Therefore, C++ won't permit you to overload `gronk()` in this fashion. You can have different return types, but only if the signatures also are different:

```cpp
long gronk(int n, float m);        // different signatures, hence allowed
double gronk(float n, float m);    // hence allowed
```

After we discuss templates later in this chapter, we'll further discuss function matching.

## An Overloading Example

We've already developed a `left()` function that returns a pointer to the first \( n \) characters in a string. Let's add a second `left()` function, one that returns the first \( n \) digits in an integer. You can use it, for example, to examine the first three digits of a U.S. postal ZIP code stored as an integer, a useful act if you want to sort for urban areas.

The integer function is a bit more difficult to program than the string version, because we don't have the benefit of each digit being stored in its own array element. One approach is first to compute the number of digits in the number. Dividing a number by 10 lops off one digit, so you can use division to count digits. More precisely, you can do so with a loop like this:

```cpp
unsigned digits = 1;
while (n /= 10)
    digits++;
```

This loop counts how many times you can remove a digit from \( n \) until none are left. Recall that \( n /= 10 \) is short for \( n = n / 10 \). If \( n \) is 8, for example, the test condition assigns to \( n \) the value 8 / 10, or 0, because it's integer division. That terminates the loop, and `digits` remains at 1. But if \( n \) is 238, the first loop test sets \( n \) to 238 / 10, or 23. That's nonzero, so the loop increases `digits` to 2. The next cycle sets \( n \) to 23 / 10, or 2. Again, that's nonzero, so `digits` grows to 3. The next cycle sets \( n \) to 2 / 10, or 0, and the loop quits, leaving `digits` set to the correct value, 3.

Now suppose you know the number has five digits, and you want to return the first three digits. You can get that value by dividing the number by 10 and then dividing the answer by 10 again. Each division by 10 lops one more digit off the right end. To calculate the number of digits to lop, just subtract the number of digits to be shown from the total number of