Arithmetic Operators

- Normal binary arithmetic operators:
  + - * /
- Modulus or remainder operator: `x % y`
  is the remainder when `x` is divided by `y`
- Unary operators:
  - +

Precedence (see H&S, section 7.2.1)
  highest unary `- +
  * /
  % /
  lowest + -
so `-2*a + b`
  is parsed as
  `((-2)*a) + b`

Associativity: left to right

Print a number in decimal
```c
void putd(int n) {
  if (n < 0) {
    putchar('0');
    n = -n;
  }
  if (n >= 10)
    putd(n/10);
  putchar(n%10 + '0');
}
```

Can this program print `INT_MIN == -2147483648`?

Convert to negative numbers
```c
static void putneg(int n) {
  if (n <= -10)
    putneg(n/10);
  putchar('0123456789'[-(n%10)]);
}
```

Remainder is a mess:
```c
int a, b, q, r;
q = a/b; r = a%b;
```
- ANSI Standard guarantees only `q*b + r == a`
  `|r| < |b|`
  `r` might be negative if `a` is

Check for sign of `n%10`, handle both cases
```c
static void putneg(int n) {
  int q = n/10, r = n%10;
  if (r > 0) {
    r -= 10;
    q++;
  }
  if (n <= -10)
    putneg(q);
  putchar('0123456789'[-r]);
}
```
### An Easier Way

#### Use unsigned arithmetic

```c
#include <limits.h>
#include <stdio.h>

static void putu(unsigned n) {
    if (n > 10)
        putu(n/10);
    putchar("0123456789"[n%10]);
}

void putd(int n) {
    if (n == INT_MIN) {
        putchar('-');
        putu((unsigned)INT_MAX + 1);
    } else if (n < 0) {
        putchar('-');
        putu(-n);
    } else
        putu(n);
}
```

### Increment/Decrement

- **Pre** `x` operator increments operand before returning the value.
- **Post** `x` operator increments operand after returning the value.

```c
n = 5;
x = ++n;
x is 6, n is 6
```

- **Operands of** `++` and `--` must be variables.
- **Opands of** `+` and `--` must be integers.
- **Operands of** `&&` and `||` must be boolean expressions.
- **Operands of** `<`, `<=`, `>`, `>=` are integers.

### Relational & Logical Operators

- Logical values are **ints**: 0 is false, `!0` is true
- Normal relational operators:
  - `>`, `>=`, `<`, `<=`
- Equality operators:
  - `==`, `!=`
- Unary logical negation:
  - `!`
- Logical connectives:
  - `&&`, `||`

**Evaluation rules:** left-to-right; `&&` stops when the outcome is known to be `0`;
`||` stops when the outcome is known to be `!0`

```c
if (i >= 0 && i < 10 && a[i] == max)
    ++a[i];
```

### Bit Manipulation

- Bitwise logical operators apply to all the bits of an integer value:
  - `&` bitwise AND
    - `1&1=1`, `0&1=0`
  - `|` bitwise inclusive OR
    - `1|0=1`, `0|0=0`
  - `^` bitwise exclusive OR
    - `1^1=0`, `1^0=1`
  - `~` bitwise complement
    - `~1=0`, `~0=1`

- The `|` operator can be used to "turn on" one or more bits
- The `&` operator can be used to "turn off" one or more bits

**Examples using 16-bit quantities**

- `BIT0 = 0x1`
- `BIT1 = 0x2`
- `BITS = BIT0 | BIT1`
- `flags = flags | BITS;`
- `test = flags & BITS;`

**Examples using 32-bit quantities**

- `BIT0 = 0x1000000000000001`
- `BIT1 = 0x2000000000000002`
- `BITS = BIT0 | BIT1`
- `flags = flags | BITS;`
- `test = flags & BITS;`
Shift operators:

- \( x \ll y \) shifts \( x \) left \( y \) bit positions
- \( x \gg y \) shifts \( x \) right \( y \) bit positions

- When shifting right:
  - if \( x \) is signed, shift may be arithmetic or logical
  - if \( x \) is unsigned, shift is logical
  - arithmetic shift fills with sign bit
  - logical shift fills with 0

- When shifting left, the vacated bits are always filled with 0

Examples using 16-bit quantities

\[
\begin{align*}
\text{bits} & = \begin{array}{c} 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\
\end{array} \\
\text{bits} \ll 2 & = \begin{array}{c} 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array} \\
\text{bits} \gg 2 & = \begin{array}{c} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array} \\
\text{bits} \gg 2 & = \begin{array}{c} 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array} \\
\end{align*}
\]

Assignment

- Assignment is an operator, not a statement
- \( c = \text{getchar();} \) can be written as \( \text{if } (c = \text{getchar();} ) \) ...
- ... if \( c = \text{getchar();} \) can be written as \( c = \text{getchar();} \) ...
- Assignment is an operator, not a statement
- except for \( \& \), \( \&= \), \( \| \), \( ||= \), \( -= \), \( += \), \( *= \), \( /= \), \( %= \), \( &= \), \( ^= \), \( |= \), \( >>= \), \( <<= \)