To save space, programmers sometimes put several case labels on the same line:

```
switch (grade) {
  case 4: case 3: case 2: case 1:
          printf("Passing");
          break;
  case 0:  printf("Failing");
          break;
  default: printf("Illegal grade");
          break;
}
```

Unfortunately, there's no way to write a case label that specifies a range of values, as there is in some programming languages.

A `switch` statement isn't required to have a `default` case. If `default` is missing and the value of the controlling expression doesn't match any of the case labels, control simply passes to the next statement after the `switch`.

The Role of the `break` Statement

Now, let's take a closer look at the mysterious `break` statement. As we've seen, executing a `break` statement causes the program to "break" out of the `switch` statement; execution continues at the next statement after the `switch`.

The reason that we need `break` has to do with the fact that the `switch` statement is really a form of "computed jump." When the controlling expression is evaluated, control jumps to the case label matching the value of the `switch` expression. A case label is nothing more than a marker indicating a position within the `switch`. When the last statement in the case has been executed, control "falls through" to the first statement in the following case; the case label for the next case is ignored. Without `break` (or some other jump statement), control will flow from one case into the next. Consider the following `switch` statement:

```
switch (grade) {
  case 4:  printf("Excellent");
  case 3:  printf("Good");
  case 2:  printf("Average");
  case 1:  printf("Poor");
  case 0:  printf("Failing");
  default: printf("Illegal grade");
}
```

If the value of `grade` is 3, the message printed is

```
GoodAveragePoorFailingIllegal grade
```

Forgetting to use `break` is a common error. Although omitting `break` is sometimes done intentionally to allow several cases to share code, it's usually just an oversight.