```
int i, *p;
...
p = &i;
scanf("%d", p);
```

Since `p` contains the address of `i`, `scanf` will read an integer and store it in `i`. Using the & operator in the call would be wrong:

```
scanf("%d", &p);    /*** WRONG ***/
```

`scanf` would read an integer and store it in `p` instead of in `i`.

Failing to pass a pointer to a function when one is expected can have disastrous results. Suppose that we call `decompose` without the & operator in front of `i` and `d`:

```
decompose(3.14159, i, d);
```

`decompose` is expecting pointers as its second and third arguments, but it's been given the *values* of `i` and `d` instead. `decompose` has no way to tell the difference, so it will use the values of `i` and `d` as though they were pointers. When `decompose` stores values in `*int_part` and `*frac_part`, it will attempt to change unknown memory locations instead of modifying `i` and `d`.

If we've provided a prototype for `decompose` (as we should always do, of course), the compiler will let us know that we're attempting to pass arguments of the wrong type. In the case of `scanf`, however, failing to pass pointers often goes undetected by the compiler, making `scanf` an especially error-prone function.

PROGRAM

Finding the Largest and Smallest Elements in an Array

To illustrate how pointers are passed to functions, let's look at a function named `max_min` that finds the largest and smallest elements in an array. When we call `max_min`, we'll pass it pointers to two variables; `max_min` will then store its answers in these variables. `max_min` has the following prototype:

```
void max_min(int a[], int n, int *max, int *min);
```

A call of `max_min` might have the following appearance:

```
max_min(b, N, &big, &small);
```

`b` is an array of integers; `N` is the number of elements in `b`. `big` and `small` are ordinary integer variables. When `max_min` finds the largest element in `b`, it stores the value in `big` by assigning it to `*max`. (Since `max` points to `big`, an assignment to `*max` will modify the value of `big`.) `max_min` stores the smallest element of `b` in `small` by assigning it to `*min`.

To test `max_min`, we'll write a program that reads 10 numbers into an array, passes the array to `max_min`, and prints the results: