```
  result_denom = denom1 * denom2;
  printf("The sum is %d/%d\n", result_num, result_denom);

  return 0;
}
```

A session with this program might have the following appearance:

```
Enter first fraction: 5/6
Enter second fraction: 3/4
The sum is 38/24
```

Note that the resulting fraction isn't reduced to lowest terms.

Q & A

***Q: I've seen the `%i` conversion used to read and write integers. What's the difference between `%i` and `%d`? [p. 39]**

A: In a `printf` format string, there's no difference between the two. In a `scanf` format string, however, `%d` can only match an integer written in decimal (base 10) form, while `%i` can match an integer expressed in octal (base 8), decimal, or hexadecimal (base 16). If an input number has a `0` prefix (as in `056`), `%i` treats it as an octal number; if it has a `0x` or `0X` prefix (as in `0x56`), `%i` treats it as a hex number. Using `%i` instead of `%d` to read a number can have surprising results if the user should accidentally put `0` at the beginning of the number. Because of this trap, I recommend sticking with `%d`.

octal numbers ➤7.1

hexadecimal numbers ➤7.1

Q: If `printf` treats `%` as the beginning of a conversion specification, how can I print the `%` character?

A: If `printf` encounters two consecutive `%` characters in a format string, it prints a single `%` character. For example, the statement

```
printf("Net profit: %d%%\n", profit);
```

might print

```
Net profit: 10%
```

Q: The `\t` escape is supposed to cause `printf` to advance to the next tab stop. How do I know how far apart tab stops are? [p. 41]

A: You don't. The effect of printing `\t` isn't defined in C; it depends on what your operating system does when asked to print a tab character. Tab stops are typically eight characters apart, but C makes no guarantee.

Q: What does `scanf` do if it's asked to read a number but the user enters nonnumeric input?