COL 100 : L1 notes

- Computer Science fundamentally involves computing!

Note: Computing can be performed even w/o Computers. → mechanism to perform computation

- Computation $\xrightarrow[\text{follow}]{\text{we}}$ A sequence of unambiguous rules → This explicitly written down set of rules is called an algorithm

Eg: Addition, GCD .... [Mathematics]

Bisecting an angle, ... [Geometry, Art...]

Chemical reaction & reaction rate .... [Chem.]

Homeostasis ?... [Bio, Med. ...]

- Let us take some examples
  - Computing tools :
    - abacus, sticks & stones, paper & pen, ...
    - Straight Edge & Compass

How is it a computing tool?

Unmarked : only specify lines ; can't measure lengths

Q: Given a square construct another square of twice the area of the original square.

A: Assume $\square ABCD$ of side $a > 0$

Recipe :

- Draw the diagonal AC
- Draw a square with AC as an edge

Justification

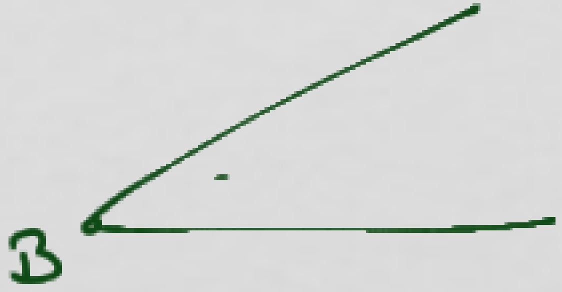
$AC = \sqrt{2}\, a$

Area of $\square ACEF = 2a^2$

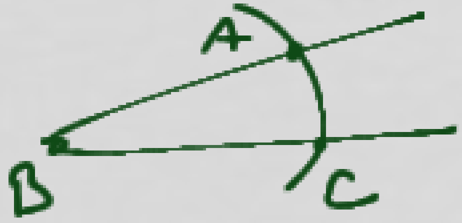
Q. Bisect a given angle using straight-edge & Compass

A: Assume a point B is given as the vertex of the angle

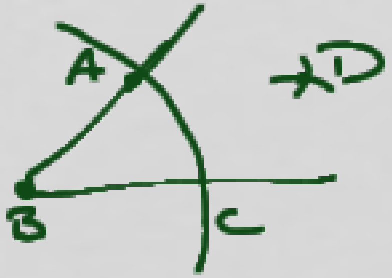
Vitte



- Place compass on B, stretch it to any length that will stay ON the angle
- Draw an arc; two intersection pts. are obtained



- Place the compass on A & C and draw arcs in the interior of the angle



- Connect the vertex D with B s.t. $\angle ABD = \angle CBD$

Justification

- Congruent Δs
$\Rightarrow$ equal areas
$\Rightarrow$ equal angles

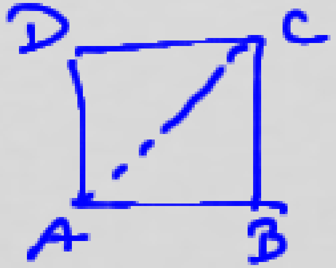
Limitations of Computing tools

Eg: With

- straight edge + compass we cannot trisect an angle.

Revisit the square example

– Draw a square with AC as an edge



Not a $\boxed{\text{primitive}}$ step

something that can't be broken down further

Observations

1. Every computing tool has a $\boxed{\text{"model" of computation}}$

Q: what is ?

A: Model of computation gives us a framework to organize our ideas about processes

i.e: 3 mechanisms

1 Primitive operation and expressions

→ simplest entities of the computation

Eg: Nat, +

2. Method of combination

→ means of building complex & richer expressions

3. Methods of abstraction

→ how compound exprs can be named & manipulated as units

Eg:
- Perpendicular at a pt on a line was used to construct a square
- Sq. Construction alg to construct a sq. on a line segment

Use of abstraction

→ separates logical subproblems

→ avoiding repetitions of copies of similar solutions

2. Based on these primitives, the computational model's power or expressivity is defined

Turing's Result [with just 6 primitive expressions] can compute any "computably enumerable" problem

⇒ As a result power of (C) = power (Java)
= power (Python)
= power (SML) ....

- But based on particular combinations & abstraction their suitability for a task may differ

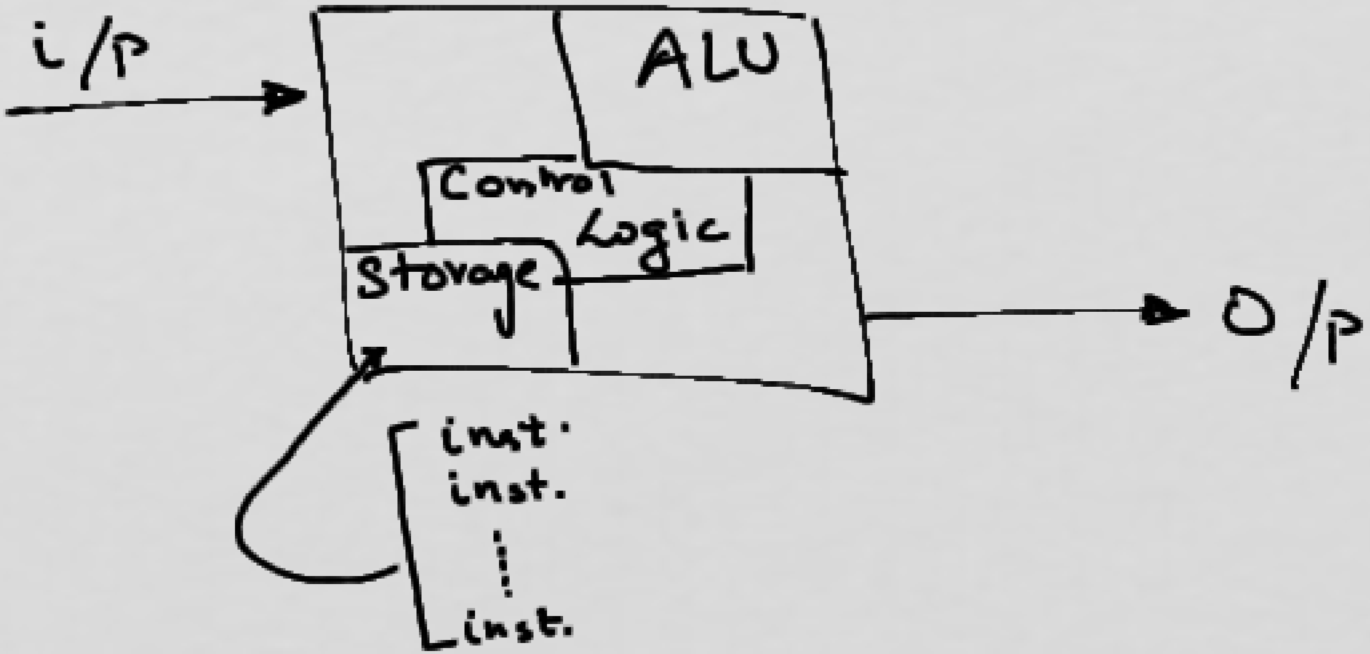
for instance:

MATLAB: Excellent for vectors, matrices

C : for data n/ws

LISP/SML: for arbitrary structured data

Our Computing Tool



- Definite i/p and o/p
- Unambiguous
- Specifies solution as a finite process

$\Rightarrow$ # of steps in the computation is finite

Programming Language

Language via which communication with m/c is possible

Program : Algorithm written in a prog. language

[well defined grammar -- syntax]

Computing models

- functional (declarative) : mathematical Expressions
- Imperative : sequence of Commands.

Eg: Compute $y = \sqrt{x}$

s.t. $y^2 = x$

Mathematical Defn.

- Take a guess g
- Compute g^2
- check if $g^2 \sim x$
- If yes $\longrightarrow$ return g, stop
- else
  new guess $= (g + g/x)$
  repeat.

Imperative

Our model of computation (Functional model)

→ Closeness to mathematics makes it convenient to use

- **Primitive Expressions**
  - Constants, Variables and functions

Eg: $\mathbb{N}, \mathbb{Z}, \mathbb{R}, \mathbb{B}$ | x, y, z | $\mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$

Identifiers to data objects

$\mathbb{N} \times P \rightarrow \mathbb{N}$

$\vdots$

  - Basic operations: addition, subtraction, division, multiplication, boolean operations, ... etc.

  - Naming mechanism for various entities to be used w/o repeating definitions

Eg: Square (n) = n*n

- **Methods of Combination**
  - Composition of functions

    Eg: sum-sq (x,y) = square(x) + square(y)
  - $\boxed{\text{Inductive definitions}}$ why ??

    Eg: factorial

- **Methods of abstraction**
  - Naming & using groups of objects and expressions as a single unit

Eg: **Factorial**

Math notation

$$n! = \begin{cases} 1 & \text{if } n < 1 \\ 1 \times 2 \times \ldots \times n & \text{otherwise} \end{cases}$$

Inductive Variant

$$n! = \begin{cases} 1 & \text{if } n < 1 \\ n \times (n-1)! & \text{otherwise} \end{cases}$$

```
fun fact n = if n < 1 then 1 else n * fact (n-1);
    val fact = fn: int -> int
  - fact 4;
    val it = 24 : int
```

→ Evaluation has taken place

1. Vars $\xrightarrow{\text{to}}$ values
   fn names $\xrightarrow{\text{with}}$ definitions
   args $\xrightarrow{\text{with}}$ actual arg.
2. Order left to right
3. Priority of operators

Unless bracketing & association of operators tell otherwise

Computation

$$
\begin{aligned}
3! &= 3 \times (3-1)! \\
&= 3 \times 2! \\
&= 3 \times (2 \times (2-1)!) \\
&= 3 \times (2 \times 1!) \\
&= 3 \times (2 \times (1 \times (1-1)!)) \\
&= 3 \times (2 \times (1 \times 0!)) \\
&= 3 \times (2 \times (1 \times 1)) \\
&= 3 \times (2 \times 1) = 3 \times 2 = 6
\end{aligned}
$$

Q. How about

$$n!_0 = \left\{ \begin{array}{ll} 1 & \text{if } n < 1 \\ (n+1)! / (n+1) & \text{Otherwise} \end{array} \right\}$$

Mathematically correct but computationally incorrect

$$3!_0 = (3+1)! / (3+1)$$

$$= 4!/4$$

$$= \left((4+1)! / 4+1 \right) / 4$$

$$= (5!/5)/4$$

$$\vdots$$

Observation

- Algorithms should guarantee that all its computation terminate
- Not all mathematical functions are algorithms

Another Eg:

$$Sqrt(x) = \left\{ \begin{array}{ll} y & \text{if } y * y = x \\ 0 & \text{if } \nexists y : y * y = x \end{array} \right\}$$

Another Eg:

$$\text{Sqrt}(x) = \begin{cases} y & \text{if } y*y = x \\ 0 & \text{if } \nexists y : y*y = x \end{cases} \longrightarrow$$

No Description of "how to evaluate"

SML Intro

− 5 $\boxed{;}$ ⟶ terminate the i/p

val it = 5 : int

− $\boxed{\text{val}}$ x = 5;

⟶ Keyword: define a new name