlixRemoveSeries(MtmFlix mtmflix, const char* name);
```

תיאור הפעולה: פקודה זו מסירה סדרה מהמערכת. במקרה הזה, כל המשתמשים שאהבו את הסדרה מפסיקים לאהוב אותה שכן היא לא קיימת יותר.

- **פרמטרים:**

mtmflix – המערכת שאיתה אנו עובדים.
name – השם של הסדרה.

- **מחזירה:**

MTMFLIX_SUCCESS – הפעולה בוצעה בהצלחה.
MTMFLIX_NULL_ARGUMENT – הפוינטר שהועבר מצביע לNULL.
MTMFLIX_SERIES_DOES_NOT_EXIST – הסדרה לא קיימת במערכת.

הדפסת הסדרות במערכת

```
MtmFlixResult mtmFlixReportSeries(MtmFlix mtmflix, unsigned int seriesNum, FILE* outputStream);
```

תיאור הפעולה: פקודה זו מדפיסה את הסדרות במערכת ממוינות בסדר אלפביתי לoutputStream, **לפי ז'אנר הסדרה**. כלומר כל הסדרות עם ז'אנר Action תודפסנה לפני סדרות עם ז'אנר Comedy, ובתוך כל ז'אנר הסדרות תהיינה ממוינות לפי סדר אלפביתי של שם הסדרה. אם הפרמטר שהוכנס גדול ממספר הסדרות בקטגוריה יש להדפיס את כולן. יש להדפיס את הסדרות תוך שימוש בפונקציה הנמצאת בmtm_ex3.h.

- **פרמטרים:**

mtmflix – המערכת שאיתה אנו עובדים.
seriesNum – מספר הסדרות שתודפסנה בכל קטגוריה, או 0 כדי להדפיס את כולן. הניחו שלא יתקבל מספר שלילי.
outputStream – הקובץ שאליו מדפיסים את הפלט

- **מחזירה:**

MTMFLIX_SUCCESS – הפעולה בוצעה בהצלחה.
MTMFLIX_NULL_ARGUMENT – הפוינטר שהועבר מצביע לNULL.
MTMFLIX_NO_SERIES – אין סדרות במערכת.

הדפסת המשתמשים במערכת

```
MtmFlixResult mtmFlixReportUsers(MtmFlix mtmflix, FILE* outputStream);
```

תיאור הפעולה: פקודה זו מדפיסה את המשתמשים במערכת ממוינים בסדר לקסיקוגרפי עולה, **לפי שם המשתמש**. יש להדפיס את המשתמשים תוך שימוש בפונקציה הנמצאת בmtm_ex3.h.

- **פרמטרים:**

mtmflix – המערכת שאיתה אנו עובדים.

outputStream – הקובץ שאליו מדפיסים את הפלט

- **מחזירה:**

MTMFLIX_SUCCESS – הפעולה בוצעה בהצלחה.

MTMFLIX_NULL_ARGUMENT – הפוינטר שהועבר מצביע לNULL.

MTMFLIX_NO_USERS – אין משתמשים במערכת.

הוספת סדרה אהובה למשתמש

```
MtmFlixResult mtmFlixSeriesJoin(MtmFlix mtmflix, const char* username, const char* seriesName);
```

תיאור הפעולה: פקודה זו שומרת סדרה מסוימת בתור סדרה שהמשתמש אוהב. אם הסדרה כבר אהובה על המשתמש, לא יתבצע דבר והפעולה נחשבת כאילו היא בוצעה בהצלחה.

- **פרמטרים:**

mtmflix – המערכת שאיתה אנו עובדים.

username - השם של המשתמש.

seriesName – השם של הסדרה.

- **מחזירה:**

MTMFLIX_SUCCESS – הפעולה בוצעה בהצלחה.

MTMFLIX_NULL_ARGUMENT – הפוינטר שהועבר מצביע לNULL.

MTMFLIX_USER_DOES_NOT_EXIST – המשתמש לא קיים במערכת.

MTMFLIX_SERIES_DOES_NOT_EXIST – הסדרה לא קיימת במערכת.

MTMFLIX_USER_NOT_IN_THE_RIGHT_AGE – גיל המשתמש לא מתאים לטווח הגילאים של הסדרה.

המשתמש כבר לא אוהב סדרה

```
MtmFlixResult mtmFlixSeriesLeave(MtmFlix mtmflix, const char* username, const char* seriesName);
```

תיאור הפעולה: מעתה והלאה לא נראה את הסדרה בתור אהובה על המשתמש. אם היא לא אהובה עליו, לא יתבצע דבר.

- **פרמטרים:**

mtmflix – המערכת שאיתה אנו עובדים.

username - השם של המשתמש.

seriesName – השם של הסדרה.

- **מחזירה:**

MTMFLIX_SUCCESS – הפעולה בוצעה בהצלחה.

MTMFLIX_NULL_ARGUMENT – הפוינטר שהועבר מצביע לNULL.

MTMFLIX_USER_DOES_NOT_EXIST – המשתמש לא קיים במערכת.

MTMFLIX_SERIES_DOES_NOT_EXIST – הסדרה לא קיימת במערכת.

הוספת חבר למשתמש

```
MtmFlixResult mtmFlixAddFriend(MtmFlix mtmflix, const char* username1, const char* username2);
```

תיאור הפעולה: פקודה זו מוסיפה את משתמש2 לחברים של משתמש1. שימו לב – הפעולה אינה סימטרית! יכול להיות שמשתמש1 ירצה המלצות על סדרות לפי משתמש2, אך משתמש2 חושב שהטעם של משתמש1 גרוע. אם משתמש2 כבר חבר של משתמש1, לא יתבצע דבר.

- **פרמטרים:**

mtmflix – המערכת שאיתה אנו עובדים.

Username1 - השם של המשתמש שמוסיפים לו את משתמש2 בתור חבר.

Username2 – השם של המשתמש שנוסיף בתור חבר.

- **מחזירה:**

MTMFLIX_SUCCESS – הפעולה בוצעה בהצלחה.

MTMFLIX_NULL_ARGUMENT – הפוינטר שהועבר מצביע לNULL.

MTMFLIX_USER_DOES_NOT_EXIST – אחד המשתמשים לא קיים במערכת.

הסרת חבר

```
MtmFlixResult mtmFlixRemoveFriend(MtmFlix mtmflix, const char* username1, const char* username2);
```

תיאור הפעולה: הטעם של משתמש2 נהיה גרוע ומשתמש1 לא רוצה אותו בתור חבר יותר. נסיר את משתמש2 מהחברים של משתמש1. שימו לב – גם פעולה זו אינה סימטרית!

- **פרמטרים:**

mtmflix – המערכת שאיתה אנו עובדים.

Username1 - השם של המשתמש שמורידים לו את משתמש2 מרשימת החברים.

Username2 – השם של המשתמש שנוריד מרשימת החברים.

- **מחזירה:**

MTMFLIX_SUCCESS – הפעולה בוצעה בהצלחה.

MTMFLIX_NULL_ARGUMENT – הפוינטר שהועבר מצביע לNULL.

MTMFLIX_USER_DOES_NOT_EXIST – אחד המשתמשים לא קיים במערכת.

המלצה על סדרות

```
MtmFlixResult mtmFlixGetRecommendations(MtmFlix mtmflix, const char* username, int count, FILE* outputStream);
```

תיאור הפעולה: פקודה זו נותנת רשימת המלצות למשתמש לגבי סדרות שעשויות לעניין אותו. המערכת תיתן ציון לכל סדרה לפי נוסחה שתפורט בהמשך, ובסופו של התהליך כל סדרה תקבל ציון. לאחר מכן תודפסנה הסדרות לקובץ outputStream, כאשר סדר ההדפסה הוא לפי הציון בסדר יורד, ואם ישנן שתי סדרות בעלות אותו הציון, נדפיס אותן בסדר לקסיקוגרפי עולה. סדרות שגיל המשתמש חורג מהגיל המומלץ שלהן, לא תודפסנה. הדפסת סדרה תיעשה באמצעות הפונקציה בקובץ mtm_ex3.h.

- **פרמטרים:**

mtmflix – המערכת שאיתה אנו עובדים.

Username - השם של המשתמש שממליצים לו על סדרות.

count – מספר הסדרות המקסימלי להדפסה (יתכנו פחות), או 0 בשביל להדפיס את כולן.

- **מחזירה:**

MTMFLIX_SUCCESS – הפעולה בוצעה בהצלחה.
MTMFLIX_NULL_ARGUMENT – הפוינטר שהועבר מצביע לNULL.
MTMFLIX_USER_DOES_NOT_EXIST – המשתמש לא קיים במערכת.
MTMFLIX_ILLEGAL_NUMBER – המספר שלילי

- **הנוסחה:**

כל סדרה במערכת תקבל ציון לפי שלושה קריטריונים: הז'אנר שלה, האורך הממוצע של פרק, והעדפות החברים של המשתמש.
נסמן:
מספר הסדרות עם ז'אנר זהה מתוך הסדרות האהובות על המשתמש יסומן ב-G.
האורך הממוצע של פרקי הסדרות האהובות של המשתמש יסומן ב-L, והאורך של הסדרה הנבדקת ב-CUR.
מספר החברים של המשתמש שהסדרה אהובה עליהם יסומן ב-F.
אז הציון של כל סדרה יהיה:

$$Rank(Series) = \frac{G * F}{1 + |CUR - L|}$$

*** שימו לב כי אם הציון הוא 0 או שגיל המשתמש חורג מהגילאים המותרים של הסדרה, או שהסדרה כבר קיימת בסדרות האהובות על המשתמש, אין להדפיס את הסדרה כי היא לא מעניינת. ***

הבהרות חשובות מאוד מאוד:

- האורך הממוצע של פרקי הסדרות L הוא פשוט סכום האורכים הממוצעים של כל הסדרות, לחלק למספרן (בלי לשקלל בפנים את מספר הפרקים הממוצע).
- החישוב עצמו צריך להיעשות כאילו המספרים הם floating points, ולאחר שקיבלתם את התוצאה, יש להמיר אותה לint (כלומר לעגל כלפי מטה) כדי לקבל את הניקוד של הסדרה.

3.2.3 הגבלות על המימוש

- המימוש חייב לציית לכללי כתיבת הקוד המופיעים תחת Course Material -> Code Conventions. **אי עמידה בכללים אלו תגרור הורדת נקודות.**
- על המימוש שלכם לעבור ללא שגיאות זיכרון (גישות לא חוקיות וכדומה) וללא דליפות זיכרון.
- המערכת צריכה לעבוד על CSL2.
- **אין לשנות את הקבצים שסופקו לכם.** קבצים אלו **אינם** מוגשים ונשתמש בקבצים המקוריים לבדיקת הקוד.
- יש לכתוב את הקוד בפונקציות קצרות וברורות.
- 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.

3.2.5 Makefile

עליכם לספק Makefile כמו שנלמד בקורס עבור בניית הקוד של תרגיל זה.

הכלל הראשון בMakefile יקרא mtmflix ויבנה את התוכנית mtmflix המתוארת למעלה. יש לכתוב את הקובץ כפי שנלמד וללא שכפולי טקסט.

על הmakefile לקמפל את התוכנית שלכם כפי שלמדנו, עם הדגלים המתאימים, עם קובץ הטסטים שפרסמתי באתר בתיקיה tests. הוסיפו גם כלל clean. תוכלו לבדוק את הmakefile שלכם באמצעות הרצת הפקודה make והפעלת קובץ ההרצה שנוצר בסופו.

3.2.6 הידור, קישור ובדיקה

התרגיל ייבדק על שרת csl2 ועליו לעבור הידור בעזרת הפקודה הבאה:

```
gcc -std=c99 -o mtmflix -Wall -pedantic-errors  -Werror -DNDEBUG *.c mtm_ex3.o   -L.  –lmtm
```

משמעות הפקודה:

- "–std=c99": קביעת התקן לשפה להיות C99
- "–o mtmflix": הגדרת שם הקובץ המהודר
- "–Wall": דווח על כל האזהרות
- "–pedantic-errors": דווח על סגנון קוד שאינו עומד בתקן הנבחר כעל שגיאה
- "–Werror": התייחס לאזהרות כאל שגיאות (הקוד חייב לעבור ללא אזהרות)
- "–DNDEBUG": מוסיף את השורה "#define NDEBUG" בתחילת כל יחידת קומפילציה. מתג זה יגרום לכך שהמאקרו assert (אם בשימוש) לא יפריע ולא יופעל בריצת התוכנית.
- "*.c": בחירת כל קבצי הקוד.
  - על התרגיל להיות מורכב בקבצי c וקבצי h בלבד (אשר נכללים באמצעות פקודות include). שימו לב שעל כל הקבצים להיות בספרית השורש של הקובץ zip אשר תגישו.
  - אנו נצרף לקבצים שאתם מגישים גם קובץ בדיקה שיבדוק את התכנית שלכם (ראו הסבר על הבדיקה הרטובה למטה).
  - בנוסף, עליכם לוודא שתרגיל מתקמפל עם הקבצים אשר מסופקים על ידינו. אין להגיש קבצים אלה!
- "–L.": גורם לgcc לחפש ספריות בתיקיה הנוכחית
- "–lmtm": קישור לספריה שמסופקת לכם ע"י הצוות

- Mtm_ex3.o: בשביל פונקציות ההדפסה האחידות

התרגיל ייבדק בדיקה יבשה ובדיקה רטובה.

הבדיקה היבשה כוללת מעבר על הקוד ובודקת את איכות הקוד (שכפולי קוד, קוד מבולגן, קוד לא ברור, שימוש בטכניקות תכנות "רעות").

הבדיקה הרטובה כוללת את הידור התכנית המוגשת והרצתה במגוון בדיקות אוטומטיות. על מנת להצליח בבדיקה שכזו, על התוכנית לעבור הידור, לסיים את ריצתה, ולתת את התוצאות הצפויות.

תרגיל אשר אינו עובר בדיקות יקבל 0 בבדיקה הרטובה. לא תהיינה הנחות בנושא זה!

- וודאו את נכונות התכנית שלכם במקרים כללים ובמקרי קצה.
- מומלץ לחשוב על הבדיקות ולכתוב אותן כבר בזמן כתיבת הקוד.
- נסו ליצור הרבה בדיקות קצרות ולא בדיקה אחת גדולה – כישלון בבדיקות קצרות עוזר לאתר את התקלה.
- וודאו את נכונות הקוד גם לאחר שינויים קטנים אשר אינם נראים משמעותיים ולפני ההגשה.
- עליכם לוודא שהרצה זו מסתיימת בהצלחה וללא דליפות זיכרון או גישות לא חוקיות לזיכרון. כדי לוודא שאין דליפות זיכרון ניתן להשתמש בvalgrind. זהו כלי אשר מותקן בcsl2 ומיועד בין היתר למציאת דליפות זיכרון.

4 הגשה

4.1 הגשה יבשה

יש להגיש לתא הקורס את פתרון החלק היבש של התרגיל – מודפס משני צדי הדף.
אין להגיש את הקוד שכתבתם בחלק הרטוב.

4.2 הגשה רטובה

את ההגשה הרטובה יש לבצע דרך אתר הקורס, תחת Assignments → Exercise 3 → Electronic submission.
הקפידו על הדברים הבאים:

- יש להגיש את קבצי הקוד והmakefile מכווצים לקובץ zip (לא פורמט אחר) כאשר כל הקבצים עבור הפתרון מופיעים בתיקיית השורש בתוך קובץ הzip ותיקיה בשם map_mtm שבתוכה יהיה המימוש עבור החלק הראשון של התרגיל הרטוב.
- **<u>אין להגיש אף קובץ מלבד קבצי h וקבצי c אשר כתבתם בעצמכם ואת הmakefile אשר נדרשתם לעשות</u>**.
- הקבצים אשר מסופקים לכם: list.h, map.h, set.h, libmtm.a יצורפו על ידנו במהלך הבדיקה, וניתן להניח כי הם יימצאו בתיקייה הראשית. הקובץ test_utilities.h אשר מסופק לכם, יימצא <u>תחת התיקייה tests. (גם קובץ זה יצורף על ידנו, ואין להוסיפו לקובץ הzip שלכם).</u>
- על מנת לבטח את עצמכם נגד תקלות בהגשה האוטומטית:
  - שימרו את קוד האישור עבור ההגשה. עדיף לשלוח גם לשותף.
  - שימרו עותק של התרגיל על חשבון הcsl2 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.
- ניתן להגיש את התרגיל מספר פעמים, רק ההגשה האחרונה נחשבת.

5 דגשים ורמזים

- **הקפידו על כללי הזהב הבאים כדי להקל על העבודה:**
  - **הקוד תמיד מתקמפל** - בכל שלב של העבודה, ניתן לקמפל את הקוד ללא שגיאות או להגיע למצב זה על ידי מספר קטן של שינויים..
  - **כל הקוד שנכתב נבדק** - מאחר שהקוד תמיד ניתן להרצה, ודאו בכל הרצה שכל הבדיקות עוברות. הקפידו להיות במצב שבו על ידי לחיצת כפתור ניתן להריץ בדיקה אוטומטית לכל הקוד שכבר נכתב. דבר זה יחסוך זמן דיבוג יקר.

אם לא נתונות לכם בדיקות, כתבו אותן בזמן כתיבת הקוד עצמו.

- יש להקפיד על תכנון נכון של התוכנית וכתיבה נכונה ב-C.
- מומלץ להגדיר פונקציות עזר להמרת ADT אחד לאחר (למשל list ל-set ולהפך) כדי לנצל את חזקות הADTs.
- זכרו כי אין דרישות סיבוכיות, לכן שאפו לקוד פשוט וקריא על פני קוד "יעיל". קוד מסורבל ללא צורך אמיתי ייאבד נקודות.
- אתם רשאים להניח כי לא קיימות שגיאות נוספות מלבד אלו שפורטו.
- מומלץ להשתמש במאקרו assert המוגדר בקובץ assert.h כפי שהוסבר בתרגולים.
- במקרה של הסתבכות עם באג קשה:
  - בודדו את הבאג ושחזרו אותו בעזרת קבצי בדיקה (main) קטנים ככל הניתן. ניתן לעשות זאת על ידי comment לחלק מהשורות בקוד ובדיקה אם הבאג מתרחש, והמשך בהתאם. שיטה זו פותרת כמעט כל באג אפשרי, וברוב המקרים ביעילות.
  - ניתן להשתמש בדיבאגר או בהדפסות זמניות כדי לבדוק את מצב התכנית בכל שלב. אנו ממליצים להפנים ולהשתמש לפחות באחת משתי האפשרויות האלה על מנת לדבג את הקוד בהצלחה.

בהצלחה! 😊