2.11 Integer division and modulo

Division: Integer rounding

When the operands of / are integers, the operator performs integer division, which does not generate any fraction.

PARTICIPATION ACTIVITY 2.11.1: Integer division does not generate any fraction.

1 2 3 4 2x speed

y = 10 / 4;	y = 3 / 4;	a = (1 / 2) * b * h	f = c * (9/5) + 32	int w = 10;
				int x = 4;
				y = w / x;
					int w = 10;
					double x = 4.0;
					y = w / x;

2.5 2	0.75 0	0 0 ...	1	2	2.5
		Always 0	*Always c*1 + 32*		

If at least one operand of / is a floating-point type, then floating-point division occurs. So if int w = 10 and double x = 4.0, then w / x is 2.5.

Feedback?

The / operator performs floating-point division if at least one operand is a floating-point type.

PARTICIPATION ACTIVITY 2.11.2: Integer division modulo.

Determine the result. Some expressions only use literals to focus attention on the operator, but most practical expressions include variables.

1) `13 / 3`

4

Check Show answer

Correct

4

3 divides into 13 four times, yielding 12 with remainder 1 thrown away.

2) `4 / 9`

0

Check Show answer

Correct

0

9 does not divide into 4, so the answer is 0. No fraction is generated.

3) `(5 + 10 + 15) * (1 / 3)`

0

Check Show answer

Correct

0

1 / 3 performs integer division, evaluating to 0. Thus the entire expression evaluates to 0. A better expression would be (5 + 10 + 15) / 3.

4) `x / y` where int x = 10 and int y = 4.

2

Check Show answer

Correct

2

Both operands of /, x and y, are integers, so integer division is applied: 10 / 4 is 2.

5) `10 / 4.0`

2.5

Check Show answer

Correct

2.5

At least one operand of /, namely 4.0, is a floating-point type, so floating-point division is applied: 10 / 4.0 is 2.5.

6) `x / y` where int x = 10 and double y = 4.0.

2.5

Check Show answer

Correct

2.5

At least one operand of /, namely y, is a floating-point type, so floating-point division is applied: 10 / 4.0 is 2.5.

Feedback?

Division: Divide by 0

For integer division, the second operand of / or % must never be 0, because division by 0 is mathematically undefined. A ***divide-by-zero error*** occurs at runtime if a divisor is 0, causing a program to terminate. A divide-by-zero error is an example of a ***runtime error***, a severe error that occurs at runtime and causes a program to terminate early. In the example below, the program terminates and outputs the error message "Floating point exception" when the program attempts to divide by daysPerYear, which is 0.

Figure 2.11.1: Divide-by-zero example: Compute salary per day.

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

int main() {
   int salaryPerYear; // User input: Yearly salary
   int daysPerYear;   // User input: Days worked per year
   int salaryPerDay;  // Output:     Salary per day

   cout << "Enter yearly salary: ";
   cin  >> salaryPerYear;

   cout << "Enter days worked per year: ";
   cin  >> daysPerYear;

   // If daysPerYear is 0, then divide-by-zero causes program
termination.
   salaryPerDay = salaryPerYear / daysPerYear;

   cout << "Salary per day is: " << salaryPerDay << endl;

   return 0;
}
```

```
Enter yearly salary:
60000
Enter days worked per
year: 0
Floating point exception
```

Feedback?

PARTICIPATION ACTIVITY 2.11.3: More integer division.

Determine the result. Type "Error" if the program would terminate due to divide-by-zero. Only literals appear in these expressions to focus attention on the operators; most practical expressions include variables.

1) 100 / 2

50

Check Show answer

Correct

50

100 / 2 is just 50.

2) 100 * (1 / 2)

0

Check Show answer

Correct

0

The expression in the parentheses is evaluated first, yielding 0 due to integer division. Then 100 * 0 is 0.

3) 100 * 1 / 2

50

Check Show answer

Correct

50

Because * and / have the same precedence, evaluation is left-to-right. Thus 100 * 1 / 2 is 100 / 2, which is 50.

4) 100 / (1 / 2)

Error

Check Show answer

Correct

Error

The expression in the parentheses is evaluated first, yielding 0 due to integer division. Then 100 / 0 is undefined and causes the program to terminate.

5)
```
x = 2;
y = 5;
z = 1 / (y - x - 3);
```

Error

Check Show answer

Correct

Error

The expression in the parentheses is evaluated first: 5 - 2 - 3 is 0. Then 1 / 0 is undefined and causes the program to terminate.

Feedback?

Modulo (%)

The basic arithmetic operators include not just +, -, *, /, but also %. The ***modulo operator*** (**%**) evaluates the remainder of the division of two integer operands. Ex: 23 % 10 is 3.

Examples:

- 24 % 10 is 4. Reason: 24 / 10 is 2 with remainder 4.
- 50 % 50 is 0. Reason: 50 / 50 is 1 with remainder 0.
- 1 % 2 is 1. Reason: 1 / 2 is 0 with remainder 1.
- 10 % 4.0 is not valid. "Remainder" only makes sense for integer operands.

Figure 2.11.2: Division and modulo example: Minutes to hours/minutes.

```
Enter minutes:  367
367 minutes is 6 hours and 7 minutes.

...

Enter minutes:  180
180 minutes is 3 hours and 0 minutes.
```

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

int main() {
   int userMinutes;   // User input: Minutes
   int outHours;      // Output hours
   int outMinutes;    // Output minutes (remaining)

   cout << "Enter minutes: ";
   cin  >> userMinutes;

   outHours   = userMinutes / 60;
   outMinutes = userMinutes % 60;

   cout << userMinutes << " minutes is ";
   cout << outHours    << " hours and ";
   cout << outMinutes  << " minutes." << endl;

   return 0;
}
```

Feedback?

PARTICIPATION ACTIVITY 2.11.4: Modulo.

Determine the result. Type "Error" if appropriate. Only literals appear in these expressions to focus attention on the operators; most practical expressions include variables.

1) `50 % 2`

0

Check Show answer

Correct

0

50 / 2 is 25 with remainder 0.

2) `51 % 2`

1

Check Show answer

Correct

1

Note that any odd number % 2 is 1, while any even number % 2 is 0.

3) `78 % 10`

8

Check Show answer

Correct

8

78 / 10 is 7 with remainder 8.

4) `596 % 10`

6

Check Show answer

Correct

6

Note that any number % 10 yields the digit in the rightmost (1s) place, in this case 6.

5) `100 % (1 / 2)`

Error

Check Show answer

Correct

Error

1 / 2 is 0. 100 % 0 is undefined (as 100 / 0 is undefined) and causes the program to terminate.

6) `100.0 % 40`

Error

Check Show answer

Correct

Error

% is only defined for integer operands; "remainder" makes no sense if floating-point division is performed. 100.0 / 40 is 2.5; no remainder exists because a fraction was generated. In contrast, 100 % 40 is 20 (100 / 40 is 2 remainder 20).

Feedback?

PARTICIPATION ACTIVITY 2.11.5: Integer division and modulo.

A florist wants to create as many bunches of 12 flowers as possible. totalFlowers holds the total number of flowers available.

1) Complete the statement to assign numBunches with the maximum number of bunches that can be made.

```
numBunches =
totalFlowers / 12 ;
```

Check Show answer

Correct

totalFlowers / 12

totalFlowers / 12 calculates the number of bunches that can be made with exactly 12 flowers. If totalFlowers is 45, 45 / 12 or 3 bunches can be made.

2) Using only the variable totalFlowers, complete the statement to assign remainingFlowers with the number of remaining flowers after creating as many bunches of 12 as possible.

Correct

totalFlowers % 12

totalFlowers % 12 calculates the remainder of totalFlowers divided by 12, which is the number of remaining flowers. If totalFlowers is 45, 3 bunches of 12 can be made, with 45 % 12 or 9 remaining flowers.

```
remainingFlowers =
totalFlowers % 12 ;
```

Check Show answer

Feedback?

Why parentheses matter

The following summary of a dialog on a popular programmer discussion forum shows the importance of using parentheses, in this case in an expression involving modulo.

Use these

(Poster A): Tried rand() % (35 - 18) + 18, but it's wrong.

(Poster B): I don't understand what you're doing with (35 - 18) + 18. Wouldn't that just be 35?

(Poster C): The % operator has higher precedence than the + operator. So read that as (rand() % (35 - 18)) + 18.

CHALLENGE ACTIVITY 2.11.1: Enter the output of the integer expressions.

Jump to level 1

Type the program's output.

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

int main() {
   int a;
   int x;
   int y;

   a = 2;
   x = 4;
   y = a * x;
   y = y + 6;

   cout << x << " " << y;

   return 0;
}
```

```
4 14
```

1 2 3 4

Check Next **Done**. Click any level to practice more. Completion is preserv

✔ First y is assigned with the result of a * x, which is 2 * 4, then 6 is added to y.

Yours `4 14`

Expected `4 14`

Feedback?

Modulo examples

Modulo has several useful applications. Below are just a few.

Example 2.11.1: Random number in range.

Given a random number randNum, % can generate a random number within a range:

- `randNum % 10`
  Yields 0 - 9: Possible remainders are 0, 1, ..., 8, 9. Remainder 10 is not possible: Ex: 19 % 10 is 9, but 20 % 10 is 0.
- `randNum % 51`
  Yields 0 - 50: Note that % 50 would yield 0 - 49.
- `(randNum % 9) + 1`
  Yields 1 - 9: The % 9 yields 9 possible values 0 - 8, so the + 1 yields 1 - 9.
- `(randNum % 11) + 20`
  Yields 20 - 30: The % 11 yields 11 possible values 0 - 10, so the + 20 yields 20 - 30.

Feedback?

Example 2.11.2: Getting digits.

Given a number, % and / can be used to get each digit. For a 3-digit number userVal like 927:

```
onesDigit     = userVal % 10;    // Ex: 927 % 10 is 7.
tmpVal        = userVal / 10;

tensDigit     = tmpVal % 10;     // Ex: tmpVal = 927 / 10 is 92. Then 92 % 10 is 2.
tmpVal        = tmpVal / 10;

hundredsDigit = tmpVal % 10;     // Ex: tmpVal = 92 / 10 = 9. Then 9 % 10 is 9.
```

Feedback?

Example 2.11.3: Get prefix of a phone number.

Given a 10-digit phone number stored as an integer, % and / can be used to get any part, such as the prefix. For phoneNum = 9365551212 (whose prefix is 555):

```
tmpVal = phoneNum / 10000; // / 10000 shifts right by 4, so 936555.
prefixNum = tmpVal % 1000; // % 1000 gets the right 3 digits, so 555.
```

Dividing by a power of 10 shifts a value right. 321 / 10 is 32. 321 / 100 is 3.

% by a power of 10 gets the rightmost digits. 321 % 10 is 1. 321 % 100 is 21.

Feedback?

PARTICIPATION ACTIVITY 2.11.6: Modulo examples.

1) Given a non-negative number x, which yields a number in the range 5 - 10?

- ○ x % 5
- ○ x % 10
- ○ x % 11
- ◉ (x % 6) + 5

Correct

% 6 yields 0 - 5. Then + 5 yields 5 - 10.

2) Given a non-negative number x, which expression has the range -10 to 10?

- ○ x % -10
- ◉

Correct

x % 21 yields 0 to 20. Then - 10 yields -10 to 10.

(x % 21) - 10

(x % 20) - 10

3) Which gets the tens digit of x. Ex: If x = 693, which yields 9?

- x % 10
- x % 100
- (x / 10) % 10

Correct

x / 10 shifts right one place, putting the tens digit in the ones place. Then % 10 gets the (new) ones digit. Ex: 693 / 10 is 69, then 69 % 10 is 9.

4) Given a 16-digit credit card number stored in x, which gets the last (rightmost) four digits? (Assume the fourth digit from the right is non-zero).

- x / 10000
- x % 10000

Correct

x % 10000 yields 0 - 9999, being the rightmost four digits. To get other digits like the next four digits, divide first to shift the desired digits to the rightmost digits, then use % to get just those digits.

Feedback?

CHALLENGE ACTIVITY 2.11.2: Compute change.

A cashier distributes change using the maximum number of five dollar bills, followed by one dollar bills. For example, 19 yields 3 fives and 4 ones. Write a single statement that assigns the number of 1 dollar bills to variable numOnes, given amountToChange. Hint: Use the % operator.

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

int main() {
   int amountToChange;
   int numFives;
   int numOnes;

   cin >> amountToChange;
   numFives = amountToChange / 5;

   /* Your solution goes here  */
   numOnes = amountToChange%5;

   cout << "numFives: " << numFives << endl;
   cout << "numOnes: " << numOnes << endl;

```

```
18    return 0;
19 }
```

Run ✓ All tests passed

✓ Testing with 19

Your output

```
numFives: 3
numOnes: 4
```

✓ Testing with 4

Your output

```
numFives: 0
numOnes: 4
```

✓ Testing with 50

Your output

```
numFives: 10
numOnes: 0
```

✓ Testing with 0

Your output

```
numFives: 0
numOnes: 0
```

Feedback?