### The "Dangling Pointer" Problem

Although the `free` function allows us to reclaim memory that's no longer needed, using it leads to a new problem: ***dangling pointers.*** The call `free(p)` deallocates the memory block that `p` points to, but doesn't change `p` itself. If we forget that `p` no longer points to a valid memory block, chaos may ensue:

```
char *p = malloc(4);
...
free(p);
...
strcpy(p, "abc");    /*** WRONG ***/
```

Modifying the memory that `p` points to is a serious error, since our program no longer has control of that memory.

---

Attempting to access or modify a deallocated memory block causes undefined behavior. Trying to modify a deallocated memory block is likely to have disastrous consequences that may include a program crash.

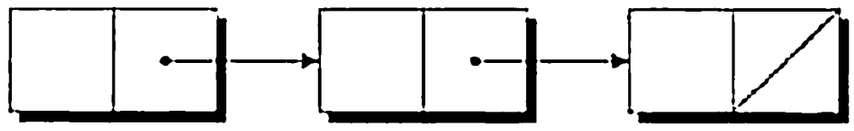
---

Dangling pointers can be hard to spot, since several pointers may point to the same block of memory. When the block is freed, all the pointers are left dangling.

## 17.5 Linked Lists

Dynamic storage allocation is especially useful for building lists, trees, graphs, and other linked data structures. We'll look at linked lists in this section; a discussion of other linked data structures is beyond the scope of this book. For more information, consult a book such as Robert Sedgewick's *Algorithms in C, Parts 1–4: Fundamentals, Data Structures, Sorting, Searching*, Third Edition (Reading, Mass.: Addison-Wesley, 1998).

A ***linked list*** consists of a chain of structures (called ***nodes***), with each node containing a pointer to the next node in the chain:



The last node in the list contains a null pointer, shown here as a diagonal line.

In previous chapters, we've used an array whenever we've needed to store a collection of data items; linked lists give us an alternative. A linked list is more flexible than an array: we can easily insert and delete nodes in a linked list, allowing the list to grow and shrink as needed. On the other hand, we lose the "random access" capability of an array. Any element of an array can be accessed in the same