

be legal if `a` has automatic storage duration—it would be treated as a variable-length array—but not if it has static storage duration.)

- It's legal to apply the address operator (`&`) to a `const` object, since it has an address. A macro doesn't have an address.

There are no absolute rules that dictate when to use `#define` and when to use `const`. I recommend using `#define` for constants that represent numbers or characters. That way, you'll be able to use the constants as array dimensions, in `switch` statements, and in other places where constant expressions are required.

18.4 Declarators

A declarator consists of an identifier (the name of the variable or function being declared), possibly preceded by the `*` symbol or followed by `[]` or `()`. By combining `*`, `[]`, and `()`, we can create declarators of mind-numbing complexity.

Before we look at the more complicated declarators, let's review the declarators that we've seen in previous chapters. In the simplest case, a declarator is just an identifier, like `i` in the following example:

```
int i;
```

Declarators may also contain the symbols `*`, `[]`, and `()`:

- A declarator that begins with `*` represents a pointer:

```
int *p;
```

- A declarator that ends with `[]` represents an array:

```
int a[10];
```

The brackets may be left empty if the array is a parameter, if it has an initializer, or if its storage class is `extern`:

```
extern int a[];
```

Since `a` is defined elsewhere in the program, the compiler doesn't need to know its length here. (In the case of a multidimensional array, only the first set of brackets can be empty.) C99 provides two additional options for what goes between the brackets in the declaration of an array parameter. One option is the keyword `static`, followed by an expression that specifies the array's minimum length. The other is the `*` symbol, which can be used in a function prototype to indicate a variable-length array argument. Section 9.3 discusses both C99 features.

- A declarator that ends with `()` represents a function:

```
int abs(int i);
void swap(int *a, int *b);
int find_largest(int a[], int n);
```

C99