Students:
Section 2.7 is a part of 2 assignments: CSC108 CH02.1-2.10 C2A

This assignment's due date has passed. Activity will still be recorded, but will not count towards this assignment (unless the due date is changed). See this article for more info.

Includes: CA
Due: 02/04/2025, 11:59 PM EST

2.7 Floating-point numbers (double)

Floating-point (double) variables

A ***floating-point number*** is a real number containing a decimal point that can appear anywhere (or "float") in the number. Ex: 98.6, 0.0001, or 65,167. A ***double*** variable stores a floating-point number. Ex: `double milesDriven;` declares a double variable.

A ***floating-point literal*** is a number with a fractional part, even if the fraction is 0, as in 1.0, 0.0, or 99.573. *Good practice is to always have a digit before the decimal point, as in 0.5, since .5 might mistakenly be viewed as 5.*

Figure 2.7.1: Variables of type double: Travel time example.

```
#include <iostream>
using namespace std;

int main() {
   double milesTravel;  // User input of miles to travel
   double hoursFlying;  // Travel hours if flying those miles
   double hoursDrive;   // Travel hours if driving those miles

   cout << "Enter miles to travel: ";
   cin  >> milesTravel;

   hoursFly  = milesTravel / 500.0; // Plane flies 500 mph
   hoursDrive = milesTravel / 65.0;  // Car drives 65 mph

   cout << milesTravel << " miles would take:" << endl;
   cout << "  " << hoursFly << " hours to fly" << endl;
   cout << "  " << hoursDrive << " hours to drive" << endl;

   return 0;
}
```

```
Enter miles to travel: 1855
1855 miles would take:
  3.71 hours to fly
  28.5385 hours to drive.

Enter miles to travel: 450.5
450.5 miles would take:
  0.901 hours to fly.
  6.93077 hours to drive.
```

Feedback?

PARTICIPATION ACTIVITY 2.7.1: Declaring and assigning double variables.

All variables are of type double and already declared unless otherwise noted.

1) Declare a double variable named *personHeight*.

Check Show answer

2) Declare a double variable named *packageWeight* and initialize the variable to 7.1.

Check Show answer

3) Assign *ballRadius* with *ballHeight* divided by 2.0. Do not use the fraction 1.0 / 2.0; instead, divide *ballHeight* directly by 2.0.

Check Show answer

4) Assign *ballRadius* with *ballHeight* multiplied by (1.0 / 2.0). Include the parentheses around the fraction.

Check Show answer

Feedback?

PARTICIPATION ACTIVITY 2.7.2: Floating-point literals.

1) Which statement best declares and initializes the double variable?
   - `double currHumidity = 99;`
   - `double currHumidity = 99.0;`
   - `double currHumidity = 99.`

2) Which statement best assigns the variable? Both variables are of type double.
   - `cityRainfall = measuredRain - 5;`
   - `cityRainfall = measuredRain - 5.0;`

3) Which statement best assigns the variable? *cityRainfall* is of type double.
   - `cityRainfall = .97;`
   - `cityRainfall = 0.97;`

Feedback?

Scientific notation

Very large and very small floating-point values may be printed using scientific notation. Ex: If a floating variable holds the value 299792458.0 (the speed of light in m/s), the value will be printed as 2.99792e+08.

Choosing a variable type (double vs. int)

A programmer should choose a variable's type based on the type of value held.

- Integer variables are typically used for values that are counted, like 42 cars, 10 pizzas, or -95 days.
- Floating-point variables are typically used for measurements, like 98.6 degrees, 0.00001 meters, or -55.667 degrees.
- Floating-point variables are also used when dealing with fractions of countable items, such as the average number of cars per household.

Floating-point for money

Some programmers warn against using floating-point for money, as in 14.53 representing 14 dollars and 53 cents, because money is a countable item (reasons are discussed further in another section). int may be used to represent cents or to represent dollars when cents are not included as for an annual salary, as in 40000 dollars, which are countable.

PARTICIPATION ACTIVITY 2.7.3: Floating-point versus integer.

Choose the best type for a variable to represent each item.

1) The number of cars in a parking lot.
   - double
   - int

2) The current temperature in Celsius.
   - double
   - int

3) A person's height in centimeters.
   - double
   - int

4) The number of hairs on a person's head.
   - double
   - int

5) The average number of kids per household.
   - double
   - int

Feedback?

Floating-point division by zero

Dividing a nonzero floating-point number by zero is undefined in regular arithmetic. Many programming languages produce an error when performing floating-point division by 0, but C++ does not. C++ handles this operation by producing infinity or -infinity, depending on the signs of the operands. Printing a floating-point variable that is holds infinity or -infinity outputs `inf` or `-inf`.

If the dividend and divisor in floating-point division are both 0, the division results in a "not a number". **Not a number (NaN)** indicates an unrepresentable or undefined value. Printing a floating-point variable that is not a number outputs `nan`.

Figure 2.7.2: Floating-point division by zero example.

```
#include <iostream>
using namespace std;

int main() {
   double gasVolume;
   double milesDriven;
   double mileRatio;

   cout << "Gallons of gas volume: ";
   cin  >> gasVolume;

   cout << "Miles (mi) volume: ";
   cin  >> milesDriven;

   mileRatio = gasVolume / milesDriven;

   cout << "Gas to mi ratio is " << mileRatio << "\n" << endl;

   return 0;
}
```

```
Gallons of gas volume: 10.5
Miles (mi) volume: 0.0
Gas to mi ratio is inf.
```

Feedback?

PARTICIPATION ACTIVITY 2.7.4: Floating-point division.

Determine the result.

1) 13.0 / 3.0
   - 4
   - 4.333333
   - Positive infinity

2) 0.0 / 5.0
   - 0.0
   - Positive infinity
   - Negative infinity

3) 13.0 / 0.0
   - 13.0
   - Positive infinity
   - Negative infinity

4) 0.0 / 0.0
   - 0.0
   - Infinity
   - Not a number

Feedback?

Manipulating floating-point output

Some floating point numbers have many digits after the decimal point. Ex: Irrational numbers (Ex: 3.14159265359...) and repeating decimals (Ex: 4.33333333...) have an infinite number of digits after the decimal. By default, most programming languages output at least 5 digits after the decimal point. But for many simple programs, this level of detail is not necessary. A common approach is to output floating-point numbers with a specific number of digits after the decimal to reduce complexity or produce a certain numerical type (Ex: Representing currency with two digits after the decimal). The syntax for outputting the double myFloat with two digits after the decimal point is:

```
cout << fixed << setprecision(2) << myFloat;
```

When outputting a certain number of digits after the decimal using cout, C++ rounds the last output digit, but the floating-point value remains the same. Manipulating how numbers are output is discussed in detail elsewhere.

Note: setprecision() is found in the iomanip library. fixed and setprecision() are manipulators that need only be written once if the desired number of digits after the decimal point is the same for multiple floating-point numbers. Ex:

```
cout << fixed << setprecision(2) << 3.1264 << endl;
cout << 2.7003 << endl;
```

outputs 3.13 and 2.70.

PARTICIPATION ACTIVITY 2.7.5: Reducing the output of pi.

Start | 2x speed

```
cout << "Default output of pi: " << M_PI << endl;
cout << fixed << setprecision(4) << M_PI << endl;
```

Default output of pi: 3.14159
pi reduced to 4 digits after the decimal: 3.1416

Captions

1. The mathematical constant pi (π) is irrational, a floating-point number whose digits after the decimal point are infinite and non-repeating. The cmath library defines the constant M_PI with the value of pi.
2. Though C++ does not attempt to output the full value of pi, by default, 5 digits after the decimal are output.
3. cout << fixed << setprecision(4); outputs pi to only four digits after the decimal. The last digit is rounded up in the output, but the value of pi remains the same.

Feedback?

PARTICIPATION ACTIVITY 2.7.6: Reducing floating-point output.

1) Which manipulator(s) is/are used to set cout to output two digits after the decimal point?
   ```
   cout << _____ << (7.0 / 3.0);
   ```
   - setprecision(2)
   - fixed
   - fixed << setprecision(2)

2) What is output by `cout << fixed << setprecision(1) << 0.1257`?
   - 0
   - 0.1
   - 0.13

3) What is output by `cout << fixed << setprecision(3) << 9.1357`?
   - 9.136
   - 9.135
   - 9.14

Feedback?

CHALLENGE ACTIVITY 2.7.1: Sphere volume.

Given sphereRadius, compute the volume of a sphere and assign sphereVolume with the result. Use (4.0 / 3.0) to perform floating-point division, instead of (4 / 3) which performs integer division.

Volume of sphere = (4.0 / 3.0) π r^3 (π and r^3 can be computed using *. Use the constant M_PI for the value of pi.)

(Sphere volume notes)

See How to Use zyBooks for info on how our automated program grader works.

```
1 #include <iostream>
2 #include <iomanip>
3 #include <cmath>
4
5 using namespace std;
6
7 int main() {
8    double sphereVolume;
9    double sphereRadius;
10
11   cin >> sphereRadius;
12
13   /* Your solution goes here */
14
15   cout << fixed << setprecision(3) << sphereVolume << endl;
16
17   return 0;
```

Run

View your last submission

Feedback?

CHALLENGE ACTIVITY 2.7.2: Floating-point numbers (double).

Start

The reciprocal of input_int is 1 / input_int. The following program intends to read a floating-point value from input, compute the reciprocal of the value, and output the reciprocal, but the code contains errors. Find and fix the errors.

Ex: If the input is 0.770, then the output is:

```
The reciprocal of 0.770 = 1.299
```

```
1 #include <iomanip>
2 #include <iostream>
3 using namespace std;
4
5 int main() {
6    // Modify the following code
7    float input_int;
8    int recipal;
9
10   cin >> input_int;
11
12   recipal = 1 / input_int;
13
14   cout << "The reciprocal of length = 1 / " << fixed << setprecision(3) << input_int;
15   cout << " = " << recipal << endl;
16
17   return 0;
```

1 2 3 4

Check Next level

Feedback?

How was this section? | Provide section feedback