מכון טכנולוגי חולון
Holon Institute of Technology

COMPUTER SCIENCE DEPARTMENT מחלקה למדעי המחשב

סדנה מתקדמת בתכנות 61108
סמסטר ב' תשפ"ב

מטלה 2

מטריצות, רשימות מקושרות

דרישות חובה והסברים כלליים לתרגיל הגשה:

הדרישות הכלליות המופיעות בסעיף זה, הינם דרישות חובה. אי קיום הדרישות יוביל לפסילת שאלות וחבל.

- תרגיל הגשה זה מורכב מ-5 שאלות.
- לכל שאלה ישנו קובץ 'c' מצורף, המכיל את השלד של התרגיל. אין לשנות את שמות הקבצים.
  למשל: עבור השאלה הראשונה שם הקובץ הוא ex2_q1.c, שאלה שנייה: ex2_q2.c וכן הלאה...
- בכל קובץ קיימים מספר פונקציות, חלקם עליכם לממש.
  אין לשנות את שמות הפונקציות, ואין לשנות את חתימת הפונקציות
- את המימושים יש לכתוב החל מהשורה שלאחר ההערה: // your code
  אין למחוק את הערות אלו
- בכל קובץ שאלה מופיעה פונקציה student_id() - על הפונקציה להחזיר את מס' תעודת הזהות של הסטודנט, כפי שמופיעה במודל.
- לפני חלק מהפונקציות מופיעה הערה:
  // DO NOT CHANGE the following function
  אין לשנות פונקציה המופיעה לאחר הערה כזאת
- לפני חלק מהפונקציות מופיעה הערה:
  // DO NOT CHANGE from this point
  אין לשנות או להוסיף קוד החל מהערה זו עד סוף הפונקציה
- אין לשנות את שמות הפונקציות שכבר מופיעות בקבצים
- **למעט שאלה מס' 1 בה אתם נדרשים להוסיף פונקציה אחת,** אין להוסיף פונקציות חדשות לאף אחד מין התרגילים.
- אין להוסיף include או define מעבר למה שכבר מופיע
- יש להתייחס באופן מלא להערות נוספות אשר מופיעות בקבצי ההרצה, אין למחוק הערות אלה.

בדיקת התרגיל תתבצע באופן אוטומטי תחת סביבת Linux.
רצוי וניתן לבדוק את התרגילים באמצעות Online C Compiler.
https://www.onlinegdb.com/online_c_compiler

שימו לב:
עבור קליטת משתנים, יש להשתמש בפונקציה scanf ולא scanf_s.

שאלה 1:

נגדיר את שכניו של איבר $a[i][j]$ במטריצה להיות כל האיברים הסובבים אותו בקו ישר ובאלכסון. לדוגמא:

$$\begin{bmatrix} 7 & 6 & 1 \\ 8 & 3 & 2 \\ 9 & 4 & 0 \end{bmatrix}$$

עבור המטריצה הבאה, לאיבר המרכזי שערכו **3** יש 8 שכנים, עבור האיברים שנמצאים בפינות יש 3 שכנים.

נגדיר מבנה "מספר מעורב" כלומר מספר שלם שאחריו שבר (שערכו קטן מ-1).
המבנה מכיל שלושה שדות מטיפוס int: חלק שלם (num), מונה (numerator) ומכנה (denominator)

```
typedef struct fraction
{
        int num, numerator, denominator;
} fraction;
```

כתבו פונקציה בשם matrixAverageNeighbor שמקבלת מטריצה A המיושמת כמערך דו-ממדי סטטי וגדליה. על הפונקציה להקצות מערך דו-ממדי חדש - מטריצה B מסוג **Fraction** בעלת אותם גדלים כמו המטריצה A כך שכל איבר $b[i][j]$ שלה יהיה שווה לממוצע של כל שכני האיבר $a[i][j]$.

דוגמה:

$$A:\begin{bmatrix} 5 & 12 & 6 & 8 \\ 4 & 7 & 0 & 9 \\ 13 & 20 & 8 & 2 \\ 18 & 0 & 2 & 6 \end{bmatrix} \rightarrow B:\begin{bmatrix} 7\frac{2}{3} & 4\frac{2}{5} & 7\frac{1}{5} & 5 \\ 11\frac{2}{5} & 8\frac{1}{2} & 9 & 4\frac{4}{5} \\ 9\frac{4}{5} & 6\frac{1}{2} & 5\frac{3}{4} & 5 \\ 11 & 12\frac{1}{5} & 7\frac{1}{5} & 4 \end{bmatrix}$$

שימו לב: עבור המטריצה A ערכי המטריצה B צריכים להיות בדיוק כפי שהם מופיעים בדוגמא. אם איבר של המטריצה B הוא בפועל מספר שלם, אז השדה numerator של המשתנה יהיה 0 וערך השדה denominator שלו יכול להיות שרירותי.

הפונקציה תחזיר את הכתובת של המטריצה הדינאמית B החדשה.

ניתן להניח שיש בזיכרון מספיק מקום להקצאה.

יש להשתמש בפונקציית עזר בשם neighborFractionAverage אשר מקבלת מערך דו-ממדי סטטי עם קואורדינטות i ו- j של איברו ומחזירה טיפוס מסוג fraction.

יש למלא בהתאמה את שאר הפונקציות המופיעות בקובץ השלד:
הקצאת מטריצה דינאמית, הדפסת מטריצה דינאמית ושחרור מטריצה דינאמית.

יש לכתוב את הפונקציה להדפסת המטריצה B לפי ייצוג מסוג double, עם 2 ספרות אחרי הנקודה.

לשיקולכם: בשאלה זו מותר להוסיף פונקציית עזר **אחת בלבד** נוספת לשימושכם.
מיקום הפונקציה יוגדר בהתאם באזור מוגדר בקובץ השלד.

שאלה 2:

כתבו פונקציה בשם createArrayAndList המקבלת מטריצה כמערך דו-ממדי סטטי וגדליה. הפונקציה מוצאת את כל אחד מאיברי המטריצה שערכם מהווים סדרה חשבונית ביחד עם ערכי ה-i ו-j שלהם (המתחילה ב-i ומסתיימת ב-a[i][j]).

לדוגמא:

$a[1][2] = 3$ (הסדרה משמאל לימין: 1, 2, 3)
$a[2][4] = 6$ (הסדרה משמאל לימין: 2, 4, 6)
$a[4][2] = 0$ (הסדרה משמאל לימין: 4, 2, 0)

על הפונקציה לבנות מערך דינאמי חד-ממדי של רביעיות, וכן רשימה מקושרת **חד כיוונית** של רביעיות. כל רביעייה במערך או ברשימה תכיל את ערכו של איבר המטריצה שעונה לתנאי הנ"ל, את הקואורדינטות i ו- j (מס' השורה ומס' העמודה) של אותו האיבר במטריצה ואת הערך d של הסדרה החשבונית. הפונקציה תחזיר את גודל המערך (ששווה לאורכה של הרשימה), ותעביר (reference by) את המערך ואת הרשימה.

לדוגמא, עבור המטריצה המקורית:

i / j	0	1	2	3	4
0	0	6	5	6	6
1	8	9	5	6	7
2	7	6	5	4	7
3	9	8	1	6	7

סדרת הרביעיות תהיה:

איבר	0	6	7	4	1
i	0	0	1	2	3
j	0	3	4	3	2
d	0	3	3	1	-1

5. והפונקציה תחזיר

למימוש הרביעיות יש להשתמש בטיפוס struct four
בעל ארבעה שדות נומריים. איברי המערך יהיו מבנים מסוג זה.
כל איבר של הרשימה יהיה מורכב משני שדות:
נתון ומצביע לאיבר הבא. הנתון הוא struct four
(כמו איברי המערך).

```
typedef struct four
{
        int i, j, d, value;
} four;
```

```
typedef struct list
{
        four data;
        struct list *next;
} list;
```

יש למלא ולהשתמש בהתאם בשאר הפונקציות המופיעות בקובץ השלד:
(1) פונקציה בשם createFour המקבלת ארבעה מספרים שלמים ומחזירה רביעייה אחת המורכבת מארבעת הפרמטרים שקיבלה, (2) פונקציה להקצאת איבר כלשהו ברשימה מקושרת, (3) פונקציה להדפסת מערך, (4) פונקציה להדפסת רשימה ו- (5) פונקציה לשחרור אובייקטים שהוקצו באופן דינאמי.

יש להתייחס לחתימות הפונקציות כפי שהן כתובות בשלד, ולהנחיות שמופיעות בהן.
ניתן להניח שיש בזיכרון מספיק מקום להקצאה, יש להתייחס לכל מקרי הקצה האפשריים (רשימה ריקה וכו'). **אין להוסיף פונקציות נוספות מעבר לפונקציות הקיימות.**

שאלה 3:

כתבו פונקציה בשם splitList אשר מקבלת כתובת של מצביע של רשימה מקושרת **חד כיוונית** (ללא איבר דמה) של מספרים שלמים אי-שליליים. על הפונקציה להסיר מהרשימה את כל אחד מהאיברים שגדול מהאיבר הקודם וגם גדול מהאיבר הבא. יש לבנות רשימה חדשה (ללא איבר דמה) המורכבת מהמספרים שנמחקו מהרשימה המקורית. יש לשמור ברשימה החדשה על הסדר של האיברים שבו הם הופיעו ברשימה המקורית. **על הפונקציה להחזיר את גודל הרשימה המקושרת החדשה**

לדוגמא,
הרשימה המקורית בעלת תוכן 5 ,4 ,8 ,9 ,1 ,6 ,3 (משמאל לימין) תעודכן לרשימה המורכבת ממספרים 4 ,8 ,1 ,3 (משמאל לימין). איברי הרשימה החדשה יהיו 5 ,9 ,6 (משמאל לימין).

הפונקציה תחזיר מצביע לרשימה החדשה.

```
typedef struct list
{
        int data;
        struct list *next;
} list;
```

שימו לב:

- המצביע לרשימה המקורית יכול להשתנות אם האיבר הראשון שלה נמחק.
- האיבר הראשון נמחק מהרשימה המקורית ועובר לרשימה החדשה אם הוא גדול מהאיבר הבא. האיבר האחרון נמחק מהרשימה המקורית ועובר לרשימה החדשה אם הוא גדול מהאיבר הקודם.
- אם ברשימה המקורית יש רק איבר אחד הוא נמחק ועובר לרשימה החדשה.
- הרשימה המקורית יכולה להיות ריקה.

א. יש לכתוב, לממש ולהשתמש בפונקציה deleteFirst אשר מקבלת מצביע למצביע של ראש הרשימה המקושרת ומוחקת אותו. אם הרשימה ריקה, הפונקציה צריכה להחזיר 0 אחרת 1. **(אין חובה להשתמש בערך המוחזר בפונקציה splitList)**

ב. יש לכתוב, לממש ולהשתמש בפונקציה deleteAfter אשר מקבלת מצביע curr לאיבר כלשהו ברשימה, ומוחקת את האיבר הבא שאחריו. אם curr שווה ל-NULL, או curr מצביע לאיבר האחרון, הפונקציה מחזירה 0, אחרת 1. **(אין חובה להשתמש בערך המוחזר בפונקציה splitList)**

ג. יש לממש את הפונקציות printList ו- freeList אשר מקבלות מצביע לראש הרשימה המקושרות ומדפיסות / מבצעת שחרור זיכרון (בהתאמה).
לאחר שחרור רשימה מקושרת, המצביע לראש הרשימה צריך להיות מאותחל ל-NULL.

ד. ניתן להשתמש בפונקציה ליצירת איבר createElement לפי הצורך **(לא חובה)**

יש להתייחס לחתימות הפונקציות כפי שהן כתובות בשלד, ולהנחיות שמופיעות בהן.
ניתן להניח שיש בזיכרון מספיק מקום להקצאה. יש להתייחס לכל מקרי הקצה האפשריים (רשימה ריקה וכו'). **אין להוסיף פונקציות נוספות מעבר לפונקציות הקיימות.**

שאלה 4:

```
typedef struct list
{
        int data;
        struct list *next;
} list;
```

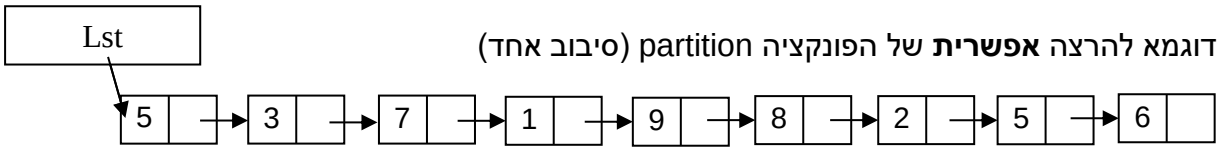
כתבו פונקציה בשם quickSortList המקבלת רשימה מקושרת **חד כיוונית** (ללא איבר דמה) של מספרים שלמים וממיינת אותה בדיוק באותו אופן בה פועלת הפונקציה quickSort על מערך רגיל.

שימו לב: quickSortList מכילה בדיוק את אותם האלמנטים של quickSort הסטנדרטית, כלומר זאת פונקציה רקורסיבית המשתמשת בפונקציית עזר partition אשר מפצלת את קבוצת האיברים ל-2 חלקים, כאשר איבר ה-Pivot ממוקם באמצע.

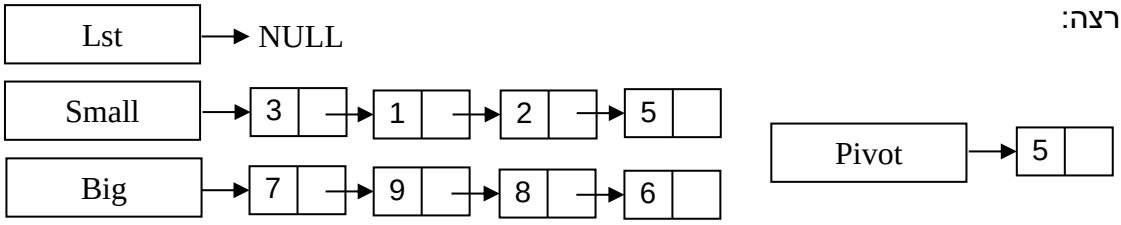
נגדיר את הפונקציה partition באופן הבא:
הפונקציה מקבלת כתובת של רשימה מקושרת כלשהי, מגדירה את האיבר שנמצא בראש הרשימה להיות Pivot ויוצרת 2 רשימות חדשות בה כל האיברים שקטנים או שווים מ-Pivot נמצאים ברשימה הראשונה (ללא חשיבות לסדר), וכל האיברים שגדולים מ-Pivot נמצאים ברשימה השנייה (ללא חשיבות לסדר). Pivot עצמו לא מחברים לאף אחת מהרשימות ומחזירים את כתובתו בתור איבר בודד. לאחר העברת כל הפרמטרים מתוך Lst לשתי הרשימות, Lst שווה ל-NULL.

על הפונקציה לשנות את הקישורים **ללא פעולות של שחרור והקצאה מחודשת**, יש לנתק את האיברים מהרשימה המקורית ולהעביר אותם לרשימות החדשות ולקשר אותם בהתאמה. **אין חשיבות לסדר האיברים בפעולת partition על הרשימות Small ו-Big**

דוגמא להרצה **אפשרית** של הפונקציה partition (סיבוב אחד)



לאחר הרצה:



לאחר ביצוע ההפרדה של הרשימה ל-2 רשימות מקושרות נפרדות, ממשיכים לבצע את פעולת המיון על כל חלק בנפרד, עד אשר מקבלים 2 רשימות ממוינות ולבסוף מחברים את שלושת החלקים יחדיו (הרשימה עם הערכים הקטנים, ה-Pivot והרשימה עם הערכים הגדולים) כאשר המצביע המקורי (Lst) מצביע על תחילת הרשימה הממוינת.
שימו לב: ייתכן שלאחר פעולת partition אחת מין הרשימות יכולה להיות ריקה.

א. יש לכתוב, את המימוש של partition ו-quickSortList.

ב. יש לממש את הפונקציות printList ו- freeList אשר מקבלות מצביע לראש הרשימה המקושרות ומדפיסות / מבצעת שחרור זיכרון (בהתאמה). לאחר שחרור רשימה מקושרת, המצביע לראש הרשימה צריך להיות מאותחל ל-NULL.

ג. **אסור** להשתמש בפונקציה ליצירת איבר createElement.
שכן, אין הקצאה של איברים חדשים, אלא שינוי המיקום של האיברים הקיימים.

יש להתייחס לחתימות הפונקציות כפי שהן כתובות בשלד, ולהנחיות שמופיעות בהן.
ניתן להניח שיש בזיכרון מספיק מקום להקצאה. יש להתייחס לכל מקרי הקצה האפשריים (רשימה ריקה וכו'). **אין להוסיף פונקציות נוספות מעבר לפונקציות הקיימות.**

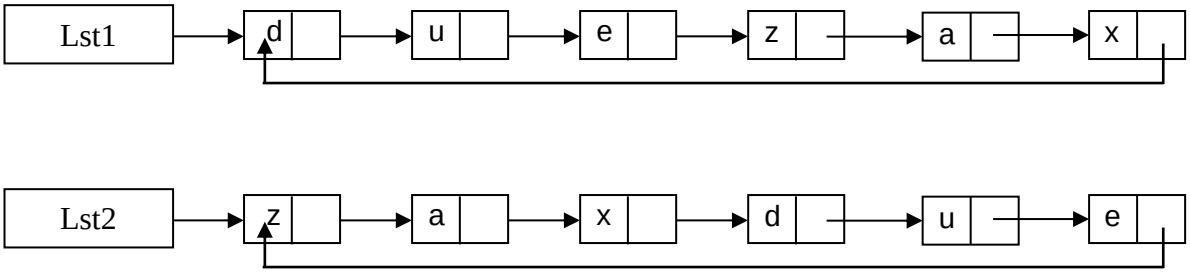
שאלה 5:

כתבו פונקציה בשם compareCircleLists המקבלת שתי כתובות של שני איברים כלשהם ששייכים ל-2 רשימות מקושרת מעגליות שונות (האיבר האחרון מצביע לראשון) **חד כיווניות** (ללא איבר דמה). ה-data של 2 הרשימות הוא מסוג char.
ידוע כי אין שום תו שחוזר על עצמו בכל אחת מין הרשימות.
ניתן להניח שאכן 2 הרשימות הן מעגליות.

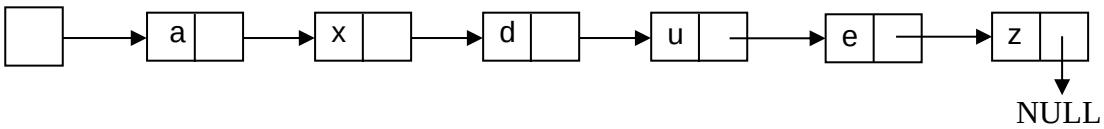
```
typedef struct list
{
        char data;
        struct list *next;
} list;
```

א. על הפונקציה לבצע השוואה בין 2 הרשימות המקושרות ולקבוע האם הן שוות באופן מעגלי.
ב. על הפונקציה לקבוע את המצביע לכל רשימה כך שערך ה-Ascii הקטן ביותר יהיה בראש ולבטל את המעגל.

לדוגמא: שתי הרשימות הבאות שוות באופן מעגלי ולכן הפונקציה צריכה להחזיר 1 (True)



לאחר הפיכת הרשימות לרשימות רגילות (כאשר האיבר האחרון מצביע ל-NULL) שתי הרשימות (כל אחת בנפרד) יעודכנו באופן הבא:



א. יש לכתוב, לממש ולהשתמש בפונקציה circleListLength שמקבלת מצביע לאיבר כלשהו ברשימה מקושרת **מעגלית** ובודקת כמה איברים יש ברשימה.

ב. יש לממש את הפונקציות printList ו- freeList אשר מקבלות מצביע לראש הרשימה המקושרות ומדפיסות / מבצעת שחרור זיכרון (בהתאמה). לאחר שחרור רשימה מקושרת, המצביע לראש הרשימה צריך להיות מאותחל ל-NULL.

ג. **אסור** להשתמש בפונקציה ליצירת איבר createElement.
שכן, אין הקצאה של איברים חדשים, אלא שינוי המיקום של האיברים הקיימים.

יש להתייחס לחתימות הפונקציות כפי שהן כתובות בשלד, ולהנחיות שמופיעות בהן.
ניתן להניח שיש בזיכרון מספיק מקום להקצאה. יש להתייחס לכל מקרי הקצה האפשריים (רשימה ריקה וכו'). **אין להוסיף פונקציות נוספות מעבר לפונקציות הקיימות.**

הוראות:

1. **במידה ושם הפונקציה במסמך אינו תואם את השם שמופיע בקובץ השלד, שם הפונקציה בקובץ השלד היא זאת שקובעת.**
2. **יש להשתמש בשמות משמעותיים וגם בהערות.**
3. **יש להקפיד לכתוב בצורה מבנית.**
4. **אין צורך לבצע בדיקת תקינות קלט וניתן להניח שיש בזיכרון מספיק מקום להקצאה.**
5. **יש להתייחס לכל מקרי הקצה האפשריים (רשימה ריקה וכו')**
6. **אין צורך לבצע קליטת נתונים בתרגיל זה.**
7. **תכנית שלא עוברת קומפילציה לא תתקבל!**

בהצלחה!!!