2.10 Using math functions

Basics

Some programs require math operations beyond +, -, *, /, like computing a square root. A standard ***math library*** has about 20 math operations, known as functions. A programmer can include the library and then use those math functions.

A ***function*** is a list of statements executed by invoking the function's name, such invoking known as a ***function call***. Any function input values, or ***arguments***, appear within ( ), separated by commas if more than one. Below, function sqrt is called with one argument, areaSquare. The function call evaluates to a value, as in sqrt(areaSquare) below evaluating to 7.0, which is assigned to sideSquare.

PARTICIPATION ACTIVITY 2.10.1: Using a math function.

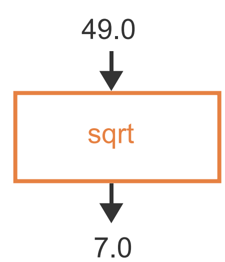
1 2 3 2x speed

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

int main() {
   double sideSquare;
   double areaSquare = 49.0;

   sideSquare = sqrt(areaSquare);
                    7.0
   cout << "Square root of " << areaSquare
        << " is " << sideSquare << endl;

   return 0;
}
```



Thus, sqrt(49.0) evaluates to 7.0.

Feedback?

Table 2.10.1: A few common math functions from the math library.

Function	Behavior	Example

sqrt(x)	Square root of x	sqrt(9.0) evaluates to 3.0.
pow(x, y)	Power: x^y	pow(6.0, 2.0) evaluates to 36.0.
fabs(x)	Absolute value of x	fabs(-99.5) evaluates to 99.5.

Feedback?

Other available functions are log (natural log), log2 (log base 2), log10 (log base 10), exp (raising e to a power), ceil (rounding up), floor (rounding down), various trigonometric functions like sin, cos, tan, and more. See this math functions link for a comprehensive list of built-in math functions.

PARTICIPATION ACTIVITY 2.10.2: Math functions.

1) sqrt(36.0) evaluates to _____ .

- (selected) 6.0
- 36.0

Correct

The function takes argument 36.0 as input, executes some statements to compute the square root 6.0, and evaluates to that value.

2) What is y?

```
y = sqrt(81.0);
```

- (selected) 9.0
- 81.0

Correct

sqrt(81.0) evaluates to 9.0.
y is assigned with that value.

3) What is y?

```
y = pow(2.0, 8.0);
```

- 64.0
- (selected) 256.0

Correct

2^8 is 256. Notice that the arguments are separated by a comma: pow(2.0, 8.0).

4) Is this a valid function call?

```
y = sqrt(2.0, 8.0);
```

- Yes
- (selected) No

Correct

sqrt() only accepts one argument. Trying to pass two arguments is an error.

5) Is this a valid function call?

```
y = pow(8.0);
```

Correct

Yes

No

pow() requires two arguments. Trying to call pow() with only one argument is an error.

6) If w and x are double variables, is this a valid function call?

```
y = pow(w, x);
```

Yes

No

Correct

The values of w and x will be passed as arguments to pow(). If w is 3.0 and x is 2.0, pow() will evaluate to 3.0^2 or 9.0.

7) What is y?

```
w = 3.0;
y = pow(w + 1.0, 2.0);
```

8.0

16.0

Correct

pow's first argument is 3.0 + 1.0 or 4.0. Thus, pow() evaluates to 4.0^2 or 16.0. Arguments may be expressions.

Feedback?

Example: Mass growth

The example below computes the growth of a biological mass, such as a tree. If the growth rate is 5% per year, the program computes 1.05 raised to the number of years. A similar program could calculate growth of money given an interest rate.

Figure 2.10.1: Math function example: Mass growth.

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

int main() {
   double initMass;   // Initial mass of a substance
   double growthRate; // Annual growth rate
   double yearsGrow;  // Years of growth
   double finalMass;  // Final mass after those years

   cout << "Enter initial mass: ";
   cin  >> initMass;

   cout << "Enter growth rate (Ex: 0.05 is 5%/year): ";
   cin  >> growthRate;

   cout << "Enter years of growth: ";
   cin  >> yearsGrow;

   finalMass = initMass * pow(1.0 + growthRate, yearsGrow);
   // Ex: Rate of 0.05 yields initMass * 1.05^yearsGrow

   cout << "Final mass after " << yearsGrow
        << " years is: " << finalMass << endl;

   return 0;
}
```

```
Enter initial mass: 10000
Enter growth rate (Ex: 0.05 is 5%/year): 0.06
Enter years of growth: 20
Final mass after 20 years is: 32071.4

...

Enter initial mass: 10000
Enter growth rate (Ex: 0.05 is 5%/year): 0.40
Enter years of growth: 10
Final mass after 10 years is: 289255
```

Feedback?

PARTICIPATION ACTIVITY 2.10.3: Growth rate.

1) If initMass is 10.0, growthRate is 1.0 (100%), and yearsGrow is 3, what is finalMass?

```
finalMass = initMass * pow(1.0 + growthRate, yearsGrow);
```

80.0

Check Show answer

Correct

80.0

10.0 * pow(1.0 + 1.0, 3.0)
10.0 * pow(2.0, 3.0)
10.0 * 8.0
80.0

Feedback?

PARTICIPATION ACTIVITY

2.10.4: Calculate Pythagorean theorem using math functions.

Select the three statements needed to calculate the value of x in the following:

$x = \sqrt{y^2 + z^2}$

For this exercise, calculate y^2 before z^2.

1) First statement is:

- ( ) temp1 = pow(x , 2.0);
- ( ) temp1 = pow(z , 3.0);
- (•) temp1 = pow(y , 2.0);
- ( ) temp1 = sqrt(y);

Correct

Statement assigns y squared to temp1.

2) Second statement is:

- ( ) temp2 = sqrt(x , 2.0);
- (•) temp2 = pow(z , 2.0);
- ( ) temp2 = pow(z);
- ( ) temp2 = x + sqrt(temp1 + temp2);

Correct

Statement assigns z squared to temp2.

3) Third statement is:

- ( ) temp2 = sqrt(temp1 + temp2);
- ( ) x = pow(temp1 + temp2, 2.0);
- ( ) x = sqrt(temp1) + temp2;
- (•) x = sqrt(temp1 + temp2);

Correct

Statement assigns square root of (temp1 + temp2) to x.

Feedback?

Calls in arguments

Commonly a function call's argument itself includes a function call. Below, x^y is computed via pow(x, y). The result is used in an expression that is an argument to another call, in this case to pow() again: pow(2.0, pow(x, y) + 1).

PARTICIPATION ACTIVITY | 2.10.5: Function call in an argument.

1 2 3 2x speed

$$z = 2^{(x^y + 1)}$$

$x^y \longrightarrow$ `pow(x, y)`

$2^{(x^y + 1)} \longrightarrow$ `pow(2.0, pow(x, y) + 1)`

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

int main() {
   double x;
   double y;
   double z;

   x = 3.0;             9.0      1024.0
   y = 2.0;
   z = pow(2.0, pow(x, y) + 1);

   return 0;
}
```

Upon execution, if x = 3.0 and y = 2.0, then pow(x, y) is called and evaluates to 9.0. Next, pow(2.0, 9.0+1) is called, yielding 1024.0.

Feedback?

PARTICIPATION ACTIVITY | 2.10.6: Function calls in arguments.

Type the ending value of z.

1)
```
z = pow(2.0, pow(2.0, 3.0));
```

256.0

Check Show answer

Correct

256.0

The innermost expression is evaluated first, namely pow(2.0, 3.0) yielding 2^3 or 8. Then pow(2.0, 8.0) is called, evaluating to 2^8 or 256.0.

2)
```
x = 9.0;
z = pow(sqrt(x) + sqrt(x), 2.0);
```

36.0

Check Show answer

Correct

36.0

First, sqrt(x) is called, so sqrt(9.0) evaluated to 3.0. Then, the second sqrt(x) is called, also yielding 3.0. Next, 3.0 + 3.0 is evaluated, yielding 6.0. Finally, pow(6.0, 2.0) is evaluated, yielding 36.0.

3)
```
x = -9.0;
z = sqrt(fabs(x));
```

3.0

Check Show answer

Correct

3.0

First, fabs(x) is evaluated, so fabs(-9.0) evaluates to 9.0. Then, sqrt(9.0) is called, evaluating to 3.0.

Feedback?

cmath and cstdlib

The "c" in cmath indicates that the library comes from a C language library.

Some math functions for integers are in a library named cstdlib, requiring: `#include <cstdlib>`. Ex: abs() computes the absolute value of an integer.

CHALLENGE ACTIVITY 2.10.1: Coordinate geometry.

Determine the distance between point (x1, y1) and point (x2, y2), and assign the result to pointsDistance. The calculation is:

$$Distance = \sqrt{(x2 - x1)^2 + (y2 - y1)^2}$$

Ex: For points (1.0, 2.0) and (1.0, 5.0), pointsDistance is 3.0.

```
   double xDist;
   double yDist;
   double pointsDistance;

   xDist = 0.0;
   yDist = 0.0;
   pointsDistance = 0.0;

   cin >> x1;
   cin >> y1;
   cin >> x2;
   cin >> y2;

```

```
   /* Your solution goes here  */
   pointsDistance = sqrt(pow((x2-x1),2)+ pow((y2-y1),2));


   cout << pointsDistance << endl;

   return 0;
}
```

Run

All tests passed

Testing with (1.0, 2.0) and (1.0, 5.0)

Your value 3

Testing with (2.0, 2.0) and (2.5, 3.5)

Value differs. See highlights below.

Your value 1.5811388300841898

Feedback?

CHALLENGE ACTIVITY 2.10.2: Tree height.

Simple geometry can compute the height of an object from the object's shadow length and shadow angle using the formula: tan(angleElevation) = treeHeight / shadowLength.

1. Using simple algebra, rearrange that equation to solve for treeHeight. (Note: Don't forget tangent).

2. Complete the below code to compute treeHeight. For tangent, use the tan() function, described in the "math functions" link above.

(Notes)

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

int main( ) {
   double treeHeight;
   double shadowLength;
   double angleElevation;

   cin >> angleElevation;
   cin >> shadowLength;

   /* Your solution goes here  */
```

```
    treeHeight = shadowLength*tan(angleElevation);

    cout << treeHeight << endl;

    return 0;
}
```

Run

All tests passed

Testing with shadowLength = 17.5, angleElevation = 0.11693706

Value differs. See highlights below.

Your value 2.055777526488276

Testing with shadowLength = 22.9, angleElevation = 0.34906585

Value differs. See highlights below.

Your value 8.334918354351979

Feedback?