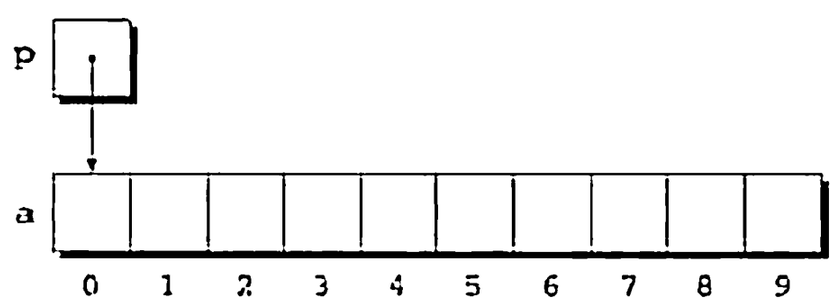
```
int a[10], *p;
```

We can make p point to a[0] by writing

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

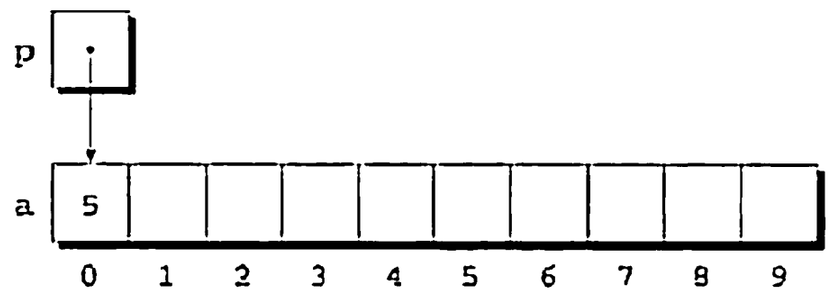
Graphically, here's what we've just done:



We can now access a[0] through p; for example, we can store the value 5 in a[0] by writing

```
*p = 5;
```

Here's our picture now:



Making a pointer p point to an element of an array a isn't particularly exciting. However, by performing ***pointer arithmetic*** (or ***address arithmetic***) on p, we can access the other elements of a. C supports three (and only three) forms of pointer arithmetic:

Adding an integer to a pointer
Subtracting an integer from a pointer
Subtracting one pointer from another

Let's take a close look at each of these operations. Our examples assume that the following declarations are in effect:

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

Adding an Integer to a Pointer

Adding an integer j to a pointer p yields a pointer to the element j places after the one that p points to. More precisely, if p points to the array element a[i], then p + j points to a[i+j] (provided, of course, that a[i+j] exists).

Q&A

The following example illustrates pointer addition; diagrams show the values of p and q at various points in the computation.