מבוא למדעי המחשב
מועד א', סמסטר א' תשס"ג, 13/1/03

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. stdlib.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]`. (N הוא קבוע המוגדר ב-`#define`). הגדירו פונקציה המקבלת שני מערכים כאלו, ומדפיסה (בסדר כלשהו) את כל האברים הנמצאים באחד המערכים אך לא במערך האחר. לדוגמה, עבור N=4, אם המערכים הם:

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

הפונקציה תדפיס את המספרים הבאים (בסדר כלשהו): 12,11,8. אם המערכים זהים לחלוטין, הפונקציה לא תדפיס דבר. אם המערכים זרים לגמרי באברים, הפונקציה תדפיס את כל אברי שני המערכים.

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

```
int symdiff (int array[ ], int barray[ ])
```

שאלה 2- 25 נקודות:

שביל ניתן לריצוף במרצפות משלושה צבעים: מרצפות אדומות באורך 2, שחורות באורך 2 ומרצפות אפורות באורך 1. הגדירו פונקציה רקורסיבית המקבלת מספר שלם n ומחזירה את מספר האפשרויות השונות לריצוף שביל באורך n .

```
int paths (int n)
```

נתונה הפונקציה הרקורסיבית הבאה:

```
int recur (char *a)
{
  if (!*a) {
    return 1;
  } else {
    return (2*recur(++a));
  }
}
```

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 (char *a)
```

שאלה 3- 25 נקודות:

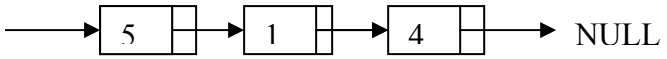
נתונה רשימה מקושרת אשר כל צומת בה מוגדר כך:

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

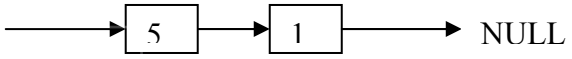
הגדירו פונקציה בשם new_node המקבלת מספר שלם ומחזירה מצביע לצומת מטיפוס Node. על הפונקציה להקצות זיכרון עבור הצומת ולאתחל את שדה info של הצומת למספר שהתקבל בקלט.

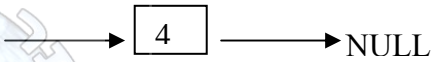
כתבו פונקציה בשם split המקבלת רשימה שבה לפחות איבר אחד, וידוע כי האיבר האחרון בה מצביע ל-NULL. על הפונקציה להחזיר (דרך רשימת הארגומנטים שלה) שתי רשימות: האחת מכילה את האברים ברשימת הקלט ששדה info שלהם זוגי, והאחרת את האברים ששדה info שלהם אי-זוגי. בכל אחת מרשימות הפלט סדר האברים צריך להיות הסדר המקורי שלהם בקלט. כל אחת מרשימות הפלט צריכה להסתיים ב-NULL. אין לפגוע ברשימת הקלט.

לדוגמה: אם רשימת הקלט היא הרשימה הבאה:



תחזיר הפונקציה את שתי הרשימות הבאות:





כתבו כאן את שורת הקריאה לפונקציה, אם ידוע שהמשתנה list מצביע אל תחילת הרשימה:

קריאה:

הגדרת הפונקציה:

שאלה 4- 25 נקודות:

ליד כל פקודת הדפסה בתוכנית שלהלן, כתבו מה יודפס:

```
#include <stdio.h>
#define LA 5
#define LB 10

int bb;

int foo(int *a) {
     int bb;

     bb=*a+5;
     (*a)++;
     printf("%d, %d\n",*a,bb);
     return bb;
}

int* bar(int *a, int c) {
     bb=*a-2;
     c++;
     printf("%d, %d, %d\n", *a, bb, c);
     return &bb;
}

int main() {
     int a[]={7,2,5,1,3},i,aa=5,c=0,*bb;
     char b[]={'z','a','r','t','q','e','j','g','o','u'};
     bb=&aa;
     *bb=3;
     for(i=0;i<LA;++i)
          printf("%c",b[a[i]]);
     printf("\n");
     printf("%d, %d, %d\n", aa, *bb, c);
     c=foo(&aa);
     printf("%d, %d, %d\n", aa, *bb, c);
     bb=bar(bb,c);
     printf("%d, %d, %d\n", aa, *bb, c);
     return !*bb;
}
```

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

1. אם בכותרת של פונקציה אחד הפרמטרים הפורמליים מוגדר: `int *x` אזי `x` יכול להיות מצביע למספר שלם או מצביע למערך של מספרים שלמים.
2. אם `a` ו-`b` הם מערכים מאותו טיפוס ובאותו אורך אזי הביטוי `a=b` גורם להעתקת תוכן המערך `b` אל המערך `a`.
3. ניתן לממש כל מבנה `if-else` באמצעות לולאות `do-while`.
4. אם `g` היא פונקציה המקבלת שני מספרים שלמים ומחזירה את סכומם, ו-`f` היא פונקציה כלשהי המקבלת ומחזירה מספר שלם, אזי `g(f(1),f(2))` שקול תמיד ל- `g(f(2),f(1))`. (שני ביטויים הם שקולים אם ניתן להחליף ביניהם בכל תוכנית מבלי לשנות את התנהגות התוכנית.)
5. אם `x` הוא מטיפוס מצביע ל-`Node` (שהוגדר בשאלה 3) אזי הביטוי `x->next->next` הוא תקין תחבירית וערכו הוא מטיפוס מצביע ל-`Node` .

Problem 1

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

void symdiff(int array[], int barray[])
{
  int i;

  sort(array,N);
  sort(barray,N);

  for (i=0; i<N; ++i) {
    if (binsearch(array[i],barray,0,N)<0)
      printf("%d ", array[i]);
  }

  for (i=0; i<N; ++i) {
    if (binsearch(barray[i],array,0,N)<0)
      printf("%d ", barray[i]);
  }
  putchar('\n');
}
```

Complexity: sort an array with N elements using mergesort costs O(N log N), and we did this twice (complexity remains). 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).

Notice that the function here is declared as void while in the test it was declared as returning an int. So anyone that returned some int (any int) should be OK.

Problem 2

```
int paths (int n)
{
  if (n<=1) return 1;
  else if (n==2) return 3;
  else return (paths(n-1)+2*paths(n-2));
}
```

Different configurations of tiles with different colors make different solutions. Of course using tiles with the SAME shape and color in different order does NOT make a different solution because all solutions look the SAME.

```
int iter (char *a)
{
  int result=1;

  for (; *a != '\0'; ++a) {
    result *=2;
```

```
  }
  return result;
}
```

Problem 3

```
Node* new_node(int info) {
	Node* node;
	if (NULL==(node=(Node*)malloc(sizeof(Node)))) {
		printf("Memory allocation error\n");
		exit(EXIT_FAILURE);
	}
	node->info=info;
		node->next=NULL;
	return node;
}
split_lists(list,&list_odd,&list_even);
void split_lists(Node* list, Node** list_odd, Node** list_even) {
	Node *p_odd, *p_even, *p;
	p=list;
	*list_even=NULL;
	*list_odd=NULL;
	while(NULL!=p) {
		if ( (p->info)%2 ) {
			p_odd=*list_odd;
			*list_odd=new_node(p->info);
			(*list_odd)->next=p_odd;
		} else {
			p_even=*list_even;
			*list_even=new_node(p->info);
			(*list_even)->next=p_even;
		}
		p=p->next;
	}

}
```

Problem 4

great
3, 3, 0
4, 8
4, 4, 8
4, 2, 9
4, 2, 8

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

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