17.3 Dynamically Allocated Arrays

Dynamically allocated arrays have the same advantages as dynamically allocated strings (not surprisingly, since strings *are* arrays). When we're writing a program, it's often difficult to estimate the proper size for an array; it would be more convenient to wait until the program is run to decide how large the array should be. C solves this problem by allowing a program to allocate space for an array during execution, then access the array through a pointer to its first element. The close relationship between arrays and pointers, which we explored in Chapter 12, makes a dynamically allocated array just as easy to use as an ordinary array.

Although `malloc` can allocate space for an array, the `calloc` function is sometimes used instead, since it initializes the memory that it allocates. The `realloc` function allows us to make an array "grow" or "shrink" as needed.

Using `malloc` to Allocate Storage for an Array

We can use `malloc` to allocate space for an array in much the same way we used it to allocate space for a string. The primary difference is that the elements of an arbitrary array won't necessarily be one byte long, as they are in a string. As a result, we'll need to use the `sizeof` operator to calculate the amount of space required for each element.

sizeof operator ➤7.6

Suppose we're writing a program that needs an array of `n` integers, where `n` is to be computed during the execution of the program. We'll first declare a pointer variable:

```
int *a;
```

Once the value of `n` is known, we'll have the program call `malloc` to allocate space for the array:

```
a = malloc(n * sizeof(int));
```

Always use `sizeof` when calculating how much space is needed for an array. Failing to allocate enough memory can have severe consequences. Consider the following attempt to allocate space for an array of `n` integers:

```
a = malloc(n * 2);
```

If `int` values are larger than two bytes (as they are on most computers), `malloc` won't allocate a large enough block of memory. When we later try to access elements of the array, the program may crash or behave erratically.

Once it points to a dynamically allocated block of memory, we can ignore the fact that `a` is a pointer and use it instead as an array name, thanks to the relation-