Pointers المؤشرات

1. Introducing pointer
2. Declaring reference
3. Pointer Arithmetic العمليات الحسابية على المؤشرات
4. Pointers and Const المؤشرات والثوابت
5. The relation between pointer and arrays العلاقة بين المؤشرات والمصفوفات

المؤشرات Pointers

المؤشرات عبارة عن متغيرات تحتوي على عناوين في الذاكرة كقيم مخزنة ضمنها، يحتوي المتغير عادة قيمة محددة مخزنة مباشرة ضمنه لكن المؤشر يتضمن عنواناً في الذاكرة لمتغير يحتوي على قيمة محددة.

يجب الإعلان عن المؤشرات قبل استخدامها ويجب أن ننسب له عنوان ويمكن أن يكون عند التصريح أو بعد ذلك ويمكن نسب الصفر أوNULL لكن يُفضل NULL، لأنه يبرز حقيقة أن المتغير من نوع المؤشر.

وفي هذه الحالة يؤشر إلى لا شئ وهى قيمة معرفة ضمن الملفات الرأسية لمكتبة التوابع المعيارية

الإعلان عن المؤشر Declaring pointer

يعلن عن المؤشر بالشكل:

type * namepointer ;

حيث أن:

type: نوع القيمة التي يشير لها المؤشر.

namepointer : اسم المؤشر.

* : معامل عندما يستخدم بالشكل السابق يعلن أن ما بعده متغير من نوع مؤشر يشير إلى قيمة من النوع الذي قبله.

الإعلان التالي:

```
int * x ;
```

يعلن أن xمؤشر لرقم صحيح.

أما الإعلان :

```
int * x , y ;
```

يعلن أن xمؤشر لرقم صحيح في حين أن yمتغير صحيح.

ولكي يكون xو yمتغيرات من النوع مؤشر لعدد صحيح يجب أن يعلن عنهما بالشكل:

```
int *x, *y ;
```

Following are the valid pointer declaration:

```
int *ip;        // pointer to an integer
double *dp;     // pointer to a double
float *fp;      // pointer to a float
char *ch ;      // pointer to character
```

مؤثرات المؤشرات Pointer operators

هنالك مؤثران خاصان بالمؤشرات هما *و & .

المؤثر & هو مؤثر أحادي يرجع عنوان الذاكرة لمعامله.

```
int x;
int y = 10;
const int * p = &y;
x = *p; // ok: reading p
*p = x; // error: modifying p, which is const-qualified
```

التصريح عن المؤشرات

يجب تحديد المؤشرات ، مثل جميع المتغيرات ، قبل استخدامها.

التعريف

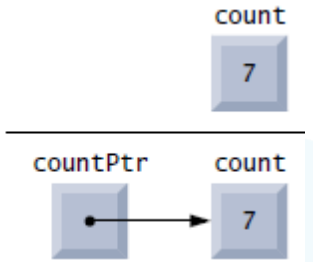
int *countPtr, count;

يجب تحديد المؤشرات ، مثل جميع المتغيرات ، قبل استخدامها.

يمكن أن يحتوي المؤشر على عنوان متغير يحتوي على قيمة خاصة. يشيراسم المتغير مباشرة إلى قيمة ومؤشر بشكل غير مباشر الى العنوان

The name count directly references

a variable that contains the value 7



The pointer countPtr indirectly

references a variable that contains the value 7

يمكن أيضًا استخدام التمرير بالمرجع لتمكين دالة من "إرجاع" قيم متعددة إلى المتصل بها عن طريق تعديل المتغيرات في المتصل

Use & and * to Accomplish Pass-By-Reference

```
int *ip; // pointer to an integer
double *dp; // pointer to a double
float *fp; // pointer to a float
char *ch; // pointer to character
```

استخدامات مختلفة لعلامة النجمة

تستخدم علامة النجمة (*) بطريقتين مختلفتين مع المؤشرات:

كجزء من إعلان المؤشر وأيضًا كعامل إشارة.

عندما تقوم بتعريف مؤشر ، يكون * جزءًا من الإعلان ويتبع نوع ملف أشار الكائن إليه. على سبيل المثال

```
unsigned short * pAge = 0;
```

عندما يتم إلغاء الإشارة إلى المؤشر ، يشير عامل الإسناد (أو غير المباشر) أن القيمة الموجودة في موقع الذاكرة المخزنة في المؤشر يجب الوصول إليها ، وليس العنوان نفسه.

```
// إسناد 5 إلى القيمة عند pAge* *pAge = 5;
*pAge = 5;
```

لاحظ أيضًا أن هذا الحرف نفسه (*) يُستخدم كعامل الضرب.

المترجم يعرف أي عامل يجب الاتصال به بناءً على كيفية استخدامه (السياق)

المؤشرات والعناوين والمتغيرات

Pointers, Addresses, and Variables

من المهم التمييز بين المؤشر والعنوان الذي يحمله المؤشر والقيمة في العنوان الذي يحمله المؤشر.

```
int theVariable = 5;
int * pPointer = &theVariable ;
```

تستخدم في أغلب الأحيان لثلاث مهام:

- إدارة البيانات
- الوصول إلى بيانات ودوال أعضاء الصف
- تمرير المتغيرات بالمرجع إلى الدوال
- كيفية اجتياز المتغيرات باستخدام المؤشرات ، وهو ما يسمى بالتمرير بالمرجع

يرتبط الإعلان عن المتغير من نوع pointer ثلاثة مفاهيم أساسية:
اسم المتغير، نوع المتغير، عنوان المتغير في الذاكرة.
حيث أن الإعلان وفق الشكل التالي: int n;
. يربط بين الاسم n والنوع int وعنوان المتغير في الذاكرة

من المعلوم أن عنوان المتغير ضمن الذاكرة يتحدد من قبل إدارة الذاكرة ومكان توضع المتغير ضمن البرنامج حيث يتم تخصيص
1- نطاق للمتغيرات العامة variable Global
2- نطاق آخر للمتغيرات المحلية variable locale
3- ونطاق للمتغيرات الديناميكية variable dynamic وتختلف هذه المجالات وفق أنظمة التشغيل المستخدمة

يمكن التعامل مع قيمة المتغير بواسطة اسمه، فمثلاً يمكن طباعة قيمة المتغير n بالأمر التالي: cout << n;

أما عنوان المتغير يمكن التعامل معه بعامل العنوان & ( ) operator address
فمثلاً يمكن طباعة عنوان المتغير n بالأمر التالي:

```
cout << & n ;
```

أما عنوان المتغير يمكن التعامل معه بعامل العنوان & ( ) operator address
فمثلاً يمكن طباعة عنوان المتغير n بالأمر التالي:

```
cout << & n ;
```

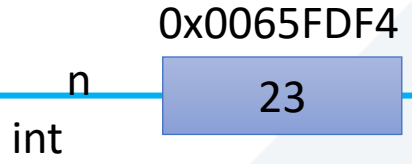
عامل العنوان & يسبق باسم المتغير لينتج العنوان وله أسبقية على عامل النفي المنطقي وعامل الزيادة المسبقة ++

مثال

```
#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;
}
```

OUTPUT
the value of n is: n=23
the address of n is: &n=0x6ffe0c

من خرج البرنامج يتضح العنوان 0x0065FDF4 الذي خزن فيه المتغير n في الذاكرة، وهذا العنوان يبدأ بالبادئة " 0 x"وهي بادئة تدل بأن العدد مكتوب بالنظام الست عشري.

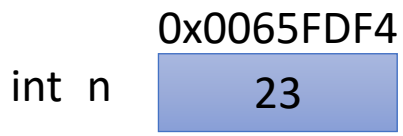


```
#include <iostream>
using namespace std;
int main ()
{
 int firstvalue = 5, secondvalue = 15;
 int *p1, *p2;
 p1 = &firstvalue;          // p1 = address of firstvalue
 p2 = &secondvalue;       // p2 = address of secondvalue
 *p1 = 10;                    // value pointed to by p1 = 10
 *p2 = *p1;          // value pointed to by p2 = value pointed by p1
 p1 = p2;              // p1 = p2 (value of pointer is copied)
 *p1 = 20;             // value pointed by p1 = 20

 cout << "firstvalue is " << firstvalue << '\n';
 cout << "secondvalue is " << secondvalue << '\n';
 return 0;
}
```

حيث يمثل الصندوق مكان تخزين المتغير في الذاكرة، اسم المتغير على اليسار، عنوان المتغير من الأعلى، ونوع المتغير أسفل الصندوق.

int n=23;



أما عنوان المتغير يمكن التعامل معه بعامل العنوان & ( ) operator address
فمثلاًيمكن طباعة عنوان المتغير بالأمر التالي

cout << & n ;

عامل العنوان & يسبق باسم المتغير لينتج العنوان وله أسبقية على عامل النفي المنطقي وعامل الزيادة المسبقة ++ .

Output
firstvalue is 10
secondvalue is 20

الإعلان بالمرجع:Declaring reference

يعلن عن متغير صحيح اسمه nوخصصت له قيمة ابتدائية مقدارها ،23 كما

```
int n = 23 ;
```

أعلن عن متغير rبأنه مرجع لـ : n

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

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

OUTPUT

```
the value of n is: n=23
the value of r is: r=23
&n=0x6ffe04
&r=0x6ffe04

the value of n is: n=22
the value of r is: r=22

the value of n is: n=44
the value of r is: r=44
```

عندما يعرف rعلى أنه مرجع لـ nهذا يعني أن قيمة rهي ذاتها قيمة nوأي تغير في قيمة n سيصيب قيمة rوالعكس صحيح بمعنى أن أي تغير في قيمة rسيصيب قيمة n، أضف إلى ذلك فإن nو rلهما نفس العنوان في الذاكرة، وبالتالي فإن nو rهما اسمان رمزيان لنفس المكان في الذاكرة.

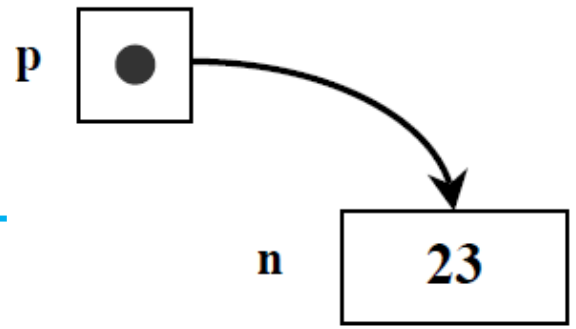
المتغيران nو rهما اسمان مختلفان لنفس المتغير، ودائماً لهما نفس القيمة، إنقاص قيمة nيغير قيمة كل من nو r إلى 22 ومضاعفة rيزيد كل من nو rإلى 44 .
كما أن لـ n و r نفس العنوان 0x6ffe04 في الذاكرة.

برنامج يعيد عنوان وقيمة المتغير والمؤشر.

```
#include<iostream.h>
void main()
     {
       int n=23;
       int *p=&n;// p holds the address of 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;
      }
```

Output

```
the value of n is : n = 23
the address of n is : &n = 0x0065FDF4
the value of p is : p = 0x0065FDF4
the address of n is : &p = 0x0065FDF0
```



عندما نكتب:

```
int * p = & n;
```

أو

```
int *p;
p=&n;
```

فإننا نضع في p عنوان المتغير n في الذاكرة وهذا العنوان هو الموقع الداخلي في الحاسب لهذا المتغير.

إذن يمكن أن نقول أن عمل & هو إرجاع عنوان لمتغير ما، ولهذا يمكننا قراءة التعليمة p يستقبل عنوان n

نلاحظ من خرج البرنامج أن المتغير n استخدم العنوان 0x0065FDF4 لتخزين القيمة 23 لذلك وبعد تنفيذ المساواة

```
int*n = & p;
```

سيتم إسناد هذا العنوان إلى المؤشر p وبالتالي سيملك p قيمة 0x0065FDF4.

هناك بعض الخطوات المهمة للمؤشرات في كثير من الأحيان:

أ) نحدد متغير المؤشر.

ب) تعيين عنوان متغير لمؤشر.

ج) الوصول إلى القيمة على العنوان المتاح في المؤشر.

- يتم ذلك باستخدام عامل التشغيل الأحادي * الذي يُرجع قيمة المتغير الموجود في العنوان المحدد بواسطة المعامل الخاص به.

```
#include <iostream>
using namespace std;
int main () {
 int var = 20;     // actual variable declaration.
 int *ip;     // pointer variable
 ip = &var;     // store address of var in pointer variable
 cout << "Value of var variable: ";
 cout << var << endl;
          // print the address stored in ip pointer variable
 cout << "Address stored in ip variable: ";
 cout << ip << endl;
    // access the value at the address available in pointer
 cout << "Value of *ip variable: ";
 cout << *ip << endl;
 return 0;
}
```

بعد التنفيذ خرج البرنامج

```
Value of var variable: 20
Address stored in ip variable: 0xbfc601ac
Value of *ip variable: 20
```

إخراج قيمة المؤشر وما يشير إليه

للمؤشرات * كمؤشر على المؤشر.

المؤثر * مؤثر متمم للمؤثر & وهو مؤثر أحادي يرجع قيمة المتغير الموجود في العنوان الذي يشكل معامله.

إن عامل العنوان & وعامل إعادة المرجعية * عاملان متتامان حيث أن:
وأيضاً n = =*&n، ويمكن التعبير عن ذلك أيضاً بـ p = =&n عندما n = = *p
حيث نجد &* ما قبل المتغير و *& ما قبل المؤشر.. p == &*p

```
#include<iostream.h>
void main()
{
      int n=23;
      int *p=&n;//p point to n
      cout<<"n="<<n<<" and &n= "<<&n<<endl;
      cout<<"*p="<<*p<<" and p="<<p<<endl;
}
```

Output
n=23 and &n= 0x0065FDF4
*p=23 and p=0x0065FDF4

وبالتالي فإن التعليمة : n يحوي عنوان المتغير p نلاحظ من خرج البرنامج أن
cout<<*p;
ستطبع قيمة المتغير p، وهي القيمة التي يشير لها المؤشر n

```
#include<iostream.h>
void main()
   {
      int n=23;
      int *p=&n;
      int &r=*p;
      cout<<"r="<<r<<endl;
   }
```

خرج البرنامج:

r =23

بعض المفاهيم حول المؤشرات في C ++

1	**مؤشر القيمة الفارغة Null pointers .** يدعم C ++ المؤشر الفارغ ، وهو ثابت بقيمة تم تعريف الصفر في العديد من المكتبات القياسية.
2	Pointer Arithmetic العمليات الحسابية هناك أربع معاملات حسابية يمكن استخدامهاالمؤشرات: ++ ، - ، + ، -
3	**المؤشرات والمصفوفات Pointers vs. Arrays** هناك علاقة وثيقة بين المؤشرات والمصفوفات **مصفوفة المؤشرات Array of Pointers .** يمكنك تحديد المصفوفات لتحتوي على عدد من المؤشرات.
4	**مؤشر لمؤشرPointer to Pointer** يتيح لك C ++ أن يكون لديك مؤشر على مؤشر
5	**تمرير المؤشرات إلى الدوالPassing pointers to functions** تمرير وسطاء بالمرجع أو عن طريق العنوان على حد سواء تمكين يتم تغيير الوسيطة التي تم تمريرها في دالة الاستدعاء بواسطة لدالة المطلوبة.
6	**مؤشر العودة من الدالة** يسمح C ++ لدالة بإرجاع مؤشر إلى متغير محلي ،متغير ثابت وذاكرة مخصصة ديناميكيًا أيضًا

Null pointers مؤشرات القيمة الفارغة

يدعم C ++ المؤشر الفارغ ، وهو ثابت بقيمة صفر محدد في العديد من المكتبات القياسية.

مؤشر فارغ

لمتغير المؤشر في NULL• من الممارسات الجيدة دائمًا تعيين المؤشر حالة عدم وجود عنوان دقيق لتعيينه.

- من الممارسات الجيدة دائمًا تعيين المؤشر NULL لمتغير المؤشر في حالة عدم وجود عنوان دقيق لتعيينه.
- يتم ذلك في وقت إعلان المتغير. وهو مؤشر يسمى NULL المعيّن بمؤشر فارغ.
- المؤشر الفارغ هو ثابت بقيمة صفر معرّفة في العديد من المكتبات القياسية ، بما في ذلك iostream.

```
#include <iostream>
using namespace std;
int main () {
 int *ptr = NULL;
 cout << "The value of ptr is " << ptr ;
 return 0;
}
```

Output

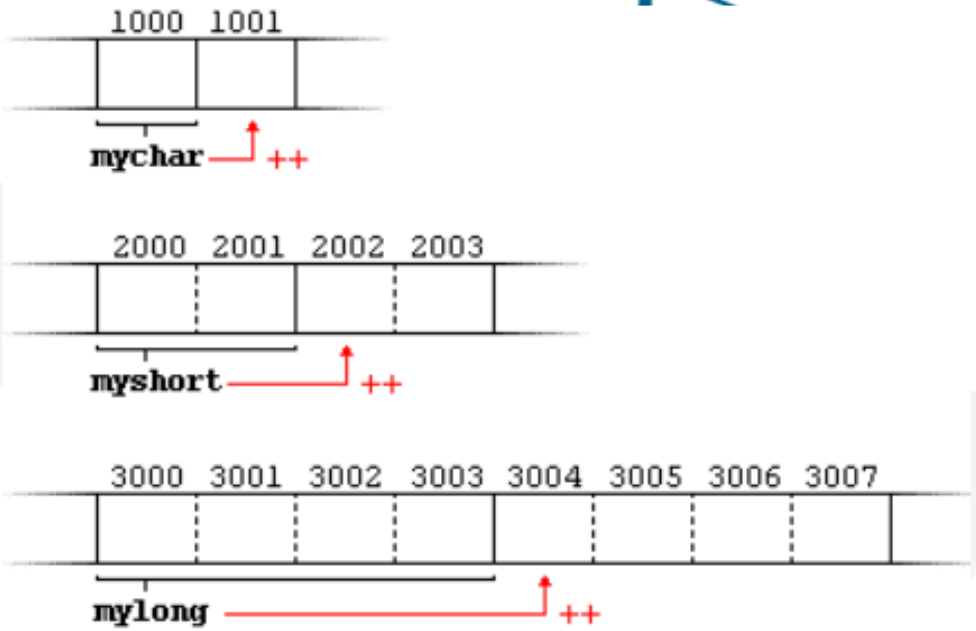
The value of ptr is 0

العمليات الحسابية على المؤشرات Pointer Arithmetic

تعتمد عمليات الجمع والطرح على حجم نوع البيانات
التي يشير اليها المؤشر
مثال
المؤشر من نوع char حجم البيانات 1byte
من نوع short 2 byte
من نوع int 4 byte
من نوع float 4 byte
من نوع double 8 byte

```
char *mychar;
short *myshort;
long *mylong;

  ++mychar;
  ++myshort;
  ++mylong;
```




```
mychar = mychar + 1;
myshort = mychar + 1;
mylong = mychar + 1;
```

start at address 1000, 2000 and 3000

their values will end up as 1001, 2002, and 3004

```
  mychar = mychar + 3;
  myshort = mychar + 3;
  mylong = mychar + 3;
```

Would end up as 1003, 2006 and 3012

اضافة على العنوان بحسب نوع البيانات

*p++ // same as *(p++): increment pointer, and dereference unincremented address
*++p // same as *(++p): increment pointer, and dereference incremented address

اضافة على قيمة المؤشر بحسب نوع البيانات

++*p // same as ++(*p): dereference pointer, and increment the value it points to
(*p)++ // dereference pointer, and post-increment the value it points to

فهم العمليات الحسابية على المؤشر ،

نعتبر أن ptr هو مؤشر عدد صحيح يشير إلى العنوان 1000.

- بافتراض الأعداد الصحيحة 32 بت ، int * ptr;

دعونا نجري العمليات الحسابية التالية عملية على المؤشر:

ptr ++;

سيشير ptr إلى الموقع 1004 لأنه في كل مرة يكون ptr ستشير إلى العدد الصحيح التالي.

- ستحرك هذه العملية المؤشر إلى موقع الذاكرة التالي دون التأثير على القيمة الفعلية في موقع الذاكرة.
- إذا كان ptr يشير إلى حرف عنوانه 1000 ، ثم أعلاها لعملية ستشير إلى الموقع 1001 لأن الحرف التالي سيكون متاحًا في 1001.
- يقوم البرنامج التالي بزيادة مؤشر المتغير للوصول إليه كل عنصر لاحق من المصفوفة:

```
#include <iostream>
using namespace std;
const int MAX = 3;
int main ()
{
 int var[MAX] = {10, 100, 200};
 int *ptr;
 // let us have array address in pointer.
 ptr = var;

 for (int i = 0; i < MAX; i++) {
 cout << "Address of var[" << i << "] = ";
 cout << ptr << endl;
 cout << "Value of var[" << i << "] = ";
 cout << *ptr << endl;
 // point to the next location
 ptr++;}
return 0;
}
```

output

```
Address of var[0] = 0x6ffe00
Value of var[0] = 10
Address of var[1] = 0x6ffe04
Value of var[1] = 100
Address of var[2] = 0x6ffe08
Value of var[2] = 200
```

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

```
the value of n is : n=23
the address of n is : &n=0x6ffe0c
the value of p is : p=0x6ffe0c
the address of n is : &p=0x6ffe00
the value of p is : p=23
```

```
int x;
int y = 10;
const int *p = &y;
x = *p;        // ok: reading p
*p = x;        // error: modifying p, which is const-qualified
```

المؤشرات والثوابت Pointers and Const

Pointers can be used to access a variable using it's address, and may also be used to modify the value at that address.

استخدام المقارنة من المؤشرات

```
#include<iostream.h>
void main()
{
int n=23;
int x=29;
int y=25;
 int *p1=&n;
int *p2=&y;
cout<<"&n="<<&n<<" and p1="<<p1<<endl;
cout<<"&y="<<&y<<" and p2="<<p2<<endl;
 if(p1<p2)
cout<<"p1 points to lower memory than p2."<<endl;
 else
 cout<<"p2 points to lower memory than p1."<<endl;
}
```

نلاحظ من خرج البرنامج ما يلي:

زيادة المؤشر بـ 1 ستزيد قيمة هذا المؤشر بمقدار حجم النوع الذي يشير إليه المؤشر.

زيادة المؤشر بـ 2 ستزيد قيمة هذا المؤشر بمقدار حجم النوع الذي يشير إليه المؤشر

لنوع n ا.ستزيد قيمة المؤشر بمقدار حجم nمضروباً بـ 2 ،وبالتالي فإن زيادة المؤشر بـ

الذي يشير إليه المؤشر مضروباً بـ

Output

```
& n=0x0065FDF4 and p1=0x0065FDF4
& y=0x0065FDEC and p2=0x0065FDEC
p2 points to lower memory than p1.
```

تعابير المؤشرات the pointer expression

يمكن استخدام المؤشرات في التعابير الحسابية وتعابير الإسناد والمقارنة

استخدام التابع sizeof () لمعرفة حجم النوع:
يستخدم التابع sizeof () لمعرفة حجم النوع
ويكون بوضع النوع كقيمة مدخلة للتابع أي مابين قوسي التابع

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

```
number of bytes used:
        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 :16
```

2. المؤشرات والعمليات الحسابية Pointer Arithmetic

تختلف العمليات الحسابية على المؤشرات قليلاً عن العمليات الحسابية على أنواع الأعداد الصحيحة العادية.

أولاً ، يُسمح فقط بالجمع والطرح ؛ لأن العمليات الأخرى لا معنى لها للمؤشرات (التي تشير فقط إلى عناوين الذاكرة).

أيضًا ، يعمل الجمع والطرح بشكل مختلف ، اعتمادًا على حجم نوع البيانات التي يشير إليها المؤشر

هناك أربع معاملات حسابية يمكن استخدامها المؤشرات: ++ ، - ، + ، -

A typical, but not so simple statement is:

```
*p++ = *q++;
```

This is roughly equivalent to:

```
*p = *q;
++p;
++q;
```

ملاحظة :

يوصي بشدة باستخدام الأقواس لمنع الالتباس مع هذا النوع من البيانات لأن ++ لها أسبقية أعلى من *

يمكن استخدام المؤشرات للوصول إلى متغير باستخدام عنوانه ويمكن أيضًا استخدامها لتعديل القيمة في هذا العنوان.

بل هو أيضا من الممكن إعلان المؤشرات إلى القيم التي يمكنها الوصول إلى القيمة أشار إليه وقرائته ، لكن دون تعديله.

نلاحظ من خرج البرنامج ما يلي:
زيادة المؤشر بـ 1 ستزيد قيمة هذا المؤشر بمقدار حجم النوع الذي يشير إليه المؤشر.
زيادة المؤشر بـ 2 ستزيد قيمة هذا المؤشر بمقدار حجم النوع الذي يشير إليه المؤشر مضروباً بـ 2 ،وبالتالي فإن زيادة المؤشر بـ n ستزيد قيمة المؤشر بمقدار حجم النوع الذي يشير إليه المؤشر مضروباً بـ n
كذلك الأمر بالنسبة لإنقاص المؤشر بـ 1 سينقص من قيمة هذا المؤشر بمقدار حجم النوع الذي يشير إليه المؤشر وهكذا ....
طرح مؤشر من مؤشر سيعيد عدد العناصر من النوع المشار إليه والتي يمكن تخزينها بين العنوانين الذين يشير إليهما المؤشران.

```
#include<iostream>
using namespace std;
 int 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;
cout<<"*p1="<<*p1<<endl;
}
```

```
&n=0x6ffdfc and p=0x6ffdfc
after p++ : p=0x6ffe00
after p+=2: p=0x6ffe08
after p-- : p=0x6ffe04
after p-=5: p=0x6ffdf0
p1=0x6ffdfc
p1-p=3
*p1=23
```

```
#include<iostream>
#include<conio.h>
using namespace std;
int main( )
{
int a, b, x, y;
int *ptr1, *ptr2;
a = 30;
b = 6;
ptr1 = &a;
ptr2 = &b;
x=*ptr1 + *ptr2 - 6;
y = 6 - *ptr1 / *ptr2 + 30;
cout<<"Address of a = "<<ptr1<<endl;
cout<<"Address of b = "<<ptr2<<endl;
cout<<"a = "<<a<<"   "<<"b = "<<b<<endl;
cout<<"x = "<<x<<"   "<<"y = "<<y<<endl;
*ptr1 = *ptr1 + 70;
*ptr2 = *ptr2 * 2;
cout<<"a = "<<a<<"b = "<<b<<endl;
}
```

نلاحظ مما سبق أن المؤشرات الحسابية تفقد الكثير من محتواها إلا إذا طبقت على المصفوفات، لأننا لا نستطيع أن نفترض أن متحولين من نفس النمط متجاوران في الذاكرة إلا إذا كانا عنصرين متجاورين من عناصر مصفوفة.

يمكن إجراء العملية التالية على المؤشرات.

- يمكننا إضافة قيمة عددية إلى المؤشر.
- يمكننا طرح قيمة عدد صحيح من المؤشر.
- يمكننا مقارنة مؤشرين ، إذا كانا يشيران إلى عناصر نفس المصفوفة.
- يمكننا طرح مؤشر واحد من مؤشر آخر إذا كان كلاهما يشير إلى نفس المصفوفة.
- يمكننا تعيين مؤشر واحد لمؤشر آخر بشرط أن يكون كلاهما من نفس النوع.

لا يمكن إجراء العمليات التالية على المؤشرات

1. إضافة اثنين من المؤشرات.
2. طرح مؤشر واحد من مؤشر آخر عندما لا يشيرون إلى نفس المصفوفة.
3. ضرب مؤشرين.
4. قسمة مؤشرين..

Output

```
Address of a = 0x6ffdf4
Address of b = 0x6ffdf0
a = 30   b = 6
x = 30   y = 31
a = 100b = 12
```

العلاقة بين المؤشرات والمصفوفات:
The relation between pointer and arrays

توجد علاقة وثيقة بين المؤشرات و الصفوف في لغة C ++ ويمكن استخدام أحدها للحلول مكان الآخر في معظم الحالات، حيث يشبه اسم الصف في عمله عمل مؤشر ثابت، كما أنه يمكن استخدام المؤشرات للقيام بأي عملية تستدعي استخدام صف مع تحديد أدلة عناصره المعنية.

بفرض أنه تم التصريح عن مصفوفة اللأعداد الصحيحة a [4] وعن مؤشر p للنمط الصحيح، وطالما أن اسم المصفوفة يشير إلى بداية المصفوفة أي إلى أول عنصر في المصفوفة (بمعنى أنه يشير إلى عنوان أول عنصر في المصفوفة) فإنه يمكن إسناد اسم المصفوفة إلى المؤشر p ليشير p إلى بداية المصفوف .

ذلك كما يلي:

```
p = a ;
```

إن التعليمة السابقة تكافئ :

```
p = & a[ 0 ] ;
```

أي تم نسب عنوان العنصر الأول في الصف إلى المؤشر p.

أضف إلى ذلك فإن شيفرة استدعاء التابع بالصف :

```
int a[ ] ⟺ int * a
```

يمكن الوصول إلى العنصر الرابع في الصف a [3] باستخدام التعبير:

```
cout << * ( p+3 ) ;
```

الاختلاف الرئيسي يمكن تعيين للمؤشرات عناوين جديدة
بينما لا يمكن تعيين المصفوفات

بفرض أنه تم التصريح عن صف للأعداد الصحيحة a [4] وعن مؤشر p للنمط الصحيح، وطالما أن اسم الصف يشير إلى بداية الصف أي إلى أول عنصر في الصف (بمعنى أنه يشير إلى عنوان أول عنصر في الصف) فإنه يمكن إسناد اسم الصف إلى المؤشر p ليشير p إلى بداية الصف.

ذلك كما يلي:

```
p = a ;
```

إن التعليمة السابقة تكافئ :

```
p = & a[ 0 ] ;
```

أي تم نسب عنوان العنصر الأول في الصف إلى المؤشر p.

أضف إلى ذلك فإن شيفرة استدعاء التابع بالصف :

int a[ ] ⟺ int * a

يمكن الوصول إلى العنصر الرابع في الصف a[3] باستخدام التعبير:

```
cout << * ( p+3 ) ;
```

حيث أن (p+3) * هي القيمة التي يشير إليها المؤشر p بعد زيادته بمقدار 3.

مما سبق نستنتج أنه إذا أشر مؤشر 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.

وعندما تكتب العبارة السابقة بدون أقواس أي p + 3 *فإن ذلك يعني إضافة 3 إلى العنصر a[0] (بفرض أن p مؤشر مايزال يدل على بداية الصف) .

كذلك الأمر فإن :

& a[3] ⟺ p+3

أيضاً يمكن أن نعامل اسم الصف نفسه على أنه مؤشر ويمكن استخدامه في العمليات الحسابية فمثلاً التعبير: (a + 3) *يعطي قيمة العنصر a [3] من عناصر الصف.

إن التعبير السابق لا يغير اسم الصف بأي شكل من الأشكال بل يبقى a يؤشر إلى بداية الصف

```
#include<iostream>
using namespace std;
int main( )
{
int a[10], i, n;
cout<<"Enter the input for array";
cin>>n;
cout<<"Enter array elements:";
for(i=0; i<n; i++)
cin>>*(a+i);
cout<<"The given array elements are :"<<endl;
for(i=0; i<n; i++){
cout<<"\t"<<*(a+i);
cout<<"\t"<<(a+i);}
return (0);
}
```

Output

```
Enter the input for array   5
Enter array elements:  3    5    6    7    8
The given array elements are :
        3       0x6ffde0
        5       0x6ffde4
        6       0x6ffde8
        7       0x6ffdec
        8       0x6ffdf0
```

ترتبط المصفوفات ارتباطًا وثيقًا بالمؤشرات ، وفي الواقع تكون المصفوفة بشكل عام مجرد مؤشر مع بعض الصيغ المختلفة. من الممكن دائمًا التحويل بين مصفوفة ومؤشر من النوع المناسب:

```
int myarray [20];
int * mypointer;
mypointer = myarray;
```

تدعم المؤشرات والمصفوفات نفس العمليات بنفس المعنى لكليهما. الاختلاف الرئيسي يمكن تعيين المؤشرات عناوين جديدة بينما لا يمكن تعيين المصفوفات

مثال : اكتب برنامج لادخال عناصر مصفوفة أحادية البعد باستخدام المؤشرات وطباعة هذه العناصر

```
// more pointers
#include <iostream>
using namespace std;
int main ()
{
 int numbers[5];
 int * p;
 p = numbers; *p = 10;
 p++; *p = 20;
 p = &numbers[2]; *p = 30;
 p = numbers + 3; *p = 40;
 p = numbers; *(p+4) = 50;
 for (int n=0; n<5; n++)
 cout << numbers[n] << “   ";
 return 0;
```

Output

10 20 30 40 50

مثال : اكتب برنامج يستخدم المؤشرات لقراءة عناصر مصفوفة أحادية a[n] يتم ادخال قيمة n من لوحة المفاتيح البعد وطباعة عناصرها

```
#include<iostream.h>
void main( )
{
int a[10], i, n;
cout<<”Enter the input for array”;
cin>>n;
cout<<”Enter array elements:”;
for(i=0; i<n; i++)
cin>>*(a+i);
cout<<The given array elements are :”;
for(i=0; i<n; i++)
cout<<”\t”<<*(a+i);
}
```

OUTPUT:
Enter the input for array 5
Enter array elements
1 2 3 4 5
The given array elements are :
1 2 3 4 5

مثال اكتب برنامج لحساب مجموع مصفوفة احادية البعد مصرح عنها باستخدام التوابع و يمرر عناصر المصفوفة احادية كمؤشر للتابع

For example, we can define a function to sum the terms of an array of int :

```
#include <iostream>
using namespace std;
int sum(int *x, int sz) {
int s = 0;
for(int k = 0; k < sz; k++) s = s + x[k];
return s;
}
int main() {
int cool[10] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
cout << sum(cool, 10) << endl;
}
```

مثال : اكتب برنامج لادخال عناصر مصفوفة أحادية البعد باستخدام المؤشرات من لوحة المفاتيح و يقوم بطباعة مجموع عناصر مصفوفة احادية البعد باستخدام التوابع و يمرر عناصر المصفوفة احادية كمؤشر للتابع طباعة مجموع هذه العناصر

```
#include <iostream>
using namespace std;
int sum(int *x, int sz) {
int s = 0;
for(int k = 0; k < sz; k++) s = s + x[k];
return s;
}
int main() {
        int size ;
        cout<<"enter size"<<endl;
        cin>> size;
        int cool[size];
        cout<<"cout enter of array elements"<<endl;
        for(int i=0; i<size; i++)
        cin>>*(cool+i);
        cout << "sum="<<sum(cool, size) << endl;
        }
```

مثال : عدل البرنامج السابق باستخدام تابع Void

```
#include <iostream>
using namespace std;
void sum(int *x, int sz) {
int s = 0;
for(int k = 0; k < sz; k++) s = s + x[k];
cout << "sum="<<s << endl;
}
int main() {
        int size ;
        cout<<"enter size"<<endl;
        cin>> size;
        int cool[size];

cout<<"cout enter of array elements"<<endl;
for(int i=0; i<size; i++)
cin>>*(cool+i);
sum(cool, size);

}
```

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

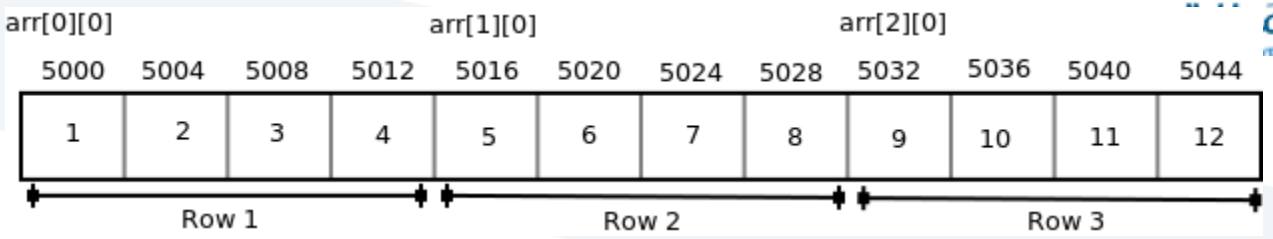
خرج البرنامج:

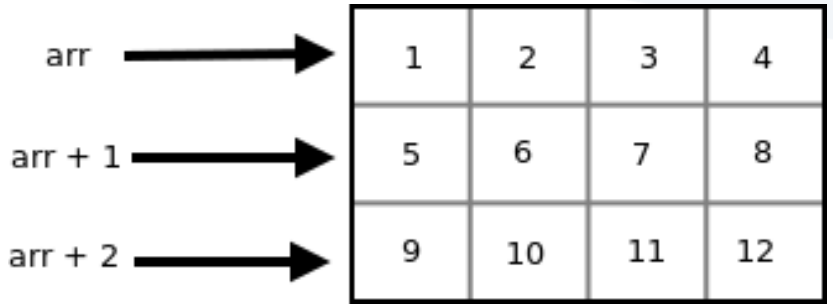
يتم إخراج قيم المصفوفة باستخدام أدلة النسق a[i] وفق قاعدة النسق
ثم يتم طباعة العناصر وفق المؤشر الذي يشير إلى بداية المصفوفة
مزاح بقيمة i وفق *(a+i)
يستخدم اسم المصفوفة وبعدها يتم استخدام المؤشر ليشيرإلى بداية
المصفوفة وكأنه اسم للمصفوفة كما في الحالة الأولى وفي الحالة
الأخيرة يتم استخدام اسم المؤشر وانزياح
بمقدار الدليل *(p+j) وفي الحالات الأربع يتم اخراج قيم المصفوفة.

```
a array is:
a[0]=5
a[1]=7
a[2]=9
a[3]=11
pointer/offset notation where
the pointer is the array name
*(a+0)=5
*(a+1)=7
*(a+2)=9
*(a+3)=11
```

```
pointer subscript notation
p[0]=5
p[1]=7
p[2]=9
p[3]=11
pointer/offset notation
*(p+0)=5
*(p+1)=7
*(p+2)=9
*(p+3)=11
```

المصفوفات متعددة الأبعاد
Pointers and two dimensional Arrays:



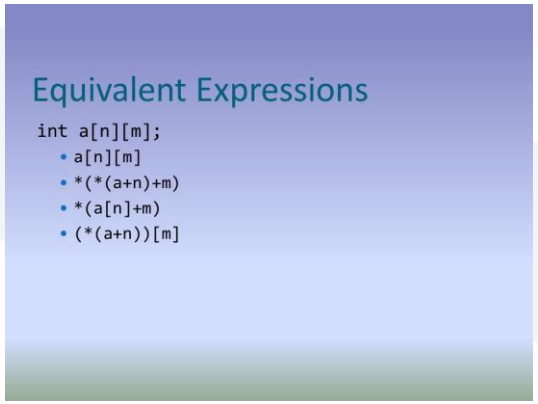


int arr[3][4] = { {1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12} };

	Col 1	Col 2	Col 3	Col 4
Row 1	1	2	3	4
Row 2	5	6	7	8
Row 3	9	10	11	12

arr - Points to 0th element of arr - Points to 0th 1-D array - 5000
arr + 1 - Points to 1th element of arr - Points to 1nd 1-D array - 5016
arr + 2 - Points to 2th element of arr - Points to 2nd 1-D array - 5032

نظرًا لأن الذاكرة في الكمبيوتر منظمة خطيًا ، فلا يمكن تخزين المصفوفة ثنائية الأبعاد في صفوف وأعمدة.
يعتبر مفهوم الصفوف والأعمدة نظريًا فقط ، في الواقع ، يتم تخزين المصفوفة ثنائية الأبعاد بترتيب الصف الرئيسي ، أي يتم وضع الصفوف بجوار بعضها البعض. يوضح الشكل التالي كيف سيتم تخزين المصفوفة ثنائية الأبعاد أعلاه في الذاكرة.



بفراض أن a مصفوفة ثنائية البعد ويمكن الحصول على عنوان عنصر a[i][j] من خلال التعليمة:

```
cout<<*(a)+(m*i)+j;
```

حيث أن m هو عدد أعمدة المصفوفة ثنائية البعد، من المهم التذكير أن عناصر المصفوفة ثنائية البعد تخزن في الذاكرة في أماكن متجاورة كما لو كانت عناصر تلك المصفوفة تنتمي لصف عدد عناصره هو n*m حيث أن n هو عدد الأسطر في المصفوفة ثنائية البعد.

يمكن أن نحصل على عنصر المصفوفة الثنائية البعد a[i][j] من خلال التعليمة:

```
cout<<*(*(a)+(m*i)+j);
```

نلاحظ من خرج البرنامج إنه بالفعل عناصر المصفوفة ثنائية البعد تخزن في الذاكرة كما لوكانت صفاً وحيد البعد عدد عناصره يساوي جداء بعديها.

مثال : اكتب برنامج بلغة c++ لقراة عناصر مصفوفة أعداد صحيحة a[2][3] وطباعة عناصرها استخدم المؤشرات لطباعة عناوين عناصر المصفوفة وقيم الممصفوفة

```
#include<iostream>
using namespace std;
const int n=2;
const int m=3;
int main()
{
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;}
}
```

*(a+i)+j

((a+i)+j)

Output

```
enter a [2][3]
2 3 4 5 6 7 8
a array is:
2 3 4
5 6 7
0x6ffdf0     2
0x6ffdf4     3
0x6ffdf8     4
0x6ffdfc     5
0x6ffe00     6
0x6ffe04     7
```