Lab: Arithmetic Operators

Tutorial Lab 1: Arithmetic Operators

Arithmetic operations in C++ are mostly the same as what you learned in math class. However, the symbols used in C++ may be different.

Operation	Symbol	Notes
Addition	+	
Subtraction	-	
Multiplication	*	
Division	/	
Modulo	%	Returns the remainder after division is performed.

Use the text editor on the left to enter the following code:

```
cout << 10 + 3 << endl;   -> 13
cout << 10 - 3 << endl;   -> 7
cout << 10 * 3 << endl;   -> 30
cout << 10 / 3 << endl;   -> 3
cout << 10 % 3 << endl;   -> 1
```

Program results:
1) Addition works as expected.
2) Subtraction works as expected.
3) Multiplication works as expected.
4) Division with integers will return a truncated integer result.
5) Modulo returns 1 because that is the remainder (not the decimal) after division is performed.

Lab: Strings

Tutorial Lab 2: Strings

You can use the + operator with strings, even though the result is not based on math. Using the + operator with strings is called *concatenation*.

Use the text editor on the left to enter the following code:

```
string string1 = "hip ";
string string2 = string1 + string1;
string string3 = "hoo";
string string4 = "ray!";
string string5 = string3 + string4;
cout << string2;
cout << string5 << endl;
```

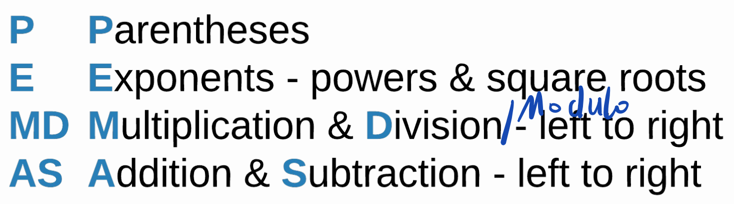
$\rightarrow$ hip hip hooray

Below are the steps that C++ takes when evaluating the code above.
1) Assign the value `"hip "` to the variable `string1`. Note the inclusion of a space after the word `hip`.
2) The variable `string2` will have the value of `"hip hip "` because `string1 + string1` repeats the value of `string1` two times.
3) Declare `string3` and assign it the value of `"hoo"`.
4) Declare `string4` and assign it the value of `"ray!"`.
5) Declare `string5` and assign it the value of `string3` combined with the value of `string4` (`"hooray!"`).
6) Print the value of `string2` (`"hip hip "`) without the newline character.
7) Print the value of `string5` (`"hooray!"`) to the end of `string2`.

Lab: Order of Operations

Tutorial Lab 3: Order of Operations

C++ uses PEMDAS when determining the order of operations.



P	**P**arentheses
E	**E**xponents - powers & square roots
MD	**M**ultiplication & **D**ivision/Modulo - left to right
AS	**A**ddition & **S**ubtraction - left to right

.guides/img/PEMDAS

▼ Modulo and PEMDAS

Since modulo is based on division, modulo operations happen at the time of multiplication and division, going from left to right.

Use the text editor on the left to enter the following code:

$\overbrace{40/3}^{13}$ $\overset{0}{10\%2}$

```
cout << (5 * 8 / 3 + (18 - 8) % 2 - 25) << endl;
```

$\rightarrow -12$

Below are the steps that C++ takes when evaluating the code above.

1) `5 * 8 / 3 + (18 - 8) % 2 - 25`
1) `5 * 8 / 3 + 10 % 2 - 25`
1) `40 / 3 + 10 % 2 - 25`
1) `13 + 10 % 2 - 25`
1) `13 + 0 - 25`
1) `13 - 25`
1) $\underline{\texttt{-12}}$

Lab: Boolean Operators

Tutorial Lab 4: Boolean Operators

Boolean operators are used within expressions to return either `true` or `false`.

Operation	Symbol	Notes
Equal to	`==`	The = operator is the assignment operator, not the equality operator.
Not equal to	`!=`	
Less than	`<`	
Less than or equal to	`<=`	
Greater than	`>`	
Greater than or equal to	`>=`	
And	`&&`	Compares two boolean expressions. Both must be true for the whole to be true. Everything else is false.
Or	`||`	Compares two boolean expressions. Both must be false for the whole to be false. Everything else is true.
Not	`!`	Returns the opposite result of an evaluated boolean expression.

The following code is available within the text editor on the left. Click the `TRY IT` button below to see the printed result.

false   true

```
cout << boolalpha << ((5 > 7) && (false || 1 < 9) || 4 != 5 && !
        (2 >= 3)) << endl;
```

$\longrightarrow$ true

Below are the steps that C++ takes when evaluating the code above.

Evaluate all arithmetic operators according to PEMDAS

1. `(5 > 7) && (false || 1 < 9) || 4 != 5 && ! (2 >= 3)`
2. **`false`** `&& (false || 1 < 9) || 4 != 5 && ! (2 >= 3)`
3. `false && (false ||` **`true`**`) || 4 != 5 && ! (2 >= 3)`
4. `false && (false || true) ||` **`true`** `&& ! (2 >= 3)`
5. `false && (false || true) || true && !` **`false`**

Evaluate all boolean operators according to this order - Parentheses (`()`), Not (`!`), And (`&&`), then Or (`||`)

6. `false &&` **`true`** `|| true && ! false`
7. `false && true || true &&` **`true`**
8. **`false`** `|| true && true`
9. `false ||` **`true`**
10. **`true`**

==**Note**== that **arithmetic** operators are performed *before* **boolean** operators.

Lab Challenge: Operators

Operators Challenge

Write a boolean expression that incorporates *ONE* of the **equality operators**, *ONE* of the **less than** operators, *ONE* of the **greater than** operators, and *TWO* of the **logical** operators. The result of your overall boolean expression *MUST* be `false`. Make sure to use `cout << boolalpha <<` in your code. Otherwise, the system will print `0` or `1` instead of `false` or `true`.

Equality	Less Than	Greater Than	Logical
`==`	`<`	`>`	`&&`
`!=`	`<=`	`>=`	`||`
			`!`