Chapter 4
Operators & Arithmetic
Arithmetic in C++

Arithmetic expressions can be made up of constants, variables, operators and parentheses. The *arithmetic operators* in C++ are as follows:

- `+` (addition)
- `-` (subtraction)
- `*` (multiplication)
- `/` (division)
- `%` (modulus - the remainder from integer division)

**NOTE:** The `%` operator may appear ONLY with integer values.

When expressions are evaluated, they are evaluated left to right according to the following *precedence rules*:

```
( )  
* / %
+ -
```

The expression `4 + 8 / 2` is evaluated as follows:

```
4 + 8 / 2 =  (division has the highest precedence)
4 + 4 = 8
```

The expression `(4 + 8) / 2` is evaluated as follows:

```
(4 + 8) / 2 =  (parentheses have the highest precedence)
12 / 2 = 6
```

The expression `8 % 3` is evaluated as follows:

```
8 % 3 = 2  (the whole number remainder)
```

The expression `8 % 3.0` would be invalid (3.0 is a floating point value and therefore not valid), as would the expression `8.0 % 3`.

4-2
Evaluate the following expressions:

1) \( \frac{12}{3} \times 3 = \)

2) \( 10 \mod 3 - \frac{6}{2} = \)

3) \( 5.0 \times 2.0 / 4.0 \times 2.0 = \)

4) \( 5.0 \times 2.0 / (4.0 \times 2.0) = \)

5) \( 5.0 + 2.0 / (4.0 \times 2.0) \)
Mixed Mode Arithmetic

Recall that we are using two different data types to represent numeric values, int and float. These two data types require different amounts of memory and the values are stored in different manners. In other words, the integer 6 is stored differently than the floating point value 6.0. When we combine integer and floating point values in the same expression this is called *mixed mode* arithmetic.

One method that may be used to place a value into a variable is via the *assignment expression*. The general form for an assignment expression is as follows:

An *assignment expression* is an expression with a value that has the side effect of storing that value into a variable.

\[
\text{variable} = \text{expression}
\]

When a semi-colon is added to the *assignment expression*, it becomes an *expression statement*. In CS1A we called this an assignment statement.

\[
\text{variable} = \text{expression};
\]

The following are valid C++ *expression statements*:

\[
2 \times 5 / 4 + \text{num}; \quad \text{// valid - does nothing}
52; \quad \text{// valid - does nothing}
\text{num2} = \text{num} + 10; \quad \text{// valid - stores the expression value in num2}
\]
Given the declarations
    int num1, num2;
    float num3, average;

num1 and num2 may hold integer values only and average will hold a floating point value. What happens if the following assignment statements appear in a program?
    num1 = 3;
    num2 = 7.75;
    num3 = 5;
    average = (num1 + num2) / 2.0;

The value 3 is a valid integer and is stored in location num1. The value 7.75 is a floating point value so the compiler simply truncates (cuts off) the fractional part and stores the integer value 7 in location num2. **NOTE: The value is not rounded.** The location num3 is a float data type so the integer value 5 is converted to 5.0 and stored. The calculation for the average adds the two integer values 3 and 7 and divides the result by the floating point value 2.0. The result is the floating point value 5.0. This automatic conversion performed by the compiler is referred to as *type coercion.*

Consider the following:
    num1 = 3;
    num2 = 7.75;
    average = (num1 + num2) / 20;

The result is the integer value 0. The integer values 3 and 7 are added then divided by the integer value 20. The integer quotient is 0 and is stored in the variable average. To obtain the desired value the expression must contain a floating point value. Write the expression as follows

    average = (num1 + num2) / 20.0;

to store the floating point value 0.5 in the variable average. **Be very careful with mixed mode arithmetic.**
Assignment Expression Usage

Since an assignment is really an expression and not a statement, it may be used anywhere an expression may be used. Evaluate the following assignment expressions.

```cpp
int n1,n2;
float n3,n4;

1.   n1 = ( n3 = (n2 = 5) * 4 / 8.0 ) * 2;
    cout << n1 << endl << n2 << endl << n3 << endl;

2.   n1 = ( n3 = (n2 = 5) * 4 / 8 ) * 2;
    cout << n1 << endl << n2 << endl << n3 << endl;

3.   n1 = ( n3 = (n2 = 5) * (4 / 8.0) ) * 2;
    cout << n1 << endl << n2 << endl << n3 << endl;

4.   n1 = ( n3 = (n2 = 5) * (4 / 8) ) * 2;
    cout << n1 << endl << n2 << endl << n3 << endl;
```
5.  
\[
n1 = ( n3 = (n2 = 5) \times \frac{4}{8.0}) \times 2;
\]
\[
\text{if}( (n4 = (n1 = (n2 \times 2) + n3)) > 10 )
\{
    \text{cout} << n4;
\}
\text{else}
\{
    \text{cout} << "Test val is 10 or less ";
\}
\]

6.  
\[
n1 = ( n3 = (n2 = 5) \times \frac{4}{8}) \times 2;
\]
\[
\text{if}( (n4 = (n1 = (n2 \times 2) + n3)) > 10 )
\{
    \text{cout} << n4;
\}
\text{else}
\{
    \text{cout} << "Test val is 10 or less ";
\}
\]

Although this style of programming can be used, many programmers find it to be confusing and error-prone. We will avoid this type of programming in CS1B.
Exercises in C++ Arithmetic Expressions

What value is stored into the integer variable num after each of the following expressions has been evaluated?

1) num = 17 % 3;

2) num = 8 / 3 + 2;

3) num = 6.0 / 12.0 + 5.25;

4) num = 3 + 5 * 2;

For each of the following indicate whether the expression is valid or invalid. If the expression is valid, indicate whether the result is integer or floating point.

1) 10.0 / 3.0 + 5 * 2

2) 10 / 4 + 6 / 3

3) 10 % 4 + 6 % 3

4) (10.0 / 3.0 % 2) / 3

5) 13.25 + (5.0 / (3.0 / 3.5))

6) -4 * (-5 + 6)
Increment & Decrement Operators

In addition to the math operators, C++ provides *increment* and *decrement* operators.

++  Increment
--  Decrement

These are unary operators (an operator that operates on only one operand) that use a single variable as an operand. The effect is to add 1 to or subtract 1 from integer and floating point values.

Example: The value 20 is currently stored in a variable called age. The statement

age++;

would cause the variable age to become 21. This is logically equivalent to the statement

age = age + 1;

You will typically see programs that use the increment and decrement operators.

These operators may be either *prefix* operators or *postfix* operators.

prefix  +++age;
postfix  age++;

Both statements would produce the same result.
A word to the wise, be VERY CAREFUL using these operators. C++ allows these operators to be used in the middle of a larger expression. Suppose num contains the value 3, the statement

someNum = num++ * 4;

yields a different result than the statement

someNum = ++num * 4;

What do you think the two results are?

<table>
<thead>
<tr>
<th>Increment and Decrement Operators Used in Expressions</th>
</tr>
</thead>
<tbody>
<tr>
<td>++ prefix     ++n</td>
</tr>
<tr>
<td>++ postfix    n++</td>
</tr>
<tr>
<td>-- prefix     --n</td>
</tr>
<tr>
<td>-- postfix    n--</td>
</tr>
</tbody>
</table>
Given:
   cout << (num2 = 10);

Two tasks are performed. First, the value 10 is assigned to the variable num2 and then it displays the contents of variable num2 on the screen.

Given:
   num1 = num2 = num3 = num4 = 0;

This is called multiple assignment and may be used to set several variables to the same value.

Precedence order (arithmetic, relational, and assignment operators)

* /  %
+ -
< <=  > >=
==    !=
  =
Assume num2 = 3:

\[ \text{num3} = \text{num2} + 5 \times (\text{num1} = 7); \]

1. The embedded assignment statement is handled => num1 is assigned the value 7.
   \[ \text{num3} = \text{num2} + 5 \times 7 \]
2. num3 = num2 + 35
3. num3 = 38

Two assignment statements were made here, the value 7 was stored in num1 and the value 38 was stored in num2.

Rewrite the following in a more readable style:

\[ \text{if}(x = y < z) \]

Comment on the following:

\[ \text{if}(x = y < z) \]

IT IS NOT RECOMMENDED THAT YOU PROGRAM THIS WAY - IMAGINE HOW CONFUSING YOUR EXPRESSIONS COULD BECOME.
Combined Operators

C++ allows operators to be combined for a short-hand notation. Many environments discourage the use of operators in this way as it can make a program more difficult to read and understand.

```
+=    value = value + 5;
    value += 5;

-=    value = value - 3;
    value -= 3;

*=
    value = value * 10;
    value *= 10;

%=
    value = value % 2;
    value %=
```

```
/=    value = value / 2;
    value /= 2;

Given:
    num3 *= num1 + 10;

Rewrite the above statement
    num3 = num3 * (num1 + 10);

NOTE: The precedence of the combined assignment operators is lower than that of the regular math operators.
Write statements using combined assignment operators to perform the following:

a) Subtract 5 from n1

b) Add n1 \times 8 to n2

c) Store in n3 the remainder of n3 divided by 5