STMC HKOI Training

Lesson 3: Modulo, number systems and binary numbers

Chan Yan Mong

December 21, 2021

Goal today

- Division algorithm
- Modulo operator and application
- Number system
- Binary numbers, bit, byte

Division algorithm

- Back in primary school, you should have learnt that we can divide two integers and get a **quotient** and **remainder**
- For example:
  - $5/2 = 2 \cdots 1$
  - $13/9 = 1 \cdots 4$
  - $78/5 = 15 \cdots 3$
- How do we formalize this idea? The answer lies in the **division algorithm**

Division algorithm

Theorem (Division Algorithm)

Given two integers (can be positive or negative) a, b and $b \neq 0$. There exist two unique integers q, r such that:

$$a = qb + r \quad \text{where} \quad 0 \leq r < |b|$$

*where $|b|$ denote the absolute value of b. a is called the **dividend**, b is called the **divisor**, q is called the **quotient** and r is called the **remainder***

Division algorithm

- Let's look at some examples:
  - $a = 78, b = 9$: $78 = 15 \times 9 + 3$
  - $a = -29, b = 7$: $-29 = (-5) \times 7 + 6$
  - $a = 54, b = -13$:
- Question: Is such decomposition unique?

Uniqueness of division algorithm (Optional)

Proof.

Suppose not, that is, there exist two q_1, q_2 and r_1, r_2 such that:

$$a = q_1 b + r_1 \quad (0 \leq r_1 < |b|)$$
$$a = q_2 b + r_2 \quad (0 \leq r_2 < |b|)$$

so that $q_1 \neq q_2$ and $r_1 \neq r_2$.

Uniqueness of division algorithm (Optional)

Proof.

Then:

$$\begin{aligned} b(q_1 - q_2) &= (r_2 - r_1) \\ |b||q_1 - q_2| &= |r_2 - r_1| < |b| \end{aligned}$$

This would imply $|q_1 - q_2| = 0$ and thus $|r_2 - r_1| = 0$, so $r_1 = \pm r_2$.
But r_1, r_2 have the same sign, so $r_1 = r_2$. This immediately leads to $q_1 = q_2$ $\square$

Divisibility

Now we can introduce the notion of divisibility:

Definition (Factor and divisibility)

Let $a, b \in \mathbb{Z}$. We say a is **divisible** by b if the remainder of a/b is zero. We also say that b is a **factor** of a. To save writing, sometimes we denote that as:

$$b|a \iff \text{b divides a} \tag{1}$$

For example:

10 is divisible by 2 because $10 = 2 \times 5 + 0$. We also say 2 is a factor of 10

11 is not divisible by 2 because $10 = 2 \times 5 + \mathbf{1}$ and $1 \neq 0$

Modulo operator

- For reasons that will be clear later, sometimes we would want to calculate the remainder of a/b
- In python, there's a build-in way for us to do that directly (at least for $b > 0$)
- This is called the **modulo operator** %
- The modulo operator is defined as follows:

```
a === floor(a/b)*b + a%b #Python definition
```

Here `a/b` is a floating point division

Modulo operator

- Let's see some of the implications of that:
- Consider the case when $a = 19, b = 3$:
  - Since `floor(a/b)=6`; `a%b = 19 - 6*3 = 1`
  - This is consistent with $19/3 = 6 \cdots 1$
  - So in general if $a > 0$ and $b > 0$, `a%b` gives the **remainder** when `a` divides by `b`
  - The range of `a%b` in this case is $[0, b-1]$
- Now consider the case when $a = 0, b = 3$:
  - Since `floor(a/b)=0`; `a%b = 0 - 0 = 0`
  - So in general `a%b = 0` when `a=0`

Modulo operator

- Consider the case when $a = -19, b = 3$:
  - Since `floor(a/b)=floor(-6.333) = -7`; `a%b = -19 - (-7)*3 = 2`
  - In general if $a < 0$ and $b > 0$, it's still equivalent to the remainder
  - The range in this case is $[0, b-1]$
- Now consider the case when $a = -19, b = -3$:
  - Since `floor(a/b)=6`; `a%b = -19 - 6*(-3) = -1`
  - This is *different* from the remainder defined above!
  - In fact the true one is $|-3| + (-1) = 2$
  - So in general if $a < 0$ and $b < 0$, `a%b = r - |b|`, where r is the remainder defined above

Modulo operator

- Finally consider the case when $a = 19, b = -3$:
  - Since `floor(a/b)=-7`; `a%b = 19 - (-7)*(-3) = -2`
  - This is again *different* from the remainder defined above
  - The true one this time is $|-3| + -2 = 1$
  - So in general if $a < 0$ and $b < 0$, `a%b = r - |b|`, where *r* is the remainder defined above
- The range of `a%b` is summarized in the table below:

	a<0	a=0	a>0
b>0	[0,b-1]	0	[0,b-1]
b=0		undefined	
b<0	[-(b-1),0]	0	[-(b-1),0]

Modulo and periodicity

- The output of `a%b` is **periodic**
- The period is `b` (i.e. the values will wrap around after `b`)
- Also note that `a%b == 0` iff `a` is divisible by `b`
- Here's an example for $a > 0$:

a	0	1	2	3	4	5	6	7	8	9
a%2	0	1	0	1	0	1	0	1	0	1
a%3	0	1	2	0	1	2	0	1	2	0
a%4	0	1	2	3	0	1	2	3	0	1

Modulo and periodicity

- Some examples:
- `11%3 = 2`, `14 % 3 = 2`, `17% 3 = 2`, `8 % 3 = 2`, `5 % 3 == 2`
- So in general $11+3m$, $m \in \mathbb{Z}$ have the same modulo as 11
- In general, `(a+bm)%m = a % m`
- Hence, modulo operator is extensively used in things that wrap around (e.g. time around the clock)

Example: Minutes to hour

Problem

Write a program that time from minutes to form hh:mm

Solution

Let's first analyse what the problem want us to do. Begin with some examples:

3 minutes $\rightarrow 00 : 03$

63 minutes $\rightarrow 01 : 03$

123 minutes $\rightarrow 02 : 03$

Notice that `mm` part is periodic with a period of 60.

Hence one might guess that `mm = minutes % 60`. Is it true?

Example: Minutes to hour

Solution

The answer is **yes**. Consider a time of format `hh:mm`. Then the corresponding time in minutes is :

$$\text{minute} = 60 \times \text{hh} + \text{mm}$$

Notice mm is just the remainder of minute divided by 60. So to get `mm` in general what we do is:

```
mm = minute % 60  # Get mm from minutes
```

Example: Minutes to hour

Solution

Once we get `mm`, the rest are simple, because:

$$\text{hh} = (\text{minute} - \text{mm})/60$$

So we can get `hh` by

```
hh = (minute - mm) //60  # Get hh from minutes
```

Example: Minutes to hour

Solution

The full code is thus:

```
minute = int(input('Time in minute: '))
mm = minute % 60  # Get mm from minutes
hh = (minute - mm) //60  # Get hh from minutes
print('Time in hh:mm ',hh,':','mm')
```

Exercise: Seconds to hh:mm:ss

- Write a program that converts seconds to hh:mm:ss format (hour, minutes, seconds)
- For example, 43753s is 12:09:13 (12 hrs 9 mins and 13 seconds)
- (Hint: $T_{\text{sec}} = T_h \times 60^2 + T_m \times 60 + T_s$)
- For example: $43753 = 12 \times 60^2 + 9 \times 60 + 13$
- So $43753 = (12 \times 60 + 9) \times 60 + 13$ and thus 13 is the remainder of 43752/60

Example input and output of the code:

```
$./main
43753
12 09 13
```

Example: Checking factors and multiples

Checking odd or even

The modulo operator `%` can help us check whether a number is odd or even. This is because the remainder of `n % 2` is 0 if and only if `n` is even and is 1 if and only if `n` is odd. For example:

```python
n = int(input('Enter a number: '))

if n % 2 == 0:
  print("It's even")
else:
  print("It's odd")
```

Example: Checking factors and multiples

Checking multiples of *n*

More generally, the expression `m % n = 0` if and only if $n|m$ (i.e. m is a multiple of n). Note that if n divides m, n is a factor of m, so we can check factors in a Similar way.

```
n = int(input('Enter a number: '))

if n % 11 == 0: # n is a multiple of 11 / 11 is a factor of n
  print('n is a multiple of 11')
else:
  print('n is not a multiple of 11')
```

Exercise: Checking factors and multiples

Exercise (Finding factors)

Write a program that print and count the numbers from 1-100 that are

1. divisible by 3
2. divisible by 5
3. divisible by 3 and 5
4. divisible by 3 or 5 only

Exercise: Checking factors and multiples

Solution

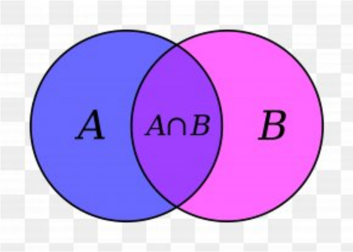
```python
count3,count5,count35 = 0,0,0

for i in range(1,100+1):
  if i % 3 == 0:
    print(i,' is a multiple of 3')
    count3 = count3 + 1
  if i % 5 == 0:
    print(i,' is a multiple of 5')
    count5 = count5 + 1
  if i % 3 == 0 and i % 5 == 0:
    print(i,' is a multiple of both 3 and 5')
    count35 = count35 + 1

print('3: ',count3,'; 5: ',count5,'; 3 and 5: ',count35, '; 3 or 5: ',count3+count5-count35)
```

Exercise: Check factor and multiples

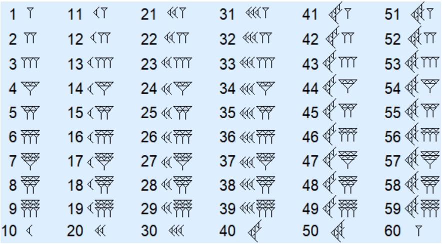
- Curious among you might wonder why we need to subtract `count35` for the last one
- This is because both `count3` and `count5` contains multiples of 15, so `count3 + count5` will count the multiples that are both 3 and 5 twice
- Hence we need to subtract them from `count3 + count5`
- See the **Venn diagram** on the right



Source: https://bit.ly/3D8dqBo

Exercise: Sexagesimal numbers

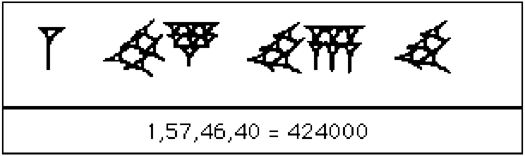
- What is Sexagesimal system?
- Instead of having 10 symbols representing the digits (0,1,2,3,4,5,6,7,8,9), they have *60* symbols representing a digits
- Then every 60 they carry one digit



Babylonian numerals (Source: Dr. Aart)

Exercise: Sexagesimal numbers

- For example, for the number represented below:
- The digits are 1,57,46,40
- So the number representing is $1 \times 60^3 + 57 \times 60^2 + 46 \times 60 + 40 = 424000$



Source : MacTour

Exercise: Sexagesimal numbers

- Now let's write a program that converts a decimal number into sexagesimal number
- Instead of those funny numerals in the previous table, we shall use the usually 0-59 separated by comma to represent the sexagesimal digit
- For example, 424000 in previous example will be written as 1,57,46,40
- Some more examples:

Base 10	1343	6948	67	382	23432
Base 60	0,22,23	1,55,48	0,1,7	0,6,22	6,30,32

Exercise: Sexagesimal number

Problem

You have two task in this exercise:

1. Given a number in sexagesimal representation $\{d_1, d_2, d_3\}$, where $0 \leq d_1, d_2, d_3 \leq 59$, convert the number into base 10

Example:

6 30 32 in base-60 is 23432 in base-10 and 0 6 22 base-60 is 382 and so on

```
    Example 1:        Example 2:        Example 3:
    ./main            ./main            ./main
    6 30 32           0 6 22            1 55 48
    23432             382               6948
```

Exercise: Sexagesimal number

Problem

2. Given a integer N in base-10 representation, where $0 \leq N < 60^4$, return a 3 digit base-60 representation of the number

Example:

23432 in base-10 is 6 30 32, 1343 is 0 22 23 and so on

```
    Example 1:      Example 2:      Example 3:
    ./main          ./main          ./main
    23432           1343            67
    6 30 32         0 22 23         0 1 7
```

Number system

- In fact, what the previous exercise illustrates is the idea of a **number system**
- In short, the same number can be written in vastly different ways, depending on where you choose to carry you digit

Number system

- To illustrate this, let's consider the number 37
- What do we actually mean by 37?
- Actually we meant:

$$37 = 3 \cdot 10 + 7$$

- Similarly, when we write 139

$$139 = 1 \cdot 10^2 + 3 \cdot 10 + 9$$

Number system

- In general, if we write a *m* digit number as:

$$n = a_{m-1}a_{m-2}a_{m-3}\cdots a_2a_1a_0$$

We mean:

$$n = a_{m-1}\cdot 10^{m-1} + a_{m-2}\cdot 10^{m-2} + \cdots + a_2\cdot 10^2 + a_1\cdot 10^1 + a_0$$

This is what we meant by expressing a number in base 10

Number system

- To give a concrete example, consider a 3 digit number $n = 293$
- Then $n = a_2 a_1 a_0$, with $a_2 = 2, a_1 = 9, a_0 = 3$
- Substituting it back to our expression:

$$\begin{aligned} n &= a_2 \cdot 10^2 + a_1 \cdot 10^1 + a_0 \\ &= 2 \cdot 10^2 + 9 \cdot 10^1 + 3 \end{aligned}$$

As expected

Number system

- But why choose 10? What if we choose to use another number instead of 10?
- Let's choose 2 for example. Then for $n = 5$:

$$5 = 1 \cdot 2^2 + 0 \cdot 2^1 + 1$$

- In this case, we write $5 = 101_2$ here the subscript 2 means we are using 2 instead of 10
- In fact this is called **binary numbers**

Number system

Definition (Binary numbers)

Let n be a positive integer. Then we can write n in the following way:

$$n = a_{m-1} \cdot 2^{m-1} + a_{m-2} \cdot 2^{m-2} + \cdots + a_2 \cdot 2^2 + a_1 \cdot 2^1 + a_0$$

where a_i is either 0 or 1 for $0 \leq i \leq m-1$. We say n is a m digit binary number and a_i are the $i+1$th digits of n. Furthermore, we say $n = (a_{m-1}a_{m-2}\cdots a_1 a_0)_2$ is the **binary representation** of n in base 2

Number system

- For example, $n = 19$:

$$19 = 1 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1$$

- So $n = 19$ is a 5 digit binary number
- Also $a_0 = 1, a_1 = 1, a_2 = 0, a_3 = 0, a_4 = 1$ are the 1st, 2nd, 3rd, ..., 5th digits of n in base 2.
- Furthermore, the binary representation of $n = 19$ is 10011_2

Number system

- We now introduce an algorithm for finding the binary representation of n in base 2

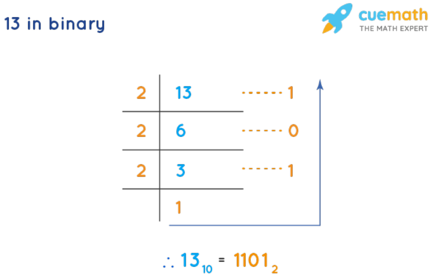
Algorithm (Binary number by short divison)

Let m be an integer, the remainder of m divided by 2 is $m\%2$. Then the following algorithm gives the binary representation for an integer n:

1. $n_0 = n$
2. $a_i = n_i \% 2$
3. $n_{i+1} = (n_i - a_i)/2$
4. repeat 2,3 until $n_i = 0$

Number system

- Example of executing the algorithm by hand:



Source: Cuemath

Number system

Exercise:

1. Convert the following numbers into binary: 16,93,34,11
2. What is the last digit in the binary representation of the following numbers (Hint: You can read out the ans directly): 19,30,44,21
3. Convert the following binary numbers back to deciaml: $101_2, 1011_2, 111_2, 1111_2$

Number system

- Here is a code that does decimal to binary conversion

```python
n = int(input('Enter an integer: '))
a = [] # List storing the digits

while n != 0:
  ai = n%2       # Step 2
  a.append(ai)  # Add i th digit on list
  n = (n-ai)//2 # Step 3

a = a[::-1] # Reverse the list (e.g. [1,0,1,1] -> [1,1,0,1])
print(a) # Print the digits
```

Number system

- We can extend the definition to include arbitary bases:

Definition (Base b numbers)

Let n be a positive integer. Then we can write n in the following way:

$$n = a_{m-1} \cdot b^{m-1} + a_{m-2} \cdot b^{m-2} + \cdots + a_2 \cdot b^2 + a_1 \cdot b^1 + a_0$$

where a_i is from 0 to $b-1$ for $0 \leq i \leq m-1$. We say n is a m digit number in base b and a_i are the $i+1$th digits of n. Furthermore, we say $n = (a_{m-1}a_{m-2}\cdots a_1 a_0)_b$ is the representation of n in base b

Number system

Example (Base 16)

Another often used base in computer science is base 16. By our previous definition, the digits of base 16 numbers ranges from $0 - 15$ and $b = 16$. For example, the number 57 is represented in base 16 as:

$$57 = 3 \cdot 16^1 + 9$$

Hence, $57 = (39)_{16}$

Number system

Example (Base 16) Cont.

What about 31? Note that:

$$31 = 1 \cdot 16^1 + 15$$

Hence $31 = ([1][15])_{16}$. Here I use $[\cdots]$ to represent a digit in base 16. Note that 15 here is treated as a digit in base 16

Number system

Example (Base 16) Cont.

This notation is clumsy, so in practice people use another set of conventions to denote digits from 10 to 15:

Digit	10	11	12	13	14	15
Symbol	A	B	C	D	E	F

Hence, the number $31 = (1F)_{16}$ in this new notation

Number system

Exercise

1. Convert the following numbers from decimal to base 16: 372, 271,31,39,86
2. Convert the following binary numbers to base 16: $(10110101)_2, (111001010110)_2, (0100110010)_2$
3. Convert the following base 16 numbers to binary and decimal: $(1A)_{16}, (43)_{16}, (9E)_{16}, (2D)_{16}, (7B)_{16}$

More on binary numbers

- Since binary numbers are of great importance in computer science, let's discuss more about it
- First, let's introduce some notations

Denoting binary in Python

- In python, we denote a binary number like this : `0b<binary digits>`
- For example, `17 = 0b10001` because
  $17 = 1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 1$
- So in python we writes:

```
x = 0b10001 # same as x = 17
```

Exercise

Convert the following numbers from binary to decimal and check your answer against that by python:

`0b01010, 0b11011, 0b01010, 0b10011`

Denoting hexadecimal in Python

- Similarly we can denote hexadecimal numbers like this: `0x<hex digits>`
- For example, `17=0x11` because $17 = 1 \times 16^1 + 1$
- So in python we write

```
x = 0x11 # same as x = 17
```

Exercise

Convert the following numbers from hexadecimal to decimal and check your answer against that by python:

`0xA31, 0xD13, 0x12C, 0x14`

Bits and Bytes

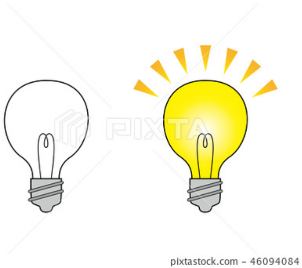
- Using the ideas of binary numbers, we can talk about bit and bytes
- You may think every bits as a 0 or 1 in a binary number
- More generally, we can also use bits to represent binary states
- For example, we can say a game character is dead if the `isAlive` status is `0` and alive if it is `1`
- In the following slides, we will talk about how bits and byte relates to storage capacities

Bit

- **Bit** is the most basic unit of information in computing and digital communications.
- The short hand is **b**
- It represent a **logical state** of *either true or false (0 or 1)*
- Information are stored in computer as an array of bits

Bit

- Obviously, for a *fixed length* of bits, there is a *limited number of states* storable. That define the ”maximum storage capacity” of that array
- For example, consider a storage area of 2 bits (represented by the two light bulbs) on the right
- Since there are only two light bulbs, it follows there can only be 4 states: $(0,0),(0,1),(1,0),(1,1)$



Source: PIXTA

Bit

- In general, suppose we have N bits. Then each of the bits can either be on or off (1 or 0), so there are:

$$\overbrace{2 \times 2 \times \cdots 2 \times 2}^{\text{N times}} = 2^N$$

  states storable in an array of N bits
- This result will be useful when we discuss about overflow and size limits in later chapters
- Just remember for now variables like `int`, `float` cannot store every value in the world, they are limited by the number of states available for a N bit number

Byte

- A **byte** is defined to be 8 bit
- E.g. 4 byte storage is 8x4=32 bit large
- Short form: **B**
- E.g. 16B = 16 byte = 128 bit

Kilo, Mega, Giga and Tera

- In SI units, 1 kilo is 1000
- But in computer science 1 kilo is actually $2^{10} = 1024$
- For example:
- 1 kilobyte (KB) = 1024 B
- 1 megabyte (MB) = 1024 KB
- 1 gigabyte (GB) = 1024 MB
- 1 terabyte (TB) = 1024 GB

Kilo, Mega, Giga and Tera

Exercise

1. A USB drive has a size of 4GB. A typical image has size of 6MB. Roughly how many images can a USB store?
2. James was trying to download a file of 132KB from the internet. He did a speed test on his network and found that is download speed is 57.78Mbps (Mega*bit* per second). Estimate how long is needed for him to download the file.
3. Now he has to upload a file of 0.0001TB through the internet. If his upload speed is 23.69Mbps, how long would it takes to upload the file?