# Integer and Floating Point Arithmetic

## ENEE 140

**Prof. Tudor Dumitraș**
Assistant Professor, ECE
University of Maryland, College Park

http://ter.ps/enee140

## Today's Lecture

- Where we've been
  - Using variables and constants
  - Variable assignment and operators
  - Iterating (`while`) and branching (`if`)
  - `printf()` and `scanf()`
  - Functions
  - Basics of data types
- Where we're going today
  - Unsigned data type
  - The % operator
  - Prefix and postfix increment operators
  - Assignment operators (+=)
  - Review of integer & floating point arithmetic
  - Overflow and underflow
- Where we're going next
  - Data types and type conversion

2

## Prefix and Postfix Increment Operators

- We've seen: increment and decrement operators in C

| | |
|---|---|
| `a++;` | same as a = a + 1; |
| `++a;` | same as a = a + 1; |
| `a--;` | same as a = a - 1; |
| `--a;` | same as a = a - 1; |

- We've also seen: value of assignment expressions

| | |
|---|---|
| `c = b = a = 0;` | a, b and c become 0 |

- Both a++ and ++a increment a, but they return different values

| | |
|---|---|
| `b = a++;` | postfix increment: a becomes 1, b becomes 0 |
| `b = ++a;` | prefix increment: a becomes 1, b becomes 1 |

  - Same for a-- and --a

3

## Assignment Operators

- We've seen

| | |
|---|---|
| `a++;` | increment a by 1 |
| `a = a + 10;` | increment a by 10 |

- Prefix and postfix operators

| | |
|---|---|
| `a = 1;` | |
| `b = a++;` | a becomes 2, b becomes 1 |
| `b = ++a;` | a becomes 3, b becomes 3 |

- Other assignment operators

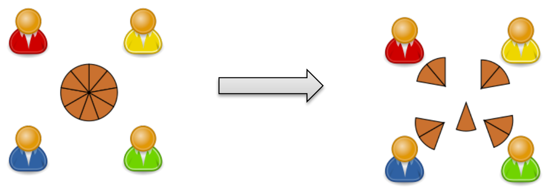
| | |
|---|---|
| `a += 10;` | increment a by 10 |

Same for -=, *=, /=, %=

4

## Integer (Euclidean) Division – In Math

- Dividing two integers produces a **quotient (q)** and a **remainder (r)**
  - The quotient and the remainder always exist and are unique
  - Example: dividing a pie with 9 slices among 4 people (source: Wikipedia)



  - 9 / 4 => q = 2 and r = 1
- Mathematical definition:
  - Given integers a and b, with $b \neq 0$: there exist **unique** integers r,q such that:
    - $a = b*q + r$ and
    - $0 \leq r < b$

5

## Integer (Euclidean) Division - Examples

- What are the remainders and quotients when dividing:
  - 8 by 4 — q = 2, r = 0
  - 4 by 8 — q = 0, r = 4
  - 10 by 10 — q = 1, r = 0

- What is the remainder when dividing:
  - $(2^n - 1)$ by 2 — r = 1
  - $2*n$ by n — r = 0
  - $(2^n - 1)$ by $2^n$ — $r = 2^n - 1$

(assume that n is a positive integer)

6

## Computer Arithmetic – Operations

- We've seen
  `+ - / *` arithmetic operators
  - Work for both integer and floating-point variables
  - Integer division truncates toward 0 (i.e. the fractional part is discarded)
- The modulus operator %
  - Works only for integers
  - Produces the remainder from integer division

```
int a = 5 / 3;
int b = 5 % 3;
```

value of a is `1`
value of b is `2`

- The values of `a % n` range between `0` and `(n-1)`

7

## Order of Evaluation

- Operator precedence (complete rules in K&R Table 2.1)
  1. `! ++ --` (unary operators)
  2. `* / %`
  3. `+ -`
  4. `< <= > >=`
  5. `== !=`
  6. `&&`
  7. `||`
  8. `=`
- Rule of thumb:
  - Division and multiplication come before addition and subtraction
  - Put parentheses around everything else

8

## Unsigned Data Types

- We've seen

```
int a = -1;
long b = -1;
```

- Unsigned data types are always positive

```
unsigned a = 1;
unsigned long b = 1;
```

- Unsigned literals

| | |
|---|---|
| `1U` | 1 as unsigned constant |
| `1LU` | 1 as unsigned long constant |

9

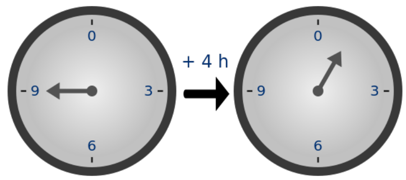
## Limits for Computer Integers

- Limits for unsigned integers (`unsigned`, not `unsigned long`)
  - UINT_MIN = 0
  - UINT_MAX = $2^w - 1$
- Limits for signed integers (`int`, not `long int`)
  - INT_MIN = $-2^{w-1}$
  - INT_MAX = $2^{w-1} - 1$
- $w$ is machine dependent
  - $w = 32$ on the GRACE machines
  - `UINT_MAX`, `INT_MIN` and `INT_MAX` are defined as constants in `limits.h`

10

## Integer Overflow

- What happens when you add 1 to UINT_MAX?
  - The mathematical value $(2^w - 1) + 1 = 2^w$ cannot be stored in an `unsigned` variable
  - The result of the operation is 0
- Intuition: unsigned numbers **wrap around**
  - Think of a 12h clock
  - 11 o'clock + 1h = 0
  - We count time **modulo 12h**
  - This means that the time displayed is the **remainder** from a division by 12




Source: Wikipedia

- **Unsigned operations are done modulo $2^w$**

11

## Unsigned Integer Addition

- Mathematical addition
  - $s \quad = u + v$
- `unsigned` addition: implements modular arithmetic on $w$ bits
  - $s \quad = (u + v) \mod 2^w$
  - Example: `UINT_MAX` + 1 $\quad = 2^w \mod 2^w = 0$ (**overflow**)
- Multiplication can overflow in similar manner
  - Same for addition and multiplication of signed integers

12

## Properties of Signed Integers – Examples

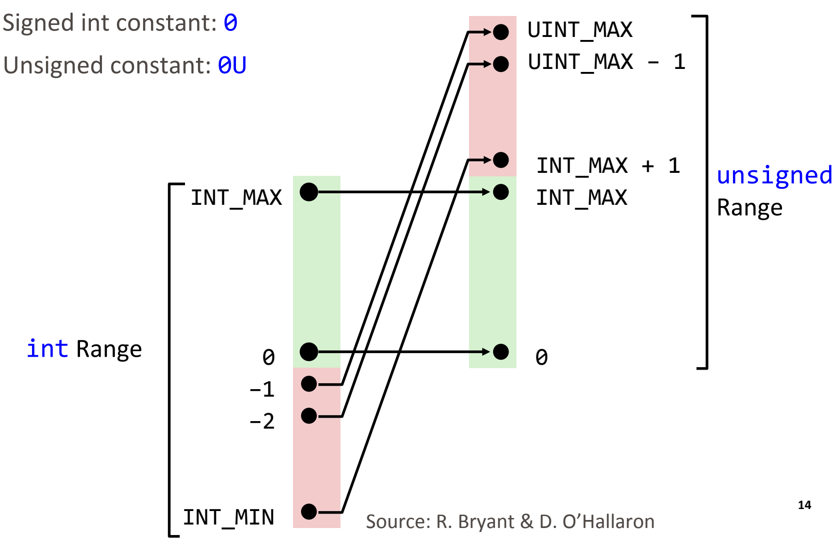
- You can represent more negative than positive numbers
  - Positive range: 1 .. $(2^{w-1} - 1)$
  - Negative range: -1 .. $-2^{w-1}$

- Signed integers can overflow as well
  - INT_MAX + 1 = INT_MIN
    - **Adding 2 positive numbers may produce a negative number!**
  - INT_MIN – 1 = INT_MAX
    - **Adding 2 negative numbers may produce a positive number!**
  - INT_MIN = -INT_MIN
    - INT_MIN is its own inverse

13

## Conversion Between Signed and Unsigned

- Type conversion `int` → `unsigned` visualized
  - Signed int constant: `0`
  - Unsigned constant: `0U`




Source: R. Bryant & D. O'Hallaron

14

## Mathematical Properties of Integer Arithmetic

- Closed under addition and multiplication
  - Result of signed/unsigned operation is also a signed/unsigned integer
- Commutative
- Associative
- 0 is additive identity; 1 is multiplicative identity
- Multiplication distributes over addition
  - a * (b + c) = a*b + a*c

- **Does not obey the ordering properties** of math integers

$u > 0 \quad \not\Rightarrow \quad u + v > v$

$u > 0, v > 0 \quad \not\Rightarrow \quad u \cdot v > 0$

15

## Properties of Floating Point Numbers

- As many negative as positive numbers

- Special values (constants for some of these defined in `float.h`)
  - Max floating point number ⇒ operations may **overflow**
  - Min floating point > 0 ⇒ operations may **underflow**
  - Smallest ε such that 1.0 + ε ≠ 1.0 ⇒ operation results may be **rounded**
  - +Inf, -Inf, NaN (not a number)

- **Avoid testing the equality** of values resulting from floating point operations

```
if (FLT_MAX == (FLT_MAX+1)) {…}        condition is true
if (cos(M_PI / 2) != 0.0) {…}          condition is true
```

16

## Mathematical Properties of Floating Point Arithmetic

- Closed under addition and multiplication
  - But may generate infinity or NaN
- Commutative
- **Not associative**
  - $(a + b) + c \neq a + (b + c)$
  - $(a * b) * c \neq a * (b * c)$
  - Possibility of overflow, inexactness of rounding
- Multiplication **does not distribute** over addition
  - $a * (b + c) \neq a * b + a * c$
  - Possibility of overflow, inexactness of rounding
- Monotonicity
  - $a \geq b \quad \Rightarrow a+c \geq b+c$
  - $a \geq b \ \& \ c \geq 0 \quad \Rightarrow a * c \geq b * c$
  - **Exceptions**: ±Inf and NaN

17

## Review of Lecture

- What did we learn?
  - Unsigned integers
  - The % operator
  - Prefix and postfix increment operators
  - Assignment operators (+=)
  - Sizes of data types
  - Limits of integer types and overflow
  - Properties of integer and floating point arithmetic
- Next lecture
  - Data types and type conversion
- Assignments for this week
  - Read **K&R Chapters 2.2, 2.9, 3.3, 6.1, B5, B6**
    - Note: some of these chapters refer to strings (e.g. `char s[]`), which we'll cover later
    - For now, think of `s[i]` as a character variable
  - Weekly challenge: **dec2bin.c**
  - Homework: **`lab05.pdf`** (on http://ter.ps/enee140), due on Friday at 11:59 pm
  - No quiz this week

18