



به نام خدا

دانشکده‌ی مهندسی برق و کامپیوتر دانشکده فنی دانشگاه تهران

مبانی کامپیوتر و برنامه‌نویسی



استاد: دکتر مرادی

عنوان:  
آزمایشگاه ششم (آرایه‌های پویا و ساختمان‌ها)

نیمسال دوم  
۹۸-۹۹

## 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 آرایه پویا آشنا شویم. طول این آرایه‌ها در هنگام کامپایل نامشخص بوده و در هنگام اجرا تعیین می‌گردد.

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

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>));
```

<type>: نوع داده‌ای که می‌خواهید آرایه‌ای پویا از آن داشته باشید.

number: طول آرایه‌ای که می‌خواهید.

حال به نکات زیر **توجه کنید**:

۱. آرگومان تابع malloc مقدار حافظه درخواستی بر حسب بایت می‌باشد.
۲. `sizeof` از عملگرهای زبان C است که سایز هر `type`‌ای که به آن بدهید را بر حسب بایت برمی‌گرداند. چون سایز یک `type` (مثلاً `int`) در سیستم‌های مختلف ممکن است متفاوت باشد، بهتر است از عملگر `sizeof` استفاده کنید.
۳. مقدار برگشتی تابع malloc در صورتی که تخصیص حافظه موفقیت آمیز باشد، اشاره‌گر به سرآرایه‌ی 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.
۴. 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.
۵. هر حافظه‌ای که توسط تابع malloc می‌گیرید را در پایان **باید** توسط تابع `free(pointer)` آزاد کنید.

<sup>1</sup> dynamic

<sup>2</sup> static

<sup>3</sup> memory allocation

۶. نحوه‌ی استفاده از آرایه‌های ایستا و پویا هیچ تفاوتی با هم ندارد. پس همان گونه که قبلاً با آرایه های ایستا کار می‌کردید، می‌توانید با آرایه های پویا نیز کار 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;
}
```

## ۱. انجام دهید ←

یکی از توابع کاربردی برای تخصیص حافظه‌های پویا تابع `realloc` است. 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 اشاره‌گر فعلی و اندازه‌ی جدید مورد نظر را گرفته و اشاره‌گر جدید را بر می‌گرداند. برای آشنایی با رفتار این تابع، قطعه کدهای زیر را اجرا کنید. تفاوت نتیجه‌ی این دو قطعه کد در چیست؟

قطعه کد اول:

```
#include <stdio.h>
#include <stdlib.h>
#define INITIAL_ARRAY_SIZE 4
#define FINAL_ARRAY_SIZE 5
#define ZERO 0
#define ONE 1
#define FIVE 5

void print_array(int* array_of_int, int size) {
    for (int i = 0; i < size; i++)
        printf("%d ", array_of_int[i]);
    printf("\n");
}

void initislizing_the_array(int* array_of_int, int size) {
    for (int i = ZERO; i < size; i++)
        array_of_int[i] = ZERO;
}

int main(int argv, char** argc) {
    int* array_of_int = (int*)malloc(INITIAL_ARRAY_SIZE *
sizeof(int));
    initislizing_the_array(array_of_int, INITIAL_ARRAY_SIZE);
    print_array(array_of_int, INITIAL_ARRAY_SIZE);
    array_of_int = (int*)realloc(array_of_int, FINAL_ARRAY_SIZE *
sizeof(int));
    initislizing_the_array(array_of_int, FINAL_ARRAY_SIZE);
    array_of_int[FINAL_ARRAY_SIZE - ONE] = FIVE;
    print_array(array_of_int, FINAL_ARRAY_SIZE);
    return 0;
}
```

قطعه کد دوم:

```
#include <stdio.h>
#include <stdlib.h>
#define INITIAL_ARRAY_SIZE 4
#define FINAL_ARRAY_SIZE 5
#define ZERO 0
#define ONE 1
#define FIVE 5
```

```

void print_array(int* array_of_int, int size) {
    for (int i = 0; i < size; i++)
        printf("%d ", array_of_int[i]);
    printf("\n");
}

void initislizing_the_array(int* array_of_int, int size) {
    for (int i = ZERO; i < size; i++)
        array_of_int[i] = ZERO;
}

int main(int argv, char** argc) {
    int* array_of_int = (int*)malloc(INITIAL_ARRAY_SIZE *
sizeof(int));
    initislizing_the_array(array_of_int, INITIAL_ARRAY_SIZE);
    print_array(array_of_int, INITIAL_ARRAY_SIZE);
    array_of_int = (int*)realloc(NULL, FINAL_ARRAY_SIZE *
sizeof(int));
    array_of_int[FINAL_ARRAY_SIZE - ONE] = FIVE;
    print_array(array_of_int, FINAL_ARRAY_SIZE);
    return 0;
}

```

**قسمت ۱:** نتایج به دست آمده و همچنین یافته‌های خود را در کادر زیر بنویسید.

در قطعه کد اول آرایه پویایی را با طول ۴ میسازیم. آنرا با استفاده از تابع مقداردهی می‌کنیم. و سپس پرینت می‌کنیم.

بعداً با استفاده از دستور `realloc` طول آنرا به ۵ تغییر می‌دهیم. و مجدداً مراحل قبل را تکرار می‌کنیم.

البته یک نکته در اینجا حائز اهمیت است و آن این است که میتوان دیگر آرایه را مقداردهی نکرد زیرا مقادیر آرایه تا ۴ تا خانه اول در حافظه وجود دارد و تنها نیاز است تا خانه ۵ام را مقداردهی کنیم.

تنها تفاوت قطعه کد اول با دوم هم همین مورد است. در قطعه کد دوم هنگام استفاده از دستور `realloc` بجای استفاده از نام خود آرایه از `NULL` استفاده شده به همین علت تمام مقادیر قبلی پاک شده و نیاز به مقداردهی مجدد دارد.

## ۲. انجام دهید (Free)



(۱) هنگام کار با تخصیص حافظه‌ی پویا به دو نکته باید توجه نمود :

۱. لزوم آزاد کردن حافظه‌های گرفته شده

۲. عدم استفاده از حافظه‌ی آزاد شده

(۲) قطعه کد زیر را اجرا کنید.

```
int main() {
    int* p = (int*)malloc(10 * sizeof(int));
    int i;
    printf("P = 0x%p\n", p);
    for (i = 0; i < 10; i++)
    {
        p[i] = i;
    }
    free(p);
    printf("P = 0x%p\n", p);
    printf("P[0] = %d", *p);
    return 0;
}
```

**قسمت ۲:** قطعه کد داده شده را اجرا کنید، نتایج به دست آمده و یافته‌های خود را در کادر زیر بنویسید.

- پس از آزادسازی حافظه‌ی گرفته شده دقیقاً چه اتفاقی می‌افتد؟ خروجی‌ها را توجیه کنید.
- فکر کنید : چرا اگر در کد بالا اقدام به چاپ `p[15]` کنیم برنامه خطا نمی‌دهد؟ دلیل خود را در کادر زیر بنویسید.

خروجی برنامه نشان دهنده این است که بعد از آزادسازی حافظه ، آدرس خود آرایه تغییر نمی‌کند ولی محتوای آن خالی می‌شود و مقداری را ، خود کامپایلر به آن اختصاص می‌دهد.

درسومین پرینت ما در واقع داریم به اولین خانه آرایه اشاره می‌کنیم و در هنگام پرینت محتوای آن پرینت می‌شود. ولی چون قبلاً محتوای آنرا خالی کردیم ، خود کامپایلر مقداری را جایگزین آن کرده و آن را پرینت می‌کند.

اگر ما اقدام به چاپ خانه ۱۵ام کنیم برنامه خطایی نمی‌دهد ولی مقداری را از توی حافظه که در ۶ خانه بعد از آرایه قرار دارد ، برمی‌دارد و پرینت می‌کند.

### ← ۳. انجام دهید (Memory Leak)

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.

(۱) کد زیر را در حالت debug و با قرار دادن breakpoint در محل ذکر شده اجرا کنید.

```
void main() {  
    int *p = NULL;  
    int i = 500000;  
    while (1) {  
        p = realloc(p, i * sizeof(int)); /*put breakpoint here*/  
        i+=500000;  
    }  
    free(p);  
    return 0;  
}
```

(۲) با استفاده از کلیدهای ctrl+shift+esc پنجره‌ی task manager را باز نمایید.

(۳) سربرگ Processes را انتخاب کنید.

(۴) برنامه‌ی your\_project\_name.exe را پیدا کنید و مقدار memory مورد استفاده‌ی آن را مشاهده کنید.

(۵) با فشردن دکمه‌ی f5 به تعداد ۱۰ بار حلقه‌ی while را اجرا کنید.

(۶) مقدار memory گرفته شده توسط برنامه را مجددا مشاهده کنید.

(۷) چه نتیجه‌ای می‌گیرید؟

**قسمت ۳:** موارد خواسته شده را انجام دهید. نتایج به دست آمده و یافته‌های خود را در کادر زیر بنویسید.

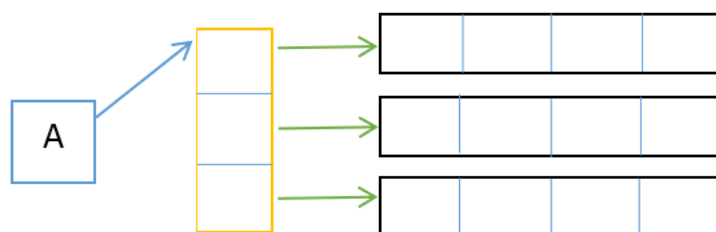
فکر کنید : آیا آدرس p که به ابتدای حافظه‌ی مورد نظر ما اشاره می‌کند همیشه یکسان است؟ نظر خود را در کادر زیر بنویسید.

کاملا مشخص است که اگر حافظه رو free نکنیم با هر بار realloc کردن آن ، حافظه استفاده شده از سیستم به طور تصاعدی افزایش میابد. پس لازم است حتما حافظه را free کنیم.  
با هر بار realloc کردن ، آدرس p عوض میشود.

## آرایه پویای دو بعدی (!):

قبلاً با آرایه های چند بعدی هم کار کرده بودیم. حال می خواهیم همان آرایه ها را نیز به شکل پویا درست کنیم. یک آرایه ی دوبعدی، عملاً آرایه ای از آرایه های یک بعدی است. مثلاً `int a[3][4]` یک آرایه ی ۳ تایی از آرایه هایی به طول ۴ می باشد. شکل زیر را نگاه کنید:

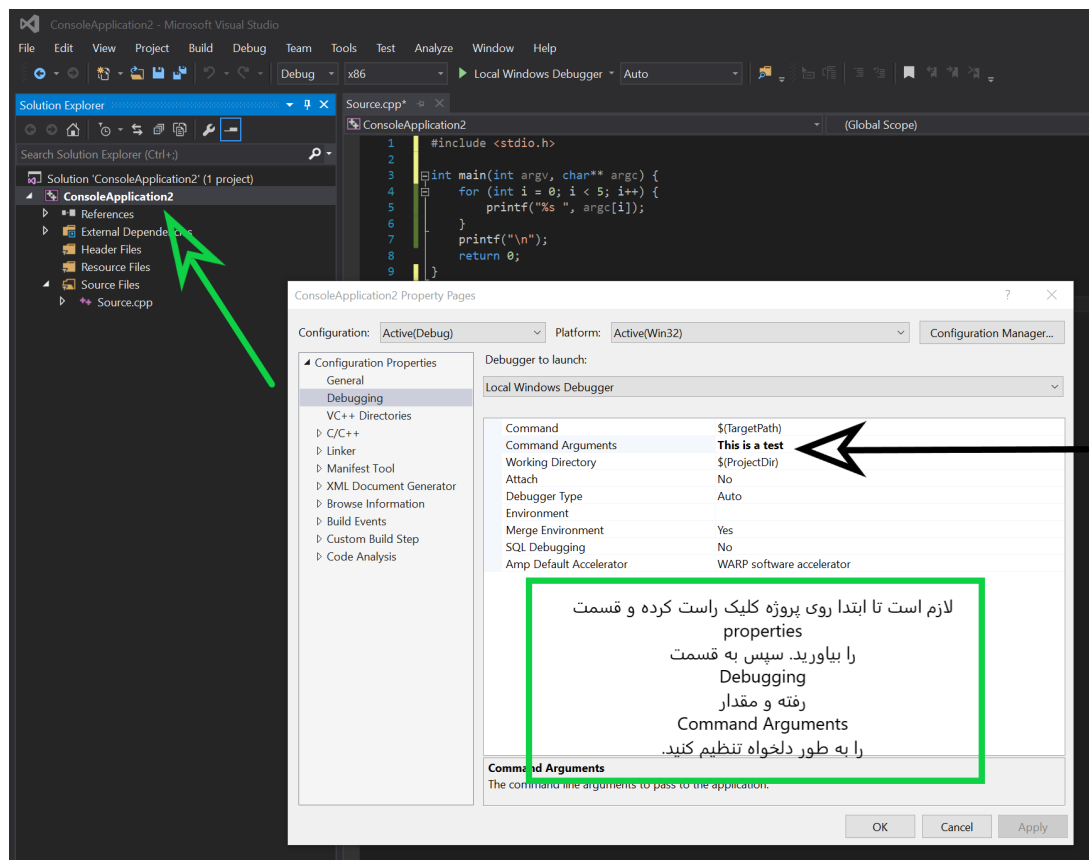
	Column 0	Column 1	Column 2	Column 3
Row 0	<code>a[0][0]</code>	<code>a[0][1]</code>	<code>a[0][2]</code>	<code>a[0][3]</code>
Row 1	<code>a[1][0]</code>	<code>a[1][1]</code>	<code>a[1][2]</code>	<code>a[1][3]</code>
Row 2	<code>a[2][0]</code>	<code>a[2][1]</code>	<code>a[2][2]</code>	<code>a[2][3]</code>



پس اگر ما ابتدا یک آرایه از جنس `int**` به طول ۳ بسازیم که `A` سر آن باشد، و سپس هر کدام از خانه های آن آرایه را برابر با سر یک آرایه ی ۴ تایی قرار دهیم، یک آرایه ی دوبعدی ساخته ایم.

## ← ۴. انجام دهید!

در این قسمت قرار است تا با شیوه ی دیگری از دادن ورودی به برنامه آشنا شوید. تا کنون، با استفاده از تابع `scanf` و در زمان اجرای برنامه، ورودی هایی را از کاربر دریافت می کردید. اکنون شیوه ی دیگری را به شما معرفی می کنیم که بوسیله ی آن می توانید ورودی هایی را پیش از اجرای برنامه دریافت کنید. در نتیجه هنگامی که برنامه اجرا می شود، متغیرهایی که ورودی های مذکور را نگهداری می کنند، مقدار گرفته اند. نام این متغیرها در زبان `C` به ترتیب `argv` و `argc` است. در ادامه با کاربرد آنها آشنا می شوید. اکنون مراحل زیر را مطابق شکل ها انجام دهید:



پس از انجام دستور بالا، قطعه کد زیر را اجرا کرده و مقادیر موجود در **argc** را تفسیر کنید. در اینجا مقدار **argv** نشان‌دهنده‌ی چیست؟

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

```
int main(int argv, char** argc) {
    for (int i = 0; i < 5; i++) {
        printf("%s ", argc[i]);
    }
    printf("\\n");
    return 0;
}
```

**قسمت ۴:** موارد خواسته شده را انجام دهید. نتایج به دست آمده و یافته‌های خود را در کادر زیر بنویسید.



پس از اجرای کد گفته شده میتوان فهمید که خروجی کد ، آن کلمه یا جمله ای است که خودمان قبلا در آدرس گفته شده نوشتیم.

در رابطه با مقدار `argv` میتوان گفت که تعداد کلمات نوشته شده به اضافه `NULL` را نشان میدهد. یعنی با نوشتن جمله `This is a test` مقداری که `argv` نشان میدهد عدد ۵ است. ۴ کلمه و در آخر `NULL`.

۵. انجام دهید!



قسمت های مشخص شده در کد زیر را کامل کرده، سپس کد زیر را اجرا کنید.

```
#include <stdio.h>
#include <stdlib.h>
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 */

    Your Program Goes Here.

    /* 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;
}
```

**سوال :** نحوه ی آزاد سازی حافظه در سوال قبل را توضیح دهید. آیا لازم است که در یک حلقه هر سطر را جداگانه آزاد کنیم؟ یا دستور `free(A)` به تنهایی اکتفا می کند؟

**توجه:** همه ی نکات ذکر شده به راحتی قابل تعمیم به یک آرایه ی  $n$  بعدی است

**قسمت ۵ :** موارد خواسته شده را انجام دهید. نتایج به دست آمده و همچنین پاسخ سوالات را در کادر زیر بنویسید.

کد خواسته شده بصورت زیر خواهد بود :

```
#include <stdio.h>
#include <stdlib.h>
int main() {
    int row, col;
    int** A;
    printf("Enter row and column:\n");
    scanf_s("%d %d", &row, &col);
    A = (int**) malloc(row * sizeof(int*));
    if (A == NULL) exit(EXIT_FAILURE);
    for (int i = 0; i < row; i++) {
        A[i] = (int*) malloc(col * sizeof(int));
        if (A[i] == NULL)
            exit(EXIT_FAILURE);
    }
    for (int i = 0; i < row; i++) {
        for (int j = 0; j < col; j++) {
            A[i][j] = i * col + j;
        }
    }
    for (int i = 0; i < row; i++) {
        for (int j = 0; j < col; j++) {
            printf("%d ", A[i][j]);
        }
        printf("\n");
    }

    for (int i = 0; i < row; i++)
        free(A[i]);
    free(A);
    return 0;
}
```

اگر هنگام خالی کردن حافظه ، هر سطر را جداگانه خالی نکنیم و صرفاً آرایه اول که آدرس آرایه بعدی در هر خانه از آن قرار دارد را خالی کنیم در اینصورت آرایه بعدی (درونی) باقی میماند و فضایی را در حافظه اشغال میکند پس لازم است در هنگام خالی کردن آرایه های چند بعد اول از آرایه های درونی اقدام کنیم.

← ۶. فکر کنید.

کد زیر را اجرا کنید و درباره‌ی خطوط اشاره شده فکر کنید.

```
#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;
    for(i=0; i<40; i++) /* Pay attention to bound of for */
        printf("%c", s[i]); /* What's happening here? what is value of s[i]? why? */
    printf("\n");
    p++;
    printf("p[1] is %d\n",*p);
    free(p); /* What's happening here? */
}
```

قسمت ۶: نتایج به دست آمده و همچنین پاسخ های خود را در کادر زیر بنویسید.

حافظه اختصاص داده شده به `p` از نوع `int` میباشد پس ، اندازه آن برای هر خانه از آرایه ، ۴ بایت است. ولی هنگام تعریف `s` ، آنرا `char*` تعریف کردیم پس تنها یک بایت از هر ۴ بیت یک خانه استفاده میشود و بقیه آن `NULL` میشود. چون قبل از تعریف `s` ، آرایه `p` را مقداردهی کردیم پس همین مقدارها برای `s` قرار میگیرد. هنگام پرینت `s` هم کاراکتر متناظر با اسکی کد هر عدد در `s` ، پرینت میشود. پس از پرینت هر کاراکتر (که در اینجا اعداد ۰ تا ۹ میباشد) ، ۳ تا `space` که به خاطر همان `NULL` میباشد پرینت میشود.

هنگام خالی کردن آرایه `p` چون حافظه آن را به `s` اختصاص داده‌ایم ، برنامه اجازه نمیدهد که آنرا `free` کنیم و `runtime error` میدهد.

7. فکر کنید. ←

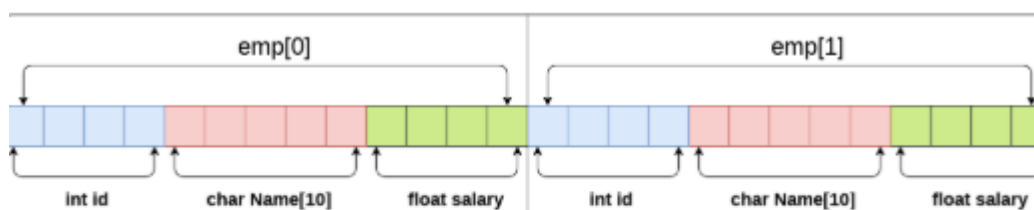
در درس با ساختار ساختمان ها (structures) در زبان C آشنا شدید. عکس پایین نمونه ای برای یاد آوری شما میباشد:

## Records (structs)

- struct: collection of a fixed number of components (members), accessed by name
  - Members may be of different types
- Syntax:

```
struct structName
{
    dataType1 identifier1;
    dataType2 identifier2;
    .
    .
    .
    dataTypeN identifierN;
};
```

همچنین میتوان آرایه ای از struct ها داشت :



```
struct employee
{
    Int id;
    char name[5];
    float salary;
};
struct employee emp[2];
```

`sizeof (emp) = 4 + 5 + 4 = 13 bytes`

هدف شما در این قسمت صرفاً تحلیل قطعه کدی است که در کنار این فایل به نام `part7_code` قرار دارد.

**قسمت 7:** قطعه کد داده شده را مطالعه کرده و تحلیل خود را در کادر زیر بنویسید.

یک استراکچر تعریف شده که شامل شماره دانشجویی، نام دانشجو، نمره ۳ تا درس در یک آرایه ۳ تایی و در نهایت معدل مییاشد.

در کد اصلی ابتدا از کاربر خواسته میشود تا شماره دانشجویی را وارد کند. سپس باید نام دانشجو وارد شود. و در یک حلقه `for` از کاربر خواسته میشود تا نمره دروس ۱ تا ۳ را وارد کند. مجموع نمرات را داخل متغیر `Total` ریخته و سپس مقدار تقسیم بر ۳ آنرا در `S.Avg` که معدل دانشجوی `S` هست میریزیم. در نهایت مشخصات دانشجو پرینت میشود.

در قسمت دوم یک استراکچر دیگر میسازیم که فرمت آرایه دارد. مشابه قسمت قبل مراحل انجام میشود. ولی چون ما یک آرایه با طول ۳ ساخته ایم تمام آن مراحل را در یک حلقه ۳ بار انجام میدهیم و سپس پرینت میکنیم.

بخش های اختیاری در یک فایل جداگانه بر روی سایت قرار گرفته است. توصیه می شود که حتماً آن ها را انجام دهید.

موفق باشید