

Lec 12

Pointer with strings :-

string declaration with a pointer :

char *str = "PIC";

Str

91C0

0x0802

I	P	→	49	50	0x91C0
10	C		00	43	0x91C2

* دى خريطة أخرى لو أنا عايز اعرف string بس مفترش

اعدل قيمة str لأنها سمتا هتخزن

و هكانت (Read only) مساحة مكتبة

البواينر دا Global أو Local

مفترش اعدل فيه ولو حاولت

crash كدا البرنامج <

لوباء المفعول str

printf("%s", str);

* لوباء افتح الـ crash بالشكل دا :-

const char *str = "PIC";

لدا أنا عرفته إن البواينر بسها ورعل const data ولو حاولت

اعدل فيه وهو بالشكل دا هيفعل

* لوعاينز أثير الـ string هروح آخرته في arry واعدل فيه براحته

const char *Message1 = "STR";
 const char *Message2 = "EMbedded";

لأننا نعمل طبق بالشكل دا -

Message1 = Message2;

* كلانا ضيعت محتوى Message1 وبما نحن فيها النص دا
 "STR" وعشان القادر على حجمه زي دا

const char *const Message1 = "str";

* لأننا نغير اطبع رسالة message2 من أول حرف b
 printf("%s\n", (message2 + 2)); لآخر الـ string

* منالمعروف أن Message2 دا يوينر من نوع char فستاور له أول حرف b و string فالآن جمعت له 2 بقباس ستاور يعني عنوان حرف b offset واتأله دى اسما

printf("%s", *(Message1 + 2));

* البرنامج هيحصله crash لعدة دا يوينر من نوع char crash فالآن يحاول access والآن ينادي بقا دا :-

printf("%c\n", *(Message1 + 2));

* هنا صيغ حرف b بس

Double Pointer :-

`unsigned int NumberOne = 0x11;`

`unsigned int *ptrNumberOne = &NumberOne;`

`unsigned int **ptrPtrNumberOne = &ptrNumberOne;`

double Pointer

Address

Value

0x0000 0041

0x11

0x0000 0024

0x0000 0041

0x0000 0038

0x0000 0024

Pointer ایضاً double Pointer است خصم الـ ~~double~~
 مثلاً Variable ایضاً عنوان دیگرین داشت

unsigned int NumberOne = 0x11;

unsigned int *PtrNumberOne = &NumberOne;

unsigned int **PtrPtrNumberOne = &PtrNumberOne;

printf("NumberOne Address = 0x%0X\n", &NumberOne);

printf("PtrNumberOne Value = 0x%0X\n", PtrNumberOne);

printf("NumberOne Value = 0x%0X\n", *PtrNumberOne);

printf("PtrNumberOne Address = 0x%0X\n", &PtrNumberOne);

printf("PtrNumberOne Value = 0x%0X\n", *(&PtrNumberOne));

printf("PtrPtrNumberOne Value = 0x%0X\n", PtrPtrNumberOne);

printf("PtrPtrNumberOne Value = 0x%0X\n", *PtrPtrNumberOne);

→ Content of PtrPtrNumberOne

printf("PtrPtrNumberOne Value = 0x%0X\n", **PtrPtrNumberOne);

→ Content of NumberOne = 0x11

double Pointer (Pointer to pointer) مفهوم المزدوجية بين المنشئ والمتغير

** → *

→ Variable

Return Address from function in C

- If you need to return the address of a variable from a function, this address must be valid address after the function execution.
- This means that, the variable needs to be **"static variable"**

* لو الفانكتشن هترجع عنوان بليت قبل إسم الفانكتشن (Astriks)

* لوبيرجع عنوان لياتا مثلك من فانكتشن لازم الرايادي ميتكونش مخبوظ لها مساحة في stack frame رباع الـ function لذة function ملأ يتمام execute ييصلها destroyed

الـ stack بيل محتوياته عسان كدا لو هرجع عنوان داتا لـ variable لازم يكون الـ static وباراي

هتيحجز له مساحة في data segment وبالخلف لـ الـ static خلاص تغيرن الـ variable ييفضل موجود عسان صنوع

* هتلعالي الجزء دا لو جاييز الـ فانكتشن ترجع array

unsigned int * Return_Array(void) { }

Static unsigned int Numbers[5] = {11, 12, 13, 14, 15};

هناك اسفلت من خصائص الـ array
وهي ان اسم الـ array ببارقة عن pointer يستaura على عنوان أول عنصر
وبالخلف ارجع عنوان أول عنصر من الفانكتشن كما رجعت الـ array

Arrays :-

- Relationship between Pointer and array inc.
- If my_name is an array of character then "my_name" will be the address of the first element.
- You can't change the value of this address "Fixed"
- Always points to the first element of the array.
- You can use it + offset to access the array

expression describe how array decay in pointer:

1- one dimensional array :-

$$\text{my_name}[\text{index}] = *(\underbrace{\text{my_name}}_{\text{array}} + \underbrace{\text{index}}_{\text{فقط العنصر}})$$

2. Two dimensional array :-

$$\text{my_name}[\text{index_1d}][\text{index_2d}] = *(\underbrace{\text{my_name}[\text{index_1d}]}_{\substack{\text{array or} \\ \text{عنوان أول عرض}}\leftarrow} + \underbrace{\text{index_2d}}_{\substack{\text{رقم الصف} \\ \downarrow}})$$

رقم العمود

رقم الصف

عنوان أول عرض

3 - Two dimensional array

`my_name [index_1d] [index_2d] = *(* (my_name + index_1d) + index_2d)`

$$\frac{*(*(\underline{\text{my_name}} + \underline{\text{index_1d}}))}{\downarrow} + \frac{\underline{\text{index_2d}}}{\downarrow}$$

عنوان أول عنصر
عنوان الarray
رقم العمود
رقم الصف

• Relationship between Pointer and array in C (2D)

`char names [] []`

لوعاين الاستخدم الـ 2D في تخزين الـ strings او أول [] يدل على طرد الـ strings او عندى ونات [] يدل على size بل يلزم عل عنصر زائد بـ 1 او string نفسه ويفضل اتن بل يلزم عل عنصر زائد 8 او [] الثالث عسان " \0 " NULL terminator

لوعاين اغلل array يتحزن addresses يعني العنادم يتحزنها مبارقة عن بويش يتحزن عن اول عنده الـ array اسمها {array of Pointers}

Syntax:-

Data type * name of array [size]

Ex: unsigned int * PTR ARR [2]

لوعاين اخلي الـ array لا يخزن عنادون مخربق محتوى العنادون
الـ 2D الـ arrays هخليل الـ pointers const وبانتاي معتبرة اغل على ++ -- اعلانات حسابية
unsigned int * CONST PTR ARR [2] = { , }

* لما حذفنا const من指针 pointer لا يحتوى العناوين
 ولأن المترجع على الـ offset هو الـ index
 داعي الـ element او ملحوظة الـ offset من يغير محتوى
 الـ array على عكس الـ Post,pre increment يغير محتوى العنصر

String array using the array of pointer to char.

```
char *Names[2] = {"Mostafa", "Mohamed"};
```

* المريحة هي حملت اعل بيهما او انعرف بيهما strings هنا مفروض
 يجب ترتيب الميموري بحسب نوع اولين كذا المثال دا هيخزن عنوان
 اول حرف من كل string على الراي ، اهمال string نفسه في مكان ثالث
 الالوميل يحجز لحرف واحد بایه وبابايز زياده لـ "١٥"
 ه آخر كل string

```
char names_2D [2][8] = {"Mostafa", "Mohamed"};
```

(array of arrays لـ 2D)

* لوعاين اعل array بـ Point بـ Pointer لـ 8 مكعبات اولى كلها الـ size بتاعها 8.
 يعني في المثال الـ first دا لوعاين بـ pointer هندا يشاور على اول
 عنصر الـ array وعايز اعلم انه ++ يشاور على تالي عنصر وهو
 تالي string يعني كلها تتحرك 8 الميموري بمقدار الـ size الـ 8
 محددة

```
char (*ptr)[8] = NULL;
```

* ptr is a pointer point to array → length 8 element. each
 element is a char

```

ptr = Names-2d;
printf ("%s\n", ptr); /* Mostafa */
ptr++;
printf ("%s\n", ptr); /* Mohamed */

```

* هنا خالصت الـ Pointer (بيانو على أول عنصر (عنوان أول حرف) ولما أحضنته طبع أول string في المايكرو (٤٥) وقف.

* لآنني أخذت ptr++ البوشر إلهاً إلى الميموري 8 بایت بس لأنه أنا آلة - محمد الذي size الا size الـ string فيه.

* طبع - بعد ١٥ صبح من بعد الزباده ان حملت فطبع تأكيد عنصر الarray في المايكرو (٤٥)

EX:-

```

unsigned int Numbers[2][3] = { {11, 22, 33},
                                {44, 55, 66} };

```

```
unsigned int (*ptrNumbers)[3] = Numbers;
```

```
printf ("value = %i\n", *ptrNumbers);
```

* دى كلها هتطبع عنوان أول عنصر في الـ array (عنوان الـ 11) مجب لو عايز اطبع الـ 11 فلتدعها ١٢

- ①→ printf ("value = %i\n", *(*ptrNumbers));
- ②→ printf ("value = %i\n", *ptrNumbers[0]);
- ③→ printf ("value = %i\n", *(ptrNumbers[0]));

```
ptrNumbers++;
```

* مطلب لأتاعيل

* إخالص دى لوحدي بالـ cout الـ printf هو فيك دول وحياتهem حـتـتـ الـ ++
* يطبع ٤٤ لأن أنا أحـد عـقـارـ الـ زـيـادـةـ بـتـاعـ الـ بوـشـرـ الـ دـوـ
* ٣ * ٤ = ١٢ byte \leftarrow ٤ bytes element كل element تـانـهـ الـ int4

* لوكايز المربع الـ 55 المستخدم الـ 1؟ ← offset

`printf("Value = %i\n", *(ptrNumbers[0]+1));`

طبعاً كذا لو ادور على عنوان 44 يعني
`ptrNumber[0] → 44` عنوان الـ 44
 فلما اجمع عليه واحد هيزيد بقدر 4 وباقيه مطبع
 dereference ← * 55 لانه غ

* علماً لوكايز علىها بالشكل دا

`printf("Value = %i\n", *(ptrNumber+1));`

دى هنا هنوجي نفس وظيفة الـ ++ هيزيد بقدر 12 byte

* طب لوكايز جاء بالشكل دا

`printf("Value = %i\n", *ptrNumbers[0]+1);`

خالد دي هو يستوف الـ ptrNumbers كان يستاورد على عنوان 45
 ويجيب عنوان القيمة المخزنـة في العنوان دا وينزول على العـدة
 وليست كان يستاورد على 44 هيزيد بقدر 1 فهو مطبع

Access 2D array of characters using the pointer to the array:-

`char (*ptrName)[Array_1D_size][Array_2D_size]=NULL;`

↓ ↓ ↓

Pointer اسم الـ strings بـ size الـ strings

Ex:-

```
char Names_2D[2][8] = {"Mostafa", "Mohamed"};
```

char (*ptrNumbers_2)[2][8] = Names_2D;

is a pointer point to array of two dimensional of two element. each element is array of 8 elements from kind character.

لما ذكرنا مخزن عنوان أول حرف لـ ② pointer لـ ① pointer لـ string والـ 8 هو الـ length من المجموعة من المعرفة

Two dimensional to pointer لـ ① pointer لـ ② pointer لـ string

لـ ① pointer لـ ② pointer لـ string

printf ("%c\n", Names_2D[0][0]); array بـ الاستخدام لـ

printf ("%c\n", *(ptrNumbers_2[0][0])); pointer بـ الاستخدام لـ

array لـ ① pointer لـ ② pointer لـ string

printf ("%c\n", *(ptrNumbers_2[0][0]+1)); first character of Mohamed

printf ("%c\n", *(ptrNumbers_2[0][1])); second character of Mohamed

1 < offset last character of Mohamed

Access array of pointer to string

1) Pointer to the 1D array.

```
char *Names[ARRAY_2D_SIZE] = {"Mostafa", "Mohamed"};
```

```
char *Names[2] = {"Mostafa", "Mohamed"};
```

Pointer to char (هذا يشير إلى أن كل عنصر في المảng هو مассив من نوع character)

كل بولندر يحتوى على عدد وان أول حرف من كل عنصر

Pointer اى ال array & Point اى pointer اى ال array
لذلك + سارف array بس ال pointer اى الشط انه كل العناصر
لديهم نفس المتساوية size (أي عدد الحروف)

```
char *(*ptr)[8] = &Names;
```

char *ptr[8] ← array & Point ← pointer لـ اى العناصر

printf("%s\n", (*ptr)[0]); ← element اطبع أول عنصر ← pointer لـ اى العناصر

[1] خاتمة

2) Pointer to Pointer

double pointer (إذا استخدمناها فالفارق تدريسي)

```
char **ptr_name = &Names;
```

```
printf("%s\n", ptr_name[0]); ← اطبع أول عنصر
```

Recursion in C

- Recursion is a process in which function call itself and the function that calls itself directly or indirectly called a recursive function.

* الريكتيون هي عملية في которой في المقدمة نفسها أو في مقدمة أخرى.

- A recursive function calls itself so there can be several numbers of the recursive call.
- So the recursive function should have the termination condition to break the recursion.
- If there is not termination condition in a recursive function, a stack overflow will occur and your program will crash.
- Recursive functions are useful to solve many mathematical problems, like calculating the factorial of a number.

Example:-

```
void function(int Parameter)
{
    function(Parameter);
}
```

الريتوريون من الحالات الادارة التعامل معها اخراج مجال الديناميكي لكن كل هرة المانعشن يستغرق نفس ما ينبع منها كل call وبالتالي لو متغير متغيرات العناوين محجز مساحة crash stack وبالتالي البرنامج يحصل stack overflow

Recursion الامر condition فيه تكون فيه Recursion لأنها stack overflow دلائل على ان دفعات المانعشن

Direct vs indirect

direct recursion:- is a function that is called itself.

```
Void Function (int Parameter)
{
    /* code */
    if (condition)
    {
        function (Parameter);
    }
    /* code */
}
```

indirect recursion:- A function is called indirect recursive if it calls another function and the called function call the calling function.

Void Function_A (int Parameter)

{

 Function_B (Parameter);

}

Void Function_B (int Parameter) {

 Function_A (Parameter);

}

* Factorial Example :-

unsigned int Get_Factorial (unsigned int Number)

{

 unsigned int Result = 1;

 if (Number == 0)

{

 Result = 1;

 } else {

 Result = Number * Get_Factorial (Number - 1); } }

 return Result;

}

int main () {

 unsigned int LResult = Get_Factorial (5);

 printf ("LResult = %i \n", LResult);

}

create ← frame \rightarrow *
 Terminate ← \rightarrow حوالى واحد

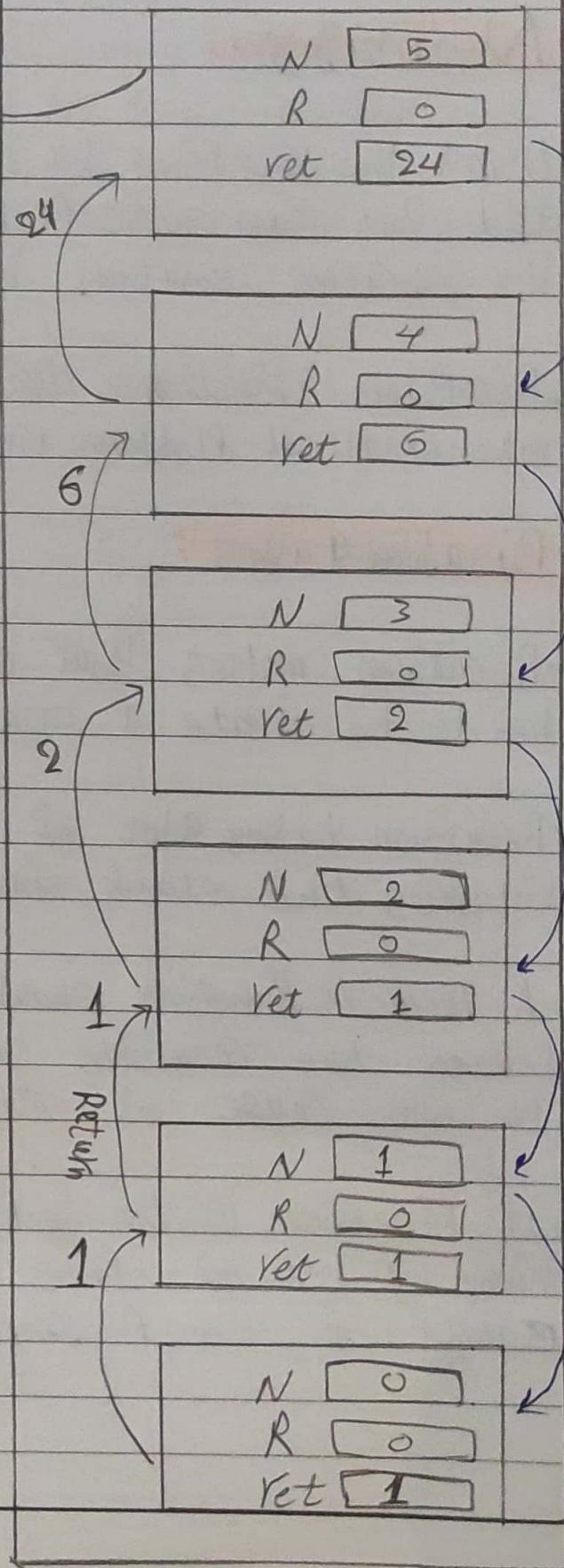
* stack Frame *

$R \leftarrow 120$

لـ N يخزن الـ R \rightarrow *

Recursive function

اعادة بالشكل \rightarrow وصول باخطوات
ما افراد المدخلات



Advantage and Disadvantage of Recursion in C

Advantages:-

- Using the recursion you can make your code simpler and you can solve problems in an easy way while its iterative solution is very big and complex.
- Recursion functions are useful to solve many mathematical problems, like calculating factorial of a number.

Disadvantages

- Recursion makes your code slow because each time needs to create a stack frame for the function call.
- Recursion takes a lot of stack memory because it requires the stack space in every invocation.
- A recursive function should have a base condition to break the recursive calling of the function either it will cause of stack overflow.
- if recursion is too deep, then there is a danger of running out of space on the stack and it can cause of the program crashes.

recursion**VS****iteration** → Loops

- Due to the overhead of maintaining the stack usually, recursion is slower as compared to the iteration.
- Usually, recursion uses more memory as compared to the iteration.
- Sometimes it is much simpler to write the algorithm using recursion as compared to the iteration.