מבוא לתכנות מערכות
תרגיל בית מספר 3 (ADT)
סמסטר אביב 2016-2017

תאריך פרסום : 17.5.2017

תאריך הגשה : 14.6.2017, 23:59

משקל התרגיל :10% מהציון הסופי (תקף)

מתרגל אחראי : דור גרנט (gdor.mtm@gmail.com)

1 הערות כלליות

- שימו לב: לא יינתנו דחיות במועד התרגיל. תכננו את הזמן בהתאם.
- לשאלות בנוגע לתרגיל נא לפנות למייל **gdor.mtm@gmail.com** וודאו שהיא לא נענתה כבר ב-F.A.Q ושהתשובה אינה ברורה ממסמך זה, מהדוגמא או מהבדיקות שפורסמו עם התרגיל.
- קראו מסמך זה עד סופו ועיברו על הדוגמא שפורסמה לפני תחילת הפתרון.
- חובה להתעדכן בעמוד ה- F.A.Q של התרגיל
- העתקות קוד בין סטודנטים יטופלו בחומרה!

2 חלק יבש

2.1 בחירת מבני נתונים

לפניכם מספר דוגמאות אשר דורשות שימוש במבני נתונים. עליכם להחליט איזה מבנה נתונים בסיסיים שנלמדו בקורס (רשימה, קבוצה, מחסנית) הוא המתאים ביותר לכל דוגמא. הסבירו מדוע הוא מתאים יותר מן האחרים.

1) ברשת החברתית פייסבוק נהוג לשמור לכל משתמש את החברים שלו, לשם כך יש צורך במבנה נתונים מתאים אשר ישמור מזהים ייחודיים לכל חברי המשתמש.
2) חברת ביטוח מעוניינת לשמור את מספרי הלקוחות אשר רשומים לחברה על פי הסדר בו נרשמו למערכת.
3) בעל מועדון מעוניין לייעל את מערכת שמירת החפצים של המועדון. כאשר מגיע לקוח למועדון והוא מעוניין להשאיר מעיל או כל פרט לבוש אחר להשגחה, הלקוח נותן את הפריט להשגחה ובתמורה מקבל מספר. במועדון הזה – לקוחות שמגיעים מאוחר יותר, עוזבים את המועדון מוקדם מלקוחות אחרים. כאשר הלקוח מעוניין לעזוב הוא נותן את המספר, ובאמצעות מספר זה מחפש העובד את הפריט שהשאיר הלקוח להשגחה. איזה מבנה נתונים מתאים לניהול מערכת שמירת החפצים של הלקוחות?

2.2 תכנות גנרי:

ברצוננו לממש אלגוריתם חיפוש בינארי המקבל מערך ממויין של איברים ואיבר לחיפוש, ומחזיר את האינדקס של האיבר אם הוא נמצא במערך, אחרת יחזיר -1.

1. כתבו את הפונקציה binaryFind המקבלת מערך של עצמים מטיפוס לא ידוע, ועצם לחיפוש ומחזירה את האינדקס המתאים. על הפונקציה להתאים לעצמים מכל סוג שהוא.
2. הסבירו בקצרה מה תפקידו וטיפוסו של כל פרמטר שיש לשלוח לפונקציה.

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;
};
```

כתבו את הפונקציה concatLists המקבלת שתי רשימות מקושרות ותנאי ומחזירה רשימה מקושרת חדשה אשר מכילה עותק של הרשימה השנייה משורשרת לסוף עותק הרשימה הראשונה ומכילה אך ורק איברים העומדים בתנאי הנתון, על התנאי להיות גנרי ככל האפשר.

לדוגמא, השרשור של הרשימה (1, 2, 3) עם הרשימה (4, 5, 6, 7) והתנאי שהתוצאה תכיל רק מספרים זוגיים ייצור את הרשימה (2, 4, 6).

בנוסף כתבו תכנית אשר יוצרת שתי רשימות לפי הדוגמא הנ"ל ומשתמשת בפונקציה concatLists כדי ליצור שתי רשימות חדשות, רשימה אחת תכיל רק מספרים אי זוגיים, ורשימה אחת תכיל רק מספרים ראשוניים.

הערות:

- ניתן לממש פונקציות עזר על מנת לפשט את הפתרון.
- יש להימנע משכפול קוד ככל הניתן וליצור קוד גנרי כראוי.
- יש להשלים את חתימת הפונקציה המופיעה למטה ואת מימושה.
- הניחו כי קיימת הפונקציה void destroyList(Node) אשר משחררת את כל הזיכרון עבור רשימה מקושרת נתונה (מלבד פנוקציה זו יש לממש את כל הפונקציות אשר אתם משתמשים בהן).
- התייחסו למצביע ל-NULL כאל רשימה ריקה.

```
Node concatLists (Node head1, Node head2, … ) {

//TODO: complete code …
```

3 חלק רטוב

3.1 מימוש ***GDT –Generic Data Type*** עבור רשימה מקושרת

בחלק זה נממש תחילה ADT גנרי עבור רשימה מקושרת. קובץ המנשק list_mtm1.h נמצא בתיקיית התרגיל על שרת הcsl2 . עליכם לכתוב את הקובץ list_mtm1.c הממשמ את מבנה הנתונים המתואר.

מאחר והרשימה גנרית, יש לאתחל אותה עם מספר מצביעים לפונקציות אשר יגדירו את אופן הטיפול בעצמים המאוחסנים ברשימה.

על מנת לאפשר למשתמשים ברשימה לעבור בצורה סדרתית על איבריה, לכל רשימה מוגדר איטרטור פנימי (יחיד) אשר בעזרתו יכול המשתמש לעבור על כל איברי הרשימה.

כדי לבדוק את התנהגות מבנה הנתונים, מסופקת לכם תוכנית קצרה בקובץ list_example_test.c. בנוסף, עליכם לכתוב בדיקת יחידה מקיפה יותר לרשימה, יפורט בחלק 3.3.

3.1.1 פעולות

1. listCreate – יצירת רשימה חדשה, בפעולה זו מוגדרות לרשימה הפעולות בעזרתן ניתן להעתיק ולשחרר עצמים ברשימה.
2. listDestroy – מחיקת רשימה קיימת תוך שחרור מסודר של כל הזיכרון שבשימוש.
3. listCopy – העתקת רשימה קיימת לעותק חדש, כולל העתקת האיברים.
4. listFilter – יצירת עותק חדש של הרשימה המכיל רק איברים המקיימים את התנאי שהועבר לפונקציה.
5. listGetSize – החזרת מספר האיברים ברשימה.
6. listGetFirst – החזרת האיטרטור הפנימי לתחילת הרשימה והחזרת האיבר הראשון.
7. listGetNext – קידום האיטרטור הפנימי והחזרת האיבר המוצבע על ידו.
8. listGetCurrent – החזרת האיבר המוצבע על ידי האיטרטור הפנימי.
9. listInsertFirst – הכנסת איבר לתחילת הרשימה.
10. listInsertLast – הכנסת איבר לסוף הרשימה.
11. listInsertBeforeCurrent – הכנסת איבר לפני האיבר עליו מצביע האיטרטור.
12. listInsertAfterCurrent – הכנסת איבר אחרי האיבר עליו מצביע האיטרטור.
13. listRemoveCurrent – הסרת האיבר המוצבע ע"י האיטרטור הפנימי מהרשימה.
14. listSort – מיון איברי הרשימה על פי פונקציית השוואה המסופקת לפונקציה.
15. listClear - הסרת כל האיברים ברשימה.

לכל פקודה ייתכנו שגיאות שונות. ניתן למצוא את השגיאות האפשריות לכל פקודה בתיעוד בקובץ list.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 ברשימה.
- במקרה של שגיאה יש לשמור על שלמות מבנה הנתונים ולוודא שאין דליפות זיכרון.
- להשתמש במבנה של רשימה מקושרת בלבד (לא מערך למשל).

3.2 מערכת לניהול הזמנות של חדרי בריחה

הטכניון החליט לתת לסטודנטים לפתוח חדרי בריחה בקמפוס, ומכיוון שהסטודנטים במת"מ ידועים ביכולות התכנות המעולות שלהם, הסטודנטים צריכים לבנות מערכת לניהול ההזמנות של חדרי הבריחה בקמפוס.

המערכת תתן לסטודנטים לפתוח חברות שמפעילות חדרי בריחה, שיכולות להריץ חדרי בריחה מסוגים שונים. הסטודנטים יוכלו להזמין חדרי בריחה ולבקש המלצה על חדר בריחה שמתאים לטעמם וכן לראות סטטיסטיקות לגבי חדרי הבריחה.

בשלב הראשון המערכת תופעל במנשק טקסטואלי על מחשב לינוקס, וכשהטכניון ייתן לפיתוח המערכת תקציב נוסף, המערכת תועבר למנשק גרפי ונוח. מכיוון שאנו רוצים שיהיה ניתן להשתמש במערכת שלנו גם לאחר המעבר, אנו נעצב את המערכת בעזרת טיפוסי נתונים מופשטים שיוכלו להתחבר למנשקים שונים לפי הצורך.

עליכם לממש את ה-ADT שנקרא EscapeTechnion, לאחר מכן עליכם להוסיף קוד מתאים לניתוח פקודות טקסט והפעלתן דרך מנשק ה-ADT.

שימו לב, עליכם לתכנן את מנשק ה-ADT באופן עצמאי - לא נתון קובץ מנשק EscapeTechnion.h.

בכדי להקל על עבודתכם - נתונים שני מימושים של מבני נתונים Set, List שמנשקיהם נתונים ב-set.h, list.h.

כל פעולות ההדפסה יבוצעו על ידי פונקציות ספרייה הנתונות בקובץ המנשק mtm_ex3.h.

מימוש פונקציות ההדפסה ומבני הנתונים נתונים בקובץ libmtm.a שמסופק לכם.

3.2.1 הפעלת התוכנית

התוכנית אשר תקרא mtm_escape, תופעל משורת הפקודה באופן הבא:

```
mtm_escape [-i <input_file>] [-o <output_file>]
```

- שימוש בפרמטר "i-" יגרום למערכת לקבל הוראות מהקובץ <input_file>. אחרת, המערכת תקבל הוראות מערוץ הקלט הסטנדרטי.(standard input)
- שימוש בפרמטר "o-" יגרום למערכת לשלוח את הפלט שהיא מפיקה לקובץ ששמו <output_file>. אחרת, המערכת תשלח את הפלט לערוץ הפלט הסטנדרטי.(standard output)
- שימו לב ששני הפרמטרים הם רשות (כלומר יכולים להיות 0,2 או 4 פרמטרים).
- שימוש ב "i-"או "o-" ללא העברת שמות הקבצים, או העברת ארגומנטים אחרים מלבד הנ"ל אינו חוקי !במקרים אלו יש לדווח על השגיאה המתאימה באמצעות `mtmPrintErrorMessage` עם פרמטר מתאים ולצאת מהתוכנית.
- שינוי סדר הארגומנטים (קודם "o-" ואז "i-") הינו חוקי .
- במקרה שמבנה הפרמטרים לתוכנית אינו תקין יש לדווח על שגיאה מסוג *MTM_INVALID_COMMAND_LINE_PARAMETERS.*
- אם הפרמטרים תקינים אך פתיחת אחד הקבצים נכשלת יש לדווח על.*MTM_CANNOT_OPEN_FILE*

3.2.2 הקלט לתוכנית

בין אם מערוץ הקלט הסטנדרטי ובין אם מקובץ, הקלט מורכב מסדרת שורות טקסט, כאשר בכל שורה תהיה פקודה אחת. ייתכנו שורות ריקות שלא יכילו פקודות כלשהן.

במידה והקלט הוא מערוץ הקלט הסטנדרטי נסיים את התוכנית ע"י שילוב המקשים ctrl+d (EOF). אחרת התוכנית תסיים את ריצתה ברגע שתגמור לקרוא את הקלט מהקובץ. ניתן להניח כי הקלט תקין מבחינה סינטקטית (כלומר, הפקודות מאויתות נכון ומספר הפרמטרים לכל פקודה תקין). יכולות להיות שורות שיתחילו בסימן סולמית ("#"), אלו שורות הערה ואין להתייחס אליהן אלא להמשיך לשורה הבאה.

כלומר כל שורת קלט היא אחת משלושת הסוגים הבאים:

1. שורה המכילה פקודה חוקית (ייתכנו רווחים וטאבים בתחילת השורה).
2. שורה ריקה (שורה שאינה מכילה כלל תווים או לחילופין מכילה רק רווחים וטאבים).
3. שורת הערה (שורה המתחילה בסולמית או ברווחים וטאבים ואחריהם סולמית).

מבנה שורת הפקודה הוא כדלקמן:

<command> <subcommand> [<arg1> <arg2> ...]

כאשר <command> הוא שם הפקודה, <subcommand> הוא שם תת הפקודה ו<arg1> <arg2> היא רשימת הפרמטרים לפקודה. אורך רשימת הפרמטרים יכול להשתנות מפקודה לפקודה וייתכנו פרמטרים אופציונאליים אשר לא תמיד יופיעו.

בין כל זוג מילים יופיע תו רווח/טאב אחד לפחות אך ייתכנו מספר רווחים/טאבים בין כל זוג מילים.

3.2.3 טיפול בשגיאות

במקרה של שגיאות, על EscapeTechnion להחזיר ערך שגיאה מתאים לתכנית הראשית, והתוכנית תדווח על השגיאה בעזרת הפונקציה mtmPrintErrorMessage.

במקרה שיש כמה שגיאות אפשריות יש לדווח על השגיאה החשובה ביותר כאשר סדר החשיבות מוגדר בהגדרת הטיפוס MtmErrorCode בקובץ mtm_ex3.h.

במקרה של כישלון באתחול המערכת בגלל מבנה שורת פקודה לקוי, תקלה בפתיחת קבצי הקלט/פלט או תקלה בהקצאת זיכרון יש לדווח על השגיאה המתאימה ולצאת מהמערכת תוך שחרור מסודר ומפורש של כל הזיכרון והמשאבים האחרים.

מאחר וברצוננו להשתמש בADT הראשי (EscapeTechnion) בעתיד, חובה לוודא כי במקרה של שגיאה כלשהי המערכת נשמרת במצב תקין. כלומר - לאחר גילוי שגיאה מצב המערכת יהיה כאילו לא בוצעה פקודה כלשהי.

הטיפול בשגיאות צריך להתבצע על ידי החזרת קוד שגיאה מתאים מ- EscapeTechnion - והדפסת המידע אודות השגיאה צריכה להתבצע מהתוכנית המשתמשת ב- EscapeTechnion וזאת על מנת לאפשר גמישות בדיווח ההודעות בעתיד.

ניתן להניח כי לא קיימות שגיאות נוספות מלבד אלו המפורטות לכל פקודה (ושגיאות *MTM_OUT_OF_MEMORY* היכולות להתרחש בכל נקודה בתוכנית כתלות במימוש).

בכל מקרה, לא ניתן להשתמש ב exit לשם יציאה לאחר שגיאה ומאחר ופונקציה זו מקשה על שינויים עתידיים בקוד ופוגעת בקריאותו.

3.2.4 פקודות

על המערכת להכיר את הפקודות הבאות:

הוספת חברה למערכת

```
company add <email> <faculty>
```

תיאור הפעולה: פקודה זו מוסיפה חברת חדרי בריחה חדשה למערכת.

פרמטרים:

`<email>` – המייל של החברה, מייל חוקי הוא כל מחרוזת שמכילה תו '@' אחד בדיוק.

`<faculty>` – הפקולטה שהחברה שייכת אליה, הפקולטות נתונות ב-enum בקובץ mtm_ex3.h ויתקבלו על פי מספרן המזהה ב-enum.

שגיאות:

MTM_INVALID_PARAMETER - פרמטר לא חוקי.

MTM_EMAIL_ALREADY_EXISTS - המייל שהוכנס כבר שייך למשתמש מסוג כלשהו במערכת.

הסרת חברה מהמערכת

```
company remove <email>
```

תיאור הפעולה: פקודה זו מסירה חברה קיימת מן המערכת יחד עם כל החדרים שלה. אסור להסיר חברה שיש הזמנות קיימות לאחד החדרים שלה.

פרמטרים:

`<email>` - מייל החברה אותה יש להסיר.

שגיאות:

MTM_INVALID_PARAMETER – פרמטר לא חוקי.

MTM_COMPANY_EMAIL_DOES_NOT_EXIST - המייל שהוכנס לא שייך לאף חברה במערכת.

MTM_RESERVATION_EXISTS - קיימת הזמנה לאחד החדרים של החברה.

הוספת חדר בריחה למערכת

```
room add <email> <id> <price> <num_ppl> <working_hrs> <difficulty>
```

תיאור הפעולה: פקודה זו מוסיפה חדר בריחה לאוסף חדרי הבריחה במערכת.

פרמטרים:

`<email>` - מייל החברה שחדר הבריחה שייך לה.

`<id>` - המספר המזהה של החדר, לפקולטה אחת לא יהיו שייכים שני חדרים עם אותו מספר מזהה. המזהה הוא מספר חיובי.

`<price>` - מחיר הכניסה לחדר לבן אדם, מספר חיובי, כפולה של 4.

`<num_ppl>` - מספר האנשים המומלץ לכניסה לחדר, מספר חיובי.

`<working_hrs>` - שעות העבודה של החדר, מחרוזת מהצורה "xx-yy" כאשר xx שעת הפתיחה בכל יום ו-yy שעת הסגירה. שימו לב כי השעות שלמות והערכים החוקיים עבור כל אחד מהקלטים נעים בין 00-24, וכן xx<yy.

`<difficulty>` - רמת הקושי של החדר. מספר בתחום 1-10.

שגיאות:

MTM_INVALID_PARAMETER – פרמטר לא חוקי.

MTM_COMPANY_EMAIL_DOES_NOT_EXIST - המייל שהוכנס לא שייך לאף חברה במערכת.

MTM_ID_ALREADY_EXISTS - קיים חדר בעל מספר מזהה כזה ששייך לאותה הפקולטה.

הסרת חדר בריחה מהמערכת

```
room remove <faculty> <id>
```

תיאור הפעולה: פקודה זו מסירה חדר בריחה מן המערכת. אסור להסיר חדר בריחה שיש עבורו הזמנות קיימות.

פרמטרים:

`<faculty>` - הפקולטה שהחדר שייך לה, הפקולטות נתונות בקובץ mtm_ex3.h.

`<id>` - המספר המזהה של חדר הבריחה אותו יש להסיר.

שגיאות:

MTM_INVALID_PARAMETER – פרמטר לא חוקי.

MTM_ID_DOES_NOT_EXIST – המספר המזהה שהוכנס לא קיים עבור הפקולטה.

MTM_RESERVATION_EXISTS - קיימת הזמנה לחדר שברצוננו למחוק.

הוספת לקוח למערכת

```
escaper add <email> <faculty> <skill_level>
```

תיאור הפעולה: פקודה זו מוסיפה משתמש חדש עבור לקוח שיכול להזמין חדרי בריחה.

פרמטרים:

`<email>` - המייל של הלקוח, מייל חוקי הוא כל מחרוזת שמכילה תו '@' אחד בדיוק.

`<faculty>` - הפקולטה שהלקוח שייך אליה, הפקולטות נתונות בקובץ mtm_ex3.h.

`<skill_level>` - רמת המיומנות של הלקוח בחדרי בריחה, מספר בתחום 1-10.

שגיאות:

MTM_INVALID_PARAMETER - פרמטר לא חוקי.

MTM_EMAIL_ALREADY_EXISTS - המייל שהוכנס כבר שייך למשתמש מסוג כלשהו במערכת.

הסרת לקוח מהמערכת

```
escaper remove <email>
```

תיאור הפעולה: פקודה זו מסירה משתמש של לקוח ואת כל ההזמנות שהוא ביצע.

פרמטרים:

`<email>` - המייל של הלקוח, מייל חוקי הוא כל מחרוזת שמכילה תו '@' אחד בדיוק.

שגיאות:

MTM_INVALID_PARAMETER - פרמטר לא חוקי.

MTM_CLIENT_EMAIL_DOES_NOT_EXIST - המייל שהוכנס לא שייך לאף לקוח במערכת.

יצירת הזמנה ללקוח

```
escaper order <email> <faculty> <id> <time> <num_ppl>
```

תיאור הפעולה: פקודה זו יוצרת הזמנה עבור חדר בריחה כלשהו, התשלום מתבצע בעת **מימוש ההזמנה (ההגעה לחדר)** והזמנות של משתמשים מאותה הפקולטה שאליה שייך החדר מקבלים הנחה של 25% מהמחיר.

פרמטרים:

`<email>` - המייל של הלקוח, מייל חוקי הוא כל מחרוזת שמכילה תו '@' אחד בדיוק.

`<faculty>` - הפקולטה שהחדר שייך אליה, הפקולטות נתונות בקובץ mtm_ex3.h.

`<id>` - המספר המזהה של החדר בפקולטה.

`<time>` - הזמן שבו רוצים להגיע לחדר הבריחה, הזמן מתקבל בפורמט day-hour כאשר day הוא מספר הימים שנשארו עד שיום ההזמנה יגיע, חייב להיות מספר אי-שלילי (אם 0=day אז ההזמנה להיום, אם 10=day אז ההזמנה עוד 10 ימים) ו-hour הוא הזמן ביום שבו רוצים לעשות את ההזמנה, שחייב להיות בין 0 ל-23. כל הזמנה נמשכת שעה שלמה אחת, יש להתחשב בשעות הפתיחה של החדר, לא

ניתן להזמין שני חדרים באותו השעה.

`<num_ppl>` - מספר האנשים שעבורם מתבצעת ההזמנה, מספר חיובי כלשהו.

שגיאות:

MTM_INVALID_PARAMETER - פרמטר לא חוקי.

MTM_CLIENT_EMAIL_DOES_NOT_EXIST - המייל שהוכנס לא שייך לאף לקוח במערכת.

MTM_ID_DOES_NOT_EXIST - לא קיים חדר עם מספר מזהה כזה בפקולטה המבוקשת.

MTM_CLIENT_IN_ROOM - ללקוח יש הזמנה קיימת לחדר באותה השעה.

MTM_ROOM_NOT_AVAILABLE - החדר המבוקש לא זמין בזמן המבוקש.

הזמנת חדר מומלץ

```
escaper recommend <email> <num_ppl>
```

תיאור הפעולה: פקודה זו יוצרת הזמנה לזמן הכי קרוב שבו החדר פנוי עבור המשתמש לחדר הכי מומלץ עבורו. החדר הכי מומלץ יחושב על פי הנוסחא הבאה (החדר שמקבל את התוצאה **הנמוכה** ביותר):

(P_r-P_e)^2+(difficulty-skill_level)^2

כאשר P_r הוא מספר האנשים המומלץ לחדר, P_e מספר האנשים שעבורם מתבצעת ההזמנה, difficulty הוא רמת הקושי של החדר ו-skill_level רמת המיומנות של המשתמש המזמין. אם יש שני חדרים שקיבלו את אותה התוצאה אז נתן עדיפות לחדר שנמצא בפקולטה עם ה-id הכי קרוב ל-id של הפקולטה של מזמין החדר (אם גם פה נתקלנו ברב משמעות, הפקולטה עם ה-id המינימלי מבין השניים), ואז על פי id מינימלי של החדר בפקולטה.

פרמטרים:

`<email>` – המייל של הלקוח, מייל חוקי הוא כל מחרוזת שמכילה תו '@' אחד בדיוק.

`<num_ppl>` – מספר האנשים שעבורם מתבצעת ההזמנה, מספר חיובי כלשהו.

שגיאות:

MTM_INVALID_PARAMETER - פרמטר לא חוקי.

MTM_CLIENT_EMAIL_DOES_NOT_EXIST - המייל שהוכנס לא שייך לאף משתמש במערכת.

MTM_NO_ROOMS_AVAILABLE - לא קיימים חדרים במערכת.

סיום יום עבודה

```
report day
```

תיאור הפעולה: פקודה זו מדפיסה את כל ההזמנות שמומשו במהלך אותו היום (אלו שהגיע זמנן והאנשים הלכו לחדר, לא ההזמנות שנשלחו), מסיר אותן מהמערכת ומקדם את זמן המערכת ביום. ההדפסה מתבצעת על ידי פונקציית הדפסה שנמצאת בקובץ mtm_ex3.h, ויש לעקוב אחרי התיעוד שם. יש להדפיס את **ההזמנות על פי הזמן ביום בו הגיעו אליהם, ומיון משני לפי ה-id של הפקולטה**, מהנמוך לגבוה (אם עדיין יש חוסר בהירות - מיון שלישי על פי הid המינימלי של החדר בפקולטה).

הדפסת הפקולטות המנצחות

```
report best
```

תיאור הפעולה: פקודה זו מדפיסה את שלושת הפקולטות שהרוויחו הכי הרבה כסף מאנשים שהלכו לחדרים שלהן. אם יש מספר פקולטות עם אותו רווח מהחדרים שלהן, יש להדפיסן החל מה-id הנמוך ל-id הגבוה. יש להשתמש בפונקציית ההדפסה שנמצאת בקובץ mtm_ex3.h ולשים לב לתיאור שלה.

3.2.5 הגבלות על המימוש

על המימוש שלכם לעמוד במגבלות הבאות:

- אין חסם על הרשומות במערכת מכל סוג שהוא.
- את כל ההדפסות לפלט יש לבצע באמצעות מחרוזות המתקבלות באמצעות פונקציות הפלט המסופקות בקובץ mtm_ex3.h. ניתן למצוא את אופן השימוש בפונקציות בקובץ זה.
- לכל המחרוזות הנשמרות במערכת יש להקצות את גודל הזיכרון הדרוש וללא זיכרון מיותר.
- לצורך מימוש מבני הנתונים נתונים לכם מימושים של רשימה (List) וקבוצה (Set) אשר המנשקים שלהם מוגדרים בקבצים h. ומימושם נמצא בספריה המצורפת לתרגיל.
  - לכל מבני הנתונים בתרגיל יש לבחור האם להשתמש ברשימה, בקבוצה או מילון בהתאם לנלמד בקורס.

<u>הבהרה:</u> כדי שלא יהיו תלויות בין התרגילים קיים קובץ h שנקרא list.h שבאמצעותו תקשרו את התוכנית שלכם למימוש עובד של רשימה בספריה שנצרף לכם (אין צורך להשתמש בlist_mtm1 שמימשתם בחלק 3.1).

- 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 הסופי.
- יש לכתוב את הקוד בפונקציות קצרות וברורות.
- מימוש כל המערכת (מלבד התוכנית הראשית) 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.
- ניתן לכתוב את חלק ניתוח הקלט (הקורא לפעולות ה ADT הראשי) ללא חלוקה לADTs.

3.2.6 הנחות

מאחר והתוכנית הראשית תשמש את התוכנה זמנית, ניתן להניח את ההנחות הבאות בשלב ניתוח הקלט:

- המחרוזות לפקודות במנשק הטקסט אינן מכילות רווחים.
- ניתן להניח שמספר הפרמטרים לכל פקודה נכון.
- ניתן להניח שיתנו רק שמות חוקיים של פקודות.
- לכל פרמטר מספרי ניתן להניח שאכן יסופק מספר ,עם זאת אין להניח שהמספר בטווח החוקי .
- לכל פונקצית ADT - ניתן להניח שהפרמטר של ה-ADT עצמו אינו NULL ועליכם לוודא את העניין עם assert.

3.3 בדיקות יחידה

כאמור על המימוש להיות מורכב מטיפוסי נתונים מופשטים. עליכם לספק לכל אחד מה-ADT אותם תכתבו בדיקות יחידה. בדיקת יחידה היא קוד אשר מוודא את נכונות ה-ADT וניתן להרצה בקלות כך שבמקרה של באג בקוד, הבדיקה תיכשל ויהיה קל לאתר את הבאג.

עליכם לממש בדיקת יחידה דומה לכל ה-ADT שתגישו. למשל עבור Account עליכם לספק את הקובץ account_test.c אשר בודק בצורה אוטומטית את ה-ADT.

בנוסף עליכם ליצור בדיקת יחידה עבור הרשימה שמימשתם בסעיף 3.1

להלן מספר דגשים לגבי כתיבת בדיקות יחידה:

- אין להוסיף פונקציות למנשק של ה ADT-כדי לאפשר עוד בדיקות .ניתן להסתפק בבדיקות אותן ניתן לבצע בעזרת המנשק הקיים.
- אין צורך לבדוק פקודות קלט/פלט באמצעות בדיקות היחידה למעט מקרים טריוויאליים (לדוגמא עודף/מחסור בפלט/קלט).
- הקפידו לבדוק את התנהגות הפונקציה גם במצבי שגיאות ולא רק בעבור קלט מוצלח.
- הקפידו על איכות הקוד גם בבדיקות היחידה – הבדיקות צריכות להיות קצרות ,מחולקות לפונקציות וקריאות ,כך שבמקרה של כישלון יהיה קל למצוא את מקור התקלה.
    - מומלץ לחלק את הבדיקות כך שלכל פונקציה קיימת פונקציה הבודקת אותה.
- כדי לראות כיצד צריכה להיראות בדיקת יחידה – הסתכלו על הבדיקה המסופקת עבור הרשימה, הנמצאת בקובץ list_example_test.c.

<u>המלצה</u>: כתיבת בדיקות לוקחת זמן אך מקלה משמעות את תחזוקת הקוד בהמשך ועל צמצום הזמן הדרוש למציאת תקלות בתוכנית. מומלץ לכתוב את הבדיקות במקביל לכתיבת הקוד, כך למשל לפני שאתם כותבים פונקציה מסוימת הכינו כבר את הפונקציה הבודקת אותה. את כתיבת הבדיקות כדאי לעשות כאשר חושבים כיצד ניתן "להכשיל" את הקוד ובכך לוודא את איכות הקוד במקרי קצה. כתיבת כל הבדיקות לאחר סיום התרגיל תהיה מעיקה משמעותית ותתבצע לאחר השלב בו הבדיקות יוכלו לעזור בפועל.

3.4 *Makefile*

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

הכלל הראשון בMakefile יקרא mtm_escape ויבנה את התוכנית mtm_escape המתוארת בסעיף 3.2. בנוסף יופיע כלל בשם tests אשר יבנה את בדיקות היחידה. לכל בדיקת יחידה (למשל list_test.c) תיבנה תכנית בשם מתאים (במקרה זה list_test) אשר תאפשר את הרצת הבדיקה.

- יש להניח שהקבצים מסודרים במבנה הבא :כל בדיקות היחידה נמצאות תחת תיקיה בשם tests וכל שאר הקבצים בתיקיה בה נמצא ה.Makefile
- יש לכתוב את הקובץ כפי שנלמד וללא שכפולי טקסט.

3.5 הידור ,קישור ובדיקה

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

```
gcc -std=c99 -o mtm_escape -Wall -pedantic-errors -Werror -DNDEBUG *.c   -L.  -lmtm   ➢
```

משמעות הפקודה:

- "std=c99–": קביעת התקן לשפה להיותC99
- "o mtm_escape–": 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.
  - בנוסף עליכם לוודא שתרגיל מתקמפל עם הקבצים אשר מסופקים על ידינו .אין להגיש קבצים אלה!
- ".L–": גורם ל gcc לחפש ספריות בתיקיה הנוכחית
- "lmtm–": קישור לספריה שמסופקת לכם ע"י הצוות

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. ע"מ להצליח בבדיקה שכזו על התוכנית לעבור הידור, לרוץ את הבדיקה בשלמות ולעבור השוואה של קבצי הפלט עם קבצי הפלט המצופים. ההשוואה תתבצע בעזרת הפקודה diff המחזירה את ההבדלים בין זוג קבצים או אינה מדפיסה כלום במקרה של זהות בין הקבצים. ע"מ לבדוק את תכניתכם מסופק קובץ בדיקה עם טסטים בסיסיים לכל סוג פקודה וקובץ הפלט המצופה ממנו. את הקובץ ניתן למצוא תחת ספרית הקורס בcsl2 בכתובת הבאה:

```
➢ ~mtmchk/public/1617b/ex3/
```

העתיקו קבצים אלה לספרית העבודה שלכם.

לבדיקת הפתרון שלכם ניתן להריץ את הפקודות הבאות

```
➢ ./mtm_escape -i test1.in -o tempout 2> temperr
➢ diff test1.out tempout
➢ diff test1.err temperr
```

אם אין הבדלים הפקודות לא ידפיסו כלום, אחרת יודפסו ההבדלים. לשם צפייה בהבדלים בממשק גרפי ובצורה קריאה יותר ניתן להיעזר בsdiff או kompare (שתי התוכנות דורשות ממשק גרפי).

שימו לב שעל התוכנית שלכם לעבור גם עבור קלט/פלט סטנדרטי וגם עבור שימוש בקבצים כפי שהוגדר קודם.

התכנית שלכם תיבדק עם בדיקות נוספות!

הבדיקות המסופקות מכסות רק חלק בסיסי מהמקרים האפשריים. בדיקת הקוד מהווה חלק מהתרגיל!

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

- כתבו בדיקות נוספות בעצמכם בהתאם למפרט.
- וודאו את נכונות התכנית שלכם במקרים כללים ובמקרי קצה.
- מומלץ לחשוב על הבדיקות 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 ב valgrind. זהו כלי אשר מותקן בשרת 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.

כמו כן, עליכם **להגיש את החלק היבש יחד עם ההגשה הרטובה.**

אין צורך להגיש את הקוד מודפס.

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 וה makefileמכווצים לקובץ) zip ולא שום סוג כיווץ אחר (כאשר כל הקבצים עבור הפתרון מופיעים בתיקיית השורש בתוך קובץ הzip, ואילו כל בדיקות היחידה (כולל בדיקת היחידה עבור הרשימה מהרטוב הראשון) נמצאות תחת התיקייה tests בתוך קובץ הzip ותיקיה בשם list_mtm שבתוכה יהיה המימוש עבור החלק הראשון של התרגיל הרטוב. כמו כן, הוסיפו את הקובץ dry.pdf שמכיל את התשובות שלכם לחלק היבש של התרגיל לתיקיית השורש של הzip.
- <u>אין להגיש אף קובץ מלבד קבצי **h** וקבצי **c** אשר כתבתם בעצמכם ואת ה **makefile** אשר נדרשתם לעשות וכמובן את קובץ התשובות שלכם לחלק היבש.</u>
- הקבצים אשר מסופקים לכם: list.h, set.h, libmtm.a, mtm_ex3.h 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) והקובץ list_mtm1.h יימצא תחת התיקייה list_mtm.</u>
- יש להגיש את בדיקות היחידה שכתבתם לכל ADT, למשל עבור ADT בשם foo אשר מורכב בשני קבצים שהם foo.h ו foo.c- יש להגיש קובץ בשם foo_test.c המכיל בדיקות עבור foo. הקפידו על שמות נכונים ועקביים לקבצי הבדיקה עדי להימנע מבעיות עם הבודק האוטומטי.
- ע"מ לבטח את עצמכם נגד תקלות בהגשה האוטומטית:
  - שימרו את קוד האישור עבור ההגשה .עדיף לשלוח גם לשותף.
  - שימרו עותק של התרגיל על חשבון ה csl2שלכם 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 ללהגיע למצב זה בכמה שניות עבודה.
  - כל הקוד שנכתב נבדק - מאחר והקוד תמיד ניתן להרצה, ודאו בכל הרצה שכל הבדיקות עוברות. הקפידו להיות במצב שבו ע"י לחיצת כפתור ניתן להריץ בדיקה אוטומטית לכל הקוד שכבר נכתב. דבר זה יחסוך זמן דיבוג יקר. אם לא נתונות לכם בדיקות, כתבו אותם 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.
- יש להקפיד להשתמש בפונקציות שניתנו לכם ליצירת מחרוזות ,ולהקפיד שהפונקציות שאתם כותבים היוצרות מחרוזות ,ייצרו אותם כך שתעברו את הטסטים המסופקים .טעות קלה של אפילו תו רווח יחיד תכשיל בדיקה שלמה.
- הנכם רשאים להניח כי לא קיימות שגיאות נוספות מלבד אלו שפורטו.
- מומלץ להשתמש במאקרו assert המוגדר בקובץ assert.h כפי שהוסבר בתרגול מס' 2.
- במקרה של הסתבכות עם באג קשה:
  - בודדו את הבאג ושחזרו אותו בעזרת קבצי בדיקה) קובץ + main קובץ קלט (קטנים ככל הניתן .ניתן לעשות זאת ע"י מחיקת חלק מהשורות בקוד ובדיקה אם הבאג מתרחש ,והמשך בהתאם .שיטה זו פותרת כמעט כל באג אפשרי ,וברוב המקרים ביעילות.
  - ניתן להשתמש ב"דיבאגר "או הדפסות זמניות כדי לבדוק את מצב התכנית בכל שלב .לפחות אחת משתי האפשרויות האלה היא חובה ע"מ להצליח ב"דיבוג "הקוד.

סקריפט הבדיקה הסופית

על מנת לוודא שקובץ ההגשה שלכם תקין ,לפני שאתם מגישים ,עליכם להריץ דרך חשבון ה T2-שלכם סקריפט ייעודי שהכנו .הסקריפט מוודא שמבנה קובץ ה zip-שאתם מגישים תקין, ושהקוד שלכם עובר את הבדיקות שפורסמו עם התרגיל .את סקריפט הבדיקה מריצים ע"י הפקודה הבאה (על ה-csl2):

~mtmchk/public/1617b/ex3/final_check **<submission file>**

כאשר **<submission file>** הוא קובץ ה zip-שאתם מעוניינים להגיש.

לדוגמא ,כך נראת הרצה מוצלחת של סקריפט הבדיקה:

```
>~mtmchk/public/1617b/ex3/final_check hw3.zip
This script will now compile your code and run your program with the published tests.
Continue? (y/n)y
Compiling your generic list ADT...
Running the published list_example_test.c test... success
Compiling your mtm_escape program...
Running the published mtm_escape test:
without '-i'/'-o' flags... success
with the '-i' flag... success
with the '-o' flag... success
Everything went well :)
Final check passed.
```

בהצלחה!