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

סמסטר חורף תשס"ז

מבחן מסכם מועד א', 15 פברואר 2007

מספר סטודנט	*שם משפחה*	*שם פרטי*

משך המבחן: 3 שעות.
חומר עזר: אין להשתמש בכל חומר עזר בכתב, מודפס או אלקטרוני.

הנחיות והוראות:

- מלאו את הפרטים בראש דף זה.
- בדקו שיש 26 עמודים (4 שאלות) במבחן, כולל עמוד זה.
- כתבו את התשובות על טופס המבחן בלבד, במקומות המיועדים לכך. שימו לב שהמקום המיועד לתשובה אינו מעיד בהכרח על אורך התשובה הנכונה.
- העמודים הזוגיים בבחינה ריקים. ניתן להשתמש בהם כדפי טיוטה וכן לכתיבת תשובותיכם. סמנו טיוטות באופן ברור על מנת שהן לא תבדקנה.
- יש לכתוב באופן ברור, נקי ומסודר. אין לכתוב הערות והסברים לתשובות אם לא נתבקשתם מפורשות לכך.
- בכל השאלות, הינכם רשאים להגדיר ולממש פונקציות עזר כרצונכם. לנוחיותכם, אין חשיבות לסדר מימוש הפונקציות, ובפרט ניתן לממש פונקציה לאחר הפונקציה שמשתמשת בה.
- **אין להשתמש בפונקציות ספריה או בפונקציות שמומשו בכיתה אלא אם צויין אחרת בשאלה.**

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

צוות הקורס

מרצים: סאהר אסמיר, תמיר לוי, דר' מיכאל אלעד (מרצה אחראי).

מתרגלים: איתי שרון, אנדריי קלינגר, אסנת טל, אריה מצליח, ירון יורה, נוגה טל, עידו חניאל, רג'א גיריס,רון בגלייטר, סשה סקולוזוב, גיל בכר, צפריר קמלו, רן רובינשטיין (מתרגל אחראי).

בהצלחה!

שאלה 1 (25 נקודות)

סעיף א

נתונה תוכנית ה-C הבאה, כאשר הפרמטר N הוא קבוע #define שלם וחיובי כלשהו:

```
int i = 2*N;


void strange(int *p)
{
   int j = *p;
   while (j) {
     printf("Backtracking is strange");
     j--;
   }
   *p = *p-1;
}


void charm()
{
   static int k = 8;
   int i;

   while (k) {
      printf("Arrays are charming");
      k = k/2;
   }
   i = 2*k;
}


int main ()
{
  while (i>0) {
    strange(&i);
    charm();
    i--;
  }
  return 0;
}
```

כתבו כמה פעמים תודפס כל אחת מן המחרוזות "Backtracking is strange" ו- "Arrays are charming!" כפונקציה של N (הכוונה למספר מדויק, לא לקירוב אסימפטוטי):

"Backtracking is strange": _____ "Arrays are charming": _____

סעיף ב

בכל אחד מהסעיפים הבאים מופיעות מספר שורות קוד. לכל קטע קוד, הקיפו בעיגול את התיאור המתאים:

א. **ללא שגיאות** – הקוד יתקמפל ללא כל שגיאה וירוץ ללא תקלות.
ב. **שגיאת זמן ריצה** – הקוד יתקמפל ללא שגיאות, אולם הוא עלול לבצע שגיאה בזמן הריצה.
ג. **שגיאת קומפילציה** – הקוד לא יעבור קומפילציה.

1.
```
int  i = 0;
int *j = *i;
```

א. ללא שגיאות
ב. שגיאת זמן ריצה
ג. שגיאת קומפילציה

2.
```
int i[8] = {0,1,2,3,4,5};
*(i + 2) = 8;
```

א. ללא שגיאות
ב. שגיאת זמן ריצה
ג. שגיאת קומפילציה

3.
```
int *j;
int *i = j;
*i = 5;
```

א. ללא שגיאות
ב. שגיאת זמן ריצה
ג. שגיאת קומפילציה

4.
```
int *i, j, k;
i = &(j+k);
```

א. ללא שגיאות
ב. שגיאת זמן ריצה
ג. שגיאת קומפילציה

5.
```
char* s = "Hello World!";
while (*s) {
  *s = *s + 1;
}
```

א. ללא שגיאות
ב. שגיאת זמן ריצה
ג. שגיאת קומפילציה

שאלה 2 (25 נקודות)

נתון מערך order באורך קבוע של 26 (שימו לב להערה בהמשך). מערך זה מכיל את כל האותיות הקטנות בא"ב האנגלי, כל אות פעם אחת בדיוק. המערך מגדיר סדר חדש על אותיות הא"ב, כאשר על פי סדר זה, אות אחת תחשב לקטנה מאות אחרת אם ורק אם אות זו נמצאת במערך order לפני האות האחרת.

לדוגמה, עבור המערך order הבא מתקיים b<a, כיוון ש- b מופיע לפני a במערך. לעומת זאת, יתר האותיות במקרה זה הן בסדר לכסיקוגרפי רגיל.

order[26] = { 'b','a','c','d','e','f','g','h','i','j','k','l','m','n','o','p','q','r','s','t','u','v','w','x','y','z'} ;

הערה: בשאלה זו, מספר האותיות בא"ב האנגלי (26) יכול להיחשב כקבוע לצורך חישובי סיבוכיות.

סעיף א

השלימו את הפונקציה הבאה, המקבלת כקלט מחרוזת s המכילה אותיות אנגליות קטנות בלבד, את אורכה של המחרוזת n (לא כולל תו ה-null), ומערך order המתאר סדר של אותיות הא"ב. הפונקציה מחזירה 1 אם האותיות ב-s מסודרות על פי הסדר המוגדר במערך order, ו-0 אחרת (עבור מחרוזת ריקה החזירו 1). **יש לממש את הפונקציה בסיבוכיות זמן ומקום טובים ככל הניתן**. פתרון לא אופטימאלי יזכה בניקוד מופחת. כמו כן, השלימו את סיבוכיות האלגוריתם שלכם במקום המתאים למטה, כפונקציה של n.

```
int is_ordered(char *s, int n, char order[26]) {
```

סיבוכיות זמן: Θ(_____) סיבוכיות מקום נוסף: Θ(_____)

<u>סעיף ב</u>

בסעיף זה נממש פונקציה המבצעת חיפוש של אות נתונה במחרוזת ממוינת. השלימו את הפונקציה הבאה, המקבלת מחרוזת s הממוינת על פי סדר המוגדר ב-order, את אורך המחרוזת n (לא כולל תו ה-null) וכן אות לחיפוש x. על הפונקציה להחזיר את האינדקס של **המופע האחרון** של x במחרוזת s, או 1- אם x אינה מופיעה המחרוזת.

לדוגמה, עבור הסדר order בדף הקודם והמחרוזת הבאה (n=9):
char s[] = "bbacdddef"

על הפונקציה להחזיר 2 במקרה של חיפוש האות a, 1 במקרה של חיפוש האות b, 6 במקרה של חיפוש האות d ו-(1-) במקרה של חיפוש האות y.

יש לממש את הפונקציה בסיבוכיות זמן ומקום טובים ככל הניתן. פתרון לא אופטימאלי יזכה בניקוד מופחת. כתבו את סיבוכיות האלגוריתם שלכם במקום המתאים:

סיבוכיות זמן: Θ(_____)	סיבוכיות מקום נוסף: Θ(_____)

```
int search(char *s, int n, char order[26], char x) {
```

שאלה 3 (25 נקודות)

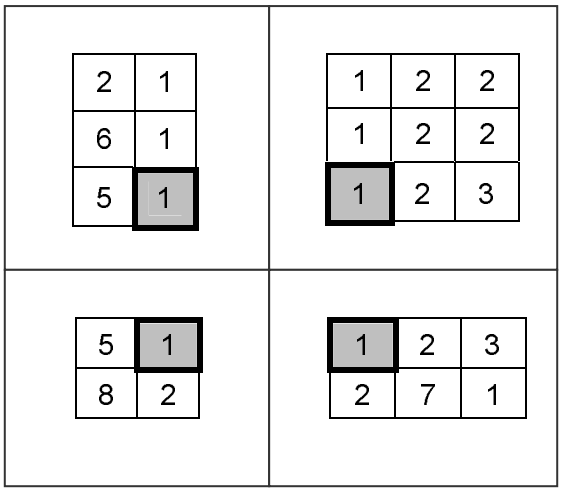
נתונה מטריצה a בגודל N על N, המכילה מספרים שלמים. לכל איבר a[i][j] במטריצה זו, נגדיר את ארבע תת המטריצות הבאות שאיבר זה מהווה פינה שלהם:

תת-מטריצה שמאלית עליונה: האיברים a[0..i][0..j]
ימנית עליונה: האיברים a[0..i][j..N-1]
שמאלית תחתונה: האיברים a[i..N-1][0..j]
ימנית תחתונה: האיברים a[i..N-1][j..N-1]

לדוגמה, עבור המטריצה הבאה (N=4),

2	1	2	2
6	1	2	2
5	1	2	3
8	2	7	1

האיבר a[2][1] (המסומן באפור) מגדיר את ארבע תת-המטריצות:



אנו נאמר כי איבר a[i][j] במטריצה **מאוזן** אם סכום האיברים בכל אחת מארבע תת המטריצות שהוא מגדיר **זהה**. למשל, בדוגמה למעלה a[2][1] מאוזן, כיוון שסכום כל אחת מארבע תת המטריצות שהוא מגדיר שווה ל-16: שימו לב שעבור תת-המטריצה השמאלית העליונה מתקיים 2+6+5+1+1+1=16, עבור תת המטריצה השמאלית התחתונה מתקיים 5+1+8+2=16, וכך הלאה.

עליכם לממש (בדף הבא) פונקציה שמקבלת מטריצה המיוצגת כמערך דו-ממדי `a[N][N]` (N מוגדר כ-#define). הפונקציה מחזירה 1 אם קיים במטריצה איבר מאוזן, ו-0 אחרת.

על הפונקציה לעבוד בסיבוכיות זמן $O(N^2)$ וסיבוכיות מקום $O(N^2)$. שימו לב: פתרונות בסיבוכיות גרועה מזו יזכו בניקוד מופחת, בהתאם לסיבוכיות הזמן של הפתרון.

```
int equal_submatrix_sum(int a[N][N]) {
```

שאלה 4 (25 נקודות)

בשאלה זו נתכנת פונקציה שמנסה לסדר אבני דומינו על לוח. הלוח מתואר באמצעות מערך דו מימדי a שגודלו N על N (כש-N קבוע #define). חלק ממשבצות הלוח ריקות, וחלקן מלאות. הפונקציה תקבל את המערך הדו מימדי a כאשר בכל משבצת ריקה ישנו 0, ובכל משבצת מלאה יש 1-. עליכם לממש פונקציה שבודקת אם ניתן לרצף את משבצות הלוח הריקות בעזרת אבני דומינו בגודל 1על 2, ובמידה וכן, מחזירה את הפתרון שמצאה.

אבני הדומינו יצוינו על ידי מספרים טבעיים: ...1,2,3 (ניתן להניח שמספר אבני הדומינו אינו מוגבל). בתום ריצת הפונקציה, אם אכן ניתן לרצף את הלוח אזי התא `a[i][j]` יכיל את מספרה של אבן הדומינו שמוקמה בתא זה (אם לא ניתן לרצף אז אין חשיבות לתוכן a בסוף הריצה). כמו כן, הפונקציה תחזיר 1 במידה ומצאה פתרון, ו-0 אם אין פתרון. שימו לב שניתן למקם אבן דומינו על הלוח במאוזן או במאונך – ראו דוגמאות להלן.

הערה: אם יש יותר מאפשרות אחת לריצוף הלוח, ניתן להחזיר פתרון אחד כלשהו.

<u>דוגמא ראשונה:</u>

N=3

```
a[3][3]={  0,    0,    0
           0,    -1,   0,
           0,    0,    0 }
```

אחרי ריצת הפונקציה פתרון אפשרי הוא:

```
a[3][3]={  1,    1,    2
           3,    -1,   2,
           3,    4,    4 }
```

<u>דוגמא שנייה:</u>

N=3

```
a[3][3]={  0,    -1,   0
           0,    0,    0,
           0,    0,    0 }
```

לוח כזה אי אפשר לרצף.

לנוחותכם, ניתן להשתמש בפונקציות העזר הבאות:

פונקציה שמחזירה 1 אם התא (i, j) בתחום הלוח:

```
int on_board(int i, int j)
{
      if (i>=0 && i<N && j>=0 && j<N)
            return 1;
      return 0;
}
```

פונקציה שמקבלת זוג אינדקסים (i, j) של ריבוע בלוח, ומעדכנת אותם כך שיכילו את האינדקסים של הריבוע הבא בלוח. הפונקציה מתקדמת לריבוע הבא באותה השורה במידה ויש כזה, או לריבוע הראשון בשורה הבאה במידה והגענו לקצה השורה. הפונקציה מחזירה 0 אם הגענו לקצה הלוח:

```
int next_square(int* i, int* j)
{
      (*j)++;
      if (*j == N)
      {
            *j = 0;
            (*i)++;
            if (*i == N)
                  return 0;
      }
      return 1;
}
```

פונקציה שמחזירה 1 אם התא (i, j) הינו ריבוע פנוי בלוח (0 אחרת):

```
int square_is_free(int i, int j, int a[N][N])
{
      if (on_board(i,j) && a[i][j]==0)
            return 1;
      return 0;
}
```

פונקציה שמחזירה 1 אם הלוח מלא (0 אחרת):

```
int board_is_full(int a[N][N])
{
      int i=0, j=0;
      do {
            if (square_is_free(i,j,a))
                  return 0;
      } while (next_square(&i, &j);

      return 1;
}
```

פונקציה שמחזירה 1 אם ניתן להניח לוחית דומינו בתא ה-(i, j) בלוח, ו-0 אחרת. עבור הפרמטר horizontal==1 ייבדקו זוג הריבועים האופקיים (i, j), (i, j+1), ואילו עבור הפרמטר horizontal==0 ייבדקו זוג הריבועים האנכיים (i, j) (i+1, j):

```
int can_put_tile(int i, int j, int a[N][N], int horizontal)
{
      if (!square_is_free(i,j,a))
            return 0;
      if (horizontal && square_is_free(i,j+1,a))
            return 1;
      if (!horizontal && square_is_free(i+1,j,a))
            return 1;
      return 0;
}
```

נתונה פונקצית המעטפת הבאה, המקבלת את הלוח ההתחלתי כפרמטר, וכותבת לתוכו את הפתרון (אם קיים כזה). הפונקציה מחזירה 1 אם אמנם נמצא פתרון, ו-0 אחרת. שימו לב שפונקציה זו עוטפת קריאה לפונקציה **tile_board_aux**. עליכם להשלים את הפרמטרים המתאימים בקריאה לפונקציה **tile_board_aux**, ולממש פונקציה זו בדף הבא.

שימו לב: ניתן לקבוע את זהות ומספר הפרמטרים של הפונקציה **tile_board_aux** כרצונכם.

```
int tile_board(int a[N][N])
{
    return tile_board_aux(a,____________________________________);
}
```

```
int tile_board_aux( int a[N][N], ________________________
        ____________________________________________________ )
{
```