COMP 250

Lecture 2

mod and remainder,
base representations (e.g. binary, hex),
base conversions,
exponential and log

Mon. Jan. 10, 2022

Division (last lecture)

Q: What do we mean by a/b ? (where $a > 0,\ b > 0$)

A: How many times can we subtract b until remainder is between 0 and $b - 1$.

By $a\ /\ b$, we mean the integer q such that $a = q * b + r, \quad 0 \leq r < b$

a is dividend, b is divisor
q is quotient, r is remainder

Division (in mathematics, not Java)

Q: What do we mean by a/b , if a can be positive or negative and $b > 0$?

A: How ~~many times can we subtract b until remainder is between 0 and $b - 1$.~~

By $a \,/\, b$, we mean the integer q such that $a = q * b + r, \quad 0 \leq r < b$

a is dividend, b is divisor
q is quotient, r is remainder

The "quotient remainder theorem" from mathematics says q and r are uniquely defined.

modulus (or mod) operator

Q: What do we mean by a/b , if a can be positive or negative and $b > 0$?

A: By $a \ / \ b$, we mean the integer q such that $a = q * b + r, \quad 0 \leq r < b$

a is dividend, b is divisor
q is quotient, r is remainder

The remainder r is called the modulus.

The *modulus* (or mod) operator is written :

$$a \ mod \ b = r.$$

Examples

$$13 \ mod \ 3 = 1$$

$$10 \ \ mod \ 5 = 0$$

$$-13 \ mod \ 5 = 2$$

For the last example, we have $-13 = (-3) * 5 + 2$ i.e. $a = q * b + r$.

ASIDE: Integer division operator / in Java

Q: What is a/b in Java ?

If a is an integer, and b is non-zero ***but now possibly negative***.

A: The *sign* of a/b is the sign of $a * b$.

The *magnitude* of a/b is the ***rounded down*** value of $\frac{|a|}{|b|}$.

Examples

$$13 \ / \ 5 = \ ?$$

$$-13 \ / \ 5 \ = \ ?$$

$$13 \ / \ -5 \ = \ ?$$

$$-13 \ / \ -5 \ = \ ?$$

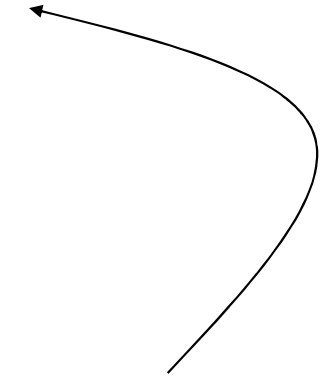
Examples

$$13 \;/\; 5 = 2$$

$$-13 \;/\; 5 = -2$$

$$13 \;/\; -5 = -2$$

$$-13 \;/\; -5 = 2$$



Not to be confused with $-13 = (\mathbf{-3}) * 5 + 2$ from quotient remainder theorem in mathematics, where the quotient is $\mathbf{-3}$.

ASIDE: remainder operator % in Java

Java does not have a mod operator. Instead, it has a "remainder" operator %.

Let a , b be integers, where b can be positive or negative but not 0.

The remainder operator % is uniquely defined, namely it satisfies :

$$a = (a \,/\, b) * b + (\boldsymbol{a \,\%\, b})$$

The % operator behaves like the mathematical "mod" when $a, b > 0$ and this is the only way you will use it in COMP 250.

You can check out *for yourself* how it behaves when $a < 0, b < 0$.

e.g. $-3 \; mod \; 5 = 2$ but $-3 \,\%\, 5 = -3$.

COMP 250

Lecture 2

mod and remainder,
base representations (e.g. binary, hex),
base conversions,
log

Mon. Jan. 10, 2022

Base 10 (decimal) "digits" {0,1,2,…, 9}

e.g. $5819 = 5 * 10^3 + 8 * 10^2 + 1 * 10^1 + 9 * 10^0$

$$m = \sum_{i=0}^{N-1} d[i]\, 10^i$$

digits

Base 5 "digits" {0, 1, 2, 3, 4}

e.g. $(21024)_5 = 2 * 5^4 + 1 * 5^3 + 0 * 5^2 + 2 * 5^1 + 4 * 5^0$

The brackets and subscripts are used to show what base is being used.

$$m = \sum_{i=0}^{N-1} d[i]\, 5^{\,i}$$

digits

Base 2 (binary) "bits" {0, 1}

e.g. $(11010)_2 = 1 * 2^4 + 1 * 2^3 + 0 * 2^2 + 1 * 2^1 + 0 * 2^0$

The brackets and subscripts are used to show what base is being used.

$$m = \sum_{i=0}^{N-1} b[i]\, 2^i$$

bits

Hexadecimal (base 16)

$$m = \sum_{i=0}^{N-1} h_i \, (16)^i$$

The hexadecimal coefficients h_i are in {0, 1, ….. , 9, 10, 11, … 15}

We often write them using symbols h_i in {0, 1, ….., 8, 9, **a, b, c, d, e, f**}

COMP 250

Lecture 2

mod and remainder,
base representations (e.g. binary, hex),
base conversions,
exponential and log

Mon. Jan. 10, 2022

How to convert *from binary to decimal* ?

You need to know the powers of 2.

i	2^i
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
10	1024
11	2048
:	:

Converting from binary to decimal

$$(11010)_2 = 1 * 2^4 + 1 * 2^3 + 0 * 2^2 + 1 * 2^1 + 0 * 2^0$$

$$= 16 + 8 + 0 + 2 + 0$$

$$= 26$$

You can verify the binary representation below for yourself….

decimal	binary
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001
10	1010
11	1011
:	:

decimal	binary
0	00000000
1	00000001
2	00000010
3	00000011
4	00000100
5	00000101
6	00000110
7	00000111
8	00001000
9	00001001
10	00001010
11	00001011
:	:

Fixed number of bits
(typically 8, 16, 32, 64)

8 bits is called a "**byte**".

Decimal	Binary	Hexadecimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	a
11	1011	b
12	1100	c
13	1101	d
14	1110	e
15	1111	f

Converting from binary to hexadecimal

A common use of hexadecimal is to represent long bit strings in more human readable form.

Example 1: 0010 1111 1010 0011

2 f a 3

We write 0x2fa3 or 0X2FA3.

x or X denotes hexadecimal.

Example 2: 10 1100

This number is treated as 0010 1100 where we have padded with 0's to the left. We write this number in hexadecimal as 0x2c.

Converting from binary to hexadecimal

To understand *why the conversion is correct*, consider binary representation of m.

$$m = b_k 2^k + \cdots + b_8 2^8 + b_7 2^7 + b_6 2^6 + b_5 2^5 + b_4 2^4 + b_3 2^3 + b_2 2^2 + b_1 2^1 + b_0 2^0$$

We can rewrite the right hand side by grouping into 4-tuples, starting from the right.

$$b_k 2^k + \cdots + (b_{11} 2^3 + b_{10} 2^2 + b_9 2^1 + b_8) 2^8 + (b_7 2^3 + b_6 2^2 + b_5 2^1 + b_4) 2^4 + (b_3 2^3 + b_2 2^2 + b_1 2^1 + b_0)$$

which is just $\sum_{i=0} h_i 16^i$ where the coefficients h_i are the terms in brackets.

Converting from decimal to binary

$$(241)_{10} = (\ ?\ )_2$$

Recall the brackets and subscripts are used to show what base is being used.

Soon I will present an algorithm for doing this conversion. The algorithm is based on the following ideas.

For any integer $m > 0$ and any $base > 0$, we can write :

$$m = (m/base) * base + m \,\%\, base$$

For example, base 10:

$$m = (m/10) * 10 + m \,\%\, 10$$

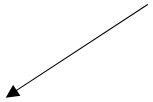
For example, base 2:

$$m = (m/2) * 2 + m \,\%\, 2$$

Converting from base 10 to base 2 uses the last of these relationships. Let's unpack this.

$$m = m/2 * 2 + m \,\%\, 2$$

i.e. this is the solution to "Convert m to binary."



Suppose that m is written in binary as $(b_4 b_3 b_2 b_1 b_0)_2$ where b_i are bits.

Then:

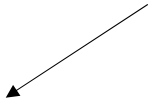
$$m \,\%\, 2 = ?$$

$$m \,/\, 2 = ?$$

$$(m/2) * 2 = ?$$

$$m = m/2 * 2 + m \,\%\, 2$$

i.e. this is the solution to "Convert m to binary."



Suppose that m is written in binary as $(b_4 b_3 b_2 b_1 b_0)_2$ where b_i are bits.

Then:

$$m \,\%\, 2 = (b_0)_2$$

$$m \,/\, 2 = (b_4 b_3 b_2 b_1)_2$$

$$(m/2) * 2 = (b_4 b_3 b_2 b_1 0)_2$$

$$m = m/2 * 2 + m \,\%\, 2$$

A specific example: $m = (11010)_2$

$$m \,\%\, 2 = (0)_2$$

$$m/2 = (1101)_2$$

$$(m/2) * 2 = (11010)_2$$

Algorithm to convert from decimal to binary

Given a positive integer m in some base 10, convert it to a binary representation, where .... *b[3] b[2] b[1] b[0]* are the bits.

The arguments on the previous slide suggest the following algorithm:

$i \leftarrow 0$
while $m > 0$ **do**
  $b[i] \leftarrow m \ \% \ 2$
  $m \leftarrow m \ / \ 2$
  $i \leftarrow i + 1$
end while

Example: Convert 241 from decimal to binary

```
i ← 0
while m > 0 do
   b[i] ← m % 2
   m ← m / 2
   i ← i + 1
end while
```

i	m	$b[i]$
	241	
0	120	1

241 = (241/2)*2 + 1

Each row of the table shows the values after executing this instruction, i.e. before incrementing i.

Example: Convert 241 from decimal to binary

```
i ← 0
while m > 0 do
   b[i] ← m % 2
   m ← m / 2
   i ← i + 1
end while
```

i	m	$b[i]$
	241	
0	120	1
1	60	0

241 = (241/2)*2 + 1
120 = (120/2)*2 + 0

Each row of the table shows the values after executing this instruction, i.e. before incrementing i.

Example: Convert 241 from decimal to binary

```
i ← 0
while m > 0 do
   b[i] ← m % 2
   m ← m / 2
   i ← i + 1
end while
```

i	m	$b[i]$
	241	
0	120	1
1	60	0
2	30	0

241 = (241/2)*2 + 1
120 = (120/2)*2 + 0
60 = (60/2)*2 + 0

Each row of the table shows the values after executing this instruction, i.e. before incrementing i.

Example: Convert 241 from decimal to binary

```
i ← 0
while m > 0 do
   b[i] ← m % 2
   m ← m / 2
   i ← i + 1
end while
```

i	m	$b[i]$
	241	
0	120	1
1	60	0
2	30	0
3	15	0

241 = (241/2)*2 + 1
120 = (120/2)*2 + 0
60 = (60/2)*2 + 0
30 = (30/2)*2 + 0

Each row of the table shows the values after executing this instruction, i.e. before incrementing i.

Example: Convert 241 from decimal to binary

```
i ← 0
while m > 0 do
   b[i] ← m % 2
   m ← m / 2
   i ← i + 1
end while
```

i	m	$b[i]$
	241	
0	120	1
1	60	0
2	30	0
3	15	0
4	7	1

241 = (241/2)*2 + 1
120 = (120/2)*2 + 0
60 = (60/2)*2 + 0
30 = (30/2)*2 + 0
15 = (15/2)*2 + 1

Each row of the table shows the values after executing this instruction, i.e. before incrementing i.

Example: Convert 241 from decimal to binary

```
i ← 0
while m > 0 do
   b[i] ← m % 2
   m ← m / 2
   i ← i + 1
end while
```

Each row of the table shows the values after executing this instruction, i.e. before incrementing i.

i	m	$b[i]$
	241	
0	120	1
1	60	0
2	30	0
3	15	0
4	7	1
5	3	1
6	1	1
7	0	1
8	0	0

241 = (241/2)*2 + 1
120 = (120/2)*2 + 0
60 = (60/2)*2 + 0
30 = (30/2)*2 + 0
15 = (15/2)*2 + 1
7 = (7/2)*2 + 1
3 = (3/2)*2 + 1
1 = (1/2)*2 + 1
....

Example: Convert 241 from decimal to binary

```
i ← 0
while m > 0 do
   b[i] ← m % 2
   m ← m / 2
   i ← i + 1
end while
```

Each row of the table shows the values after executing this instruction, i.e. before incrementing i.

i	m	$b[i]$
	241	
0	120	1
1	60	0
2	30	0
3	15	0
4	7	1
5	3	1
6	1	1
7	0	1
8	0	0

Answer:

$b[\ ]$ = …011110001

Converting from base 10 into any $base$

Given $m > 0$ expressed in base 10, the base conversion algorithm allows us to convert m into any $base > 0$, namely it allows us to express m as a sum of powers of that $base$.

The algorithm uses the fact that $m = (m/base) * base + m \,\%\, base$.

$i \leftarrow 0$
while $m > 0$ **do**
  $b[i] \leftarrow m \,\%\, base$
  $m \leftarrow m \,/\, base$
  $i \leftarrow i + 1$
end while

COMP 250

Lecture 2

mod and remainder,
base representations (e.g. binary, hex),
base conversions,
exponential and log

Mon. Jan. 10, 2022

The New York Times

OPINION

Stop Saying 'Exponential.' Sincerely, a Math Nerd.

By Manil Suri
Dr. Suri teaches math at the University of Maryland, Baltimore County.

Here's a trend I find worrisome enough to consider petitioning the Word Police. The word "exponential" has been growing in media usage — pretty much, well, exponentially.

Do a keyword search in The New York Times's archives for the various forms of the word and you'll see a rough doubling each decade: 105 hits for the 1970s, 279 for the 1980s, 604 for the 1990s and 1,375 for the 2000s (the rise continues with 1,753 hits so far in the 2010s, though it might not quite double this decade). Crude and unscientific evidence, but the hits for "linear" (for example) don't reveal comparable growth.

You would think such popularization of a technical term would warm the cockles of my mathematician heart. How wonderful if the surging acceleration of our times (in digital capacity, epidemics, chaos in general) is being described mathematically!

Alas, some recent hits reveal that the word is often used just to mean "lots." Tasks are "exponentially more difficult" if one is blind, space war "exponentially more dangerous" than that on earth. The Washington Post reports on "exponentially richer private-sector jobs" and even an e-sports industry that in 2015 was "exponentially more nascent than today." Hyperbole, more than acceleration, seems to define our times.

For the full opinion piece, see here although you might need to pay for it.

[This slide was updated (Jan 10 after class). See replacement below. For more details, see lecture notes.]

Definition of exponential and logarithm

Let $b > 0$ be a positive real number, and let x be a real number.

The exponential (or power) function for base b is $\boldsymbol{y = b^x}$.

The inverse of this function is the *logarithm* (base b). It is written: $\boldsymbol{y = log_b\, x.}$

(This inverse function represents the same equation as $\boldsymbol{x = b^y}$.)

In Calculus, one uses the natural logarithm (base e).

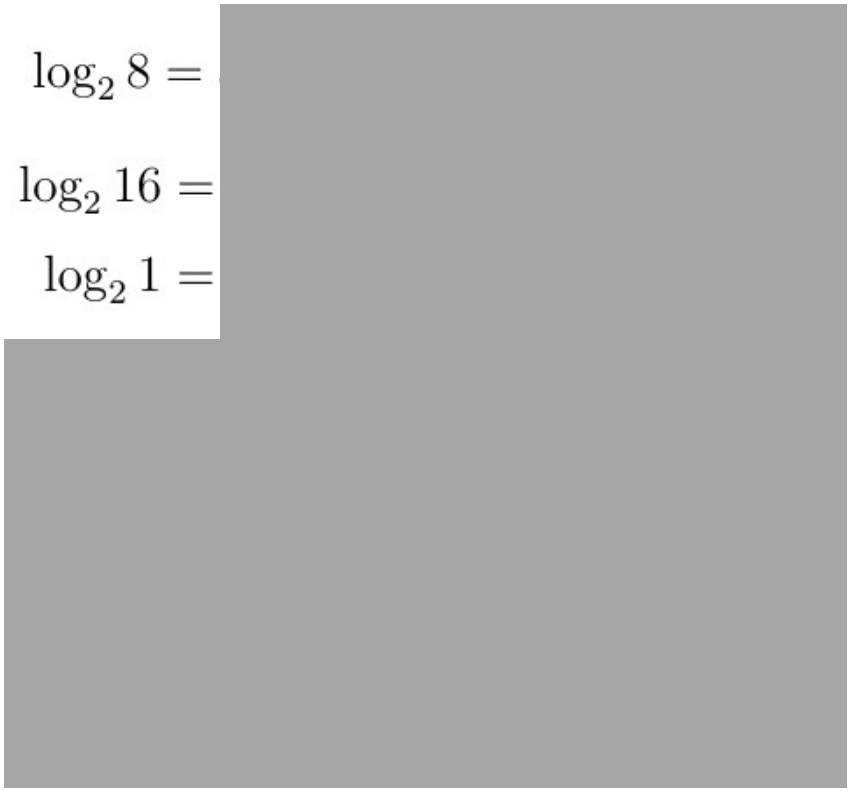
In Computer Science, one typically uses base 2 and $\boldsymbol{x}$ is typically an integer.

Examples

$\log_2 8 =$

$\log_2 16 =$

$\log_2 1 =$



Examples

$$\log_2 8 = 3, \qquad i.e.\ \ 2^3 = 8$$

$$\log_2 16 = 4 \qquad i.e.\ \ 2^4 = 16$$

$$\log_2 1 = 0 \qquad i.e.\ \ 2^0 = 1$$

$$\log_8 2 =$$

$$\log_2 \frac{1}{8} =$$

$$\log_8 \frac{1}{2} =$$

Examples

$$\log_2 8 = 3, \qquad i.e.\ \ 2^3 = 8$$

$$\log_2 16 = 4 \qquad i.e.\ \ 2^4 = 16$$

$$\log_2 1 = 0 \qquad i.e.\ \ 2^0 = 1$$

$$\log_8 2 = \frac{1}{3} \qquad i.e.\ \ 8^{\frac{1}{3}} = 2$$

$$\log_2 \frac{1}{8} = -3 \qquad i.e.\ \ 2^{-3} = \frac{1}{8}$$

$$\log_8 \frac{1}{2} = -\frac{1}{3} \qquad i.e.\ \ 8^{-\frac{1}{3}} = \frac{1}{8^{\frac{1}{3}}} = \frac{1}{2}$$

More Properties of exponential & logarithm (review)

Let $b > 0$ be a positive real number (base), and let n and m be integers.

- $b^{n+m} = b^n b^m$
- $(b^n)^m = b^{nm}$
- $log_b\,(xy) = log_b\,x + log_b\,y$
- $log_b\,(x^n) = n\,log_b\,x$
- $log_b\,(x) = log_b\,a * log_a\,x$

ASIDE: For proofs of these properties, see the lecture notes.

You do need to memorize the properties, but you do not need to memorize the proofs.

Coming up…

Lectures

- Wed. Jan 12
  more on log and binary
  Java primitive types
- Friday. Jan. 14
  basic Java

Homework (TODO)

- See link from last lecture to w3schools.
- Install either Eclipse or IntelliJ.
- See Content -> tutorials.
  Tutorial TA office hours this week.