

# HKOI Training

*ami ~ wkc*

Last modified: March 16, 2010

<b>Lecture 03</b>	<b>2</b>
<b>Mathematical reasoning</b>	<b>3</b>
Verification of propositions.....	4
Ambiguity of IF-THEN.....	5
Direct Proof.....	6
Proof by counterexample.....	8
Verifying everything.....	9
Proof by induction.....	10
Well ordering principle.....	11
Idea of induction.....	12
Proof of induction.....	13
<b>Induction example</b>	<b>14</b>
Sum of power one.....	15
Sum of power two.....	16
Summation notation.....	17
Properties of summation.....	18
Direct proof.....	21
Sum of G.S.....	22
<b>Variation of induction</b>	<b>23</b>
Second induction.....	24
Backward induction.....	25
<b>Control flows and algorithm</b>	<b>26</b>
Summation in computer.....	27
Pseudo-code.....	28
<b>C's Syntax</b>	<b>33</b>
Character and String.....	34
Boolean expression.....	35
Comparison expression.....	36
truth value in computer.....	37
Bitwise operation.....	38
XOR.....	39
<b>End</b>	<b>40</b>

## Lecture 03

### Mathematical reasoning.

### Control flows and algorithms.

### More on C.

2 / 40

### Mathematical reasoning

3 / 40

#### Verification of propositions

In last lecture, we discussed about the construction of propositions.

We know the association of the truth values of those further constructed propositions.

However, we did not mention how one could know the truth value of a given proposition.

For example,

- 2 is a prime number.
- Pythagoras theorem is true.<sup>a</sup>
- Angle sum of triangle is always  $180^\circ$ . (What is the meaning of "always" here?)
- The Extended Euclidean algorithm is correct.<sup>b</sup> (Implicitly "always")
- Today is hot. (Another way to verify)
- $\forall x, \sqrt{x^2} = x$ . (Counterexample is demonstrated.)
- If  $n$  is a 5-digit square integer, then  $n = 29929$ .<sup>c</sup> (Mostly yes.)

4 / 40

---

<sup>a</sup>Read the solution of week 1 exercise for complete proof.

<sup>b</sup>A proposition involving recursive definition. It will be proved in number theory.

<sup>c</sup>What is its truth value, when  $n$  is 10, 29929, 10000?

#### Ambiguity of conditional proposition

Let's discuss a conditional proposition with a propositional variable,

$P(n) :=$ "If  $n$  is a 5-digit square integer, then  $n = 29929$ ".

Indeed, "If  $n$  is a 5-digit square integer, then  $n = 29929$ " is not a well-defined proposition. <sup>a</sup>

To discuss the truth value, one need to give a value to  $n$ .

For example,

- $P(2)$  is true as 2 is not a five digit integer.
- $P(10000)$  is false.
- $P(29929)$  is true.

In mathematics, we would usually interpret the sentence " $Q(n) \implies P(n)$ " to be

" $\forall n, Q(n) \implies P(n)$ ".

That's how we say  $P(n)$  is false without explicitly giving a value to  $n$ .

Similarly, the word "always" has the implicit "FOR-ALL" meaning.

5 / 40

---

<sup>a</sup>You cannot tell whether it is true or not by merely reading the  $n$  as a variable

**Direct Proof - Proof from definition :  $n \cdot 0 = 0$** 

How to prove / disprove the proposition " $\forall n, 0 \cdot n = 0$ "?

To verify this above proposition, we need to know about what do we mean by 0.

**Definition.** 0 is the number such that for any  $n$ ,  $n + 0 = n$ .

**Definition.**  $-n$  is the number such that  $n + (-n) = 0$ .

*Proof.* We use the distributive law and the definition of 0,

$$n \cdot 0 = n \cdot (0 + 0) = n \cdot 0 + n \cdot 0$$

Adding  $-(n \cdot 0)$  to both side, we get  $n \cdot 0 = 0$ . □

This proof seems meaningless, however, remember what properties we used.

Later on, if there is another kind of arithmetic system also having the same properties, we can claim that the above still holds.

For example, the modular arithmetic, i.e. doing arithmetic on a clock.

6 / 40

**Direct Proof - Proof from definition :  $(-1)(-1) = 1$** 

How to prove / disprove the proposition " $(-1)(-1) = 1$ "?

**Definition.** 1 is the number such that for any non-zero  $n$ ,  $1 \cdot n = n$ .

*Proof.*  $(1 + (-1))^2 = 1^2 + 1 \times (-1) + (-1) \times 1 + (-1)(-1)$

Using the definition of 1, we have,  $1^2 = 1$ ,  $1 \times (-1) = -1$  and  $(-1) \times 1 = -1$ .

From the definition of  $-n$ , we have  $(1 + (-1)) = 0$  and  $(1 + (-1))^2 = 0 \cdot 0 = 0$ .

Therefore, we have  $0 = 1 + (-1) + (-1) + (-1)(-1)$ .

Evaluating from left to right,  $1 + (-1)$  becomes 0.

This gives,  $0 = (-1) + (-1)(-1)$ .

Finally, adding 1 to the left of both sides, we have  $1 = 1 + (-1) + (-1)(-1)$

Therefore,  $1 = (-1)(-1)$ . □

We will come back to these proofs at Abstract Algebra.

7 / 40

### Proof by example / counterexample

Usually, to deal with propositions with FOR-ALL, one can prove its falsity by showing a counterexample.

"Every people eat meat", this is false as we can find a vegetarian.

Similarly, one could show the correctness of a THERE-EXISTS statement by giving a particular example.

"There is a pair of integers  $x$  and  $y$  such that  $7x + 9y = 1$ ".

$7(-5) + 9(4) = 1$ , hence the above proposition is true.

8 / 40

### Verifying everything

However, to prove the correctness of a FOR-ALL statement, or the falsity of a THERE-EXISTS statement, we need another way.

For example, the following statements cannot be proved easily in logic

- Every horses have four legs. <sup>a</sup>
- There is no any extraterrestrial life. <sup>b</sup>

9 / 40

---

<sup>a</sup>You need to grab all the horses and observe one by one

<sup>b</sup>You need to observe everywhere outside the Earth

## Proof by induction

Given the **domain of discourse** is the positive integers.

One can use a tool called **Mathematical Induction** to prove a FOR-ALL statement.

Let  $P(n) ::= "1 + 2 + \dots + n \text{ is always equal to } \frac{1}{2}n(n + 1)"$ .

We can check  $P(1), \dots, P(100)$  one by one.

- $1 = \frac{1}{2}(1)(2)$
- $1 + 2 = 3 = \frac{1}{2}(2)(3)$

However, we still don't know whether  $P(101)$  is true or not.

Even you can check the proposition up to 100000,

there are still infinitely many propositions to be verified.

To verify all of them, we cannot check one by one.<sup>a</sup>

10 / 40

---

<sup>a</sup>You may use a direct proof to show

## Well ordering principle

**Axiom** (Well ordering principle). *Let  $S$  be a collection of positive integers.*

*If  $S$  is non-empty, then it has the smallest integer.*

*i.e. Every non-empty collection of positive integers has the smallest integer.*

The above axiom says that

you can always find the smallest integer whenever you have some integers.

For example, we can mention about

- the smallest positive even number.
- the smallest positive odd number.
- the smallest common multiple of 15 and 20.
- the smallest positive prime.

11 / 40

## Idea of induction

Assume the **domain of discourse** is the positive integers.

**Theorem** (Mathematical induction). Let  $P(n)$  be a proposition with propositional variable  $n$ . Suppose the following:

1.  $P(1)$  is true.
2.  $\forall k, P(k) \implies P(k + 1)$  is true.

Then  $\forall k, P(k)$  is true

Therefore, to show that a FOR-ALL proposition, one can check  $P(1)$  is true and prove the second conditional statement.

The first condition is called the **base case**.

The second condition is called the **induction step**.

12 / 40

## Proof of induction

*Proof.* Let  $S$  be the collection of positive integers  $n$  such that  $P(n)$  is false.

we want to show that this collection is empty.

Let's see what happen if it is non-empty.

Suppose it is non-empty, the well-ordering principle tells that the smallest integer exists.

i.e. There is a smallest integer  $m$  such that  $P(m)$  is false.

Denote that integer by  $m$  and clearly  $m \neq 1$  since  $P(1)$  is true.

Therefore,  $m > 1$  and  $m - 1$  is again a positive integer.

Since  $m$  is the smallest one such that  $P(m)$  is false, we have  $P(m - 1)$  is true.

Finally, the second part of our assumption, tells that  $P(m - 1) \implies P(m)$  is true.

Since  $P(m - 1)$  is true, we must have  $P(m)$  be true. (contradiction, as  $P(m)$  is false.)

Therefore, the very most assumption of " $S$  is non-empty" is false.

Hence, there is no any positive integer  $n$  such that  $P(n)$  is false.

" $\forall n, P(n)$ " is logically equivalent to " $\sim \exists n, \sim P(n)$ "

Therefore,  $\forall n, P(n)$  is true. □

The above argument works for any propositions as long as it satisfy our assumption.

Usually, after we checked the assumption holds, we can use this theorem to conclude.

By "using a theorem", we actually means that we re-apply the arguments of a proof again.

13 / 40

**Sum of power one**

Let  $P(n) ::= "1 + 2 + \dots + n$  is always equal to  $\frac{1}{2}n(n + 1)"$ .  
 we shall show that " $\forall n, P(n)$ " by using the mathematical induction.

*Proof.*

Firstly, we check that  $P(1)$  is true, which is obvious in this case.

Secondly, for each given  $k$ ,

we want to show that the proposition  $P(k) \implies P(k + 1)$  is also true.

To prove a conditional proposition always holds,

we need to check all the possible truth value associated between the two statements.

If  $P(k)$  is false, then the whole conditional statement is true.

Remaining is to verify the case for  $P(k)$  is true to show  $P(k + 1)$  is true.

i.e.  $1 + 2 + \dots + k = \frac{1}{2}k(k + 1)$  is true.

Now, consider  $1 + 2 + \dots + k + (k + 1) = (1 + 2 + \dots + k) + k + 1$ .

Our assumption says that  $(1 + 2 + \dots + k) + k + 1 = \frac{1}{2}k(k + 1) + (k + 1)$

$\frac{1}{2}k(k + 1) + (k + 1) = \frac{1}{2}(k + 1)((k + 1) + 2)$

Hence,  $1 + 2 + \dots + k + (k + 1) = \frac{1}{2}(k + 1)((k + 1) + 2)$ , i.e.  $P(k + 1)$  is true.

By induction, we know that  $\forall k, P(k)$  is true.

□

Try to show the above proposition using a direct proof.

**Sum of power two**

Let  $P(n) ::= "1^2 + 2^2 + \dots + n^2$  is always equal to  $\frac{1}{6}n(n + 1)(2n + 1)"$ .

*Proof.*

Firstly, we check that  $P(1)$  is true, which is obvious in this case.

Secondly, for each given  $k$ ,

assume  $P(k)$  is true, i.e.  $1^2 + 2^2 + \dots + k^2 = \frac{1}{6}k(k + 1)(2k + 1)$ .

Now, consider  $1^2 + 2^2 + \dots + k^2 + (k + 1)^2 = (1^2 + 2^2 + \dots + k^2) + (k + 1)^2$ .

Hence,  $1^2 + 2^2 + \dots + k^2 + (k + 1)^2 = \frac{1}{6}(k)(k + 1)(2k + 1) + (k + 1)^2$

$\frac{1}{6}(k)(k + 1)(2k + 1) + (k + 1)^2 = \frac{1}{6}(k + 1)(k(2k + 1) + 6(k + 1))$

$k(2k + 1) + 6(k + 1) = 2k^2 + 7k + 6 = (2k + 3)(k + 2)$

Hence,  $\frac{1}{6}(k + 1)(k(2k + 1) + 6(k + 1)) = \frac{1}{6}(k + 1)(k + 2)(2k + 3)$ .

$\frac{1}{6}(k + 1)(k + 2)(2k + 3) = \frac{1}{6}(k + 1)((k + 1) + 1)(2(k + 1) + 1)$

Hence,  $P(k + 1)$  is true.

By induction,  $\forall k, P(k)$  is true.

□

How can one come up with the expression at the right hand side?

## Summation notation

Let  $f(i)$  be a function of  $i$ , e.g.  $\sin$ ,  $i^2$ .

and  $n, m$  be two integers with  $n < m$ .

The summation notation  $\sum_{i=n}^m f(i)$  is defined as  $f(n) + f(n+1) + \dots + f(m)$ .<sup>a</sup>

Literally, it means keep adding from  $i = n$  up to  $i = m$ .

For example,

- $\sum_{i=1}^m i$  is the same as  $1 + 2 + \dots + m$ .
- $\sum_{i=3}^m i^2$  is the same as  $3^2 + 4^2 + \dots + m^2$ .

The previous two identities can be written as

- $\sum_{i=1}^m i = \frac{1}{2}m(m+1)$ .
- $\sum_{i=1}^m i^2 = \frac{1}{6}m(m+1)(2m+1)$ .

17 / 40

---

<sup>a</sup>This is just a notation to simplify our mathematical expression.

## Properties of summation

Let  $f, g$  be functions,  $n, m$  be integers with  $n < m$  and  $c$  be a constant.

**Theorem.** *The following are true:*

1. If  $n \leq k < m$ , then  $\sum_{i=n}^m f(i) = \sum_{i=n}^k f(i) + \sum_{i=k+1}^m f(i)$ .
2.  $\sum_{i=1}^n c = nc$  (adding total  $n$ 's  $c$  is  $nc$ .)
3.  $\sum_{i=n}^m c = (m - n + 1)c$
4.  $\sum_{i=n}^m c \cdot f(i) = c \cdot \sum_{i=n}^m f(i)$
5.  $\sum_{i=n}^m [f(i) + g(i)] = \left( \sum_{i=n}^m f(i) \right) + \left( \sum_{i=n}^m g(i) \right)$

The first: "adding from  $n$  to  $m$  is the same as adding from  $n$  to  $k$  then from  $k+1$  to  $m$ ".

The fourth:  $(cf(n) + cf(n+1) + \dots + cf(m)) = c(f(n) + f(n+1) + \dots + f(m))$

18 / 40

## Properties of summation

The fifth:

$$(f(n) + g(n)) + (f(n+1) + g(n+1)) + \cdots + (f(m) + g(m)) = (f(n) + f(n+1) + \cdots + f(m)) + (g(n) + g(n+1) + \cdots + g(m))$$

i.e. we can rearrange the terms as long as there are finitely many.

*Proof.* Let  $P(n)$  be the proposition that " $\sum_{i=1}^n c = nc$ ".

$P(1)$  is true, as adding from  $m$  to  $m$  means added once only.

$$\text{Assume } \sum_{i=1}^k c = kc, \text{ then } \sum_{i=1}^{k+1} c = \sum_{i=1}^k c + \sum_{i=k+1}^{k+1} c = kc + c = (k+1)c$$

Hence,  $P(n)$  is true for all  $n$ .

To prove that  $\sum_{i=n}^m c = (m-n+1)c$ , we used the fact  $P(m)$  and  $P(n-1)$  is true.

$$\text{From the first one, we have } \sum_{i=1}^m c = \sum_{i=1}^{n-1} c + \sum_{i=n}^m c.$$

$$\text{Hence, } mc = (n-1)c + \sum_{i=n}^m c \iff \sum_{i=n}^m c = (m-n+1)c. \quad \square$$

To proof the next identity, one can similarly do an induction on  $\sum_{i=1}^m c \cdot f(i)$ .

On the next slide, we shall prove the final one.

19 / 40

## Properties of summation

*Proof.* Let  $P(n)$  be the proposition that

$$" \sum_{i=1}^n [f(i) + g(i)] = \left( \sum_{i=1}^n f(i) \right) + \left( \sum_{i=1}^n g(i) \right) "$$

$P(1)$  is true, as LHS is  $[f(1) + g(1)]$  and the right is  $(f(1)) + (g(1))$ .

Assume  $P(k)$  is true, then

$$\sum_{i=1}^{k+1} [f(i) + g(i)] = \sum_{i=1}^k [f(i) + g(i)] + [f(k+1) + g(k+1)]$$

$$\sum_{i=1}^{k+1} [f(i) + g(i)] = \left( \sum_{i=1}^k f(i) \right) + \left( \sum_{i=1}^k g(i) \right) + [f(k+1) + g(k+1)]$$

$$\sum_{i=1}^{k+1} [f(i) + g(i)] = \left( \sum_{i=1}^k f(i) + f(k+1) \right) + \left( \sum_{i=1}^k g(i) + g(k+1) \right)$$

$$\sum_{i=1}^{k+1} [f(i) + g(i)] = \left( \sum_{i=1}^{k+1} f(i) \right) + \left( \sum_{i=1}^{k+1} g(i) \right)$$

Hence,  $P(n)$  is true for all  $n$ .

Finally, the part from  $n$  to  $m$  can be completed by consider 1 to  $n-1$  and 1 to  $m$ .

Note that, the rearrangement always holds if we have  $f(i) + g(i) + h(i)$  and more functions. □

20 / 40

### The direct proof for power sum of two

Let  $f(n) = (n+1)^3 - n^3 = 3n^2 + 3n + 1$  and consider  $\sum_{i=1}^n f(i)$ .

Viewing  $f(n)$  as the difference of two numbers, we have

$$\sum_{i=1}^n ((i+1)^3 - i^3) = \sum_{i=1}^n (i+1)^3 - \sum_{i=1}^n i^3$$

It is equal to  $\sum_{i=2}^{n+1} i^3 - \sum_{i=1}^n i^3 = (n+1)^3 - 1$

On the other hand,  $\sum_{i=1}^n (3i^2 + 3i + 1) = \sum_{i=1}^n (3i^2) + \sum_{i=1}^n (3i) + \sum_{i=1}^n (1)$

Therefore,  $(n+1)^3 - 1 = \sum_{i=1}^n (3i^2) + \sum_{i=1}^n (3i) + \sum_{i=1}^n (1)$

Rearrange the terms, we have  $\sum_{i=1}^n (3i^2) = (n+1)^3 - 1 - \sum_{i=1}^n (3i) - \sum_{i=1}^n (1)$ .

Hence,  $3 \sum_{i=1}^n i^2 = (n+1)^3 - 3 \sum_{i=1}^n (i) - (n+1)$ .

Using the identity for  $1 + 2 + \dots + n = \frac{1}{2}n(n+1)$ , we have

$$3 \sum_{i=1}^n i^2 = (n+1)^3 - \frac{3}{2}n(n+1) - (n+1)$$

factorize gives,  $\frac{1}{2}(n+1)(2n^2 + 4n + 2 - 3n - 2) = \frac{1}{2}(n+1)(n(2n+1))$

Divide both side by 3, we have  $\sum_{i=1}^n i^2 = \frac{1}{6}n(n+1)(2n+1)$ .

21 / 40

### Sum of geometric sequence

$$\sum_{i=0}^n x^i = \frac{x^{n+1}-1}{x-1}, \text{ i.e. } 1 + x + x^2 + \dots + x^n = \frac{x^{n+1}-1}{x-1}$$

*Proof.* Similarly, we let  $P(n)$  be the above proposition.

Clearly,  $P(0)$  and  $P(1)$  are true.

For the induction step, assume  $P(k)$  is true,

$$\text{then } \sum_{i=0}^{k+1} x^i = \frac{x^{k+1}-1}{x-1} + x^{k+1} = \frac{x^{k+2}-1}{x-1}$$

Hence, by MI, the above  $P(n)$  is true for all  $n$ . □

Alternatively,

$$\text{Proof. } x \sum_{i=0}^n x^i - \sum_{i=0}^n x^i = (x-1) \sum_{i=0}^n x^i$$

On the other hand,  $x \sum_{i=0}^n x^i = \sum_{i=0}^n x^{i+1} = \sum_{i=1}^{n+1} x^i$ .

Hence,  $x \sum_{i=0}^n x^i - \sum_{i=0}^n x^i = x^{n+1} - 1$ .

Therefore,  $x^{n+1} - 1 = (x-1) \sum_{i=0}^n x^i$ . □

Find  $1 + \frac{1}{2} + \frac{1}{4} + \dots + \frac{1}{32768}$  using the above identity.

22 / 40

**Second induction**

The base case becomes " $P(1)$  and  $P(2)$ ".

The induction step becomes  $P(k)$  and  $P(k+1) \implies P(k+2)$ .

Define  $F_0 = 0, F_1 = 1$  and  $F_n = F_{n-1} + F_{n-2}$  for  $n > 1$ .

Let  $r_1 = \frac{1+\sqrt{5}}{2}, r_2 = \frac{1-\sqrt{5}}{2}$ , prove that  $F_n = \frac{r_1^n - r_2^n}{\sqrt{5}}$ .

*Proof.* Use the second induction,

we let the proposition  $P(n)$  to be  $F_{n-1} = \frac{r_1^{n-1} - r_2^{n-1}}{\sqrt{5}}$ .

Prove that  $r_1 + r_2 = 1$  and show the induction step.

Left as exercise. □

24 / 40

**Backward induction proof of the AGM-inequality**

**Theorem.** Let  $a_1, a_2, \dots, a_n$  be  $n$  non-negative numbers. Then

$$\frac{a_1 + a_2 + \dots + a_n}{n} \geq \sqrt[n]{a_1 \cdot a_2 \cdot \dots \cdot a_n}$$

*Proof.* We proved that  $P(2)$  is true and want to show that  $P(2^n)$  is true.

Using the induction for  $P(2^k) \implies P(2^{k+1})$ , we have  $\forall n, P(2^n)$ .

Finally, we showed that  $\forall k, P(k) \implies P(k-1)$ .

We starts from  $2 \implies 4 \implies 8 \implies 16 \implies \dots$

then  $4 \implies 3, 8 \implies 7 \implies 6 \implies 5, \dots$

The detail are left as exercise. □

Be familiar with the mathematical induction, we will use it quite often.

25 / 40

**Summation in computer**

We introduce the summation notation for summing up terms.

However, there is no such direct concept for computer.

We should use the basic things that a computer can do to simulate the summation

Remember that a computer can do looping.

A **control flow** is a statement which tells the computer what it should perform.

We shall express these ideas in pseudo-code.

An **algorithm** is a sequence of operations to tell the computer how to solve a problem.

27 / 40

**Pseudo-code - Assignment**

Pseudo-code is a hand-written notation to specify how should a computer to perform a task.

The code is merely for human being.

It represents what the computer can do.

In pseudo-code, we denote " $x \leftarrow 1$ " to represent storing 1 to  $x$ .

This is called an **assignment**.

In general, the grammar structure for an assignment expression is

$\langle Assignment \rangle ::= \langle Name \rangle \leftarrow \langle Value \rangle$

28 / 40

## IF-THEN / IF-THEN-ELSE

IF-THEN and IF-THEN-ELSE are control flow which have two and three parts respectively. They tell the computer to perform a task based on a given condition.

1. The condition for checking.
2. The actions to be performed once the condition holds.
3. The actions to be performed once the condition is false. (Only for IF-THEN-ELSE)

The only difference between them is that, when the condition does not hold, the IF-THEN-ELSE tells the computer to do some task. This is called a **conditional flow**.

Usually, we denote it by "if *condition* then *action list*".  
or "if *condition* then *action list* else *action list*".

In general, the grammar structure for a conditional flow is

$\langle \text{IF-THEN} \rangle ::= \text{if } \langle \text{Condition} \rangle \text{ then } \langle \text{Actions-for-true} \rangle$

$\langle \text{IF-THEN-ELSE} \rangle ::= \text{if } \langle \text{Condition} \rangle \text{ then } \langle \text{Actions-for-true} \rangle \text{ else } \langle \text{Actions-for-false} \rangle$

29 / 40

## Example - Condition flow

You are given a positive integer  $x$ , we want to find the next even number.

How can we tell the computer to find the next even number?

The following is an algorithm.

**Data:**  $x$  is a positive integer.

**Result:** The even number just after  $x$ .

```
if  $x$  is odd then
| return  $x + 1$ 
else
| return  $x + 2$ 
end
```

How can we check if  $x$  is odd?

**Data:**  $x$  is a positive integer.

**Result:** The even number just after  $x$ .

```
if  $x \bmod 2 \neq 0$  then
| return  $x + 1$ 
else
| return  $x + 2$ 
end
```

30 / 40

## Looping - Iterative

In the summation notation, we start from  $i = n$  and then count one by one until  $i = m$ .

A similar control flow called the FOR-LOOP also exists in computer.

It consists of three parts,

1. The starting/initial value  $n$
2. The ending value  $m$
3. Actions to be performed.

It is usually called a **for-loop** or an **iterative loop**.

The grammar is  $\langle \text{FOR-LOOP} \rangle ::= \text{for } \langle \text{name} \rangle \leftarrow \langle \text{value} \rangle \text{ to } \langle \text{value} \rangle \text{ do } \langle \text{Actions} \rangle$ .

Therefore, the summation notation  $\sum_{i=1}^{10} i^2$  can be simulated as,

```
t ← 0 ;
for i ← 1 to 10 do
  | t ← t + i2
end
return t
```

31 / 40

## Looping - Conditional

Usually, we would like to repeat a task whenever the condition holds.

For example, in checking password,

the computer keeps asking for a password until a correct one is entered.

A similar control flow called the WHILE-LOOP also exists in computer.

It consists of two parts,

1. The condition  $n$
2. Actions to be performed.

It is usually called a **while-loop** or a **conditional loop**.

The grammar is  $\langle \text{WHILE-LOOP} \rangle ::= \text{while } \langle \text{condition} \rangle \text{ do } \langle \text{Actions} \rangle$ .

Therefore, a password checking algorithm is as follow,

**Input:** Password

**while** *Password is wrong* **do**

  | **Output:** You enter a wrong password.

  | **Input:** Password.

**end**

**Output:** Login successfully.

32 / 40

### Character and String

Previous lecture, we know that a computer can recognize value.

A computer integer is of the form [SIGN]?[0-9]+.

A computer real number is given similarly.

These two are called **numerical values**.

Another type of values the computer can recognize is "English".

It can be divided into two types.

A "letter" or a "vocabulary".

The computer-letter is called a **character**.

The computer-vocabulary is called a **string**.

Unlike English, we do not impose restriction on whatever we called a string.

There is no spell checking.

A character in C's syntax is enclosed by two single quotation mark.

For example, 'a', 'b', ' '.

A string is enclosed by two double quotation mark.

For example, "abc", "hot".

As similar to the regular expression, whenever a character has special meaning, use a \.

Therefore, to denote a character single quote, we should use '\ '.

And similar, a string with a double quote, we should use "asd\"".

34 / 40

### Boolean expression

A computer also knows logic and propositions.

Given a proposition,

it create new propositions through the three logical operators (NOT, AND, OR).

In C's syntax, the three are expressed as follow:

! The logical NOT operator.

&& The logical AND operator.

|| The logical OR operator.

Since " $P \implies Q$ " is equivalent to " $\sim P$  or  $Q$ ".

Using the above three operators, it is enough to construct the IF-THEN and IFF propositions.

The remaining is to express comparisons in C's syntax.

For example,  $n \geq 2$ ,  $n > 2$ ,  $n = 2$ ,

**Remark.** Be careful, it is necessary to write two symbols for the logical AND, OR operators.

35 / 40

## Comparison expression

A computer can do comparison about numbers only.

It knows the following mathematical relations:

> Greater than.

>= Greater than or equal to.

< Less than.

<= Less than or equal to.

!= Not equal to.

== Equal to.

For example, the statement “ $n$  is greater than 4 and less than 10” can be expressed as

$n >= 4 \&\&n < 10$ .

**Remark.** *Be careful, it is necessary to write two symbols for the “Equal to” comparison.*

36 / 40

## Representation of truth value in computer

As mentioned, computer can only represent two essentially different types.

1. Numbers: Computer-integers and Computer-real-numbers.
2. Text: Strings<sup>a</sup>.

Therefore, there is no any other types in the computers.

The truth value, false and true, are represented using either one of the above type.

Indeed, the computer use numbers to represent truth value.

It chooses 0 to represent false and any other non-zero numbers are true, usually 1.

37 / 40

---

<sup>a</sup>In a computer, a character is actually represented as an integer

## Bitwise operation

We mentioned that a logical AND , OR are represented using && and ||.

What happens if we use a single & and | instead?

What is the meaning of 3^5 in C language?

The above three operators are called **bitwise operators**

The bitwise one acts like the logical one, except it operates on every binary digits.

For example, 10 bitwise-or 6 is performed as follow:

1. The binary representation of 10 is  $1010_{(ii)}$
2. The binary representation of 6 is  $0110_{(ii)}$
3. Do the Logical OR operation on digit by digit.
4. Therefore, the result is  $1110_{(ii)} = 14$

Similarly for the bitwise-and operation.

Hence, in C's syntax, the result of  $10 \& 6$  is  $0010_{(ii)} = 2$ .

**Remark.** *There is bitwise NOT operation.*

*However, due to its complexity, we will postpone its discussion.*

38 / 40

## Exclusive OR

The ^ symbol is called the exclusive or, usually denote by (XOR), which means "only one of them is true, but not both".

The truth table for XOR is

$P$	$Q$	$P \text{ XOR } Q$
true	true	false
true	false	true
false	true	true
false	false	false

In other words, it is more or less the same as "logically not equal to".

The computation of the result is similar to the other two bitwise operations.

Therefore, the expression  $3^5$  in C's syntax is computed as follow:

1. The binary representation of 3 is  $011_{(ii)}$
2. The binary representation of 5 is  $101_{(ii)}$
3. Do the XOR operation on digit by digit.
4. Therefore, the result is  $110_{(ii)} = 6$

39 / 40

