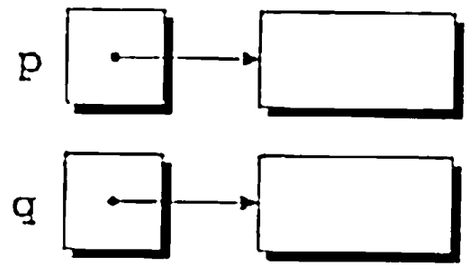
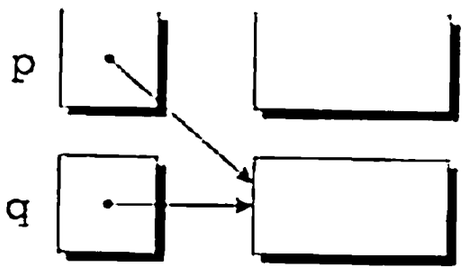
After the first two statements have been executed, p points to one memory block, while q points to another:



After q is assigned to p, both variables now point to the second memory block:



There are no pointers to the first block (shaded), so we'll never be able to use it again.

A block of memory that's no longer accessible to a program is said to be ***garbage***. A program that leaves garbage behind has a ***memory leak***. Some languages provide a ***garbage collector*** that automatically locates and recycles garbage, but C doesn't. Instead, each C program is responsible for recycling its own garbage by calling the `free` function to release unneeded memory.

The `free` Function

The `free` function has the following prototype in `<stdlib.h>`:

```
void free(void *ptr);
```

Using `free` is easy; we simply pass it a pointer to a memory block that we no longer need:

```
p = malloc(...);
q = malloc(...);
free(p);
p = q;
```

Calling `free` releases the block of memory that `p` points to. This block is now available for reuse in subsequent calls of `malloc` or other memory allocation functions.

The argument to `free` must be a pointer that was previously returned by a memory allocation function. (The argument may also be a null pointer, in which case the call of `free` has no effect.) Passing `free` a pointer to any other object (such as a variable or array element) causes undefined behavior.