7.5 Type Definitions

In Section 5.2, we used the `#define` directive to create a macro that could be used as a Boolean type:

```
#define BOOL int
```

Q&A There's a better way to set up a Boolean type, though, using a feature known as a *type definition:*

```
typedef int Bool;
```

Notice that the name of the type being defined comes *last*. Note also that I've capitalized the word `Bool`. Capitalizing the first letter of a type name isn't required; it's just a convention that some C programmers employ.

Using `typedef` to define `Bool` causes the compiler to add `Bool` to the list of type names that it recognizes. `Bool` can now be used in the same way as the built-in type names—in variable declarations, cast expressions, and elsewhere. For example, we might use `Bool` to declare variables:

```
Bool flag;   /* same as int flag; */
```

The compiler treats `Bool` as a synonym for `int`; thus, `flag` is really nothing more than an ordinary `int` variable.

Advantages of Type Definitions

Type definitions can make a program more understandable (assuming that the programmer has been careful to choose meaningful type names). For example, suppose that the variables `cash_in` and `cash_out` will be used to store dollar amounts. Declaring `Dollars` as

```
typedef float Dollars;
```

and then writing

```
Dollars cash_in, cash_out;
```

is more informative than just writing

```
float cash_in, cash_out;
```

Type definitions can also make a program easier to modify. If we later decide that `Dollars` should really be defined as `double`, all we need do is change the type definition:

```
typedef double Dollars;
```