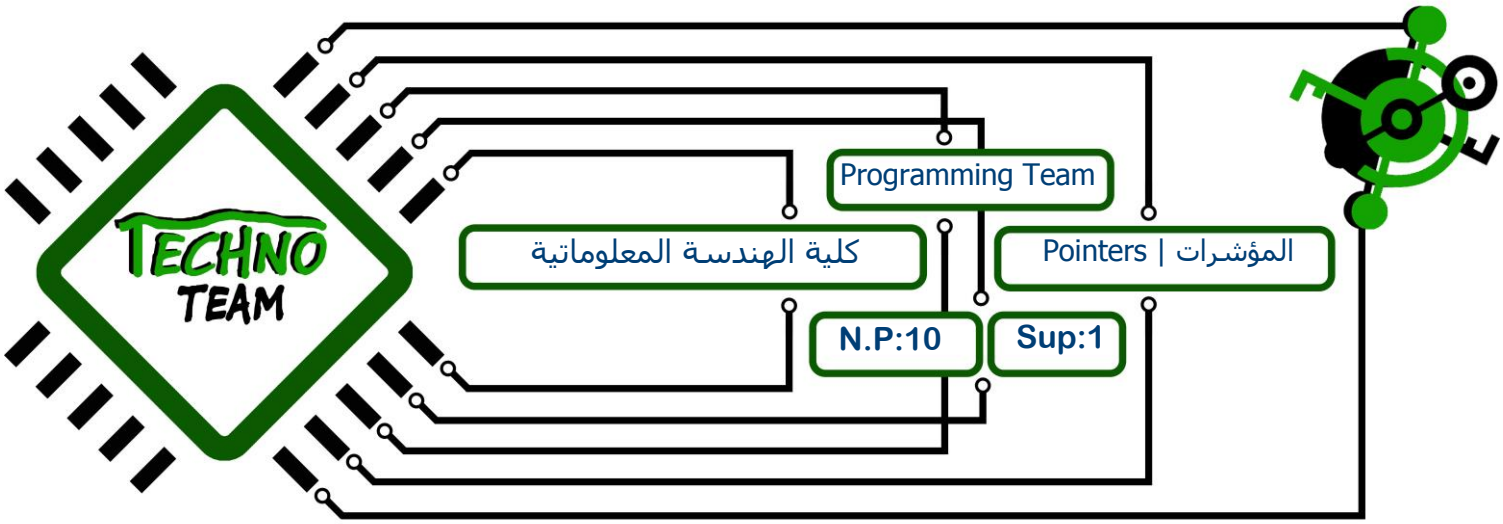
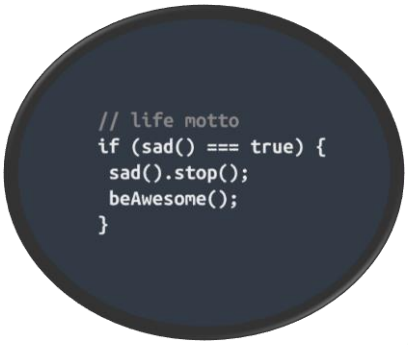




المؤشرات في لغة C :

مقدمة:

تعد المؤشرات من المواضيع الأساسية في فهم البرمجة ولغاتها. وهي كغيرها من عناصر لغة البرمجة، يتطلب استخدامها في البرنامج التصريح عنها أولاً، ثم إعطائها قيماً أولية أو استخدامها لتغيير قيمة المتغيرات.

المؤشر: هو متغير تكون قيمته عنواناً لمتغير آخر في ذاكرة الحاسب.

وتشبه وظيفة المؤشر إلى حد كبير وظيفة عنوان المنزل أو المكتب حيث يستخدم موظفو البريد هذا العنوان لإيصال الطرد إلى وجهته. كذلك يستخدم برنامج الحاسب عناوين المتغيرات المخزنة في الذاكرة للوصول إلى هذه المتغيرات وكتابة أو قراءة أو تغيير القيم المخزنة فيها.

ذاكرة الحاسب هي المكان الذي تخزن فيه المتغيرات , حيث تقسم الذاكرة إلى مواقع تخزينية معنونه أي يمكن الوصول إليها من خلال عناوين هذه المواقع. وهكذا يكون لكل متغير موجود في الذاكرة عنوان محدد يستخدم للوصول إليه.

ملاحظة:

كل متحول له أكثر من مؤشر يمكن أن يؤشر عليه وكل مؤشر يحمل عنوان واحد فقط.

التصريح عن المؤشرات وإعطائها قيماً أولية :

الصيغة العامة للتصريح عن المؤشرات هي:

Variable_Type *Pointer_Name=&Variable_Name;

- حيث Variable_Type : نوع بيانات المتغير الذي يشير إليها المؤشر.
- حيث أن Type من الممكن أن يكون أي نوع من أنواع البينات مثل:
  int, char, float, double, ...
- **Pointer_Name**:هو اسم المؤشر.
- النجمة *: تدل على أن المتغير Pointer_Name سوف يتم فيه تخزين عنوان الذاكرة.

مثال:

```
int  *i_ptr;
float  *d_ptr;
```

يجب إعطاء المؤشرات قيماً قبل استخدامها, وذلك من خلال تشخيص عناوين المتغيرات التي تشير إليها هذه المؤشرات.

ويستخدم المعامل (address:&) لإرجاع عنوان المتغير الذي يشير إليه المؤشر.

فمثلاً إذا كان i متغيراً صحيحاً فإن المعامل &i يرجع العنوان الذي يوجد فيه المتغير i.

مثال:

```
int i=5;
int *i_ptr=&I;
*i_ptr=6;
```

وهنا في السطر الأخير استخدمنا المؤشر * لوضع القيم في المتغيرات بطريقة غير مباشرة

مثال لتوضيح ما سبق:

```
#include<stdio.h>
main()
1.{ int x=5;
2.  int *ptr=&x;
3.  printf("location Ptr=%d",ptr);
4.  printf("\nlocation var=%d",&x);
5.  printf("\nvalue Ptr=%d",*ptr);
6.  printf("\n value var=%d",x); }
```

توضيح البرنامج السابق:

في الخطوة رقم (1) عرفنا المتغير x من النوع int (عدد صحيح).
في الخطوة رقم (2) عرفنا مؤشر من النوع (int)أيضا وجعلناه يؤشر على المتغير x (أي يحمل عنوانه)
في الخطوة رقم (3) استخدمنا التابع printf لطباعة العنوان الذي يؤشر عليه المؤشر (ptr) أي لطباعة عنوان المتغير x
في الخطوة رقم (4) استخدمنا التابع printf لطباعة عنوان المتغير (x)(سوف يطبع عنوان المتغير x بسبب وضع العلامة & قبله عند استخدامه)
في الخطوة رقم (5) استخدمنا التابع (printf) لطباعة محتوى العنوان الذي يؤشر عليه المؤشر(ptr) أي لطباعة قيمة المتغيرx
في الخطوة رقم (6) استخدمنا التابع (printf) لطباعة قيمة المتغيرx

فائدة(1):

(يمكن استخدام المؤشر لتغير قيمة المتغير الذي يؤشر عليه)

مثال:

```
#include<stdio.h>
main()
1. {int x=12;
2.  int *ptr=&x;
3.  *ptr=16;
4. printf("\nx=%d ",x);}
```

التوضيح:

في الخطوة رقم(1) عرفنا المتغير x من النوع int وأعطيناه قيمة ابتدائية(12)
في الخطوة رقم(2) عرفنا مؤشر من النوع int أيضا و جعلناه يؤشر على المتغير x (أي يحمل عنوانه)
في الخطوة رقم(3) استخدمنا المؤشر (ptr) لتغيير قيمة المتغير x أي قمنا بتغير محتوى العنوان الذي يؤشر عليه المؤشر ptr
في الخطوة رقم(4)استخدمنا التابع printf لطباعة القيمة الجديدة للمتغير x أي لطباعة القيمة 16

ملاحظة:

يجب أن يكون المؤشر من نفس نوع المتغير المؤشَر عليه

فائدة(2):(مؤشر يحمل عنوان مؤشر آخر):

يكون من الشكل:

```
TYPE **PTR=&P;
```

حيث :

- 1. **TYPE**:هو نوع المؤشر **(PTR)** الذي يحمل عنوان المؤشر **(P).**
- 2. **(PTR)**:مؤشر يحمل عنوان المؤشر **(P)** يجب أن يسبق ب**.

مثال توضيحي:

```
#include<stdio.h>
main()
{ int a=5;
int *p=&a;
int  **pi=&p;
printf("%d\n",a);
printf("%d\n",&a);
printf("%d\n",*p);
printf("%d\n",p);
printf("%d\n",**pi);
printf("%d\n",*pi);
printf("%d\n",pi); }
```

النتائج التي سنحصل عليها عند تنفيذ البرنامج السابق بالترتيب:

1.	قيمة المتغير a	5
2.	عنوان المتغير a	2686744
3.	قيمة المتغير a	5
4.	عنوان المتغير a	2686744
5.	قيمة المتغير a	5
6.	عنوان المتغير a	2686744
7.	عنوان المؤشر p	2686740

ملاحظات:

1.ليس بالضرورة أن تحصل على نفس قيم العناوين السابقة.

2.اذا أردنا الحصول على عنوان المؤشر p بطريقة مختلفة نكتب:

```
Printf("%d",&p);
```

بعض العمليات على المؤشرات:

يمكن إجراء بعض العمليات على المؤشرات في لغة c شأنها في ذلك شأن أي مكون من مكونات لغة c ومن العمليات الحسابية يمكن استخدام عمليتي الجمع والطرح فقط على المؤشرات, وتعتمد نتائج هاتين العمليتين على نوع البيانات التي يشير إليها المؤشر.
فعلى سبيل المثال لنفترض أنه لدينا مؤشرين كل واحد منهما يشير إلى متغير من نوع بيانات مختلف كما يأتي:

char *ch;

int *num;

ولنفترض أن العنوانين (100 , 200) هما الموقعان اللذان يشير إليهما المؤشران (num , ch) على الترتيب ولنفترض أننا كتبنا:

++ch;

++num;

بعد إنجاز هذه العمليات سوف تزداد عناوين الذاكرة فنجد أن المؤشر الأول سوف ينتقل إلى الخانة التي تلي العنوان السابق أي إلى الرمز الذي يليه لأنه يستهلك بايت واحد فقط من الذاكرة لأنه من النوع Char أما المؤشر الثاني يستهلك 2 بايت فينتقل بمقدار خانتين...
كما يمكن إجراء عملية الإسناد على المؤشرات كونها تشير إلى عناوين متغيرات أخرى فإذا كتبنا p1=p2 فهذا يعني أن المؤشر p1 سوف يشير إلى العنوان الذي يشير إليه p2 .

تذكرة:المصفوفة الأحادية هي مجموعة من المواقع المتتالية المحجوزة في الذاكرة.

أمثلة توضيحية :

:(1)

```
#include<stdio.h>
main()
{
1. int a[5]={1,2,3,4,5};
2. int *p=&a[0];
3. p+=1;
4. printf("%d",*p); }
```

الشرح:	عنوان ثاني عنصر من عناصر المصفوفة(a[1])
1.في الخطوة (2) عرفنا مؤشر من النوع int وجعلناه يؤشر على أول عنصر من عناصر المصفوفة (a[0]) (أي جعلناه يحمل عنوانه). 2.في الخطوة رقم (3) زدنا قيمة العنوان المخزن بمقدار 1 أي أصبح المؤشر يحمل عنوان الخانة التالية في الذاكرة أي	3.في الخطوة رقم (4) استخدم التابع printf لطباعة محتوى العنوان الذي يحمله المؤشر p أي لطباعة القيمة (2).

:(2)

```
#include<stdio.h>
main()
{
1.int a[4]={4,5,6,7};
2.int *p=&a[1];
3.printf("%d",*p);
4.p+=2;
5.*p*=3;
6.printf("\n%d",*p);}
```

الشرح:	
1.في الخطوة (2) عرفنا مؤشر من النوع int وجعلناه يؤشر على ثاني عنصر من عناصر المصفوفة (a[1])(أي جعلناه يحمل عنوانه). 2.في الخطوة رقم (3) استخدم التابع printf لطباعة محتوى العنوان الذي يحمله المؤشر p أي لطباعة القيمة(5). 3.في الخطوة رقم (4) أضفنا (2) الى قيمة العنوان الذي يحمله المؤشر p أي أصبح يحمل العنوان على بعد خانتين متتاليتين من الذاكرة	أي أصبح يحمل عنوان العنصر الرابع من عناصر المصفوفة(a[3]). 4.في الخطوة رقم(5) ضربنا محتوى العنوان الذي يشير اليه المؤشر p بالعدد (3). 5.في الخطوة رقم (6) استخدم التابع printf لطباعة محتوى العنوان الذي يحمله المؤشر p أي لطباعة القيمة(21).

ملاحظة هامة:

يمكن كتابة ال(code) السابق بطريقة أخرى للحصول على نفس النتيجة:

```
#include<stdio.h>
main()
{
1.  int a[4]={4,5,6,7};
2. int *p=&a[1];
3. printf("%d",*p);
4. *(p+2)=*(p+2)*3;
5. printf("\n%d",*(p+2);

}
```

فائدة (3):

إن اسم المصفوفة عبارة عن مؤشر إلى العنصر الأول في المصفوفة وله نفس أنواع عناصر المصفوفة.

فائدة (4):

المصفوفة ثنائية البعد هي مجموعة من المواقع المتتالية المحجوزة في الذاكرة ويمكن أن نستخدم المؤشر مع المصفوفة الثنائية وجعله يؤشر على أحد القيم وسهولة تمريره على جميع العناصر عن طريق زيادة قيمة عنوان المؤشر بواحد فينتقل المؤشر ليؤشر على الموقع التالي الذي يليه.

مثال توضيحي:

لدينا المصفوفة ثنائية البعد **a[2][2]** بحيث:

a[2][2]={{1,3},{0,9}}

أو بالشكل:

a[0][0]=1	a[0][1]=3	a[1][0]=0	a[1][1]=9

كيفية اصطفاف عناصر المصفوفة السابقة ضمن خلايا الذاكرة:

العنصر	الموقع في الذاكرة	المحتوى
	12444	Data
a[0][0]	12445	1
a[0][1]	12446	3
a[1][0]	12447	0
a[1][1]	12448	9
	12449	Data

عناصر السطر الأول

عناصر السطر الثاني

من استخدامات المؤشرات(الاستدعاء بالمرجع):

إذا أردنا أن ندخل مجموعة من القيم إلى داخل تابع ما, كنا نستخدم لذلك المتحولات الوسيطة, و عند استدعاء التابع كنا نرسل القيم التي نريدها عن طريق المتحولات الوسيطة الفعلية, و لكن ما معنى الاستدعاء بالقيمة و الاستدعاء بالمرجع ؟

توضيح:

بفرض لدينا التابع:

```
Int face(int n,int m);
```

متحول وسطي أول

متحول وسطي ثاني

أولاً:الاستدعاء بالقيمة:

مثال عن الاستدعاء بالقيمة:

```
#include<stdio.h>
void sum(int a,int b);

main()
{
int  x,y;
scanf("%d%d",&x,&y);
sum(x,y);
}
```

```
void sum(int a,int b)

{

printf("%d",a+b);

}
```

عند الاستدعاء سيتم إجراء نسخة عن كل من x و y و إرسالها إلى المتحولات الوسيطة a و b الخاصين بالتابع sum و بالتالي لن يتأثر المتغيرين x و y بنتائج عمليات التابع (أي لن تتغير قيمتهما حتى بعد الانتهاء من التابع) لأن المترجم قام بنسخ قيم x و y ووضعها في المتحولين الوسيطين a و b و بالتالي نستنتج أيضاً أنه سيتم تحميل الذاكرة بقيمتين مشابهتين تماماً لقيم x و y .

مثال توضيحي:

```
#include<stdio.h>
void increase(int a);

main()

{
```

```
1.int x=2;

2.increase(x);

3. printf("\n%d",x);

}

void increase(int a)
```

```
{

a+=2;

printf("%d",a);

}
```

توضيح:

في الخطوة رقم (1) عرفنا المتغير x من النوع int وأسندنا له قيمة ابتدائية(2).
في الخطوة رقم (2) استدعينا التابع (increase) والمعرف فيما بعد في البرنامج على أنه يزيد قيمة المتغير المستدعى من أجله بمقدار (2) ومن ثم يطبعه على الشاشة.
في الخطوة رقم (3) استخدمنا التابع (printf) لطباعة قيمة المتغير x

عند تنفيذ البرنامج السابق و بعد استدعاء التابع **(increase)** سوف تطبع قيمة المتغير **x** بعد زيادة قيمته بمقدار**(2)** ولكن في الخطوة 3 عند طباعة قيمة المتغير **x** نجد أنها لم تتغير وسبب ذلك أن قيمة المتغير **x** الأصلية لم تتأثر بل تأثرت النسخة عن قيمة المتغير **x**التي أرسلت الى التابع **increase** ولكن ماذا لو أردنا أن تتأثر قيمة المتغير **x**؟

الحل هو عن طريق الاستدعاء بالمرجع

ثانيا:الاستدعاء بالمرجع:

الاستدعاء بالمرجع : (by Reference)

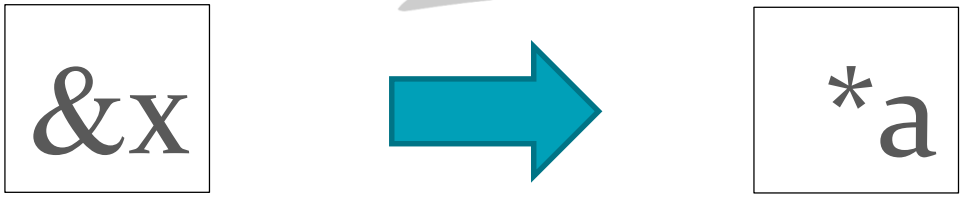
يختلف الاستدعاء بالمرجع عن الاستدعاء بالقيمة بأنه هذا الاستدعاء (بالمرجع) سيقوم بالتعامل مع عنوان المتحول الوسيط الفعلي في الذاكرة(x من المثال السابق), و بالتالي إذا حدثت أي تغييرات على المتحول الوسيط الشكلي(a من المثال السابق) فإن هذا التغير سيطرأ أيضاً على المتحول الوسيط الفعلي(x).. أي أن قيمة المتحول الوسيط الشكلي(a) ستعطى للمتحول الوسيط الفعلي (x)

اعادة المثال السابق(مع استخدام الاستدعاء بالمرجع):

```
#include<stdio.h>
void increase(int *a);
main()
{
1.int x=2;
increase(&x);
printf("\n%d",x);
}
void increase(int *a)
{
*a+=2;
printf("%d",*a);
}
```

توضيح:

عند الاستدعاء بالمرجع نرسل عنوان المتغير المراد التعامل معه ويجب أن يقابله في التابع مؤشر من نفس نوعه.



عند تنفيذ هذا البرنامج نحصل على النتائج التالية على الترتيب.

1.	4
2.	4

هذا يعني أن قيمة المتغير x قد تغيرت بالفعل.

مثال توضيحي:

```
#include<stdio.h>
void num(int *a);
main()
{
int x;
scanf(%d",&x);
num(&x);
Printf("%d",x);
}
void num(int *a)
{
*a+=1;
}
```

الشرح:

في هذا المثال سيدخل المستخدم عدد ما (x) ثم سنرسل هذا العدد كمرجع (كعنوان) إلى التابع num و بهذه الطريقة سيتم إخراج الناتج عن طريق (x) أيضاً و لكن لن يتم هنا إجراء نسخة مطابقة للمتغير x بل سيتم تعديلها فقط عن طريق المتحول الشكلي (a) فإذا تغير a يتغير x و كأن a هو اسم آخر لـ x أي يتم التعامل معه و كأنه المتحول.

و أريد التأكيد على معنى المرجعية إن المتغيرات تستخدم لتخزين قيم في الذاكرة و بالتالي لكل متغير في الذاكرة معلومتين : الأولى و هي القيمة التي يحملها و الثانية هي عنوانه في الذاكرة (مكان وجوده) و بالتالي عند استخدام الاستدعاء بالمرجع نكون قد وضعنا متغير آخر (الشكلي) يشير إلى نفس القيمة المرسلة فإذا تغيرت قيمة أحد المتغيرين (الفعلي و الشكلي) يكون قد تغير الأخر و كأننا وضعنا عنوانين في الذاكرة يشيران إلى نفس القيمة وهذا يعني أنه لن يتم نسخ قيمة المتحول الفعلي كما هو الحال في الاستدعاء بالقيمة حيث يتم في ذلك الاستدعاء إجراء نسخة كاملة عن المتحول الفعلي ووضعها في المتحول الشكلي بحيث يكون لها عنوان مختلف و قيمة مستقلة عن المتحول الفعلي وبالتالي إذا تغير أحدهم لن يتغير الأخر.
وفي هذا المثال, إذا فرضنا أن المستخدم أدخل العدد 5 فسيظهر على الشاشة العدد 6, و هكذا.

كتابة : **Mohammad Jourieh**

تنسيق : **Amjad Gneem**

تدقيق : **Hasan Alhamwi**

في حال ورود أي خطأ يرجى مراسلة إدارة الفريق عبر الإيميل أو عبر الفيسبوك

ite.technoteam@gmail.com

https://www.facebook.com/technoteam.ite

https://www.facebook.com/TechnoTM

May You Do Your Best And Don't Be Afraid.
Success Is Like A Ball In Air It Depends On You, How You Catch It In Your Hand.