Students:
Section 2.12 is a part of 2 assignments: CSC108 CH02.11-2.24 C2B

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: 02/06/2025, 11:59 PM EST

2.12 Type conversions

Type conversions

A calculation sometimes must mix integer and floating-point numbers. For example, given that about 50.4% of human births are males, then `0.504 * numBirths` calculates the number of expected males in numBirths births. If numBirths is an int variable (int because the number of births is countable), then the expression combines a floating-point and integer.

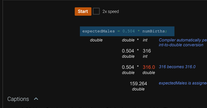
A ***type conversion*** is a conversion of one data type to another, such as an int to a double. The compiler automatically performs several common conversions between int and double types, such automatic conversions are known as ***implicit conversion***.

- For an arithmetic operator like + or *, if either operand is a double, the other is automatically converted to double, and then a floating-point operation is performed.
- For assignments, the right side type is converted to the left side type.

int-to-double conversion is straightforward: 25 becomes 25.0.

double-to-int conversion just drops the fraction: 4.9 becomes 4.

PARTICIPATION ACTIVITY 2.12.1: Implicit type conversion: int-to-double.



Captions

1. 0.504 is a floating-point literal. numBirths is an int variable. The compiler sees 'double * int', and performs an implicit int-to-double type conversion.
2. If numBirths is 316, 316.0 is first converted to 316.0.
3. Then, the program computes 0.504 * 316.0 yielding 159.264. expectedMales is a double variable and is assigned with that result.

Feedback?

PARTICIPATION ACTIVITY 2.12.2: Implicit conversions among double and int.

Type the value of the expression given int numItems = 5. For any floating-point answer, type answer to tenths. Ex: 8.0, 6.5, or 0.1.

1) 3.0 / 1.5

Check Show answer

2) 3.0 / 2

Check Show answer

3) (numItems + 10) / 2

Check Show answer

4) (numItems + 10) / 2.0

Check Show answer

Feedback?

PARTICIPATION ACTIVITY 2.12.3: Implicit conversions among double and int with variables.

Type the value held in the variable after the assignment statement, given int numItems = 5, and double itemWeight = 0.5. For any floating-point answer, type answer to tenths. Ex: 8.0, 6.5, or 0.1.

1) `// someDoubleVar is type double`
`someDoubleVar = itemWeight * numItems;`

Check Show answer

2) `// someIntVar is type int`
`someIntVar = itemWeight * numItems;`

Check Show answer

Feedback?

Assigning doubles with integer literals

Because of implicit conversion, statements like `double someDoubleVar = 0;` or `someDoubleVar = 5;` are allowed, but discouraged. Using 0.0 or 5.0 is preferable.

Type casting

A programmer sometimes needs to explicitly convert an item's type. Ex: If a program needs a floating-point result from dividing two integers, then at least one of the integers needs to be converted to double so floating-point division is performed. Otherwise, integer division is performed, evaluating to only the quotient and ignoring the remainder. A ***type cast*** explicitly converts a value of one type to another type.

The ***static_cast*** operator (`static_cast<type>(expression)`) converts the expression's value to the indicated type. Ex: If myIntVar is 7, then `static_cast<double>(myIntVar)` converts int 7 to double 7.0.

The program below casts the numerator and denominator each to double so floating-point division is performed (actually, converting only one would have worked).

Figure 2.12.1: Using type casting to obtain floating-point division.

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

int main() {
   int kidsInFamily1;        // Should be int, not double
   int kidsInFamily2;        // (know anyone with 2.3 kids?)
   int numFamilies;

   double avgKidsPerFamily;  // Expect fraction, so double

   kidsInFamily1 = 5;
   kidsInFamily2 = 4;
   numFamilies = 2;

   avgKidsPerFamily = static_cast<double>(kidsInFamily1 + kidsInFamily2)
                      / static_cast<double>(numFamilies);

   cout << "Average kids per family: " << avgKidsPerFamily << endl;

   return 0;
}
```

Average kids per family: 3.5

Feedback?

PARTICIPATION ACTIVITY 2.12.4: Type casting.

Determine the resulting type for each expression. Assume numSales1, numSales2, and totalSales are int variables.

1) (numSales1 + numSales2) / 2
   - int
   - double
2) static_cast<double>(numSales1 + numSales2) / 2
   - int
   - double
3) (numSales1 + numSales2) / totalSales
   - int
   - double
4) (numSales1 + numSales2) / static_cast<double>(totalSales)
   - int
   - double

Feedback?

Common errors

A common error is to accidentally perform integer division when floating-point division was intended. The program below undesirably performs integer division rather than floating-point division.

Figure 2.12.2: Common error: Forgetting cast results in integer division.

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

int main() {
   int kidsInFamily1;        // Should be int, not double
   int kidsInFamily2;        // (know anyone with 2.3 kids?)
   int numFamilies;

   double avgKidsPerFamily;  // Expect fraction, so double

   kidsInFamily1 = 5;
   kidsInFamily2 = 4;
   numFamilies = 2;

   avgKidsPerFamily = kidsInFamily1 + kidsInFamily2 / numFamilies;

   // Should be 3.5, but is 3 instead
   cout << "Average kids per family: " << avgKidsPerFamily << endl;

   return 0;
}
```

Average kids per family: 3

Feedback?

Another common error is to cast the entire result of integer division, rather than the operands, thus not obtaining the desired floating-point division.

PARTICIPATION ACTIVITY 2.12.5: Common error: Casting final result instead of operands.

Captions

1. The programmer wants to use floating-point division to compute the average of midtermScore (an int) and finalScore (an int). (midtermScore + finalScore) is evaluated first. If midtermScore is 90 and finalScore is 85, the expression evaluates to (90 + 85) or 175.
2. Next, 175 / 2 is evaluated. Both operands are integers, so integer division is performed, yielding 87.
3. The type cast converts 87 to 87.0. Casting the result of integer division does not perform the desired floating-point division.

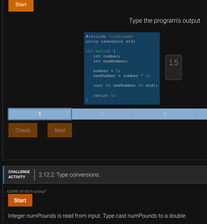
Feedback?

PARTICIPATION ACTIVITY 2.12.6: Type casting.

1) Which yields 2.5?
   - static_cast<int>(10) / static_cast<int>(4)
   - static_cast<double>(10) / static_cast<double>(4)
   - static_cast<double>(10 / 4)
2) Which does NOT yield 3.75?
   - static_cast<double>(15) / static_cast<double>(4)
   - static_cast<double>(15) / 4
   - 15 / static_cast<double>(4)
   - static_cast<double>(15 / 4)
3) Given aCount, bCount, and cCount are integer variables, which variable must be cast to a double for the expression `(aCount * bCount) / cCount` to evaluate to a double value?
   - None
   - All variables
   - Only one variable

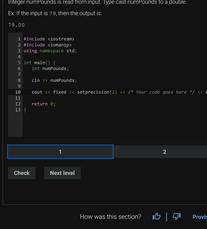
Feedback?

CHALLENGE ACTIVITY 2.12.1: Enter the output of the conversion expression.



Feedback?

CHALLENGE ACTIVITY 2.12.2: Type conversions.



Feedback?

How was this section? Provide section feedback

Activity summary for assignment: CSC108 CH02.11-2.24 C2B

Due: 02/06/2025, 11:59 PM EST

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.

51 / 51 points

51 / 51 points submitted to BlackboardLearn

Completion details