# Common Operators

Last updated 6/16/23

These slides introduce common C operators

# Common Operators

- Operation
  - Manipulation of one or more value to create a result
- Operator
  - Symbolic representation for an operation
  - Used extensively in mathematics and computer programming

    1 + 3 = 4

    c = a / b
- Operand
  - The element(s) operated on by the operator

    1 + 3 = 4

    c = a / b

# Common Operators

| Precedence | Operator | Description | Associativity |
|---|---|---|---|
| 1 | ++ -- | Suffix/postfix increment and decrement | Left-to-right |
| | () | Function call | |
| | [] | Array subscripting | |
| | . | Structure and union member access | |
| | -> | Structure and union member access through pointer | |
| | (*type*){*list*} | Compound literal(C99) | |
| 2 | ++ -- | Prefix increment and decrement | Right-to-left |
| | + - | Unary plus and minus | |
| | ! ~ | Logical NOT and bitwise NOT | |
| | (*type*) | Type cast | |
| | * | Indirection (dereference) | |
| | & | Address-of | |
| | sizeof | Size-of | |
| | _Alignof | Alignment requirement(C11) | |
| 3 | * / % | Multiplication, division, and remainder | Left-to-right |
| 4 | + - | Addition and subtraction | |
| 5 | << >> | Bitwise left shift and right shift | |
| 6 | < <= | For relational operators < and ≤ respectively | |
| | > >= | For relational operators > and ≥ respectively | |
| 7 | == != | For relational = and ≠ respectively | |
| 8 | & | Bitwise AND | |
| 9 | ^ | Bitwise XOR (exclusive or) | |
| 10 | \| | Bitwise OR (inclusive or) | |
| 11 | && | Logical AND | |
| 12 | \|\| | Logical OR | |
| 13 | ?: | Ternary conditional | Right-to-Left |
| 14 | = | Simple assignment | |
| | += -= | Assignment by sum and difference | |
| | *= /= %= | Assignment by product, quotient, and remainder | |
| | <<= >>= | Assignment by bitwise left shift and right shift | |
| | &= ^= \|= | Assignment by bitwise AND, XOR, and OR | |
| 15 | , | Comma | Left-to-right |

Precedence Chart of C Operators

# Common Operators

- Special note on binary numbers in C programming
  - Some but not all compilers allow binary numbers to be represented in C code directly

    95 → 0b01011101

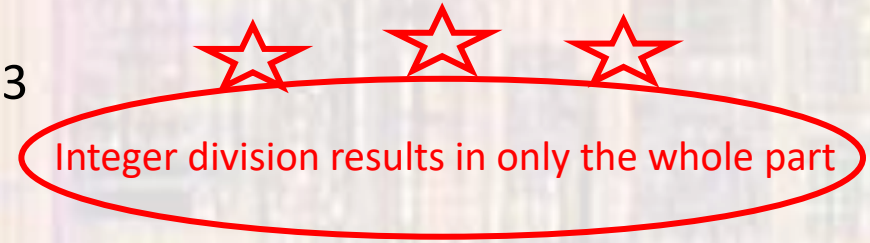
  - To be safe and ensure our code is portables we will NOT use this notation.
  - Binary numbers can be represented with:
    - Their decimal equivalents 95
    - Their hexadecimal equivalents 0x5D

# Common Operators

```
int a;
int b;
int c;
a = 13;
b = 5;
```

```
float x;
float y;
float z;
x = 5.566;
y = 2.2;
```

- Basic Math
  - +, - addition and subtraction
    - c = a + b; c = 18
  - *, / multiplication and division
    - c = a * b; c = 65
    - z = x / y; z = 2.53
    - c = a / b; c = 2




  - % modulo
    - 12 % 7 → 1 r5 = 5
    - c = a % b; c = 3
    - Not defined for anything but integers (or char)

modulo returns the remainder from dividing (just like in grade school)

# Common Operators

```
int a;
int b;
int c;
a = 13;
b = 5;
c = 18
```

```
float x;
float y;
float z;
x = 5.566;
y = 2.2;
```

- Relational Operators
  - Compares 2 values
    - Provides a T/F result
  - ==, <, >, <=, >=, !=
    - equals, LT, GT, LE, GE, not equal
    - evaluates to Boolean T or F
      - true == true → true
      - a == b → false
      - c == a + b → true
      - x >= y → true
      - x <= y → false
      - b < 5 → false
      - y != 2.2 → false

Note: equals is ==, not =

*** Logic will be discussed in another lecture

# Common Operators

```
int a;
int b;
int c;
a = 13;
b = 5;
c = 0
```

```
float x;
float y;
float z;
x = 5.566;
y = 2.2;
```

- Logical Operators
  - ! logical not
    - inverts the logical value
    - !true → false
    - !b → false
  - || logical OR
    - evaluates both sides logically then does an OR
    - true || false → true
    - c || 0 → false
    - c || b → true
  - && logical AND
    - evaluates both sides logically then does an AND
    - true && true → true
    - c && b → false
    - x && y → true

Reminder: The ONLY value that is false is 0 (0.0)

*** Logic will be discussed in another lecture

# Common Operators

```
int_8 a;
int_8 b;
int_8 c;
a = 0x86;
b = 0xA5;
c = -35
```

- Bitwise Operators
  - ~ bitwise not
    - inverts the individual bits in a number
      - This is NOT the 2's complement
    - ~a → ~(1000 0110) → 0111 1001 → 0x79
    - ~c → ~(1101 1101) → 0010 0010 → 34
  - | bitwise or
    - ORs the individual bits
    - a | b → (1000 0110) | (1010 0101) → 1010 0111 → 0xA7
  - & bitwise and
    - ANDs the individual bits
    - a & b → (1000  0110) & (1010 0101) → 1000 0100 → 0x84
  - ^ bitwise xor
    - XORs the individual bits
    - a ^ b → (1000 0110) ^ (1010 0101) → 0010 0011 → 0x23

*** Logic will be discussed in another lecture

# Common Operators

```
uint_8 a;

int_8 b;

a = 0xA6;
b = 0xA6;
```

- Bitwise Operators
  - >> bitwise shift right
    - shifts the individual bits in a number to the right
      - Uses sign extension to fill in the bits
    - a >> 2 → (1010 0110) >> 2 → 0010 1001 - unsigned

      OR
    - b >> 2 → (1010 0110) >> 2 → 1110 1001 - signed
  - << bitwise shift left
    - shifts the individual bits in a number to the left
      - Fills the bits with 0
    - a << 3 → (1010 0110) << 3 → 0011 0000 - unsigned

      OR
    - b << 3 → (1010 0110) << 3 → 0011 0000 - signed

# Common Operators

```
int a;
int b;
int c;
a = 10;
b = 20;
```

- Assignment
  - = assignment
    - variable = expression
    - places value into the variable on the left side
    - c = a + b; → c assigned the value 30
  - Compound variations *=, /=, +=, -=, %=
    - 2 operations performed
    - Basic math followed by assignment

    a *= b → a = a * b

    a += 10 → a = a + 10

    a -= b + c → a = a – (b + c)  note: the whole right side is evaluated first

# Common Operators

- Pre/Post Fix
  - ++ increment
  - -- decrement
  - The operation of these operators is covered in the expressions notes