# Review

Introduction to Operators & Arithmetic

# Halstead’s Theory

- Maurice Halstead’s Theory (1971~1979):
  - A program *P* is a collection of tokens, composed of two basic elements: ***operands*** and ***operators***
  - ***Operands*** are variables, constants, addresses
  - ***Operators*** are defined operations in a programming language (language constructs)

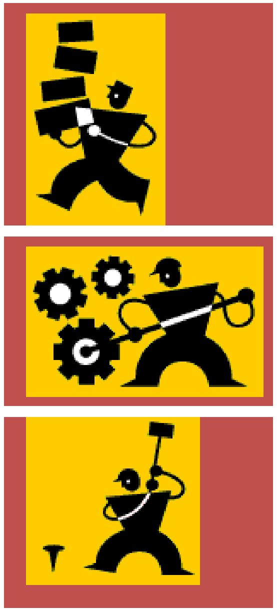
if … else
+ - > = ;
main()
goto

a, b, x
100

Based on Halstead’s theory:
operators and operands can be used to estimate the size of a program

# Operators

- Java (like C++) is language with many operator.
- There are two groups of operators: Unary and Binary Operators.
  - Unary operators are those that need one operand: such as plus (+) and minus (-) signs.
  - Binary operators are those that need two operands: such as addition (+), subtraction (-), multiplication (*), …
- A subset of operators can be categorized as:
  - Arithmetic Operators
  - Increment and Decrement Operators
  - Relational and Logical Operators
  - Bit-wise Operators // not discussed in ENGG 233
- Let's review some of the operators.



# Operators Precedence

# Operators Precedence

- Operators are executed based on predefined precedence. → See next slide's Table
- Higher precedence means "earlier execution"
- Operators have the same precedence as other operators in their group, and higher precedence than operators in lower groups
- If not sure, always use ( ) to force your preferred precedence

# Operators Precedence /1

- Operators have the same precedence as other operators in their group, and higher precedence than operators in lower groups

| | | |
|---|---|---|
| **Level 1** | **()** | Function call |
| | **++** | Post-increment |
| | **--** | Post-decrement |
| **Level 2** | **!** | Logical NOT |
| | **++** | Pre-increment |
| | **--** | Pre-decrement |
| | **-** | Unary minus |
| | **+** | Unary plus |
| | **(type)** | Cast to a given type |
| | **sizeof** | Return size of an object |

# Operators Precedence /2

| | | |
|---|---|---|
| **Level 3** | ***** | Multiplication |
| | **/** | Division |
| | **%** | Modulus |
| **Level 4** | **+** | Addition |
| | **-** | Subtraction |
| **Level 5** | **<** | Comparison less-than |
| | **<=** | Comparison less-than-or-equal-to |
| | **>** | Comparison greater-than |
| | **>=** | Comparison greater-than-or-equal-to |
| **Level 6** | **==** | Comparison equal-to |
| | **!=** | Comparison not-equal-to |

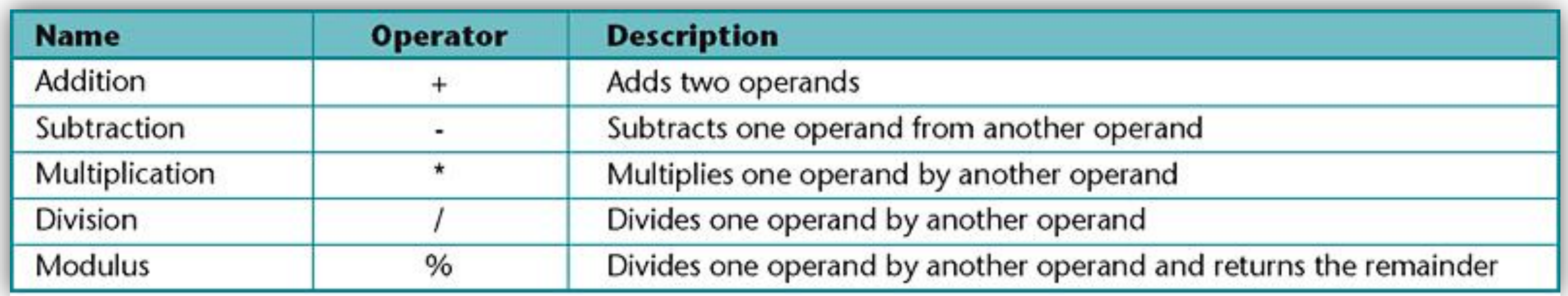
# Operators Precedence /3

| | | |
|---|---|---|
| **Level 7** | **&&** | Logical AND |
| **Level 8** | **\|\|** | Logical OR |
| **Level 9** | **? :** | Ternary conditional (if-then-else) |
| **Level 10** | **=** | assignment |
| | **+=** | add and assign |
| | **-=** | subtract and assign |
| | ***=** | multiply and assign |
| | **/=** | divide and assign |
| | **%=** | mod and assign |

# 1. Arithmetic Operators

# Arithmetic Operators

- Arithmetic operators includes:



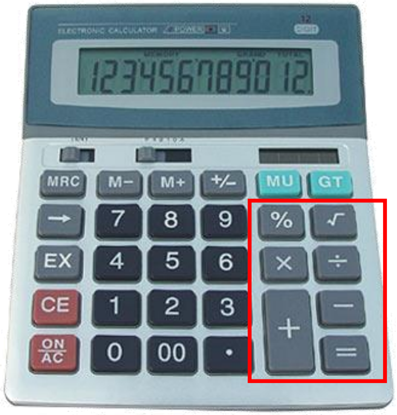
| Name | Operator | Description |
|---|---|---|
| Addition | + | Adds two operands |
| Subtraction | - | Subtracts one operand from another operand |
| Multiplication | * | Multiplies one operand by another operand |
| Division | / | Divides one operand by another operand |
| Modulus | % | Divides one operand by another operand and returns the remainder |

- The last three have higher precedence

```
2 + 4 * 3 -  4 / 2;
```

Will be implemented as?

```
(4 * 3) - (4 / 2) + 2 = 12
```



# Integer vs. Real Division

- If both **operands** in a division operation are **integer** the result will be integer division.
- If **one** or **both operands** in a division operation are **real numbers** the result will be a real division.

```
double a = 7, b = 2;
int x = 5, y = 2;

System.out.println ( a / b ) ;     // prints: ……………
System.out.println ( a / y );      // prints: ……………
System.out.println ( x / b );      // prints: ……………
System.out.println ( x / y) ;      // prints: ……………
```

- The last statement truncates the fraction portion of the result, because of the integer division

# Modulus Operator

- Modulus operator returns the *remainder* of an integer division on its operands:
- It gives a compilation error if one or both operands are not integer or integer compatible like character type.
- What is the result of following statements:

```
int  x = 5,  y = 2,  z = 4,  result;
result = x % 2;
result = z % 2;
result = y % 5;
```

```
// result is 1
// result is 0
// result is 2
```

# Implicit & Explicit Type Conversion

- You can convert any type to another type explicitly, **type casting (Type Conversion)**.
- Format: (type) variableName;

**Example 1:**

```
double x = 10.5;
int y = (int) x;
```

**Example 2:**

```
int x = 4, y = 7;
double ratio = ((double)x)/y;
```

The above example, first converts x to a double type then stores the result of a real division into variable ratio. Without the type cast operation, the result would have been zero.