```
    square += odd;
  }

  return 0;
}
```

The `for` statement in this program initializes one variable (`square`), tests another (`i`), and increments a third (`odd`). `i` is the number to be squared, `square` is the square of `i`, and `odd` is the odd number that must be added to the current square to get the next square (allowing the program to compute consecutive squares without performing any multiplications).

The tremendous flexibility of the `for` statement can sometimes be useful; we'll find it to be a great help when working with linked lists. The `for` statement can easily be misused, though, so don't go overboard. The `for` loop in `square3.c` would be a lot clearer if we rearranged its pieces so that the loop is clearly controlled by `i`.

linked lists ➤ 17.5

6.4 Exiting from a Loop

We've seen how to write loops that have an exit point before the loop body (using `while` and `for` statements) or after it (using `do` statements). Occasionally, however, we'll need a loop with an exit point in the middle. We may even want a loop to have more than one exit point. The `break` statement makes it possible to write either kind of loop.

After we've examined the `break` statement, we'll look at a couple of related statements: `continue` and `goto`. The `continue` statement makes it possible to skip part of a loop iteration without jumping out of the loop. The `goto` statement allows a program to jump from one statement to another. Thanks to the availability of statements such as `break` and `continue`, the `goto` statement is rarely used.

The `break` Statement

We've already discussed how a `break` statement can transfer control out of a `switch` statement. The `break` statement can also be used to jump out of a `while`, `do`, or `for` loop.

Suppose that we're writing a program that checks whether a number `n` is prime. Our plan is to write a `for` statement that divides `n` by the numbers between 2 and `n - 1`. We should break out of the loop as soon as any divisor is found; there's no need to try the remaining possibilities. After the loop has terminated, we can use an `if` statement to determine whether termination was premature (hence `n` isn't prime) or normal (`n` is prime):

```
for (d = 2; d < n; d++)
  if (n % d == 0)
    break;
```