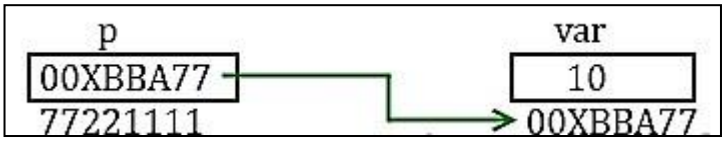
| نیمسال دوم 99-98 | عنوان: آزمایشگاه پنجم( اشاره‌گرها) | استاد : دکتر مرادی |
|---|---|---|

در این جلسه شما با اشاره‌گرها (pointer) و ارتباط آن‌ها با آرایه‌ها آشنا خواهید شد.

**تعریف اشاره‌گر :** اشاره‌گر یک متغیر است که حاوی آدرس یک متغیر دیگر در فضای حافظه است.




می‌توانیم به ازای هر نوع متغیر اشاره‌گر مخصوص به آن نوع متغیر را به صورت زیر تعریف کنیم :

```
Variable_Type *variable_name;
```

همانطور که می‌دانید می توانیم با استفاده از علامت & به آدرس یک متغیر دسترسی پیدا کنیم. همچنین برای دسترسی به محتوای متغیری که اشاره‌گر به آن اشاره می‌کند، از علامت * قبل از نام اشاره گر استفاده می‌کنیم:

```
int a = 1;
int *p;
p = &a;
int b = *p; // b’s value equals to 1
```

## 1. انجام دهید!

1) قطعه کد مقابل را نوشته، کامپایل و اجرا نمایید و با قرار دادن breakpoint در برنامه در هر قسمت مقادیر خواسته شده را مشاهده کنید:

```c
int main() {
	int x;
	int *ptr;
	int **ptr2;
	 /*مقدار موجود در اشاره‌گرها  و متغیر فوق را توجیه کنید */
	x = 25;
	ptr = &x;
	ptr2 = &ptr;
	   /*اکنون مقادیر دو اشاره‌گر فوق نشان‌دهنده چه هستند؟*/
	*ptr = 2 * **ptr2;
	printf("x = %d and address of x = 0x%p = 0x%p = 0x%x = 0x%p \n",
x, ptr, &x, &x, *ptr2);
		/* مقدار خروجی را مشاهده کنید. */
	return 0;
}
```

**نکته :** هنگام کار با اشاره‌گرها باید بسیار دقت کرد زیرا ممکن است در صورت مقداردهی اشتباه به اشاره‌گر با خطای زمان اجرا مواجه شویم.

به عنوان مثال، قطعه کد زیر را در debug mode اجرا و نتیجه را مشاهده کنید:

```c
int main(){

	int *ptr = 0x1;
	*ptr = 25;
	return 0;
}
```

قسمت 1 : آزمایش خواسته شده را انجام داده و نتایج به دست آمده را در کادر زیر بنویسید.

در قسمت اول ptr یک پوینتر است و ptr2 یک پوینتر به پوینتر دیگری است حال در قسمت بعد مقدار x برابر 25 می شود و ptr برابر آدرس x می شود و ptr2 برابر آدرس ptr که خود یک آدرس است می شود حال در قسمت بعد مقدار درون آدرس ptr برابر مقدار داخل پوینتر ptr2 (که مقدار داخل آن خود یک پوینتر است) و مقدار این پوینتر ضربدر 2 قرار می گیرد یعنی در واقع 50 می شود و در چاپ کردن هم مقدار 50 و آدرس آن چاپ می شود.

در کد بعدی با تعیین یک آدرس نمی توان در آن آدرس دلخواه ما مقدار قرار داد زیرا اجازه نوشتن بر روی حافظه در هر آدرس دلخواه به ما داده نمی شود.

رابطه ی نزدیکی بین اشاره‌گرها و آرایه‌ها وجود دارد. وقتی یک آرایه تعریف می کنید، آدرس اولین خانه‌ی آن در متغیر مربوطه ریخته می‌شود. مثلاً اگر داشته باشیم int x[10] یک آرایه با 10 خانه از نوع integer تعریف کرده‌ایم که آدرس اولین خانه‌ی آن در متغیر x ریخته شده است. حال به دو نکته زیر **توجه کنید**:

1) برای دسترسی به محتوای یک خانه ی آرایه دو روش وجود دارد:

a. از اندیس مربوطه استفاده کنیم. مثلاً x[6] (یعنی محتوای هفتمین خانه ی آرایه)
b. به روش base + offset عمل کنیم: مثلاً *(x + 6) (یعنی محتوای هفتمین خانه ی آرایه)

2) آدرس یک خانه ی آرایه نیز به طور مشابه به دو صورت می تواند بیان شود:

a. از اندیس مربوطه استفاده کنیم. مثلاً &x[6] (یعنی آدرس هفتمین خانه ی آرایه)
b. به روش base + offset عمل کنیم: مثلاً (x + 6) (یعنی آدرس هفتمین خانه ی آرایه)

## 2. انجام دهید!

با توجه به کد داده شده در سمت چپ، دو قسمت جا افتاده در کد سمت راست را با استفاده از اشاره گرها کامل کنید.

```c
#include <stdio.h>
#define SIZE 4
int main () {
    int i, sum = 0;
    int num[SIZE];
    printf("Enter %d numbers:\n", SIZE);
    for (i = 0; i < SIZE; i++)
        scanf("%d", &num[i]);
    for (i = 0; i < SIZE; i++)
        sum += num[i];
    printf("Sum: %d\n", sum);
    return 0;
}
```

```c
#include <stdio.h>
#define SIZE 4
int main () {
    int i, sum = 0;
    int num[SIZE];
    printf("Enter %d numbers:\n", SIZE);
    for (i = 0; i < SIZE; i++)
        scanf ("%d", ...); /* Complete this instruction */
    for (i = 0; i < SIZE; i++)
        sum += ...; /* Complete this instruction */
    printf("Sum: %d\n", sum);
    return 0;
}
```

قسمت 2 : نتایج به دست آمده را در کادر زیر بنویسید.

```c
#include <stdio.h>
#define SIZE 4
int main(){
	int i, sum = 0;
	int num[SIZE];
	printf("Enter %d numbers:\n" , SIZE);
	for (int i = 0; i < SIZE; i++){
		scanf("%d", num + i);
	}
	for (int i = 0; i < SIZE; i ++){
		sum += *(num + i);
	}
	printf("Sum: %d\n", sum);
	return 0;
}
```

## 3. فکر کنید!

چگونه می‌توان یک نسخه‌ی دیگر از یک آرایه داشت؟ به نظر شما روش زیر پاسخ مناسبی برای این سوال است؟ پاسخ خود را توجیه کنید.

```c
int main(){

	int arr[4] = { 1, 2, 3, 4 };
	int *arr_cpy;
	arr_cpy = arr;
	return 0;

}
```

قسمت 3 : نتایج به دست آمده و یافته‌های خود را در کادر زیر بنویسید.

با استفاده از این روش ما آدرس اولین خانه آرایه اول را در متغیر دوم قرار می دهیم و با این کار می توان مقادیر آرایه اول را به دست آورد ولی مشکل در اینجاست که چون پوینتر به قسمتی از حافظه اشاره می کند که آرایه اول در آن قرار دارد با تغییر دادن مقداری با استفاده از arr_cpy مقدار آن در آرایه اصلی نیز عوض می شود و دیگر به مقادیر قبل از تغییر دسترسی نخواهیم داشت.

برای کپی کردن می توان با استفاده از یک حلقه، آرایه ای جدید ساخته و تک تک خانه های آن را با مقادیر آرایه اصلی مقداردهی کنیم.

**نکته:** رشته‌ها که با نام دیگر string در زبان c شناخته می‌شوند علاوه بر آرایه‌ای از متغیرهای char به صورت char* نیز می‌توانند نمایش داده شوند(که در وقاع معادل یکدیگرند). برای درک بیشتر این مطلب کد زیر را مشاهده کنید.

```
char s1[10] = "Hello";
char* s2 = "Hello";
char* s3 = s1;
```

## 4. انجام دهید!

می خواهیم تابعی به نام compare بنویسیم که با گرفتن دو رشته ی first و second تعیین کند که آیا این دو رشته برابرند یا خیر؟ در صورت مساوی بودن مقدار true و در غیر این صورت مقدار false برگرداند.

برای این کار مراحل زیر را طی کنید:

1) در تابع main دو رشته از کاربر بگیرید و آن ها را در آرایه هایی به طول 70 بریزید.
یادآوری: برای خواندن رشته توسط تابع scanf به صورت زیر عمل کنید:

```
char first_array [70], second_array [70];
scanf("%s", first_array);
scanf("%s", second_array);
```

2) حال دو رشته را به عنوان ورودی به تابع compare دهید و مقدار بازگشتی را چاپ کنید.

**راهنمایی:**

```
int compare(char* first_array, char* second_array);
```

یک نمونه از اجرای برنامه فوق به صورت زیر است:

Input:
Hardware
Software
Output:
False

**توجه:** برای این که بررسی کنید 2 آرایه برابرند یا نه، باید درون یک حلقه تمامی عناصر دو آرایه را نظیر به نظیر باهم مقایسه کنید. (در مورد رشته ها تا جایی پیش می رویم که به کاراکتر null یا پایان رشته برسیم.)

به نظر شما، چگونه می‌توان تنها با یک پیمایش همزمان روی دو آرایه، هم طول و هم برابری کاراکترهای آنها را مقایسه کرد؟

قسمت 4 : موارد خواسته شده را انجام دهید. نتایج به دست آمده و همچنین یافته‌های خود را در کادر زیر بنویسید.

```c
#include <stdio.h>

int compare(char* first_array, char* second_array){
	int result = 1 , i = 0 , j = 0;
	while (*(first_array + i) != '\0' || *(second_array + j) != '\0'){
		if (*(first_array + i) != *(second_array + i)){
			result = 0;
		}
		if (*(first_array + i) != '\0'){
			i++;
		}
		if (*(second_array + j) != '\0'){
			j++;
		}
	}
	return result;
}

int main(){
	char first_array[70], second_array[70];
	scanf(" %s", first_array);
	scanf(" %s", second_array);
	if (compare(first_array, second_array)){
		printf("True\n");
	}
	else{
		printf("False\n");
	}
	return 0;
}
```

برای این که طول دو آرایه را نیز باید به دست بیاوریم i و j را تعریف کردیم که این دو مقدار در نهایت اندازه هر کدام از آرایه های ورودی را به ما می دهد ولی اگر فقط برابری مهم بود می توانستیم با تعریف فقط یک متغیر i هر بار شرط اول درون حلقه را چک کنیم و اگر برقرار بود break بزنیم ولی در اینجا برای آن که طول هر دو آرایه را باید به دست بیاوریم این کار را نمی کنیم و دو متغیر i و j را تعریف می کنیم با این کار با یکبار طی کردن همزمان هر دو آرایه هم طول هر کدام و هم برابری هر دوی آن ها را به دست می آوریم. (تا جایی پیش می رویم که رشته ما تمام شود یعنی به کاراکتر \0 برسیم)

**نکته:** در کتابخانه‌ای به نام string.h مجموعه توابعی برای کار کردن با رشته ها نوشته شده اندکه هم از نظر هزینه زمانی[1] بهینه اند و هم کار شما در کار کردن با رشته ها را آسان می کند. (برای مطالعه‌ی بیشتر، می‌توانید عبارت string.h را در Google جستجو و توابع آن را بررسی کنید.)

## 5. انجام دهید!

**نکته:** همان‌طور که پیشتر نیز دیده‌اید یکی از مزایای استفاده از اشاره‌گرها پاس دادن متغیر ها به توابع به صورتی است که بتوان مقدار آن‌ها را در تابع تغییر داد.

**نکته:** یکی دیگر از مزایای استفاده از اشاره‌گرها پاس دادن آرایه ها به توابع است به صورتی که تنها نیاز است اشاره‌گر ابتدای آرایه را به تابع منتقل کرد. برای درک بهتر این مطلب به ساختار زیر توجه کنید. در هر دو ساختار زیر آرگومان ورودی تابع یک آرایه است.

```
int func(int *a);
int func(int a[]);
```

شما قبلا با مفهوم آرایه‌ی چند بعدی آشنا شده‌اید. در مورد رابطه‌ی آرایه‌های چند بعدی با اشاره‌گر مربوطه باید به این نکته توجه نمود که حافظه‌ی کامپیوتر مانند یک آرایه‌ی یک بعدی است. لذا برای شبیه‌سازی آرایه‌هایی با ابعاد بیش‌تر سطرهای آن را پشت سر هم قرار می‌دهد و با استفاده از اشاره‌گر به آن‌ها دسترسی پیدا می‌کند. به همین دلیل جنس (Type) یک آرایه‌ی دو بعدی از int معادل int** است.

در این قسمت بنا است تا معکوس یک ماتریس را بوسیله‌ی کار با آرایه‌های دو بعدی به این سبک حساب کنید. ابتدا فایل ضمیمه شده آزمایش را مشاهده نمایید. در این قسمت شما می‌بایست تا تابعی که در ابتدای فایل مذکور تنها اظهار آن آورده شده است، پیاده‌سازی کنید:

- تابع `calc_transposed_matrix`، با گرفتن یک آرایه به عنوان آرگومان اول و در نظر گرفتن آن به عنوان یک ماتریس، ترانهاده آن را وارد آرایه‌ای که به عنوان آرگومان دوم گرفته شده می‌نماید. توصیه میشود تمامی توابع را بررسی فرمایید.

قسمت 5 : آزمایش خواسته شده را انجام دهید . نتایج به دست آمده و یافته‌های خود را در کادر زیر بنویسید.

[1] time complexity

```c
#include <stdio.h>
#define matrix_size 3
#define zero 0
#define one 1
#define two 2

void calc_transposed_matrix(double matrix[matrix_size][matrix_size],
	double transposed_matrix[matrix_size][matrix_size]){
	for (int j = 0; j < matrix_size; j++){
		for (int i = 0; i < matrix_size; i++){
			*(*(transposed_matrix + i) + j) = *(*(matrix + j) + i);
		}
	}
}

void mult_const_to_matrix(const double c, double matrix[matrix_size][matrix_size]){
	for (int i = zero; i < matrix_size; i++){
		for (int j = zero; j < matrix_size; j++){
			*(*(matrix + i) + j) *= one / c;
		}
	}
}

void print_matrix(double matrix[matrix_size][matrix_size]){
	for (int i = zero; i < matrix_size; i++){
		for (int j = zero; j < matrix_size; j++){
			printf("%lf ", *(*(matrix + i) + j));
		}
		printf("\n");
	}
}

double determinant(double matrix[matrix_size][matrix_size]){
	double determinant = *(*(matrix + zero) + zero) * ((*(*(matrix + one) + one) * *(*(matrix
+ two) + two)) - ((*(*(matrix + two) + one)) * *(*(matrix + one) + two))) - *(*(matrix + zero) +
one) * (*(*(matrix + one) + zero)
		* *(*(matrix + two) + two) - *(*(matrix + two) + zero) * *(*(matrix + one) + two))
+ *(*(matrix + zero) + two) * (*(*(matrix + one) + zero) * *(*(matrix + two) + one) - *(*(matrix
+ two) + zero) **(*(matrix + one) + one));
	return determinant;
}

int inverse_of_matrix(double matrix[matrix_size][matrix_size], double
inversed_matrix[matrix_size][matrix_size]){
	double determinant_of_matrix = determinant(matrix), ZERO = 0;
	if (determinant_of_matrix == ZERO)
		return zero;
	calc_transposed_matrix(matrix, inversed_matrix);
	mult_const_to_matrix(determinant_of_matrix, inversed_matrix);
	return one;
}

int main(){
	double a[matrix_size][matrix_size] = { { 9, 2, 3 }, { 4, 5, 6 }, { 7, 8, 9 } };
	double a_inverse[matrix_size][matrix_size];
	if (inverse_of_matrix(a, a_inverse))
		print_matrix(a_inverse);
	else
		printf("Undefined determinant\n");
	return zero;
}
```

همه قسمت های کد به پوینتر تبدیل شده است و از پوینتر برای دسترسی به اعضای ماتریس استفاده شده است.

## 6. انجام دهید!

می‌خواهیم تابعی به نام `Cyclic_Swap` تعریف کنیم به صورتی که سه متغیر را دریافت کنید و به صورت چرخش مقادیر آن‌ها را جابه‌جا کند. (مقدار متغیر اول در متغیر دوم، مقدار متغیر دوم در متغیر سوم و مقدار متغیر سوم در متغیر اول قرار داده شود.) برای انجام این کار :

1) تابع `Cyclic_Swap` را با خروجی `void` و سه ورودی از جنس `int*` تعریف کنید.
2) عملیات جابه‌جایی گردشی را درون تابع انجام دهید.
3) در تابع `main` کد آزمایش برنامه را بنویسید و با دریافت ورودی مناسب، خروجی مناسب را چاپ نمایید.

قسمت 6 : موارد خواسته شده را انجام دهید. نتایج به دست آمده و یافته‌های خود را در کادر زیر بنویسید.

```c
#include <stdio.h>
void Cyclic_Swap(int* a, int* b, int* c){
	int temp1, temp2;
	temp1 = *b;
	temp2 = *c;
	*b = *a;
	*c = temp1;
	*a = temp2;
}

int main(){
	int a, b, c;
	printf("Please Enter 3 numbers:\n");
	scanf(" %d" , &a);
	scanf(" %d", &b);
	scanf(" %d", &c);
	Cyclic_Swap(&a, &b, &c);
	printf("%d %d %d\n", a, b, c);
	return 0;
}
```

از آدرس متغیر ها برای swap استفاده شده است.

موفق باشید