Pointers المؤشرات

• **A pointer** is a variable which contains the address in memory of another variable. We can have a pointer to any variable type.
– The operator **&** gives the "address of a variable".
– The **indirection** or **dereference operator *** gives the "contents of an object pointed to by a pointer".

Address-of operator (&)

The address of a variable can be obtained by preceding the name of a variable with an ampersand sign (`&`), known as *address-of operator*. For example:

```
foo = &myvar;
```

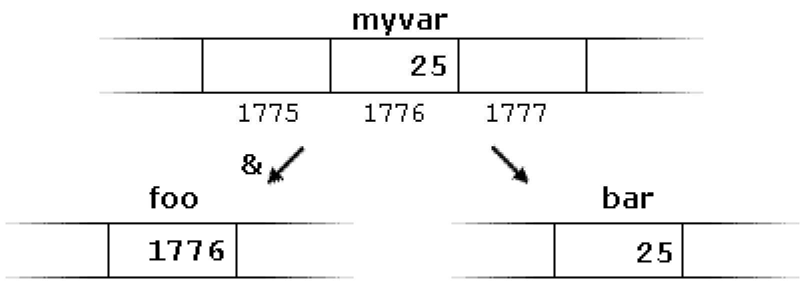
This would assign the address of variable `myvar` to `foo`; by preceding the name of the variable `myvar` with the *address-of operator* (`&`), we are no longer assigning the content of the variable itself to `foo`, but its address.

The actual address of a variable in memory cannot be known before runtime, but let's assume, in order to help clarify some concepts, that `myvar` is placed during runtime in the memory address `1776`.

In this case, consider the following code fragment:

```
myvar = 25;
foo = &myvar;
bar = myvar;
```

The values contained in each variable after the execution of this are shown in the following diagram:



• First, we have assigned the value `25` to `myvar` (a variable whose address in memory we assumed to be `1776`).
• The second statement assigns `foo` the address of `myvar`, which we have assumed to be `1776`.
• Finally, the third statement, assigns the value contained in `myvar` to `bar`. This is a standard assignment operation.

The main difference between the second and third statements is the appearance of the *address-of operator* (`&`).

The variable that stores the address of another variable (like `foo` in the previous example) is what in C++ is called a *pointer*. Pointers are a very powerful feature of the language that has many uses in lower level programming.

Dereference operator (*)

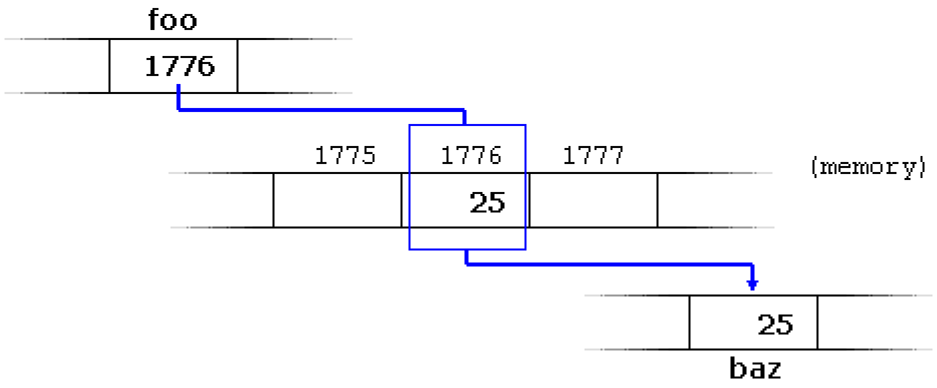
As just seen, a variable which stores the address of another variable is called a *pointer*. Pointers are said to "point to" the variable whose address they store.

pointers they can be used to access the variable they point to directly. This is done by preceding the pointer name with the *dereference operator* (`*`). The operator itself can be read as "value pointed to by".

Therefore, following with the values of the previous example, the following statement:

```
baz = *foo;
```

This could be read as: "`baz` equal to value pointed to by `foo`", and the statement would actually assign the value `25` to `baz`, since `foo` is `1776`, and the value pointed to by `1776` (following the example above) would be `25`.



It is important to clearly differentiate that `foo` refers to the value `1776`, while `*foo` (with an asterisk `*` preceding the identifier) refers to the value stored at address `1776`, which in this case is `25`. Notice the difference of including or not including the *dereference operator* (I have added an explanatory comment of how each of these two expressions could be read):

```
baz = foo;    // baz equal to foo (1776)
baz = *foo;   // baz equal to value pointed to by foo (25)
```

The reference and dereference operators are thus complementary:

- `&` is the *address-of operator*, and can be read simply as "address of"
- `*` is the *dereference operator*, and can be read as "value pointed to by"

Thus, they have sort of opposite meanings: An address obtained with `&` can be dereferenced with `*`.

Earlier, we performed the following two assignment operations:

```
myvar = 25;
foo = &myvar;
```

Right after these two statements, all of the following expressions would give true as result:

```
myvar == 25
&myvar == 1776
foo == 1776
*foo == 25
```

الصيغة العامة للإعلان عن المؤشرات :

data type *pointer name ;

حيث:-

data type هي نوع البيانات التي يشير إليها المؤشر

pointer name هو اسم المؤشر.

أمثلة:

1. int * p;	إعلان عن مؤشر يشير إلى متغير من النوع الصحيح
2 . int i,j,a[10],b[2],*p,*q;	إعلان عن مجموعة من المتغيرات والمصفوفات والمؤشرات من النوع الصحيح.
3. float *q	إعلان عن مؤشر يشير إلى متغيرً من النوع الحقيقي
4. char *r;	إعلان عن مؤشر يشير إلى متغيرً من النوع الحرفي
5. float *m, *n	إعلان عن مؤشرين باسم m,n ليؤشرا على قيمتنٌ حقيقيتنٌ .

In order to demonstrate that a pointer may point to different variables during its lifetime in a program, the example repeats the process with second value and that same pointer, mypointer.

```
// my first pointer
#include <iostream>
using namespace std;
int main ()
{
  int firstvalue, secondvalue;
  int * mypointer;

  mypointer = &firstvalue;
  *mypointer = 10;
  mypointer = &secondvalue;
  *mypointer = 20;
  cout << "firstvalue is " <<firstvalue << '\n';

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

Output:

```
firstvalue is 10
secondvalue is 20
```

Here is an example a little bit more elaborated:

```
// more pointers
#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 p1 = 10
 *p2 = *p1;         // value pointed to p2=value
                       pointed to  p1
  p1 = p2;          // p1 = p2
                    (value of pointer is copied)
 *p1 = 20;         // value pointed to by p1= 20

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

Output:

```
firstvalue is 10
secondvalue is 20
```

Each assignment operation includes a comment on how each line could be read: i.e., replacing ampersands (&) by "address of", and asterisks (*) by "value pointed to by".

Pointers and arrays

The concept of arrays is related to that of pointers. In fact, arrays work very much like pointers to their first elements, and, actually, an array can always be implicitly converted to the pointer of the proper type. For example, consider these two declarations:

تستطيع المؤشرات الإشارة إلى المتغيرات المفردة كما وتشير إلى عناصر المصفوفة array

لنفترض أن a,p عرفت كما يلي:

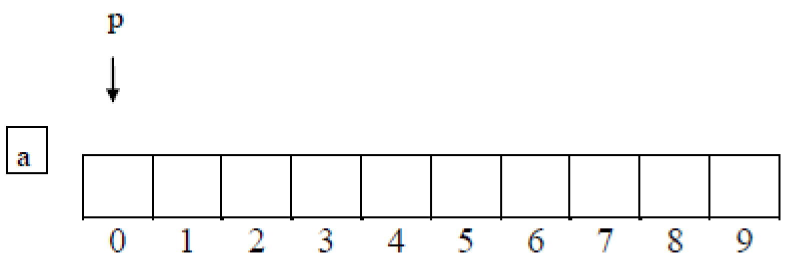
```
int a[10],*p;
```

فبهذا يمكن استخدام المؤشر p للإشارة إلى العنصر الأول من المصفوفة a (a[0])

كما يلي:

p=&a[0]; وهي مكافئة للعبارة p=a;

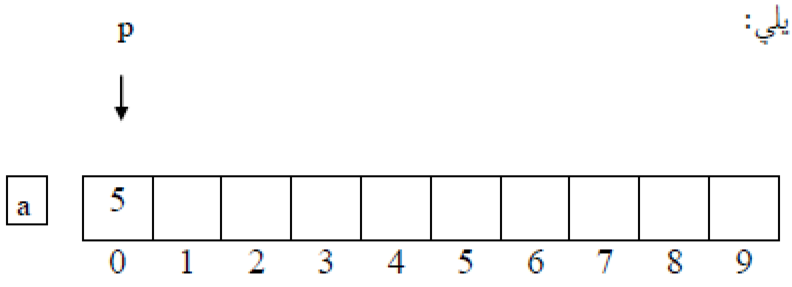
أي



وبهذا يمكن الوصول إلى a[0] عن طريق p وبإمكاننا تخزين القيمة 5 على سبيل المثال كعنصر في الموقع الأول من المصفوفة كما يلي:

```
*p=5;
```

فيصبح الشكل كما يلي:



يمكن تنفيذ ثلاث عمليات حسابية أساسية على المؤشر وهي:

1. زيادة رقم صحيح إلى المؤشر.
2. طرح رقم صحيح من المؤشر.
3. طرح مؤشر من مؤشر.

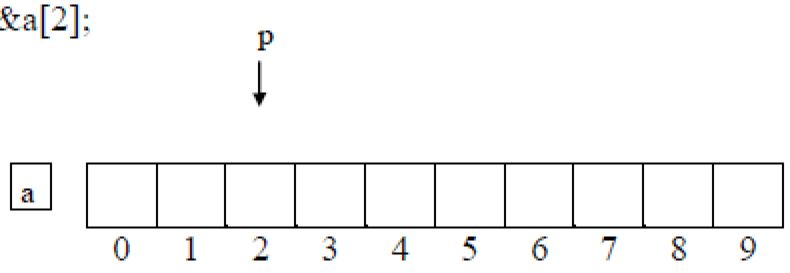
لنفترض التعريف التالي:

```
int a[10],*p,*q;
```

فإذا كان المؤشر p يشير إلى a[i] فإن زيادة j إلى المؤشر تعني أن المؤشر سوف يشير إلى العنصر a[i+j]

ويمكن توضيح العملية كما يلي:

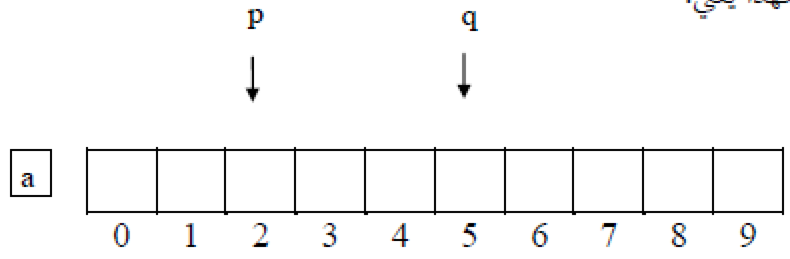
```
p=&a[2];
```



إذا جعلنا

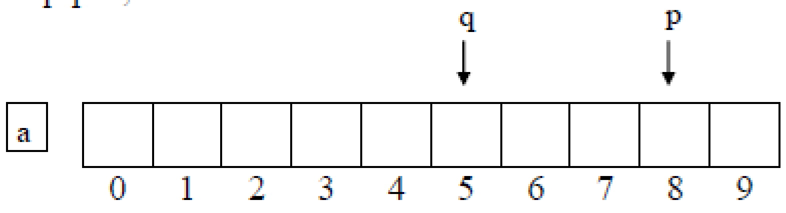
```
q=p+3;
```

فهذا يعني:



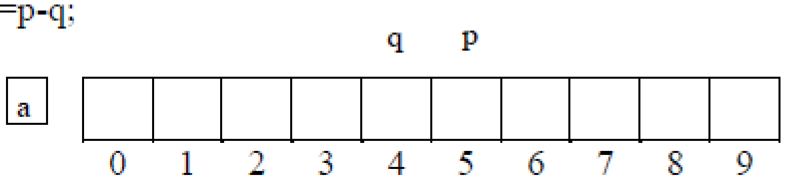
أما عملية الطرح من المؤشر فيمكن توضيحها بالشكل التالي:

```
p=&a[8];
q=p-3;
```



كما يمكن طرح مؤشر من مؤشر آخر كما يلي:

```
p=&a[5];
q=&a[1];
q=p-q;
```



Let's see an example that mixes arrays and pointers:

```
// 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,
```

EX: By using pointers read 10 elements of array used to store data of integer type then calculate and print the sum of its elements :

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

int main()
{
int a[10],sum,*p,i;

sum=0;
cout<<"enter the values of array:";
for(p=&a[0];p<&a[10];p++)
   cin>> *p;

for(p=&a[0];p<&a[10];p++)
  sum+=*p;
cout<<"sum="<<sum<<endl;
return 0;
}
```

المثال التالي يوضح استخدام المؤشرات مع الخيوط الرمزية :

EX: Write a program in c++ to reverse and print the string:

```
#include<iostream>

#include<string>

using namespace std;
void   reverse(char  *s ,int  k)  {

  char   *first, *last, temp;

  first=&s[0];

  last=&s[k-1];

  for(int  i=0;i<(k\2);i++)

   {   temp= *first;

       *first=*last;

       *last=temp;

       first++;

       last--;

   }
```

```
int main () {
   int  n;
   char  str[ ]=”hello”;
   cout<<str<<endl;
   reverse(str,n);
   cout<<str<<endl;
   return 0;
}
```

H.W:

Trace the following program, and try to find the output :

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

int main()
{
  Int  i =5;
  int  *p,*q;

  p=&i;
 *p = i * *p;
 q = p;
 *q =*q+*p;
 cout<<" i= "<<i<<" \n";
 cout<<" *p= "<<*p<<" \n";
 cout<<" *q= "<<*q<<" \n";
 return 0;
}
```