



# كلية الهندسة – قسم المعلوماتية

مقرر برمجة 2

ا.د. علي سليمان

محاضرات الأسبوع الأول

**Pointers**

الفصل الاول 2025-2026

1. Course overview.	1. نظرة عامة على المقرر.
2. Pointers and Strings.	2. المؤشرات والسلاسل المحرفية.
3. Classes and Objects.	3. الأصناف والأغراض
4. Arrays, Pointers, References, And the Dynamic Allocation Operators.	4. المصفوفات، المؤشرات، المراجع، ومعاملات التخصيص الديناميكي
5. Function Overloading ,Copy constructors , and Default Arguments.	5. التحميل الزائد للتوابع، التوابع البانية الناسخة، والوسطاء الافتراضية
6. Operator Overloading.	6. التحميل الزائد للمعاملات
7. Inheritance.	7. الوراثة
8. Virtual Functions and Polymorphism.	8. التوابع الظاهرية وتعدد الأشكال
9. Templates.	9. القوالب
10. Exception Handling.	10. معالجة الاستثناءات
11. The C++ I/O System Basics.	11. أساسيات نظام الدخل / الخرج في لغة C++
12. C++ File I/O.	12. الإخراج والإدخال من وإلى الملفات في C++

المحاضرة من المراجع :

[1]- Deitel & Deitel, C++ How to Program, Pearson; 10th Edition (February 29, 2016)

[2]- د.علي سليمان, البرمجة غرضية التوجه (حاسبات), برمجة 2 (اتصالات) في لغة C++ جامعة اللاذقية 2009-2010, 2009-2008

عند الانتهاء من دراسة هذا الفصل سيكون الدارس قادراً على ما يلي :

- ذكر بعض ميزات البرمجة كائنية التوجه مقارنة مع البرمجة الإجرائية بالنقاط التالية:
- تعريف مفهوم التغليف، تعدد الأشكال، الوراثة.
- تعريف المؤشر والتصريح عنه والاستفادة من بعض استخداماته.
- تمييز العلاقة بين المؤشرات والمصفوفات .
- استخدام المؤشرات كبارامترات للتوابع أو كقيم معادة منها .
- استخدام توابع التخصيص الديناميكي.
- تعريف واستخدام مصفوفات المؤشرات.
- التصريح عن السلاسل المحرفية وإجراء عمليات الإدخال والإخراج.
- استخدام بعض توابع المكتبة القياسية `string.h`.

■ نذكر بعض ميزات البرمجة كائنية التوجه مقارنة مع البرمجة الإجرائية وفق النقاط التالية:

- إعتماؤها على الكائنات وليس على الأفعال.
- تعتبر البرمجة غرضية التوجه الأسرع والأسهل في التنفيذ.
- تؤمن البرمجة غرضية التوجه بنية أكثر وضوحاً للبرامج.
- تجعل البرنامج خال من التكرارات وأسهل في الصيانة والتعديل.
- تتيح إمكانية خلق تطبيقات مختلفة من نفس البرمجة بتعليمات أقل ووقت أقصر للتطوير والتعديل.

# Introduction to Encapsulation



## مقدمة إلى التغليف

- التغليف Encapsulation:

التغليف هو الآلية التي تربط الشيفرة والبيانات التي تتعامل معها، وتحميها من كل من التدخل الخارجي و سوء الاستخدام غير المقصود.

عند تجميع الشيفرة والبيانات في كتلة يمكن النظر إليها «كصندوق أسود» قائم بذاته، ويتم إنشاء نسخ عنه تعرف بالاعراض object، بمعنى آخر الغرض هو من يدعم التغليف.

يمكن ضمن غرض ما أن تكون البيانات، الشيفرة أو كلاهما، خاصة private بذلك الغرض أو عامة public.

الشيفرة أو البيانات الخاصة تكون مرئية ويمكن الوصول إليها فقط من قبل الأجزاء الأخرى للغرض، أي أن الشيفرة أو البيانات الخاصة لا يمكن الوصول إليها من قبل أي أجزاء من البرنامج توجد خارج الغرض.

الأجزاء العامة (شيفرة أو بيانات)، تتيح للأجزاء الأخرى من البرنامج أن تصل إليها حتى ولو كانت معرفة ضمن غرض.

نموذجياً الأجزاء العامة للغرض تستخدم لتحقيق واجهة متحكم بها للأجزاء الخاصة للغرض.

# Introduction to Polymorphism

## مقدمة إلى تعدد الأشكال

- تعدد الأشكال Polymorphism:

واجهة واحدة، طرائق متعددة one interface , multiple methods " كمثال من العالم الحقيقي عن تعدد الأشكال هو منظم الحرارة thermostat. فبغض النظر عن طريقة توليد الحرارة في منزلك ( غاز، نפט، كهرباء ... إلخ ) فإن منظم الحرارة يعمل بالطريقة ذاتها، في هذه الحالة، منظم الحرارة ( الذي يمثل الواجهة interface ) هو نفسه بغض النظر عن نوع مصدر الحرارة (الذي يمثل الطريقة method) الذي تقوم باستخدامه. أي إذا أردت درجة حرارة الغرفة 23 درجة مئوية، فسيضبط الحرارة على 23 درجة ولا يكون مهماً ما هو مصدر الحرارة.

كمثال آخر لنفرض برنامج يعرف ثلاثة أنماط مختلفة من المكذسات، للقيم الصحيحة، المحرفية ولقيم الفاصلة العائمة، وبفضل تعدد الأشكال يمكن تعريف مجموعة من الطرائق منها push()، pop() التي يمكن استخدامها من أجل المكذسات الثلاث، يمكن في البرنامج إنشاء ثلاث نسخ مختلفة من نداء التوابع، واحدة لكل نوع من المكذسات لكن أسماء التوابع ستكون نفسها، وسيقوم المترجم تلقائياً باختيار التابع الملائم بحسب البيانات المخزنة، وبالتالي فإن الواجهة إلى المكذس – التوابع push() و pop() هي نفسها بغض النظر عن نوع المكذس المستخدم.

تساعد تعدد الأشكال على تقليل التعقيد من خلال السماح لنفس الواجهة لأن تصل إلى صنف عام من الأفعال، وتكون مهمة المترجم هي اختيار الفعل المحدد (الطريقة) التي سيقوم باستخدامها في كل حالة ولا يحتاج المبرمج لاختيارها يدوياً، فقط يحتاج لأن يعرف، ويستخدم الواجهة العامة general interface التحميل الزائد.

## - الوراثة inheritance:

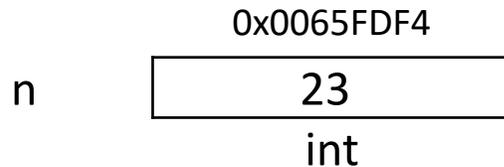
"الوراثة هي العملية التي يستطيع من خلالها غرض ما الحصول على خصائص غرض آخر، إن هذه العملية مهمة جداً لأنها تدعم مبدأ التصنيف classification والذي يعني أن أغلب المعلومات قابلة للإدارة من خلال تصنيف شجري hierarchical classification.

مثال : التفاحة الحمراء هي جزء من الصنف تفاح apple، الذي هو بدوره جزء من الصنف فاكهة fruit، وهو بدوره أيضاً جزء من صنف أكبر هو الطعام food. دون القيام باستخدام الصنف، فإن كل غرض يجب تعريف جميع خصائصه بشكل صريح، في حين أنه باستخدام الصنف، فإن كل غرض يحتاج لأن يتم تعريف الخصائص التي تميزه فقط ضمن الصنف.

إن مبدأ الوراثة هو ما يجعل من الممكن لغرض أن يكون حالة خاصة من حالة أكثر عمومية، سنرى لاحقاً أن الوراثة هي إحدى أهم مفاهيم البرمجة غرضية التوجه.

- يعود استخدام المؤشرات لثلاثة أسباب:
  - ✓ المؤشرات تعطي إمكان قيام التوابع بتعديل وسطاء الاستدعاء.
  - ✓ المؤشرات تدعم التخصيص الديناميكي dynamic allocation.
  - ✓ المؤشرات يمكن أن تحسن من فعالية بعض الإجراءات.
- الإعلان عن المتغير `int n;` ينطوي على ثلاثة مفاهيم أساسية:
  - ✓ اسم المتغير.
  - ✓ نوع المتغير.
  - ✓ عنوان المتغير في الذاكرة.

فمثلاً ، التصريح والاسناد التالي عن المتحول `n` : `int n=23;` يربط بين الاسم `n` والنوع `int` والقيمة `23` وعنوان المتغير في الذاكرة `0x0065FDF4` ، وبالتالي يمكن تصور المتغير المعلن عنه كما يلي:



- 1- اسم المتغير على اليسار.
- 2- عنوان المتغير من الأعلى.
- 3- نوع المتغير أسفل الصندوق.
- 4- والقيمة المخزنة تساوي 23 وهي محتوى الصندوق.

- هذه العناوين يتحكم بها كل من أنظمة التشغيل و المترجمات المختلفة ومتغيرة .
- يعود ذلك إلى آليات أمنية مثل:
- التوزيع العشوائي لتخطيط مساحة العنوان (ASLR) Address Space Layout Randomization واستراتيجيات تخصيص الذاكرة المتنوعة التي يستخدمها نظام التشغيل لمنع هجمات تلف الذاكرة.
- تعمل ASLR عن طريق تغيير عناوين ذاكرة مكونات النظام الرئيسية، مثل المكس والكومة والمكتبات، عشوائيًا في كل مرة يتم فيها تشغيل التطبيق، مما يُصعب على المهاجمين استغلال الثغرات الأمنية التي تعتمد على مواقع ذاكرة متوقعة. كما أن استراتيجيات تخصيص الذاكرة، كتلك المستخدمة في نظام Windows للتوزيع العشوائي لمواقع الكومة، تُعقد محاولات الاستغلال بشكل أكبر، إذ تُصعب على المهاجم التنبؤ بدقة بمكان الوصول إلى الذاكرة.

- عنوان المتغير يمكن التعامل معه بعامل العنوان & ( address operator ) .
- ✓ طباعة عنوان المتغير n بالأمر التالي: `cout << &n ;` .
- ✓ عامل العنوان & يسبق اسم المتغير لينتج العنوان وله أسبقية على عامل النفي المنطقي وعامل الزيادة المسبقة ++.
- مثال للاستخدام مع عامل العنوان:

```
#include<iostream>
using namespace std;
int main() { int n=23;
    cout<<"the value of n is: n="<<n<<endl;
    cout<<"the address of n is: &n="<<&n<<endl;
    system("pause"); return 0;
} // end main
```

تعريف المتغير n واسناد 23 كقيمة له

طباعة قيمة المتغير n

طباعة عنوان المتغير n

```
the value of n is: n=23
the address of n is: &n=010FEFA8
Press any key to continue . . .
```

يعطي هذا البرنامج على خرجه:

• المرجع: هو اسم آخر مرادف لاسم المتغير، يعلن عنه باستخدام المعامل المرجعي & والذي يلحق بنوع المرجع.

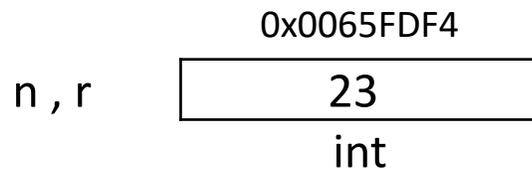
• التصريح عن المرجع

• التصريح التالي:

```
int n = 23;
```

```
int &r = n; // r is a reference for n;
```

✓ الإعلان عن متغير صحيح اسمه n وخصصت له قيمة ابتدائية مقدارها 23.  
✓ أعلن عن متغير r بأنه مرجع لـ n:



✓ إن قيمة r هي ذاتها قيمة n وأي تغير في قيمة n سيصيب قيمة r والعكس صحيح.

✓ إن n و r لهما نفس العنوان في الذاكرة.

✓ إن n و r هما إسمان رمزيان لنفس المكان في الذاكرة.

## References

## المراجع 4

```
#include<iostream>
using namespace std;
void main()
{
    int n=23;
    int &r=n;// r is reference for n
    cout<<"the value of n is: n="<<n<<endl;
    cout<<"the value of r is: r="<<r<<endl;
    cout<<"&n="<<&n<<endl;
    cout<<"&r="<<&r<<endl;
    cout<<endl;
    --n;
    cout<<"the value of n is: n="<<n<<endl;
    cout<<"the value of r is: r="<<r<<endl;
    cout<<endl;    r*=2;
    cout<<"the value of n is: n="<<n<<endl;
    cout<<"the value of r is: r="<<r<<endl;
    system("pause"); //return 0;
} // end main
```

طباعة قيمة المتغير n

طباعة قيمة المتغير r

طباعة عنوان المتغير n

طباعة عنوان المتغير r

إنقاص قيمة المتغير n بمقدار 1

طباعة قيمة كل من n,r  
بعد إنقاص n بمقدار 1

طباعة قيمة كل من n,r  
بعد مضاعفة قيمة r

• المؤشرات: هي عبارة عن متغيرات تحتوي على عناوين والتي تشير إلى قيم مخزنة ضمنها في الذاكرة .

✓ إن المتغير يدل بشكل مباشر على قيمة.

✓ يدل المؤشر بشكل غير مباشر على قيمة.

✓ تسمى عملية الدلالة على قيمة بواسطة مؤشر بالعملية غير المباشرة indirection.

• يمكن التصريح عن المؤشر من خلال أحد الصيغ العامة التالية:

`type *pointerName ; type * pointerName ; type* pointerName ;`

✓ حيث أن:

- type: نوع القيمة التي يشير لها المؤشر.

- pointerName: اسم المؤشر.

- المعامل \* هو معامل المؤشر.

- عندما يستخدم بالشكل السابق فإنه يعلن أن مابعده مباشرة متغير من نوع مؤشر يشير إلى قيمة من النوع الذي قبله.

# Pointers



## المؤشرات 2

`int *x;`

• التصريح:

✓ يعلن أن  $x$  مؤشر لرقم صحيح.

`int *x, y;`

• التصريح:

✓ يعلن أن  $x$  مؤشر لرقم صحيح وأن  $y$  متغير صحيح.

`int *x, *y;`

• التصريح:

✓ يعلن أن كل من  $x$  و  $y$  مؤشران لرقمين صحيحين.

• معاملات المؤشرات هنالك مؤثران خاصان بالمؤشرات هما `*` و `&`.

✓ المؤثر `&` هو مؤثر أحادي يرجع عنوان الذاكرة لمعامله.

✓ المؤثر `*` يرجع مايدل إليه العنوان من الذاكرة.

```
#include<iostream>
using namespace std;
void main()
{
    int n=23;
    int *p=&n;
    cout<<"the value of n is : n="<<n<<endl;
    cout<<"the address of n is : &n="<<&n<<endl;
    cout<<"the value of p is : p="<<p<<endl;
    cout<<"the address of n is : &p="<<&p<<endl;
    system("pause"); //return 0;
} // end main
```

p holds the address of n

طباعة قيمة n من الذاكرة

طباعة عنوان تخزين n في الذاكرة

طباعة قيمة p من الذاكرة

طباعة عنوان تخزين p في الذاكرة

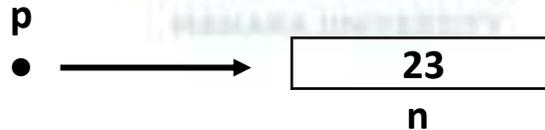
the value of n is : n=23  
 the address of n is : &n=0093F008  
 the value of p is : p=0093F008  
 the address of n is : &p=0093F00C  
 Press any key to continue . . .

إن عنوان تخزين n موضوع في p وبالتالي هو قيمة p المؤشر p محجوز له عنوان يخزن ضمنه قيمته وهو 4byte زياده لعنوان n كونه صحيح

- إن  $int * p = \&n;$  سيخزن عنوان  $n$  في المكان المحجوز لـ  $p$ .
  - نلاحظ من خرج البرنامج أن المتغير  $n$  استخدم العنوان  $0x0093F008$  لتخزين القيمة  $23$  وسيملك  $p$  قيمة  $0x0093F008$ .
- يمكن تصور المتغيرين  $n$  و  $p$  كما في التخطيط التالي:



- إن قيمة المؤشر هي عنوان ويعتمد على المطابق ونظام التشغيل للحاسب الذي يجري عليه البرنامج.
- في معظم الحالات تكون القيمة الفعلية لهذا العنوان غير مهمة للمبرمج ولذلك فإن التخطيط السابق عادة ما يتم رسمه كما في الشكل:



- المؤشر هو محدد وضع حيث يبين أين توجد قيمة أخرى.
- يستعمل المؤشر وحده للحصول على القيمة التي يشير إليها.

- معامل العنوان & ومعامل إعادة المرجعية \* عاملان متتامان حيث أن:  $*p = n$  (1) عند إخراج  $*p$  و  $n$  سيعطيان القيمة ذاتها. عندما  $p = \&n$  (2)، عند إخراج  $\&p$  و  $\&n$  سيعطيان القيمة ذاتها، بتبديل 2 في 1 نحصل على  $\&n = n$  و أيضاً بتبديل 1 في 2 نحصل  $\&*p = p$ .
- المعامل \* معامل متمم للمعامل & وهو معامل أحادي يرجع قيمة المتغير الموجود في العنوان الذي يشكل معاملة.

```
#include<iostream>
```

```
using namespace std;
```

```
void main()
```

```
{ int n=23; int *p=&n; int &r=*p;
```

```
cout<<"&n="<<&n
```

```
<<"\t &p="<<&p
```

```
<<"\t &r="<<&r;
```

```
cout<<"\n n="<<n
```

```
<<"\t\t *p="<<*p
```

```
<<"\t\t r="<<r<<"\n";
```

```
system("pause"); //return 0;
```

```
// end main
```

r مرجع للقيمة التي يشير إليها p، وبالتالي تتحدد قيمة r بالقيمة التي يشير إليها p وهي قيمة n.

ستطبع عنوان تخزين المتغير n  $\&n=00B8F1D4$

ستطبع عنوان تخزين المؤشر p.  $\&p=00B8F1D8$

ستطبع عنوان تخزين المؤشر r.  $\&r=00B8F1D4$

ستطبع القيمة التي يملكها المتغي n.  $n=23$

ستطبع القيمة التي يشير لها العنوان الموجود في المؤشر p  $*p=23$

ستطبع القيمة التي يملكها المرجع r وهي القيمة التي يُوْشِر لها المؤشر p

- تستخدم مؤشرات التعابير الحسابية وتعابير الإسناد والمقارنة.
- المساواة الإسنادية للمؤشرات:

✓ استخدام المؤشر كطرف أيمن في العلاقات الإسنادية لإسناد قيمته إلى مؤشر آخر.

✓ إن المؤشرين يجب أن يشيرا إلى معطيات من نفس النمط.

كمثال: بقرض أن العنوان الحر للحجز هو **&n=0135F3D4**

```
#include<iostream>
using namespace std;
void main()
{
    int n=23;
    int *p1,*p2;
    p1=&n;
    p2=p1;
    cout<<"&n="<<&n<<endl;
    cout<<"p1="<<p1<<endl;
    cout<<"p2="<<p2<<endl;
    system("pause");
} // end main
```

اسناد عنون المتغير n الى المؤشر p1.

اسناد قيمة المؤشر p1 الى المؤشر p2.

ستكون قيم الخرج الثلاث متساوية **p1=0135F3D4**

كمثال بخرج التابع sizeof() الحجم المجوز

```
#include<iostream>
using namespace std;
void main()
{cout<<"number of bytes used:\n";
cout<<"\t char :"<<sizeof(char)<<endl;
cout<<"\t short :"<<sizeof(short)<<endl;
cout<<"\t int :"<<sizeof(int)<<endl;
cout<<"\t long :"<<sizeof(long)<<endl;
cout<<"\t unsigned short :"<<sizeof(unsigned short)<<endl;
cout<<"\t unsigned char :"<<sizeof(unsigned char)<<endl;
cout<<"\t unsigned int :"<<sizeof(unsigned int)<<endl;
cout<<"\t unsigned long :"<<sizeof(unsigned long)<<endl;
cout<<"\t signed char :"<<sizeof(signed char)<<endl;
cout<<"\t float :"<<sizeof(float)<<endl;
cout<<"\t double :"<<sizeof(double)<<endl;
cout<<"\t long double :"<<sizeof(long double)<<endl;
system("pause");    }// end main
```

sizeof() يستخدم مع المعطيات الأولية والمعرفة والكائنات

```
// char :1
// short :2
// int :4
// long :4
// unsigned short :2
// unsigned char :1
// unsigned int :4
// unsigned long :4
// signed char :1
// float :4
// double :8
// long double :8
```

- تطبق عمليات الزيادة والنقصان للقيم الصحيحة (+, ++, --, +=, -).
- طرح أو جمع مؤشرين من أو على بعضهما.
- زيادة أو نقصان المؤشر بـ 1 ستزيد أو تنقص قيمة هذا المؤشر بمقدار حجم النوع الذي يشير إليه المؤشر. يمكن معاينة جميع هذه العمليات من خلال المثال التالي :

```
#include<iostream>
using namespace std;
void main()
{   int n=23;   int *p=&n;
    cout<<"&n="<<&n<<" and p="<<p<<endl;
    ++p;cout<<"after ++p :   p="<<p<<endl;
    p+=2;cout<<"after p+=2:   p="<<p<<endl;
    p--;cout<<"after p-- :   p="<<p<<endl;
    p-=5;cout<<"after p-=5:   p="<<p<<endl;
    int *p1=&n;cout<<"p1="<<p1<<endl;
    cout<<"p1-p="<<p1-p<<endl;
    system("pause");
} // end main
```

طباعة عنوان &n ومحتوى p ولتكن &n=006FEF34 and p=006FEF34

طباعة محتوى p بعد الزيادة بـ 1 لتصبح after p++ : p=006FEF38

طباعة محتوى p بعد الزيادة بـ 2 لتصبح after p+=2: p=006FEF40

طباعة محتوى p بعد النقصان بـ 1 لتصبح after p-- : p=006FEF3C

طباعة محتوى p بعد النقصان بـ 5 لتصبح after p-=5: p=006FEF28

اسناد عنوان n للمؤشر p1 وطباعته p1=006FEF34

طباعته ناتج طرح p1-p والمساويه 3 أي بعد تحويله للنوع p1-p=3

- اسم المصفوفة ليس إلا مؤشر صحيح يدل لبدايتها والعكس صحيح.
- يمكن استخدام أحدها للحلول مكان الآخر ليعمل عمل الآخر.
- التصريح `int a[ 4 ]` يدل لمصفوفه اسمها `a` عدد عناصرها 4 ومن النمط الصحيح.
- التصريحين متكافئين ينسب عنوان العنصر الأول في الصف إلى المؤشر `p`:  
`int *p = a;`  
OR  
`int *p = &a[0]`
- الصيغتان تستخدمان لاستدعاء التابع بالصف :  
`int * a ↔ int a[ ]`
- يمكن الوصول إلى العنصر الرابع في الصف `a[ 3 ]` باستخدام التعبير:  
`cout << * ( p+3 );`
- حيث `( p+3 ) *` هي القيمة التي يشير إليها المؤشر `p` بعد زيادته بمقدار 3.
- إذا أشر مؤشرا إلى بداية الصف فإن الانزياح المضاف إليه يدل على أحد عناصر الصف الذي يطابق دليله قيمة ذلك الانزياح.
- ويجب التأكيد على وجود القوسين في الكتابة السابقة لأن أولوية العملية `*` هي أعلى من أولوية الجمع.
- العبارة السابقة دون أقواس أي `p + 3 *` فإن ذلك يعني إضافة 3 إلى العنصر `a[ 0 ]`

- إن التعبيرين متكافئين:  $\&a[3] \leftrightarrow p+3$
- يستخدم اسم الصف في العمليات الحسابية فمثلاً التعبير:  $(a+3) *$  يعطي قيمة العنصر  $a[3]$  من عناصر الصف.
- يضاف الدليل للمؤشرات حيث أن التعبير:  $p[1]$  يدل على العنصر  $a[1]$ .
- اسم الصف عبارة عن مؤشر ويستحسن أن يكون ثابت وهو يشير دوماً إلى بداية الصف ولذلك فإن التعبير:  $a+=3$  غير محبب.
- المصفوفة ثنائية البعد ينظر لها مصفوفة على مصفوفة.
- يمكن الحصول على عنوان أول عنصر في مصفوفة متعددة الأبعاد من خلال التعليمة:  
`cout<<*(a);`
- يمكن الحصول على عنوان العنصر  $a[i][j]$  من خلال التعليمة:  
`cout<<*(a)+(m*i)+j;`
- إن  $m$  هو عدد أعمدة المصفوفة ثنائية البعد،
- إن عناوين عناصر المصفوفة ثنائية البعد تخزن في الذاكرة في أماكن متجاورة كما لو كانت عناصر تلك المصفوفة تنتهي لصف عدد عناصره هو  $m * n$  حيث  $n$  هو عدد الأسطر و  $m$  عدد الأعمدة في المصفوفة ثنائية البعد.
- قيمة عنصر المصفوفة الثنائية البعد  $a[i][j]$  يمكن أن نحصل عليها من خلال التعليمة:  
`cout<<*(*(a)+(m*i)+j);`

المثال التالي يوضح هذه العلاقة مصفوفة أحادية البعد :

```
#include<iostream>
using namespace std;
void main()
{int a[]={5,7,9,11};
  int *p=a;
  cout<<"a array is:"<<endl;
  for(int i=0;i<4;i++)cout<<"a["<<i<<"]="<<a[i]<<endl;
  cout<<endl<<"pointer/offset notation where"<<endl;
  cout<<"the pointer is the array name"<<endl;
  for(int j=0;j<4;j++)cout<<"*(a+"<<j<<")="<<*(a+j)<<endl;
  cout<<endl<<"pointer subscript notation"<<endl;
  for(int i=0;i<4;i++)cout<<"p["<<i<<"]="<<p[i]<<endl;
  cout<<endl<<"pointer/offset notation"<<endl;
  for(int j=0;j<4;j++)cout<<"*(p+"<<j<<")="<<*(p+j)<<endl;
  system("pause");
} // end main
```

طباعة محتوى المصفوفة باستخدام اسمها ودليلها

طباعة محتوى المصفوفة باستخدام اسمها كمؤشر

طباعة محتوى المصفوفة باستخدام مؤشر كإسم لها

طباعة محتوى المصفوفة باستخدام مؤشر لها

# The Relation Between Pointers And Arrays



## العلاقة بين المؤشرات والمصفوفات 4

نتائج المثال :

a array is:

a[0]=5  
a[1]=7  
a[2]=9  
a[3]=11

طباعة محتوى المصفوفة باستخدام اسمها ودليلها

pointer/offset notation where  
the pointer is the array name

\*(a+0)=5  
\*(a+1)=7  
\*(a+2)=9  
\*(a+3)=11

طباعة محتوى المصفوفة باستخدام اسمها كمؤشر

pointer subscript notation

p[0]=5  
p[1]=7  
p[2]=9  
p[3]=11

طباعة محتوى المصفوفة باستخدام مؤشر كإسم لها

pointer/offset notation

\*(p+0)=5  
\*(p+1)=7  
\*(p+2)=9  
\*(p+3)=11

طباعة محتوى المصفوفة باستخدام مؤشر لها

Press any key to continue . . .

المثال التالي يوضح هذه العلاقة مصفوفة ثنائية البعد :

```
#include<iostream>
using namespace std;
void main()
{const int n=2;      const int m=3;
int a[n][m];        int i,j;
cout<< "enter a ["<<n<<"]["<<m<<"]"<<endl;
for(i=0;i<n;i++)
    for(j=0;j<m;j++)      cin>>a[i][j];
cout<<"a array is:"<<endl;
for(i=0;i<n;i++)
{
    for(j=0;j<m;j++) cout<<a[i][j]<<" ";    cout<<endl;
}
for(i=0;i<n;i++)
{
    for(j=0;j<m;j++)
        cout<<*(a)+(m*i)+j<<" " <<*(*(a)+(m*i)+j)<<endl;
}
system("pause");
} // end main
```

طباعة عناصر المصفوفة باستخدام اسمها

طباعة عناوين عناصر المصفوفة

طباعة عناصر المصفوفة

نتائج مثال مصفوفة ثنائية البعد :

enter a [2][3]

11 22 33

44 55 66

a array is:

11 22 33

44 55 66

0133F374 11

0133F378 22

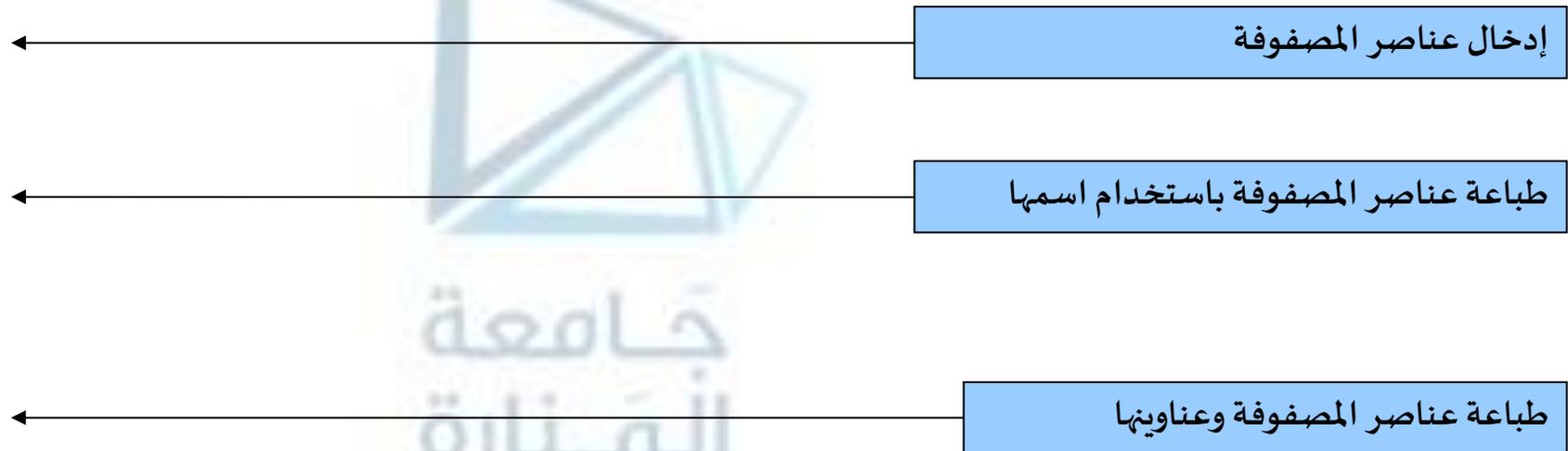
0133F37C 33

0133F380 44

0133F384 55

0133F388 66

Press any key to continue . . .



مثال يوضح كيفية الاستفادة من stack ومبدأ LIFO في عملية تحويل العدد من النظام العشري إلى النظام الثنائي:

```
#include<iostream>
#include<stdlib.h>
using namespace std;

const int stacksize=50;
int*p ; int stack[stacksize];

void push (int i)
{
    if (p==(stack+stacksize))
    { cout<<"stack over flow"; exit(1); }
    *p=i; p++;
}

void pop(int &i)
{
    if(p==stack){cout<<"stack under flow";exit(1);}
    p--;i=*p;
}
```

LIFO



إستخدام المؤشر مع المكديس 1

```
void main()
{
    int x;int i,n=0;
    while(1)
        { cout <<"enter positive number:"; cin >>x; cout<<endl;
          if (x<0) break;
          p=stack;
          if (x==0)cout<<"0";
              while (x!=0)
                  { i=x%2; push(i); n++; x=x/2;}
                  while (n>0)
                      { pop(i); cout<<i<<" ";n--;}
                  cout<<endl; }
    system("pause");
} // end main
```

LIFO



## إستخدام المؤشر مع المكس 1

سنحصل على النتيجة التالية:

enter positive number:23

1 0 1 1 1

enter positive number:123

1 1 1 1 0 1 1

enter positive number:2444

1 0 0 1 1 0 0 0 1 1 0 0

enter positive number:-1

Press any key to continue . . .



## إستخدام الواصلف const مع المؤشرات 1

إذا سبق الواصلف const متغير محدد وعند المحاولة لتغيير قيمته فإن المترجم يلتقط هذه المحاولة ويمنع القيام بها مرسلًا إما رسالة تحذير أو خطأ حسب نوع هذا المترجم، هذا ويمكن استخدام المؤشر مع const في أربعة أشكال:

1- مؤشر غير ثابت لمعطيات غير ثابتة:

نحصل على أعلى مستوى للوصول ( أي يمكن تغيير المعطيات باستخدام المؤشرات عليها ويمكن أيضاً تغيير وجهة المؤشر ليؤشر على معطيات أخرى ). وهنا لا يتضمن التصريح عن المؤشر غير الثابت على المعطيات غير الثابتة على الواصلف const .

2- مؤشر غير ثابت لمعطيات ثابتة:

تستخدم للتأشير على معطيات ثابتة لا يمكن تغيير المعطيات المؤشر عليها. `const int * p;` ويمكن تغيير المؤشر ليؤشر على معطيات أخرى من نفس النمط.

3- مؤشر ثابت لمعطيات غير ثابتة:

يقوم بالتأشير دوماً على نفس موضع الذاكرة الذي يمكن تغيير المعطيات المخزنة فيه باستخدام المؤشر ويذكرنا هذا النوع من المؤشرات بأسماء المصفوفات التي هي عبارة عن مؤشرات ثابتة على أول عنصر من عناصرها، ويمكن الوصول إلى كافة عناصر المصفوفة وتغييرها باستخدام اسم المصفوفة ودليل هذه

العناصر: `int * const p;`

4- مؤشر ثابت لمعطيات ثابتة:

تؤمن أقل إمكانيات الوصول للمعطيات، تؤشر المؤشرات السابقة دوماً على نفس الموضع من الذاكرة وتبقى المعطيات في ذلك الموضع على حالها دون تغيير، يستخدم هذا النوع من المؤشرات لتمرير مصفوفة إلى تابع يقوم باستعراض عناصرها بالاعتماد على أدلة العناصر ولا يستطيع تغيير قيم هذه العناصر كما

`const int * const p;`

هو الحال عند طباعة عناصر المصفوفة أو البحث عن عنصر ضمن مصفوفة .

حالما تتم عملية الترجمة للبرامج المكتوبة بلغة ++C، تقوم هذه البرامج بتنظيم ذاكرة الحاسب إلى أربع مناطق:

- ✓ منطقة تحمل البرنامج المكتوب (program code).
- ✓ منطقة تحمل المتحولات الشاملة (global variables).
- ✓ منطقة تعمل كمكدس (stack).
- ✓ منطقة تعمل كمكوم (heap).

والمكوم هو المساحة من الذاكرة الحرة والتي يتم إدارتها بواسطة معاملات التخصيص الديناميكي `new` و `delete`. يقوم المعامل `new` بتخصيص صريح لذاكرة المؤشر نفسه، فهو يعيد في المؤشر عنوان مجموعة من `s` بايت غير مخصصة بالذاكرة، حيث `s` هو حجم النوع الذي يشير إليه المؤشر، تنسيب هذا العنوان للمؤشر يضمن أن القيمة التي يشير إليها المؤشر غير مستخدمه بواسطة متغير آخر. يستخدم المعامل `new` كمايلي:

```
int *p;           p=new int;           *p=3;
```

حيث تم بداية الإعلان عن مؤشر إلى النوع الصحيح اسمه `p`.

يقوم `new int` بتخصيص عنوان 4 بايت من الذاكرة الحرة (حجم النوع الصحيح) إلى المؤشر `p`، هذا التخصيص يمكننا من تحميل القيمة 3 في العنوان السابق.

الأسطر الثلاثة السابقة يمكن كتابتها بالشكل:

```
int*p=new int; *p=3;
```

```
int *p=new int(3);
```

كذلك هذين السطرين يمكن أن يكتبوا بالشكل:

- لا شيء يسبب المتاعب كاستخدام المؤشر الذي لم يعط قيمة ابتدائية ويمكننا القول أن للمؤشرات مزيج من الميزات الحسنة والسيئة؟
  - فبينما تمدنا المؤشرات بطاقة عظيمة وهي ضرورية في أكثر البرامج إلا أن القيمة الخاطئة للمؤشر من الصعب جداً كشفها.
  - ليس المؤشر بحد ذاته مشكلة, بل المشكلة هي أنه في كل مرة تنجز فيها عملية ما على المؤشر فإننا نقرأ أو نكتب في جزء ما من الذاكرة؟
  - إذا كنا نقرأ منه فأسوأ ما يمكن أن يحدث هو أن نحصل على قيمة عديمة المعنى.
  - عندما نكتب فقد نكتب فوق جزء آخر من برنامجنا أو معطياتنا.
  - وهذا لا يظهر إلا بعد تنفيذ البرنامج وقد يتطلب منا معرفة مكان الخطأ كما ذكرنا سابقاً.
  - من أكثر الأخطاء الناجمة عن استخدام المؤشرات شيوعاً ما يلي:
- الخطأ الناجم عن عدم تهيئة المؤشر (uninitialized pointer).
- الخطأ الناجم عن عدم فهم كيفية استخدام المؤشرات هذا موضح في البرنامج التالي.

## 1- الخطأ الناجم عن عدم تهيئة المؤشر (uninitialized pointer).

```

#include<iostream>
using namespace std;
void main()
{
    int x,*p;           x=10;           *p=x;           cout<<*p<<endl;
    system("pause");
} // end main

```

إن الإعلان `int *p` قام بتخصيص ذاكرة للمؤشر `p` فقط، قيمة المؤشر ستكون عنوان الذاكرة، ولكن الذاكرة في هذه اللحظة لم يتم تخصيصها، هذا يعني أن عنوان الذاكرة هذا قد يكون مخصص بواسطة متغير آخر، لذلك عند تنفيذ البرنامج سيعطي المترجم تحذيراً بأن المؤشر استخدم بدون إعطائه قيمة ابتدائية، ولن ينفذ البرنامج حيث سيتم إعطاء رسالة مفادها أن البرنامج قام بعملية غير شرعية وسيتم إغلاقه وفق نوع المطابق وقد تكون من قبيل .

An unhandled exception of type 'System.NullReferenceException' occurred in pointerAndString.exe

Additional information: Object reference not set to an instance of an object.

`p=&x;`

لتصحيح البرنامج لابد من كتابة العبارة التالية بعد الإعلان عن `x` و `p`:

2- الخطأ الناجم عن عدم فهم كيفية استخدام المؤشرات هذا موضح في البرنامج التالي.

```
#include<iostream>
using namespace std;
void main()
{
    int x,*p;      x=10;  p=x; // p=&x
    cout<<*p<<endl;
    system("pause");
} // end main
```

إلبرنامج واضح أن المقصود منه هو طباعة قيمة  $x$ , لكنه لن ينفذ لأن المترجم سيعطي خطأ وهو عدم إمكانية التحويل من النمط الصحيح إلى النمط مؤشر للنمط الصحيح ورسالة الخطأ هي التالية:

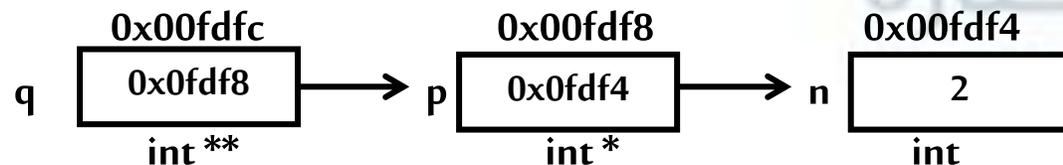
error C2440: '=' : cannot convert from 'int' to 'int \*'

Conversion from integral type to pointer type requires reinterpret\_cast, C-style cast or function-style cast

هذا البرنامج ينفذ بشكل صحيح ويطبع القيمة 10 إذا تم الإسناد قبل الطباعة وليس ما بعد التعريف مباشرة كما في الحالة السابقة:

```
p=&x;
```

```
#include <iostream>
using namespace std;
void main () {int a[4]={2,3,4,5}; int **q,*p;p=&a[0];q=&p;
cout<<"*P = "<< *p<<endl;cout<<"**q = "<< **q<<endl;
for( int i=0;i<4;i++)cout<<*(p+i)<<endl;
cout<<"output using **\n";
for( int i=0;i<4;i++)cout<<*(*q+i)<<endl;
system ("pause");
}
```



*P = 2	output using **
**q = 2	2
2	3
3	4
4	5
5	Press any key to continue ...

انتهت محاضرة الأسبوع 1

من الممكن أن تكون القيمة العائدة من التابع مؤشراً عندئذ لا بد أن يتم التصريح عن التابع على النحو التالي :

```
type *function_name(parameter_lists)
```

يقوم البرنامج التالي بالبحث عن سلسلة ضمن سلسلة أخرى ويعيد تابعه loc مؤشراً على مكان وجود السلسلة الجزئية ضمن الأخرى.

```
#include<iostream>
using namespace std;
int*loc(int*a1,int *a2,int n1,int n2)
{int *endl=a1+n1;
  for(int *p1=a1;p1<endl;p1++)
    if(*p1==*a2)
      { int j;
        for( j=0;j<n2;j++)
          if(p1[j]!=a2[j])break;
          if(j==n2)return p1;
        }
  return 0;
}
```

```
void main()
{
    int a1[9]={11,11,11,11,11,22,33,44,55};
    int a2[5]={11,11,11,22,33};
    cout<<"array a1 begins at location\t"<<a1<<endl;
    cout<<"array a2 begins at location\t"<<a2<<endl;
        int *p=loc(a1,a2,9,5);
    if(p){
        cout<<"array a2 found at location\t"<<p<<endl;
        for(int i=0;i<5;i++)
            cout<<"\t"<<&p[i]<<": "<<p[i]<<"\t"<<&a2[i]<<": "<<a2[i]<<endl;
        }
    else cout<<"not found.\n";
    system("pause");
} // end main
```

## Returning Pointer From Function



## إعادة مؤشر من التابع 2

نتيجة تنفيذ التمرين:

```
array a1 begins at location 00F9EFD4
array a2 begins at location 00F9EFF8
array a2 found at location 00F9EFDC
00F9EFDC:11 00F9EFF8:11
00F9EFE0:11 00F9EFFC:11
00F9EFE4:11 00F9F000:11
00F9EFE8:22 00F9F004:22
00F9EFEC:33 00F9F008:33
Press any key to continue . . .
```



ذكرنا سابقاً طرق استدعاء التابع وذكرنا أنه يمكن استدعاء التابع بالعنوان ولهذا الاستدعاء نوعان الاستدعاء بالمرجع والاستدعاء بالمؤشر. عند استدعاء التابع بالمؤشر يجب على التابع الذي يتلقى عنواناً لوسيط فعلي أن يتضمن في تعريفه مؤشراً كوسيط شكلي. على سبيل المثال ، ليكن البرنامج التالي :

```
#include<iostream.h>
void cube(int *p) {      *p=*p* *p* *p; }

void main()
{      int number=5;   cube(&number);
      cout<<"5 power 3 = "<<number<<endl;
}
```

في هذا المثال يتضمن إعلان التابع cube المؤشر int \* p الذي يتلقى عنوان المتغير الفعلي number (& number) الذي خصصت له القيمة 5 ويتم داخل التابع cube تم تغيير القيمة التي يشير إليها المؤشر إلى 125 أي 5\*5\*5 لذلك أصبح p يشير إلى القيمة 125 وطالما أن p يشير إلى المتغير number لذلك أصبحت قيمة number هي 125. لنحصل على هذه القيمة لا بد من طباعة number بعد استدعاء التابع cube .

يمكن إيضاح استخدام الواصلف const مع المؤشرات من خلال المثال الشامل التالي الذي يتضمن الأشكال الأربعة وقد نفذت التعبيرات الممكنة عليها أما التعبيرات الخاطئة فقد كتبت كتعليق :

```
#include<iostream>
using namespace std;
void main() {
    int x =23;int *p =&x;
    cout<<"x="<<x<<" and *p="<<*p<<endl;
    cout<<"&x="<<&x<<" and p="<<p<<endl;
    cout<<" p is non_constant pointer to non-constant data"<<endl;
    cout<<"which is x so p and x can be changed."<<endl;

    ++x; ++p;
    cout<<"after ++x : x="<<x<<endl; cout<<"after ++p : p="<<p<<endl<<endl;

    int * const cp=&x;
        cout<<"x="<<x<<" and cp="<<cp<<endl;
    cout<<" cp is constant pointer to non-constant data"<<endl;
    cout<<"which is x so only x can be changed ."<<endl;
```

```

++x;
//++cp;error C2166:l-value specifies const object
cout<<"after ++x : x="<<x<<"\n\n\n";

int y=44; const int *cddp=&y;
cout<<"y="<<y<<" and cddp="<<cddp<<endl;
cout<<" cddp is non_constant pointer to constant data"<<endl;
cout<<"which is y so only dp can be changed."<<endl;

cddp++; //>(*cddp)++;error C2166:l-value specifies const object
cout<<"after cddp++ :cddp="<<cddp<<endl;
cout<<endl;

const int* const cdcp=&y;
cout<<"y="<<y<<" and cdcp="<<cdcp<<endl;
cout<<" cdcp is constant pointer to constant data"<<endl;
cout<<"which is y so cdcp and y can not be changed ."<<endl;
//++(*cdcp);error C2166:l-value specifies const object
//++cdcp;error C2166:l-value specifies const object

system("pause");
} // end main

```

## Using The Const Qualifier With Pointers



## إستخدام الواصلف const مع المؤشرات 5

**x=23 and \*p=23**

**&x=00CFF09C and p=00CFF09C**

**p is non\_constant pointer to non-constant data  
which is x so p and x can be changed.**

**after ++x : x=24**

**after ++p : p=00CFF0A0**

**x=24 and cp=00CFF09C**

**cp is constant pointer to non-constant data  
which is x so only x can be changed .**

**after ++x : x=25**

**y=44 and cddp=00CFF090**

**cddp is non\_constant pointer to constant data  
which is y so only dp can be changed.**

**after cddp++ :cddp=00CFF094**

**y=44 and cdcp=00A2EC50**

**cdcp is constant pointer to constant data  
which is y so cdcp and y can not be changed .**

**Press any key to continue . . .**



1. مثال يوضح تطبيق مفيد لصف المؤشرات, ويمكن من ترتيب صف بطريقة غير مباشرة بتغيير المؤشرات للعناصر بدلاً من تحريك العناصر نفسها.

2. استخدام صف من المؤشرات للتوابع: أنه يمكن أن يطلب من مستخدم اختيار أحد الخيارات ضمن قائمة ما ( يمكن اختيار من 1 حتى 5 خيارات ), ويتعامل كل خيار من الخيارات السابقة مع تابع محدد, لإنجاز ذلك يمكن استخدام مؤشر على كل تابع من التوابع المعرفة, وتخزن ضمن مصفوفة من المؤشرات على توابع, ويستفاد من خيار المستخدم على أنه دليل عنصر المصفوفة المطلوب والذي هو عبارة عن مؤشر على التابع المطلوب استدعاؤه.

3. تطوير تمرين استخدام المؤشر مع المكس ليتعامل مع مختلف أمثلة العد مثل رباعي، ثماني، سداسي عشر.



## Non-constant pointer to non-constant data

الشكل العام `int *p;` ( أي غياب الواصل نستطيع تغيير القيمة وتغيير موضع المؤشر )

```
#include "stdafx.h"
#include<iostream>
using namespace std;

void changetopositive(int *p)
{for(int i=1;i<=6;i++) { *p=-1*(*p); ++p; } }

void main(){ int a[]={-5,-6,-7,-8,-9,-10};int i; cout<<"a array : ";
for( i=0;i<6;i++) cout<<a[i]<<" "; cout<<endl;
changetopositive(a);
cout<<"a array after the change is :"<<endl;
for(i=0;i<6;i++) cout<<a[i]<<" "; cout<<endl;
system("pause"); }// end main
```

a array : -5 -6 -7 -8 -9 -10

a array after the change is :

5 6 7 8 9 10

Press any key to continue...

non\_constant pointer to constant data



## إستخدام الواصف const مع المؤشرات 2

( أي لانستطيع تغيير القيمة ونستطيع تغيير موضع المؤشر )

الشكل العام; const int \* p;

```
#include<iostream>
using namespace std;
void changetopositive (const int *p)
{for(int i=1;i<=6;i++) { *p=-1*(*p); ++p; } }
void main(){ int a[]={-5,-6,-7,-8,-9,-10};int i; cout<<"a array : ";
for( i=0;i<6;i++) cout<<a[i]<<" "; cout<<endl;
changetopositive(a);
cout<<"a array after the change is :"<<endl;
for(i=0;i<6;i++) cout<<a[i]<<" "; cout<<endl;
system("pause"); }// end main
```

سنحصل على رسالة الخطأ التالية:

error C3892: 'p' : you cannot assign to a variable that is const

non\_constant pointer to constant data



## إستخدام الواصف const مع المؤشرات 3

(يمكن استخدام النوع السابق من المؤشرات مع الصفوف التي يراد تمريرها إلى تابع يقوم بمعالجة كل عنصر من عناصرها دون تغيير قيمتها, مثل طباعة عناصر المصفوفة, البحث عن عنصر فيها)

```
#include<iostream>
using namespace std;

void printarray(const int *p)
{for(int i=1;i<=6;i++) {cout<<*p<<" "; p++; }
  cout<<endl;
}

void main()
{  int a[]={-5,-6,-7,-8,-9,-10};
  cout<<"a array is : "; printarray(a);      system("pause");
}// end main
```

سنحصل على النتيجة التالية:

```
a array is : -5 -6 -7 -8 -9 -10
Press any key to continue . . .
```

constant pointer to non-constant data



## إستخدام الواصف const مع المؤشرات 4

( أي نستطيع تغيير القيمة ولانستطيع تغيير موضع المؤشر )

الشكل العام; `int *const p;`

```
#include<iostream>
using namespace std;
void changetopositive( int* const p)
{
    for(int i=0;i<6;i++)        *(p+i)=-*(p+i);}
void main()
{
    int i, a[]={-5,-6,-7,-8,-9,-10};        cout<<"a array is : ";
    for( i=0;i<6;i++)cout<<a[i]<<" ";        cout<<endl;
    changetopositive(a);        cout<<"a array after the change is : ";
    for(i=0;i<6;i++)cout<<a[i]<<" ";cout<<endl;    system("pause");
}// end main
```

لا يمكن استخدام P++ وسنحصل على النتيجة التالية :

a array is : -5 -6 -7 -8 -9 -10

a array after the change is : 5 6 7 8 9 10

Press any key to continue . . .

## Const pointer to const data



## إستخدام الواصف const مع المؤشرات 5

(أي لانستطيع تغيير القيمة ولانستطيع تغيير موضع المؤشر)

الشكل العام: `const int *const p;`

```
#include<iostream>
using namespace std;
void printarray(const int * const p)
{ for(int i=0;i<6;i++) { cout<<*(p+i)<<" "; }
}
void main()
{int a[]={-5,-6,-7,-8,-9,-10};
  cout<<"a array is : ";
  printarray(a);
  system("pause");
} // end main
```

`cout<<endl;`

a array is : -5 -6 -7 -8 -9 -10  
Press any key to continue . . .

سنحصل على النتيجة التالية: