Functions such as `concat` that dynamically allocate storage must be used with care. When the string that `concat` returns is no longer needed, we'll want to call the `free` function to release the space that the string occupies. If we don't, the program may eventually run out of memory.

`free` function ➤ 17.4

Arrays of Dynamically Allocated Strings

In Section 13.7, we tackled the problem of storing strings in an array. We found that storing strings as rows in a two-dimensional array of characters can waste space, so we tried setting up an array of pointers to string literals. The techniques of Section 13.7 work just as well if the elements of an array are pointers to dynamically allocated strings. To illustrate this point, let's rewrite the `remind.c` program of Section 13.5, which prints a one-month list of daily reminders.

PROGRAM

Printing a One-Month Reminder List (Revisited)

The original `remind.c` program stores the reminder strings in a two-dimensional array of characters, with each row of the array containing one string. After the program reads a day and its associated reminder, it searches the array to determine where the day belongs, using `strcmp` to do comparisons. It then uses `strcpy` to move all strings below that point down one position. Finally, the program copies the day into the array and calls `strcat` to append the reminder to the day.

In the new program (`remind2.c`), the array will be one-dimensional; its elements will be pointers to dynamically allocated strings. Switching to dynamically allocated strings in this program will have two primary advantages. First, we can use space more efficiently by allocating the exact number of characters needed to store a reminder, rather than storing the reminder in a fixed number of characters as the original program does. Second, we won't need to call `strcpy` to move existing reminder strings in order to make room for a new reminder. Instead, we'll merely move *pointers* to strings.

Here's the new program, with changes in **bold.** Switching from a two-dimensional array to an array of pointers turns out to be remarkably easy: we'll only need to change eight lines of the program.

remind2.c

```
/* Prints a one-month reminder list (dynamic string version) */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_REMIND 50   /* maximum number of reminders */
#define MSG_LEN 60      /* max length of reminder message */

int read_line(char str[], int n);
```