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

סמסטר אביב תשס"ז

מבחן מסכם מועד א', 12 ספטמבר 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 27 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 תיבדקנה.
- 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.
- **אין להשתמש בפונקציות ספריה או בפונקציות שמומשו בכיתה אלא אם צוין אחרת בשאלה.**
- **פתרון שלא עומד בדרישות הסיבוכיות יקבל ניקוד חלקי בלבד.**

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

צוות הקורסים 234114/7
מרצים: סאהר אסמיר, ד"ר אלי בן ששון (מרצה אחראי).
מתרגלים: מירי בן-חן, אלעד הרמתי, עידו חניאל, אסנת טל, סשה סקולוזוב, אנדרי קלינגר, איתי שרון (מתרגל אחראי).

בהצלחה!

שאלה 1 (25 נקודות)

סעיף א

נתונה תוכנית ה-C הבאה:

```
#define N 5

void mish(char** arr1, char** arr2, int n)
{
    char *p;
    if((n<=0) || (arr1 == arr2)) return;
    p=*arr1;
    *arr1=*arr2;
    *arr2 = p;
    mish(arr1+1, arr2-1, n-1);
}

void mash(char* arr[][N], int n)
{
    if(n>N/2) return;

    mish(&arr[n][0], &arr[N-n-1][N-1], N);
    mash(arr, n+1);
}

int main()
{
    int i, j;
    char* arr[][N] = {{"It", "is", "the", "end", "of"},
      {"the", "world", "as", "we"}, {"know", "it", "and"},
      {"I", "feel"}, {"fine"}};

    mash(arr, 0);
    for(i=0; i<N; i++)
        for(j=0; j<N; j++)
            printf("%s%c", arr[i][j]? arr[i][j] : "",
                           (j==N-1)? '\n' : '\t');

    return 0;
}
```

1. מה הם הפרמטרים אותם מקבלת הפונקציה `mish()` ומה מבצעת הפונקציה?
2. מה תדפיס התכנית?

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 arr[]={1,2,3,4,0};
while(*arr)
  printf("%d ", *(arr++));
```

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

2.

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

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

3.

```
int i=1, *j=&i+1;
(double(i/2))? *j=2 : *(j-1)=2;
```

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

4.

```
void f(int x) {
  --x;
}
void h(int n) {
  if(!n) return;
  h(f(n));
}
int main() {
  h(3);
  return(0);
}
```

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

5.

```
char s[5]={'H','e','l','l','o'},*p=s;
do {
  putchar(++(*p));
} while (*(++p));
```

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

שאלה 2 (25 נקודות)

סעיף א

נתונות שתי מחרוזות num1, num2 (המחרוזות מסתיימות ב-'\0') המייצגות שני מספרים חיוביים הקסדצימליים (בבסיס 16). עליך לממש פונקצית השוואה ביניהן שמקבלת כקלט את num1 ואת num2 ומחזירה 1 אם num1 > num2, 1- אם num1 < num2 ו-0 במקרה של שוויון.

אין להניח שאורך המחרוזות חסום, כלומר המספרים ההקסדצימליים יכולים להיות גדולים מהמספרים הגדולים ביותר הניתנים לייצוג ב- int ועל-כן פתרון המתרגם את המחרוזות למספר לא יתקבל. אין צורך לבדוק את תקינות הקלט (כלומר ניתן להניח שהמחרוזות אכן מייצגות מספר הקסדצימלי). כמו כן ניתן להניח שהאותיות האנגליות במחרוזות גדולות, וכן שהמספרים אינם מכילים אפסים לפני המספר עצמו (למשל: 00FFF)

לדוגמא :

FFF < 123A < 123B < 11D11

ניתן להשתמש בפונקציות הספרייה הבאות:

- `int isalpha(int ch)` – מחזירה ערך שונה מ-0 אם `ch` הוא אות (קטנה או גדולה), או 0 אחרת.
- `int isdigit(int ch)` – מחזירה ערך שונה מ-0 אם `ch` הוא ספרה בין 0 ל-9, או 0 אחרת.
- `size_t strlen(char* str)` – מחזירה את אורך המחרוזת `str`.

```
int hex_cmp(char* num1, char* num2)
{
```

סעיף ב

יש לממש פונקציה המקבלת מערך ממוין של מחרוזות המייצגות מספרים הקסדצימליים (מהקטן לגדול), את גודלו n, ואת המחרוזת num. על הפונקציה להחזיר את מספר המופעים של num במערך (אם המחרוזת לא נמצאת במערך יש להחזיר 0).
ניתן (ואף מומלץ) לממש פונקציות עזר וכן להשתמש בפונקציות מסעיפים קודמים.
יש לממש את הפונקציה בסיבוכיות הזמן הטובה ביותר האפשרית, ובסיבוכיות המקום הטובה ביותר עבור סיבוכיות זמן זו. מהן סיבוכיות המקום והזמן כפונקציה של n ושל K בהנחה שאורך המחרוזת הארוכה ביותר אינו גדול מ-K?

```
int count_appearances(char* nums[], int n, char* num)
{
```

שאלה 3 (25 נקודות)

בעיית מספרי הנוי דומה מאוד לבעיית מגדלי הנוי: ישנם 3 מוטות ו-n טבעות המסודרות בהתחלה על אחד המוטות, מהגדולה (בתחתית) לקטנה (בפסגה). ההבדלים:

- **הטבעות ממוספרות**: מספר הטבעת הגדולה ביותר n ומספר הטבעת הקטנה ביותר 1
- **המגדלים ממוספרים**: לכל אחד משלושת המגדלים יש מספר (0, 1 או 2)
- **המצב הסופי** הדרוש הוא שכל טבעת תהיה ממוקמת על המוט שמספרו הוא כשארית חלוקת מספר הטבעת ב-3. כלומר הטבעות 3,6,9... על מוט מספר 0, טבעות 1,4,7... על מוט מספר 1 וטבעות 2,5,8... על מוט מספר 2.

חוקי המשחק נשמרים:

1. מותר להזיז רק טבעת בודדת בכל צעד
2. אסור למקם טבעת מעל לטבעת קטנה ממנה

לנוחותכם, מובא להלן הקוד של בעיית הנוי המקורית, כפי שנלמד בכיתה:

```
typedef enum{A,B,C} tower_t;

void hanoi(int n, tower_t from, tower_t to)
{
  tower_t via = (A + B + C) - from - to;
  if (n == 0) return;
  hanoi(n-1, from, via);
  printf("Move disc %d from %c to %c\n",
                  n, 'A' + from, 'A' + to);
  hanoi(n-1, via, to);
}
```

סעיף א'

כתבו פונקציה **רקורסיבית** הפותרת את בעיית מספרי האנוי עם **הקלה**: חוק מספר 1 לעיל מתבטל, כלומר **ניתן** להזיז מספר טבעות עליונות בבת אחת (תוך שמירה על הסדר היחסי שלהן) ממגדל למגדל. הזזה של מספר טבעות ניתן לעשות אך ורק ע"י קריאה לפונקציה `move_several` אשר חתימתה:

```
void move_several(int n, tower_t from, tower_t to)
```

על הפונקציה שלכם לגרום להדפסת סדרת ההזזות הדרושה. שימו לב שהפונצקיה `move_several` מדפיסה הודעה המתאימה להזזות שהיא מבצעת, ולכן רק במקרה בו אתם מזיזים טבעת שלא ע"י קריאה ל-`move_several` עליכם להדפיס הודעה מתאימה.

```
void hanoi2(int n, int loc)
{
```

סעיף ב'

כתבו פונקציה רקורסיבית הפותרת את בעיית מספרי הנוי תוך קיום כל הדרישות, בפרט דרישה 1 לעיל.
רמז: חשבו במה ניתן להחליף את הקריאה לפונקציה `move_several`.

```
void hanoi3(int n, int loc)
{
```

שאלה 4 (25 נקודות)

נוסחת CNF נקראת *דלילה* אם כל משתנה מופיע בעשר פסוקיות שונות לכל היותר. מטרת שאלה זו היא לכתוב גרסא יעילה יותר של פונקצית DPLL המיועדת לפתור את בעיית הספיקות (אך ורק) על נוסחאות דלילות. לאורך השאלה, מספר המשתנים יסומן ב-N ומספר הפסוקיות ב-M (שני המספרים הם מספרים שלמים ומוגדרים ע"י `#define`). הגדרתה של נוסחת CNF מופיעה בתחתית העמוד, כתזכורת.

לצורך שמירת נוסחת ה-CNF והצבות חלקיות לנוסחה נשתמש בטיפוס `truth` שהוגדר בכיתה כך:

```
typedef enum {FALSE, UNSET, TRUE} truth;
```

נוסחת ה-CNF נשמרת במערך דו-מימדי `truth CNF[m][N]` וכל שורה בו מייצגת פסוקית כך:

- `CNF[i][j]=TRUE` אם הליטרל x_j שייך לאילוץ ה-i
- `CNF[i][j]=FALSE` אם הליטרל x_j! שייך לאילוץ ה-i
- `CNF[i][j]=UNSET` כאשר גם x_j וגם x_j! אינם מופיעים כלל באילוץ ה-i

הצבה חלקית נשמרת במערך `truth assignment[N]`.

הקוד של אלגוריתם DPLL שהוצג בכיתה מובא לנוחיותכם.

תזכורת: נוסחת CNF היא מהצורה C_0 AND C_1 AND ... AND C_{M-1} כאשר:

- כל C_i נקראת **פסוקית**, והינה מהצורה: $C_i \equiv \left(L_j \text{ OR } L_k \text{ OR } \ldots \text{ OR } L_t\right)$
- כל L_j נקרא **ליטרל**, והינו משתנה בולאני (x_j) או ההופכי שלו ($!x_j$).

דוגמא לנוסחת CNF:

$$\left(x_1 \text{ OR } x_4\right) \text{ AND } \left(!x_2 \text{ OR } x_0 \text{ OR } !x_3\right) \text{ AND } \left(!x_4\right) \text{ AND } \left(!x_1 \text{ OR } x_3 \text{ OR } x_0 \text{ OR } !x_4\right)$$

הצבה היא קביעת ערך (TRUE או FALSE) למשתנים הבוליאניים. הצבה חלקית משמעותה קביעת ערך לחלק מן המשתנים. הצבה חלקית **מספקת** את הנוסחה אם ורק אם בכל פסוקית, יש לפחות ליטרל אחד אשר מקבל ערך TRUE.

לדוגמא: ההצבה החלקית $x_0=x_1=$TRUE, $x_2=x_3=$UNSET, $x_4=$FALSE מספקת את הנוסחה שלעיל, ולעומת זאת ההצבה החלקית $x_0=x_1=x_2=x_3=$UNSET, $x_4=$TRUE אינה מספקת הנוסחה, שכן הפסוקית השלישית מקבלת ערך FALSE תחת הצבה זו.

```
truth DPLL(truth CNF[][N], truth assignment[], int m, int i)
// i denotes the smallest unassigned variable. To use the
// function call DPLL(CNF, assignment, m, 0).
{
  truth current_truth;
  current_truth = CNF_sat(CNF,assignment,m);
  if (current_truth==TRUE)
    return TRUE;
  if (current_truth==FALSE)
    return FALSE;
  assignment[i]=FALSE;
  if (DPLL(CNF, assignment, m, i+1)==TRUE)
    return TRUE;
  assignment[i]=TRUE;
  if (DPLL(CNF, assignment, m, i+1)==TRUE)
    return TRUE;
  assignment[i]=UNSET;
  return FALSE;
}

truth CNF_sat (truth CNF[][N], truth assignment[], int m)
{
  truth t=TRUE;
  int i;
  truth clause_truth;
  for (i=0;i<m;i++){
    clause_truth=clause_sat(CNF[i], assignment, N);
    if (clause_truth==FALSE)
      return FALSE;
    if (clause_truth==UNSET)
      t=UNSET;
  }
  return t;
}
truth clause_sat (truth clause[], truth assignment[], int n)
{
  int j;
  truth t=FALSE;
  for (j=0;j<n;j++) {
    if (clause[j]!=0) {
      if (assignment[j]==clause[j])
        return TRUE;
      if (assignment[j]==UNSET)
        t=UNSET;
    }
  }
  return t;
}
```

סעיף א'

לצורך שיפור זמן הריצה על נוסחאות דלילות, נעשה שימוש במערך `int appears[N][10]` אשר יציין לכל משתנה את מספריהן של הפסוקיות בהן הוא מופיע. כתבו קוד בעל סיבוכיות זמן ריצה קטנה ככל האפשר לפונקציה הבאה המאתחלת את המערך `appears`:

```
truth appears_init(truth CNF[][N], int m, int appears[][10])
```

לאחר היציאה מהפונקציה על השורה ה-i של המערך `appears` להכיל את מספריהן של עשר הפסוקיות בהן מופיע המשתנה ה-i. אם משתנה זה מופיע בפחות מעשר פסוקיות, על התאים הנותרים להכיל את הערך 1-. על הפונקציה להחזיר ערך `TRUE` אם הנוסחה היא דלילה. אחרת, עליה להחזיר ערך `FALSE`.

```
truth appears_init(truth CNF[][N], int m, int appears[][10])
{
```

סעיף ב'

עדכנו את הקוד של פונקצית `CNF_sat` שהוצגה בכיתה כך שהפונקציה תבדוק **רק** את הספיקות של הפסוקיות המכילות את המשתנה האחרון שהוצב. סיבוכיות זמן הריצה צריכה להיות O(N).
הפונקציה תחזיר ערך FALSE **אם ורק אם** אחת הפסוקיות הללו מקבלת ערך FALSE. בכל מקרה אחר הפונקציה תחזיר ערך UNSET. הפונקציה תקבל כקלטים נוספים את מספר המשתנה האחרון לו הוצב ערך (`last`) ואת המערך `appears` מסעיף א'.
ניתן להניח כי המערך מאותחל כמתואר בסעיף א', גם אם לא פתרתם את סעיף א'. מותר לעשות שימוש בקוד DPLL המוצג לעיל.

```
truth CNF_sat2(truth CNF[][N], truth assignment[], int m,
               int appears[][10], int last)
{
```

סעיף ג'

עדכנו את הקוד של פונקצית DPLL כך שסיבוכיות זמן הריצה על נוסחאות דלילות תהיה $O(2^N*N)$.
לצורך כך יש להשתמש בפונקציה המתוארת בסעיף ב', ומותר להשתמש בה אפילו אם לא פתרתם את סעיף ב'. שימו לב לתנאי העצירה של הרקורסיה ולנכונות הפלט של הפונקציה.

```
truth DPLL2(truth CNF[][N], truth assignment[], int m,
            int i, int appears[][10])
{
```