מבוא למדעי המחשב מ' / ח' (234114 / 234117)

סמסטר אביב תשע"ב - מבחן מועד ב'

מספר סטודנט

- רשום/ה לקורס: **234114** / **234117**
- **תואר ראשון** / **לימודי חוץ** / **אחר (לפרט)** : _____

- משך הבחינה – **3 שעות.**
- בדקו שיש 18 עמודים (4 שאלות) במבחן, כולל עמוד זה.
- השימוש בחומר עזר כלשהו, כתוב או אלקטרוני, **אסור**.
- המבחן כתוב בלשון זכר אך מתיחס לנבחנים ולנבחנות כאחד.
- ניתן להשתמש בפונקציות קלט-פלט סטנדרטיות והקצאת זיכרון ב- C. שימוש בכל פונקציה אחרת, לרבות כזו שהוגדרה במהלך הקורס, אסור. אתם יכולים להגדיר פונקציות עזר כרצונכם. אין צורך להצהיר עליהן.
- כל זיכרון שאתם מקצים, אתם חייבים בשחרורו. אין צורך לבדוק שההקצאה הצליחה.
- ניתן לכתוב בעיפרון ולהשתמש במחק.

שאלה	ערך	ציון	בודק
1	25		
2	25		
3	25		
4	25		
סה"כ	100		

<u>צוות הקורס</u> :
<u>סמסטר אביב תשע"א</u> :
מרצים: ד"ר ניר אילון (מרצה אחראי), דן רביב, תמיר לוי
מתרגלים : חביאר טורק (מתרגל אחראי), אבישי גרץ, תהילה מייזלס, רן זילברשטיין, רועי פורן

בהצלחה !

שאלה 1 (25 נקודות):

נתונה הפונקציה הבאה:

```
int mystery(int m,int n)  {
    int a;
    if(n==0){
        return 1;
    }
    else if(n%2){
        a = mystery(m, (n-1)/2);
        return m * a * a;
    }
    else {
        a = mystery(m, n/2);
        return a * a;
    }
}
```

א. מה תחזיר הפונקציה עבור קלט m=4, n=3? _____

מה מחשבת הפונקציה? _____

ב. מהי סיבוכיות הזמן של הפונקציה? _____

שאלה 2 (25 נקודות) :

ברצוננו לאפשר חישובים במספרים שלמים (אי-שליליים – להקלת הפתרון) שגודלם אינו חסום - **מספרים גדולים**. מספר גדול ייוצג ע"י מחרוזת של התווים '0'...'9'. למשל את המספר השלם 10^{12} ניתן לאתחל כך:

char billion[] = "1000000000000";

שימו לב: **אין** אפסים מובילים, והספרה הגדולה נמצאת בתחילת המחרוזת – **באינדקס 0**.

מטרת השאלה לכפול מספר גדול במספר d בתחום $1 \le d \le 9$.

ממשו את הפונקציה:

int length(char bn[], int d)

המקבלת מספר גדול bn ומספר d המקיים $1 \le d \le 9$ ומחזירה:

1 אם ורק אם בתוצאת הכפל של המספר הגדול bn במספר d יש ספרה אחת יותר מאשר ב bn .

0 אחרת (בתוצאת הכפל של המספר הגדול bn במספר d יש אותו מספר ספרות כמו ב bn) .

למשל אם bn הוא "333333" והערך של d הוא 3 אז הערך המוחזר הוא 0, אבל אם bn הוא "334333" והערך של d הוא 3 אז הערך המוחזר הוא 1.

אסור להקצות זיכרון, ואין להשתמש ברקורסיה. סיבוכיות זמן נדרשת: לינארית באורך bn.

```
int length(char bn[], int d){
```

שאלה 3 (25 נקודות) :

עליכם לכתוב פונקציית חיפוש במערך ממוין **לא-יורד** של מספרים **חיוביים**, ש"משמידה" את האברים במקומות שנמצאו בעבר. במילים אחרות, לפונקציה אסור להחזיר פעמיים את אותו המיקום. מותר לפונקציה לשנות את תוכן מערך החיפוש, אבל הדרישה הבאה חייבת להתקיים:

אם המספר x מופיע k פעמים במערך המקורי (לפני החיפוש הראשון) במקומות $i_1..i_k$, אז k החיפושים הראשונים של x יחזירו $i_1..i_k$ בסדר כלשהו, והקריאה ה- k+1 ואילך מחזירה 1-.

חתימת הפונקציה:

int seek_and_destroy(int* a, int n, int x)

a הוא מערך החיפוש, n הוא אורכו ו- x הוא המספר המבוקש. על הפונקציה להחזיר את המיקום של x אם קיים, ו- 1- אחרת.

דרישות:

סיבוכיות זמן של כל קריאה לפונקציה O(log n), סיבוכיות מקום נוסף של קריאה לפונקציה: O(1)

הצעה: ניתן להשתמש במספרים שליליים על-מנת לסמן ערכים שכבר נמצאו.

הערה: אם כתבתם פתרון בסיבוכיות זמן שהיא לא O(log n), תוכלו לקבל בחזרה חלק מהנקודות אם תכתבו כאן את הסיבוכיות של הפתרון שלכם כתלות בגודל המערך n ומספר החזרות M של האיבר שמופיע הכי הרבה פעמים.

סיבוכיות הזמן: _____

דוגמת שימוש:

```
int a[5] = {1,1,1,5,5};
printf("%d\n", seek_and_destroy(a, 9, 5));
printf("%d\n", seek_and_destroy(a, 9, 1));
printf("%d\n", seek_and_destroy(a, 9, 5));
printf("%d\n", seek_and_destroy(a, 9, 1));
printf("%d\n", seek_and_destroy(a, 9, 1));
printf("%d\n", seek_and_destroy(a, 9, 5));
printf("%d\n", seek_and_destroy(a, 9, 1));
```

פלט חוקי של הדוגמא בעמוד הבא.

פלט חוקי של הדוגמא הנ"ל:

3	*מיקום כלשהו של 5 במערך*
0	*מיקום כלשהו של 1 במערך*
4	*מיקום של 5 השונה מהמיקום שנמצא קודם*
1	*מיקום של 1 השונה מהמיקום שנמצא קודם*
2	*מיקום של 1 השונה משני המיקוממים שנמצאו קודם*
-1	*אין יותר מיקומים חדשים למספר 5*
-1	*אין יותר מיקומים למספר 1*

```
int seek_and_destroy(int* a, int n, int x){
```

שאלה 4 (25 נקודות) :

בידינו גומיה מעגלית. ברצוננו לחבר את הגומיה לאוסף של מסמרים נתונים על לוח עץ. לגומיה יש עשר "נקודות חיבור" (ראו ציור בעמוד הבא). עליכם להתאים לכל אחת מעשר נקודת החיבור מסמר על לוח העץ. התאמה היא חוקית אם כל נקודת חיבור מותאמת למסמר, אף מסמר לא מותאם ליותר מנקודת חיבור אחת, וכן:
(1) יציבות: המרחק בין המסמרים המותאמים לשתי נקודות חיבור צמודות לפחות 10 ס"מ.
(2) עמידה בעומס: המרחק בין המסמרים המותאמים לשתי נקודות חיבור צמודות לכל היותר 20 ס"מ.

אורך התאמה חוקית יוגדר כסכום המרחקים בין כל זוגות נקודות החיבור הצמודות.
מיקום של מסמר נתון במערכת צירים דו-מימדית באמצעות המבנה הבא:

```
struct nail_pos {
   float x;   /* x coordinate position of nail in cm */
   float y;   /* y coordinate position of nail in cm */
};
typedef struct nail_pos NAIL_POS;
```

הניחו שקיימת פונקציית ספרייה: float dist(NAIL_POS* nail1, NAIL_POS* nail2)
המחזירה את המרחק (בס"מ) בין שני מסמרים נתונים.

עליכם להשלים את הפונקציה הבא:
float find_best_fit(NAIL_POS nails[], int n)
הפונקציה מקבלת כקלט את מיקום המסמרים ואת מספרם. על הפונקציה להחזיר את אורך ההתאמה החוקית המינימלית. אם לא קיימת התאמה חוקית, על הפונקציה להחזיר מספר שלילי כלשהו. יש להשתמש ב- backtracking. בשאלה זו אין דרישות סיבוכיות, אולם כמקובל ב-backtracking יש לוודא שלא מתבצעות קריאות רקורסיביות מיותרות עם פתרונות שאינם חוקיים.

הערה: אין להשתמש במשתנים סטטיים

שימו לב:

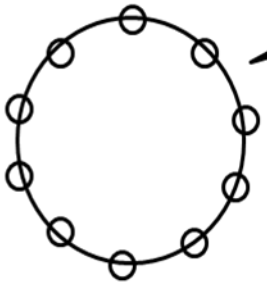
1. מותר לגומי לחתוך את המסלול של עצמו (ראו ציור), אבל, כאמור, אסור להשתמש באותו מסמר עבור שתי נקודות חיבור.

2. אינכם נדרשים להדפיס פתרונות. פתרון שמגלה התאמות חוקיות אבל לא מחזיר את אורך ההתאמה הקצרה ביותר יזכה בניקוד חלקי.

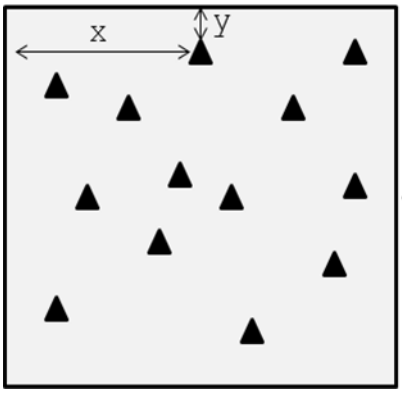
3. ניתן להניח שמספר המסמרים n הוא לפחות 10 ושהמרחק בין כל שני מסמרים גדול מ-0.

דוגמא בעמוד הבא.

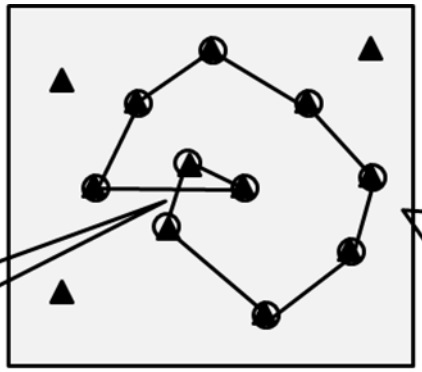
גומי עם 10 נקודות חיבור.



הקלט: מערך של מיקומים של לפחות 10 מסמרים. לכל מסמר (מסומן כמשולש) מיקום על ציר ה- x ועל ציר ה- y.



דוגמא להתאמת נקודות חיבור למסמרים.
ההתאמה חוקית אם כל נקודת חיבור מותאמת למסמר כך שהמרחק בין זוג מסמרים המותאמים לנקודות חיבור צמודות בין 10 ס"מ ל- 20 ס"מ.



מותר לגומי לחתוך את המסלול של עצמו אבל אסור להתאים אותו מסמר לשתי נקודות חיבור.

```
float find_best_fit(NAIL_POS nails[], int n){
```