الفصل 3: القوائم المترابطة

1. مقدمة:

رأينا في السداسي الأول ان البرنامج عبارة عن مجموعة من البيانات ومجموعة من التعليمات حيث تخزن هاته البيانات في الذاكرة على شكل متغيرات.

المتغير variable هو مكان في الذاكرة له عنوان للتخزين، له اسم، له نوع وله قيمة.

- **عنوان**: لكل متغير مخزن في الذاكرة عنوان يشير إلى مكانه وهو عبارة عن عدد طبيعي يحدد رقم الثمانية (octet) الاولى التي يوجد بها المتغير وفي العادة يكتب في النظام 16 مثل: 0x5A63
- **اسم**: عبارة عن معرف يستعمله المبرمج للرجوع إلى القيمة المخزنة والتعامل مع المتغير بدل العنوان. مثل: poid
- **نوع**: كل شيء في الحاسوب عبارة عن 0 و1 فالنوع يحدد كيفية ترجمتها كما يحدد الحجم اللازم حجزه على الذاكرة أي عدد البيتات (bits) والعمليات المسموح بها. مثال: int (32 bits)
- **قيمة**: هي محتوى البيتات التي يتكون منها أي قيمتها وفي العادة هي الشيء الذي يتغير اثناء تنفيذ البرنامج مثل:15

اثناء تنفيذ البرنامج وعندما يصادف تعليمة من نوع التصريح عن متغير مثلا (int age ;) var age :entier فان البرنامج يطلب من نظام التشغيل (Windows) ان يحجز مكانا في الذاكرة بالحجم المطلوب (حسب النوع) وبعد الحجز يقوم النظام بإرجاع عنوان المكان الذي يمكن استعماله كمتغير.

للحصول على قيمة المغير يكفي كتابة اسمه ولكن للحصول على عنوانه أي موقعه في الذاكرة فإننا نقوم في الخوارزم بوضع الرمز @ قبل اسم المتغير وفي C نضع الرمز & قبل اسم المتغير.

مثال:

ecrire("valeur de age =", age," son adresse =",@age);

```
printf("valeur de age = %d son adresse = %p",age,&age);
```

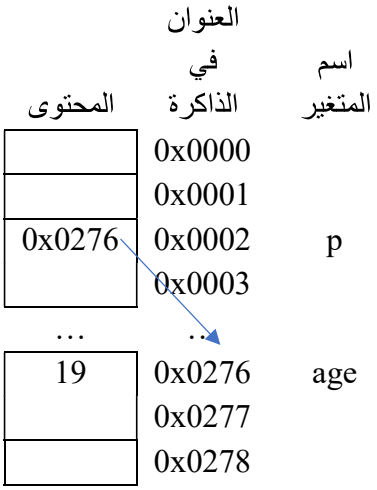
%p هي صيغة للتعامل مع القيمة &age على انها عنوان في الذاكرة أي عدد مكتوب في نظام السادس عشر 16. حيث يمكننا استعمال %d لرؤيته في النظام العشري. هنا age هي قيمة المتغير اما &age فهو عنوانه في الذاكرة حيث يمكنه ان يتغير كلما قمنا بتنفيذ البرنامج.

2. المؤشرات

المؤشر هو متغير تشير قيمته إلى عنوان في ذاكرة الحاسوب، حيث يكون هذا العنوان اما لمتغير او لبرنامج. حيث تستعمل من اجل تمرير المعاملات بالعنوان او حجز الذاكرة بطريقة ديناميكية او تعريف الأنواع التراجعية (القوائم المكدسات والطوابير) وله عدة استعمالات أخرى.

مثال:

يمكن تخيل الذاكرة على انها جدول مرقم من 0 الى سعة الذاكرة -1 في المثال التالي تم حجز متغيرين الأول age من نوع عدد صحيح يوجد في العنوان 0x0276 ويحتوي على القيمة 19 هنا 0x على ان العدد مكتوب في النظام 16 (0x0276=630 في النظام العشري). اما المتغير الثاني فهو p وقيمته هي 0x0276 والتي تمثل الموقع الذي يوجد به age. لذلك نقول ان p يشير الى age.



الانشاء

لإنشاء متغير من نوع مؤشر، في الخوارزمية نقوم بإضافة الرمز ^ **أمام نوع** المتغير. حيث يأخذ الصيغة التالية:

```
var p1,p2 :^type
```

ولإنشاء متغير من نوع مؤشر في لغة C نقوم بإضافة * **قبل اسم** المتغير `type *p1,*p2;`

هنا ^ او * تدل على ان المتغير من نوع مؤشر أي عنوان مكان في الذاكرة اما type فهو نوع المكان الذي سيحمل عنوانه.

مثال: نقوم بالتصريح عن ست متغيرات x وy من نوع عدد صحيح و p1 وp2 من نوع مؤشر على عدد صحيح وz من نوع عدد حقيقي وpz من نوع مؤشر على عدد حقيقي.

`int x,*p1,y,*p2 ;` `float z,*pz;`	`Var x, y : entier  p1, p2 : ^ entier` `z : réel  pz : ^réel`

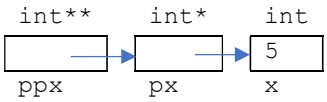
عند التصريح بمتغير فانه يحمل قيمة غير محددة لذلك ينصح بان تسند له القيمة **NULL** بحروف كبيرة والتي تعني ان المؤشر لا يؤشر الى أي مكان (معرفة داخل `stdio.h` والتي تمثل العدد 0) `p1= NULL;`

يمكن للمتغير p1 ان يأخذ عنوان المتغير x او قيمة المتغير p2 ولكن لا يمكنه ان يأخذ عنوان المتغير z ولا عنوان p2 ولا قيمة pz.

الشرح	العمليات المرفوضة	العمليات الصحيحة
p1 من نوع مؤشر اما x فهو عدد صحيح	p1=x;	p1=&x ;
p1 من نوع عنوان عدد صحيح اما &z فهو عنوان عدد حقيقي	p1=&z ;	p2=p1;
pz مؤشر عدد حقيقي اما p1 فهو مؤشر عدد صحيح	pz=p1;	pz=&z ;
P2 عنوان عدد صحيح اما & فهو عنوان مؤشر عدد صحيح	p2=&p1 ;	
يجب ان يكون متغير وليس عدد.	p1=&(0x0276);	

يجب ان نفرق بين العنوان المخزن في المؤشر وعنوان المؤشر في حد ذاته اذ أن المؤشر عبارة عن متغير له عنوان كباقي المتغيرات وبالتالي يمكن اسناد عنوانه الى مؤشر اخر ولكن في هذه الحالة يجب ان يكون نوع المؤشر الثاني هو عنوان لمؤشر من النوع الأول.

مثلا: x من نوع عدد صحيح (int) وpx يحمل عنوان x اذن نوعه هو(int*) وppx يحمل عنوان px اذن نوعه هو (int**) كما هو موضح في الرسم



يتم التصريح عنها كالتالي:

```
int x,*px,**ppx;
x=5 ;
px=&x ;
ppx=&px ;
```

يمكن استعمال typedef لإنشاء أنواع جديدة ويصبح التصريح السابق كالتالي

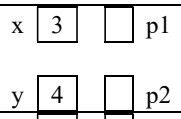
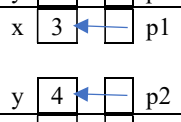
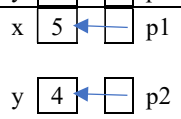
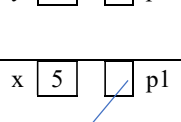
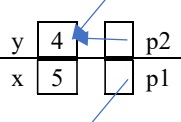
```
typedef int* pint;
typedef int** ppint;
pint px;
ppint ppx;
```

الاستعمال

من النادر ان نتعامل مع عناوين الذاكرة كأعداد مباشرة وانما نتعامل معها على انها عناوين لمتغيرات موجودة، وللحصول على عنوان متغير ما نقوم باستخدام العملية @ في الخوارزم او **&** في لغة البرمجة C **قبل** اسم المتغير، ولاسترجاع قيمة المتغير (Déréférencement) انطلاقا من عنوانه المخزن في المؤشر نستعمل الرمز ^ **بعد** اسم المتغير في الخوارزم و* **قبل** اسم المتغير في لغة البرمجة C.

```
p←@x ⇒ p^ ⇔ x
p=&x ⇒ *p ⇔ x
```

مثال:

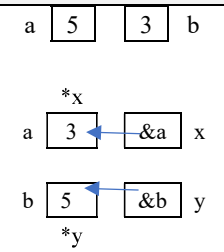
C	الخوارزمية	الذاكرة	الشرح
`int x,*p1,y,*p2 ;`	Var x, y : entier p1, p2 : ^ entier		
`x=3 ;` `y=4 ;`	x←3 y←4		
`p1=&x ;` `p2=&y ;`	p1←@x p2←@y		هنا p1 يحمل عنوان x و2p يحمل عنوان y
`*p1=5;`	p1^←5		نسند العدد 5 الى المتغير الذي عنوانه موجود في p1 وفي هاته اللحظة هو المتغير x كأن المتغير x أصبح له اسم ثان هو *p1 يمكن تعويضها بالتعليمة ; x=5
`p1=p2;`	p1←p2		نسند قيمة p2 والتي تمثل عنوان y الى p1 ليصبح y و *p1 و *p2 نفس المتغير في هاته اللحظة
`*p1=6;`	p1^←6		نسند العدد 6 الى المتغير الذي عنوانه موجود في p1 وفي هاته اللحظة هو المتغير y يمكن تعويضها بالتعليمة ; y=6 او ; *p2=6

ملاحظات:

- لفهم المؤشرات ينصح دوما برسم المتغيرات حيث يحمل المؤشر سهما ينطلق منه الى المتغير الذي يحمل عنوانه ونرمز للمؤشر الذي يحمل القيمة NULL أي لا يشير الى أي مكان بـ ⦸
- المؤشر دوما عبارة عن نوع بسيط بينما يمكن للمتغير الذي يحمل عنوانه ان يكون من نوع مركب (جدول او بنية).
- محاولة استرجاع قيمة مؤشر غير مهيأ او يحمل القيمة NULL يؤدي الى اغلاق البرنامج لذلك:
  - يجب اسناد قيمة (عنوان متغير) الى المؤشر قبل محاولة استرجاع القيمة التي يشير اليها.
  - قبل استرجاع القيمة التي يشير اليها المؤشر يجب التأكد من انه لا يحمل القيمة NULL.
- الان يمكن فهم تمرير المعاملات بالعنوان في البرامج الجزئية.

مثال

الشرح	الذاكرة	C

هنا x وy عبارة عن مؤشران واثناء استدعاء الدالة نسند لx عنوان المتغير a أي x=&a ولy عنوان المتغير b أي y=&b وداخل الدالة echanger للحصول على المتغير الذي x يحمل عنوانه نستعمل العملية * حيث x* في هذه اللحظة تمثل المتغير a و y* تمثل المتغير b		`void echanger(int *x, int *y){`
		`int t;`
		`t=*x;`
		`*x=*y;`
		`*y=t;`
		`}`
		`int a=5,b=3;`
		`echanger(&a,&b);`

3. العمليات على المؤشرات

لنفرض انه لدينا P وQ مؤشران وi عدد صحيح. الجدول التالي يلخص العمليات التي يمكن اجراؤها على المؤشرات

العملية في الخوارزمية	العملية C	نوع المعامل الثاني	نوع النتيجة	مثال	ملاحظة
+	+	عدد صحيح int	مؤشر	P + i	ترجع مؤشر على i عنصر بعد P في جدول
	++		مؤشر	P++	ترجع مؤشر على العنصر الذي يلي P مباشرة في جدول
-	-	عدد صحيح int	مؤشر	P – i	ترجع مؤشر على i عنصر قبل P في جدول
	--		مؤشر	P--	ترجع مؤشر على العنصر الذي يسبق P مباشرة في جدول
-	-	مؤشر من نفس النوع	عدد صحيح int	P - Q	ترجع عدد العناصر الموجودة بين P وQ حيث يجب ان يكون P وQ يؤشران على نفس الجدول
=	==	مؤشر	منطقي	P == Q	تكون صحيحة في حالة P وQ يحملان نفس العنوان أي يؤشران لنفس المكان
≠	!=	مؤشر	منطقي	P != Q	تكون صحيحة في حالة P وQ مختلفان
^	*		نوع القيمة	*P	لاسترجاع القيمة التي يحمل عنوانها

4. إدارة الذاكرة بطريقة ديناميكية

الطريقة التي نعرفها حتى الان لحجز المتغيرات في الذاكرة تسمى بالحجز الثابت (la réservation statique) حيث يتم التصريح بالمتغير في بداية البرنامج ويقوم المجمع بحجز الذاكرة اللازمة بطريقة اوتوماتيكية ولا يتم حذف المتغير الا عند الانتهاء من تنفيذ البرنامج (او البرنامج الجزئي في حالة متغير محلي). لكن في بعض الأحيان نحتاج الى حجز كمية من الذاكرة ولتكن جدولا ذا n عنصر مثلا، ولا يمكن معرفة n الا اثناء التنفيذ، فنقوم بالتصريح عن مؤشر وحين توفر n نقوم بحجز الجدول.

يمتلك المبرمج مجموعة من الدوال تسمح له بإدارة الذاكرة بطريقة ديناميكية اي اثناء التنفيذ.

في الخوارزم:

توجد ثلاث **إجراءات** لادارة الذاكرة بطريقة ديناميكية وهي:

allouer() لحجز جدول حيث يأخذ كمعامل اسم المؤشر nom_tab (اسم الجدول) وعدد العناصر nb_elements

```
allouer(nom_tab,nb_elements)
```

```
allouer(t,10)
```

()réallouer لتغيير احجم الجدول سواء بالزيادة او بالنقصان ويأخذ كمعامل اسم المؤشر nom_tab (اسم الجدول) وعدد العناصر الجديدnouvelle_taille حيث يبقي على قيم العناصر المحجوزة مسبقا ويحذف الزائد او يضيف عناصر جديدة للجدول

```
réallouer(nom_tab, nouvelle_taille)
réallouer(t,15)
```

()désallouer لحذف الجدول الذي تم حجزه بـallouer ويأخذ كمعامل اسم المؤشر nom_tab (اسم الجدول)

```
désallouer(nom_tab)
désallouer(t)
```

بعد انشاء جدول t بواسطة allouer يمكن الوصول الى عناصره بواسطة العارضتين **[ ]** او بواسطة عملية الاسترجاع ^ حيث نعلم ان المؤشر t يحمل عنوان العنصر الأول t[0] اي t=@t[0] و t^=t[0] وللحصول على عنوان العنصر الثاني t[1] نضيف 1 الى t اي t+1=@t[1] و (t+1)^=t[1] وعليه فان عنوان t[i] هو t+i. اي (t+i)^=t[i] و @t[i]=(t+i).

مثال:

الخوارزم	الذاكرة	الشرح
var t : ^réel n :entier	t n	يتم التصريح بمؤشر t ومتغير n الذي يمثل عدد عناصره
début ecrire("entrer le nombre des éléments") lire(n)	t n 3	نفرض ان n تأخذ 3
allouer(t ,n)	t →	تقوم allouer بحجز جدول من ثلاث عناصر وتضع عنوانه في t
t[0] ←1 t[1] ←2 t[2] ←3 او t^ ←1 (t+1)^ ←2 (t+2)^ ←3	t → 1 2 3	نقوم بتعبئة الجدول حيث يمكن استعمال العارضتين [ ] او استعمال ^ حيث (t+i)^ ⇔ t[i]
reallouer(t,n+2)	t → 1 2 3	استدعاء reallouer يقوم بتغيير حجم الجدول الى 5
t[3] ←4 t[4] ←5 او (t+3)^ ←4 (t+4)^ ←5	t → 1 2 3 4 5	نعبئ العنصرين المضافين
désallouer(t)	t n 3	نقوم باستدعاء désallouer لحذف الجدول

في C

تختلف إدارة الذاكرة في C نوعا ما عنها في الخوارزم وقبل ان نتعرف عنها يجب التعرف على sizeof وتغيير النوع.

4.1. حجم متغير في الذاكرة باستعمال العملية sizeof:

يأخذ المتغير مساحة كبيرة أو صغيرة في الذاكرة اعتمادا على نوعه. حيث ان متغيرا من نوع char يأخذ ثمانية واحدة بينما يحتاج متغير من نوع int الى ثمانيتين او أربع حسب اصدار C. ولمعرفة الحجم اللازم لنوع ما نستعمل sizeof() التي تأخذ اسم متغير او اسم النوع لترجع عدد الثمانيات التي يحتاجها في الذاكرة.

```
int sizeof(type);
```

مثال:

```
float t[20] ;
printf("char   : %d octets\n", sizeof(char));
printf("int    : %d octets\n", sizeof(int));
printf("double : %d octets\n", sizeof(double));
printf("la taille de t: %d octets\n", sizeof(t));
printf("la taille de t: %d octets\n", 20*sizeof(float));
```

التي تظهر على الشاشة

```
char   : 1 octets
int    : 4 octets
double : 8 octets
la taille de t: 80 octets
la taille de t: 80 octets
```

يمكن معرفة حجم جدول بضرب حجم خانة واحدة في عدد الخانات.

4.2. تغيير النوع transtypage/casting:

في بعض الأحيان نحتاج تحويل قيمة معينة من نوع الى نوع اخر ولإجبار المجمع على تغيير نوع قيمة معينة نستعمل الصيغة التالية:

```
(type_name) expression
```

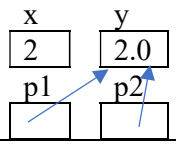
حيث يتم تحويل قيمة expression الى النوع type_name

مثال 1

`int A=8,B=3 ;`	
`float R=A/B ;`	بما ان المعاملين A وB صحيحين فان العملية / تقوم بالقسمة الاقليدية R=8/3
`printf("no casting R=%f \n",R);`	تظهر no casting R=2.000000
`R=(float)A/B ;`	نقوم بتحويل قيمة A (وليس المتغير A) الى عدد حقيقي ثم نقوم بعملية القسمة حيث تصبح العملية R=8.0/3
`printf("with casting R=%f \n",R);`	تظهر with casting R=2.666666

مثال 2

`int x,*p1 ;`	عدد صحيح ومؤشر لعدد صحيح
`float y=2,*p2 ;`	عدد حقيقي ومؤشر لعدد حقيقي
`x=(int)y ;`	تحويل قيمة y الى عدد صحيح ووضعه في x إذن x يأخذ القيمة 2

p2=&y ;	p2 يأخذ عنوان y
p1=(int*)p2 ;	تحويل العنوان من عنوان float الى عنوان int لكن يبقى في كلا المتغيرين عنوان نفس المتغير الا وهو y 
printf("x=%d \n",x);	يظهر x=2
printf("*p2=%f \n",*p2);	يظهر *p2=2.000000 هي نفسها y
printf("*p1=%d \n",*p1);	يظهر *p1=1073741824 لان ترجمة بتات عدد حقيقي على انها عدد صحيح لا تعطي نفس النتيجة

4.3. إدارة الذاكرة في C

تتم الإدارة الديناميكية لذاكرة في C باستخدام أربع دوال معرفة في للمكتبة stdlib هي:

- malloc()، (**m**emory **alloc**ation وتعنى حجز الذاكرة) تطلب من نظام التشغيل حجز الكمية المطلوبة من الذاكرة.

```
void * malloc(int taille);
```

تأخذ كمعامل حجم الذاكرة المطلوب (عدد الثمانيات) وترجع مؤشر الى الذاكرة التي تم حجزها او ترجع NULL في حالة فشل العملية لعدم توفر الحجم المطلوب.

مثال:

```
float *t;
t=(float *)malloc(10*sizeof(float));
```

```
t=      (float *)    malloc(    10*    sizeof(    float    ))
;
```

	نوع كل خانة	حجم كل خانة	عدد الخانات	لحجز الجدول	التحويل الى نوع المؤشر	اسم الجدول

- free()، لإرجاع الذاكرة المحجوزة سابقا عن طريق malloc لنظام التشغيل حتى يمكن استعمالها في برامج اخرى.

```
void free( void * pointeur );
```

تأخذ كمعامل مؤشر الى الذاكرة التي تم حجزها مسبقا. ينصح بإسناد NULL الى المؤشر بعد استدعاء free للتأكد من ان المؤشر لا يشير الى أي مكان وتفادي أي خطا.

مثال:

```
free(t);
```

- realloc()، لتغيير حجم الذاكرة المحجوزة سواء بالزيادة او بالنقصان.

```
void * realloc(void * pointeur, int nouvelle_taille);
```

حيث تقوم الدالة باستدعاء malloc لحجز مكان جديد بحجم nouvelle_taille ثم تقوم بنسخ جميع قيم الجدول pointeur الى الموقع الجديد (او تحذف الكمية الزائدة إذا كان nouvelle_taille اقل من الحجم القديم) ثم تقوم بحذف الذاكرة القديمة المحجوزة باستدعاء free وفي حالة نجاح العملية ترجع مؤشر الى الموقع الجديد والا ترجع NULL.

مثال:

```
t=(float*)realloc(t, 20*sizeof(float));
```

- calloc()، مثل malloc، الا انها تضع اصفار في الذاكرة المحجوزة.

```
void * calloc(int nb_element, int taille_element);
```

تأخذ nb_element الذي يمثل عدد عناصر الجدول وtaille_element الذي يمثل حجم خانة واحدة وترجع مؤشر الى المكان المحجوز.

مثال:

```
t=(float*)calloc(10,sizeof(float));
```

ملاحظة:

- في درس الدوال رأينا ان void معناها ان الدالة لا ترجع أي شيء لكن *void معناها ان الدالة ترجع مؤشر من نوع غير محدد.
- يجب تغيير النوع *void الى نوع المؤشر الذي سيحمل العنوان وذلك بوضع نوع المؤشر بين قوسين قبل اسم الدالة malloc، calloc وrealloc ولكن هذا التحويل ليس ضروريا في لغة C++.
- لاستعمال هاته الدوال يجب استحضار المكتبة stdlib او alloc عن طريق التعليمة:

```
#include <stdlib.h>
```

او

```
#include <alloc.h>
```

- العملية sizeof ليس دالة لذلك يمكن الاستغناء عن القوسين.

عندما نقوم بحجز الذاكرة نتبع الخطوات التالية:

1. نقوم بحجز الذاكرة بواسطة malloc.
2. نتأكد من ان عملية الحجز قد تمت بنجاح باستعمال `if(pointeur !=NULL)`
3. عند الانتهاء من استعمال المكان المحجوز نرجع الذاكرة للنظام عن طريق `free`

مثال

الشرح	C
استحضار المكتبة stdlib	`#include <stdio.h>` `#include <stdlib.h>`
التصريح بمؤشر من نوع رمز char	`int main(void) {` `  char *str;`
حجز جدول يتسع لـ 4 رموز	`str = (char *) malloc(4*sizeof(char));`
ملء الجدول بالسلسلة الحرفية "ASD" باستعمال [ ] والرمز '\0' لتحديد نهاية السلسلة.	`str[0]='A'; str[1]='S'; str[2]='D';` `str[3]='\0';`
ملء الجدول بالسلسلة الحرفية "ASD2" باستعمال عملية الاسترجاع * حيث ان str[i] ⇔ *(str+i)	`*str='A'; *(str+1)='S'; *(str+2)='D';` `*(str+3)='\0';`
لإظهار السلسة وعنوانها حيث نلاحظ عدم استعمال & لان str عبارة عن عنوان	`printf("String is %s\n  Address is %p\n",` `str, str);`
تغيير سعة الجدول من 4 الى 5	`str = (char *) realloc(str, 5*sizeof(char));`
ملء الخانتين الأخيرتين لتصبح السلسلة الحرفية "ASD2"	`str[3]='2'; str[4]='\0';` `*(str+3)='2'; *(str+4)='\0';`
إظهار السلسلة الحرفية "ASD2" وعنوانها الجديد	`printf("String is %s\n  New address is` `%p\n", str, str);`

ارجاع الذاكرة المحجوزة	`free(str);`
	`return 0;`
	`}`

4.4. المؤشرات والمصفوفات في C

المصفوفات في C هي عبارة عن جدول كل عنصر منه عبارة عن جدول. نريد انشاء مصفوفة M[3][4] بـ 3 أسطر و4 اعمدة.

لنفرض انه لدينا 3 جداول M0,M1,M2

```
float M0[4],M1[4],M2[4] ;
```

يمكن انشاء هذه الجداول باستعمال المؤشرات

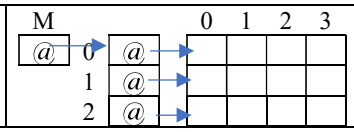
```
float *M0,*M1,*M2 ;
M0=(float *)malloc(4*sizeof(float));
M1=(float *)malloc(4*sizeof(float));
M2=(float *)malloc(4*sizeof(float));
```

نلاحظ ان M0,M1,M2 كلها من نفس النوع (* float) لذلك يمكن تعويضها بجدول M من نوع (* float)

```
float * M[3] ;
for(int i=0;i<3;i++)
  M[i]=(float *)malloc(4*sizeof(float));
```

الان يمكن استعمال المؤشرات لإنشاء الجدول M

الشرح	الذاكرة	C
يتم التصريح بمؤشر M من نوع ** float	M	`float **M ;`
		`M=(float**) malloc( 3*sizeof(float*));`
يتم انشاء الجدول M الذي يحتوي على 3 عناصر التي تمثل عدد الاسطر نوع كل خانة منها هو * float	M @ 0 1 2	
		`for(int i=0;i<3;i++)` `  M[i]=(float*) malloc(4*sizeof(float));`

ننشئ الجداول التي يمثل كل جدول منها سطرا في المصفوفة. 4 هو عدد الاعمدة ونوع كل خانة هو float. يمكن استعمال (M+i)* بدل M[i]		

يمكن الوصول لاي عنصر من المصفوفة باستعمال [] او باستعمال عملية الاسترجاع * حيث

M[i][j] ⇔ *(*(M+i)+j)

باستعمال typedef

```
typedef float ** matrix;
typedef float * table;
matrix M ;
M=(matrix)malloc(3* sizeof(table));
for(int i=0;i<3;i++)
  M[i]=(table) malloc(4*sizeof(float));
```

ملاحظة مهمة الجدول الثابت في لغة C عبارة عن عنوان في الذاكرة ثابت أي لا يمكن تغييره.

مثال:

```
int *p,t[10];
```

```
p=t;
```
مقبولة لان t عنوان لأول int

```
t=p;
```
غير مقبول لان t ثابت لا يمكن تغييره.