מכון טכנולוגי חולון
Holon Institute of Technology

COMPUTER SCIENCE DEPARTMENT מחלקה למדעי המחשב

סדנה מתקדמת בתכנות 61108
סמסטר ב' תשפ"ג

מטלה 1

מערכים דינאמיים, מבנים, 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-3 שאלות.
- לכל שאלה ישנו קובץ 'c' מצורף, המכיל את השלד של התרגיל.
  אין לשנות את שמות הקבצים.
  למשל: עבור השאלה הראשונה שם הקובץ הוא ex1_q1.c.
- בכל קובץ קיימים מספר פונקציות, חלקם עליכם לממש.
  אין לשנות את שמות הפונקציות
- את המימושים יש לכתוב החל מהשורה שלאחר ההערה: // your code
- 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
- אין להוסיף פונקציות חדשות
- אין לשנות את חתימת הפונקציות
- אין להוסיף include או define מעבר למה שכבר מופיע
- יש להתייחס באופן מלא להערות נוספות אשר מופיעות בקבצי ההרצה, אין למחוק הערות אלה.

שאלה 1:

כתבו פונקציה יעילה בשם arrangeArray המקבלת **מצביע** (**by reference**) למערך דינאמי המכיל מספרים שלמים וגודלו n. המערך מחולק לשני חלקים רציפים של שתי סדרות **עולות ממש**, כך המספר המינימאלי בקבוצה הראשונה גדול ממש מהמספר המקסימאלי בקבוצה השנייה.

נגדיר את הפרמטר k $(0 < k < n)$ – גודלה של הקבוצה הראשונה.

לדוגמא:

32	64	66	69	72	78	81	87	94	95	1	2	4	8	**16**

המספר **32** שהוא המספר הכי קטן בקבוצה הראשונה, גדול ממש מ-**16** שזה המספר הכי גדול בקבוצה השנייה. ערכו של k הוא 10.

א. על הפונקציה למצוא ולהחזיר את ערכו של k.
ב. על הפונקציה למיין את המערך בצורה יעילה **במינימום קריאות** של הפונקציות **realloc** ו-**memcpy**.

הערה: המערך בוודאות איננו ריק ואין צורך לבדוק זאת, בנוסף ניתן להניח שבכל קבוצה יש לפחות איבר אחד.

למרות שניתן לפתור את התרגיל באמצעות הקצאת מערך עזר נוסף ע"י הפונקציות malloc או calloc, פעולה זו אסורה לביצוע. כמו כן, אין לבצע מיון מערך באמצעות אלגוריתמי מיון קלאסיים כגון: MergeSort, QuickSort, BubbleSort וכו'...

ניתן להניח שיש בזיכרון מספיק מקום להקצאה מחדש.

הערה: הפונקציה memcpy שייכת לספרייה <string.h> והחתימה שלה כלהלן:

```
void *memcpy(void *dest, const void * src, size_t n)
```

פונקציה זו מעתיקה בלוק של זיכרון (מערך) מכתובת אחת לכתובת אחרת.
ניתן להתייחס לטיפוס `size_t` בתור `unsigned`

- **dest** - מצביע למערך היעד אליו יש להעתיק את התוכן.
- **src** - מצביע למערך המקור ממנו יש להעתיק את התוכן.
- **n** - מספר הבתים שיש להעתיק ממערך המקור למערך היעד.

שאלה 2:

כתבו תוכנית אשר קולטת מצולע בעל n קודקודים, ומחשבת את ההיקף שלו.
לביצוע המשימה נגדיר שני טיפוסים חדשים:

```
typedef struct point
{
        int x, y;
}point;
```

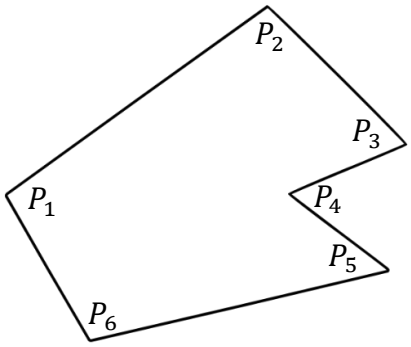
- המבנה point, מכיל את השדות:
שני מספרים שלמים המציינים נקודת ציון (קואורדינטה) במישור.
- המבנה polygon, מכיל את השדות:
מצביע למערך מסוג point, גודל המערך, והיקף המצולע.

```
typedef struct polygon
{
        int n;
        point *points;
        double scope;
}polygon;
```

יש להצהיר בתחילת התוכנית על מבנה מסוג polygon, ולבצע קלט של n נקודות.

אין צורך למיין את הנקודות, יש להניח כי סדר קליטת הנקודות הוא באופן רציף, כך שבין כל 2 נקודות סמוכות קיימת צלע השייכת למצולע.

דוגמא ל-Polygon עם 6 קודקודים:



להלן רשימת הפונקציות הנדרשות לביצוע המשימה:

1. **scanPoint** – מקבלת כתובת של מבנה מסוג point, מבקשת מהמשתמש לקלוט ערכים באמצעות scanf וקולטת בהתאמה.

2. **createPolygon** – מקצה זיכרון עבור polygon חדש ללא נקודות. הפונקציה מאתחלת את הערכים של n, points ו-scope לאפסים ו-NULL. הפונקציה תחזיר כתובת של polygon או NULL עבור כישלון.

3. **distance** – מקבלת כתובות של 2 נקודות ומחשבת את המרחק בין שתיהן.

4. **calculateScope** – מקבלת כתובת של מערך מסוג point וגודלו, מחשבת ומחזירה את ההיקף באמצעות שימוש בפונקציה **distance.**

5. **addPoint** - מקבלת כתובת של מבנה מסוג polygon, מוסיפה נקודה חדשה בסוף מערך הנקודות (יש להגדילו בהתאמה באמצעות **realloc**), מקדמת את ערכו של n, קוראת לפונקציה **scanPoint** ומעדכנת את ערכו של scope באמצעות **calculateScope.** הפונקציה תחזיר 1 עבור הצלחה, 0 עבור כישלון.

6. **removePoint** - מקבלת כתובת של מבנה מסוג polygon ואינדקס (יש לוודא שהאינדקס חוקי). על הפונקציה להסיר את הנקודה באינדקס הנדרש, ולהזיז בהתאמה את שאר איברי המערך. בנוסף הפונקציה תקטין את המערך באמצעות **realloc** תקטין ערכו של n. הפונקציה תעדכן את ערכו של scope באמצעות **calculateScope.** הפונקציה תחזיר 1 עבור הצלחה, 0 עבור כישלון.

7. **freeMemory** - מקבלת כתובת של מבנה מסוג polygon ודואגת לשחרר אותו.

הערה: יש להניח כי היקף פוליגון עם פחות מ-3 קודקודים שווה ל-0.

<u>שאלה 3:</u>
כתבו תוכנית הקולטת רשימה מקושרת של מלבנים, ומחשבת ומחזירה את הקואורדינטות של המלבן הקטן ביותר שמכיל את כל המלבנים ברשימה.

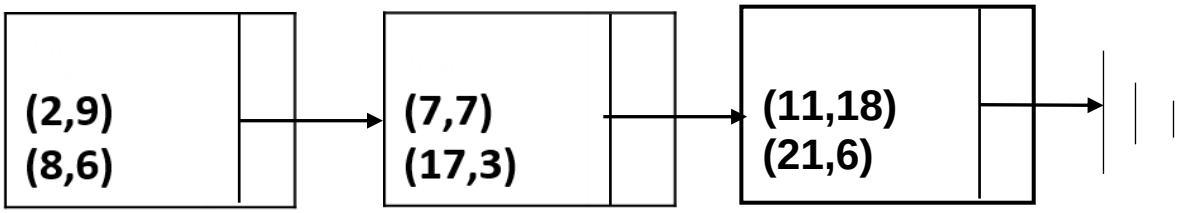
מלבן מיוצג ע"י 2 קודקודים של מלבנים אשר הצלעות שלהם מקבילות לצירים. המלבנים מיוצגים על ידי הקואורדינטות של הקודקוד השמאלי העליון והקודקוד הימני התחתון.

הפונקציה שומרת את נתוני המלבנים כרשומות ברשימה מקושרת בה יש לקלוט קודקודי מלבנים <u>עד שמתקבל מלבן אשר שני הקודקודים שלו באותו מיקום (כלומר ערכי ה-X וגם ערכי ה-Y של שני הקודקודים שווים ואז מתקבלת נקודה במקום מלבן)</u>. המלבן\נקודה המציין את סיום הקלט לא יכנס לרשימה המקושרת של המלבנים.

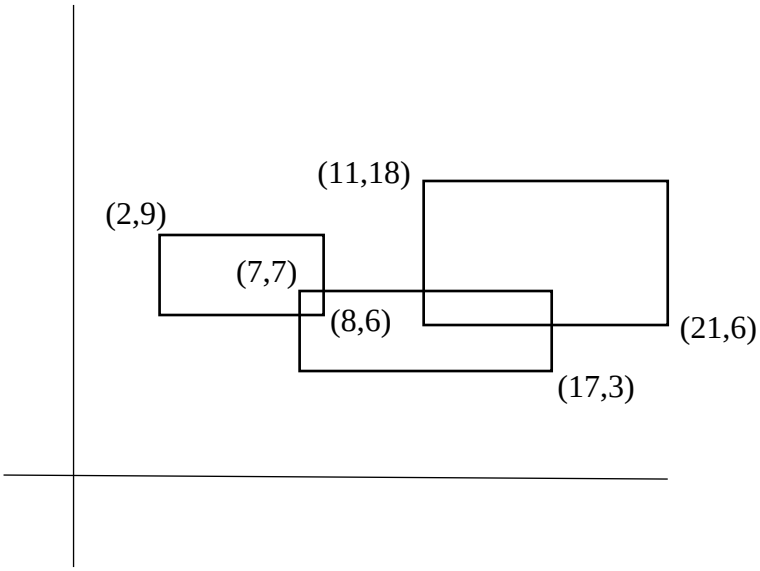
על התוכנית לרוץ על רשימת המלבנים בפונקציה נפרדת, לחשב ולהחזיר את הקואורדינטות של המלבן הקטן ביותר שמכיל את כל המלבנים ברשימה. שימו לב:

- המלבן המוחזר לא בהכרח קיים ברשימה המקושרת.
- ניתן להניח שיש מספיק מקום בזיכרון להקצאת האיברים ברשימה.

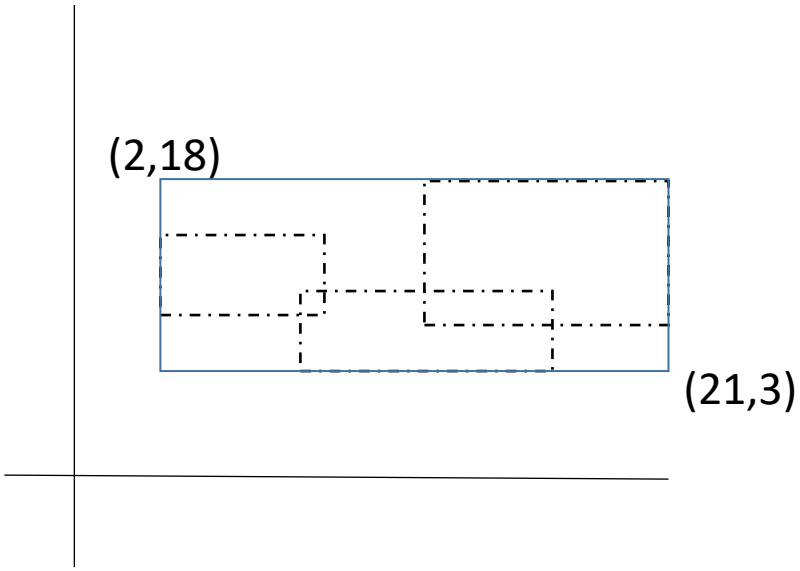
לדוגמא:



שרטוט מלבני הקלט לעייל על מערכת הצירים:



המלבן שהפונקציה צריכה להחזיר לפי הדוגמא לעייל:



שימו לב:

א. יש לקלוט את קודקודי המלבנים בפונקציה נפרדת המיועדת לכך בלבד.

ב. במקרה והקלט לא תקין: כאשר בשני הקודקודים ערכי ה-X או ערכי ה-Y שווים (שאז מתקבל קו ישר ולא מלבן) יש לבקש מהמשתמש להזין קודקודים חדשים למלבן ולתקן את הקלט שלו. יתכן מקרים נוספים של שגיאות בקלט שעליכם לזהות ולבקש מהמשתמש לתקן את הקלט שלו. בכול מקרה של קלט לא תקין יש לבקש מהמשתמש להזין קלט של שני קודקודים מחדש עד אשר מתקבל מלבן (או נקודה , שאז מסתיים קליטת המלבנים מהמשתמש).

ג. אומנם, בדוגמא לעייל כל המלבים נמצאים ברביע אחד(החיובי) ,אולם, המלבנים יכולים להיות בכול אחד מהרביעים של מערכת הצירים.

להלן ההגדרות של המבנים\Struct בהם יש להשתמש בפתרון:

```
typedef struct rectangle {
	int xTopSmall;
	int yTopSmall;
	int xButtomRight;
	int yButtomRight;
}rectangle;

typedef struct recElement {
	rectangle Rect;
	struct recElement* next;
}recElement;
```

להלן רשימת הפונקציות הנדרשות לביצוע המשימה:

1. **validateRectangle** - מקבלת כתובת של מבנה מסוג מלבן, ומבצעת ולידציה. אם המלבן מהווה נקודה, הפונקציה מחזירה 2. (יכול לשמש כתנאי עצירה) אם המלבן תקין (נקלט בהתאמה נקודה שמאלית עליונה, וימנית תחתונה) הפונקציה מחזירה 1. אם המלבן אינו תקין - הפונקציה מחזירה 0.

2. **scanRectangle** - מבצעת קליטה של מלבן לכל ארבעת הפרמטרים שלו, ומבצעת ולידציה באמצעות הפונקציה **validateRectangle.** אם הקליטה איננה תקינה ממשיכה לקלוט עד אשר נקלט מלבן או נקודה. הפונקציה מחזירה את המלבן.

3. **createElement** - מייצרת איבר חדש מסוג recElement ומחזירה את הכתובת שלו. הפונקציה קוראת ל-**scanRectangle** תוך כדי היצירה. אם נקלטה נקודה (ניתן להשתמש שוב בפונקציה **validateRectangle**) הפונקציה מחזירה NULL אחרת את האיבר החדש המכיל את המלבן שנקלט.

4. **createRectList** - מייצרת רשימה מקושרת חד כיוונית מסוג recElement. הפונקציה קוראת ל-**createElement** בלולאה ומפסיקה לייצר איברים כאשר **createElement** מחזירה NULL. הפונקציה מחזירה את הכתובת של האיבר הראשון ברשימה.

5. **findSmallest** - מקבלת את ראש הרשימה המקושרת ומחזירה את המלבן התוחם בהתאם להגדרות השאלה. במידה והרשימה ריקה, הפונקציה תחזיר מלבן מאותחל עם הנקודות: {0, 0, 0, 0}.

6. **printRectangle** - מקבלת כתובת של מלבן ומדפיסה את הערכים שלו על המסך. דוגמא להדפסה: [(2,18)(21,3)].

7. **printList** - מקבלת כתובת של ראש הרשימה המקושרת ומבצעת הדפסה של האיברים. רצוי ומומלץ להשתמש בפונקציה **printRectangle** להדפסת המלבנים.

8. **freeList** - מקבלת כתובת של ראש הרשימה ומבצעת שחרור של כל איברי הרשימה.

<u>הוראות נוספות</u>

1. **על התוכניות להיות יעילות ככל האפשר**
2. **יש להשתמש בשמות משמעותיים וגם בהערות.**
3. **יש להקפיד לכתוב בצורה מבנית.**
4. **בשאלות 1 ו-2 (למעט סעיף 6) אין צורך לבצע בדיקת תקינות קלט.**
5. **בשאלות 1 ו-3 ניתן להניח שיש בזיכרון מספיק מקום להקצאה.**
6. **יש לקלוט את כל הנתונים המקוריים, להדפיס תוצאות כנדרש ולשחרר זיכרון דינאמי.**
7. **קליטת הנתונים מתבצעת אך ורק דרך הפונקציה scanf!**
8. **תכנית שלא עוברת קומפילציה לא תתקבל!**