# שפת C

מותאם להנדסאי אלקטרוניקה – כיתה י"ג

מספר שאלון: 711911



31 דצמבר 24

1

1

## פרק 6: מצביעים וכתובות

**יעדים**

היכרות עם הדרך בה נשמרים טיפוסים (char, short int, int, long, float, double) שונים בזיכרון המחשב וכן מערכים של טיפוסים שונים. הכרות עם מושגים הכתובת, מצביע לכתובת, תוכן הכתובת, מצביעים לטיפוסי משתנים שונים ומצביעים לפונקציות.

**תכנים**

1. הגדרת מצביעים מטיפוסים שונים
2. אופרטור הכתובת & , ותוכן הכתובת *
3. הקשר בין מצביעים למערכים ולחישוב האינדקסים של מערכים מטיפוסים שונים
4. אריתמטיקה של מצביעים (בהקשר של טיפוס המצביע)
5. מצביעים למערכים ומצביעים למצביעים
6. העברת פרמטרים by value, by reference והדגמה של פונקציית swap
7. אתחולים של מצביעים ושל מערכים
8. שימוש בהקצאות זיכרון דינמיות alloc, malloc, free, sizeof
9. ארגומנטים של שורת פקודה argc, argv
10. מצביעים לפונקציות: הגדרה, שימוש כולל העברת פרמטרים וקבלת ערך מוחזר
11. מערך של מצביעים לפונקציות עבור פניה מהירה
12. מנגנון callback לקבלת התראה על מאורעות

**סיכום שעות ההוראה: 18 שעות עיוניות ו- 8 שעות התנסותיות. סה"כ 26 שעות.**

31 דצמבר 24

2

2

## פרק 6- מצביעים וכתובות:

# 1. הגדרת מצביעים





3

## תזכורת..

- **בשיעורים הראשונים דיברנו על משתנים.**
- **אמרנו שלכל משתנה יש שם, כתובת, וכו'.**
- **כתובת: מקום בזכרון של המחשב, לדוגמא: 7884, וכו'.**

4

# זיכרון המחשב

- **הזיכרון מחולק לבתים כאשר לכל בית יש כתובת.**
- **המשתנים של התוכנית יושבים בתאים בזיכרון. כאשר למשל משתנה מסוג char תופס תא אחד ואילו משתנה מסוג int יושב ב-2 או 4 בתים.**
- **לכל תוכנית מוקצה חלק מהזיכרון. כל פעם שהתוכנית רוצה לכתוב או לקרוא מהזיכרון היא צריכה לציין את הכתובת של אותו משתנה.**

| 1000 | 1001 | 1002 | 1003 | 1004 | 1005 | 1006 | 1007 | 1008 | 1009 | 1010 | 1011 | 1012 | 1013 | 1014 | 1015 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Int x | | | | Char c | | | | | | Long y | | | | | 555 |

5

# מהו מצביע?

- **מצביע הוא משתנה מיוחד בשפת C שמכיל את כתובת הזיכרון של משתנה אחר**
- **במילים אחרות, מצביע "מצביע" על מיקום בזיכרון המחשב.**
- **במקום לאחסן מידע ישירות, המצביע "מצביע" למיקום המידע בזיכרון**





6

# מהו מצביע? (2)


- **דמיינו מצביע כמו תג על מגירה:**
  - **התג (המצביע) מראה איפה המגירה (המידע) נמצאת**
  - **אנחנו משתמשים בתג כדי למצוא את המגירה במהירות**



# נראה זאת כקוד


```
int x = 5;          // משתנה רגיל
int* ptr = &x;      // מצביע למשתנה x
```

x הוא המגירה עם המספר 5

ptr הוא התג\התווית שמראה איפה המגירה של x נמצאת

## מהו פוינטר (מצביע)? (3)

- ההבדל העקרוני בין משתנה לפוינטר: בעוד שמשתנים מאחסנים ערכים, הטיפוס מצביע מאחסן כתובת.
- כל הפוינטרים (בלי שום קשר לאיזה סוג הם מקושרים) הם בגודל 4 בתים.

9

## כיצד מגדירים מצביע?

הגדרת מצביע נעשית על ידי ציון סוג הנתונים שאליו הוא מצביע, ואחריו כוכבית (*) ושם המצביע



10

31 דצמבר 24

10

# סוגי מצביעים בסיסיים

בשפת C, ניתן להגדיר מצביעים לכל סוגי הנתונים הבסיסיים. הנה רשימה של סוגי המצביעים הנפוצים:

1. **char*** - מצביע למשתנה מסוג תו ((characte
2. **int*** - מצביע למשתנה מסוג מספר שלם
3. **float*** - מצביע למשתנה מסוג מספר עשרוני (דיוק יחיד)
4. **double*** - מצביע למשתנה מסוג מספר עשרוני (דיוק כפול)
5. **void*** - מצביע כללי (יכול להצביע על כל סוג של נתונים)





# דוגמאות

```
int *ptr_int;          //מצביע למשתנה מסוג int
char *ptr_char;        // מצביע למשתנה מסוג char
float *ptr_float;      // מצביע למשתנה מסוג float
double *ptr_double;    // מצביע למשתנה מסוג double
void *ptr_void;        // מצביע כללי
```

# חשיבות המצביעים

- גישה יעילה לזיכרון
- עבודה עם מערכים ומחרוזות
- העברת פרמטרים לפונקציות בצורה יעילה
- הקצאה דינמית של זיכרון







# הצהרה על מצביעים(1)

- **על מנת להצהיר על מצביע נכתוב:**

**<type> *<VarName>;**

**For example:**

| פקודה | הסבר פקודה |
|---|---|
| **char *str;** | |
| **int *arr;** | |
| **char *cptr, tav;** | |
| **int num, *ptr1 , p2;** | |

# הצהרה על מצביעים(2)

- על מנת לבצע השמה למשתנה מסוג מצביע לא ניתן לכתוב כך (תתקבל שגיאה):

```
Int *A;
*A = 3;
```

אלא יש לבצע תחילה הקצאת זיכרון למשתנה A.
ז"א כדי שמחוון יצביע על כתובת של משתנה יש לבצע השמה ( העברה ) של כתובת המשתנה אל המחוון.
המשמעות של האופרטור * היא להתייחס לתוכן של המשתנה מסוג מצביע.

דוגמא :

| | |
|---|---|
| הגדרנו שני משתנים מטיפוס שלם. את המשתנה x התחלנו בערך 50. | int x = 50, *px; |
| העברנו אל המחוון (מצביע) px את הכתובת של המשתנה x . ( הסימן & - אמפרסנט - משמש כאן במשמעות של כתובת). | px = &x; |


15

## סיכום ביניים

- אם כן, ניתן להגדיר משתנה שיכיל ערכים שהם כתובות של תאי זכרון - ומשתנה זה נקרא מצביע.
- הכתובת שנמצאת בתוך משתנה זה נקראת: הכתובת אליה הוא מצביע.
- **אופן הגדרת מצביע:**

```
type   *name;
```

- **לדוגמא:**

```
int   *p;
```

16

# תרגול:

| | |
|---|---|
| int *p=0; | |
| int a; | |
| int *r=&a; | |
| p=r; | |
| p=&a; | |

17

17

## פרק 6: מצביעים וכתובות

**יעדים**

היכרות עם הדרך בה נשמרים טיפוסים (char, short int, int, long, float, double) שונים בזיכרון המחשב וכן מערכים של טיפוסים שונים. הכרות עם מושגים הכתובת, מצביע לכתובת, תוכן הכתובת, מצביעים לטיפוסי משתנים שונים ומצביעים לפונקציות.

**תכנים**

1. ✓ הגדרת מצביעים מטיפוסים שונים
2. אופרטור הכתובת & , ותוכן הכתובת *
3. הקשר בין מצביעים למערכים ולחישוב האינדקסים של מערכים מטיפוסים שונים
4. אריתמטיקה של מצביעים (בהקשר של טיפוס המצביע)
5. מצביעים למערכים ומצביעים למצביעים
6. העברת פרמטרים by value, by reference והדגמה של פונקציית swap
7. אתחולים של מצביעים ושל מערכים
8. שימוש בהקצאות זיכרון דינמיות alloc, malloc, free, sizeof
9. ארגומנטים של שורת פקודה argc, argv
10. מצביעים לפונקציות: הגדרה, שימוש כולל העברת פרמטרים וקבלת ערך מוחזר
11. מערך של מצביעים לפונקציות עבור פניה מהירה
12. מנגנון callback לקבלת התראה על מאורעות

**סיכום שעות ההוראה: 18 שעות עיוניות ו- 8 שעות התנסותיות. סה"כ 26 שעות.**

18 31 דצמבר 24

18

## פרק 6- מצביעים וכתובות:

# 2. אופרטור הכתובת (&) ואופרטור התוכן (*)






## אופרטור הכתובת (&)

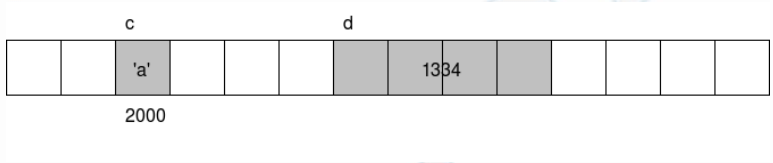
- האופרטור & משמש לקבלת הכתובת בזיכרון של משתנה.
- **הסבר:**
- כאשר מציבים & לפני שם של משתנה, מתקבלת **הכתובת** בזיכרון שבה **המשתנה** מאוחסן.
- זה שימושי במיוחד כאשר רוצים להציב את כתובת המשתנה בתוך מצביע.

**<u>לדוגמה</u>**:

```
int x = 5;
int *ptr = &x;  // ptr מכיל כעת את הכתובת של X
```

- **כאשר אנו מגדירים משתנים, אין לנו שליטה לגבי מיקומם בזיכרון - המהדר ומערכת ההפעלה קובעים זאת.**

**- אם למשל, נגדיר שני משתנים C,D , העובדה ש-C הוגדר לפני D לא מבטיח לנו ש – C יהיה לפני D בזיכרון.**




```c
#include <stdio.h>

int main()
{
    int C, D;
    printf("%p\n%p\n",&C,&D);
     return 0;
}
```

```
C:\Windows\system32\cmd.exe
0015F944
0015F938
Press any key to continue . . .
```

21

# אופרטור התוכן (*)

- האופרטור * משמש לגישה לערך שנמצא בכתובת שאליה מצביע המצביע.

**הסבר:**

- כאשר מציבים * לפני שם של מצביע, מתקבל **<u>הערך</u>** שנמצא בכתובת שאליה המצביע מצביע.
- מכונה לעתים "דה-רפרנס" ( (dereferencing של המצביע.

דוגמה:

```
int x = 5;
int *ptr = &x;
printf(" הערך של x:  %d\n", *ptr);
```

ידפיס 5 //





22

# חשוב לשים לב:

- **אופרטור & (address): כאמור מספק לנו את כתובת המשתנה.**
- **אופרטור * (indirection): בעל שני משמעויות:**
  - **בהצהרה – יצירת פוינטר.**
  - **לאחר הצהרה – גישה למשתנה עליו הפוינטר מצביע.**

**(גישה לערך שהכתובת שלו שמורה בתוך משתנה הפוינטר).**

23

# דוגמא

```
int x = 50, *px;
px = &x;
```

| שם המשתנה | כתובת | ערך |
|---|---|---|
| x | 1000 | 50 |
| px | 1002 | 1000 |

**נניח שלשני המשתנים שהוגדרו, הוקצו הכתובות 1000 ו 1002 בזיכרון.**
**המשתנה x הותחל בערך 50 .**
**המשפט השני אומר: העבר למשתנה px את כתובתו של המשתנה x .**
**לכן, רואים ש px קיבל את הערך 1000 שהיא כתובתו של x .**

24

## דוגמה מעשית

```c
#include <stdio.h>
int main() {
    int num = 42;
    int *ptr_num = &num;          // & returns the address of num
    printf("Value of num: %d\n", num);
    printf("Address of num: %p\n", &num);
    printf("Value of ptr_num (address): %p\n", ptr_num);
    printf("Value pointed to by ptr_num: %d\n", *ptr_num);

    return 0;
}
```

```
Value of num: 42
Address of num: 0x7ffd46b65fbc
Value of ptr_num (address): 0x7ffd46b65fbc
Value pointed to by ptr_num: 42
```





25

## דוגמה מס' 2

```c
#include <stdio.h>
void main() {
    int x = 10;
    int *ptr = &x;
    printf("הערך של x: %d\n", x);
    printf("הכתובת של x: %p\n", &x);
    printf("הערך של ptr (כתובת): %p\n", ptr);
    printf("הערך שאליו מצביע ptr: %d\n", *ptr);

    printf ("\n\n");
    *ptr = 20;  // שינוי הערך של x באמצעות המצביע
     printf("הערך החדש של x: %d\n", x);
    printf("הערך החדש של *ptr: %d\n", *ptr);
    printf ("\n\n");
    x=-33;
    printf("הערך החדש של *ptr: %d\n", *ptr);
    printf("הערך החדש של x: %d\n", x);
}
```

```
Output
/tmp/NBtB34zWEp.o
הערך של x: 10
הכתובת של x: 0x7ffd81fbc814
הערך של ptr (כתובת): 0x7ffd81fbc814
הערך שאליו מצביע ptr: 10


הערך החדש של x: 20
הערך החדש של *ptr: 20


הערך החדש של *ptr: -33
הערך החדש של x: -33
```





26

# תרגיל כיתה

**נתון קטע התוכנית הבא:**

```
int x, y=10, *px, *py ;
px = &x ;
py = &y ;
*px = 25 ;
y = *py + *px ;
(*px)++;
Printf("%d  %d ",x,y);
```

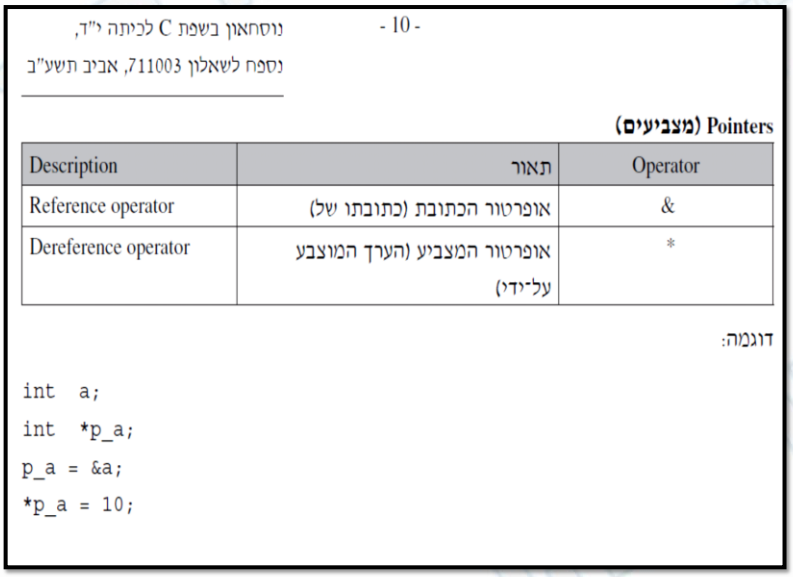
**א.רשום מה יהיה פלט התוכנית.**
**ב. הסבר במילותיך כל אחת משורות הקוד בתוכנית**

27

| | | |
|---|---|---|
| | int x,y=10, *px, *py ; | .1 |
| | px = &x ; | .2 |
| | py = &y ; | .3 |
| | *px = 25 ; | .4 |
| | y = *py + *px ; | .5 |
| | (*px)++; | .6 |
| | Printf("%d  %d ",x,y); | .7 |

28

# מצביעים בדף נוסחאות



נוסחאון בשפת C לכיתה י"ד,
נספח לשאלון 711003, אביב תשע"ב

- 10 -

**Pointers (מצביעים)**

| Operator | תאור | Description |
|---|---|---|
| & | אופרטור הכתובת (כתובתו של) | Reference operator |
| * | אופרטור המצביע (הערך המוצבע על-ידי) | Dereference operator |

דוגמה:

```
int  a;
int  *p_a;
p_a = &a;
*p_a = 10;
```

29

# תרגיל כיתה 2

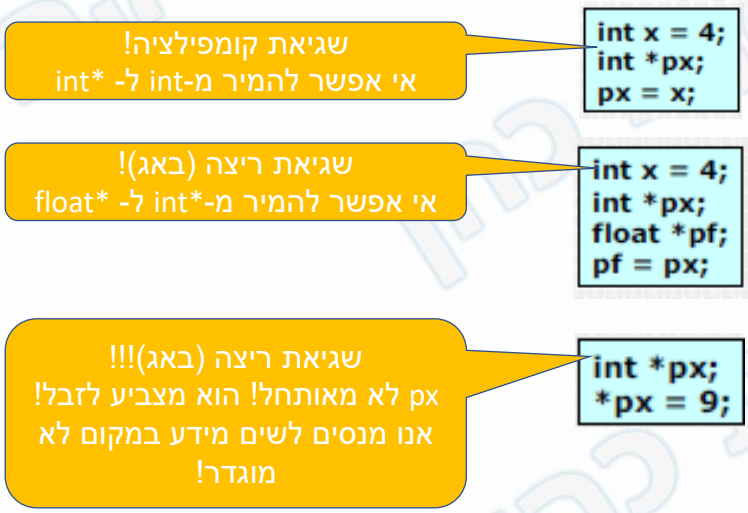
**נתון קטע התוכנית הבא:**

```
1.    int a=3, b=6, *pa, *pb;
2.    pa = &a ;
3.    pb = &b ;
4.    *pa = *pa + *pb ;
5.    a++;
6.    (*pa)++;
7.    printf("a=%d b=%d \n ",a,b);
8.    *pb = *pa;
9.     printf("a=%d b=%d ",*pa,*pb);
```

**א) הסבר כל שורה בקטע הנ"ל.**
**ב) רשום מה יהיה פלט התוכנית.**

30

# זהירות!!!




31

31

**<u>תרגיל לתלמידים:</u>**

צרו תוכנית שמגדירה שני משתנים מסוג int.
צרו מצביע שמצביע על המשתנה הראשון.
הדפיסו את הערכים של שני המשתנים.
השתמשו במצביע כדי לשנות את הערך של המשתנה הראשון.
החליפו את הכתובת שבמצביע כך שיצביע על המשתנה השני.
שנו את הערך של המשתנה השני באמצעות המצביע.
הדפיסו שוב את הערכים של שני המשתנים.

32



31 דצמבר 24

32

## פרק 6: מצביעים וכתובות

**יעדים**

היכרות עם הדרך בה נשמרים טיפוסים (char, short int, int, long, float, double) שונים בזיכרון המחשב וכן מערכים של טיפוסים שונים. הכרות עם מושגים הכתובת, מצביע לכתובת, תוכן הכתובת, מצביעים לטיפוסי משתנים שונים ומצביעים לפונקציות.

**תכנים**

1. ✓ הגדרת מצביעים מטיפוסים שונים
2. ✓ אופרטור הכתובת & , ותוכן הכתובת *
3. הקשר בין מצביעים למערכים ולחישוב האינדקסים של מערכים מטיפוסים שונים
4. אריתמטיקה של מצביעים (בהקשר של טיפוס המצביע)
5. מצביעים למערכים ומצביעים למצביעים
6. העברת פרמטרים by value, by reference והדגמה של פונקציית swap
7. אתחולים של מצביעים ושל מערכים
8. שימוש בהקצאות זיכרון דינמיות alloc, malloc, free, sizeof
9. ארגומנטים של שורת פקודה argc, argv
10. מצביעים לפונקציות: הגדרה, שימוש כולל העברת פרמטרים וקבלת ערך מוחזר
11. מערך של מצביעים לפונקציות עבור פניה מהירה
12. מנגנון callback לקבלת התראה על מאורעות

**סיכום שעות ההוראה: 18 שעות עיוניות ו- 8 שעות התנסותיות. סה"כ 26 שעות.**





# פרק 6- מצביעים וכתובות:

# 3. הקשר בין מצביעים למערכים ולחישוב האינדקסים של מערכים מטיפוסים שונים

## תזכורת

- & : **כתובת המשתנה.**
- * : **תוכן הכתובת שהמשתנה מצביע עליו**







- & : **כתובת המשתנה.**
- * : **תוכן הכתובת שהמשתנה מצביע עליו**

## הקשר בין מצביעים למערכים

**<u>שם המערך כמצביע</u>**

- שם המערך הוא למעשה מצביע קבוע לאיבר הראשון במערך.
- &arr[0] זהה ל-arr

**<u>גישה לאיברי המערך באמצעות מצביעים</u>**

- ניתן לגשת לאיברי המערך באמצעות אריתמטיקת מצביעים
- arr[i] זהה ל- *(arr+i)

# מצביעים ומערכים

- סריקת המערך אפשרית על ידי שימוש האופרטורים:
  - אופרטור []
  - אופרטור *

למעשה כשדיברנו על מערכים עד עכשיו ורשמנו array[i]
הקומפיילר התייחס לזה כמו *(ptr+i)

37

37

# דוגמה להמחשה

```
int arr[5] = {10, 20, 30, 40, 50};
int* ptr = arr;

printf("%d\n", *ptr);
printf("%d\n", *(ptr + 2));
```

38



31 דצמבר 24

38

# השפעת הטיפוס על חישוב האינדקס

• נוסחת חישוב האינדקס:

כתובת של איבר במערך = כתובת הבסיס + (אינדקס * גודל הטיפוס)

**זוהי הכתובת של האיבר הראשון במערך (אינדקס 0).**

**זהו המיקום של האיבר במערך (מתחיל מ-0).**

**זהו הגודל בבתים של כל איבר במערך.**

**<u>חשוב לציין</u>** שכאשר אנחנו כותבים arr + i בקוד C המהדר מבצע את החישוב הזה עבורנו אוטומטית.
ז"א הוא יודע את הגודל של הטיפוס ומכפיל את האינדקס בגודל הזה.
לכן מספיק לרשום arr+מיקום האינדקס הרצוי :)





# השפעת הטיפוס על חישוב האינדקס (2)

• גודל הטיפוס משפיע על המרחק בין איברי המערך בזיכרון.

• לדוגמה, int לעומת char

```
int arr_int[3] = {1, 2, 3};
char arr_char[3] = {'a', 'b', 'c'};

printf("%ld\n", sizeof(int));  // בדרך כלל 4
printf("%ld\n", sizeof(char)); // תמיד 1

printf("%p\n", (arr_int + 1));    // יגדל ב-4 בתים
printf("%p\n", (arr_char + 1));   // יגדל בבית אחד
```

```
Output
/tmp/LmkhSeKi7Q.o
4
1
0x7fffc16bf0e8
0x7fffc16bf0e2
```

## פרק 6: מצביעים וכתובות

**יעדים**

היכרות עם הדרך בה נשמרים טיפוסים (char, short int, int, long, float, double) שונים בזיכרון המחשב וכן מערכים של טיפוסים שונים. הכרות עם מושגים הכתובת, מצביע לכתובת, תוכן הכתובת, מצביעים לטיפוסי משתנים שונים ומצביעים לפונקציות.

**תכנים**

1. ✓ הגדרת מצביעים מטיפוסים שונים
2. ✓ אופרטור הכתובת & , ותוכן הכתובת *
3. ✓ הקשר בין מצביעים למערכים ולחישוב האינדקסים של מערכים מטיפוסים שונים
4. אריתמטיקה של מצביעים (בהקשר של טיפוס המצביע)
5. מצביעים למערכים ומצביעים למצביעים
6. העברת פרמטרים by value, by reference והדגמה של פונקציית swap
7. אתחולים של מצביעים ושל מערכים
8. שימוש בהקצאות זיכרון דינמיות alloc, malloc, free, sizeof
9. ארגומנטים של שורת פקודה argc, argv
10. מצביעים לפונקציות: הגדרה, שימוש כולל העברת פרמטרים וקבלת ערך מוחזר
11. מערך של מצביעים לפונקציות עבור פניה מהירה
12. מנגנון callback לקבלת התראה על מאורעות

**סיכום שעות ההוראה: 18 שעות עיוניות ו- 8 שעות התנסותיות. סה"כ 26 שעות.**



41

# פרק 6- מצביעים וכתובות:
# 4. אריתמטיקה של מצביעים





42

# עקרונות בסיסיים

- מצביע מכיל כתובת זיכרון
- אריתמטיקה של מצביעים פועלת על כתובות זיכרון
- הפעולות מתחשבות בגודל הטיפוס אליו מצביע המצביע







# תזכורת: חשיבות גודל הטיפוס

- כל טיפוס נתונים תופס כמות שונה של זיכרון
- לדוגמה: sizeof(int)= בדרך כלל 4
- sizeof(char) = 1

```
int arr[3] = {10, 20, 30};
int* ptr = arr;
printf("%d\n", *ptr);     // 10
printf("%d\n", *(ptr+1)); // 20
printf("%d\n", *(ptr+2)); // 30
```

# פעולות אריתמטיות על מצביעים

1. הוספה וחיסור של מס' שלמים
2. הפרש בין מצביעים
3. אינקרמנט ודקרמנט
4. השוואת מצביעים







# 1. הוספה וחיסור של מספרים שלמים

- ptr + n : מזיז את המצביע n יחידות קדימה
- ptr - n : מזיז את המצביע n יחידות אחורה

```
int arr[5] = {10, 20, 30, 40, 50};
int* ptr = arr;
printf("%d\n", *(ptr+2));  // 30
printf("%d\n", *(ptr-1));  // Undefined
```

```
Output
/tmp/GMddj2Mvov.o
30
0

=== Code Execution Successful ===
```

# 2. הפרש בין מצביעים

- מחזיר את מספר האיברים בין שני מצביעים
- חייב להיות מאותו טיפוס ובאותו מערך

```
int arr[5] = {10, 20, 30, 40, 50};
int* start = arr;
int* end = &arr[4];
printf("מספר האיברים.%d  \n", end - start);  // 4
```







# 3. אינקרמנט ודקרמנט

- ptr++ מזיז את המצביע איבר אחד קדימה
- ptr-- מזיז את המצביע איבר אחד אחורה

```
int arr[3] = {10, 20, 30};
int* ptr = arr;
printf("%d\n", *ptr++);  // 10 (מדפיס 10, ואז מזיז את המצביע)
printf("%d\n", *ptr);    // 20
```

# שימו לב

- במצביעים יש שוני מהותי בין

**(ptr*)++** לבין **++ptr***

ptr*++ מגדיל את הערך שאליו מצביע
ptr. *++מחזיר את הערך שאליו מצביע ,ptr ואז מקדם את ptr לאיבר הבא







# הבדלים

| *ptr++ | ++*ptr |
|---|---|
| לא משנה את הערכים בזיכרון | משנה את הערך בזיכרון |
| מזיז את המצביע לאיבר הבא | המצביע נשאר באותו מקום |
| מחזיר את הערך המקורי (לפני ההזזה) | מחזיר את הערך החדש והמוגדל |

# 3. אינקרמנט ודקרמנט

- ptr++ מזיז את המצביע איבר אחד קדימה
- ptr-- מזיז את המצביע איבר אחד אחורה

```
int arr[3] = {10, 20, 30};
int* ptr = arr;
printf("%d\n", *ptr++);        // 10 (מדפיס 10, ואז מזיז את המצביע)
printf("%d\n", *ptr);          // 20

printf("%d\n", ++*ptr);        //21
printf("%d\n", *ptr);          //21
```







# 4. השוואת מצביעים

- אופרטורים להשוואה ==, !=, <, >, <=, >=
- משמשים להשוואת כתובות זיכרון

- דוגמה:

```
int arr[5] = {10, 20, 30, 40, 50};
int* p1 = &arr[1];
int* p2 = &arr[3];

if (p1 < p2)
   printf("p1 מצביע על איבר מוקדם יותר במערך");
```

**<u>שימושים נפוצים</u>**

- בדיקת גבולות
- מעבר על מערך בעזרת לולאות

# 4. השוואת מצביעים (2)

- **השוואת מצביעים:**
  - **לכל שני פוינטרים ניתן לבדוק האם הם מצביעים לאותו מיקום.**
  - **לכל פוינטר ניתן לבדוק האם הוא NULL.**

```
int *p1 = NULL, *p2 = NULL;
if (p1==p2)…
if (p1 < p2) …
if (p1!=NULL)…
```

**מתכנת תמיד חייב לוודא שפוינטר תקף (מצביע לכתובת חוקית) לפני שהוא משתמש באופרטור * על אותו פוינטר!**

53

53

פרק 6: מצביעים וכתובות

**יעדים**

היכרות עם הדרך בה נשמרים טיפוסים (char, short int, int, long, float, double) שונים בזיכרון המחשב וכן מערכים של טיפוסים שונים. הכרות עם מושגים הכתובת, מצביע לכתובת, תוכן הכתובת, מצביעים לטיפוסי משתנים שונים ומצביעים לפונקציות.

**תכנים**

1. ✓ הגדרת מצביעים מטיפוסים שונים
2. ✓ אופרטור הכתובת & , ותוכן הכתובת *
3. ✓ הקשר בין מצביעים למערכים ולחישוב האינדקסים של מערכים מטיפוסים שונים
4. ✓ אריתמטיקה של מצביעים (בהקשר של טיפוס המצביע)
5. מצביעים למערכים ומצביעים למצביעים
6. העברת פרמטרים by value, by reference והדגמה של פונקציית swap
7. אתחולים של מצביעים ושל מערכים
8. שימוש בהקצאות זיכרון דינמיות alloc, malloc, free, sizeof
9. ארגומנטים של שורת פקודה argc, argv
10. מצביעים לפונקציות: הגדרה, שימוש כולל העברת פרמטרים וקבלת ערך מוחזר
11. מערך של מצביעים לפונקציות עבור פניה מהירה
12. מנגנון callback לקבלת התראה על מאורעות

**סיכום שעות ההוראה: 18 שעות עיוניות ו- 8 שעות התנסותיות. סה"כ 26 שעות.**

54

31 דצמבר 24

54

## פרק 6- מצביעים וכתובות:

# 5. מצביעים למערכים ומצביעים למצביעים



55

## מצביעים למערכים

- **מהו מערך?** אוסף של איברים מאותו סוג, מאוחסנים ברצף בזיכרון
- **הקשר בין מערכים למצביעים:** שם המערך הוא למעשה **מצביע לאיבר הראשון במערך**

```
int arr[5] = {10, 20, 30, 40, 50};
int* ptr = arr;        // ptr מצביע לתחילת המערך

printf("%d", *ptr);          // 10 ידפיס
printf("%d", *(ptr + 2));  // 30 ידפיס
```

**נקודות חשובות:**

1. arr == &arr[0] שם המערך זהה **לכתובת** של האיבר הראשון
2. ניתן לגשת לאיברי המערך באמצעות אריתמטיקת מצביעים:

ptr[i] <=> *(ptr+i)



56

# תרגיל

- מה יודפס בקוד הבא?

```
int arr[3] = {100, 200, 300};
int* p = arr;
printf("%d", *(p + 1));
```

**200**







# דוגמה - גישה לאיברי מערך באמצעות מצביע

```
void printArray(int* arr, int size) {
  for(int i = 0; i < size; i++) {
    printf("%d ", *(arr + i));  // הדפסת האיבר בעזרת אריתמטיקת מצביעים
     printf("%d ", arr[i]);  // שקול לשורה הקודמת
      }
}
```

# מצביע למצביע

מצביע למצביע משמש לשמירת כתובת של מצביע

שימושי במיוחד בהעברת מצביעים לפונקציות כשרוצים לשנות את ערך המצביע עצמו

```
int x = 5;
int* ptr = &x;    // מצביע למשתנה
int** pptr = &ptr; // מצביע למצביע
printf("%d\n", **pptr);
```

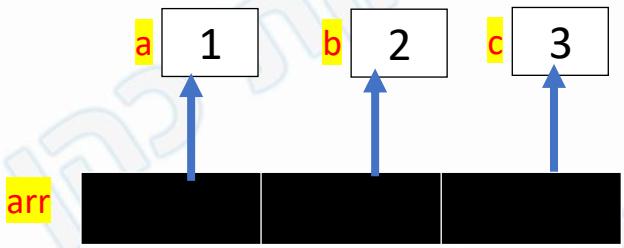
```
יודפס : 5
```






# מערך של מצביעים

```
int a = 1, b = 2, c = 3;
int* arr[3];   // מערך של מצביעים
arr[0] = &a;
arr[1] = &b;
arr[2] = &c;
```




```
printf("%d %d %d\n", *arr[0], *arr[1], *arr[2]); // 3 2 1 ידפיס
```

# מערך של מצביעים





61

# תרגול 1

**כתבו פונקציה שמקבלת מערך של מצביעים למספרים שלמים ומחזירה את סכום כל המספרים**




62

## תרגול 2

- כתבו פונקציה שמקבלת מערך של מצביעים למספרים שלמים וגודל מערך, ומחזירה את המספר המקסימלי מבין כל המספרים.




63

## תרגול 3

כתבו פונקציה שמקבלת מערך של מצביעים למספרים שלמים וגודל מערך, ומחזירה את מספר האיברים החיוביים במערך.




64

## תרגול 4

- **כתבו פונקציה שמקבלת מערך של מצביעים למספרים שלמים וגודל מערך, ומכפילה כל מספר זוגי פי 2.**




65

## תרגול עצמי

**תרגיל 1** :כתבו פונקציה שמקבלת מערך של מצביעים למספרים שלמים וגודל מערך, ומחזירה את ממוצע המספרים.

**תרגיל 2:** כתבו פונקציה שמקבלת מערך של מצביעים למספרים שלמים וגודל מערך, ומחזירה את המספר הקטן ביותר במערך.

**תרגיל 3:**כתבו פונקציה שמקבלת מערך של מצביעים למספרים שלמים וגודל מערך, ומחזירה את כמות המספרים הזוגיים במערך.




66

## פרק 6: מצביעים וכתובות

**יעדים**

היכרות עם הדרך בה נשמרים טיפוסים (char, short int, int, long, float, double) שונים בזיכרון המחשב וכן מערכים של טיפוסים שונים. הכרות עם מושגים הכתובת, מצביע לכתובת, תוכן הכתובת, מצביעים לטיפוסי משתנים שונים ומצביעים לפונקציות.

**תכנים**

1. ✓ הגדרת מצביעים מטיפוסים שונים
2. ✓ אופרטור הכתובת & , ותוכן הכתובת *
3. ✓ הקשר בין מצביעים למערכים ולחישוב האינדקסים של מערכים מטיפוסים שונים
4. ✓ אריתמטיקה של מצביעים (בהקשר של טיפוס המצביע)
5. ✓ מצביעים למערכים ומצביעים למצביעים
6. העברת פרמטרים by value, by reference והדגמה של פונקציית swap
7. אתחולים של מצביעים ושל מערכים
8. שימוש בהקצאות זיכרון דינמיות alloc, malloc, free, sizeof
9. ארגומנטים של שורת פקודה argc, argv
10. מצביעים לפונקציות: הגדרה, שימוש כולל העברת פרמטרים וקבלת ערך מוחזר
11. מערך של מצביעים לפונקציות עבור פניה מהירה
12. מנגנון callback לקבלת התראה על מאורעות

**סיכום שעות ההוראה: 18 שעות עיוניות ו- 8 שעות התנסותיות. סה"כ 26 שעות.**



67

## פרק 6- מצביעים וכתובות:

# 6. העברת פרמטרים והדגמת פונקציית Swap





68

# מבוא - שיטות להעברת פרמטרים

- למדנו את זה במהלך השיעורים – אבל עכשיו בואו נתן לזה "שם".
- בשפת C קיימות שתי שיטות עיקריות להעברת פרמטרים לפונקציה:
  - העברה לפי ערך- By Value
  - העברה לפי כתובת/הפניה By Reference

**נלמד את ההבדלים ביניהן ונדגים באמצעות פונקציית Swap**







# העברה By Value

```
void changeNumber(int x) {
   x = x + 1;   // השינוי יהיה מקומי בלבד
}
int main() {
   int num = 5;
   changeNumber(num);
   printf("%d\n", num);
    return 0;
}
```

**ידפיס 5 - הערך לא השתנה**

## העברה By Value

```
voi
  x
}
int
  i
  c
  p
  r
}
```

בהעברה by value מועבר העתק של הערך
שינויים בפונקציה לא משפיעים על המשתנה המקורי






## העברה By Reference

```
void changeNumber(int* x) {
   *x = *x + 1;   // השינוי ישפיע על המשתנה המקורי
}

int main() {
   int num = 5;
   changeNumber(&num);
   printf("%d\n", num);  // ידפיס 6 - הערך השתנה!
    return 0;
}
```

**ידפיס 6 - הערך השתנה**

## By Reference העברה

```
void changeNumber(int* x) {
   *x = *
}

int mai
   int nu
   chang
   printf
   retu
}
```

בהעברה by reference מועברת כתובת המשתנה
שינויים בפונקציה משפיעים על המשתנה המקורי
משתמשים במצביעים כדי לממש העברה by reference







**תרגיל:**
**כתוב פונקציה שמקבלת שני מספרים שלמים ומבצעת החלפה בין תוכני הערכים.**
**למשל אם x=3**
**y=5**
**לאחר הפונקציה:**
**x=5**
**y=3**

**ניסיון שגוי:**

```
void swap_wrong(int x, int y) {
   int temp = x;
   x = y;
   y = temp;
}

int main() {
   int a = 5, b = 10;
   swap_wrong(a, b);
   printf("%d %d\n", a, b); return 0;
}
```

**תרגיל:**
**כתוב פונקציה שמקבלת שני מספרים שלמים ומבצעת החלפה בין תוכני הע למשל**
**לאחר**
**3**

**ניסיון שגוי:**

```
void swap_wrong(int x, int y) {
   int temp = x;
   x = y;
   y = temp;
}

int main() {
   int a = 5, b = 10;
   swap_wrong(a, b);
   printf("%d %d\n", a, b); return 0;
}
```

למה זה לא עובד????







**תרגיל:**
**כתוב פונקציה שמקבלת שני מספרים שלמים ומבצעת החלפה בין תוכני הערכים.**
**למשל אם x=3**
**y=5**
**לאחר הפונקציה:**
**x=5**
**y=3**

**הפתרון הנכון:**

```
void swap_correct(int* x, int* y) {
   int temp = *x;
   *x = *y;
   *y = temp;
}

int main() {
   int a = 5, b = 10;
   swap_correct(&a, &b);
   printf("%d %d\n", a, b);  // עבד! - 5 10 ידפיס
    return 0;
}
```

# יתרונות וחסרונות

- **By Reference:**
  - **<u>יתרונות</u>**:
    - חסכוני במשאבים - מעבירים רק כתובת
    - מאפשר שינוי של המשתנה המקורי
  - **<u>חסרונות:</u>**
    - פחות בטוח - הפונקציה יכולה לשנות מידע בטעות
    - דורש תשומת לב בעבודה עם מצביעים

- **By Value:**
  - **<u>יתרונות</u>**:
    - בטוח יותר - הפונקציה לא יכולה לשנות את המשתנה המקורי
    - קל להבנה
  - **<u>חסרונות</u>**:
    - העתקת מידע יכולה להיות יקרה במשאבים
    - לא ניתן לשנות את המשתנה המקורי







# מתי להשתמש במה?

- **נשתמש ב- By Value כאשר:**
- אתם רק קוראים מידע
- המידע קטן כמו (int, char)
- אתם רוצים להגן על המידע המקורי
- **נשתמש ב- By Reference כאשר:**
- אתם צריכים לשנות את המשתנה המקורי
- המידע גדול (כמו מערכים, מבנים)
- אתם רוצים לחסוך במשאבים

# תרגול

1.כתבו פונקציה המקבלת שני מספרים ומחליפה ביניהם רק אם הראשון גדול מהשני.

2.כתבו פונקציה המקבלת שלושה מספרים וממיינת אותם בסדר עולה





79

כתבו פונקציה
המקבלת
שלושה
מספרים
וממיינת אותם
בסדר עולה





80

## פרק 6: מצביעים וכתובות

**יעדים**

היכרות עם הדרך בה נשמרים טיפוסים (char, short int, int, long, float, double) שונים בזיכרון המחשב וכן מערכים של טיפוסים שונים. הכרות עם מושגים הכתובת, מצביע לכתובת, תוכן הכתובת, מצביעים לטיפוסי משתנים שונים ומצביעים לפונקציות.

**תכנים**

1. ✓ הגדרת מצביעים מטיפוסים שונים
2. ✓ אופרטור הכתובת & , ותוכן הכתובת *
3. ✓ הקשר בין מצביעים למערכים ולחישוב האינדקסים של מערכים מטיפוסים שונים
4. ✓ אריתמטיקה של מצביעים (בהקשר של טיפוס המצביע)
5. ✓ מצביעים למערכים ומצביעים למצביעים
6. ✓ העברת פרמטרים by value, by reference והדגמה של פונקציית swap
7. אתחולים של מצביעים ושל מערכים
8. שימוש בהקצאות זיכרון דינמיות alloc, malloc, free, sizeof
9. ארגומנטים של שורת פקודה argc, argv
10. מצביעים לפונקציות: הגדרה, שימוש כולל העברת פרמטרים וקבלת ערך מוחזר
11. מערך של מצביעים לפונקציות עבור פניה מהירה
12. מנגנון callback לקבלת התראה על מאורעות

**סיכום שעות ההוראה: 18 שעות עיוניות ו- 8 שעות התנסותיות. סה"כ 26 שעות.**





# פרק 6- מצביעים וכתובות:

# 7. אתחולים של מצביעים ושל מערכים

# מהו אתחול?

- אתחול הוא מתן ערך התחלתי למשתנה
- חשוב מאוד לאתחל משתנים כדי למנוע באגים ובעיות בתוכנית
- אתחול נעשה בדרך כלל בהצהרה על המשתנה

**<u>דוג:</u>**

- int number=5;







# אתחול מצביעים 👉

- יש כמה דרכים לאתחל מצביע:

1) אתחול לכתובת של משתנה קיים:

```
int x = 10;
int* ptr = &x;    // המצביע מאותחל לכתובת של x
```

# אתחול מצביעים 👉

- יש כמה דרכים לאתחל מצביע:

2) אתחול ל- NULL:

```
int* ptr = NULL;  // מצביע ריק שלא מצביע לשום מקום
```







# אתחול מצביעים 👉

- יש כמה דרכים לאתחל מצביע:

3) אתחול עם הקצאה דינמית:

```
int* ptr = (int*)malloc(sizeof(int));
```

# מהו malloc? 🧩

- malloc זו פונקציה שמשמשת להקצאת זיכרון דינמית
- malloc = Memory ALLOCation = הקצאת זיכרון
- malloc מבקש מהמערכת להקצות לנו זיכרון בגודל שאנחנו צריכים
- malloc מחזיר מצביע לזיכרון שהוקצה





87

# צעד אחר צעד: 🚶

1. "היי מחשב! 👋
2. תמצא לי בזיכרון מקום פנוי
3. שיש בו מספיק מקום ל-5 מספרים שלמים
4. (כל מספר שלם תופס 4 בתים, אז סה"כ צריך 20 בתים)"

```
malloc(5 * sizeof(int))
```





88

# צעד אחר צעד: 🚶

המחשב מחפש בזיכרון שלו:

המחשב מצא 20 בתים פנויים , ואז.....

1. המחשב "שם סימון" על ה-20 בתים האלה:
   1. כמו לשים דגלון שאומר "תפוס! שמור למערך שלך!"
   2. ומחזיר את הכתובת של תחילת האזור הזה
2. המצביע arr מקבל את הכתובת הזו ועכשיו הוא "יודע" איפה המערך שלנו בזיכרון





89

# מגרש חנייה


- זה כמו להגיע למגרש חניה:
- אתה אומר לשומר: "אני צריך מקום ל-5 מכוניות"
- השומר מוצא 5 מקומות חניה רצופים
- סוגר אותם עם קונוסים
- ונותן לך את המספר של החניה הראשונה





90

# איך משתמשים ב-malloc ?

```
// התבנית הבסיסית
pointer = (type*)malloc(size_in_bytes);

// דוגמה למספר שלם
int* ptr = (int*)malloc(sizeof(int));

// דוגמה למערך של 5 מספרים
int* arr = (int*)malloc(5 * sizeof(int));
```







# חשוב לדעת! ⚠️

**1. תמיד לבדוק אם malloc הצליח:**

```
int* ptr = (int*)malloc(sizeof(int));
if (ptr == NULL) {
   printf("הקצאת הזיכרון נכשלה");
    return -1;
}
```

**2. תמיד לשחרר את הזיכרון בסיום השימוש:**

```
free(ptr);
ptr = NULL;              // מונע שימוש בזיכרון שכבר שוחרר
```

# למה חשוב לאתחל מצביעים? ⚠️

- מצביע לא מאותחל  משמעותו שהוא מכיל ערך אקראי
- שימוש במצביע לא מאותחל יכול לגרום ל:
  - קריסת התוכנית
  - באגים מסוכנים
  - התנהגות לא צפויה







# אתחול מערכים – מערך סטטי

```
// אתחול מערך מפורש
int arr1[5] = {1, 2, 3, 4, 5};

// 0 אתחול חלקי - השאר יהיה
int arr2[5] = {1, 2};  // {1, 2, 0, 0, 0}

// אתחול אוטומטי לאפסים
int arr3[5] = {0};     // {0, 0, 0, 0, 0}
```

# דוגמה : מסעדה 🍽️


**הקצאה רגילה (סטטית)** = כמו מסעדה עם 50 כיסאות קבועים מברזל שמקובעים לרצפה

- חייבים להחליט מראש כמה כיסאות (גודל המערך)
- אי אפשר להוסיף או להוריד כיסאות
- תמיד תופסים מקום, גם אם באים רק 10 אנשים
- הגודל חייב להיות ידוע בזמן הקומפילציה (כשכותבים את הקוד)

**הקצאה דינמית** = כמו מסעדה שמביאה כיסאות לפי כמות האנשים שמגיעים

- אפשר להחליט על הגודל תוך כדי ריצה (למשל, לפי קלט מהמשתמש)
- אפשר להגדיל או להקטין את הכמות לפי הצורך
- לוקחים בדיוק את הזיכרון שצריך
- הגודל יכול להשתנות במהלך הריצה

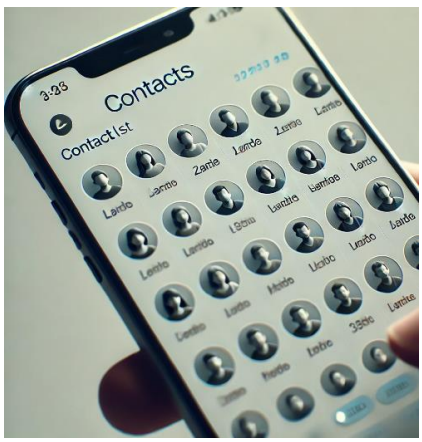
c





95

# דוגמה: אנשי קשר בפלאפון




נניח שאתם כותבים אפליקציה לניהול אנשי קשר:

- **גישה סטטית:** מערך של 1000 אנשי קשר - בזבוז זיכרון למשתמש עם 10 אנשי קשר, ובעיה למשתמש עם 1001 אנשי קשר
- **גישה דינמית:** המערך גדל לפי כמות אנשי הקשר שהמשתמש באמת צריך





96

# קניית בגדים לילד

- **הקצאה סטטית** = לקנות בגד בגודל 14 כשהילד בן 8:
- הבגד גדול מדי עכשיו (בזבוז משאבים)
- אולי יהיה קטן מדי כשיגיע לגיל 14
- אי אפשר להחליף אותו אם לא מתאים

- **הקצאה דינמית** = לקנות בגד לפי המידה הנוכחית:
- מתאים בדיוק למה שצריך
- אפשר להחליף כשצריך מידה חדשה
- לא מבזבזים כסף על בגד גדול מדי



# דוגמה קוד

**// הקצאה רגילה - סטטית**

```
int arr[50];  // חייבים להחליט על 50 בזמן כתיבת הקוד
// אם נצטרך 51 תאים? אין אפשרות להגדיל!
```

**// הקצאה דינמית**

```
int size;
printf(" ?מה מספרים תרצה להכניס");
scanf("%d", &size);     // המשתמש מחליט בזמן ריצה!
int* arr = (int*)malloc(size * sizeof(int));
```

**// ואם צריך להגדיל?**

```
int* new_arr = (int*)malloc((size + 10) * sizeof(int));
// מעתיקים את התוכן הישן
//משחררים את המערך הישן
// ויש לנו מערך גדול יותר!
```

# מתי נשתמש בכל אחד? 🤔

- **הקצאה רגילה (סטטית):**
  1. כשאנחנו יודעים בדיוק את **הגודל מראש**
  2. כשהגודל קטן וקבוע
  3. כשאנחנו **לא צריכים לשנות את הגודל**
- **הקצאה דינמית:**
  1. כשהגודל **תלוי בקלט מהמשתמש**
  2. כשאנחנו צריכים מערכים גדולים מאוד
  3. כשאנחנו רוצים **לשנות את הגודל תוך כדי ריצה**
  4. כשאנחנו רוצים **לחסוך בזיכרון** ולקחת רק מה שצריך



99

```c
#include <stdio.h>

int main()
{

    int size;
    printf("enter the size you want: \n");
    scanf("%d",&size);

    int arr[size];

    return 0;
}
```



100

## אתחול מערכים – מערך דינמי

הקצאת מערך דינמי

```
int* arr = (int*)malloc(5 * sizeof(int));

// אתחול ערכים בלולאה
for(int i = 0; i < 5; i++) {
   arr[i] = i + 1;
}
```







## טיפים חשובים! 💡

1. תמיד תאתחלו מצביעים! אל תשאירו אותם לא מאותחלים
2. בדקו תמיד שהקצאה דינמית הצליחה
3. אם אין צורך במצביע כרגע, אתחלו ל-NULL
4. שחררו זיכרון שהוקצה דינמית כשסיימתם להשתמש בו

# דוגמה מסכמת

```
// אתחול משתנה רגיל
 int num = 42;

// אתחול מצביע לכתובת של משתנה
 int* ptr1 = &num;

// אתחול מצביע להקצאה דינמית
 int* ptr2 = (int*)malloc(sizeof(int));
if(ptr2 != NULL) {
   *ptr2 = 100;
}
```

```
// אתחול מערך
    int arr[3] = {1, 2, 3};

   // הדפסת ערכים
    printf("num = %d\n", num);       // 42
   printf("*ptr1 = %d\n", *ptr1);    // 42
   printf("*ptr2 = %d\n", *ptr2);    // 100
   printf("arr[0] = %d\n", arr[0]);  // 1

   // שחרור זיכרון
    free(ptr2);
   return 0;
}
```






מעבר לשאלת בגרות בנושא

## פרק 6: מצביעים וכתובות

**יעדים**

היכרות עם הדרך בה נשמרים טיפוסים (char, short int, int, long, float, double) שונים בזיכרון המחשב וכן מערכים של טיפוסים שונים. הכרות עם מושגים הכתובת, מצביע לכתובת, תוכן הכתובת, מצביעים לטיפוסי משתנים שונים ומצביעים לפונקציות.

**תכנים**

1. ✓ הגדרת מצביעים מטיפוסים שונים
2. ✓ אופרטור הכתובת & , ותוכן הכתובת *
3. ✓ הקשר בין מצביעים למערכים ולחישוב האינדקסים של מערכים מטיפוסים שונים
4. ✓ אריתמטיקה של מצביעים (בהקשר של טיפוס המצביע)
5. ✓ מצביעים למערכים ומצביעים למצביעים
6. ✓ העברת פרמטרים by value, by reference והדגמה של פונקציית swap
7. ✓ אתחולים של מצביעים ושל מערכים
8. שימוש בהקצאות זיכרון דינמיות alloc, malloc, free, sizeof
9. ארגומנטים של שורת פקודה argc, argv
10. מצביעים לפונקציות: הגדרה, שימוש כולל העברת פרמטרים וקבלת ערך מוחזר
11. מערך של מצביעים לפונקציות עבור פניה מהירה
12. מנגנון callback לקבלת התראה על מאורעות

**סיכום שעות ההוראה: 18 שעות עיוניות ו- 8 שעות התנסותיות. סה"כ 26 שעות.**



105

# *פרק 6- מצביעים וכתובות:*

# 8. שימוש בהקצאות זיכרון דינאמיות

# alloc,malloc,free,sizeof





106

# sizeof- המפתח להקצאת זיכרון נכונה! 📏

- **מה זה sizeof?**
- אופרטור (לא פונקציה!) שמחזיר את גודל הזיכרון בבתים
- חיוני להקצאת זיכרון נכונה
- הגודל תלוי במערכת ההפעלה והמחשב

- **<u>דוגמאות</u>**

```
sizeof(char)    // 1 בית
sizeof(int)     // בדרך כלל 4 בתים
sizeof(double)  // בדרך כלל 8 בתים

int arr[10];
sizeof(arr)     // 40 בתים (4 * 10)
```






# malloc - Memory ALLOCation 🏗️

- **מה עושה malloc?**
- מקצה זיכרון בגודל שאנחנו מבקשים
- מחזיר מצביע לזיכרון שהוקצה
- אם אין מספיק זיכרון, מחזיר NULL

- **<u>דוגמאות</u>**

```
// הקצאת מספר שלם אחד
int* num = (int*)malloc(sizeof(int));

// הקצאת מערך של 5 מספרים
int* arr = (int*)malloc(5 * sizeof(int));

// הקצאת מחרוזת
char* str = (char*)malloc(50 * sizeof(char));
```

# ✨ calloc - Clear ALLOCation

- **מה מיוחד בcalloc?**
- מקצה זיכרון כמו malloc
- מאפס את כל הזיכרון אוטומטית (ממלא ב-0)
- מקבל שני פרמטרים: כמות איברים וגודל כל איבר

- **<u>דוגמאות</u>**

```
// הקצאת מערך של 10 מספרים (מאופס)
int* arr = (int*)calloc(10, sizeof(int));

// השוואה בין calloc- לmalloc:
int* arr1 = (int*)malloc(5 * sizeof(int));  // ערכים אקראיים
int* arr2 = (int*)calloc(5, sizeof(int));   // כל הערכים 0
```







# לשים לב.....

- במערכים גדולים, האיפוס של calloc יכול לקחת זמן משמעותי!
- השתמש ב- malloc כברירת מחדל
- השתמש ב- callocרק כשאתה באמת צריך איפוס אוטומטי
- תמיד חשוב על היעילות - אל תאפס אם אין צורך!

# free-שחרור הזיכרון! 

**למה חשוב לשחרר זיכרון?**
- זיכרון שהוקצה עם malloc/calloc לא משתחרר אוטומטית
- אי שחרור זיכרון גורם ל"דליפת זיכרון" (Memory Leak)
- התוכנית יכולה להתמלא ולקרוס!

**כללים חשובים:**
1. לכל malloc/calloc חייב להיות free
2. אסור לשחרר זיכרון פעמיים
3. אחרי free, כדאי להציב NULL במצביע



# טעויות נפוצות – להיזהר!

**שגיאה: שחרור כפול**
```
int* ptr = (int*)malloc(sizeof(int));
free(ptr);
free(ptr);  // שגיאה!
```

**שגיאה: שימוש אחרי שחרור**
```
int* ptr = (int*)malloc(sizeof(int));
free(ptr);
*ptr = 5;  // שגיאה!
```

**נכון:**
```
int* ptr = (int*)malloc(sizeof(int));
free(ptr);
ptr = NULL;  // מונע שימוש בטעות
```

# טיפים חשובים! 💡

- תמיד בדקו החזרה מ-malloc/calloc:

```
int* ptr = (int*)malloc(sizeof(int));
if(ptr == NULL) {
   // טיפול בשגיאה
    return -1;
}
```

- השתמשו ב- calloc כשצריך אתחול.







# טיפים חשובים! 💡

- שחרור זיכרון בסדר הנכון **<u>במערך דו-ממדי:</u>**

```
for(int i = 0; i < rows; i++) {
   free(matrix[i]);  // קודם שחררו את השורות
}
free(matrix);        // ואז את המערך עצמו
```

# סיכום

**זכרו:**

**1. תמיד לבדוק החזרה מ-malloc/calloc**

**2. תמיד לשחרר זיכרון**

**3. להיזהר משחרור כפול**

**4. להציב NULL אחרי שחרו**






## פרק 6: מצביעים וכתובות

**יעדים**

היכרות עם הדרך בה נשמרים טיפוסים (char, short int, int, long, float, double) שונים בזיכרון המחשב וכן מערכים של טיפוסים שונים. הכרות עם מושגים הכתובת, מצביע לכתובת, תוכן הכתובת, מצביעים לטיפוסי משתנים שונים ומצביעים לפונקציות.

**תכנים**

1. ✓ הגדרת מצביעים מטיפוסים שונים
2. ✓ אופרטור הכתובת & , ותוכן הכתובת *
3. ✓ הקשר בין מצביעים למערכים ולחישוב האינדקסים של מערכים מטיפוסים שונים
4. ✓ אריתמטיקה של מצביעים (בהקשר של טיפוס המצביע)
5. ✓ מצביעים למערכים ומצביעים למצביעים
6. ✓ העברת פרמטרים by value, by reference והדגמה של פונקציית swap
7. ✓ אתחולים של מצביעים ושל מערכים
8. ✓ שימוש בהקצאות זיכרון דינמיות alloc, malloc, free, sizeof
9. ארגומנטים של שורת פקודה argc, argv
10. מצביעים לפונקציות: הגדרה, שימוש כולל העברת פרמטרים וקבלת ערך מוחזר
11. מערך של מצביעים לפונקציות עבור פניה מהירה
12. מנגנון callback לקבלת התראה על מאורעות

**סיכום שעות ההוראה: 18 שעות עיוניות ו- 8 שעות התנסותיות. סה"כ 26 שעות.**

## *פרק 6- מצביעים וכתובות:*

# 9. ארגומנטים של שורת פקודה
# argc & argv







## argv - argc-הכרות

**<u>מה זה בכלל?</u>**

כשאנחנו מפעילים תוכנית במחשב ,לפעמים אנחנו רוצים להעביר לה מידע כבר בהתחלה.

argc -מספר שאומר לנו כמה דברים העברנו לתוכנית

argv- מערך שמכיל את כל הדברים שהעברנו

**<u>דוגמה פשוטה:</u>**

נניח שאנחנו מריצים תוכנית בשם program.exe עם המילים "שלום "ו"עולם:"

program.exe שלום עולם

argc- יהיה 3 : שם התוכנית +שתי המילים

argv- יכיל : program.exe , שלום ,עולם

## דוגמת קוד ראשונה

```
#include <stdio.h>
int main(int argc,  char* argv[]) {
   printf(" קיבלתי %d דברים!\n", argc);
   printf("הנה מה שקיבלתי:\n");

   for(int i = 0; i < argc; i++)
      printf("%d. %s\n", i, argv[i]);

   return 0;
}
```

119

## מצביעים למערכים -תזכורת

מה זה מצביע למערך?
מצביע למערך הוא כמו חץ שמצביע לתיבה הראשונה במערך שלנו.
אם יש לנו מערך של מספרים ,המצביע יצביע למספר הראשון.

120

# דוגמת קוד למצביעים

```
#include <stdio.h>

int main() {
   int numbers[] = {10, 20, 30, 40, 50};
   int* po = numbers;  // המצביע מצביע למספר הראשון

   printf("המספר הראשון: %d\n", * po);
   printf("המספר השני: %d\n", *(po + 1));
   printf("המספר השלישי: %d\n", po[2]);

   return 0;
}
```

121

# argv כמערך מצביעים

איך argv עובד?
argv הוא מערך של מצביעים למחרוזות .
כל מחרוזת היא המילה שהעברנו לתוכנית.

דוגמה:

```
#include <stdio.h>

int main(int argc, char* argv[]) {
   printf("המילה הראשונה היא: %s\n", argv[0]);
   printf("האות הראשונה במילה הראשונה היא: %c\n", argv[0][0]);
   return 0;
}
```

122

// אם נריץ את התוכנית כך:

```
/. //program hello world

argv[0] = "./program"   // המחרוזת הראשונה
argv[1] = "hello"       // המחרוזת השנייה
argv[2] = "world"       // המחרוזת השלישית

argv[0][0] = '.'     // התו הראשון במחרוזת הראשונה
argv[1][0] = 'h'     // התו הראשון במחרוזת השנייה
argv[2][0] = 'w'     // התו הראשון במחרוזת השלישית
```

123

# מצביע לפונקציה ו - callback-הכרות

מה זה בכלל?

-כמו שיש לנו מצביע למספר או למחרוזת ,אפשר גם להצביע לפונקציה!

-זה שימושי כשאנחנו רוצים לתת לתוכנית שלנו לבחור איזו פונקציה להפעיל

124

# תשפ"ג–2023
# שאלה 8





125

## תשפ"ג - 2023

**שאלה 8**

להלן קטע מתוכנית הכתובה בשפת C:

```
1. float avg (int * arr , int size )
2. {
3. int j,sum=0;
4.   for (j=0;j<size;j++)
5. {
6. sum+=arr[j];
7. }
8. return( ( float ) sum/size);
9. }
```

**(5 נק')** **א.** הסבירו את התהליך המתבצע על־ידי הפונקצייה avg.

**(5 נק')** **ב.** מה היה קורה אילו היו מסירים את הסימונים { ו־} משורות 5 ו־7 ?

**(5 נק')** **ג.** הסבירו את משמעות הפרמטרים המועברים לפונקצייה avg וההכרזה אליה.

**(5 נק')** **ד.** הסבירו את המושג CASTING וציינו היכן הוא בא לידי ביטוי בתוכנית.

**(5 נק')** **ה.** הסירו את הביטוי (float) משורה 8 . הסבירו את משמעות השינוי.



126

# תשפ"ב–2022
# שאלה 5





127

## תשפ"ב -2022

**(9 נק')** **א.** הסבירו את ההוראות שבשורות 3 , 9 , 18 .

**(6 נק')** **ב.** הפונקצייה bitup ממומשת בשורה מספר 12 .

(3 נק') 1. הסבירו מה הפונקצייה מקבלת ומה הפונקצייה מחזירה

(3 נק') 2. הסבירו מה הפונקצייה מבצעת.

**(5 נק')** **ג.** מהו תפקידה של התוכנית?

**(5 נק')** **ד.** מה יוצג על צג המחשב בסיום הרצת התוכנית?

```
1.  #include <stdio.h>
2.  void bitup(int *num, int bit);
3.  void bits_up (int *num, int frombit, int tobit);
4.  void main(){
5.      int num=1;
6.      bits_up(&num, 1 ,2);
7.      printf("%d\n" , num);
8.      num=1;
9.      bits_up(&num, 1, 3);
10.     printf("%d\n", num);
11. }
12. void bitup(int *num, int bit) {
13.     (*num) |= (1<< bit);
14. }
15. void bits_up(int *num, int frombit, int tobit) {
16.     int i;
17.     for (i=frombit; i<=tobit; i++)
18.         bitup(num, i);
19. }
```



128


02 ינואר 25
מצגת זו נוצרה על ידי: אינג' ברקת כהן.
cohen.bareket@gmail.com
129

129