



اساتید:  
دکترمرادی، دکتر هاشمی

عنوان:  
آرایه‌های پویا

نیمسال اول  
1402-03

### آرایه‌ی پویا<sup>۱</sup>:

آرایه‌هایی که تا به حال دیده‌اید و از آن‌ها استفاده کرده‌اید، آرایه‌های ایستا<sup>۲</sup> بوده‌اند. اگر یادتان باشد در تعریف این آرایه‌ها حتماً باید طول آن‌ها را با یک عدد ثابت مشخص می‌کردید. امروز می‌خواهیم با نوع دیگری از آرایه‌ها به نام آرایه‌ی پویا آشنا شویم. طول این آرایه‌ها در هنگام کامپایل نامشخص بوده و در هنگام اجرا تعیین می‌گردد. این اندازه را می‌توان بعداً در برنامه تغییر داد تا آرایه را بزرگ یا کوچک کند.

### دستور تخصیص حافظه (malloc)<sup>۳</sup>:

شما می‌توانید توسط تابع malloc که از توابع کتابخانه‌ی stdlib.h می‌باشد، از سیستم عامل درخواست کنید که مقدار مشخصی حافظه در heap گرفته و آن را در اختیار شما قرار دهد. نحوه‌ی استفاده از این تابع به صورت زیر است:

```
<type>* pointer = (<type>*)malloc(number*sizeof(<type>));
```

که <type> نوع داده‌ای است که می‌خواهید آرایه‌ای پویا از آن ایجاد کنید (مثلاً int) و number طول آرایه می‌باشد.

- حال به نکات زیر در مورد این تابع توجه کنید:

- 1- آرگومان تابع malloc مقدار حافظه درخواستی برحسب بایت می‌باشد.
- 2- sizeof از عملگرهای زبان C است که سائز هر type ای که به آن بدهید را بر حسب بایت برمی‌گرداند. چون سائز یک type (مثلاً int) در سیستم‌های مختلف ممکن است متفاوت باشد، بهتر است از عملگر sizeof استفاده کنید.
- 3- مقدار برگشتی تابع malloc در صورتی که تخصیص حافظه موفقیت آمیز باشد، اشاره‌گر به سر آرایه‌ی پویا خواهد بود و در غیر این صورت NULL است. لذا بعد از فراخوانی این تابع **حتماً** باید بررسی کنید که اگر مقدار بازگشتی NULL بود، ضمن دادن پیغام خطا به کاربر از برنامه خارج شوید.
- 4- همچنین مقدار برگشتی این تابع از جنس void\* (یعنی اشاره‌گری که می‌تواند از هر نوعی باشد و لزوماً قرار نیست نوع خاصی، مثلاً integer، داشته باشد) بوده و برای همین آن را به type مورد نظر cast کرده ایم.

<sup>1</sup> Dynamic array

<sup>2</sup> Static

<sup>3</sup> Memory allocation

**توجه:** یکی دیگر از راه‌های ایجاد آرایه‌های پویا، استفاده از تابع **calloc** می‌باشد. برای آشنایی با این تابع می‌توانید به این [لینک](#) و یا این [لینک](#) و یا سایت‌های مشابه مراجعه کنید.

**توجه:** در هر برنامه، هر حافظه‌ای را که توسط توابع **malloc** یا **calloc** می‌گیرید را **باید** در انتها توسط تابع **free(pointer)** آزاد کنید. در این [لینک](#) می‌توانید اطلاعات بیشتری در مورد این تابع کسب کنید.

برای درک نکات بالا، به برنامه زیر توجه کنید:

```
#include <stdio.h>
#include <stdlib.h>
int main() {
    int arr_size, i;
    int* dynamic_arr;
    printf("Enter the size of array:\n");
    scanf("%d", &arr_size);
    /* Requesting an integer array with capacity of arr_size elements.
     * On success dynamic_arr will be a pointer to the beginning of the array.
     * On failure dynamic_arr will be null. */
    dynamic_arr = (int*) malloc (arr_size * sizeof(int));
    if (dynamic_arr == NULL) {
        printf("Oops! Memory allocation failed.\n");
        exit(EXIT_FAILURE);
    }
    /* From now on you can work with the dynamic array just like static arrays! */
    printf("Enter %d numbers:\n", arr_size);
    for (i = 0; i < arr_size; i++)
        scanf ("%d", &dynamic_arr[i]);
    free(dynamic_arr); /* Do not forget to free the allocated memory! */
    return 0;
}
```

- همانطور که می‌بینید، پس از تخصیص صحیح حافظه، نحوه‌ی استفاده از آرایه‌های ایستا و پویا هیچ تفاوتی با هم ندارد. پس همان گونه که قبلاً با آرایه‌های ایستا کار می‌کردید، می‌توانید با آرایه‌های پویا نیز کار کنید.
- همانطور که پیش‌تر گفته شد، پس از تخصیص حافظه باید بررسی کنید که اگر مقدار بازگشتی **NULL** بود، ضمن دادن پیغام خطا به کاربر از برنامه خارج شوید. در این برنامه از **exit(EXIT\_FAILURE)** استفاده شده است که می‌توانید در این [لینک](#) در مورد تابع **exit** و در این [لینک](#) در مورد **EXIT\_FAILURE** بیشتر بخوانید!

## 1- انجام دهید!

قطعه کد زیر را در یک پروژه‌ی جدید کامپایل و اجرا کنید.

```
#include <stdio.h>
#include <stdlib.h>
int main() {
    int* p = (int*)malloc(10 * sizeof(int));
    int i;
    printf("P = 0x%p\n", p);
    printf("*****\n");
    for (i = 0; i < 10; i++){
        printf("p%d = %d\n", i, *(p+i));
    }
    free(p);
    printf("*****\n");
    printf("P = 0x%p\n", p);
    return 0;
}
```

آیا آدرس ذخیره شده در اشاره گر p همیشه (حتی بعد از استفاده از تابع free) ثابت است؟ چرا؟

پس از تخصیص حافظه توسط تابع malloc، مقادیر اولیه‌ی موجود در آرایه را توجیه نمایید.

**(امتیازی!):** حال تخصیص حافظه را توسط تابع calloc انجام داده و دوباره برنامه را اجرا کنید. مقادیر اولیه

موجود در آرایه چه تغییری کردند؟ چرا؟

**راهنمایی:** برای تخصیص حافظه توسط تابع calloc به صورت زیر عمل کنید:

```
int* p = (int*)calloc(10 , sizeof(int));
```

چرا در کد بالا، اگر اقدام به چاپ p[15] بکنیم، برنامه خطا نمی دهد؟

پاسخ سوالات بالا را با دستیاران آموزشی در میان بگذارید. (قسمت 1)

پیش از انجام قسمت بعدی، به بررسی یکی دیگر از توابع کاربردی برای تخصیص حافظه‌های پویا می‌پردازیم:

### تابع realloc<sup>۴</sup>:

از این تابع می‌توانید برای تغییر مقدار حافظه‌ای که قبلاً از سیستم گرفته بودید، استفاده کنید. تعریف این تابع به صورت زیر است:

```
<type>* pointer = (<type>*)realloc(pointer , number*sizeof(<type>));
```

این تابع به عنوان ورودی اشاره گر فعلی و اندازه‌ی جدید مورد نظر را گرفته و اشاره گر جدید را بر می‌گرداند.

<sup>4</sup> Re-allocation

## 2- انجام دهید! (Realloc)

اکنون قصد داریم برنامه‌ای بنویسیم که تا زمانی که کاربر عدد صفر را وارد نکرده است، اعدادی را از ورودی دریافت کرده و اعداد زوج را در یک آرایه ذخیره نموده و در انتها نمایش دهد. برنامه‌ی زیر به همین منظور نوشته شده است. جاهای خالی این برنامه را به طور مناسب تکمیل کنید.

```
#include <stdio.h>
#include <stdlib.h>
int main() {
    int size=1 , check=1 , i=0 , num;
    int* array=(int*)malloc(size*sizeof(int)); //creating a dynamic array
    while(1){ //infinite loop
        scanf("%d", &num);
        if(num!=0){
            if(num%2==0){
                ... = num; //saving the even number in the list
                size++;
                i++;
                array= ...; //extending the size of the array
            }
        }else{
            ...; //getting out of the loop
        }
    }
    printf("Here is the list of even numbers:\n");
    for(int i=0;i<size;i++){
        printf("%d\n", ...); //printing the list
    }
    free(...);
    return 0;
}
```

نتیجه را به دستیاران آموزشی نشان داده و روند اجرای برنامه را توضیح دهید.(قسمت 2) ✓

✚ قطعه کد زیر را بررسی کرده و سپس آنرا در یک پروژه‌ی جدید اجرا کنید.

```
#include <stdio.h>
#include <stdlib.h>
#define SIZE 4
void printer(int* array, int size){
    for(int i=0;i<size;i++)
        printf("%d ", array[i]);
    printf("\n");
}
int main() {
    int* array=(int*)malloc(SIZE*sizeof(int)); //creating a dynamic array
    for(int i=0;i<SIZE;i++)
        array[i]=(i+1);
    printer(array,SIZE);
    array=(int*)realloc(array,(SIZE+1)*sizeof(int));
    array[4]=5;
    printer(array , (SIZE+1));
    free(array);
    return 0;
}
```

حال عبارت زیر را جایگزین سطر مشخص شده کنید. چه اتفاقی افتاد؟

```
array=(int*)realloc(NULL,(SIZE+1)*sizeof(int));
```


✔ علت را برای دستیاران آموزشی را توضیح دهید.(قسمت 3)

### 3- فکر کنید! (Type Casting)

برنامه‌ی زیر را در یک پروژه‌ی جدید اجرا کنید و به سوالات زیر با توجه به شماره‌ی مشخص شده در برنامه پاسخ دهید.

```
#include <stdio.h>
#include <stdlib.h>
void main() {
    int i;
    char* s;
    int* p = (int*)malloc(10*sizeof(int));
    for(i=0;i<10;i++)
        p[i] = i+48;
    s=(char*)p; /*1*/
    /*2*/
    for(i=0; i<40; i++) /*3*/
        printf("%c", s[i]); /*4*/
    printf("\n");
    printf("p[0] is %d\n",*p);
    free(p);
    free(s); /*5*/
}
```

- 1- در این سطر چه اتفاقی می‌افتد؟
- 2- حال در این سطر، ابتدا مقدار `s[4]` و سپس مقدار `s[5]` را چاپ کنید. چه اتفاقی افتاد؟ (برای بررسی بیشتر، مقدار دیگری از این آرایه را چاپ و مشاهده نمایید.) علت چیست؟
- 3- به محدوده‌ی `i` در این سطر توجه کنید. چرا این حدود برای `i` انتخاب شده است؟
- 4- نتیجه‌ی نمایش داده شده را با توجه به مقدار `p[0]` که در سطر بعدی چاپ شده است، تحلیل کنید.
- 5- در این سطر چه اتفاقی می‌افتد؟

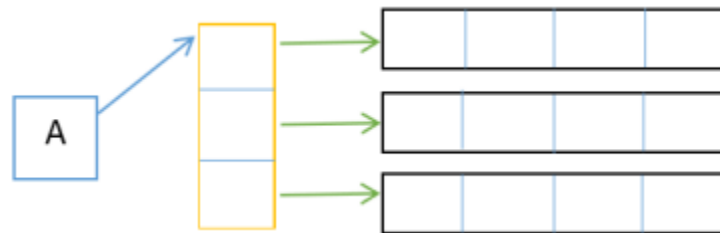
 نتایج را به دستیاران آموزشی توضیح دهید. (قسمت 4)

## آرایه‌ی پویای دو بعدی:

همانطور که در آزمایشگاه‌های گذشته بیان شد، آرایه‌ی دوبعدی چیزی نیست جز آرایه‌ای از آرایه‌های یک بعدی. برای مثال `int a[3][4]`، آرایه‌ای 3 تایی از آرایه‌هایی به طول 4 می‌باشد. به شکل زیر توجه کنید:

	Column 0	Column 1	Column 2	Column 3
Row 0	<code>a[0][0]</code>	<code>a[0][1]</code>	<code>a[0][2]</code>	<code>a[0][3]</code>
Row 1	<code>a[1][0]</code>	<code>a[1][1]</code>	<code>a[1][2]</code>	<code>a[1][3]</code>
Row 2	<code>a[2][0]</code>	<code>a[2][1]</code>	<code>a[2][2]</code>	<code>a[2][3]</code>

اکنون قصد داریم همین آرایه‌ها را به شکل پویا درست کنیم. بنابراین کافیت در ابتدا آرایه‌ای از جنس `int**` به طول 3 بسازیم که **A** به ابتدای آن اشاره کند. سپس هر یک از خانه‌های آن را برابر با ابتدای یک آرایه‌ی 4 تایی قرار دهیم. حال یک آرایه‌ی 2 بعدی پویا ساخته‌ایم: !



#### 4- انجام دهید!

در برنامه‌ی زیر قصد داریم آرایه‌ای دوبعدی با تعداد سطرهای **row** و تعداد ستونهای **col** ایجاد کرده و خانه‌های آنرا به ترتیب با اعداد طبیعی پر کرده و سپس آنرا در خروجی نمایش دهیم. جاهای خالی را تکمیل کنید و نتیجه را به

دستیاران آموزشی نشان دهید. (قسمت 5)

```
#include <stdio.h>
#include <stdlib.h>
#define ZERO 0
void printer(int** array_of_int, int size1 , int size2) {
    for (int i = 0; i < size1; i++){
        for (int j = 0; j < size2; j++){
            printf("%d ", array_of_int[i][j]);
            printf("\n");
        }
    }
}
void initializer(int** array_of_int, int size1, int size2) {
    int num= 1;
    for (int i = ZERO; i < size1; i++){
        for (int j = ZERO; j < size2; j++){
            array_of_int[i][j] = num;
            num++;
        }
    }
}
int main() {
    int row, col, i;
    int** A;
    printf("Enter row and column:\n");
    scanf("%d %d", &row, &col);
    A=(...) malloc(...*sizeof(...)); /* 1. Complete this instruction */
    if (A == NULL)
        exit(EXIT_FAILURE);
    for(i = 0; i < row ; i++) {
        A[i]=(...) malloc(...*sizeof(...)); /* 2. Complete this instruction */
        if (A[i] == NULL)
            exit(EXIT_FAILURE);
    } /* Now you have a 2D integer array */
    ...; /* initializing the array */
    ...; /* printing the array */
    /*Don't forget to free the allocated memory when you don't need it any more*/
    for (i = 0; i < row; i++)
        free(A[i]);
    free(A);
    return 0;}
```



در کد بالا، به فرآیند **free** کردن دقت کنید. علت انجام این عمل به صورت بالا را برای دستیاران آموزشی توضیح دهید. (قسمت 6)

#### 4- بررسی کنید!

حافظه‌های گرفته شده از سیستم هنگام اجرای برنامه حتما باید در انتهای برنامه آزاد شوند. در این قسمت مشاهده می‌کنیم که در صورتی حافظه‌ی گرفته شده آزاد نشود ممکن است چه مشکلاتی برای سیستم ایجاد شود. برنامه‌ی زیر را در یک پروژه‌ی جدید اجرا کنید.

```
#include<stdio.h>
#include<stdlib.h>
void main() {
    int *p = NULL;
    int i = 500000;
    for(int j=0;j<40;j++) {
        p = realloc(p, i * sizeof(int));/*put breakpoint here*/
        i+=500000;
    }
    free(p);
}
```

- 1- با استفاده از کلیدهای ترکیبی **ctrl+shift+esc** پنجره‌ی **task manager** را باز کنید.
- 2- در سربرگ **processes** برنامه‌ی اجرایی **your\_project\_name.exe** را یافته و مقدار **memory** مورد استفاده آن را مشاهده کنید.

- 3- حال مقدار نهایی متغیر **j** را در دستور **for** به عدد 80 تغییر دهید. **(for(int j=0;j<80;j++))**
- 4- مقدار **memory** گرفته شده توسط برنامه را مجدداً مشاهده کنید. چه نتیجه‌ای می‌گیرید؟ نتیجه را با

دستیاران آموزشی در میان بگذارید.(قسمت 7)

موفق باشید.