

מבוא לתכנות מערכות

הרצאה 5

מערכים דו-מימדיים, הקצאה דינאמית בשפת C

מבוסס על הרצאות של אוניברסיטת חיפה, ומבוא למדעי המחשב, אורט בראודה, תעו"ן ותכנה.

מערך דו-מימדי (מטריצה)

כמו מערך חד-מימדי, זהו מבנה נתונים שכל הנתונים בו מאותו הסוג, מופיעים תחת שם משתנה יחיד ולכל נתון מוגדר מיקום.

מסתכלים על סידור איברים במטריצה כבנוי בצורת טבלה עם **שורות ועמודות**.

דוגמאות לנתונים שמתאים לארגן במטריצה:

- לוחות משחק (דמקה, איקס-עיגול)
- מפת מרחקים בין יישובים בארץ.

הגדרת מערך דו-מימדי

```
int ar[4][5];
```

	עמודה 0	עמודה 1	עמודה 2	עמודה 3	עמודה 4
שורה 0	53	79	-23	21	0
שורה 1	11	37	-45	17	32
שורה 2	24	36	34	24	3
שורה 3	-23	8	43	65	56

הגדרת מערך דו-מימדי

```
int ar[4][5];
```

	עמודה 0	עמודה 1	עמודה 2	עמודה 3	עמודה 4
שורה 0	ar[0][0] 53	ar[0][1] 79	ar[0][2] -23	ar[0][3] 21	ar[0][4] 0
שורה 1	ar[1][0] 11	ar[1][1] 37	ar[1][2] -45	ar[1][3] 17	ar[1][4] 32
שורה 2	ar[2][0] 24	ar[2][1] 36	ar[2][2] 34	ar[2][3] 24	ar[2][4] 3
שורה 3	ar[3][0] -23	ar[3][1] 8	ar[3][2] 43	ar[3][3] 65	ar[3][4] 56

אינדקס ימני - עמודות אינדקס שמאלי - שורות

איתחול של מערכים דו-מימדיים

ההוראות הבאות הן שקולות:

```
int a[2][3] = {{1, 2, 3}, {4, 5, 6}};
```

```
int a[2][3] = {1, 2, 3, 4, 5, 6};
```

```
int a[ ][3] = {{1, 2, 3}, {4, 5, 6}};
```

- אם חסרים ערכי איתחול, מוצבים במקומם אפסים.
- אפשר שהסוגריים [] הראשונים יהיו ריקים.
- איתחול לאפס של כל האיברים במטריצה:

```
int a[3][4] = {0};
```

קליטת נתונים מהמשמש למטריצה

בעלת M שורות ו- N עמודות

```
int ar[M][N], i, j;
printf("Enter numbers: \n");
for (i=0; i<M; i++)
{
    for (j=0; j<N; j++)
    {
        printf ("row %d , column %d:", i, j);
        scanf ("%d", &ar[i][j]);
    }
}
```

פונקציה להדפסת ערכי מטריצה

```
void printArray(int a[][N], int m, int n)
{
    int i, j;
    for (i=0; i<m; i++)
    {
        for (j=0; j<n; j++)
            printf("%d ", a[i][j]);
        printf("\n");
    }
}
```

מעבר על השורות

מעבר על העמודות

מעבירים מערך דו-מימדי לפונקציה בדומה למערך חד-מימדי. בכותרת הפונקציה מספר השורות (מימד הראשון במטריצה) אפשר להשאיר ריק.

דוגמא - האם כל עמודות המטריצה ממויינות בסדר עולה מלמעלה למטה?

```
#define N 4
```

```
int isSortedCol(int a[][N], int n, int col);
```

```
int AllColSorted(int a[][N], int n);
```

(מטריצה ריבועית)

```
int main ()
```

```
{
```

```
    int arr[N][N]={ { 1, 8, 9, 16},  
                     { 2, 7, 10, 18},  
                     { 3, 9, 11, 14},  
                     { 4, 5, 12, 13} };
```

```
    if (AllColSorted(arr, N))
```

```
        printf("All columns are sorted in ascending order\n");
```

```
    else
```

```
        printf ("Not all columns are sorted in ascending order\n");
```

```
    return 0; }
```


דוגמא - האם כל עמודות המטריצה ממויינות בסדר עולה?

```
int AllColSorted(int a[][N], int n)    /* check all columns */
{
    int i;
    for (i=0; i<n; i++)
        if (isSortedCol (a, n, i) == 0)
            return 0;
    return 1;
}

int isSortedCol(int a[][N], int n, int col)    /* check one column */
{
    int i;
    for(i=0; i<n-1; i++)
        if (a[i][col]>= a[i+1][col])
            return 0;
    return 1;
}
```

מצביע כללי – void*

- ב-C ניתן להגדיר מצביע כללי – מסוג void*.
- ב-C כל המצבעים בעלי אותו גודל.
- אם רוצים לפנות לערך המוצבע בעזרת מצביע מסוג כללי, יש להשתמש ב-casting.
- דוגמא :

```
void * p;
```

```
int a;
```

```
float b;
```

```
p=&a;
```

```
*(int*)p = 4;
```

```
p=&b;
```

```
*(float*)p=4.5
```

האם *p=4 יתקמפל? אם לא, למה?


האם *p=4.5 יתקמפל? אם לא, למה?


הקצאת זיכרון דינאמית

- לעיתים, תוך כדי ריצת התוכנית נרצה כי יוקצה עבורנו זיכרון, שאת גודלו אנו יודעים רק בזמן הריצה.
- C מעמיד לרשותנו מנגנון לקבלה ושחרור זיכרון במהלך התוכנית.
- כל הפקודות שמטפלות בזכרון נמצאות בספרייה **stdlib.h**
- הקצאת זיכרון על ידי הפקודות **malloc**, **calloc**
- שחרור זיכרון על ידי הפקודה **free**
- שינוי זכרון שהוקצה ע" הפקודה **realloc**.
- עבודה עם הקצאה דינאמית מחייבת שימוש במצביעים.
- **כל הזכרון שהוקצה דינאמית לא משתחרר לבד אלא יש לשחרר אותו בסיום השימוש בעזרת פקודה free!!!!**

הפקודה malloc

הגדרה: `void *malloc (size_t size)` 

הפקודה מקצה בלוק בגודל size בתים בזיכרון בזמן ריצת התכנית. 
אם הקצאת הזיכרון הצליחה הפונקציה מחזירה מצביע לתחילת הבלוק המוקצה. המצביע אשר הפונקציה מחזירה הוא מטיפוס `void*` ויש להשתמש ב- casting (המרה של המצביע לטיפוס הרצוי).

כאשר הקצאת הזיכרון לא מצליחה (אין מספיק זיכרון רציף פנוי בזמן הריצה), 
הפקודה מחזירה NULL. לכן, לאחר כל הקצאה דינאמית חובה לבדוק האם ההקצאה הצליחה - הפקודה החזירה ערך השונה מ-NULL.

דוגמא – הקצאה דינאמית למערך מספרים שלמים באורך n: 

```
int *arr;  
arr = (int*) malloc (n * sizeof (int));  
if(arr == NULL)
```

קודם כל, יש להגדיר מצביע מתאים

בדיקה האם ההקצאה הצליחה. אם לא, בדרך כלל מסיימים את ריצת התכנית

הפקודה free

- הגדרת הפקודה: **void free (void *block)**
- הפקודה משחררת את הבלוק המוקצה מן הזיכרון.
- חובה לשחרר את הזכרון שהוקצה!
- **דוגמא** – שיחרור המערך שהוקצה בדוגמא הקודמת: **free(arr);**

הפקודה calloc

- הגדרת הפקודה: **void *calloc(size_t nitems, size_t size)**
- מקצה זכרון בדומה לפקודה malloc. ההבדל בין שתי הפקודות הוא ש-calloc גם מאפסת את זכרון – מכניסה 0 לכל תא.
- **דוגמא** – הקצאה דינאמית למערך מספרים שלמים באורך n שכל תא מכיל 0:

```
int *arr;  
arr = (int*) calloc (n , sizeof (int));
```

הפקודה realloc

- שינויי הקצאות אפשר לעשות בעזרת הפקודה realloc על ידי קריאות מהצורה הבאה:

```
void* realloc(void *block , size_t size)
```

- כאשר block היא כתובת הזיכרון המוקצאת הנוכחית, ו-size הוא הגודל החדש המבוקש (בבתים). **תשימו לב – לא כמות בתים להוספה אלא הגודל החדש!**
- מערכת ההפעלה תנסה לראות האם אפשר לשנות את רצף הזיכרון הנוכחי לגודל המבוקש. אם הדבר אפשרי, הפונקציה תחזיר את כתובת הזיכרון של הרצף הנוכחי. אם הדבר אינו אפשרי, היא תבדוק האם יש רצף אחר מתאים בזיכרון. אם היא הצליחה, היא תעתיק את תוכן הרצף הנוכחי לרצף החדש, תשחרר את הרצף הנוכחי, ותחזיר את כתובת הרצף החדש. אם אין רצף אחר מתאים בזיכרון, היא לא תשנה כלום בזיכרון (ובפרט, לא תשחרר את הרצף הנוכחי), ותחזיר NULL כדי לסמן שלא הצליחה.

הפקודה realloc

❖ דוגמא – שימוש נכון ב-realloc

```
int n=5, *p, *temp_p;  
p = (int*)malloc(n*sizeof(int));
```

הוספת תא אחד לסוף המערך

```
temp_p = (int*)realloc(p, (n+1)*sizeof(int));
```

```
if(temp_p !=NULL)
```

```
    p = temp_p;
```

```
else
```

```
{
```

```
    free(p);
```

```
    .....
```

אם הפקודה הצליחה להוסיף תא לסוף המערך, יש להעתיק את הכתובת של המערך ממצביע זמני למצביע שהוא שם המערך.

אם הפקודה לא הצליחה להוסיף תא לסוף המערך, משחררים את המערך שהיה עד כה ומסיימים את ריצת התכנית.

תכנית דוגמא

/* malloc.c - a program demonstrating memory allocation */

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
int SetSize ();
```

```
int* CreateArray (int size);
```

```
void PrintArray(int a[], int size);
```

```
int main()
```

```
{
```

```
    int i, len, *array;
```

```
    len=SetSize();
```

```
    array=CreateArray(len);
```

```
    for (i=0; i< len; i++)
```

```
        array[i]=i;
```

```
    PrintArray(array, len);
```

```
    free(array);
```

```
    return 0;
```

```
}
```



תכנית דוגמא

```
void PrintArray(int a[], int size)
{
    int i;
    for (i=0; i< size; i++)
        printf("%d ",a[i]);
    printf("\n");
}
```

תכנית דוגמא

```
int SetSize ()
{
    int size;
    printf("Please enter array size\n");
    if (scanf("%d",&size)!=1)
    {
        printf("Error reading array size\n");
        exit (1);          /* in stdlib.h library*/
    }
    if (size<1)
    {
        printf("Array size too small\n");
        exit (1);
    }
    return size;
}
```

בדיקת קלט



תכנית דוגמא

```
int* CreateArray (int size)
{
    int *array;
    array=(int *) malloc(size*sizeof(int));
    if (array==NULL)
    {
        printf("Error in memory allocation\n");
        exit (1);
    }
    return array;
}
```

יש להחזיר את הכתובת של הזכרון שהוקצה כיוון ש-array זה
משתנה לוקאלי שהולך להשתחרר ביציאה מהפונקציה
אבל המערך עצמו לא משתחרר עד שלא נשתמש בפקודה free



מצביע כפול

משתנה שמכיל כתובת של מצביע אחר הוא מצביע כפול. במילים אחרת – מצביע למצביע לנתון. למשל, שם מערך מצביעים הוא משתנה מטיפוס מצביע כפול.

□ נראה דוגמא איך ניתן ליצור דינאמית מערך מצביעים.

□ דוגמא :

```
int **p;  
p = (int**) malloc(n*sizeof(int));
```

□ ואם רוצים להקצות מערך נוסף כך שתא הראשון במערך מצביעים יצביע עליו?

```
p[0] = (int*)malloc(5*sizeof(int));
```