

## Q0:

```
#include "main.h"

/***
 * _strcat - Concatenates two strings.
 * @dest: The destination string.
 * @src: The source string.
 *
 * Return: A pointer to the resulting string @dest.
 */

char *_strcat(char *dest, char *src)
{
    char *ptr = dest;
    while (*ptr)
        ptr++;
    while (*src)
        *ptr++ = *src++;
    *ptr = '\0';
    return (dest);
}
```



## شرح الكود:

## فكرة السؤال:

الدالة `_strcat` تأخذ نصين `dest` و `src` ، وتلصق (`src`) في نهاية (`dest`) بحيث يكون النص النهائي كله في `dest`. طبعاً لازم نحط \0 في النهاية عشان النص يكون مضبوط.

- 1 تكتب دالة اسمها `_strcat` وظيفتها دمج النص الموجود في `src` داخل `dest`.
- 2 الدالة يجب أن تعدل `dest` مباشرة، بحيث يتم وضع `src` في نهايتها.
- 3 يجب التأكد من أن \0(نهاية النص) يتم وضعه بعد عملية الدمج.
- 4 الدالة ترجع مؤشر `dest` بعد التعديل.
- 5 لا يسمح باستخدام `strcat` الجاهزة في C ، لازم تكتبها بنفسك.

## ملاحظة :

- \* الرمز هذا معناه مع تعريف المتغير تعني "مؤشر".
- \* أثناء الاستخدام تعني "القيمة في العنوان اللي يمسكه المؤشر".
- \* يحرك المؤشر للأمام، و `*ptr++` يعني استخدام القيمة ثم انتقل.

مثال توضيحي `strcat` الجاهزة :

إذا كان عندنا الكود التالي:

```
char s1[20] = "Hello ";
char s2[] = "World!";
_strcat(s1, s2);
printf("%s\n", s1);
```

قبل تنفيذ الدالة:  
"Hello " و `s2` يحتوي "World!" .

بعد تنفيذ :  
"Hello World!". s1

الناتج عند الطباعة:

Hello World!

شرح الكود سطر بسطر:

تحتاج نعرف نهاية dest

في لغة C ، النصوص (strings) تنتهي دائمًا بالرمز 0 \ علامة نهاية النص.  
لازم نمشي على dest حرف إلى أن نوصل لنهاية النص (0) عشان نعرف وبين نبدأ نضيف src.

مثال عملي:

char dest[20] = "Hello ";

فيها "Hello" ، وأحرفها كلها في الذاكرة:

H e l l o \0

الحرف الأخير (0) يعلمنا أن النص وقف هنا، فابحنا نحتاج نبدأ نلصق src في هذا المكان.

تعرف مؤشر يمسك dest

char \*ptr = dest;

- هذا عرفنا متغير ptr (مؤشر) وخليته يساوي dest ، عشان لا نعدل على العنوان الأصلي لـ dest .
- بالختصار، هذا ptr راح نستخدمه كاداة تساعدنا نوصل لنهاية dest .
- ptr نتحرك إلى نهاية dest

while (\*ptr)

- الحلقة هذه تمشي على dest حرف حرف إلى توصل لأول 0 يعني نهاية النص.
- بعد ما تخلص الحلقة، ptr بيكون واقف عند المكان اللي راح نبدأ فيه اللصق.

مثال عملي:

لو كان عندنا dest كذا:

char dest[20] = "Hello ";

فالحلقة راح تمشي على الأحرف:

H → e → l → l → o → " (مسافة) 0\ تتوقف عند

ptr++;

ptr هو مؤشر (pointer) ، يعني يمسك عنوان أول حرف من النص ويتحرك بين الأحرف وحدة وحدة .  
يعني تحرك للمكان اللي بعده، أي انتقل للحرف التالي في النص .

مثال عملي:

char dest[20] = "Hello ";

char src[] = "World!";

dest مخزنة في الذاكرة كذا:

H e l l o \0 \_ \_ \_

src مخزنة كذا:

W o r l d ! \0

كيف ننسخ dest إلى src باستخدام ptr++؟

عندنا مؤشر `ptr` يمسك آخر حرف في `dest` قبل `\0`. كل مرة ننسخ حرف من `src` إلى `dest`، لازم `ptr` يتحرك للمكان اللي بعده عشان ما يكتب فوق الأحرف القديمة.

قبل النسخ:

H e l l o \0 \_ \_ \_



`ptr` واقف عند `\0`، لازم نبدأ ننسخ من `src`.

نسخ أول حرف `W` من `src` إلى `dest`:

H e l l o W \_ \_ \_



(`ptr++`) تحرك للمكان اللي بعده.

نسخ الحرف `o`:

H e l l o W o \_ \_ \_



نسخ الحرف `r`:

H e l l o W o r \_ \_



نعمل لين آخر حرف : !

H e l l o W o r l d ! \0



لما نوصل لنهاية `src`، لازم نضيف `\0` في `dest` بعد آخر حرف. (`ptr = '\0'`).

نسخ `src` داخل `dest`:

بعد ما عرفنا وبين نهاية `dest`، راح نأخذ `src` حرف حرفاً ونحطه مكان `\0` في `dest`. إذا كان `"src = "World!"`، راح ننسخها كذا:

H e l l o W o r l d ! \0

كل مرة ننسخ حرف، لازم `dest` يتحرك للأمام. (`ptr++`) في النهاية، لازم نضيف `\0` مرة ثانية عشان النص يكون مضبوط.

**بدأ نسخ dest إلى src**

**while (\*src)**

**\*ptr++ = \*src++;**

هنا راح نأخذ كل حرف من **src** ونحطه في **dest**, بحيث **ptr** بيتحرك للأمام مع كل حرف.  
بالمثل، **src** بيتحرك للأمام لين يصل لـ **\0**.

**خط ٥ عشان نضمن نهاية النص**

**\*ptr = '\0';**

هذا السطر يحط **\0** في آخر النص عشان يكون مكتمل بدون مشاكل.

**ملحوظة :** أول ما تكتب نص في **dest**, الكمبيوتر بيضيف **\0** تلقائياً.

في الدالة **\_strcat** ، نحن نحتاج نضيف **\0** يدوياً في النهاية بعد إضافة **src** إلى **dest**.

لكن بعد ما ننسخ النص من **src** إلى **dest**، نحتاج نضيف **\0** يدوياً في آخر **dest** لعدة أسباب:

لأننا في الدالة **\_strcat** ننقل النصوص يدوياً. لما ننسخ النصوص، الكمبيوتر ما يعرف متى يوقف، فاحنا نضيف **\0** يدوياً عشان نقول له هنا ينتهي النص.

**ترجم dest قيمة راجعة**

**return (dest);**

في النهاية، نرجع **dest** عشان أي أحد يستعمل الدالة يحصل النص بعد التصق.

## q1:

```
1. strncat
#include "main.h"
/***
* _strncat - Concatenates two strings, using at most n bytes
* from src.
* @dest: The destination string.
* @src: The source string.
* @n: The maximum number of bytes to use from src.
* Return: A pointer to the resulting string dest.
*/
char *_strncat(char *dest, char *src, int n)
{
    char *ptr = dest;
    while (*ptr != '\0')
        ptr++;
    while (*src != '\0' && n > 0)
    {
        *ptr = *src; ptr++; src++; n--;
    }
    *ptr = '\0';
    return dest;
}
```

## شرح الكود:

فكرة السؤال :

نكتب دالة اسمها `_strncat` ، مهمتها تجمع (تمجي) جزئيات من النص (string) الثاني `src` مع النص الأول `dest`، لكن تمجي عدد محدد من الأحرف من `src` فقط، هذا العدد هو `n`.

```
include <stdio.h>
char * strncat(char *dest, char *src, int n)
{
    char *ptr = dest;
    while (*ptr != '\0')
        ptr++;
    while (*src != '\0' && n > 0)
    {
        *ptr = *src;
        ptr++;
        src++;
        n--;
    }
    if (*ptr == '\0')
        return dest;
}
```

الدالة تستقبل ثلاثة معاملات:

.1 .قطق السلسلة النصية الأساسية التي سيتم الإضافة إليها.

.2 .قطق السلسلة النصية التي سيتم نسخ أول `n` أحرف منها إلى `dest`..3 .قطق عدد الأحرف القصوى التي يجب إضافتها من `src` إلى `dest`.الفرق بين `_strcat` و `_strncat`:• `_strcat` تنسخ كل إلى `dest` حتى تصل إلى `\0`.• `_strncat` تنسخ فقط `n` حرف كحد أقصى من `src` إلى `dest`.• `src` لا تحتاج إلى أن تكون منتهية بـ `\0` إذا كانت تحتوي على `n` أو أكثر من الأحرف.يجب إرجاع مؤشر إلى `dest` بعد الإضافة.

شرح الكود سطر بسطر:

`char *ptr = dest;`هنا نعرف متغير `ptr` من نوع `char *`(مؤشر على حرف)، ونخلية يشير إلى بداية النص `dest`.يعني بنسخدم هذا المتغير علشان نروح لنهاية النص `dest` علشان نبدأ نضيف الحروف من `src` هناك`while (*ptr != '\0')``ptr++;`نبدأ دور في النص `dest` خطوة خطوة إلى أن نوصل للـ `'\0'`، اللي يعني نهاية النص.كل ما مررنا على حرف، ننتقل للحرف اللي بعده، وهذا اللي بسويه `ptr++`. يعني كل مرة، `ptr` يتحرك خطوة واحدة.`while (*src != '\0' && n > 0)``{``*ptr = *src;``ptr++;``src++;``n--;``}`نبدأ ننسخ الحروف من `src` إلى `dest`. نوقف إذا وصلنا لنهاية `src` يعني `(*src == '\0')` أو إذا خلصنا الحروف اللي نبي ننقلها يعني إذا `(n == 0)`.نببدأ ننسخ الحرف اللي يشير له `src`، ونحطه في المكان اللي يشير له `ptr` في `dest`.`dest = *src;` يعني ننسخ قيمة الحرف من `src` إلى `dest`.بعد ما ننسخ الحرف، ننتقل للحرف اللي بعده في `dest` باستخدام `ptr++`.نفس الشيء مع `src`، ننتقل للحرف اللي بعده باستخدام `src++`.هنا، نخصم 1 من `n` علشان نعرف كم حرف باقي ننسخ.نكرر هذى العملية إلى أن نخلص الحروف اللي نبي نضيفها أو نوصل لنهاية `src`.`*ptr = '\0';`بعد ما نخلص من نسخ الحروف، نضيف `'\0'` في نهاية النص `dest` علشان نحدد إن النص انتهى.من دون هذا `'\0'`، لو حاولت طبع النص أو تستخدمه، يمكن يسبب لك مشاكل لأن ما فيه علامات تنتهي النص.

```
    return dest;
```

```
}
```

أخيراً، نرجع النص `dest` بعد ما أضفنا له الحروف من `src`.

عسان لو أحد حاب يستخدم النص المعدل، يقدر يحصل عليه بسهولة.

## q2:

- \* `_strncpy` - Copies a string up to n characters.
- \* `@dest`: The destination buffer.
- \* `@src`: The source string.
- \* `@n`: The maximum number of characters to copy.
- \* Return: A pointer to the destination string.

```
*/
```

```
char *_strncpy(char *dest, char *src, int n)
```

```
{
```

```
int i;
```

```
for (i = 0; i < n && src[i] != '\0'; i++)
```

```
{
```

```
    dest[i] = src[i];
```

```
}
```

```
for (; i < n; i++)
```

```
{
```

```
    dest[i] = '\0';
```

```
}
```

```
return (dest);
```

```
}
```

## Or Other code :

```
char *_strncpy(char *dest, char *src, int n)
```

```
{
```

```
    char *ptr = dest;
```

```
    while (*src != '\0' && n > 0)
```

```
{
```

```
        *ptr = *src; ptr++;
```

```
        src++; n--;
```

```
}
```

```
    while (n > 0)
```

```
{
```

```
        *ptr = '\0';
```

```
        ptr++;
```

```
n--;
```

LeenAlsaleh-Holbertonschool

```

    }
    return dest;
}

```

**Leen Alsaleh**

### شرح الكود :

#### فكرة السؤال :

السؤال يطلب منك تكتب دالة تنسخ نص من `src` الى `dest`، بس الفرق هنا إنك تنسخ عدد معين من الحروف (تحددده بالـ `n`). هذي الدالة تشبه `strncpy` الموجودة في مكتبة C القياسية.

- تأخذ النص الموجود في `src` وتنتقله إلى `dest`، بس ما تنسخ أكثر من `n` حروف.

- لو كان `src` أصغر من `n`، تكمل باقي `dest` بـ `\0` علشان تتأكد إنه نص صحيح.)

- ترجع `dest` بعد التعديل.

دالة `strncpy` نسخ النص

هدفها: تنسخ نص (`src`) إلى نص آخر (`dest`) مع تحديد عدد الحروف. (`n`)  
كيف تعمل؟

- تمسح `dest` بالكامل وتحطط فيه `src` حتى عدد معين. (`n`)

- لو `src` أقصر من `n`، تكمل باقي `dest` بـ `\0`.

- ما تضيف شيء على النص الأصلي في `dest`، بل تستبدل بالكامل.

### مقارنه بين الدالتين

| (الدالة) <code>_strncat</code>      | (الدالة) <code>_strncpy</code>               | (ال)       |
|-------------------------------------|--|------------|
| dest في نهاية المدى                 | نسخ النص إلى dest                            | النهاية    |
| يبدأ من نهاية dest                  | يبدأ من بداية dest                           | الموقع     |
| يتم إحلال النص القديم ويحتوي على    | ينسخ النص القديم في dest.                    | التأثير    |
| يختفي، (dest عن النهاية بعد الإضافة | dest أنفسها، بحسب (n) عن النهاية بعد الإضافة | نهاية النص |

#### متى أستخدم كل وحدة؟

- إذا كنت تبي تستبدل النص بالكامل استخدم `_strncpy`.

- إذا كنت تبي تضيف على النص بدون مسح القديم استخدم `_strncat`.

مثال توضيحي: تخيل `dest` هو ورقة، و `src` هو جملة تبي تكتبها:

- `_strncpy` مثل لو مسحت الورقة وكتبت الجملة الجديدة من البداية.

- `_strncat` مثل لو كملت الكتابة في نهاية الورقة.

### شرح الكود سطر بسطر:

`char *_strncpy(char *dest, char *src, int n)`

الدالة تأخذ 3 متغيرات:

- المكان اللي بتنسخ فيه النص.

- النص اللي بنسخه.

- الحد الأقصى للأحرف اللي بنسخها.

♦ ترجع dest بعد التعديل، بحيث يكون النص الجديد جاهز للاستخدام.

```
char *ptr = dest ;
```

// حفظ بداية dest عشان ترجعها

هنا نعرف مؤشر (ptr) ونربطه بـ dest، بحيث ptr يشير لنفس المكان اللي dest يشير له.  
السبب؟

✓ نحتاج المؤشر ptr عشان ننتقل داخل dest وننسخ فيه.  
✓ نفسه ما نبي نعمل عليه مباشرة عشان نقدر نرجعه في النهاية.

```
while (*src != '\0' && n > 0)
{
    *ptr = *src;
    ptr++;
    src++;
    n--;
}
```

هذا هو الجزء الأساسي اللي ينسخ النص.

الحالة تستمر طالما:

\*src ≠ '\0' يعني لسه فيه حروف ننسخها

✓ ما انتهى (عشان ما ننتهي الحد المسموح).  
كل دورة:

- نحط الحرف الحالي (\*src) في ptr يعني ننسخ الحرف.
- نحرك ptr و src للحرف اللي بعده.
- نقل n عشان نعرف كم باقي لنا.

```
while (n > 0)
{
    *ptr = '\0';
    ptr++;
    n--;
}
```

إذا كان أقصر من n، لازم نكمل باقي dest بـ '\0'.  
نستخدم while عشان نضيف '\0' في الأماكن الفاصلة.

```
return dest;
```

بعد ما ننتهي، نرجع dest بحيث يكون النص الجديد جاهز.

شرح الكود الآخر:

```
for (i = 0; i < n && src[i] != '\0'; i++)
{
    dest[i] = src[i];
}

while (i < n)
{
    dest[i] = '\0';
    i++;
}
```

**while (i < n)**

- هذه حلقة تكرارية (**while loop**) تشتغل طالما **i** أقل من **n**.
- معناها: إذا انتهى النص الأصلي **src** قبل ما نوصل **n**، راح نكمل المساحة الباقية في **dest** بـ **\0** حرف إنتهاء النص.

**dest[i] = '\0';**

- هذا السطر يحط **\0** في **dest[i]** عشان يتأكد إن باقي الأماكن في المصفوفة تصير فارغة.

- هذا يضمن إن **dest** يكون نص صحيح. **(null-terminated string)**.

**i++;**

- يزيد **i** بمقدار 1 عشان ينتقل للمكان اللي بعده ويضيف **\0** هناك، ويستمر لين **i == n**

مثال عشان تفهم الفكرة بوضوح:

خلتا نقول إن عندنا هذا الكود:

```
char dest[10];
```

```
char src[] = "Hello";
```

```
int n = 7;
```

```
for (i = 0; i < n && src[i] != '\0'; i++)
```

```
{
```

```
    dest[i] = src[i];
```

```
}
```

```
    dest[i] = '\0';
```

```
printf("%s\n", dest);
```

وش اللي بيصير؟

- **src** تتحتوي على **"Hello"** ( 5 أحرف+ **\0** )

- **n = 7**، يعني راح نحاول ننسخ 7 أحرف بس **src** أصفر.

- الحلقة تبدأ من **i = 0**، وتنقل الحروف:

```
dest[0] = 'H' .1
```

```
dest[1] = 'e' .2
```

```
dest[2] = 'l' .3
```

```
dest[3] = 'l' .4
```

```
dest[4] = 'o' .5
```

.6. الحلقة توقف لأن '**\0**' = **src[5]** وصلنا لنهاية النص.

- النص في **dest** الآن **"Hello"**، لكن بدون **\0** في النهاية إلا إذا ضفناه يدويا.

Q3:

```
#include "main.h"

/** 
 * _strcmp - compares two strings.
 * @s1: first string.
 * @s2: second string.
 *
 * Return: the difference between the first non-
 * matching character or 0
 * if the strings are equal.
 */

int _strcmp(char *s1, char *s2)
{
    while (*s1 && (*s1 == *s2))
    {
        s1++;
        s2++;
    }
    return (*s1 - *s2);
}
```



## شرح الكود :

فكرة السؤال:

المطلوب منك تكتب دالة اسمها `_strcmp` تقارن بين نصين (`strings`) بنفس طريقة الدالة الجاهزة `strcmp` في لغة C  
لما نقارن بين نصين، نشوف الحروف واحد واحد ونقارن الأكواذ الرقمية حقها (`ASCII values`).

- لو النصين متطابقين تماماً، نرجع 0.
- لو النص الأول (`s1`) أصغر من الثاني (`s2`) يعني أول اختلاف كان في حرف أصغر، نرجع عدد سالب.
- لو النص الأول (`s1`) أكبر من الثاني (`s2`) يعني أول اختلاف كان في حرف أكبر، نرجع عدد موجب.

مثال :

```
strcmp("Apple", "Banana")
//يرجع عدد سالب
65 - 66 = -1، وأول حرف في Banana هو B وكرده 66، الفرق. -1
strcmp("Hello", "Hello")
//يرجع 0
كل حرف يطابق الثاني، فيكمل اللوب إلى النهاية، ويرجع 0 لأن s1 == s2 صاروا
```

### وش فرق عن `_strncpy` و `_strcmp`

| الدالة                | وظيفتها  |
|-----------------------|--|
| <code>_strncpy</code> | تنسخ النص من <code>src</code> إلى <code>dest</code> وتستبدل المحتوى بالكامل. |
| <code>_strncat</code> | تضيف النص من <code>src</code> إلى نهاية <code>dest</code> بدون مسح الموجود.  |
| <code>_strcmp</code>  | نقارن النصوص وتشوف من الأكبر أو إذا كانوا متساوين.                           |

شرح الكود سطر بسطر :

`while (*s1 && *s2 && *s1 == *s2)`

{

هنا بنبدأ نقارن كل حرف في `s1` مع الحرف المقابل له في `s2`.

طالما الحروف متساوية (`*s1 == *s2`) ، ينكمّل للمقارنة مع الحرف اللي بعده.

نوقف المقارنة إذا:

.1. وصلنا لنهاية واحد من النصين '\0' == \*s1 == '\0' أو '\0' == \*s2 .

.2. لقينا اختلاف بين الحروف.

`s1++`

`s2++`

هذا بحرّك المؤشر

لانتاكم إننا نقارن كل الحروف واحد واحد (pointer).

`return (*s1 - *s2);`

لو الحروف كانت متساوية طول الوقت، راح يوصل `s1` و `s2` لنهاية النص (\0)، فيكون الفرق 0 والنصين متطابقين.

لو لقينا اختلاف، بنرجّع الفرق بين أول حرف مختلف: (ASCII value)

• عدد سالب يعني `s1` أصغر من `s2`.

• عدد موجب يعني `s1` أكبر من `s2`.

Q4 :

```
#include "main.h"

/**
 * reverse_array - reverses the content of an array of integers.
 * @a: array of integers.
 * @n: number of elements in the array.
 */

#include <stdio.h>

void reverse_array(int *a, int n)
{
    int start = 0, end = n - 1;
    int temp;

    while (start < end)
    {

```



```
temp = a[start];
a[start] = a[end];
a[end] = temp;
start++;
end--;
}
}
```

Or:

```
void reverse_array(int *a, int n)
{
    int temp, i;
    for (i = 0; i < n / 2; i++)
    {
        temp = a[i];
        a[i] = a[n - 1 - i];
        a[n - 1 - i] = temp;
    }
}
```

شرح الكود:

فكرة السؤال:

السؤال يطلب منك كتابة دالة تقلب محتويات مصفوفة أعداد صحيحة(**int array**) ، بحيث أول رقم يصير آخر رقم، وثاني رقم يصير قبل الأخير، وهكذا...

يعني لو عندك المصفوفة:

[0, 1, 2, 3, 4, 5]

بعد استدعاء

`reverse_array(a, 6);`

تصير كذا:

[5, 4, 3, 2, 1, 0]

فكرة الحل بسيطة:

نستخدم مؤشرين (**pointers**) أو فهرسين (**indexes**) ، واحد يبدأ من أول المصفوفة (0) (start = 0) والثاني من آخر المصفوفة. (end = n-1).

نبدل (swap) بين العنصر الأول (a[start]) والعنصر الأخير (a[end]).

نحرك للأمام (start++) والخلف (end--).

نكرر العملية حتى يصل start إلى end أو يتعداه.

لو عندنا المصفوفة:

a = [0, 1, 2, 3, 4, 5]

الخطوات:

أول تبديل:

start = 0, end = 5

تبديل a[5] و a[0]

[5, 1, 2, 3, 4, 0]

ثاني تبديل:

start = 1, end = 4

تبديل a[4] و a[1]

[5, 4, 2, 3, 1, 0]

ثالث تبديل:

start = 2, end = 3

تبديل a[3] و a[2]

[5, 4, 3, 2, 1, 0]

نهاية الحلقة:

لما يصير end >= start ، يتوقف التبديل والمصفوفة صارت معكوسه تماما!

شرح الكود 1 سطر بسطر :

`void reverse_array(int *a, int n);`

a هو مؤشر (pointer) يشير إلى أول عنصر في المصفوفة.

n هو عدد العناصر في المصفوفة.

وظيفة الدالة:

تأخذ المصفوفة (a) وتقلب ترتيب الأرقام اللي فيها.

`int start = 0, end = n - 1;`

`int temp;`

`start = 0;`

هذا متغير يمثل بداية المصفوفة (أول عنصر)

`end = n - 1;`

هذا يمثل نهاية المصفوفة (آخر عنصر)

`temp:`

متغير مؤقت نستخدمه عشان نبدل القيم بين

`start و end`

`while (start < end)`

اللوب (الحلقة) تستمر طالما start أصغر من end ، يعني طالما فيه عناصر لم يتم تبديلها .  
بمجرد ما يصير أكبر أو يساوي end ، نوقف ، لأنه خالص ، كل العناصر تم تبديلها .

`temp = a[start];`

`a[start] = a[end];`

`a[end] = temp;`

الخطوة 1: بحفظ قيمة a[start] في temp عشان لا تضيع قيمته بعد التبديل.

الخطوة 2: بخلي a[start] يأخذ قيمة a[end].

- الخطوة 3: نخلي  $a[start]$  يأخذ القيمة الأصلية لـ  $a[end]$  (التي حفظناها في  $temp$ ). بهذه الطريقة، يتم تبديل أول وأخر عنصر في كل تكرار للحلقة.

```
start;++
end;--
```

- بعد ما نبدل القيم، نحرك  $start$  للأمام (+1) عشان ننتقل للعنصر اللي بعده.
  - نحرك  $end$  للخلف (-1) عشان ننتقل للعنصر اللي قبله.
- بهذه الطريقة، نقترب تدريجياً من منتصف المصفوفة، وب مجرد ما يتقابل  $start$  مع  $end$ ، نوقف.

#### مثال عملى عشان تفهم الفكرة

قبل تنفيذ الدالة:

عندك المصفوفة:

[1, 2, 3, 4, 5]

الخطوات أثناء التبديل:

|   | النكرار | start | end             | المصفوفة بعد التبديل |
|---|---------|-------|-----------------|----------------------|
| 1 | 0       | 4     | [5, 2, 3, 4, 1] |                      |
| 2 | 1       | 3     | [5, 4, 3, 2, 1] |                      |
| 3 | 2       | 2     | (توقف)          |                      |

بعد التبديل النهائي، المصفوفة تكون معكوسة:

[5, 4, 3, 2, 1]