Arithmetic Operators

CMSC 104, Spring 2014

Christopher S. Marron

(thanks to John Park for slides)

Arithmetic Operators

Topics

- Arithmetic Operators
- Assignment Operators
- Operator Precedence
- Evaluating Arithmetic Expressions
- Incremental Programming

Arithmetic Operators in C

- Binary Operators
  - E.g.:

    ```
    new_value = height + margin;
    area = length * width;
    ```

- Unary Operators
  - E.g.:

    ```
    new_value = -old_value;
    negation = !true_value;
    ```

Arithmetic Operators in C

Name	Operator	Example
Addition	+	num1 + num2
Subtraction	-	initial - spent
Multiplication	*	fathoms * 6
Division	/	sum / count
Modulus	%	m % n

Types and Promotion

- Can mix types in numerical expressions
- Hierarchy of types
  - By precision: int < float
  - By size: short < long
- Lower size/precision is *promoted* to greater size/precision before operation is applied
- Result is also of promoted type

Types and Promotion

- E.g.:

```
int num_sticks = 5;
double avg_stick_length = 4.5;
double total_length;

total_length = num_sticks * avg_stick_length;
```

num_sticks would be converted to double-precision, then multiplied by avg_stick_length

Division

- If both operands of a division expression are integers, you will get an integer answer. The fractional portion is thrown away.
- Examples : 17 / 5 = 3

  4 / 3 = 1

  35 / 9 = 3

Division (con't)

- Division where at least one operand is a floating point number will produce a floating point answer.
- Examples : 17.0 / 5 = 3.4

  4 / 3.2 = 1.25

  35.2 / 9.1 = 3.86813
- What happens? The integer operand is temporarily converted to a floating point, then the division is performed.

Division (con’t)

- Example1 :

  -
    ```
    int my_integer = 5;
    int my_product;

    my_product = (my_integer / 2) * 2.0;
    /* What will following print out? */
    printf(“my_product is %d\n”, my_product);

     /* What about this? */
    my_product = (my_integer / 2.0) * 2;
    printf(“my_product is %d\n”, my_product);
    ```

Division By Zero

- Division by zero is mathematically undefined.
- If you attempt to divide by zero in a program, it will cause a **fatal error**. Your program will terminate execution and give an error message.
- **Non-fatal errors** do not cause program termination, just produce incorrect results.

Modulus

- The expression **m % n** yields the integer remainder after **m** is divided by **n**.
- Modulus is an integer operation - both operands MUST be integers.
- Examples : 17 % 5 = 2
  6 % 3 = 0
  9 % 2 = 1
  5 % 8 = 5

Uses for Modulus

- Used to determine if an integer value is even or odd

  5 % 2 = 1 odd 4 % 2 = 0 even

  If you take the modulus by 2 of an integer, a result of 1 means the number is odd and a result of 0 means the number is even.

- The Euclid’s GCD Algorithm (done earlier)

Arithmetic Operators
Rules of Operator Precedence

Operator(s)	Precedence & Associativity
( )	Evaluated first. If **nested (embedded)**, innermost first. Otherwise, left to right.
* / %	Evaluated second. If there are several, left to right.
+ -	Evaluated third. If there are several, left to right.
=	Evaluated last, right to left.

Using Parentheses

- Use parentheses to change the order in which an expression is evaluated. The expresion

  ```
  a + b * c
  ```

  multiplies `b*c`, then adds `a` to the result.

  If you really want the sum of `a` and `b` to be multiplied by `c`, use parentheses:

  ```
  (a + b) * c
  ```

- Also use parentheses to clarify a complex expression.

Practice With Evaluating Expressions

Given integer variables `a`, `b`, `c`, `d`, and `e`, where `a = 1`, `b = 2`, `c = 3`, `d = 4`, evaluate the following expressions:

```
a + b - c + d
a * b / c
1 + a * b % c
a + d % b - c
e = b = d + c / b - a
```

Good Programming Practice

- It is best not to take the **“big bang” approach** to coding.
- Use an **incremental approach** by writing your code in incomplete, yet working, pieces.
- For example, for your projects,
    - Don’t write the whole program at once.
    - Just write enough to display the user prompt on the screen.
    - Get that part working first (compile and run).
    - Next, write the part that gets the value from the user, and then just print it out.

Good Programming Practice (con't)

  - Get that working (compile and run).
  - Next, change the code so that you use the value in a calculation and print out the answer.
  - Get that working (compile and run).
  - Continue this process until you have the final version.
  - Get the final version working.

- Bottom line: Always have a working version of your program!