CMSC 104 - Lecture 6
John Y. Park, adapted by C Grasso

Arithmetic Operators in C

Arithmetic Operators

Topics

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

Define Constant Macros

- Preprocessor directive
  - **#define** IDENTIFIER value
  - MUST start in column 1

- Used to define data and values that cannot change
  - They are not variables
  - They don't use an assignment operator (=)

Constant Macros

- Example

```
#define MY_CONSTANT 4
```

  - The preprocessor will go through the source code and replace all instances of `MY_CONSTANT` with `4`
    - Will only match the entire identifier
      - Eg, Will not replace "MY_CONSTANT1"
    - Will not replace anything inside of a string (ie, " ")

Constant Macros

Source code:

```
#define MY_CONSTANT 4
main () {
   int constant = MY_CONSTANT;
   printf("MY_CONSTANT is %d", MY_CONSTANT);
}
```

Precompiler changes the program in memory to:

```
main () {
   int constant = 4;
   printf("MY_CONSTANT is %d", 4);
}
```

The Assignment Operator

```
x = a + b
```

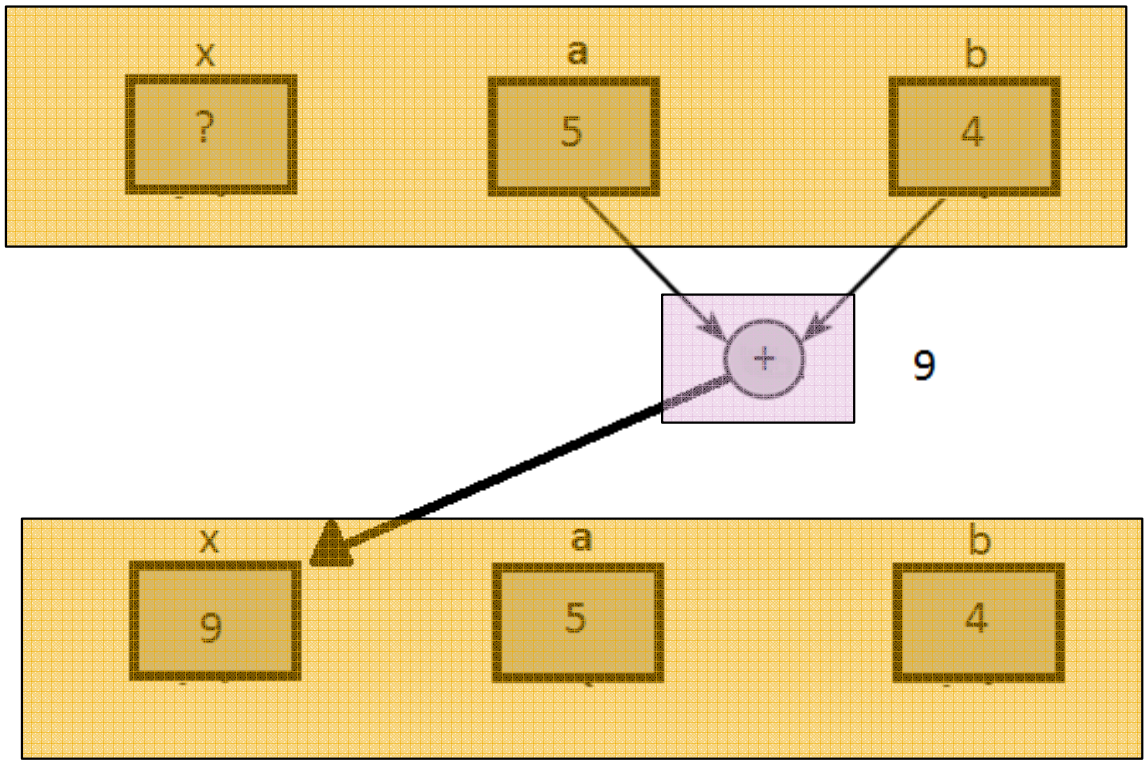
- **=** is the **assignment operator**
  - It computes the value of whatever is to the right of the **=**
  - Stores that value in the location named on the left hand side of the **=**
    - **There may only be <u>one</u> variable on the left side of the =**

The Assignment Operator

x = a + b

a + b

x =



Arithmetic Operators in C

- Unary Operators
  - Operates on **one** input value → one result

```
-old_value
!true_value
```

- Binary Operators
  - Operates on **two** input values → one result

```
height + margin
length * width
```

Arithmetic Operators in C

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

Division By Zero

- Division by zero is mathematically undefined.
- If you allow division 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

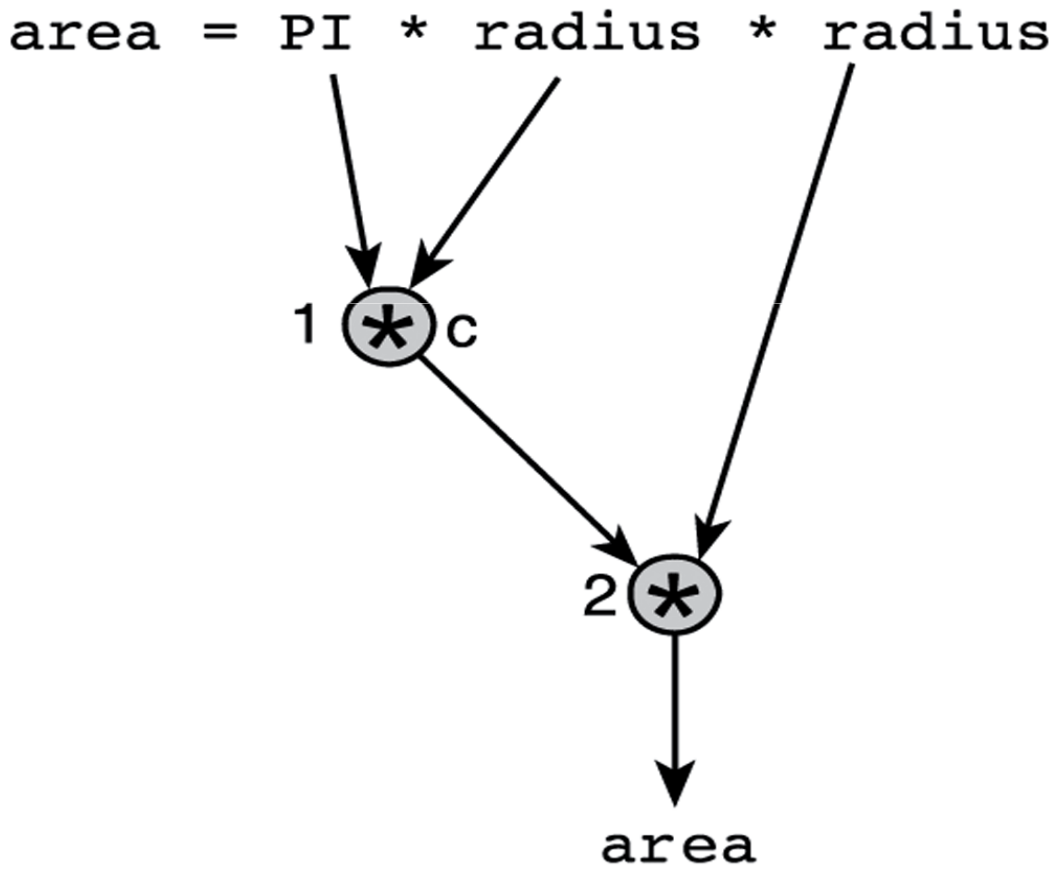
  5 % 2 = 1 odd 4 % 2 = 0 even

  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.

Rules of Operator Precedence

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

Evaluation Tree



Using Parentheses

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

  a + b * c Would multiply b * c first, then add a to the result.

  If you really want the sum of a and b to be multiplied by c, use parentheses to force the evaluation to be done in the order you want.

  (a + b) * c

- Also use parentheses to clarify a complex expression.

Writing Formulas in C

- There are no implied operations in C
  - ALL of them must be specified explicitly

Algebraic Formula	C Expression
2a --------- 1 – b	2 * a / (1 – b)
3a(b-2x)	3 * a * (b – 2 * x)

Practice With Evaluating Expressions

Evaluate the following expressions:

a = 1 c = 3
b = 2 d = 4

a + b - c + d
a * b / c
1 + a * b % c
a + d % b – c

e = b = d + c / b - a

Data Type Lengths

Data Type	Length in bytes
char	1
int	4
float	4
double	8

Types and Promotion

- Can mix data types in numerical expressions
- Hierarchy of types
  - By precision: int → float
  - By size: float → double
- Lower size/precision is *promoted* to greater size/precision before operation is applied
- Final result is of the highest 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 promoted to double-precision,

then multiplied by avg_stick_length

Division & Data Types

- 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 & Data Types

- Division where at least one operand is a floating point number will produce a floating point answer.
  - Fractional portion is kept.

- 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 & Data Types

- Example1 :

```
int my_integer = 5;
int my_product;

/* What will following print out? */
my_product = (my_integer / 2) * 2.0;
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);
```

Formatting Numerical Output

- What happens when you put a numerical expression in a printf() argument ?

```
printf("Result of  ( 127 * 5) / 3 is  %5d   \n", ( 127 * 5) / 3 );
```

Formatting Numerical Output - decimal

- printf shows 6 decimal places by default
- You can override this by using the format "%X.Yf"
  - X is the total number of digits to display (including the period)
  - Y is the number of digits after the decimal point to display
- Examples:

Value	Format	Displayed
12.789	%5.2f	12.79
12.789	%7.2f	__12.79
12.789	%8.5%	12.78900

  - In these examples, an underscore (_) is used to show where a blank would be displayed

Common Programming Errors

- There are three kinds of programming errors

1. Syntax error
   - Caught by the pre-processor, compiler, or linker
   - Basically, you typed something wrong
2. Run-time error
   - Something bad happened while the program was running
   - You should implement logic and checks to handle unexpected situations
   - Can cause the application to crash (bad) or give incorrect results (really bad)
3. Logic error
   - The wrong algorithm was implemented

- Errors in a piece of software are called bugs

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!