

• Exercises

1. Assuming there are 7.481 gallons in a cubic foot, write a program that asks the user to enter a number of gallons, and then displays the equivalent in cubic feet.
2. Write a program that generates the following table:

1990	135
1991	7290
1992	11300
1993	16200

Use a single cout statement for all output.

3. Write a program that generates the following output:

```
10
20
19
```

Use an integer constant for the 10, an arithmetic assignment operator to generate the 20, and a decrement operator to generate the 19.

4. Write a program that displays your favorite poem. Use an appropriate escape sequence for the line breaks. If you don't have a favorite poem, you can borrow this one by Ogden Nash:

```
Candy is dandy,  
But liquor is quicker.
```

5. A library function, `islower()`, takes a single character (a letter) as an argument and returns a nonzero integer if the letter is lowercase, or zero if it is uppercase. This function requires the header file `CTYPE.H`. Write a program that allows the user to enter a letter, and then displays either zero or nonzero, depending on whether a lowercase or uppercase letter was entered. (See the `SQRT` program for clues.)

6. On a certain day the British pound was equivalent to \$1.487 U.S., the French franc was \$0.172, the German deutschemark was \$0.584, and the Japanese yen was \$0.00955. Write a program that allows the user to enter an amount in dollars, and then displays this value converted to these four other monetary units.

7. You can convert temperature from degrees Celsius to degrees Fahrenheit by multiplying by $9/5$ and adding 32. Write a program that allows the user to enter a floating-point number representing degrees Celsius, and then displays the corresponding degrees Fahrenheit.

8. When a value is smaller than a field specified with `setw()`, the unused locations are, by default, filled in with spaces. The manipulator `setfill()` takes a single character as an argument and causes this character to be substituted for spaces in the empty parts of a field. Rewrite the `WIDTH` program so that the characters on each line between the location name and the population number are filled in with periods instead of spaces, as in

Portcity.....2425785

9. If you have two fractions, a/b and C/D , their sum can be obtained from the formula

$$\frac{a}{b} + \frac{c}{d} = \frac{a*d + b*c}{b*d}$$

For example, $1/4$ plus $2/3$ is

$$\frac{1}{4} + \frac{2}{3} = \frac{1*3 + 4*2}{4*3} = \frac{3 + 8}{12} = \frac{11}{12}$$

Write a program that encourages the user to enter two fractions, and then displays their sum in fractional form. (You don't need to reduce it to lowest terms.) The interaction with the user might look like this:

```
Enter first fraction: 1/2
Enter second fraction: 2/5
Sum = 9/10
```

You can take advantage of the fact that the extraction operator ($>>$) can be chained to read in more than one quantity at once:

```
cin >> a >> dummychar >> b;
```

10. In the heyday of the British empire, Great Britain used a monetary system based on pounds, shillings, and pence. There were 20 shillings to a pound, and 12 pence to a shilling. The notation for this old system used the pound sign, £, and two decimal points, so that, for example, £5.2.8 meant 5 pounds, 2 shillings, and 8 pence. (Pence is the plural of penny.) The new monetary system, introduced in the 1950s, consists of only pounds

and pence, with 100 pence to a pound (like U.S. dollars and cents). We'll call this new system decimal pounds. Thus £5.2.8 in the old notation is £5.13 in decimal pounds (actually £5.1333333). Write a program to convert the old pounds-shillings-pence format to decimal pounds. An example of the user's interaction with the program would be

```
Enter pounds: 7
Enter shillings: 17
Enter pence: 9
Decimal pounds = £7.89
```

In both Borland C++ and Turbo C++, you can use the hex character constant '\x9c' to represent the pound sign (£). In Borland C++, you can put the pound sign into your program directly by pasting it from the Windows Character Map accessory.

11. By default, output is right-justified in its field. You can left-justify text output using the manipulator `setiosflags(ios::left)`. (For now, don't worry about what this new notation means.) Use this manipulator, along with `setw()`, to help generate the following output:

Last name	First name	Street address	Town	State
Jones	Bernard	109 Pine Lane	Littletown	MI
O'Brian	Coleen	42 E. 99th Ave.	Bigcity	NY
Wong	Harry	121-A Alabama St.	Lakeville	IL

12. Write the inverse of Exercise 10, so that the user enters an amount in Great Britain's new decimal-pounds notation (pounds and pence), and the program converts it to the old pounds-shillings-pence notation. An example of interaction with the program might be

Enter decimal pounds: 3.51

Equivalent in old notation = £3.10.2.

Make use of the fact that if you assign a floating-point value (say 12.34) to an integer variable, the decimal fraction (0.34) is lost; the integer value is simply 12. Use a cast to avoid a compiler warning. You can use statements like

```
Float decpounds; // input from user (new-style pounds)
```

```
Int pounds;      // old-style (integer) pounds
```

```
Float decfrac;   // decimal fraction (smaller than 1.0)
```

```
Pounds = static_cast<int>(decpounds); // remove decimal fraction
```

```
decfrac = decpounds - pounds; // regain decimal fraction
```

You can then multiply decfrac by 20 to find shillings. A similar operation obtains pence.

Good luck...

Mostafa elbarody