نیمسال دوم 99-98	عنوان: آزمایشگاه ششم (آرایه‌های پویا و ساختمان ها)	استاد : دکتر مرادی

آرایه‌ی پویا[1]:

آرایه هایی که تا به حال دیده اید و از آن ها استفاده کرده اید، آرایه های ایستا[2] بوده اند. اگر یادتان باشد در تعریف این آرایه ها حتماً باید طول آن ها را با یک عدد ثابت مشخص می کردید. امروز می خواهیم با نوع دیگری از آرایه ها به نام آرایه پویا آشنا شویم. طول این آرایه ها در هنگام کامپایل نامشخص بوده و در هنگام اجرا تعیین می گردد.

دستور تخصیص حافظه[3] (malloc):

شما می توانید توسط تابع malloc که از توابع کتابخانه ای stdlib.h می باشد، از سیستم عامل درخواست کنید که مقدار مشخصی حافظه در heap گرفته و آن را در اختیار شما قرار دهد. نحوه ی استفاده از این تابع به صورت زیر است:

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

<type> : نوع داده ای که می خواهید آرایه ای پویا از آن داشته باشید.

number : طول آرایه ای که می خواهید.

حال به نکات زیر **توجه کنید**:

1. آرگومان تابع malloc مقدار حافظه درخواستی بر حسب بایت می باشد.
2. sizeof از عملگر های زبان C است که سایز هر type ای که به آن بدهید را بر حسب بایت برمی گرداند. چون سایز یک type (مثلاً int) در سیستم های 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، اشاره گر به سر آرایه ی 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 لزوما قرار نیست نوع خاصی، مثلا integer، داشته باشد) بوده و برای همین آن را به type مورد نظر cast کرده ایم.
5. هر حافظه ای که توسط تابع malloc می گیرید را در پایان **باید** توسط تابع free(pointer) آزاد کنید.

[1] dynamic
[2] static
[3] memory allocation

6. نحوه‌ی استفاده از آرایه‌های ایستا و پویا هیچ تفاوتی با هم ندارد. پس همان گونه که قبلاً با آرایه های ایستا کار می-کردید، می توانید 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;
}
```

1. انجام دهید

یکی از توابع کاربردی برای تخصیص حافظه‌های پویا تابع realloc است . از این تابع می‌توانید برای تغییر مقدار حافظه‌ای که قبلا 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:

```c
#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:

```c
#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 1 : نتایج به دست آمده و همچنین یافته‌های خود را در کادر زیر بنویسید.

در کد اول با ایجاد آرایه به اندازه 4 آن را با 0 مقداردهی اولیه می کنیم سپس اندازه همان آرایه را به 5 تغییر داده و مقدار 5 را در خانه آخر آرایه قرار می دهیم که دو مقدار زیر در نهایت چاپ می شوند.

0 0 0 0

0 0 0 0 5

تفاوت کد دوم در اینجاست که وقتی در realloc از NULL استفاده می کنیم در واقع مثل malloc عمل می کند و در اینجا حافظه جدیدی به اندازه 5 در اختیار ما قرار داده می شود و دیگر به مقادیر قبلی دسترسی نداریم و چون به این آرایه جدید مقداردهی اولیه نکرده ایم در نهایت در خط اول خروجی مثل قبل 0 0 0 0 نمایش داده می شود که برای قسمت اول کد است ولی در خط دوم مقادیر نامتعارفی نمایش داده می شود و مقدار 5 در انتهای آن ها چاپ می شود.

2. انجام دهید (Free)

1) هنگام کار با تخصیص حافظه‌ی پویا به دو نکته باید توجه نمود :
   1. لزوم آزاد کردن حافظه‌های گرفته شده
   2. عدم استفاده از حافظه‌ی آزاد شده
2) 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 2 : 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 دقیقا 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.

وقتی که از free استفاده می کنیم برنامه مکانی از حافظه را که به ما اختصاص داده بود آزاد می کند تا اگر برنامه نیاز به حافظه داشته باشد بتواند از آن استفاده کند و مقادیر قبلی که ما در حافظه قرار داده بودیم را حذف می کند یعنی در این کد در ابتدا مقادیر 0 تا 9 در جایی از حافظه قرار می گیرند که پوینتر p به ابتدا آن اشاره می کند ولی با استفاده از free کل حافظه اختصاص داده شده آزاد شده و دیگر مقادیر 0 تا 9 موجود نخواهد بود.

وقتی که p[15] را استفاده می کنیم در واقع از *(p+15) استفاده می کنیم و چون p+15 به آدرسی از حافظه اشاره می کند می توان به آن دسترسی داشت و مقدار درون آن که در اینجا مقداری نامتعارف است را چاپ کرد. مشکل index که در آرایه ها وجود داشت که بیشتر از اندازه آرایه نمی توانست باشد در استفاده از پوینتر ایجاد نمی شود چون اگر زیاد تر هم بود به جایی از حافظه اشاره خواهد کرد که می توانیم از آن استفاده کنیم و به خطا نخوریم.

3. 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.

1) 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;
}
```

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

3) سربرگ Processes را انتخاب کنید.

4) برنامه‌ی اجرایی your_project_name.exe را پیدا کنید و مقدار memory مورد استفاده‌ی آن را مشاهده کنید.

5) با فشردن دکمه‌ی f5 به تعداد 10 بار حلقه‌ی while را اجرا کنید.

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

7) چه نتیجه‌ای می‌گیرید؟

قسمت 3 : 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.

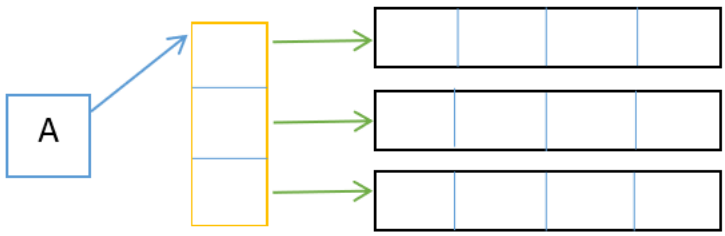
هر بار با استفاده از realloc و همچنین با افزایش i سایز اختصاص داده شده زیاد تر شده و این باعث افزایش memory در task manager می شود و این به این معناست که حافظه زیادی اشغال خواهد شد پس اگر این حافظه را آزاد نکنیم به مشکل خواهیم خورد.

با چک کردن توسط breakpoint و اجرا خط به خط دیده می شود که آدرس حافظه اختصاص داده شده هر بار تغییر می کند و این می تواند به این دلیل باشد که با افرایش حافظه اختصاص داده شده ممکن است در آدرس فعلی نتوان به میزان خواسته شده حافظه پشت سر هم کنار گذاشت برای همین مجبور به جابجایی آدرس اولین خانه بشویم تا بتوان به اندازه داده شده حافظه اختصاص داد و overwrite انجام نداد.

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 3 تایی از آرایه هایی به طول 4 می باشد. شکل زیر را نگاه کنید:

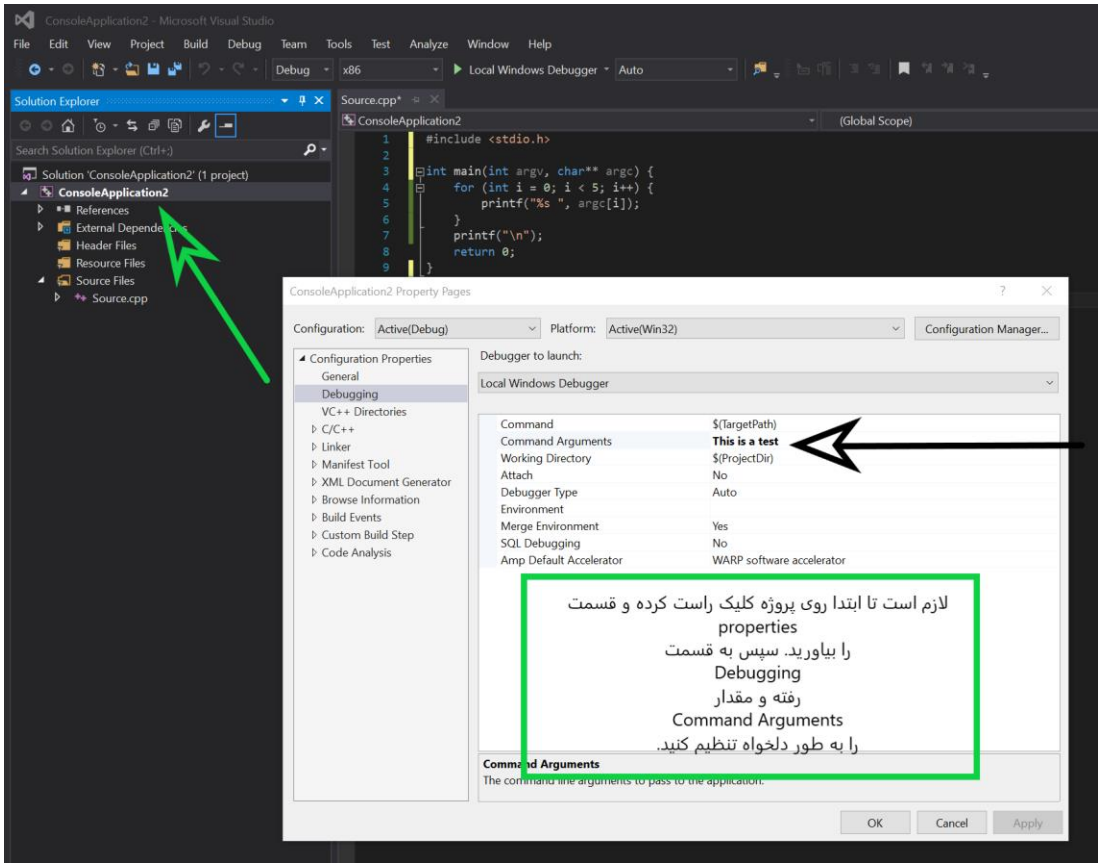
	Column 0	Column 1	Column 2	Column 3
Row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]



پس اگر ما ابتدا یک آرایه از جنس int** به طول 3 بسازیم که A سر آن باشد، و سپس هر کدام از خانه های آن آرایه را برابر با سر یک آرایه ی 4 تایی قرار دهیم، یک آرایه‌ی دوبعدی ساخته‌ایم.

4. انجام دهید!

در این قسمت قرار است تا با شیوه‌ی دیگری از دادن ورودی به برنامه آشنا شوید. تا کنون، با استفاده از تابع scanf و در زمان اجرای برنامه، ورودی‌هایی را از کاربر دریافت می‌کردید. اکنون شیوه‌ی دیگری را به شما معرفی می‌کنیم که بوسیله‌ی آن می‌توانید ورودی‌هایی را پیش از اجرای برنامه دریافت کنید. در نتیجه هنگامی که برنامه اجرا می‌شود، متغیرهایی که ورودی‌های مذکور را نگهداری می‌کنند، مقدار گرفته‌اند. نام این متغیّرها در زبان C به ترتیب argv و argc است. در ادامه با کاربرد آنها آشنا می-شوید. اکنون مراحل زیر را مطابق شکل‌ها 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 4 : موارد خواسته شده را انجام دهید. نتایج به دست آمده و یافته‌های خود را در کادر زیر بنویسید.

متغیر argc آرایه ای از رشته ها است یعنی به عنوان مثال وقتی در قسمت command argument عبارت This is a Test را قرار می دهیم در خانه اول این متغیر که یک آرایه دوبعدی است رشته This در خانه دوم is در خانه سوم a در خانه چهارم Test و در خانه آخر \0 که در واقع چون از پوینتر استفاده کرده ایم NULL قرار می گیرد پس متغیر argc رشته های داخل command argument را ذخیره می کند.

حال متغیر argv تعداد پوینترهای از جنس char یعنی *char که نشان دهنده هر سطر آرایه دوبعدی argc هستند را به ما می دهد در واقع تعداد رشته ها را نگه می دارد یعنی در مثال بالا که 5 تا رشته داشتیم (This _ is _ a _ Test _ NULL) مقدار argv برابر 5 خواهد بود.

5. 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 5 : موارد خواسته شده را انجام دهید. نتایج به دست آمده و همچنین پاسخ سوالات را در کادر زیر بنویسید.

```c
#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 = (int**) malloc(row*sizeof(int*));
	if (A == NULL) exit(EXIT_FAILURE);
	for (i = 0; i < row; i++) {
		A[i] = (int*) malloc(col*sizeof(int));
		if (A[i] == NULL)
			exit(EXIT_FAILURE);
	}

	// Print 2D Array
	for (int i = 0; i < row; i++){
		for (int j = 0; j < col; j++){
			printf("%d " , *(*(A + i) + j));
		}
		printf("\n");
	}

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

آزاد کردن حافظه درست برعکس ایجاد کردن آن است یعنی وقتی برای ایجاد کردن و اختصاص دادن با استفاده از malloc ابتدا int** را ایجاد کرده و بعد هر کدام از int* ها را ایجاد می کنیم دراینجا برعکس آن عمل می کنیم یعنی ابتدا int* ها را آزاد کرده و سپس int** را آزاد می کنیم و باید توجه داشت که اگر فقط از free(A) استفاده کنیم در واقع int** را فقط آزاد می کنیم و int* هایی که در مرحله دوم ایجاد کردن داشتیم آزاد نخواهند شد پس برای آزادسازی کامل آرایه دوبعدی چون هم int** و هم int* داریم و باید همه آن ها را آزاد کنیم باید با استفاده از حلقه و free(A[i]) ابتدا int* ها را آزاد کنیم و بعد int** را با استفاده از free(A) آزاد کنیم.

6. 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[1]? why? */
        printf("\n");
        p++;
        printf("p[1] is %d\n",*p);
        free(p); /*  What's happening here? */
}
```

قسمت 6 : نتایج به دست آمده و همچنین پاسخ های خود را در کادر زیر بنویسید.

به دلیل این که اندازه char (سایز اشغال شده در حافظه) 1 است ولی اندازه int 4 است پس وقتی که 10 خانه int داریم معادل 40 خانه char خواهد بود حال در چاپ s چون هر 4 int خانه حافظه را اشغال می کرد و 1 char خانه را،هر 4 خانه ای که به char اختصاص داده می شود به این شکل خواهد بود که خانه اول برابر مقدار ascii عدد داخل p خواهد بود و 3 خانه بعدی کاراکتر \0 در آن ها خواهد بود. پس در اینجا چون اعداد 48 تا 57 در p قرار دارند چاپ اعضا s به این شکل خواهد بود که کاراکترهای 0 تا 9 با 3 کاراکتر \0 که در میان آن ها به شکل 3 فاصله است چاپ خواهد شد.

در دستور آخر به خطا می خوریم و دلیل آن p++ است چون با این کار پوینتری اصلی خودمان یعنی p را یک واحد به جلو می بریم و جابجا می کنیم و دیگر به اولین خانه اشاره نمی کند با این کار عملیات آزادسازی حافظه به درستی انجام نخواهد شد و به خطا می خوریم.

7. فکر کنید.

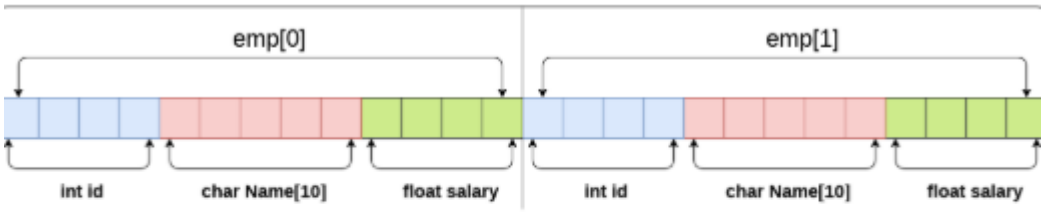
در درس با ساختار ساختمان ها (structures) در زبان C آشنا شدید. عکس پایین نمونه ای برای یاد آوری شما میباشد:

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 صرفا 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.

کاری که این کد انجام می دهد به این شکل است که هر student دارای 4 قسمت شماره دانشجویی، اسم، نمره 3 درس و میانگین این نمرات است در قسمت اول کد برای یک student که با متغیر S نشان داده می شود اطلاعات مورد نیاز از کاربر گرفته می شود یعنی شماره دانشجویی، اسم و 3 نمره دانشجو گرفته می شود و در هر قسمت S قرار می گیرد حال میانگین این 3 نمره حساب شده و در مقدار Avg این student قرار داده می شود و سپس مقادیر شماره دانشجویی و اسم و میانگین چاپ می شود. در قسمت دوم کد لیستی (آرایه) از student ها که تعداد student ها در آن 3 تا است و با متغیر Ss نشان داده می شود وجود دارد در این قسمت همه کارهای قسمت اول انجام می شود ولی به جای یک student سه student داریم و 3 بار تمام عملیات ها را انجام می دهیم و در نهایت برای سه student مقادیر را مثل قسمت قبل چاپ می کنیم.

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