# מבוא לתכנות מערכות
# תרגיל בית מספר 1 (c)
## סמסטר אביב 2015-2016

תאריך פרסום: 25.03.16
תאריך הגשה: 08.04.16, 23:55
משקל התרגיל: 4% מהציון הסופי (תקף)
מתרגל אחראי: דני רסין (234122cs@gmail.com)

## 1 הערות כלליות

- **שימו לב: לא יינתנו דחיות במועד התרגיל. תכננו את הזמן בהתאם.**
- לשאלות בנוגע לתרגיל יש לפנות לסדנאות של אחד מהמתרגלים, או לפנות במייל למתרגל האחראי על התרגיל. נא לכתוב בשורת הנושא (subject): mtm1. לפני שליחת שאלה - נא וודאו שהיא לא נענתה כבר ב-F.A.Q ושהתשובה אינה ברורה ממסמך זה, מהדוגמא או מהבדיקות שפורסמו עם התרגיל.
- **קראו מסמך זה עד סופו** ועיברו על הדוגמא שפורסמה **לפני** תחילת הפתרון.
- **חובה** להתעדכן בעמוד ה- F.A.Q של התרגיל
- **העתקות קוד בין סטודנטים יטופלו בחומרה!**

## 2 חלק יבש

### *2.1 שאלה 1*

נתון הקוד הבא שהתנהגותו הרצויה מוגדרת באופן הבא:
הפונקציות `swapIfFirstNegative` ו- `swapIfSecondNegative` מקבלות מערך קלט ומחזירות מערך פלט בשם resultArray.
המערך החדש צריך להיות מוקצה דינמית ושווה בגודלו למערך המקורי שנשלח.
המערך מורכב ממבנים מטיפוס struct temp (אשר מוגדר גם כן מטה).
בפונקציה `swapIfFirstNegative`, במידה והשדה a של המבנה הוא שלילי – יש לבצע החלפה בין הערכים a ו- b (כלומר, במקום המתאים במערך הפלט – יופיע מבנה בו הערכים של a ו- b הוחלפו. מערך הקלט לא ישתנה).
הפונקציה `swapIfSecibdNegative` היא זהה, למעט העובדה שההחלפה נעשית רק במידה ו- b שלילי.
לבסוף, המשתנה num הוא משתנה פלט אשר סופר כמה החלפות התבצעו.

ציינו את כל השגיאות המופיעות בקוד. יש להתייחס גם לשגיאות נכונות, וגם לחריגות מכללי תכנות נכון אשר נלמדו בקורס.
תקנו את השגיאות שציינתם והגישו גרסה מתוקנת של הפונקציות הנ"ל.

```
struct temp {
	int a;
	int b;
};

bool swap(struct temp s) {
	if (s.a >= 0 && s.b >= 0) {
		return false;
	}
	int temp = s.a;
	s.a = s.b;
```

```
	s.b = temp;

	return true;
}

void swapIfFirstNegative(struct temp* array, struct temp* resultArray, int* num) {

	resultArray = malloc(sizeof(*array));
	num = malloc(sizeof(int));

	for (struct temp* current = array; current != NULL; current++) {
		*resultArray = *current;
		if (resultArray->a < 0) {
			assert(swap(*resultArray));
			(*num)++;
		}
	}
}

void swapIfSecondNegative(struct temp* array, struct temp* resultArray, int* num) {

	resultArray = malloc(sizeof(*array));
	num = malloc(sizeof(int));

	for (struct temp* current = array; current != NULL; current++) {
		*resultArray = *current;
		if (resultArray->b < 0) {
			assert(swap(*resultArray));
			(*num)++;
		}
	}
}
```

### 2.2 שאלה 2

נתון המימוש הבא לצומת של רשימה מקושרת חד-כיוונית:

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

רשימה מקושרת מעגלית היא רשימה שבה אף אחד מאיברי ה- next אינו NULL. ערך השדה next של האיבר האחרון ברשימה, הוא כתובתו של האיבר הראשון. כתבו את הפונקציה print_reverse המקבלת רשימה מקושרת מעגלית ומדפיסה את איבריה בסדר הפוך, החל מהאיבר הראשון המתקבל.
כלומר, אם התקבלה הרשימה שאיבריה הם:

1 -> 5 -> 17 -> 81 -> 1

יש להדפיס:

1
81
17
5

ניתן להניח ע"י assert כי הרשימה המתקבלת אינה NULL.
- יש לכתוב פתרון רקורסיבי

# 3 חלק רטוב

## *3.1 רקע*

התרגיל עוסק במימוש טיפוס נתונים של דירה ולוח דירות. כדי לבצע מטלה זו נתחיל בכתיבת טיפוס נתונים עבור דירה, שמורכבת מאוסף משבצות, אשר כל אחד מהם מייצג קיר בדירה או משבצת פנויה. לאחר מכן נגדיר את טיפוס הנתונים עבור שירות לפרסום וחיפוש דירות **כך שישתמש בדירות שהגדרנו קודם לכן.**
כמו כן, עליכם לבדוק את נכונות הקוד שלכם ולשם כך יהיה עליכם לספק בדיקות אוטומטיות עבור טיפוסי הנתונים הנ"ל.

## *3.2 טיפוס הנתונים Apartment*

טיפוס הנתונים דירה מורכב מלוח משבצות המתאר את מבנה הדירה, וכן ממחיר הדירה. המחיר הינו מספר שלם אי שלילי כלשהו (מחיר 0 הוא מחיר חוקי). כמו כן, הדירה שומרת את המידות של לוח המשבצות הנתון.

### 3.2.1 מבנה הנתונים והפעולות

בקובץ aprtment.h מוגדר מבנה הנתונים של דירה:

```
typedef enum { EMPTY, WALL } SquareType;

/*
 * The apartment structure.
*/
struct apartment_t {
	SquareType** squares;
	int width;
	int length;
	int price;
}
```

השדה price הוא מספר שלם אי שלילי המייצג את מחיר הדירה.
השדות width ו- length הם מספרים חיוביים (ממש) המייצגים את ממדי הדירה. אנו נניח כי כל הדירות בהן אנו עוסקים הן מבלניות.
השדה squares מייצג את לוח המשבצות המתאר את מבנה הדירה.
כמו כן, בקובץ apartment.h מופיעות חתימות עבור פונקציות המבצעות פעולות על דירה. עליכם ליצור את הקובץ apartment.c ולממש פעולות אלו.
עבור כל הפעולות – ניתן להניח שבמידה והדירה המתקבלת היא מצביע מאותחל (לא NULL) – אז הדירה היא דירה תקנית שהוחזרה ע"י פעולת ה- create שאתם מימשתם (מוגדרות מטה). שימו לב שבמקרה של שגיאות – אסור לכם להפסיק את ריצת התכנית (מקרים אלו מפורטים בהמשך).

- עבור כל השגרות המפורטות מטה ומחזירות ערך מטיפוס `ApartmenmtResult`:
  - במידה והפרמטר עבור הדירה (apartment) הוא NULL, או שמועבר מצביע לקבלת פלט שערכו NULL יש להחזיר `APARTMENT_NULL_ARG`. שגיאה זו קודמת לכל שגיאה אחרת המופיעה בהמשך.
  - במידה והקלט לשגרה תקין והמימוש דורש הקצאת זיכרון דינמית שנכשלה – יש להחזיר את השגיאה `APARTMENT_OUT_OF_MEM`.
  - במידה והפעולה הסתיימה בהצלחה, יש להחזיר `APARTMENT_SUCCESS`.
- כל האינדקסים המוזכרים בתרגיל הם zero-based (כלומר, האיבר הראשון הוא באינדקס 0).

- בפונקציות המחזירות ערך מספרי, במידה ונכשלה הקצאת זיכרון – יש להחזיר (-1).

**<u>יצירת דירה חדשה:</u>**

**חתימת הפונקציה:**

```
Apartment apartmentCreate(SquareType** squares, int length, int width, int price);
```

**תיאור הפעולה:** שגרה זו יוצרת משתנה חדש מטיפוס דירה. יש ליצור עותק חדש של לוח המשבצות הנתון squares. השגרה מחזירה את הדירה החדשה שנוצרה.

**פרמטרים:** המשתנה price הוא מחיר הדירה.

המשתנה squares הוא מערך דו מימדי שמחזיק את מבנה הדירה, וממדיו הם width ו- length. הגישה ל- squares[i][j] היא חוקית כאשר i הוא בין 0 ל- (length-1) (כולל) ו- j הוא בין 0 ל- (width-1) (כולל).

**שגיאות:** במקרה של קלט לא חוקי (squares הוא NULL, אחד המימדים אינו חיובי או שהמחיר שלילי) או במקרה של שגיאה בהקצאת הזיכרון יש להחזיר NULL.

**<u>האם באותו החדר:</u>**

**חתימת הפונקציה:**

```
ApartmenmtResult apartmentIsSameRoom(Apartment apartment, int row1, int col1,
                                     int row2, int col2, bool* outResult);
```

**תיאור הפעולה:** הפעולה מקבלת דירה ושתי משבצות המייצגות מיקום בדירה, כאשר כל משבצת מיוצגת ע"י זוג אינדקסים row ו- col. המיקום הוא חוקי כאשר השורות (row1, row2) הן בין 0 ל- (length-1) והעמודות (col1, col2) הן בין 0 ל- (width-1). השגרה בודקת האם שתי המשבצות נמצאות באותו חדר. שתי משבצות נחשבות באותו חדר אם קיים ביניהן מסלול, אשר מורכב מצעדים ב- 4 הכיוונים הראשיים בלבד (קדימה, אחורה, ימינה, שמאלה) שעובר רק דרך משבצות ריקות (כלומר, לא דרך קיר).

**פרמטרים:** המשתנה apartment יחזיק את הדירה המתקבלת.

המשתנים row1, col1 מייצגים את המשבצת הראשונה והמשתנים row2, col2 מייצגים את המשבצת השנייה.

המשתנה outResult הוא משתנה **פלט** אשר בו תוחזר התשובה.

**שגיאות נוספות:** במידה ואחת הקואורדינטות אינה חוקית, יש להחזיר APARTMENT_OUT_OF_BOUNDS.

במידה ואחת הקואורדינטות היא קואורדינטה של קיר (ולא משבצת פנויה), יש להחזיר APARTMENT_NO_ROOM.

**<u>שטח הדירה:</u>**

**חתימת הפונקציה:**

```
int apartmentTotalArea(Apartment apartment);
```

**תיאור הפעולה:** הפעולה מקבלת דירה ומחזירה את מספר המשבצות הריקות בדירה.

**פרמטרים:** המשתנה apartment יחזיק את הדירה המתקבלת.

**<u>שטח החדר:</u>**

**חתימת הפונקציה:**

```
ApartmenmtResult apartmentRoomArea(Apartment apartment, int row, int col,
                                   int* outArea);
```

**תיאור הפעולה:** הפעולה מקבלת דירה ומשבצת המייצגת מיקום בדירה. השגרה סופרת כמה משבצות בדירה נמצאות באותו החדר יחד עם המשבצת הנתונה. שתי משבצות נחשבות באותו חדר אם קיים ביניהן מסלול, אשר מורכב מצעדים ב- 4 הכיוונים הראשיים בלבד (קדימה, אחורה, ימינה, שמאלה) שעובר רק דרך משבצות ריקות (כלומר, לא דרך קיר).

**פרמטרים:** המשתנה apartment יחזיק את הדירה המתקבלת.

המשתנים row, col מייצגים את המשבצת הנתונה.

המשתנה outArea הוא משתנה **פלט** אשר בו תוחזר התשובה.

**שגיאות נוספות:** במידה ואחת הקואורדינטות אינה חוקית, יש להחזיר APARTMENT_OUT_OF_BOUNDS.

במידה ואחת הקואורדינטות היא קואורדינטה של קיר (ולא משבצת פנויה), יש להחזיר APARTMENT_NO_ROOM.

**<u>פיצול דירה:</u>**

**חתימת הפונקציה:**

```
ApartmenmtResult apartmentSplit(Apartment apartment, bool splitByRow,
                                int index, Apartment* first,
                                Apartment* second);
```

**תיאור הפעולה:** הפעולה מקבלת דירה ומייצרת 2 דירות חדשות (מוקצות דינמית) אשר מתקבלות מפיצול הדירה המקורית. הפיצול מתבצע על-פי שורה או עמודה, בהתאם לדגל `splitByRow`.

אם ערכו אמת – הפיצול על פי שורה ואחרת הפיצול לפי עמודה, כאשר השורה/עמודה הרלוונטית נבחרת לפי המשתנה index.

במידה ומתבצע פיצול לפי שורה, הדירה הראשונה תורכב מכל השורות בעלות מספר נמוך יותר מהשורה המפוצלת, והדירה השנייה תורכב מכל השורות בעלות מספר גבוה יותר מהשורה המפוצלת. השורה המפוצלת לא תיכנס לאף אחת מהדירות.

הפיצול הוא חוקי רק בהנחה שהשורה (או העמודה) שעל-פיה מתבצע הפיצול מורכבת אך ורק ממשבצות קיר.

כמו כן, ניתן לבצע את הפיצול רק אם 2 הדירות המתקבלות נותרות עם מימדים חיוביים ממש.

המחיר של הדירה החדשה נקבע באופן יחסי לגדולה וביחס למחיר הישן באופן הבא:

New_Price = (Old price) x (# rows in new apartment + 1) / (# rows in original apartment)

החלוקה היא חלוקה בשלמים.

במידה והפיצול נעשה על-פי עמודות, החישוב הוא סימטרי.

שימו לב כי סכום שני המחירים החדשים גבוה מהמחיר המקורי של הדירה לפני הפיצול.

על הדירה המקורית להישאר ללא שינוי לאחר הפעולה.

**פרמטרים:** המשתנה apartment יחזיק את הדירה המתקבלת.

המשתנה spliyByRow בוחר האם לפצל לפי שורה או לפי עמודה.

המשתנה index בוחר את השורה (או העמודה) הרלוונטיות.

המשתנה `first` הוא משתנה **פלט** עבור השורות (או העמודות) עם האינדקסים הנמוכים יותר, ואילו המשתנה `second` הוא משתנה **פלט** עבור השורות (או העמודות) עם האינדקסים הגבוהים יותר.

**שגיאות נוספות:** במידה ולא קיימת בדירה שורה (או עמודה, בהתאם לאופי הפיצול) מספר index, יש להחזיר `APARTMENT_OUT_OF_BOUNDS`.

במידה ואסור לבצע את הפיצול (לפי המפורט לעיל), יש להחזיר `APARTMENT_BAD_SPLIT`.

<u>דוגמא:</u> נניח כי נתונה הדירה הבאה שמחירה 100:

Original:

| Col / Row | 0 | 1 | 2 |
|---|---|---|---|
| 0 | | | |
| 1 | | | |
| 2 | | | |
| 3 | | | |
| 4 | | | |

לאחר פיצול לפי שורה מס' 2 נקבל 2 דירות חדשות:

First:

| Col / Row | 0 | 1 | 2 |
|---|---|---|---|
| 0 | | | |
| 1 | | | |

Second:

| Col / Row | 0 | 1 | 2 |
|---|---|---|---|
| 0 | | | |

| | | | |
|---|---|---|---|
| 1 | | | |

אשר מחיר כל אחת מהן הוא 60.

**<u>שטח הדירה:</u>**

**חתימת הפונקציה:**

```
int apartmentNumOfRooms(Apartment apartment);
```

**תיאור הפעולה:** הפעולה מקבלת דירה ומחזירה את מספר החדרים השונים בדירה.
**פרמטרים:** המשתנה apartment יחזיק את הדירה המתקבלת.

**<u>קבלת ריבוע:</u>**

**חתימת הפונקציה:**

```
ApartmenmtResult apartmentGetSquare(Apartment apartment, int row, int col,
                                    SquareType* outValue);
```

**תיאור הפעולה:** השגרה מקבלת דירה ומחזירה את ערך הריבוע המופיע בשורה מסוימת ובעמודה מסוימת.
**פרמטרים:** המשתנה apartment יחזיק את הדירה המתקבלת.
המשתנה row יחזיק את מספר השורה הרצויה.
המשתנה col יחזיק את מספר העמודה הרצויה.
outValue – הוא משתנה **פלט** שיקבל את הערך בריבוע המתאים.
**שגיאות נוספות:** במידה ואחת הקואורדינטות אינה חוקית, יש להחזיר APARTMENT_OUT_OF_BOUNDS.

**<u>שינוי ריבוע:</u>**

**חתימת הפונקציה:**

```
ApartmenmtResult apartmentSetSquare(Apartment apartment, int row, int col,
                                    SquareType value);
```

**תיאור הפעולה:** השגרה מקבלת דירה, קואורדינטות של ריבוע וערך חדש, ומציבה את הערך בריבוע המבוקש.
**פרמטרים:** המשתנה apartment יחזיק את הדירה המתקבלת.
המשתנה row יחזיק את מספר השורה הרצויה.
המשתנה col יחזיק את מספר העמודה הרצויה.
value – הוא הערך החדש המבוקש.
**שגיאות נוספות:** במידה ואחת הקואורדינטות אינה חוקית, יש להחזיר APARTMENT_OUT_OF_BOUNDS.
במידה והערך הנוכחי של הריבוע הוא כבר הערך המבוקש, יש להחזיר APARTMENT_OLD_VALUE.

**<u>שינוי מחיר הדירה:</u>**

**חתימת הפונקציה:**

```
ApartmenmtResult apartmentChangePrice(Apartment apartment, int percent);
```

**תיאור הפעולה:** הפעולה מקבלת דירה ואחוז עבור שינוי המחיר ומשנה את מחיר הדירה בהתאם לאחוז זה. במידה והאחוז חיובי, מחיר הדירה עולה. במידה והאחוז שלילי, מחיר הדירה יורד. ערכים חיוביים אינם מוגבלים, אך האחוז השלילי חסום מלמטה ע"י -100 (כולל). במידה והתוצאה אינה שלמה, יש לעגל את הפרש השינוי (חיובי או שלילי) כלפי מטה בערך מוחלט.
לדוגמא, בהינתן דירה שמחירה 35, שינוי של 50% יהפוך את מחיר הדירה ל- 52, ואילו שינוי של -50% יהפוך את מחיר הדירה ל- 18.
**פרמטרים:** המשתנה apartment יחזיק את הדירה המתקבלת.
המשתנה percent מייצגת את אחוז השינוי.
**שגיאות נוספות:** במידה והאחוז קטן מ- -100, יש להחזיר APARTMENT_PRICE_NOT_IN_RANGE.

**<u>מחיר הדירה:</u>**

**חתימת הפונקציה:**

```
int apartmentGetPrice(Apartment apartment);
```

**תיאור הפעולה:** הפעולה מקבלת דירה ומחזירה את מחירה.
**פרמטרים:** המשתנה apartment יחזיק את הדירה המתקבלת.

**<u>אורך הדירה:</u>**
**חתימת הפונקציה:**

```
int apartmentGetLength(Apartment apartment);
```

**תיאור הפעולה:** הפעולה מקבלת דירה ומחזירה את אורכה (מספר השורות המרכיבות אותה).
**פרמטרים:** המשתנה apartment יחזיק את הדירה המתקבלת.

**<u>רוחב הדירה:</u>**
**חתימת הפונקציה:**

```
int apartmentGetWidth(Apartment apartment);
```

**תיאור הפעולה:** הפעולה מקבלת דירה ומחזירה את רוחבה (מספר העמודות המרכיבות אותה).
**פרמטרים:** המשתנה apartment יחזיק את הדירה המתקבלת.;

**<u>השוואת דירות:</u>**
**חתימת הפונקציה:**

```
bool apartmentIsIdentical(Apartment apartment1, Apartment apartment2);
```

**תיאור הפעולה:** השגרה מקבלת 2 דירות ובודקת האם הן זהות. 2 דירות יחשבו זהות אם שתהין NULL, או ששתיהן לא NULL וכל הפרמטרים שלהן זהים. כלומר, לשתי הדירות אותו מחיר, אותו רוחב, אותו גובה, וכל זוג משבצות באינדקסים תואמים הוא זהה.
**פרמטרים:** המשתנים apartment1 ו- apartment2 יחזיקו את הדירות המתקבלות.

**<u>העתקת דירה:</u>**
**חתימת הפונקציה:**

```
Apartment apartmentCopy(Apartment apartment);
```

**תיאור הפעולה:** השגרה מקבלת דירה ומייצרת משתנה חדש מטיפוס דירה, מוקצה דינמית, הזהה לדירה המקורית. יש לבצע העתקה עמוקה של השדות.

**פרמטרים:** המשתנה apartment יחזיק את הדירה המתקבלת.
**שגיאות:** במקרה שהדירה המתקבלת היא NULL או במקרה של שגיאה בהקצאת זיכרון, יש להחזיר NULL.

**<u>שחרור דירה:</u>**
**חתימת הפונקציה:**

```
void apartmentDestroy(Apartment apartment);
```

**תיאור הפעולה:** השגרה מקבלת דירה ומשחררת את כל הזיכרון שהוקצה עבורה.
**פרמטרים:** המשתנה apartment יחזיק את הדירה המתקבלת.

## 3.3 טיפוס הנתונים ApartmentService

לאחר שסיימתם לממש את טיפוס הנתונים Apartment, עליכם לממש טיפוס נתונים עבור שירות לפרסום וחיפוש דירות (או בקיצור "לוח דירות" או "לוח"). השירות כולל ניהול אוסף של דירות, אשר בתרגיל זה יהיה חסום במספר הדירות שהוא יכול להכיל, ע"י משתנה שיועבר באתחול.

### 3.3.1 מבנה הנתונים והפעולות

בקובץ apartment_service.h נתונה הגדרה ריקה של מבנה:

```
typedef struct apartment_service_t* ApartmentService;
struct apartment_service_t {
	// TODO: add fields here.
};
```

עליכם להוסיף שדות להגדרה, כך שתוכלו לממש את הממשק הנדרש.

בקובץ זה גם מוגדרות הפעולות, שאותן עליכם לממש בקובץ apartment_service.c (אותו אתם צריכים לכתוב).

כמו כן, עליכם להשלים את התיעוד של הפונקציות, בדומה למופיע בקובץ apartment.h.

עבור כל הפעולות – ניתן להניח שבמידה והמצביע המוחזר מאותחל (לא NULL) – אז זהו לוח תקני שהוחזר ע"י פעולת ה- create (מוגדרת מטה). שימו לב שבמקרה של שגיאות – אסור לכם להפסיק את ריצת התכנית (מקרים אלו מפורטים בהמשך).

- עבור כל השגרות המפורטות מטה ומחזירות ערך מטיפוס ApartmenmtServiceResult:
  - במידה והפרמטר עבור הלוח (service) הוא NULL, או שמועבר מצביע לקבלת פלט שערכו NULL יש להחזיר APARTMENT_SERVICE_NULL_ARG. שגיאה זו קודמת לכל שגיאה אחרת המופיעה בהמשך.
  - במידה והקלט לשגרה תקין והמימוש דורש הקצאת זיכרון דינמית שנכשלה – יש להחזיר את השגיאה APARTMENT_SERVICE_OUT_OF_MEM.
  - במידה והפעולה הסתיימה בהצלחה, יש להחזיר APARTMENT_SERVICE_SUCCESS.
- בפונקציות המחזירות ערך מספרי, במידה ונכשלה הקצאת זיכרון – יש להחזיר (-1).
- במידה ותיתכן יותר משגיאה אחת – יש להחזיר את השגיאה המוקדמת ביותר שצוינה (כלומר, לפי סדר ההופעה במסמך).

**<u>יצירת לוח חדש:</u>**

**חתימת הפונקציה:**

```
ApartmentService serviceCreate(int maxNumOfApartments);
```

**תיאור הפעולה:** שגרה זו יוצרת לוח דירות חדש. בעת היצירה, אין דירות בלוח. מספר הדירות שהלוח יכול להכיל חסום ע"י הפרמטר maxNumOfApartments.

**פרמטרים:** המשתנה maxNumOfApartments יחזיק את החסם על מספר הדירות.

**שגיאות:** אם לא ניתן להכניס דירות כלל ללוח (maxNumOfApartments אינו חיובי) יש להחזיר NULL. במקרה של שגיאה בהקצאת הזיכרון יש להחזיר NULL גם כן.

**<u>הוספת דירה חדשה:</u>**

**חתימת הפונקציה:**

```
ApartmentServiceResult serviceAddApartment(ApartmentService service,
                                           Apartment apartment, int id);
```

**תיאור הפעולה:** השגרה מקבלת לוח ודירה ומוסיפה עותק <u>חדש</u> של הדירה לסוף האוסף המנוהל ע"י הלוח.

**פרמטרים:** המשתנה service יחזיק את הלוח המתקבל.

המשתנה apartment יחזיק את הדירה המתקבלת.

המשתנה id יחזיק מזהה ייחודי עבור הדירה בלוח.

**שגיאות נוספות:** במקרה ש- apartment הוא NULL, יש להחזיר APARTMENT_SERVICE_NULL_ARG.

במידה והמזהה id שלילי, יש להחזיר APARTMENT_SERVICE_OUT_OF_BOUNDS.

במידה והלוח מלא (מספר הדירות בו הוא maxNumOfApartments שהתקבל ביצירה), יש להחזיר APARTMENT_SERVICE_FULL.

במידה וכבר קיימת דירה בלוח עם המזהה id, יש להחזיר APARTMENT_SERVICE_ALREADY_EXISTS.

**<u>מספר הדירות:</u>**

**חתימת הפונקציה:**

```
int serviceNumberOfApatments(ApartmentService service);
```

**תיאור הפעולה:** השגרה מקבלת לוח ומחזירה את מספר הדירות הנוכחי בלוח.
**פרמטרים:** המשתנה service יחזיק את הלוח המתקבל.

**חציון מחירים:**
**חתימת הפונקציה:**

```
ApartmentServiceResult servicePriceMedian(ApartmentService service,
                                          int* outResult);
```

**תיאור הפעולה:** השגרה מקבלת לוח ומחשבת את המחיר החציוני מבין כל הדירות בלוח.
במידה ומספר הדירות בלוח הוא זוגי – החציון ייחשב לממוצע שני האיברים החציוניים. במידה וממוצע זה אינו שלם – יש לעגלו כלפי מטה.
לדוגמא: החציון של {1, 2, 10, 11} הוא 6, כיוון שזהו הממוצע של 2 ו- 10.
**פרמטרים:** המשתנה service יחזיק את הלוח המתקבל.
outResult– הוא משתנה פלט שבו יוחזר החציון הרצוי.
**שגיאות נוספות:** בידה ואין דירות בלוח, יש להחזיר APARTMENT_SERVICE_EMPTY.

**חציון שטח:**
**חתימת הפונקציה:**

```
ApartmentServiceResult serviceAreaMedian(ApartmentService service,
                                         int* outResult);
```

**תיאור הפעולה:** השגרה מקבלת לוח ומחשבת את השטח החציוני מבין כל הדירות בלוח.
במידה ומספר הדירות בלוח הוא זוגי – החציון ייחשב לממוצע שני האיברים החציוניים. במידה וממוצע זה אינו שלם – יש לעגלו כלפי מטה.
לדוגמא: החציון של {1, 2, 10, 11} הוא 6, כיוון שזהו הממוצע של 2 ו- 10.
**פרמטרים:** המשתנה service יחזיק את הלוח המתקבל.
outResult– הוא משתנה פלט שבו יוחזר החציון הרצוי.
**שגיאות נוספות:** בידה ואין דירות בלוח, יש להחזיר APARTMENT_SERVICE_EMPTY.

**חיפוש דירה:**
**חתימת הפונקציה:**

```
ApartmentServiceResult serviceSearch(ApartmentService service, int area,
                                     int rooms, int price,
                                     Apartment* outApartment);
```

**תיאור הפעולה:** השגרה מקבלת לוח ושלושה חסמים לסינון: שטח, מספר חדרים ומחיר. השגרה מחפשת את הדירה האחרונה בלוח (כלומר, זו שנכנסה ללוח בשלב המאוחר ביותר) שמספקת את שלושת הקריטריונים:

- שטחה הוא לפחות area.
- מספר החדרים בה הוא לפחות rooms.
- מחירה הוא לכל היותר price.

**פרמטרים:** area, rooms ו- price מהווים חסמים עבור החיפוש.
המשתנה outApartment– הוא משתנה פלט שבו יישמר עותק חדש של הדירה הרצויה, במידה וכזו נמצאה.
**שגיאות נוספות:** במידה ואחד החסמים הוא שלילי, יש להחזיר APARTMENT_SERVICE_OUT_OF_BOUNDS.
במידה ואין כלל דירות בלוח, יש להחזיר APARTMENT_SERVICE_EMPTY.
במידה ואף אחת מהדירות בלוח לא עונה על הקריטריונים, יש להחזיר APARTMENT_SERVICE_NO_FIT.

**חיפוש לפי מזהה:**

**חתימת הפונקציה:**

```
ApartmentServiceResult serviceGetById(ApartmentService service, int id,
                                      Apartment* outApartment);
```

**תיאור הפעולה:** השגרה מקבלת לוח ומזהה עבור דירה. השגרה מחפשת דירה בלוח בעלת המזהה הנתון.
**פרמטרים:** id הוא המזהה המבוקש.
המשתנה outApartment– הוא משתנה פלט שבו יישמר עותק חדש של הדירה הרצויה, במידה וכזו נמצאה.
**שגיאות נוספות:** במידה והמזהה המתקבל שלילי, יש להחזיר APARTMENT_SERVICE_OUT_OF_BOUNDS.
במידה ואין כלל דירות בלוח, יש להחזיר APARTMENT_SERVICE_EMPTY.
במידה ואין דירה בלוח עם המזהה המתאים, יש להחזיר APARTMENT_SERVICE_NO_FIT.

**מחיקת דירה:**
**חתימת הפונקציה:**

```
ApartmentServiceResult serviceDeleteApartment(ApartmentService service,
                                              Apartment apartment);
```

**תיאור הפעולה:** השגרה מקבלת לוח ודירה, ומוחקת את הדירה האחרונה בלוח שזהה בכל הפרמטרים שלה (מבנה הדירה, אורך, רוחב ומחיר) לדירה המתקבלת.
**פרמטרים:** המשתנה service יחזיק את הלוח המתקבל.
המשתנה apartment יחזיק את הדירה שיש לחפש ולמחוק.
**שגיאות נוספות:** במקרה ש- apartment הוא NULL, יש להחזיר APARTMENT_SERVICE_NULL_ARG.
במידה ואין כלל דירות בלוח, יש להחזיר APARTMENT_SERVICE_EMPTY.
במידה ואף אחת מהדירות בלוח לא זהה לדירה שהתקבלה, יש להחזיר APARTMENT_SERVICE_NO_FIT.

**מחיקת לפי מזהה:**
**חתימת הפונקציה:**

```
ApartmentServiceResult serviceDeleteById(ApartmentService service, int id);
```

**תיאור הפעולה:** השגרה מקבלת לוח ומספר מזהה עבור דירה, ומוחקת דירה מהלוח בעלת מזהה זה.
**פרמטרים:** המשתנה service יחזיק את הלוח המתקבל.
המשתנה id יחזיק את מזהה הדירה אותה יש למחוק.
**שגיאות נוספות:** במקרה ש- id הוא שלילי, יש להחזיר APARTMENT_SERVICE_OUT_OF_BOUNDS.
במידה ואין כלל דירות בלוח, יש להחזיר APARTMENT_SERVICE_EMPTY.
במידה ואין אף דירה בלוח בעלת המזהה המתאים, יש להחזיר APARTMENT_SERVICE_NO_FIT.

**העתקת לוח:**
**חתימת הפונקציה:**

```
ApartmentService serviceCopy(ApartmentService service);
```

**תיאור הפעולה:** השגרה מקבלת לוח דירות ומייצרת משתנה חדש אשר זהה ללוח המקורי. הדירות בלוח החדש יהיו עותקים של הדירות בלוח הישן.
**פרמטרים:** המשתנה service יחזיק את הלוח המתקבל.
**שגיאות:** במקרה ש- service הוא NULL או במקרה שהקצאת זיכרון נכשלה - יש להחזיר NULL.

**מחיקת לוח:**
**חתימת הפונקציה:**

```
void serviceDestroy(ApartmentService service);
```

**תיאור הפעולה:** השגרה מקבלת לוח דירות ומשחררת את כל הזיכרון שהוקצה עבורו.

**פרמטרים:** המשתנה service יחזיק את הלוח המתקבל.

### *3.4 מימוש ובדיקות*

#### 3.4.1 בדיקות

על מנת לוודא את נכונות הקוד, נהוג לכתוב בדיקות אוטומטיות לכל טיפוס נתונים שכותבים (unit tests). למטרה זו מסופקים לכם הקבצים apartment_test.c ו- apartment_service_test.c המכילים בדיקות אוטומטיות עבור הטיפוסים Apartment ו- ApartmentService. הבדיקות המצורפות הן בדיקות דוגמא בלבד, ועליכם לבדוק את המימוש עם בדיקות נוספות משלכם. שימו לב כי התרגיל ייבדק עם בדיקות מורכבות בהרבה מבדיקות הדוגמא הנ"ל.

#### 3.4.2 הגבלת על המימוש

- ניתן להוסיף פונקציות עזר נוספות בקבצים apartment.c ו- apartment_service.c
- אין לשנות את הקובץ apartment.h ואת חתימות השגרות המופיעות ב- apartment_service.h
- במימוש הטיפוס apartment_service, אין לגשת בשום שלב לשדות הפנימיים של טיפוס הנתונים apartment.

#### 3.4.3 הידור

התרגיל ייבדק על שרת ה- t2 ועליו לעבור הידור באמצעות 2 הפקודות הבאות:

- gcc -o apartment -std=c99 -Wall -pedantic-errors -Werror -DNDEBUG apartment.c apartment_test.c
- gcc -o service -std=c99 -Wall -pedantic-errors -Werror -DNDEBUG apartment.c apartment_service.c apartment_service_test.c

**שימו לב שעבור apartment_test.c ו- apartment_service_test.c נשתמש בקבצים שלנו. תרגיל אשר אינו מתקמפל או אינו עובר בדיקות יקבל 0. לא יהיו הנחות בנושא זה.**

## 4 הגשה

#### 4.1.1 הגשה יבשה

יש להגיש לתא הקורס את פתרונות השאלות היבשות בלבד.

#### 4.1.2 הגשה רטובה

- את ההגשה הרטובה יש לבצע דרך אתר הקורס תחת Assignments , Exercise 2 , Electronic submission .

יש להגיש קובץ zip (לא RAR או כל דבר אחר) הכולל:

- קובץ בשם dry.pdf הכולל את פתרון החלק היבש.
- הקבצים apartment.c, apartment_service.c ו- apartment_service.h המעודכן.
- תיקייה בשם test שתכיל את הקבצים apartment_test.c ו- apartment_service_test.c שאתם כתבתם.

על הקבצים להיות בראש ה- zip ולא בתוך תיקייה כלשהי (למעט הקבצים הנדרשים במפורש להיות תחת התיקייה test).

**<u>הערה:</u>** קבצי ה- test המסופקים לכם מבצעים include לקבצים ומניחים שהם נמצאים בתיקייה הנוכחית. לצורך הבדיקות שלכם, ובזמן הפיתוח - תוכלו לשים את כל הקבצים באותה התיקייה. טרם ההגשה, העתיקו את קבצי ה- test שלכם בלבד לתיקייה נפרדת. אין צורך לשנות את שורות ה- include לאחר ההעתקה.

# בהצלחה !