الفصل الخامس

المؤشرات pointers والإدارة الديناميكية للذاكرة dynamic memory management

هدف الفصل

1. التعرف إلى مفهوم المؤشر
2. التعامل مع المتحولات في الذاكرة من خلال عناوينها
3. فهم علاقة المصفوفات بالمؤشرات
4. الحجز الديناميكي للذاكرة
5. تحرير الذاكرة المحجوزة ديناميكياً

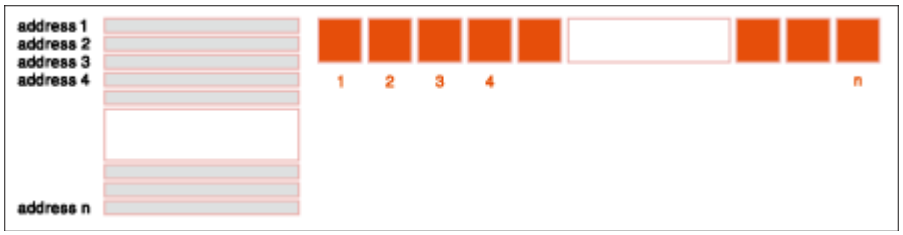
1- مقدمة

سنتناول في هذا الفصل أحد المواضيع الهامة في C++ وهو المؤشرات. وقد تطرقنا حتى الآن إلى كيفية التصريح عن متحولات من الأنماط البسيطة وعن أغراض من الـصفوف التـي يعرفهـا المبرمج دون أن نعير أي اهتمام لمواضع هذه المتحولات والأغراض فـي ذاكـرة الحاسـوب. سنركز اهتمامنا في هذا الفصل على عمليات حجز المتحولات في الذاكرة وكيفية التعامل معها.

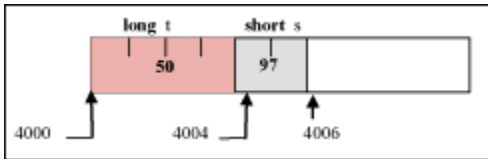
2- المؤشرات

لاستيعاب مفهوم المؤشر، لابد لنا من عرض كيفية التخزين الداخلي للمعطيات في الذاكرة. تتألف الذاكرة من تتالي خانات يتسع كل منها لثمانية بتّات، وتسمى كل خانة من هـذه الخانـات بايـت byte. يعرف كل بايت بالعنوان address الذي يوجد فيه في الذاكرة وبمحتواه من المعطيات.

يمكن التعامل مع الذاكرة وكأنها مصفوفة من البايتات. ويمكن رسمها عمودياً أو أفقياً كما يلي:



يشغل كل متحول أو غرض في C++ بايتاً أو أكثر حسب الحجم المحدد للنمط الذي ينتمي إليـه. فعلى سبيل المثال، يشغل متحول من النمط long أربعة بايتات، في حين يشغل متحول من الـنمط char بايتاً واحداً فقط. نسمي عنوان البايت الأول الذي يخزن فيه المتحول أو الغـرض عنـوان المتحول. فمثلاً لدينا متحول t من النمط long في العنوان 4000 ومتحول s من النمط short فـي العنوان 4004.



تمتلك لغة C++ نمطاً يسمى المؤشر pointer ، وتخزن متحولات هذا النمط عناوين المعطيات في الذاكرة. وقد سمي المؤشر مؤشراً لأنه "يشير إلى" العنصر. وحسبما ذكرنا فإن الذاكرة تحتـوي جميع أنواع المتحولات وهي بالتالي تحتوي على مواضع من النمط مؤشر، أي أن كل متحول من هذا النمط سيحتوي على عنوان الذاكرة التي يشير إليها وعلى عناوين لذواكر أخرى. (في مثالنـا سنجد في الذاكرة القيمة 4000 أو 4004).

3- التصريح عن متحول من النمط مؤشر

يجري التصريح عن مؤشر بوضع الرمز "*" قبل اسم المتحول في سطر تصريح اعتيادي.

مثال

```
long *ptr;              //pointer to a long integer
```

الشكل العام للتصريح عن متحول من نمط مؤشر:

```
Type *ptr;            //pointer to a variable from the type Type
```

أمثلة

```
int *intPtr;
char *charPtr;
```

4- إسناد القيم إلى المؤشرات

جرى تأمين النمط مؤشر في اللغة لكي نتمكن من خزن عناوين الذواكر في متحولات هذا النمط. غير أن التصريح عن متحول بالشكل التالي:

```
Type *ptr;
```

لايضع أية قيمة في المتحول ptr وبالتالي فإن هذا المتحول من نمط مؤشر لايشير إلى أية خانـة في الذاكرة. ولإعطاء قيمة لهذا المتحول، يجب أن نسند إليه عنوان ذاكرة موجودة. ولكن كيـف نحصل على عنوان ذاكرة موجودة؟

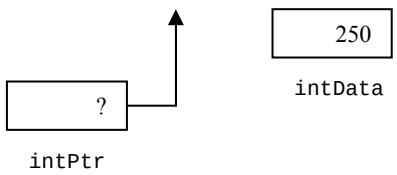
يوجد في C++ مؤثر يسمى مؤثر العنوان ورمزه "&" يعطي عند استخدامه مـع اسـم المتحـول عنوان هذا المتحول.

لنأخذ المثال التالي:

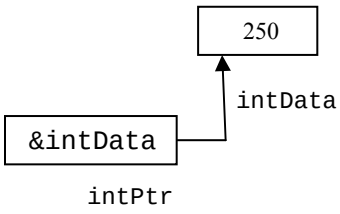
ليكن المتحول intData المصرح عنه كعدد صحيح بقيمة ابتدائية قدرها 250، ولـيكن المتحـول intPtr المصرح عنه كمؤشر على عدد صحيح. كما رأينا لايشير المؤشر عند تصريحه إلـى أي متحول، باستخدام المؤثر "&" نحصل على عنوان intData ، فنسند هذا العنوان إلى intPtr. بعـد تعليمة الإسناد هذه يكون لدينا المؤشر intPtr الذي يشير إلى المتحول intData.

```
int intData=250, *intPtr;  //pointer is uninitialized
```

بعد تنفيذ هذه التعليمة يكون الوضع كالتالي:



```
intPtr= &intData;  //&intData is the address of intData
```



تدريب:

ماعلاقة & بـ * ؟ نفذ البرنامج التالي وتحقق من العلاقة:

```
// Using the & and * operators.
#include <iostream>
using namespace std;

int main()
{
   int a; // a is an integer
   int *aPtr; // aPtr is an int * -- pointer to an integer

   a = 7; // assigned 7 to a
   aPtr = &a; // assign the address of a to aPtr

   cout << "The address of a is " << &a
        <<”\nThe value of aPtr is " << aPtr;
   cout << "\n\nThe value of a is " << a
        <<”\nThe value of *aPtr is " << *aPtr;
   cout << "\n\nShowing that * and & are inverses of"
        <<”each other.\n&*aPtr = " << &*aPtr
        << “\n*&aPtr = " << *&aPtr << endl;
   return 0;
}
```

4-1- النفاذ إلى المعطيات باستخدام المؤشرات

لاحظ هنا أنه يمكن الوصول إلى المعطيات المخزنة في المتحول intData من طريقين:

1. باستخدام اسم المتحول intData:

```
cout<< intData;
```

2. باستخدام المؤشر intPtr:

للنفاذ إلى المعطيات المخزنة في ذاكرة باستخدام عنوان هذه الذاكرة نستخدم المؤشر "*". إن وضع هذا المؤشر قبل اسم المؤشر في تعليمة إسناد أو قراءة أو إظهار أو أية صيغة حسابية أو منطقيـة يعني أننا نريد الحصول على محتوى الذاكرة التي يشير إليها المؤشر، وهكذا فـإن intPtr* هـو محتوى الذاكرة التي يشير إليها المؤشر intPtr:

```
cout<< *intPtr;  //output: 250
*intPtr=300;     //assign 300 to the memory pointed to by intPtr
cout<< intData;  //output: 300
```

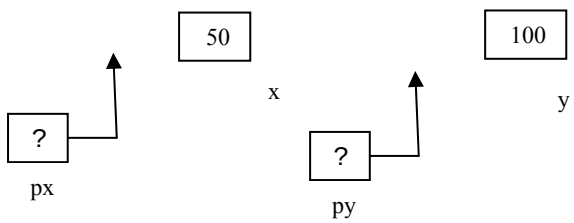
يسمى المؤشر * مؤشر النفاذ غير المباشر لانه يتيح لنا النفاذ إلى المتحول نفاذاً غير مباشر وإنمـا باستخدام المؤشر الذي يشير إليه.

تمارين:

ارسم الأشكال الموافقة لتنفيذ التعليمات التالية:

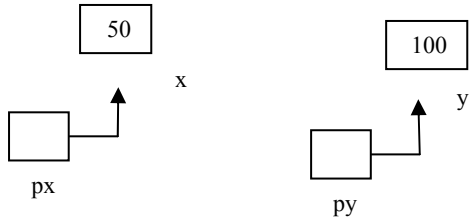
```
int x=50, y=100,*px, *py;
```

الحل:



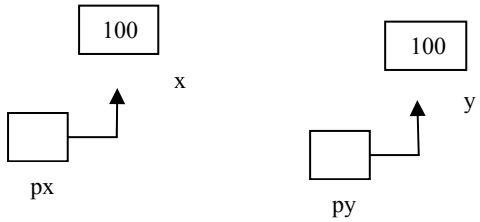
```
px=&x;
py=&y;
```

الحل:



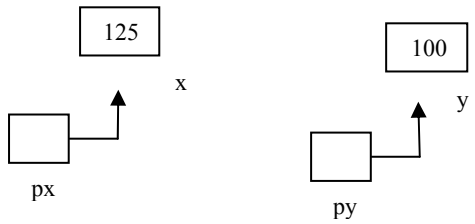

```
x=*py;
```

الحل:



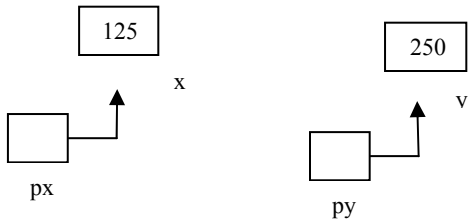

```
*px=y+25;
```

الحل:



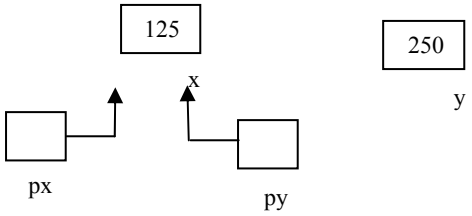
```
*py=*px*2;
```

الحل:




```
py=px;
```

الحل:




```
cout<<*px<<"  "<<*py;   // output : 125   125
cout<<*py<<"  "<<y;     // output : 125   250
```

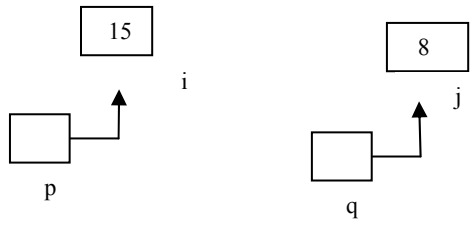
تقويم:

ليكن i وj متحولين صحيحين، و p وq مؤشرين على عددين صحيحين.

ارسم المخططات الموافقة للذواكر بعد تنفيذ كل سطر مما يلي:

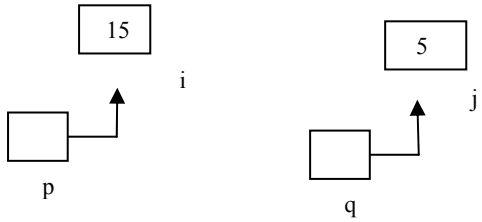
```
int i=15, j=8,*p=&i, *q=&j;
```

الحل:



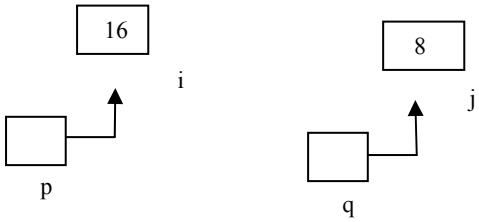

```
*q=5;
```

الحل:



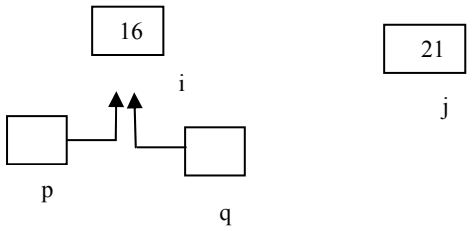

```
(*p)++;
*q+=3;
```

الحل:



```
q=p;
j=*q+5;
```

الحل:



5- المصفوفات والمؤشرات

سبق وتعرضنا إلى تعريف مفهوم المصفوفة وذكرنا أنها تعبر عن مجموعة من المعطيات التـي تشغل مواضع متجاورة ومتتالية في الذاكرة. وعند التصريح عن مصفوفة، يكون اسم المـصفوفة بمثابة عنوان العنصر الأول فيها. في C++ يعتبر اسم المصفوفة مؤشراً ثابتاً.

مثال

لو صرحنا عن المتحولين التاليين:

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

فإن arr يكون مؤشراً على العدد الصحيح الذي يقع في الموضع الأول في المصفوفة، وp مؤشر على متحول من نمط int. يقوم المترجم عند ترجمة التعليمات بحجز 10 مواضـع فـي الـذاكرة لأعداد صحيحة ويسند عنوان أولها إلى arr. يمكننا إذاً استخدام هذا العنوان لإعطاء قيمة ابتدائيـة لمتحول من نمط مؤشر على عدد صحيح.

يمكننا مثلاً كتابة:

```
p=arr;  //p points to arr[0]
```

5-1- الحسابات على العناوين

رأينا في الفقرات الخاصة بالمصفوفات أنه يمكن النفاذ إلى عنصر في المصفوفة باستخدام الـدليل index. يوجد طريقة أخرى تستخدم المؤشرات. لنفرض أن p مؤشر على مصفوفة عناصر مـن النمط T. إن الصيغة p+1 تمثل عنوان العنصر التالي في المصفوفة أي أن p[1]= *(p+1) . تعني إضافة 1 للعنوان إجراء انزياح بمقدار عنصر واحد إلى يمين p. عموماً يمكـن أن نكتـب: p+n للدلالة على عنوان العنصر الذي ترتيبه هو n+1 أي أن p[n] =*(p+n). بمعنـى لآخـر، يحـسب المترجم العنوان الموافق للصيغة p+n بإجراء انزياح معتمد على حجم معطيات النمط T بالبايتات.

```
(p+n) =p+ n*sizeof(T)
```

مثال

ليكن لدينا التصريحات التالية:

```
long arr[5]={200,-60,50,5,90};
long *p=arr;
```

لنفرض أن المصفوفة arr تبدأ عند العنوان 8500. يبين الجدول التالي عناوين العناصر الخمسة في المصفوفة. كما يتضمن الجدول القيم الناتجة عن حساب الصيغة p+n حيث 4≤ n ≤ 0 ، وقيم عناصر المصفوفة الموافقة p[n]=*(p+n)

العنوان		عنصر المصفوفة	
p	8500	*p	200
P+1	8504	*(p+1)	-60
P+2	8508	*(p+2)	50
P+3	8512	*(p+3)	5
P+4	8516	*(p+4)	90

مثال

تستخدم العمليات الحسابية على العنوان لتغيير قيمة المؤشر، فمثلاً يمكننا أن نكتب:

```
p=p+3;
p=p-2;
p++;
p--;
```

مثال

سنستخدم هنا مثالاً عن سلاسل المحارف في C++ (راجع فقرة سلاسل المحارف strings في C++ في فصل " البرامج الجزئية في C++")

```
// Converting lowercase letters to uppercase letters
// using a pointer to data.
#include <iostream>
#include <cctype> // prototypes for islower and toupper
using namespace std;

void convertToUppercase( char *);

int main()
{
   char phrase[] = "characters and $32.98”;

   cout << "The phrase before conversion is: " << phrase;
   convertToUppercase( phrase );
   cout << "\nThe phrase after conversion is:  " << phrase << endl;
   return 0;
}
// convert string to uppercase letters
void convertToUppercase( char *sPtr )
{
   while ( *sPtr != '\0' ) // loop while current character is not
'\0
 {
      if ( islower( *sPtr ) ) // if character is lowercase
      *sPtr = toupper( *sPtr ); // convert to uppercase

      sPtr++; // move sPtr to next character in string
   } // end while
}      //end function convertToUppercase
```

تقويم:

قارن الاستدعاء الوارد في المثال السابق:

```
convertToUppercase( phrase);
```

مع نموذج التابع:

```
void convertToUppercase(* char);
```

ماذا تستنتج؟

مثال

في المثال التالي نكتب خوارزمية فرز (ترتيب) لشعاع من الأعداد الصحيحة وهي selectionSort ثم نستدعيها في برنامج.

تعتمد هذه الخوارزمية على البحث عن أصغر عدد في الشعاع، وتبديل موضعه مع العدد الموجود في أول الشعاع، ثم نكرر العملية على ما تبقى من الشعاع بالبحث عن أصغر عنصر في الشعاع الناتج عن عملية التبديل، ثم وضعه في الدليل التالي وهكذا حتى بلوغ نهاية الشعاع. يبين البرنامج التالي التابع selectionSort واستدعاءه.

```
//This program puts values into an array, sorts the values into
// ascending order and prints the resulting array.
#include <iostream>
#include <iomanip>
using namespace std;

void selectionSort( int a[] ,  int ); // prototype
void swap( int & , int &  ); // prototype
int main()
{
   const int arraySize = 10;
   int a[ arraySize ] = {37, 6, 4, 8, 68, 12, 89, 10, 45, 2};
   cout << "Data items in original order\n";
   for ( int i = 0; i < arraySize; i++)
      cout << setw( 4 ) << a[ i ];
   cout<<endl;
   cout<<"begin sorting"<<endl;
   selectionSort( a, arraySize ); // sort the array
   cout << "\nData items in ascending order\n";
   for ( int j = 0; j < arraySize; j++)
      cout << setw( 4 ) << a[ j ];
   cout << endl;
   return 0;
}   //end main
```

```
//function to sort an array
void selectionSort( int   array[],  int size)
{
   int smallest; // index of smallest element

// loop over size - 1 elements
   for ( int i = 0; i < size - 1; i++)
{
      smallest = i; // first index of remaining array

      // loop to find index of smallest element
      for ( int index = i + 1; index < size; index++)
         if ( array[ index ] < array[ smallest])
            smallest = index;

      swap( array[ i ], array[ smallest]);

      for ( int j = 0; j < size; j++)
          cout << setw( 4 ) << array[ j];
      cout<<endl;

   }// end if
} // end function selectionSort
//swap values in element1 and element2
void swap( int  &element1, int &element2)
{
   int hold = element1;
   element1 = element2;
   element2 = hold;
} // end function swap
```

مثال

نريد الآن تعديل أنماط الموسطات الممررة للتابعين selectionSort و swap بحيث تصبحان كما يلي:

```
void selectionSort( int * ,  int);
void swap( int * , int *);
```

مع المحافظة على التصريح عن الشعاع كما في البرنامج السابق:

```
const int arraySize = 10;
int a[ arraySize ] = { 2, 6, 4, 8, 10, 12, 89, 68, 45, 37};
```

عدل البرنامج السابق ليتوافق مع المطلوب.

المثال المعدل

```
//This program puts values into an array, sorts the values into
// ascending order and prints the resulting array.
#include <iostream>
#include <iomanip>
using namespace std;

void selectionSort( int * ,  int ); // prototype
void swap( int * , int *  ); // prototype
int main()
{
   const int arraySize = 10;
   int a[ arraySize ] = { 37, 6, 4, 8, 68, 12, 89, 10, 45, 2};

   cout << "Data items in original order\n";

   for ( int i = 0; i < arraySize; i++)
      cout << setw( 4 ) << a[ i ];

   selectionSort( a, arraySize ); // sort the array

   cout << "\nData items in ascending order\n";

   for ( int j = 0; j < arraySize; j++)
      cout << setw( 4 ) << a[ j ];

   cout << endl;
   return 0; // indicates successful termination
}     //end main
// function to sort an array
void selectionSort( int *  array,  int size )
{
   int smallest; // index of smallest element
   // loop over size - 1 elements
   for ( int i = 0; i < size - 1; i++)
{
      smallest = i; // first index of remaining array
      // loop to find index of smallest element
      for ( int index = i + 1; index < size; index++)
         if ( array[ index ] < array[ smallest])
            smallest = index;
      swap( &array[ i ], &array[ smallest]);
      for ( int j = 0; j < size; j++)
          cout << setw( 4 ) << array[ j ];
      cout<<endl;
   } //end if
} //end function selectionSort
// swap values at memory locations to which
// element1Ptr and element2Ptr point
void swap( int *  element1Ptr, int *  element2Ptr )
```

```
{
   int hold = *element1Ptr;
   *element1Ptr = *element2Ptr;
   *element2Ptr = hold;
}  //end function swap
```

6- المؤشرات والصفوف

رأيت حتى الآن أنه يمكن التصريح عن متحولات من نمط مؤشر على أنماط بسيطة مثل int, long, char. في الحقيقة، لا يوجد ما يمنع التصريح عن مؤشر على غرض. فمثلاً لنفرض أننا قد عرفنا الصف Time كما في الفصل الرابع. (راجع الفصل الرابع فقرة التصريح الكامل عن الصف Timeوتوابعه) يمكننا أن نعرف مايلي:

```
Time morning(7,0,0);
Time *timePtr;
timePtr=&morning;
```

ولاستدعاء التوابع الأعضاء التابعة للغرض يمكننا أن نكتب:

```
(*timePtr).setTime(6,30,0);
```

وهو يكافئ morning.setTime(6,30,0);.

يجب الانتباه هنا إلى ضرورة استخدام الأقواس لأن المؤثر * له أولوية أضعف من المؤثر . ، وبدون القوسين يصبح لدينا:

```
*timePtr.setTime(6,30,0);
```

الذي يفهمه المترجم كما يلي:

```
*(timePtr.setTime(6,30,0));
```

ولتفادي الوقوع في خطأ من هذا النوع، وتبسيط شكل الاستدعاء، زودت لغة C++ بمؤثر هو -> يسمح للنفاذ إلى أعضاء الصف، وعليه يصبح استدعاء التابع setTime في الغرض المشار إليه بـ timePtr على النحو التالي:

```
timePtr->setTime(6,30,0);
```

7- الذاكرة الديناميكية

كثيراً ما نجد في التطبيقات أن الاحتياجات المتعلقة بحجز الذواكر للبرنامج لا تتحدد إلا عند التنفيذ (run time)، أي أنه يجري حجز الذاكرة عند الطلب بدلاً من حجزها عند الترجمة (compile time).

كمثال على ما سبق، لنفرض أننا بصدد بناء تطبيق لمعالجة علامات الطلاب في جامعة ما. إذا كان عدد الطلاب كبيراً (500 طالب مثلاً)، يجب أن نستخدم مصفوفة كبيرة لخزن علامات الطلاب، في حين أنه في حالة صف فيه 10 طلاب مثلاً لاحاجة لنا بحجز كامل الذاكرة السابقة (500) في حين أننا لن نستخدم منها سوى 10. وهنا يتضح لنا أن تثبيت الحجم المطلوب عند الترجمة قد يتسبب بهدر كبير في موارد الذاكرة، ويجب أن يتوفر إمكان حجز حجم الذاكرة اللازم ديناميكياً اعتماداً على عدد الطلاب في الصف. يسمى هذا المفهوم "الحجز الديناميكي للذاكرة" "dynamic memory allocation" ، وهو مفهوم يقع ضمن مجال "الإدارة الديناميكية للذاكرة" "dynamic memory management".

سنبدأ بعرض مؤثرين جديدين في C++ هما new و delete اللذان يسمحان بحجز الذاكرة وتحريرها ديناميكياً.

7-1- المؤثر new لحجز الذاكرة ديناميكياً

عند تنفيذ أي برنامج، يقوم نظام التشغيل بتهيئة جزء خاص من الذاكرة يستخدمه البرنامج عند حجز الذواكر التي يتعامل معها في البرنامج. يسمى هذا الجزء من الذاكرة "الكومة" "heap". ولكي تستطيع النفاذ إلى هذه الكومة وحجز ذاكرة ما فيها، يجب أن تستخدم المؤثر new مع المؤشرات. يحتاج المؤثر new إلى نمط (يستخدمه كمحدد أي وسيط) وعندها يحجز موضعاً في الكومة يتسع لعنصر من النمط المرفق، ويعيد عنوان الذاكرة المحجوزة الذي نستطيع فيما بعد إسناده إلى مؤشر.

لنأخذ كمثال عملية الحجز الديناميكي لذاكرتين أحدهما من النمط int والأخرى من النمط long. نبدأ بالتصريح عن مؤشر لهذين النمطين:

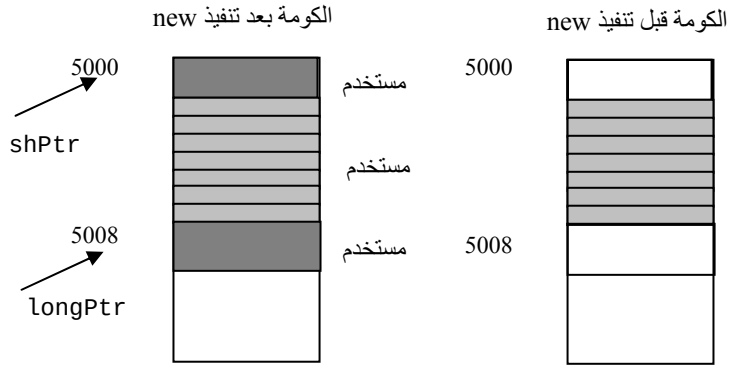
```
int *shPtr;      //assume the size of int is 2 bytes
long *longPtr;   //assume the size of long is 4 bytes
```

يطلب المؤثر new الحصول على ذاكرة من الكومة. إن توفرت الذاكرة، يجري حجز عـدد مـن البايتات يوافق الحجم اللازم للنمط، ويعيد عنوان أول بايت يبدأ عنده الحجز. إن لم تتوفر الذاكرة المطلوبة من الكومة يعيد المؤثر new القيمة 0. يطلق على هذه القيمة في معرض حـديثنا عـن المؤشرات اسم المؤشر NULL أي المؤشر الصفري ويقصد بذلك أنه لا يشير إلى أي موضع في الذاكرة. إن الثابت NULL الذي يساوي 0 معرف في مكتبة iostream ويمكـن أن نـضعه قيمـة ابتدائية لأي مؤشر من أي نمط. يعتبر NULL العدد الصحيح الوحيد الذي يـسند إلـى مؤشـر، ويستخدم في صيغ المقارنة.

وفيما يلي التعليمتان اللتان تحجزان الذاكرتين المذكورتين:

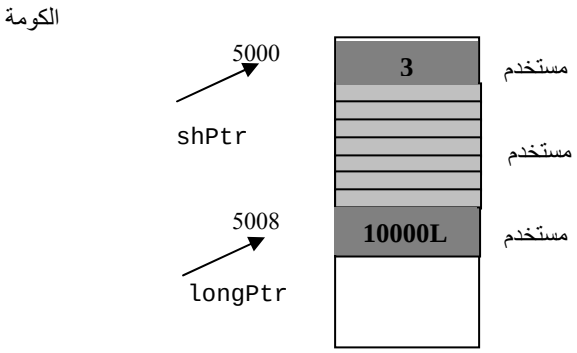
```
shPtr=new int;    //a pointer to an integer
longPtr=new long;    //a pointer to a long
```

نرى في الشكل التالي عملية حجز الذاكرتين shPtr و longPtr. لنفرض أن العناوين في الكومـة تبدأ عند العنوان 5000. يطلب المؤثر new حجز ذاكرة من النمط int أي بحجم قـدره 2 بايـت فتحجز عند الموضع 5000، ويعاد العنوان 5000 ليخزن في المتحول shPtr. عنـد ورود طلـب الحجز التالي لذاكرة من النمط long أي بحجم يعادل 4 بايتات يجري الحجز في موضـع آخـر يحوي ذواكر لم يجر استخدامها بعد. يمتلك نظام التشغيل لائحة بالذواكر المستخدمة، ومـن هنـا يمكنه تحديد الذواكر المستخدمة من تلك التي لم تستخدم. بفـرض أن الـذاكرتين اللتـان تبـدآن بالعناوين 5002 و 5007 كانتا مستخدمتين، عندها يحجز النظام الذاكرة المطلوبـة اعتبـاراً مـن العنوان 5008، ويعيد هذا العنوان الذي يخزن في الذاكرة longPtr.



لاتعطى الذواكر في الكومة قيماً افتراضية. وفي حال رغبة المبرمج إعطاء قيمة محددة ابتدائيـة، عليه أن يمررها بعد اسم النمط بين قوسين صغيرين( ). فمثلاً لإعطاء القيمتين 3 و 10000L على الترتيب للذاكرتين المحجوزتين بالتعليمتين السابقتين يمكننا أن نكتب:

```
shPtr=new int(3);
longPtr=new long(10000L);
```



يمكننا الآن التعامل مع هاتين الذاكرتين عبر المؤشر الذي يشير إلى كل منهما، ونـستخدم لهـذا الغرض المعامل *.

```
*shPtr=2;     //*shPtr becomes 6
cout<<*longPtr;    //output:10000
```

مثال

لحجز ذاكرتين من نمط string و double نكتب مايلي:

```
double *d=new double;
string *str=new string(”heap”);
//test if memory  is allocated
if (str==NULL)
{cerr<<”memory allocation failure”<<endl;
exit(1);
}
```

7-1-1- الحجز الديناميكي لمصفوفة

تظهر قوة الحجز الديناميكي عندما نرغب بحجز مصفوفات لا تتحدد أطوالها إلا لحظـة التنفيـذ. نستخدم لهذا الغرض المؤثر new نفسه لكن نستخدم بعده المؤثر [ ] مع تمرير الطول بين هذين القوسين المتوسطين. فمثلاً لحجز مصفوفة طولها 50 نكتب بما يلي:

```
const int ARRSIZE=50;
int *arr;
arr=new int[ARRSIZE];    //dynamically allocate the array
```

تحجز كتلة من الذاكرة بطول ARRSIZE ويوضع عنوان بداية الكتلة في المتحول arr. يمكن الآن التعامل مع المصفوفة باستخدام الدليل كالعادة، حيث arr[0] هو العنصر الأول و arr[ARRSIZE-1] هو العنصر الأخير فيها.

7-1-2- الحجز الديناميكي لأغراض من صفوف

يمكن استخدام المؤثر new لحجز ذواكر لأغراض من صفوف أنشأها المبرمج. يجري اسـتدعاء الباني عندما يحجز ديناميكياً ذاكرة لغرض وحيد من هذا الصف. وإن لم يكن هناك بان افتراضي للصف، يجب عندها تمرير المحددات.

مثال:

```
Time *midNight,*noon;
midNight =new Time;
noon=new Time(12,0,0);
```

لحجز مصفوفة من الأغراض ديناميكياً، يجب أن يكون الصف مزوداً ببان افتراضـي، وعنـدها يحجز المؤثر new مصفوفة جميع عناصرها تأخذ القيمة الابتدائية التي يعطيها الباني الافتراضي.

مثال

```
Time *t;
t=new Time[100];
```

2-7- المؤشر delete لتحرير الذاكرة ديناميكياً

تقع مسؤولية الإدارة الفعّالة للذاكرة في التطبيقات على عاتق المبرمج. يجب أن يدرك المبرمج أن الكومة هي مصدر قابل للنفاد، وعليه إن عملية حجز الذواكر بالمؤشر new يجب أن يعقبها تحرير للذواكر المحجوزة عندما تنتهي الحاجة إليها. تقدم C++ مؤشراً هو delete مهمته تحرير الـذاكرة التي سبق حجزها باستخدام new. يعتمد استخدام delete على أن النظام يحتفظ بلائحـة تتـضمن عناوين الذواكر التي حجزت باستخدام المؤشر new، والحجوم المحجوزة لها. يقوم المؤشر delete بإبطال مفعول new وذلك بتحرير الذاكرة الموافقة.

مثال

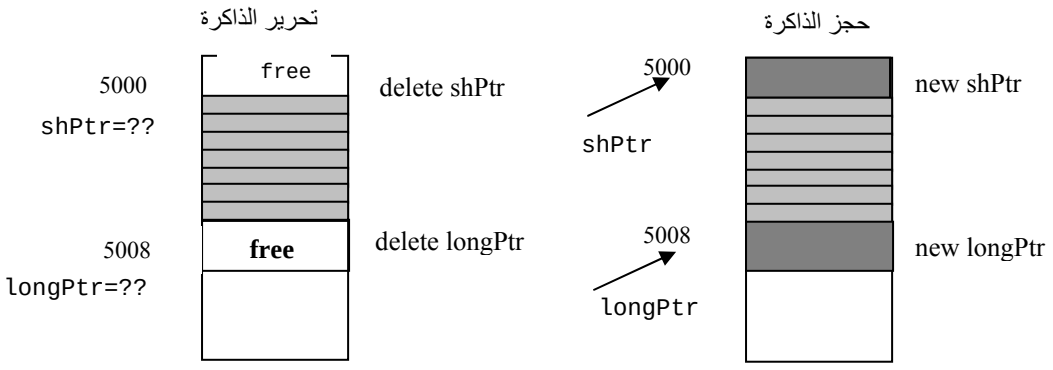
لنعد إلى مثالنا الذي قمنا فيه بحجز ذاكرتين ديناميكياً على النحو التالي:

```
int *shPtr=new int;
long *longPtr=new long;
```

لتحرير الذاكرتين المشار إليهما بـ shPtr و longPtr نستخدم delete يليها اسم المؤشر:

```
delete shPtr;   //deallocates 2 bytes starting at 5000
delete longPtr;    //deallocates 4 bytes starting at 5008
```

يوضح الشكل التالي وضع الكومة بعد عمليات new و delete.



لتحرير ذاكرة حجزت لمصفوفة ديناميكياً، نستعمل المؤثر delete متبوعاً بالقوسين المتوسطتين: [ ] .

مثال

```
int *arr=new int[ARRSIZE];  //allocate the array arr
delete [] arr;              //deallocate the array memory
```

مثال

```
  int *p;
  long *q;

  p=new int(5);
  q=new long[20];

  delete p;
  delete [] q;
```