Arithmetic Operators

Today

- Section 3.1 is required reading and is not explicitly covered in the lecture
- Arithmetic Operators & Expressions
  - Sections 2.15 & 3.2
  - Computation
  - Precedence
  - Algebra vs C++
  - Exponents

Assigning `float`s to `int`s

- Look at the following situation.

```
int intVariable;
intVariable = 42.7;
cout << intVariable;
```

- Q.1. What do you think is the output?

Assigning floats to ints

- Q.2. What is the output here?

```
int intVariable;
double doubleVariable 78.9;
intVariable = doubleVariable;
cout << intVariable;
```


Arithmetic Operators

- Operators allow us to manipulate data
  - Unary: `operator operand`
  - Binary: `operand operator operand` (left hand side) (right hand side)

Operator	Meaning	Type	Example
-	Negation	Unary	`- 5`
=	Assignment	Binary	`rate = 0.05`
*	Multiplication	Binary	`cost * rate`
/	Division	Binary	`cost / 2`
%	Modulus	Binary	`cost % 2`
+	Addition	Binary	`cost + tax`
-	Subtraction	Binary	`total - tax`

Integer Division

- Q.3. What is the output?
  - ```
    int grade;
    grade = 100 / 20;
    cout << grade;
    ```
  - ```
    int grade;
    grade = 100 / 30;
    cout << grade;
    ```

Division

- `grade = 100 / 40; grade` is 2
  - If both operands of the division operator are integers, then integer division is performed.
    - the data type of grade is not considered, why?
  - We say the integer is *truncated*. Everything after the decimal point is dropped. No rounding.

Non-Integer Division

- ```
  grade = 100.0 / 40;
  cout << grade;
  ```
- Q.4. What is the output?
- Q.5. What data type should grade be declared as?

Modulus

- Modulus is the remainder after integer division
- Q.6. What is the value of grade after each of these statements?
  - `grade = 100 % 20;`
  - `grade = 100 % 30;`
  ```

The General Case

- `rem = x % n;`
  - Q.7. What are the possible values for `rem`?

Practice

- Q.8. What value is assigned to `x` after each of these statements is executed?
  a. x = 8 + 3;
  b. x = 8 - 3;
  c. x = 8 * 3;

Practice

  d. x = 8 % 3;
  e. x = 8 / 3;

Mathematical Expressions (3.2)

- Complex mathematical expressions are created by using multiple operators and grouping symbols
  - expression: programming statement that has value
  - `sum = 21 + 3;`

    expression

    In these two examples, we assign the value of an *expression* to a variable
  - `number = 3;`

Examples

- `result = x;`
- `result = 4 + result;`
- `result = 15 / 3;`
- `result = 22 * number;`

Examples

- `result = a + b % c;`
- `result = a + b + d / c - q + 42;`
- `cout << "The value: " << (sum / 2) << endl;`

Operator Precedence

- `result = a + b + d;`
- `result = 12 + 6 / 3;`
  - `result = ?`
- Rules on how to evaluate an arithmetic expression
  - arithmetic expressions are evaluated left to right
  - when there are two operators, do them in order of precedence

Operator Precedence

Precedence of Arithmetic Operators (Highest to Lowest)
(unary negation) -
* / %
+ -

If two operators have the same precedence, evaluate them from left to right as they appear in the expression

Q.9. Practice

a. 5 + 2 * 3

b. 10 / 2 -1

c. 3 + 12 * 2 - 3

Practice

d. 4 + 17 % 3 + 9

e. 6 - 2 * 9 / 3 * 4 - 9

Summary

- Today we have looked at:
  - Arithmetic Operators & Expressions
- Next time we will:
  - Continue looking at mathematic operators
- Completed section 2.15 & started on section 3.2