ship between arrays and pointers in C. For example, we could use the following loop to initialize the array that a points to:

```
for (i = 0; i < n; i++)
  a[i] = 0;
```

We also have the option of using pointer arithmetic instead of subscripting to access the elements of the array.

The calloc Function

Although the malloc function can be used to allocate memory for an array, C provides an alternative—the calloc function—that's sometimes better. calloc has the following prototype in <stdlib.h>:

```
void *calloc(size_t nmemb, size_t size);
```

calloc allocates space for an array with nmemb elements, each of which is size bytes long; it returns a null pointer if the requested space isn't available. After allocating the memory, calloc initializes it by setting all bits to 0. For example, the following call of calloc allocates space for an array of n integers, which are all guaranteed to be zero initially:

Q&A

```
a = calloc(n, sizeof(int));
```

Since calloc clears the memory that it allocates but malloc doesn't, we may occasionally want to use calloc to allocate space for an object other than an array. By calling calloc with 1 as its first argument, we can allocate space for a data item of any type:

```
struct point { int x, y; } *p;

p = calloc(1, sizeof(struct point));
```

After this statement has been executed, p will point to a structure whose x and y members have been set to zero.

The realloc Function

Once we've allocated memory for an array, we may later find that it's too large or too small. The realloc function can resize the array to better suit our needs. The following prototype for realloc appears in <stdlib.h>:

```
void *realloc(void *ptr, size_t size);
```

When realloc is called, ptr must point to a memory block obtained by a previous call of malloc, calloc, or realloc. The size parameter represents the new size of the block, which may be larger or smaller than the original size. Although realloc doesn't require that ptr point to memory that's being used as an array, in practice it usually does.