

Lec 11

Pointer Arithmetic:-

- A Pointer in C is an address, which is a numeric value

Example 1:-

```
unsigned int NumberOne = 55;
```

```
unsigned int *ptr = &NumberOne;
```

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

```
ptr = ptr + 1;
```

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

```
ptr = ptr + 1;
```

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

* في المثال دا لما أنا بعمل operation على ال Pointer Increment بمقدار 1
هنا مقدار الزيادة في البوشر هيعتمد على نوع الداتا تايب ال يشاور
عليها ال Pointer

* يعني هنا هنفترض إنه ال int يساوي 4 bytes بمقدار الزيادة
في ال address هيساوي 4 bytes لو char هيساوي 1 byte بمقدار الزيادة في
العنوان بمقدار 1 byte

* عشان كذا بنقول إنه ال size بتاع ال data types و ال Pointers
ههمين جداً في الجزء دا ← Pointer Arithmetic

* مقدار الزيادة أو النقص في ال Pointer يزيد أو ينقص بمقدار
ال Pointer يشاور عليه

Example 2 :-

```
unsigned int Numbers[5] = {11, 22, 33, 44, 55};
```

```
unsigned int *Ptr = &Numbers[0];
```

```
→ printf("Ptr = 0x %X - Value = %i \n", Ptr, *Ptr);
```

```
Ptr += 1;
```

```
→ printf("Ptr = 0x %X - Value = %i \n", Ptr, *Ptr);
```

```
Ptr += 1;
```

```
→ printf("Ptr = 0x %X - Value = %i \n", Ptr, *Ptr);
```

* المثال هنا يوضح على فكرة ان ال Pointer يزيد بمقدار حجم ال Data
 تايب ال Data و ثانياً يعني هنا في أول printf صيغتنا تكون أول عنصر
 والعنصر الثاني ونفس الكلام مع ثاني وثالث printf لأنه ال Data
 في ال memory العناصر يتابعها ببقية ال memory ورايعونها وكل العناصر
 حجمها متساوي

* LL أجب استخدم Pointer Arithmetic مع ال array لدرج ال Pointer
 يكون من نفس نوع ال Data type يتبع ال array

* we cannot use the multiplication or division operator with the pointer *

* صيغتنا تستخدم الضرب والقسمة مع ال Pointer

لوحاولت استخدمه هيفشلني error

* هل يتفق أوجه اثنين Pointer من بعض؟

- ← أيوا يتفق أوجه اثنين Pointer من بعض
- ← يستخدم التكنيك دا لوعايز أعرف عدد ال bytes الين Pointers
- ← يستخدمه مع ال array مند لولمحت آخر عنصر ناقص أول عنصر هيدل عدد العناصر لا في ال array لأنه معروف إن ال array العناصر بتاتها مخزنة ورا بعض في الذاكرة

* هيفتحش أجمع ولا تضرب Two Pointer هيسيلر ← error

* يتفق أقران بين Two Pointer ببعض ويفضل ياتونا الإثنى بيصرو على نفس الحاجة أو يشاورو على نفس ال array مثلاً

* هل يتفق اعمل Arithmetic operation على Void Pointer؟

نعم يتفق اعمل بس لازم اعمل EXPLICIT casting ل Pointer عشان احده له هو يتعامل مع أنص نوع داتا تايب وكلام byte في الذاكرة

Void Pointer → EXPLICIT casting

Example 3:-

```
unsigned int numbers [5] = {0xAABBCCDD, 0x11223344,
                             0x55, 0x66, 0x77};
```

```
void *ptr = NULL;
int main() {
```

```
    ptr = &numbers[0];
```

```
    printf("0x%X\n", *((unsigned int *)ptr)); /* 0xAABBCCDD */
```

```
    ptr += 1;
```

```
    printf("0x%X\n", *((unsigned int *)ptr)); /* 0x44AABBCC */
```

```
    ptr += 1;
```

```
    printf("0x%X\n", *((unsigned int *)ptr)); /* 0x3344AABB */
```

* في المثال هنا معروف أنه لا جينا ال sizeof يتبع ال void في
ال gcc compiler بلع 1 byte

* هنا لما طبعت عنوان أول عنصر لمبعه قادي بس لما طبعت عليه
void pointer increment زاد بمقدار 1 byte

before inc → AA BB CC DD

after inc → 44 AA BB CC

* ال pointer لما زاد بمقدار 1 byte راح خاد عنوان أول byte من ال
عنصر الثاني ونفس الكلام أيضا مع increment التانيه كالمرة
زاد 1 byte بس

Follow Example 3:-

Ptr = &Number[0];

مستسمح بتنا

/* (unsigned int*)Ptr += 1; → Invalid */

Ptr = (unsigned int*)Ptr + 1; ← Void في الـ 4 bytes ←

Printf("0x%x\n", *((unsigned int*)Ptr)); /* 0x11223344 */

Ptr = &Numbers[0];

Ptr = (unsigned short*)Ptr + 1;

Printf("0x%x\n", *((unsigned int*)Ptr)); /* 0x3344AABB */

* هنا لما عدت - explicit casting للبايتز بـ unsigned short وتحت عليه واحد فتر زاد بمقدار 2 bytes فتر زاد بمقدار 2 byte خذت عنوان العنصر بعد 2 bytes الـ 3344

Ptr = &Numbers[0]; /* {0xAABBCCDD, 0x11223344, 0x55, 0x66, 0x77} */

Ptr = (unsigned int*)Ptr++;

Printf("0x%x\n", *((unsigned int*)Ptr)); /* 0xAABBCCDD */

Ptr = (unsigned int*)++Ptr;

Printf("0x%x\n", *((unsigned int*)Ptr)); /* 0x44AABBCC */

* هنا لما عدت - Post increment في Void Pointer وحصلت تأثير لأن الـ Post increment على نفس الحاجة (محصلة زيادة) الـ Pre حصل فيها تأثير ولكن الزيادة كانت بمقدار 1 byte ← Ptr = Ptr + 1 نفس تأثير المعادلة دي

Follow example 8:-

```
Ptr = &Numbers[0]; /* {0xAABBCCDD, 0x11223344, 0x55, 0x66, 0x77} */
```

```
Printf("0x%x\n", *((unsigned int*)Ptr)); /* 0xAABBCCDD.x/
```

```
Printf("0x%x\n", *((unsigned int*)Ptr + 1)); /* 0x11223344.x/
```

```
Printf("0x%x\n", *((unsigned int*)Ptr + 2)); /* 0x00000055.x/
```

```
Printf("0x%x\n", *((unsigned int*)Ptr)); /* 0xAABBCCDD.x/
```

* هنا أنا طبعت قيمة أول عنصر في أول Printf وادي
لكن ثاني Printf طبعت قيمة ثاني عنصر لأن الزيادة حصلت
بعد عليه ال casting فزاد بمقدار 4 byte فطبع قيمة ثاني عنصر

* نفس اللدم أيضاً في ثالث Printf زودت 2 فالبوستر زاد بمقدار
8 bytes فطبع قيمة ثالث عنصر

* آخر Printf بتوضح لي ان عليه الزيادة ال قبلها ما أثرش على قيمة
ال Pointer نفسه وطبع قيمة أول عنصر ال كان يساوي عليه ال Pointer

Example 4 :-

16 bit Data Memory

(RAM)

Address

long X[3] = {1, 2, 3}			0X07FE
long *P = &X;	X[0]	X 5 0001	0X0800
		BABF	0X0802
*P += 4 → *P = *P + 4;	X[1]	2 BEEF	0X0804
P++;		DEAD	0X0806
*P = 0XDEADBEEF;	X[2]	3 F00D	0X0808
P++;		F1D0	0X080A
*P = 0XF1D0F00D;	P	0800	0X080C
P -= 2; → P = P - 2			
*P = 0XBADF00D1;			

* قبل أي حاجة وقبل فتح مسائل أشبه دي إرسم ال Memory

16 bit = 2 byte for 1 address

1- في المثال هنا عالت Pointer بتساو على أول عنصر في ال array

2- زودت على العتبات ال بتساو عليه ال Pointer 4 وبالتالي محتوي أول عنصر

3- زودت ال Pointer بمقدار واحد فزاد بمقدار 4 byte عتبات هتتبع long

4- كتبت في العتوات التاني أو العتبات ال بتساو عليه ال Pointer

5- زودت ال Pointer بمقدار واحد أيضًا

6- كتبت في العتوان الثالث أو العتبات ال بتساو عليه ال Pointer

7- عالت ال Decrement ال Pointer بمقدار فنقص بمقدار 8 byte

فترجعك لعتوان أول عنصر

8- أخيرًا كتبت في عتوات أول عنصر ال بتساو عليه ال Pointer

Post Increment with Pointers :-

Decrement

1) P++

* ولنفترض أن لدينا two Pointer يشاور على أول عنصر في الذاكرة وعملت حاجة بالشكل دا

$Ptr = Ptrnumbers++$

* في الحالة دي يرجع للترتيب تنفيذ العمليات والعمليات الالهتاف الاول
هو ال = assignment وبعدين ال increment

وبالتالي ال Ptr هي فضل مشاورة على عنوان أول عنصر لانه ال Ptrnumbers يشاور على أول عنصر وبعد انتهاء ال statement ال Ptrnumbers هيحصل ال increment على حسب ال data ال يشاور عليه يعني هي مشاورة على ثاني عنصر في الذاكرة.

2) *P++

* لو أنا عندي نفس ال two Pointer في المثال السابق وعملت حاجة بالشكل دا

$Ptr = *Ptrnumbers++$

* عندنا قاعدة بتقول أي ++ هادقة للباينتر ومفيش بينهم فرق
فعملية الزيادة يتم على العنوان نفسه الالهتاف في ال Pointer

* في الحالة دي هو هيخزن محتوى ال Ptrnumber ال يشاور عليها
القيمة يعني هيخزنها في ال Ptr يعني هيخزن value ال Pointer
وبعدين يعمل ال increment لـ Ptrnumbers زي المثال الفات

* لو جيت لمبعت *Ptr هي عمل crash للبرنامج لانه أنا بحاول
ال access بعد مخزن فيه value

[3] $*(ptr++)$

* نفس الحالة السابقة بالفيديو بشرحها

$ptr = *(ptr numbers++)$; مخزن ال value ال يشاور عليها $ptr numbers$ في ال pointer ptr وبعدين ال increment للوينتر $ptr numbers$

$*(ptr++) = *ptr++$

* التلات حالات السابقة كانوا بيعملوا increment لل pointer نفسه بمقدار ال داتا قايب لل pointer

[4] $(*p)++$

* ال $++$ هنا مش ملاصقة لل pointer بقاء عليه الزيادة مش بتحصل على ال pointer نفسه ولكن بتحصل على value ال يشاور عليها ال pointer

$z = (*p)++$; $\rightarrow z = *p$ $\rightarrow *p = *p + 1$

Example:

16-bit Data Memory

$int x[3] = \{1, 2, 3\}$;

$int y$;

$int *p = \&x$;

$y = 5 + *(p++)$;

$y = 5 + (*p)++$;

$\rightarrow y = 5 + *p$

$= 5 + x[0]$

$= 5 + 1$

$= 6$

$\rightarrow y = 5 + *p$

$y = 5 + x[1]$

$= 5 + 2$

$= 7$

Address	
0x07FE	
0x0800	
0x0802	
0x0804	
0x0806	
0x0808	
0x080A	
0x080C	

$x[0]$

1

$x[1]$

2

3

$x[2]$

3

p

0x0800 | 0x0802

y

6

7

Pre increment with Pointers decrement

III ++P

Example :-

```
unsigned int Numbers[5] = {0xAABBCCDD, 0x11223344,
                           0x55, 0x66, 0x77};
```

```
unsigned int *Ptr = &Numbers[0];
```

```
unsigned int *PtrNumbers = &Numbers[0];
```

PtrNumbers = ++Ptr;

* هنا ال Pre عكس ال Post بمعنى

1- أول حاجة هيعمل increment ال Ptr وبالتالي Ptr بقا يساوي على تاني عنصر في الذاكرة.

2- هيتخزن عنوان تاني عنصر ال Ptr في PtrNumbers وبالتالي PtrNumbers < Ptr يساوي على تاني عنصر في الذاكرة.

3- هنا حصل ال increment الأول وبعدين ال assignment

على عكس ال Post حصل ال assignment الأول وبعدين ال increment

2) *++P

لو عندك نفس المثال الفات بس غيرت

Ptr = *++PtrNumbers;

* هنا أنا عند ال operator ال هينفذ الأول على حسب قواعد ال Precedence

1- ال PtrNumbers هيعمله increment وبالتالي هو كدا بقا يساوي على تاني عنصر في ال array

2- علامة ال * ← ال reference ← تتر علامة ال = يعني كدا هتخزن ال value ال يساوي عليه البوينتر ال PtrNumbers ال كان على كذا

Ptr = 0x11223344;

٣. وبالتالي أنا خربت قيمة ثاني عنصر في array في ptr

٤. لو أنا حاولت اعمل access لل ptr * ← اعمل printf هذا البرنامج
هيجعله crash لأنه أنا حاول اعمل access ل invalid address

[3] *(++P)

بتؤدي نفس التأثير السابقة مفسر تغيّر

* ++P = *(++P)

[4] ++(*P)

ptr = ++(*ptrNumbers);

* في الحالة دي ال ++ مفسر ملاصق لل Pointer وبالتالي مفسر هيجعل
increment في Pointer وتنت ميجعل ++ لا بين ()

* هنا عندي 3 operators ←

١- هيجعل increment لل value ال يشاور عليها ptrNumbers
لأنه في علامة * dereference

٢- ولنفترض إنه كان يشاور على أول عنصر في array
ال في المثال السابق ال هي ← 0xAABBCCDD فلما حصلها
زيادة بمقدار واحد ال value هتبقى كذا ← 0xAABBCCDE

٣- تير بعد ذلك هيجعل عليه ال assignment = وبالتالي
هتخزن ال value ال 0xAABBCCDE في ptr ← value في Pointer

٤- وأيضاً قيمة أول عنصر أصبحت 0xAABBCCDE

* لو حاولت اعمل access لل ptr * البرنامج ← crashed ← invalid address

Example :-

16-bit Data Memory

int x[3] = {1, 2, 3}

int y;

int *p = &x;

y = 5 + *(++p);

y = 5 + ++(*p);

→ y = 5 + x[1]

= 5 + 2

= 7

→ y = 5 + ++x[1]

= 5 + 3

= 8

		Address
		0x07FE
x[0]	1	0x0800
x[1]	2 3	0x0802
x[2]	3	0x0804
p	0800 0802	0x0806
y	7 8	0x0808
		0x080A
		0x080C
		0x080E

Modify the Pointer itself :-

Pre-inc : *(++p) or *++p or ++p

Post-inc : *(p++) or *p++ or p++

Modify the Value Pointed to by the Pointer :-

++(*p) and (*p)++

Effect of const qualifier on Pointers :-

* ذكرنا قبل كذا ان الـ `const` variable global variable مقررش اغير في قيمة أبدًا ولكن الـ `const` local variable اقدر اغير في قيمة بطريقة غير مباشرة لذته بيقام مجوز في سكتش الـ `stack` وبالتالي اقدر اغير في قيمته لكن طريق الـ `Pointer`

/* Global Area */

`unsigned int NumberOne = 44;`

→ `const unsigned int *ptr = &NumberOne;`

/* `ptr` is a pointer, points to const data */

/* `ptr` can't change the `NumberOne` value */

* لو أنا عندي `Global Variable` و `const Pointer` يسا ورجع الـ `variable` دا هنا أنا مقررش اعدل في قيمة الـ `variable` من خلال الـ `Pointer`

* `ptr = 66` error

* لو أنا محتاج الـ `Pointer` مغيرش في محتوى الـ `data` هكتب قبل الـ `Pointer` ← `const`

* هكتب الـ `data` أيضًا بطريقة ثانية

→ `unsigned int const *ptr = &NumberOne;`

`const unsigned int *ptr = &NumberOne;`

الـ 2 statements ليه بعض مفيش اختلاف ويفضل استخدم الـ `const`

* ال Pointer القوائم مملكت اخلية يساور على أي Variable تاف
ليس لازم يكون من نفس نوع الداتا كاي بتاع ال Pointer
وبالتالي أنا مقدرش اعدك في قيمة ال Variable دا أبدأ
من خلال ال Pointer

* لو أنت بتاعي Variable لفانكش وال Variable دا أنت
من محتاج تغير فيه بالخصوص ال Pointer حاول تلي ال Pointer
يساور على Const data

```
unsigned int *const ptr3 = &NumberOne;
```

/* ptr3 is a const pointer, points to data */

/* ptr3 can't be changed, but we can change the data */

* في الحالة دي ptr3 ← const pointer مقدرش اغير العنوان بتاعي
ولكن ptr3 يقدر يغير الداتا ال بيص عليها

* النوع دا من ال Pointer لازم وأنا بعمله definition اعلاه
initialize (لأنه لو معملناوش initialization من مقدرا غير
في ال Value المتغيرة جواه لأنه هو Const ولوجيت استخدمه
بعد كدا هيفعل error نشان كذا لازم اعلاه initialization

```
const unsigned int *const ptr4 = &NumberOne;
unsigned int const *const ptr4 = &NumberOne; } ==
```

/* ptr4 is a const pointer, points to const data */

* ptr4 ← مقدرش اغير ال memory location ال بيص عليها ولا اقدر استخدمه
في ان اغير الداتا ال بيص عليها