מבוא למדעי המחשב
מועד ב', סמסטר א' תשס"ג, 17/2/03

מרצה: שולי וינטנר.
מתרגל: שלמה יונה

משך המבחן: שעתיים וחצי.
חומר עזר: מותר כל חומר עזר, מלבד מחשב.
הנחיות:

1. ודאו כי בטופס שבידיכם 8 עמודים. יש לכתוב את התשובות על גבי טופס המבחן ולהגיש את כל הטופס ואת הטופס בלבד.
2. קראו היטב כל שאלה. ודאו כי אתם מבינים את השאלה לפני שתתחילו לענות עליה.
3. כתבו בכתב יד ברור וקריא. השתמשו בדפי הטיוטה והעתיקו לטופס המבחן רק תשובות סופיות. תשובות לא קריאות לא תיבדקנה.
4. הערות לתשובותיכם ניתן לכתוב בעברית, גם בגוף פונקציות C.
5. אם לא נכתב אחרת, כאשר עליכם להגדיר פונקציה יש להגדיר פונקציה אחת בדיוק. לא ניתן להשתמש בפונקציות חיצוניות.
6. אם לא נכתב אחרת, בתוכניות ניתן להשתמש בפונקציות מתוך הספריות הבאות בלבד:
   a. stdio.h
   b. stlib.h
   c. string.h
   d. ctype.h

בהצלחה!

שאלה	ציון
1	/25
2	/25
3	/25
4	/25
סה"כ	/100

שאלה 1- 25 נקודות:

בשאלה זו ניתן להשתמש בכל הפונקציות שהודגמו בהרצאה, ללא צורך להגדיר אותן. אם הנכם משתמשים בפונקציה חיצונית כזו, הצהירו עליה, הסבירו בהערה מה היא מבצעת וקבעו את סיבוכיותה.

נתונים שני מערכים באורך זהה המוגדרים כך: `int array[N]`, `int barray[N]`. הגדירו פונקציה המקבלת שני מערכים כאלו ומספר שלם d, ומחזירה 1 אם קיים אבר x ב-`array` ואבר y ב-`barray` כך ש: `x-y=d`. על הפונקציה להחזיר 0 אחרת. לדוגמה, אם המערכים הם:

`array:`	12	2	14	2
`barray:`	11	14	2	8

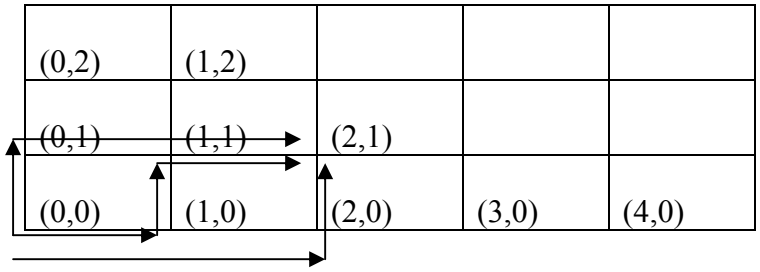
הפונקציה תחזיר 1 אם d הוא 1 או 3, ותחזיר 0 אם d הוא 5 או 2.

על הפונקציה לעבוד בזמן **O(N logN)**. פתרונות בסיבוכיות גבוהה יותר לא יתקבלו.

```
int diff (int array[ ], int barray[ ], int d)
```

שאלה 2- 25 נקודות:

נתון סריג (מערכת צירים) בגודל MxN. מותר לנוע על הסריג ימינה ולמעלה בלבד. הגדירו פונקציה המקבלת כקלט קואורדינטות של נקודה על הסריג ומחזירה את מספר האפשרויות השונות לנוע מנקודת הראשית אל הנקודה הנתונה. לדוגמה, בסריג הבא מתוארות שלוש הדרכים השונות לנוע מהראשית אל הנקודה (2,1) ולכן על הפונקציה להחזיר 3 עבור הקלט (2,1):




```
int paths (int i, int j)
```

נתונה הפונקציה הרקורסיבית הבאה:

```
int recur (int a[], int n)
{
  if (n==0) {
    return 0;
  } else if (n==1) {
    return a[0];
  } else {
    return (a[0]+a[n-1]+recur(a+1,n-2));
  }
}
```

הגדירו פונקציה לא רקורסיבית שקולה בשם iter. על הפונקציה להחזיר לכל קלט בדיוק את אותו הפלט ש-recur מחזירה. ב-iter אין לקרוא לאף פונקציה.

```
int iter (int a[], int n)
```

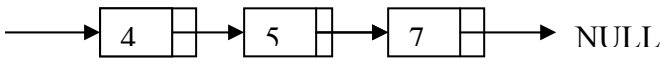
שאלה 3- 25 נקודות:

נתונה ההגדרה הבאה עבור רשימה מקושרת:

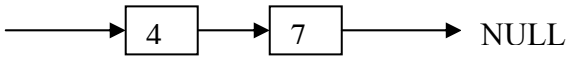
```
typedef struct node {
     int data;
     struct node *next;
} Node;
```

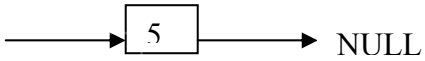
ידוע כי בכל רשימה לפחות איבר אחד, וכי האיבר האחרון בכל רשימה מצביע ל-NULL. הגדירו פונקציה בשם `free_list` המקבלת רשימה כזו שכל איבר בה הוקצה באופן דינאמי ומשחררת את הזיכרון המוקצה.

הגדירו פונקציה בשם `merge` המקבלת שתי רשימות כאלו, שידוע כי הן ממוינות, ומחזירה (דרך רשימת הארגומנטים שלה) רשימה הכוללת את כל אברי שתי רשימות הקלט. על רשימת הפלט להיות ממוינת. ניתן לפגוע ברשימות הקלט.
לדוגמה, הרשימה הבאה:



תתקבל כפלט משתי הרשימות הבאות:





כתבו כאן את שורת הקריאה לפונקציה, אם ידוע שהמשתנים `list1, list2` מצביעים אל תחילת רשימות הקלט ו-`list3` מצביע אל רשימת הפלט:

הגדרת הפונקציה:

שאלה 4- 25 נקודות:

בטבלה שבעמוד הבא, כתבו מה תדפיס התוכנית הבאה (בטבלה יותר שורות מהנדרש):

```
#include <stdio.h>
#include <stdlib.h>
#define ROW 3
#define COL 4

int foo=10;

int* have(int *bar) {
        int foo=5;
        *bar+=foo;
        foo=*bar-1;
        printf("%d, %d\n",foo,*bar);
        return bar;
}

int fun(int *bar) {
        *bar=7;
        foo+=4;
        printf("%d, %d\n",foo,*bar);
        return foo-1;
}

int what(int a[][COL]) {
        int total=0,i,j;
        for(i=0;i<ROW;++i)
                for(j=0;j<COL;++j)
                        total+=(i%2?1:-1)*a[i][j];
        return total;
}

int main() {
        int a[ROW][COL]={{0,2,5,1},{4,5,8,9},{3,2,4,2}},bar=10,*baz;
        printf("Result: %d\n", what(a));
        baz=(int*)malloc(sizeof(int));
        printf("%d, %d\n",fun(&bar),bar);
        foo=fun(&bar);
        printf("%d, %d\n",foo,bar);
        *baz=*(have(&foo));
        *baz*=2;
        printf("%d, %d\n",foo,*baz);
        free(baz);
        return 0;
}
```


ליד כל אחת מההצהרות הבאות, כתבו **אמת** אם היא נכונה ו**שקר** אם אינה נכונה:

1. ניתן לאחסן במשתנה מטיפוס מצביע כתובת של משתנה מטיפוס מצביע למספר שלם.
2. הפקודה `if (x-=2) printf("yes");` תגרום להדפסה אם ורק אם ערכו של x לפני ביצועה הוא 3.
3. ניתן לממש כל מבנה `if-else` באמצעות שתי לולאות `for` .
4. בכל תוכנית בה מופיע הבלוק: `{i=6;j=7;}` ניתן להמירו בבלוק: `{j=7;i=6;}` בלי לפגוע בנכונות התוכנית (`i` ו-`j` הם משתנים מטיפוס `int`).
5. אם x הוא מטיפוס Node (שהוגדר בשאלה 3) אזי הביטוי `x->next->next` הוא תקין תחבירית וערכו הוא מטיפוס Node .

מבוא למדעי המחשב

מועד ב', סמסטר א' תשס"ג,

17/2/03

Problem 1

```
void sort (int array[], int size); /* mergesort, O(n log n) */
int binsearch (int n, int v[], int low, int high); /* O(|v|) */

boolean diff(int array[], int barray[], int d)
{
  int i;
  sort(barray,N);

  for(i=0;i<N;++i) {
    if (binsearch(array[i]-d,barray,0,N) >= 0) {
      return 1;
    }
  }
  return 0;
}
```

Complexity: sort an array with N elements using mergesort costs O(N log N). Then we searched N times. Each search costs O(log N), so all N searches cost O(N log N). So we have in total 4 times O(N log N), which is O(N log N).

Problem 2

```
int paths (int i, int j)
{
  if (i==0 || j==0) return 1;
  else return (paths(i-1,j)+paths(i,j-1));
}
int iter (int a[], int n)
{
  int i,j,result=0;

  for (i=0,j=n-1; i<j; i++,j--) {
    result += a[i]+a[j];
  }
  if (i==j) {
    result += a[i];
  }
  return result;
}
```

Or, alternatively:

```
int iter (int a[], int n)
{
  int i,result=0;

  for (i=0; i<n; i++) {
    result += a[i];
  }
  return result;
}
```

Problem 3

```
void free_list(Node* list) {
        Node* p;
```

```
        while(list!=NULL) {
                p=list;
                list=list->next;
                free(p);
        }
}
merge(list1,list2,&list3);
void merge (Node* list1, Node* list2, Node** list3) {
        Node* p1, *p2, *p, *list;

        if (list1==NULL) {
                *list3=list2;
                return;
        }
        if (list2==NULL) {
                *list3=list1;
                return;
        }

        list=(list1->info) < (list2->info)?list1:list2;
        p1=list1->next;
        p2=list2;
        p=list;

       while( (p1!=NULL) && (p2!=NULL) ) {
                if ( (p1->info) < (p2->info) ) {
                        p->next=p1;
                        p1=p1->next;
                        p=p->next;
                        p->next=NULL;
                } else {
                        p->next=p2;
                        p2=p2->next;
                        p=p->next;
                        p->next=NULL;
                }
        }
        while (p1!=NULL) {
                p->next=p1;
                p1=p1->next;
                p=p->next;
                p->next=NULL;
        }
        while (p2!=NULL) {
                p->next=p2;
                p2=p2->next;
                p=p->next;
                p->next=NULL;
        }
        *list3=list;
        return;
}
```

Problem 4

```
Result: 7
14, 7
13, 10
18, 7
17, 7
21, 22
22, 44
```

Easiest way to check this out is to type the code into a file, compile it and run it.

1. True
2. False
3. True
4. True
5. False