# Lec 10

## Pointers :-

### What is a Pointer in c?

• A Pointer is similar to a variable but the deference is that :-

1- Pointers are store the addreess of a location in memory.
2- variable stored the value.

• in other words, we can say, a Pointer is used to reference a location in memory.

• Like any variable, You must declare a Pointer before using it.

### Syntax :-

$\longrightarrow$ Asterisk (*) unary operator

```
Data_type  *Pointer_name
Data_type*  Pointer_name
Data_type * Pointer_name
```

Data_type Must be the same type that the Pointer will Point to.

Example :-

```
unsigned int *Ptr ;
```

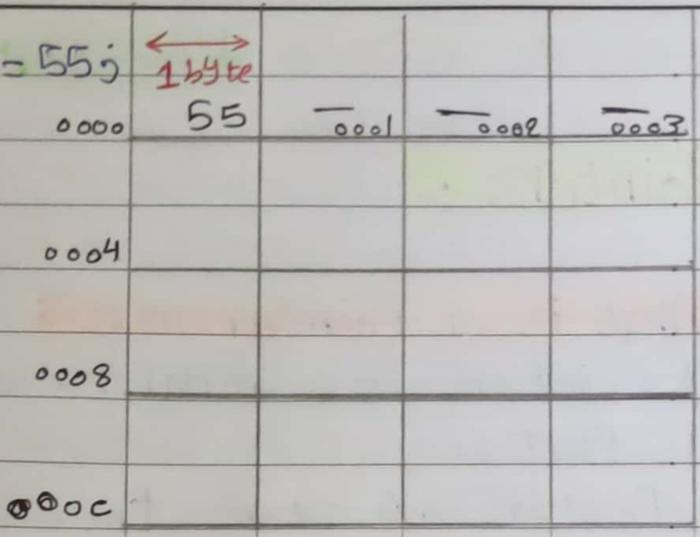
▸ Ptr is a Pointer Point to data → unsigned int

معناه إن Ptr ← Pointer ← بيتخزن فيه عنوان من نوع unsigned int

الموضوع: *هفترض إن دا شكل الميمورى        التاريخ: / /

unsigned int NumberOne = 55;

unsigned int *Ptr ;



* كل byte ع الميمورى ليها (unige address) يعنى ليها عنوان خاص بيها

الفرق بين الـ Variable ( Pointer ؟

١- الـ بوينتر بيبقا قبلها ( * ) إنما الـ Variable مفيش
٢- الـ Variable بخزن فيه قيمة Value
٣- الـ Pointer بخزن فيه عنوان address

• عشان أخزن عنوان Variable ع بوينتر لازم الـ Variable و الـ Pointer يبقو من نفس نوع الـ Data type

• إزاى اخزن عنوان NumberOne ع Ptr1 ؟!

→ Address of operator (And)

Ptr1 = &NumberOne;

← كدا هو خزن عنوان NumberOne ع Ptr1 لو حبيت أطبع printf

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

← هيطبع الـ First Address لـ First byte بتاع NumberOne وليكن مثلا عالشكل الـ فوق هيطبع 0000

%X ← يطبع الـ address بالـ hex ← upper case    %x ← يطبع الـ address بـ الـ hex ← small case

الموضوع: ________ التاريخ: / /

```
Printf("Numberone Value = %i \n", *(&Numberone));
```

* هيروح لعنوان Numberone في الميموري وشوف العنوان دا محتواه إيه كقيمة ← بسبب ← ⊛ dereference operator

* نفس الـ Result برضه لو عملت كدا *(Ptr1)
هيروح يشوف الـ Pointer دا بيشاور على مكان إيه في الميموري ويجيب محتواه

---

* ممكن اعمل override على الـ Pointer
بس لازم الـ variable من نفس نوع الداتا تايب للـ Pointer

```
Ptr1 = &Numberone;
//codes
Ptr2 = &Numbertwo;
```

---

```
unsigned int *Ptr, Number_1, Number_2 = 3;
```

Pointer

Variable Not initialized

Variable initilized

* يفضل إن أنا معملش حاجه زى كدا و اعمل كل وحده في سطر لوحدها

التاريخ: / /    الموضوع: ______

```
unsigned int Numberone = 55
unsigned int *Ptr;

Ptr = &Numberone;

*Ptr = 66;
```

* ممكن اعدل في قيمة ال Variable من خلال ال Pointer باستخدام
* ← dereference operator

لو جيت طبعت قيمة Numberone هلاقيها ب 66

---

```
Number1 = 2;          unsigned int *ptr1, *ptr2;
Number2 = 3;          ptr1 = &Number1;  ptr2 = &Numbertwo;
```

← هتروح على عنوان Number2 وتجيب محتواه هنا وكذلك مع Number1

```
unsigned Result = *(&Number1) + *(&Number2);

          Result = *(ptr1) + *(ptr2);
```

← نفس السطرين هيعملو نفس الوظيفة ويجمعو (Number1 و Number2 ويخزن الناتج في ال Result

---

```
unsigned int Numberone = 0X 11223344;

unsigned char *Ptr1 = &Numberone;
unsigned short *Ptr2 = &Numberone;
unsigned int *Ptr3 = &Numberone

Printf("value = 0X%X\n",*Ptr1);
Printf("value = 0X %X\n",*Ptr2);
Printf("value = 0X %X\n",*Ptr3);
```

output :-

```
Value = 0X44
Value = 0X 3344
Value = 0X11223344
```

الموضوع: ____ التاريخ: / /

* في المثال السابق أول printf طبعت 0x44 لأن أنا عامل البوينتر من نوع unsigned char وبالتالي هو ميقدرش يشوف في الميموري غير 1 byte بس عشان من نوع unsigned char وبالتالي طبع 1 byte بس الهي 44

* أما في ال printf التانيه طبع 0x3344 لأن البوينتر الأنا بطبعه من نوع unsigned short وبالتالي هو ميقدرش يشوف غير 2byte بس في الميموري عشان كدا طبع 0x3344

* أما ال printf الأخيرة طبع القيمة كاملة لأنه البوينتر من نوع unsigned int وهو نفس نوع ال Variable المتخزن فيه القيمة الهو من unsigned int حجمه 4 byte

* عشان كدا بنقول لازم ال Pointer من نفس نوع Data type بتاع ال Variable الهيشيل عنوانه

---

* كل مرة بأنشاء فيها Pointer بيتحجز له مساحة في الميموري ويبقى ليها عنوان ← أي Pointer ليه عنوان في الميموري
* لو عايز اعرف ال Pointer بيتحجز له كام byte في الميموري هستخدم sizeof(Pointer Name)

* حجم ال Pointer دايما في الكومبيلر ثابت أي كان نوع ال Data type ال بيشاور عليها

* حجم ال Pointer بيعتمد على ال ← compiler
↳ architecture بتاع الجهاز الهيرن عليه

الموضوع: ______ التاريخ: / /

* حجم الـ Pointer عادةً بيساوي حجم الـ address bus الخاص بالـ architecture بتاع الجهاز

---

```
unsigned int *Ptr;          // Global Pointer Not initialized
```

* الـ default ← address ← Zero
* لو حاولت أعمل access للقيمة اللي بيشاور عليها البرنامج هيعمل كراش crash لو طبعها مثلاً

```
int main(){
unsigned int *Ptr1;     // local Pointer Not initialized
Printf("address = 0x%x",Ptr1);
Printf("\n Value = 0x%X\n",*Ptr1);
}
```

* في كل مرة هعمل Build & run هيطلع لي address مختلف وقيمة مختلفة

* النوع دا من الـ Pointers اللي هو Not initilized سواء كان local أو Global اسمه ← wild Pointers

- The behavior of uninitialized Pointer is undefined because they Point to some arbitary memory location.

* عشان كدا مينفعش أعمل Pointer ← Not initilized ← أبداً

عشان أتغلب على مشاكل ال Wild Pointer :-

١- وأنا بعرف ال Pointer اديله address ← Variable ← هيتعامل معاه

٢- هعطي ال Pointer قيمة NULL ← معناه Zero

NULL = 0

$\leftarrow$ Recommended

```
unsigned int *ptr = NULL;
```

*ممكن اشيل NULL واحط Zero بس يفضل احط NULL لانه تستعمل أكثر مع ال Pointers

* لما أنا كتبت NULL لل Pointer كدا أنا أصبح نوع تاني من ال Pointers اسمه NULL Pointer's

- A NULL Pointer is a Pointer that Points to nothing.

**why do we need a NULL Pointer?**

- A null Pointers prevents the surprising behavior of the Program.

- if you forget to assign a valid address to the Pointer at the time of decleration and later you will try to access the Pointer, the Program behavior may be undefined

**what does undefined mean?**

it means your program might work as your desire or it might be crashed

الموضوع: التاريخ: / /

*سوف يتم شرح هذا الجزء بالتفصيل في المحاضرات القادمة*

*dynamic memory allocation:-

الهدف منطاوان أنا بحجز مساحة في الميموري في سكشن ال heap أثناء وقت run time .

*function malloc

Pointer = malloc ( user_size );

*الفانكشن دي بديلها عدد ال bytes اللي هتحجزها في الميموري في سكشن ال heap وهترجعلي عنوان أول byte من ال bytes الحجزتهم.

*يعني لو ال user_size بيساوي 20 هتروح تحجز 20 bytes في الهيب وترجع عنوان أول byte والعنوان دا هستقبله في Pointer عشان مفيش غير ال Pointer اللي بتخزن عنا وين

* فانكشن ال malloc ممكن وأنا بحجز مساحة في الميموري ميكونش في مساحة اساسا في الميموري ففانكشن malloc هترجع NULL

*function free

free( Pointer )

*الفانكشن دي بديلها اسم ال Pointer بس وهو بيمسح ال pointer بيشاور عليه في ال heap

* يعني لو انت حجزت 20bytes مثلا بال malloc وجيت مسحتهم بال free كدا أنا مبقاش عندي bytes محجوزة في الميموري

الموضوع: ________ التاريخ: / /

* ليس ال address ال رجع من malloc لدا مبقاش Valid
وبالتالى ظهرلنا نوع جديد من أنواع ال Pointer **Dangling Pointers**

- Dangling Pointers arise when the referencing object is deleted or deallocated, without changing the value of the Pointers.

- When we try to access dangling Pointer it craches the Program.

- We can solve this Problem using the NULL Pointer.

* مينفعش بعد معملت free لـ Pointer استخدمه ولكن يفضل إن اساوى ال Pointer دا بـ NULL وبكدا أنا تغلبت على مشكلة ال Dangling Pointer

* ال Dangling Pointer عموماً معناه إن دا Pointer كان متخزن فيه عنوان لـ location ع الميمورى كان موجود ولكن كان موجود لفترة زمنية معينة و ال location دا اتمسح من الميمورى ولكن العنوان لا اتحجز ع الميمورى لسه موجود ع ال Pointer وده Valid address يعنى مينفعش اعمله access لازم اعمله NULL الأول عشان ميحصلش مشاكل ع الميمورى أو البرنامج

## Why do we need a NULL Pointer?

- So using the NULL Pointer you can avoid the surprising behavior of your c Program.

- The behavior of the NULL Pointer is defined by C standered, if you try to dereference the NULL Pointer you will get a segmentation fault.

## Conclusion :-

- You must initilize the Pointer with NULL.
- You must validate the Pointer before its use ← يعني اعمل check على البوينتر قبل ما تستخدمه

```
unsigned int *Ptr =NULL;
      // codes
① if (Ptr != NULL)                ② if(NULL == Ptr)
  {                                  {
    /*valid address */
  } else{                            }else {
                                       /*valid Address */
  }                                  }

③ if(Ptr){
    /* valid address */
  }else{

  }
```

* الـ size بتاع الـ NULL بيعتمد ع size بتاع الـ Pointer
وممكن اعرف الـ size باستخدام sizeof();

what if we need to pointer point to any type?

To resolve the above problem, c language introduces a generic type of pointer (Void Pointer) that can store the address of any type

What is Void Pointer in c ?

- A Void Pointer in c is called a generic Pointer, it has no associated data type.
- it can store the address of any type of object and it can be type casted to any type.
- A Void Pointer declaration is similar to the normal Pointer, but the difference is that instead of data types we use the Void keyword.

* لو انا عندي أكثر من نوع داتا تايب وعايز اشاور عليهم بـ Pointer واحد ع الحالة دى بستخدم Void ← Pointer بيشاور على أى نوع داتا تايب

Syntax:-

```
Void *Ptr = NULL;
```

* الـ Void Pointer مينفعش اعمله direct access → error

* عشان اعرف اتعامل مع الـ Void Pointer لازم استعمل طريقة الـ EXPlist type casting

*((unsigned int *)Ptr)

على حسب نوع الـ Variable من أنهي داتا تايب

← بستخدم لو عايز اعمل printf
← أو اعدل ع قيمة Variable

الموضوع: ______ التاريخ: / /

```
unsigned int Numbr1 = 55;
unsigned short Number2 = 66;
unsigned char Number3 = 77;

void *Ptr = NULL;

Ptr = &Number1;
printf("Number1 value = %i \n", *((unsigned int*)Ptr));
```

* هنا أنا بقول للPointer إنه في اللحظة دى عبارة عن Pointer من نوع unsigned int يعنى بعرفه هو هيشاور على كام byte في الميمورى

Void Pointer

* لو جيت اشوف ال sizeof بتاع Ptr هلاقيه نفس الPointer العادى لأن الsize بتاع الPointer ثابت

* لو جيت اشوف برضه ال sizeof بتاع void دى لما بتعتمد على الكومبيلر ← في ال Gcc ← كان 1byte

void → incomplete data type

لو عايز اغير قيمة Number3 مثلا

```
Ptr = &Number3;
*((unsigned int *)Ptr) = 22;
```

الموضوع: ____ التاريخ: / /

## Call by Value Vs Call by reference

### -by value

Example:-

```
Void swap(int num1 , int num2);
int Numberone = 55;
int Numbertwo = 66

int main(){

printf("Numberone=%i \t Numbertwo =%i",Numberone,Numbertwo);
 swap(Numberone , Numbertwo);
printf("Numberone=%i \t Numbertwo =%i",Numberone,Numbertwo);

}
Void swap(int num1 , int num2)
{
 int temp=num1;
  num1 =num2;
  num2 =temp;
}
```

* هنامحصلش swap بين الرقمين
لأن أنا هنا بعتبر خدت نسخة من القيمة
بتاع المتغير وليس المتغير الأصلى وبالتالى
أى تأثير هيحصل على النسخة ال أخدتها

* أى تغير هيحصل داخل ال فانكشن مش هيأثر
أو هينعكس على البارمتر بتاع الفانكشن

Output:-

```
Numberone =55        Numbertwo = 66
Numberone =55        Numbertwo = 66
```

- by reference :-

Example :-

```
int Numberone = 55;
int Numbertwo = 66;

Void swap ( int* PtrNum1 , int*PtrNum2);

intmain(){

printf("Numberone = %i \t Numbertwo = %i \n",
        Numberone, Numbertwo);

 Swap(&Numberone, &Numbertwo);

 Printf("Numberone = %i  \t Numbertwo = %i \n",
        Numberone, Numbertwo);


Void swap( int*PtrNum1 , int *PtrNum2){

 int temp =*PtrNum1;
*PtrNum1 = *PtrNum2;
*PtrNum2 = temp;
}
```

* هنا حصل swap بين المتغيرين لأنه هنا أنا باصيتله عنوان المتغيرات وبالتالى يعتبر أخدت القيمة الأصلية للمتغير وطبعا استقبلته ع pointer عشان عنوان

output :-

```
Numberone = 55   Numbertwo = 66
Numberone = 66   Numbertwo = 55
```

* أى تغيير داخل الفانكشن هينعكس ع الـ Parameter

التاريخ / /   الموضوع

# Comparison from Geeks for Geeks

| Call by value | Call by reference |
|---|---|
| • While calling a function, we Pass the values of variables to it. | - while calling a function, instead of Passing the values of variables, we Pass address of variables (location of variables) |
| - in This method, the value of each variable in The calling function is copied into corresponding dummy varibles of the called function | - in this method, the address of acctual variable in the calling function is copied into the dummy variables of the called function |
| - with this method, the changes made to the dummy variables in the called function have no effect on the values of actual variables in the calling function. | - With this method using addresses we would have access to actual variables and hence we would be able to manipulate them. |
| - We cannot alter the values of acctual variables | - We can alter the values of variables through function calls. |
| - Values of variables are Passed by simple Technique | - Pointer variables are necessary to define to store the address values of variables. |

الموضوع: ________ التاريخ: / /

* ممكن اعمل اكتر من return لأكتر من Value من خلال الParameter
باستخدام الPointer ← call by refrence

Example :-

← لو عايز اعمل square لأكتر من رقم مثلاً

```
int num1   = 2;
int num2   = 3;
int num3   = 4;
               Get_square
unsigned char*(int *Ptrnum1, int*Ptrnum2, int*Ptrnum3);
int main(){
Printf("%i\t%i\t%i\n", num1, num2, num3);
               char
unsigned *Error_status = 0;

Error_status = Get_square( &num1, &num2, &num3);
printf("%i\t%i\t%i\n", num1, num2, num3);

if(1 == Error_status){
 printf("Error !!\n");
 }
}
```

التاريخ: / / **Method 1** الموضوع: ____

```
unsigned char Get_square(int*Ptrnum1, int*Ptrnum2,
                                  int*Ptrnum3)
{
unsigned char Error_RetVal=0;  /*No Error */

if((Ptrnum1==NULL) || (Ptrnum2==NULL) ||(Ptrnum3==NULL))
  {
   Error_RetVal =1;
  }else{
   *Ptrnum1 *=*Ptrnum1;
   *Ptrnum2 *= *Ptrnum2;
   *Ptrnum3 *= *Ptrnum3;
   }
return Error_RetVal;
}
```

* الفانكشن بتاخد ٣ عناوين لـ٣ بوينتر بس لازم تتأكد الأول إنه العنوان الـ جايلها مش NULL عشان ميحصلش Segmentation fault أو بمعنى تانى إنك اعمل Validate لل Pointers

* وعشان أهندل حاجه زى دى إستعملت حاجة اسمها Error status ودى بتعرفنى إذا كان حصل مشكلة أولا

* ممكن اعمل الفانكشن دى بطريقة أفضل ع الصفحة التاليه

التاريخ: / /

# Method 2

الموضوع:

```
unsigned char squareNumber( int *Ptr){
 unsigned char Error_RetVal = 0    /*No Error */

 if (NULL == Ptr){
  Error_RetVal = 1;    /* Error */
 }
 else{
   *Ptr=  *Ptr;

 }
 return  Error_RetVal;
}

unsigned char GetSquare(int*Ptr1, int *Ptr2, int*Ptr3)
{
 unsigned char Error_RetVal = 0; /*No Error */

 Error_RetVal = squareNumber(Ptr1);
 Error_RetVal |= squareNumber(Ptr2);
 Error_RetVal |= squareNumber(Ptr3);

 return Error_RetVal;
}
```

* هنا قسمت الفانكشن الـ فاتت لـ اتنين فانكشن و بيعملوا نفس وظيفه الـ فانكشن الفاتت بس دول أفضل كـ كود ، هنا عرفنه بعمل OR مع الـ فانكشن بحيث لو في مره كانت فيها الـ pointer بيدي إشارة إن في error حصل في أحد الـ calls ليساوي NULL