**2.8** What is wrong with this code:

```
enum Semester {FALL, SPRING, SUMMER};
enum Season {SPRING, SUMMER, FALL, WINTER};
```

**2.9** What is wrong with this code:

```
enum Friends {"Jerry", "Henry", "W.D."};
```

## Problems

**2.1** Write four different C++ statements, each subtracting 1 from the integer variable `n`.

**2.2** Write a block of C++ code that has the same effect as the statement

```
n = 100 + m++;
```

without using the post-increment operator.

**2.3** Write a block of C++ code that has the same effect as the statement

```
n = 100 + ++m;
```

without using the pre-increment operator.

**2.4** Write a single C++ statement that subtracts the sum of `x` and `y` from `z` and then increments `y`.

**2.5** Write a single C++ statement that decrements the variable `n` and then adds it to `total`.

**2.6** Write and run a program like the one in Example 2.2 on page 19 that prints the ASCII codes for only the 10 upper case and lower case vowels. Use Appendix A to check your output.

**2.7** Write and run a program to find which, if any, arithmetic operations can be applied to a variable that will change its value from any of the three numeric constants `inf`, `-inf`, and `nan` to something else.

**2.8** Modify the program in Example 2.15 on page 28 so that it uses type `double` instead of `float`. Then see how much better it performs on the input that illustrated round-off error.

**2.9** Write a program that converts inches to centimeters. For example, if the user enters 16.9 for a length in inches, the output would be `42.926 cm`. (One inch equals 2.54 centimeters.)

**2.10** Write and run a program that prints the sum, difference, product, quotient, and remainder of two integers that are input interactively.

## Answers to Review Questions

**2.1** All three are integer types. But `short` uses 16 bits, `int` uses 32 bits, and `long` uses 64 bit.

**2.2** The only difference is that the four constants that are defined by the `enum` statement have type `Direction` instead type `int`.

**2.3** The `unsigned` integer types are used primarily for bit strings.

**2.4** If used stand-alone, like this

```
++n;
i++;
```

then there is no difference; they both simply add 1 to the value of the variable. But if used within a larger expression, like this

```
m = ++n;
cout << i++;
```

then the difference is that the pre-increment will add 1 first and then use the resulting value within the larger expression, while the post-increment will add 1 after using the current value within the larger expression.

**2.5** The name refers to the C language and its increment operator `++`. The name suggests that C++ is an advance over C.

**2.6** *a.* `m` will be 10 and `n` will be 3.
*b.* `m` will be 6 and `n` will be 1.

**2.7** *a.* `m - 8 - n` evaluates to (25 - 8) - 7 = 17 - 7 = 10
*b.* `m = n = 3` evaluates to 3
*c.* `m%n` evaluates to 25%7 = 4
*d.* `m%n++` evaluates to 25%(7++) = 25%7 = 4
*e.* `m%++n` evaluates to 25%(++7) = 25%8 = 1
*f.* `++m - n--` evaluates to (++25) - (7--) = 26 - 7 = 19

**2.8** The second `enum` definition attempts to redefine the constants `SPRING`, `SUMMER`, and `FALL`.

**2.9** Enumerators must be valid identifiers. String literals like `"Jerry"` and `"Henry"` are not identifiers.

## Solutions to Problems

**2.1** Four different statements, each subtracting 1 from the integer variable `n`:

*a.*
```
n = n - 1;
```
*b.*
```
n -= 1;
```
*c.*
```
--n;
```
*d.*
```
n--;
```

**2.2**
```
n = 100 + m;
++m;
```

**2.3**
```
++m;
n = 100 + m;
```

**2.4**
```
z -= (x + y++);
```

**2.5**
```
total += --n;
```

**2.6**
```
int main()
{ // prints the ASCII codes of the vowels
  cout << "int('A') = " << int('A') << endl;
  cout << "int('E') = " << int('E') << endl;
  cout << "int('I') = " << int('I') << endl;
  cout << "int('O') = " << int('O') << endl;
  cout << "int('U') = " << int('U') << endl;
  cout << "int('a') = " << int('a') << endl;
  cout << "int('e') = " << int('e') << endl;
  cout << "int('i') = " << int('i') << endl;
  cout << "int('o') = " << int('o') << endl;
  cout << "int('u') = " << int('u') << endl;
}
```

```
int('A') = 65
int('E') = 69
int('I') = 73
int('O') = 79
int('U') = 85
int('a') = 97
int('e') = 101
int('i') = 105
int('o') = 111
int('u') = 117
```

**2.7** The following program changes the value of `x` from `inf` to `-inf` and *vice versa*. But no arithmetic operation will change the value of a variable once it becomes `nan`.

```
int main()
{ // changes the value of x after it becomes inf:
  float x=1e30;
```

```
    cout << "x= " << x << endl;
    x *= x;
    cout << "x= " << x << endl;
    x *= -1.0;
    cout << "x= " << x << endl;
    x *= -1.0;
    cout << "x= " << x << endl;
}
```

```
x= 1e+30
x= inf
x= -inf
x= inf
```

**2.8** With `double` in place of `float`, the output is:

```
Enter the coefficients of a quadratic equation:
        a: 2
        b: 8.001
        c: 8.002
The equation is: 2*x*x + 8.001*x + 8.002 = 0
The solutions are:
        x1 = -2
        x2 = -2.0005
Check:
        a*x1*x1 + b*x1 + c = 0
        a*x2*x2 + b*x2 + c = 0
```

**2.9** We use two variables of type `float`

```
int main()
{ // converts inches to centimeters:
  float inches, cm;
  cout << "Enter length in inches: ";
  cin >> inches;
  cm = 2.54*inches;
  cout << inches << " inches = " << cm << " centimeters.\n";
}
```

```
Enter length in inches: 16.9
16.9 inches = 42.926 centimeters.
```

**2.10**

```
int main()
{ // prints the results of arithmetic operators
  int m = 60, n = 7;
  cout << "The integers are " << m << " and " << n << endl;
  cout << "Their sum is        " << (m + n) << endl;
  cout << "Their difference is " << (m - n) << endl;
  cout << "Their product is    " << (m * n) << endl;
  cout << "Their quotient is   " << (m / n) << endl;
  cout << "Their remainder is  " << (m % n) << endl;
}
```

```
The integers are 60 and 7
Their sum is        67
Their difference is 53
Their product is    420
Their quotient is   8
Their remainder is  4
```