CS315

HOMEWORK 1 REPORT

FATİH SEVBAN UYANIK 21602486

PHP

Operator Precedence Rules in Arithmetic Expressions in PHP in Descending Order

1 → **()** → Parentheses
2 → ****** → Exponent
3 → ***, /, %*** → Multiplication, Division, Modulus
4 → **+, -** → Addition, Subtraction

Examples

Operation 1 → 10 - 50 + 80 **= 40**

```
Ordinary subtraction and addition.
$operation1 = 10 - 50 + 80;
```

Operation 2 → 10 - 50 * 10 **= -490**

```
showing that multiplication has precedence over subtraction. if subtraction had
precedence over multiplication, the expected result would be -400.
$operation2 = 10 - 50 * 10;
```

Operation 3 → 10 + 50 * 10 **= 510**

```
showing that multiplication has precedence over addition. if addition had
precedence over multiplication, the expected result would be 600.
$operation3 = 10 + 50 * 10;
```

Operation 4 → 10 - 50 / 10 **= 5**

```
showing that division has precedence over subtraction.if subtraction had precedence
over division, the expected result would be -4.
$operation4 = 10 - 50 / 10;
```

Operation 5 → 10 + 50 / 10 **= 15**

```
showing that division has precedence over  addition. if addition had precedence
over division, the expected result would be 6.
$operation5 = 10 + 50 / 10;
```

Operation 6 → 10 - 50 % 8 **= 8**

```
showing that Modulus has precedence over subtraction. if subtraction had precedence
over Modulus, the expected result would be 0.
$operation6 = 10 - 50 % 8;
```

Operation 7 → 10 + 50 % 8 **= 12**

```
showing that Modulus has precedence over addition. if addition had precedence over
Modulus, the expected result would be 4.
$operation7 = 10 + 50 % 8;
```

Operation 8 → 10 * 8 ** 2 **= 640**

```
showing that Exponent has precedence over Multiplication. if Multiplication had precedence over Exponent, the expected result would be 6400.
$operation8 = 10 * 8 ** 2;
```

Operation 9 → 100 / 2 ** 2 **= 25**

```
showing that Exponent has precedence over Division. if Division had precedence over Exponent, the expected result would be 2500
$operation9 = 100 / 2 ** 2;
```

Operation 10 → 51 % 2 ** 2 **= 3**

```
showing that Exponent has precedence over Modulus. if Modulus had precedence over Exponent, the expected result would be 1.
$operation10 = 51 % 2 ** 2;
```

Operation 11 → (100 / 10) ** 2 **= 100**

```
showing that Parentheses has precedence over Exponent. if the parentheses were not present, the expected result would be 1.0
$operation11 = (100 / 10) ** 2;
```

Operation 12 → (100 + 50) * 2 **= 300**

```
showing that Parentheses has precedence over Multiplication. if the parentheses were not present, the expected result would be 200
$operation12 = (100 + 50) * 2;
```

Operation 13 → (100 + 50) / 2 **= 75**

```
showing that Parentheses has precedence over Division. if the parentheses were not present, the expected result would be 125.0
$operation13 = (100 + 50) / 2;
```

Operation 14 → (200 + 395) % 10 **= 5**

```
showing that Parentheses has precedence over Modulus. if the parentheses were not present, the expected result would be 205.
$operation14 = (200 + 395) % 10;
```

Operator Associativity Rules In Arithmetic Expressions in PHP in Descending Order

1 → +, - → Addition, Subtraction have left associativity.
2 → *, /, % → Multiplication, Division, Modulus have left associativity.
3 → ** → Exponent have right associativity.
4 → () → Parentheses is non associative.

When two operators have the same precedence, operations are done according to associativity rules.

Examples

Operation 15 → 52 * 30 % 100 **= 60**

```
Showing that Multiplication and Modulus have left associativity. if multiplication and Modulus would be right associative, then the expected result would be 1560.
$operation15 = 52 * 30 % 100;
```

Operation 16 → 3300 / 30 % 50 **= 10**

```
Showing that Division and Modulus have left associativity. if Division and Modulus would be right associative, then the expected result would be 1560.
$operation16 = 3300 / 30 % 50;
```

Operation 17 → 3000 / 30 * 50 **= 5000**

```
Showing that Division and Multiplication have left associativity. if Division and Multiplication would be right associative, then the expected result would be 2.
$operation17 = 3000 / 30 * 50;
```

Operation 18 → 2 ** 3 ** 2 **= 512**

```
Showing that Exponent has right associativity. if Exponent had right left associativity, the expected result would be 64.
$operation18 = 2 ** 3 ** 2;
```

Order of Operand Evaluation in Arithmetic Expressions in PHP

Order of operand evaluation is always done from left to right in PHP.

Examples

testSum has worked. Result: 9
testSubtraction has worked. Result: 6
testMultiplication has worked. Result: 10
testDivision has worked. Result: 2
Operation19 (testSum(4, 5) * testSubtraction(25, 19) * testMultiplication(2, 5) * testDivision(10, 5)) = 1080

```
Showing that the operands are from left to right. in this context, the operator between the operands is multiplication
$operation19 = testSum(4, 5) * testSubtraction(25, 19) * testMultiplication(2, 5) * testDivision(10, 5);
```

testSum has worked. Result: 9
testSubtraction has worked. Result: 6
testMultiplication has worked. Result: 10
testDivision has worked. Result: 2
Operation20 (testSum(4, 5) + testSubtraction(25, 19) + testMultiplication(2, 5) + testDivision(10, 5)) = 27

```
Showing that the operands are from left to right. in this context, the operator between the operands is summation
$operation20 = testSum(4, 5) + testSubtraction(25, 19) + testMultiplication(2, 5) + testDivision(10, 5);
```

testSum has worked. Result: 2
testSubtraction has worked. Result: 3
Operation21 (testSum(1, 1) ** testSubtraction(25, 22)) **= 8**

```php
Showing that the operands are from left to right. in this context, the operator between the operands is exponent
$operation21 = testSum(1, 1) ** testSubtraction(25, 22);
```

Side Effects of Operand Evaluation in Arithmetic Expressions in PHP

RESULT1 (testFunctionSideEffectVersion1() + sideEffectVariable) = 900
RESULT2 (temp + sideEffectVariable) = 1700

if the function would not have side effects, then result1 = result2 should be equal.
However, because result1 is not equal to result2, side effect is observed in php.

RESULT3 (sideEffectVariable2 + testFunctionSideEffectVersion2() + sideEffectVariable2) = 1500

As it can be seen from result3, order of operand evaluation changes
because of side effect effect of functions.

PERL

Operator Precedence Rules in Arithmetic Expressions in Perl in Descending Order

1 → **()** → Parentheses
2 → ****** → Exponent
3 → ***, /, %** → Multiplication, Division, Modulus
4 → **+, -** → Addition, Subtraction

Examples

Operation 1 → 10 - 50 + 80 **= 40**

```
Ordinary subtraction and addition.
$operation1 = 10 - 50 + 80;
```

Operation 2 → 10 - 50 * 10 **= -490**

```
showing that multiplication has precedence over subtraction. if subtraction had
precedence over multiplication, the expected result would be -400.
$operation2 = 10 - 50 * 10;
```

Operation 3 → 10 + 50 * 10 **= 510**

```
showing that multiplication has precedence over addition. if addition had
precedence over multiplication, the expected result would be 600.
$operation3 = 10 + 50 * 10;
```

Operation 4 → 10 - 50 / 10 **= 5**

```
showing that division has precedence over subtraction.if subtraction had precedence
over division, the expected result would be -4.
$operation4 = 10 - 50 / 10;
```

Operation 5 → 10 + 50 / 10 **= 15**

```
showing that division has precedence over  addition. if addition had precedence
over division, the expected result would be 6.
$operation5 = 10 + 50 / 10;
```

Operation 6 → 10 - 50 % 8 **= 8**

```
showing that Modulus has precedence over subtraction. if subtraction had precedence
over Modulus, the expected result would be 0.
$operation6 = 10 - 50 % 8;
```

Operation 7 → 10 + 50 % 8 **= 12**

```
showing that Modulus has precedence over addition. if addition had precedence over
Modulus, the expected result would be 4.
$operation7 = 10 + 50 % 8;
```

Operation 8 → 10 * 8 ** 2 **= 640**

```
showing that Exponent has precedence over Multiplication. if Multiplication had
precedence over Exponent, the expected result would be 6400.
$operation8 = 10 * 8 ** 2;
```

Operation 9 → 100 / 2 ** 2 **= 25**

```
showing that Exponent has precedence over Division. if Division had precedence over
Exponent, the expected result would be 2500
$operation9 = 100 / 2 ** 2;
```

Operation 10 → 51 % 2 ** 2 **= 3**

```
showing that Exponent has precedence over Modulus. if Modulus had precedence over
Exponent, the expected result would be 1.
$operation10 = 51 % 2 ** 2;
```

Operation 11 → (100 / 10) ** 2 **= 100**

```
showing that Parentheses has precedence over Exponent. if the parentheses were not
present, the expected result would be 1.0
$operation11 = (100 / 10) ** 2;
```

Operation 12 → (100 + 50) * 2 **= 300**

```
showing that Parentheses has precedence over Multiplication. if the parentheses
were not present, the expected result would be 200
$operation12 = (100 + 50) * 2;
```

Operation 13 → (100 + 50) / 2 **= 75**

```
showing that Parentheses has precedence over Division. if the parentheses were not
present, the expected result would be 125.0
$operation13 = (100 + 50) / 2;
```

Operation 14 → (200 + 395) % 10 **= 5**

```
showing that Parentheses has precedence over Modulus. if the parentheses were not
present, the expected result would be 205.
$operation14 = (200 + 395) % 10;
```

Operator Associativity Rules In Arithmetic Expressions in Perl in Descending Order

1 → +, - → Addition, Subtraction have left associativity.
2 → *, /, % → Multiplication, Division, Modulus have left associativity.
3 → ** → Exponent have right associativity.
4 → () → Parentheses is non associative.

When two operators have the same precedence, operations are done according to associativity rules.

Examples

Operation 15 → 52 * 30 % 100 **= 60**

```
Showing that Multiplication and Modulus have left associativity. if multiplication and Modulus would be right associative, then the expected result would be 1560.
$operation15 = 52 * 30 % 100;
```

Operation 16 → 3300 / 30 % 50 **= 10**

```
Showing that Division and Modulus have left associativity. if Division and Modulus would be right associative, then the expected result would be 1560.
$operation16 = 3300 / 30 % 50;
```

Operation 17 → 3000 / 30 * 50 **= 5000**

```
Showing that Division and Multiplication have left associativity. if Division and Multiplication would be right associative, then the expected result would be 2.
$operation17 = 3000 / 30 * 50;
```

Operation 18 → 2 ** 3 ** 2 **= 512**

```
Showing that Exponent has right associativity. if Exponent had right left associativity, the expected result would be 64.
$operation18 = 2 ** 3 ** 2;
```

Order of Operand Evaluation in Arithmetic Expressions in PERL

Order of operand evaluation is always done from left to right in PERL.

testSum has worked. Result: 9
testSubtraction has worked. Result: 6
testMultiplication has worked. Result: 10
testDivision has worked. Result: 2
Operation19 (testSum(4, 5) * testSubtraction(25, 19) * testMultiplication(2, 5) * testDivision(10, 5)) = 1080

```
Showing that the operands are from left to right. in this context, the operator between the operands is multiplication
$operation19 = testSum(4, 5) * testSubtraction(25, 19) * testMultiplication(2, 5) * testDivision(10, 5);
```

testSum has worked. Result: 9
testSubtraction has worked. Result: 6
testMultiplication has worked. Result: 10
testDivision has worked. Result: 2
Operation20 (testSum(4, 5) + testSubtraction(25, 19) + testMultiplication(2, 5) + testDivision(10, 5)) = 27

```
Showing that the operands are from left to right. in this context, the operator between the operands is summation
$operation20 = testSum(4, 5) + testSubtraction(25, 19) + testMultiplication(2, 5) + testDivision(10, 5);
```

testSum has worked. Result: 2
testSubtraction has worked. Result: 3
Operation21 (testSum(1, 1) ** testSubtraction(25, 22)) **= 8**

```
Showing that the operands are from left to right. in this context, the operator
between the operands is exponent
$operation21 = testSum(1, 1) ** testSubtraction(25, 22);
```

Side Effects of Operand Evaluation in Arithmetic Expressions in PERL

RESULT1 (testFunctionSideEffectVersion1() + sideEffectVariable) = 900
RESULT2 (temp + sideEffectVariable) = 1700

if the function would not have side effects, then result1 = result2 should be equal.
However, because result1 is not equal to result2, side effect is observed in perl.

RESULT3 (sideEffectVariable2 + testFunctionSideEffectVersion2() + sideEffectVariable2) = 1500

As it can be seen from result3, order of operand evaluation changes
because of side effect effect of functions.

PYTHON

Operator Precedence Rules in Arithmetic Expressions in Python in Descending Order

1 → **()** → Parentheses
2 → ****** → Exponent
3 → ***, /, %** → Multiplication, Division, Modulus
4 → **+, -** → Addition, Subtraction

Examples

Operation 1 → 10 - 50 + 80 **= 40**

```
Ordinary subtraction and addition.
operation1 = 10 - 50 + 80
```

Operation 2 → 10 - 50 * 10 **= -490**

```
showing that multiplication has precedence over subtraction. if subtraction had
precedence over multiplication, the expected result would be -400.
operation2 = 10 - 50 * 10
```

Operation 3 → 10 + 50 * 10 **= 510**

```
showing that multiplication has precedence over addition. if addition had
precedence over multiplication, the expected result would be 600.
operation3 = 10 + 50 * 10
```

Operation 4 → 10 - 50 / 10 **= 5**

```
showing that division has precedence over subtraction.if subtraction had precedence
over division, the expected result would be -4.
operation4 = 10 - 50 / 10
```

Operation 5 → 10 + 50 / 10 **= 15**

```
showing that division has precedence over  addition. if addition had precedence
over division, the expected result would be 6.
operation5 = 10 + 50 / 10
```

Operation 6 → 10 - 50 % 8 **= 8**

```
showing that Modulus has precedence over subtraction. if subtraction had precedence
over Modulus, the expected result would be 0.
operation6 = 10 - 50 % 8
```

Operation 7 → 10 + 50 % 8 **= 12**

```
showing that Modulus has precedence over addition. if addition had precedence over
Modulus, the expected result would be 4.
operation7 = 10 + 50 % 8
```

Operation 8 → 10 * 8 ** 2 **= 640**

```
showing that Exponent has precedence over Multiplication. if Multiplication had precedence over Exponent, the expected result would be 6400.
operation8 = 10 * 8 ** 2
```

Operation 9 → 100 / 2 ** 2 **= 25**

```
showing that Exponent has precedence over Division. if Division had precedence over Exponent, the expected result would be 2500
operation9 = 100 / 2 ** 2
```

Operation 10 → 51 % 2 ** 2 **= 3**

```
showing that Exponent has precedence over Modulus. if Modulus had precedence over Exponent, the expected result would be 1.
operation10 = 51 % 2 ** 2
```

Operation 11 → (100 / 10) ** 2 **= 100**

```
showing that Parentheses has precedence over Exponent. if the parentheses were not present, the expected result would be 1.0
operation11 = (100 / 10) ** 2
```

Operation 12 → (100 + 50) * 2 **= 300**

```
showing that Parentheses has precedence over Multiplication. if the parentheses were not present, the expected result would be 200
operation12 = (100 + 50) * 2
```

Operation 13 → (100 + 50) / 2 **= 75**

```
showing that Parentheses has precedence over Division. if the parentheses were not present, the expected result would be 125.0
operation13 = (100 + 50) / 2
```

Operation 14 → (200 + 395) % 10 **= 5**

```
showing that Parentheses has precedence over Modulus. if the parentheses were not present, the expected result would be 205.
operation14 =  (200 + 395) % 10
```

Operator Associativity Rules In Arithmetic Expression in Python in Descending Order

1 → +, - → Addition, Subtraction have left associativity.
2 → *, /, % → Multiplication, Division, Modulus have left associativity.
3 → ** → Exponent have right associativity.
4 → () → Parentheses is non associative.

When two operators have the same precedence, operations are done according to associativity rules.

Examples

Operation 15 → 52 * 30 % 100 **= 60**

```
Showing that Multiplication and Modulus have left associativity. if multiplication and Modulus would be right associative, then the expected result would be 1560.
operation15 = 52 * 30 % 100
```

Operation 16 → 3300 / 30 % 50 **= 10**

```
Showing that Division and Modulus have left associativity. if Division and Modulus would be right associative, then the expected result would be 1560.
operation16 = 3300 / 30 % 50
```

Operation 17 → 3000 / 30 * 50 **= 5000**

```
Showing that Division and Multiplication have left associativity. if Division and Multiplication would be right associative, then the expected result would be 2.
operation17 = 3000 / 30 * 50
```

Operation 18 → 2 ** 3 ** 2 **= 512**

```
Showing that Exponent has right associativity. if Exponent had right left associativity, the expected result would be 64.
operation18= 2 ** 3 ** 2
```

Order of Operand Evaluation in Arithmetic Expressions in Python

Order of operand evaluation is always done from left to right in Python.

testSum has worked. Result: 9
testSubtraction has worked. Result: 6
testMultiplication has worked. Result: 10
testDivision has worked. Result: 2
Operation19 (testSum(4, 5) * testSubtraction(25, 19) * testMultiplication(2, 5) * testDivision(10, 5)) = 1080

```
Showing that the operands are from left to right. in this context, the operator between the operands is multiplication
operation19 = testSum(4, 5) * testSubtraction(25, 19) * testMultiplication(2, 5) * testDivision(10, 5)
```

testSum has worked. Result: 9
testSubtraction has worked. Result: 6
testMultiplication has worked. Result: 10
testDivision has worked. Result: 2
Operation20 (testSum(4, 5) + testSubtraction(25, 19) + testMultiplication(2, 5) + testDivision(10, 5)) = 27

```
Showing that the operands are from left to right. in this context, the operator between the operands is summation
operation20 = testSum(4, 5) + testSubtraction(25, 19) + testMultiplication(2, 5) + testDivision(10, 5)
```

testSum has worked. Result: 2
testSubtraction has worked. Result: 3
Operation21 (testSum(1, 1) ** testSubtraction(25, 22)) **= 8**

```
Showing that the operands are from left to right. in this context, the operator
between the operands is exponent
operation21 = testSum(1, 1) ** testSubtraction(25, 22)
```

Side Effects of Operand Evaluation in Arithmetic Expressions in Python

RESULT1 (testFunctionSideEffectVersion1() + sideEffectVariable) = 900
RESULT2 (temp + sideEffectVariable) = 1700

if the function would not have side effects, then result1 = result2 should be equal. However, because result1 is not equal to result2, side effect is observed in python.

RESULT3 (sideEffectVariable2 + testFunctionSideEffectVersion2() + sideEffectVariable2) = 1075

As it can be seen from result3, order of operand evaluation does not change because of side effect effect of functions. It is from left to right as always.

JavaScript

Operator Precedence Rules in Arithmetic Expressions in JavaScript in Descending Order

1 → **()** → Parentheses
2 → ****** → Exponent
3 → ***, /, %** → Multiplication, Division, Modulus
4 → **+, -** → Addition, Subtraction

Examples

Operation 1 → 10 - 50 + 80 **= 40**

```
Ordinary subtraction and addition.
const operation1 = 10 - 50 + 80
```

Operation 2 → 10 - 50 * 10 **= -490**

```
showing that multiplication has precedence over subtraction. if subtraction had
precedence over multiplication, the expected result would be -400.
const operation2 = 10 - 50 * 10
```

Operation 3 → 10 + 50 * 10 **= 510**

```
showing that multiplication has precedence over addition. if addition had
precedence over multiplication, the expected result would be 600.
const operation3 = 10 + 50 * 10
```

Operation 4 → 10 - 50 / 10 **= 5**

```
showing that division has precedence over subtraction.if subtraction had precedence
over division, the expected result would be -4.
const operation4 = 10 - 50 / 10
```

Operation 5 → 10 + 50 / 10 **= 15**

```
showing that division has precedence over  addition. if addition had precedence
over division, the expected result would be 6.
const operation5 = 10 + 50 / 10
```

Operation 6 → 10 - 50 % 8 **= 8**

```
showing that Modulus has precedence over subtraction. if subtraction had precedence
over Modulus, the expected result would be 0.
const operation6 = 10 - 50 % 8
```

Operation 7 → 10 + 50 % 8 **= 12**

```
showing that Modulus has precedence over addition. if addition had precedence over
Modulus, the expected result would be 4.
const operation7 = 10 + 50 % 8
```

Operation 8 → 10 * 8 ** 2 **= 640**

```
showing that Exponent has precedence over Multiplication. if Multiplication had precedence over Exponent, the expected result would be 6400.
const operation8 = 10 * 8 ** 2
```

Operation 9 → 100 / 2 ** 2 **= 25**

```
showing that Exponent has precedence over Division. if Division had precedence over Exponent, the expected result would be 2500
const operation9 = 100 / 2 ** 2
```

Operation 10 → 51 % 2 ** 2 **= 3**

```
showing that Exponent has precedence over Modulus. if Modulus had precedence over Exponent, the expected result would be 1.
const operation10 = 51 % 2 ** 2
```

Operation 11 → (100 / 10) ** 2 **= 100**

```
showing that Parentheses has precedence over Exponent. if the parentheses were not present, the expected result would be 1.0
const operation11 = (100 / 10) ** 2
```

Operation 12 → (100 + 50) * 2 **= 300**

```
showing that Parentheses has precedence over Multiplication. if the parentheses were not present, the expected result would be 200
const operation12 = (100 + 50) * 2
```

Operation 13 → (100 + 50) / 2 **= 75**

```
showing that Parentheses has precedence over Division. if the parentheses were not present, the expected result would be 125.0
const operation13 = (100 + 50) / 2
```

Operation 14 → (200 + 395) % 10 **= 5**

```
showing that Parentheses has precedence over Modulus. if the parentheses were not present, the expected result would be 205.
const operation14 = (200 + 395) % 10
```

Operator Associativity Rules In Arithmetic Expression in JavaScript in Descending Order

1 → +, - → Addition, Subtraction have left associativity.
2 → *, /, % → Multiplication, Division, Modulus have left associativity.
3 → ** → Exponent have right associativity.
4 → () → Parentheses is non associative.

When two operators have the same precedence, operations are done according to associativity rules.

Examples

Operation 15 → 52 * 30 % 100 **= 60**

```
Showing that Multiplication and Modulus have left associativity. if multiplication and Modulus would be right associative, then the expected result would be 1560.
const operation15 = 52 * 30 % 100
```

Operation 16 → 3300 / 30 % 50 **= 10**

```
Showing that Division and Modulus have left associativity. if Division and Modulus would be right associative, then the expected result would be 1560.
const operation16 = 3300 / 30 % 50
```

Operation 17 → 3000 / 30 * 50 **= 5000**

```
Showing that Division and Multiplication have left associativity. if Division and Multiplication would be right associative, then the expected result would be 2.
const operation17 = 3000 / 30 * 50
```

Operation 18 → 2 ** 3 ** 2 **= 512**

```
Showing that Exponent has right associativity. if Exponent had right left associativity, the expected result would be 64.
const operation18 = 2 ** 3 ** 2
```

Order of Operand Evaluation in Arithmetic Expressions in JavaScript

Order of operand evaluation is always done from left to right in JavaScript.

testSum has worked. Result: 9
testSubtraction has worked. Result: 6
testMultiplication has worked. Result: 10
testDivision has worked. Result: 2
Operation19 (testSum(4, 5) * testSubtraction(25, 19) * testMultiplication(2, 5) * testDivision(10, 5)) = 1080

```
Showing that the operands are from left to right. in this context, the operator between the operands is multiplication
const operation19 = testSum(4, 5) * testSubtraction(25, 19) * testMultiplication(2, 5) * testDivision(10, 5);
```

testSum has worked. Result: 9
testSubtraction has worked. Result: 6
testMultiplication has worked. Result: 10
testDivision has worked. Result: 2
Operation20 (testSum(4, 5) + testSubtraction(25, 19) + testMultiplication(2, 5) + testDivision(10, 5)) = 27

```
Showing that the operands are from left to right. in this context, the operator between the operands is summation
const operation20 = testSum(4, 5) + testSubtraction(25, 19) + testMultiplication(2, 5) + testDivision(10, 5);
```

testSum has worked. Result: 2
testSubtraction has worked. Result: 3
Operation21 (testSum(1, 1) ** testSubtraction(25, 22)) **= 8**

```
Showing that the operands are from left to right. in this context, the operator
between the operands is exponent
const operation21 = testSum(1, 1) ** testSubtraction(25, 22);
```

Side Effects of Operand Evaluation in Arithmetic Expressions in JavaScript

RESULT1 (testFunctionSideEffectVersion1() + sideEffectVariable) = 900
RESULT2 (temp + sideEffectVariable) = 1700

if the function would not have side effects, then result1 = result2 should be equal. However, because result1 is not equal to result2, side effect is observed in JavaScript.

RESULT3 (sideEffectVariable2 + testFunctionSideEffectVersion2() + sideEffectVariable2) = 1075

As it can be seen from result3, order of operand evaluation does not change because of side effect effect of functions. It is from left to right as always.

C

Operator Precedence Rules in Arithmetic Expressions in C in Descending Order

1 → **()** → Parentheses
2 → ****** → Exponent
3 → ***, /, %** → Multiplication, Division, Modulus
4 → **+, -** → Addition, Subtraction

Examples

Operation 1 → 10 - 50 + 80 **= 40**

```
Ordinary subtraction and addition.
int operation1 = 10 - 50 + 80;
```

Operation 2 → 10 - 50 * 10 **= -490**

```
showing that multiplication has precedence over subtraction. if subtraction had
precedence over multiplication, the expected result would be -400.
int operation2 = 10 - 50 * 10;
```

Operation 3 → 10 + 50 * 10 **= 510**

```
showing that multiplication has precedence over addition. if addition had
precedence over multiplication, the expected result would be 600.
int operation3 = 10 + 50 * 10;
```

Operation 4 → 10 - 50 / 10 **= 5**

```
showing that division has precedence over subtraction.if subtraction had precedence
over division, the expected result would be -4.
int operation4 = 10 - 50 / 10;
```

Operation 5 → 10 + 50 / 10 **= 15**

```
showing that division has precedence over  addition. if addition had precedence
over division, the expected result would be 6.
int operation5 = 10 + 50 / 10;
```

Operation 6 → 10 - 50 % 8 **= 8**

```
showing that Modulus has precedence over subtraction. if subtraction had precedence
over Modulus, the expected result would be 0.
int operation6 = 10 - 50 % 8;
```

Operation 7 → 10 + 50 % 8 **= 12**

```
showing that Modulus has precedence over addition. if addition had precedence over
Modulus, the expected result would be 4.
int operation7 = 10 + 50 % 8;
```

Operation 8 → (100 + 50) * 2 **= 300**

```
showing that Parentheses has precedence over Multiplication. if the parentheses were not present, the expected result would be 200
int operation8 = (100 + 50) * 2;
```

Operation 9 → (100 + 50) / 2 **= 75**

```
showing that Parentheses has precedence over Division. if the parentheses were not present, the expected result would be 125.0
int operation9 = (100 + 50) / 2;
```

Operation 10 → (200 + 395) % 10 **= 5**

```
showing that Parentheses has precedence over Modulus. if the parentheses were not present, the expected result would be 205.
int operation10  = (200 + 395) % 10;
```

Operator Associativity Rules In Arithmetic Expression in C in Descending Order

1 → +, - → Addition, Subtraction have left associativity.
2 → *, /, % → Multiplication, Division, Modulus have left associativity.
3 → ** → Exponent have right associativity.
4 → () → Parentheses is non associative.

When two operators have the same precedence, operations are done according to associativity rules.

Examples

Operation 11 → 52 * 30 % 100 **= 60**

```
Showing that Multiplication and Modulus have left associativity. if multiplication and Modulus would be right associative, then the expected result would be 1560.
int operation11 = 52 * 30 % 100;
```

Operation 12 → 3300 / 30 % 50 **= 10**

```
Showing that Division and Modulus have left associativity. if Division and Modulus would be right associative, then the expected result would be 1560.
int operation12 = 3300 / 30 % 50;
```

Operation 13 → 3000 / 30 * 50 **= 5000**

```
Showing that Division and Multiplication have left associativity. if Division and Multiplication would be right associative, then the expected result would be 2.
int operation13 = 3000 / 30 * 50;
```

Order of Operand Evaluation in Arithmetic Expressions in JavaScript

Order of operand evaluation is always done from left to right in JavaScript.

testSum has worked. Result: 9
testSubtraction has worked. Result: 6
testMultiplication has worked. Result: 10
testDivision has worked. Result: 2
Operation14 (testSum(4, 5) * testSubtraction(25, 19) * testMultiplication(2, 5) * testDivision(10, 5)) = 1080

```
Showing that the operands are from left to right. in this context, the operator between the operands is multiplication
int operation14 = testSum(4, 5) * testSubtraction(25, 19) * testMultiplication(2, 5) * testDivision(10, 5);
```

testSum has worked. Result: 9
testSubtraction has worked. Result: 6
testMultiplication has worked. Result: 10
testDivision has worked. Result: 2
Operation15 (testSum(4, 5) + testSubtraction(25, 19) + testMultiplication(2, 5) + testDivision(10, 5)) = 27

```
Showing that the operands are from left to right. in this context, the operator between the operands is summation
int operation15 = testSum(4, 5) + testSubtraction(25, 19) + testMultiplication(2, 5) + testDivision(10, 5);  // Expected 27
```

Side Effects of Operand Evaluation in Arithmetic Expressions in C

RESULT1 (testFunctionSideEffectVersion1() + sideEffectVariable) = 900
RESULT2 (temp + sideEffectVariable) = 1700

if the function would not have side effects, then result1 = result2 should be equal.
However, because result1 is not equal to result2, side effect is observed in C.

RESULT3 (sideEffectVariable2 + testFunctionSideEffectVersion2() + sideEffectVariable2) = 1500

As it can be seen from result3, order of operand evaluation changes
because of side effect effect of functions.