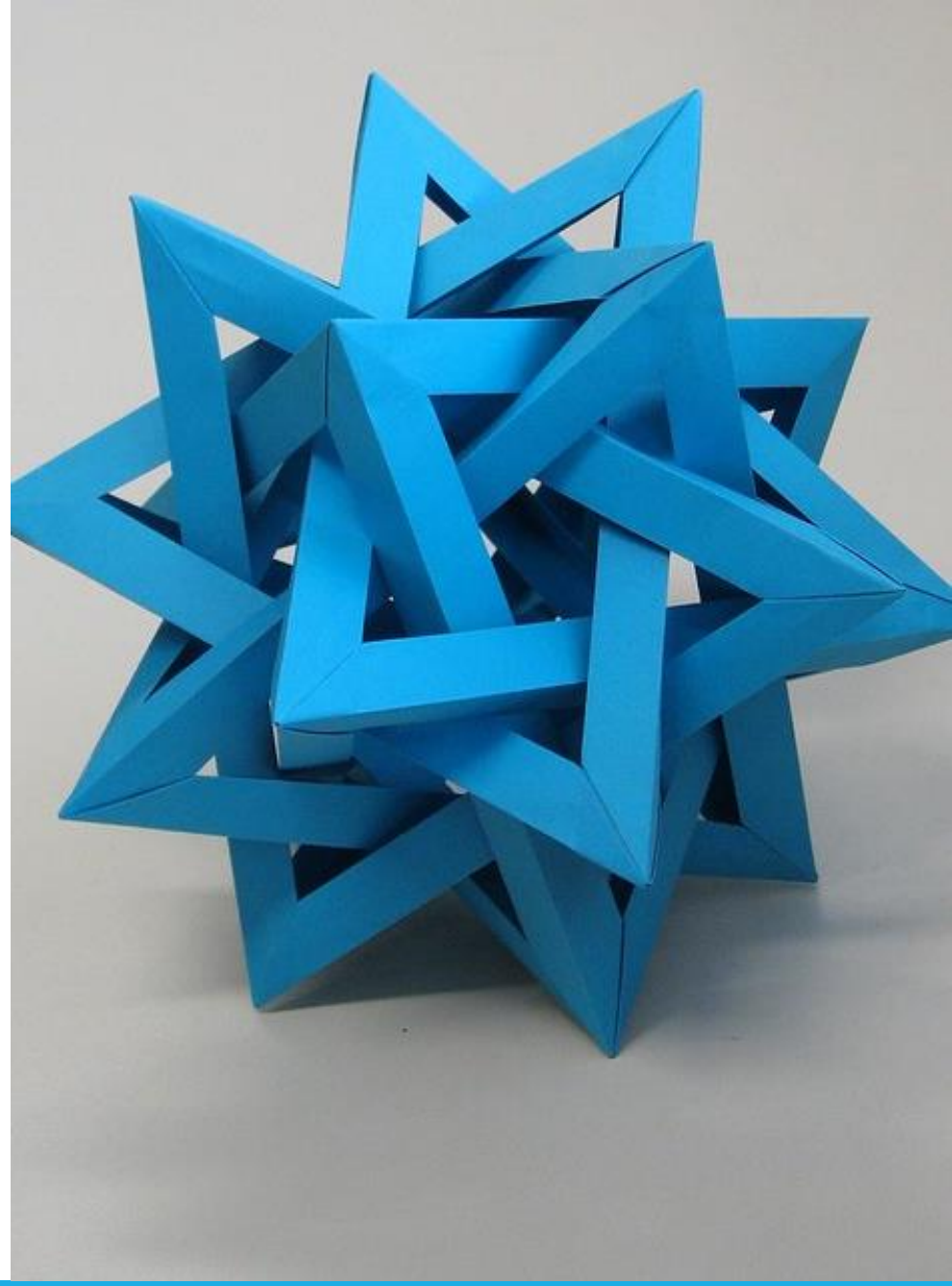
POLITECNICO
DI TORINO

Dipartimento
di Automatica e Informatica

Unit P1: Introduction to programming

A SHORT INTRODUCTION TO HARDWARE, SOFTWARE, AND ALGORITHM DEVELOPMENT

Chapter 1



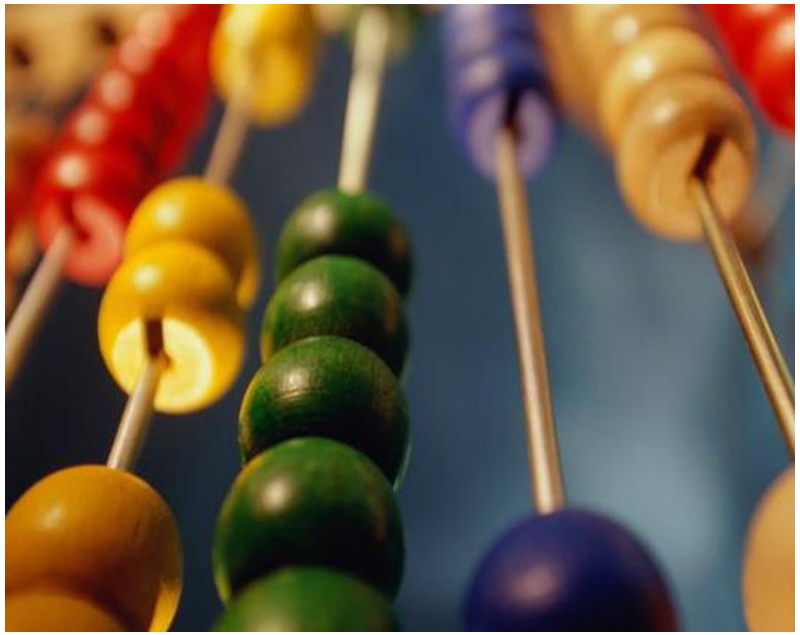
Unit P1: Goals

- Introduction to computers and programming
  - About computer hardware, software and programming
  - How to test, find and fix programming errors
  - How to use pseudocode to describe an algorithm
- Flow Charts as a support for problem solving
  - Representation
  - Design steps
- Introduction to Python
  - Tools
  - Language

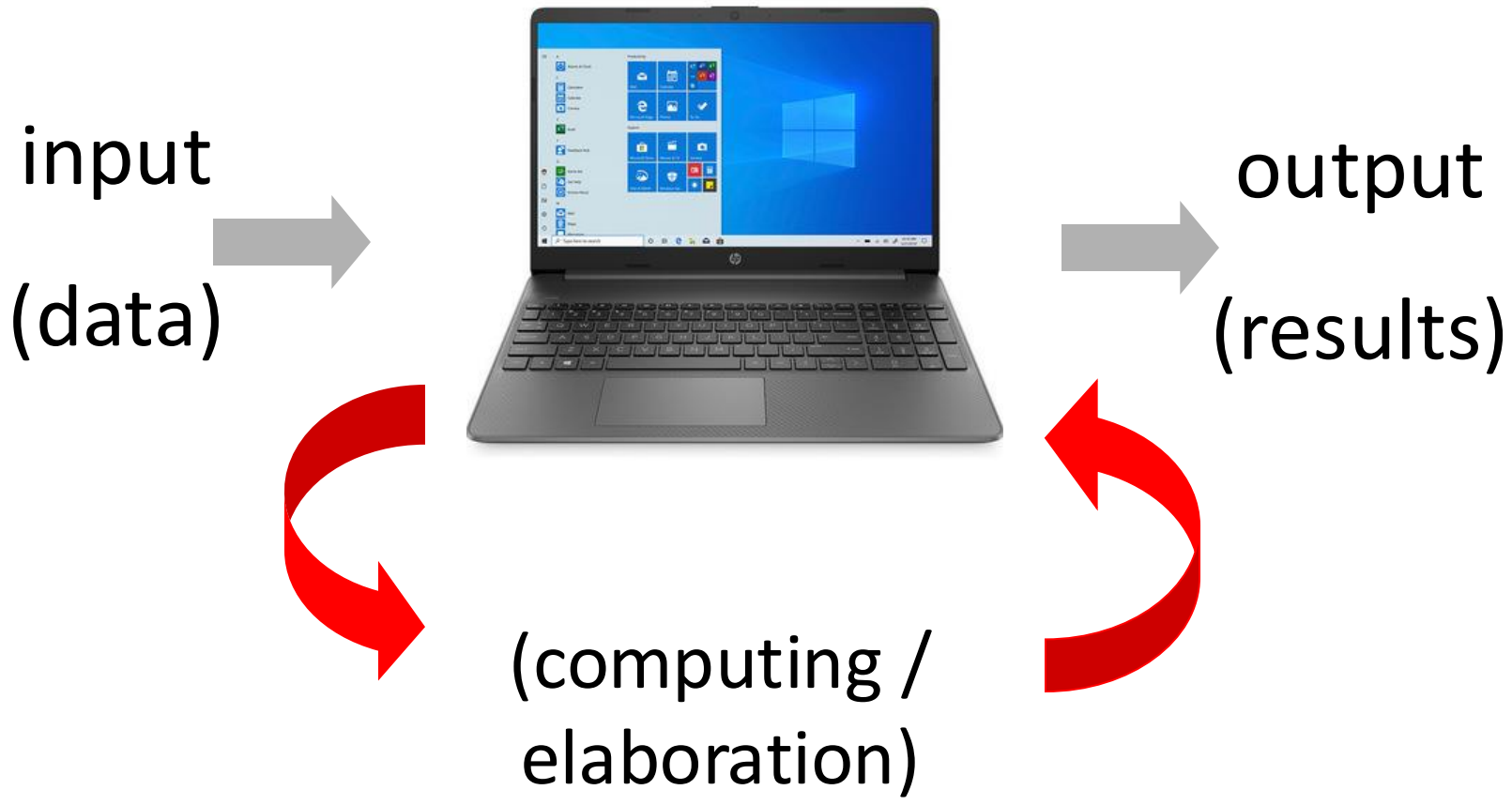
Computer Sciences Introduction

Definition of Computer Science

- Computer Science (Informatics) is the science devoted to the study of how to represent and manipulate **information**



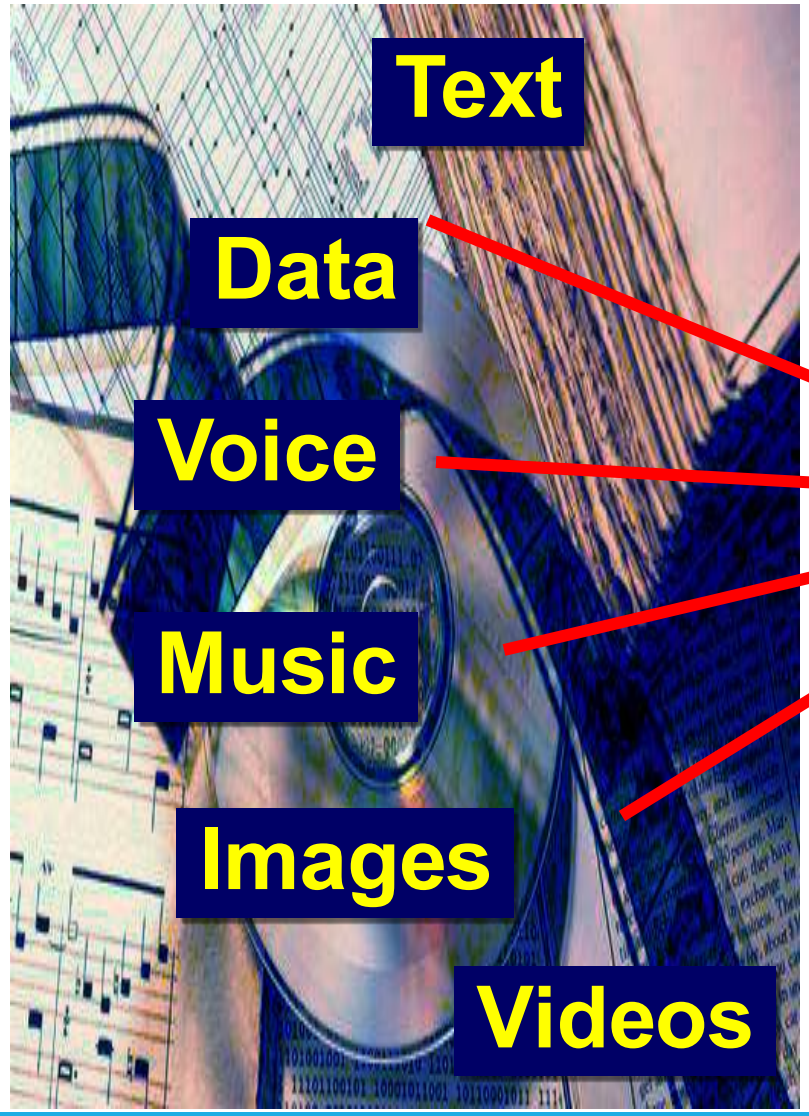
Electronic Computer



Problems

- How to **encode data** in a format that can be understood by the computer
- How to **encode the commands or instructions** into a sequence of operations that composes the desired elaboration
- How to **codify the results** in a format that can be understood by the human user

Digital information: everything becomes “bits”



01101100

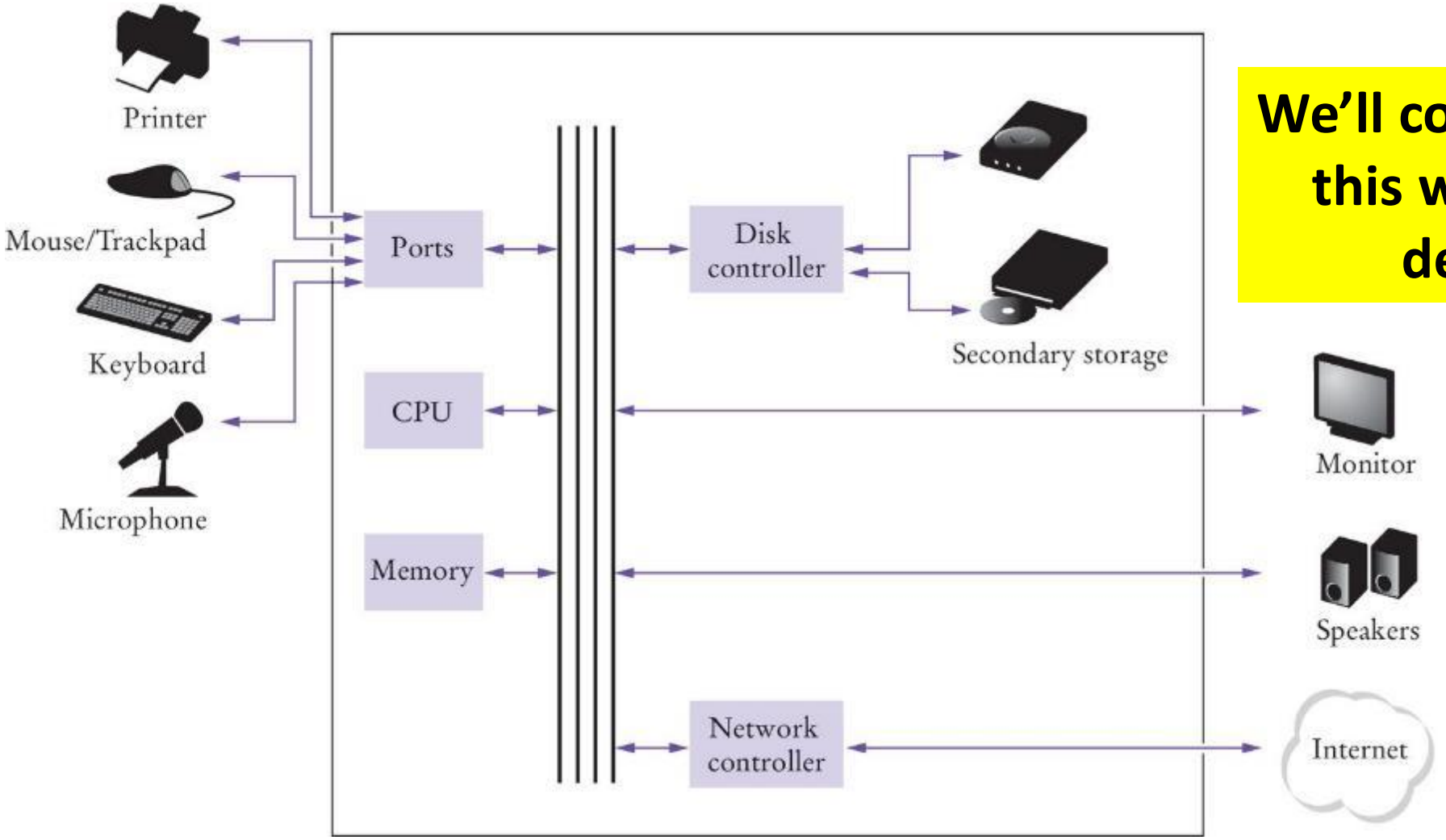
Hardware and Software

- An electronic computer is composed of two parts:
  - **Hardware:** Physical component, consisting of electronic devices, and mechanical, magnetic and optical parts
  - **Software:** "intangible" component consisting of:
    - **Programs**: The "instructions" for the hardware
    - **Data**: Information upon which programs operate

Hardware

- **Hardware** consists of the physical elements in a computer system.
  - Examples: monitor, mouse, external storage, keyboard, ….
- The **central processing unit** (CPU) performs program control and data processing
- **Storage devices** include **main memory** (RAM) and **secondary storage**
  - Hard disks
  - Flash drives
  - CD/DVD drives
- **Input / output devices** allow the user to interact with the computer
  - Mouse, keyboard, printer, screen…

Simplified View of a Computer's Hardware



We'll come back to this with more details!

Software

- **Software** is typically developed in the form of a **“program”**
  - Microsoft Word is an example of software
  - Videogames are software
  - Operating systems and device drivers are also software

Application Program (App)

- **Software**
  - Software is a **sequence of instructions, implemented in some language and translated to a form that can be executed or run on the computer.**
  - It manipulates and transforms **data**
- **Hardware** executes very basic instructions in rapid succession
  - Example: add two numbers.

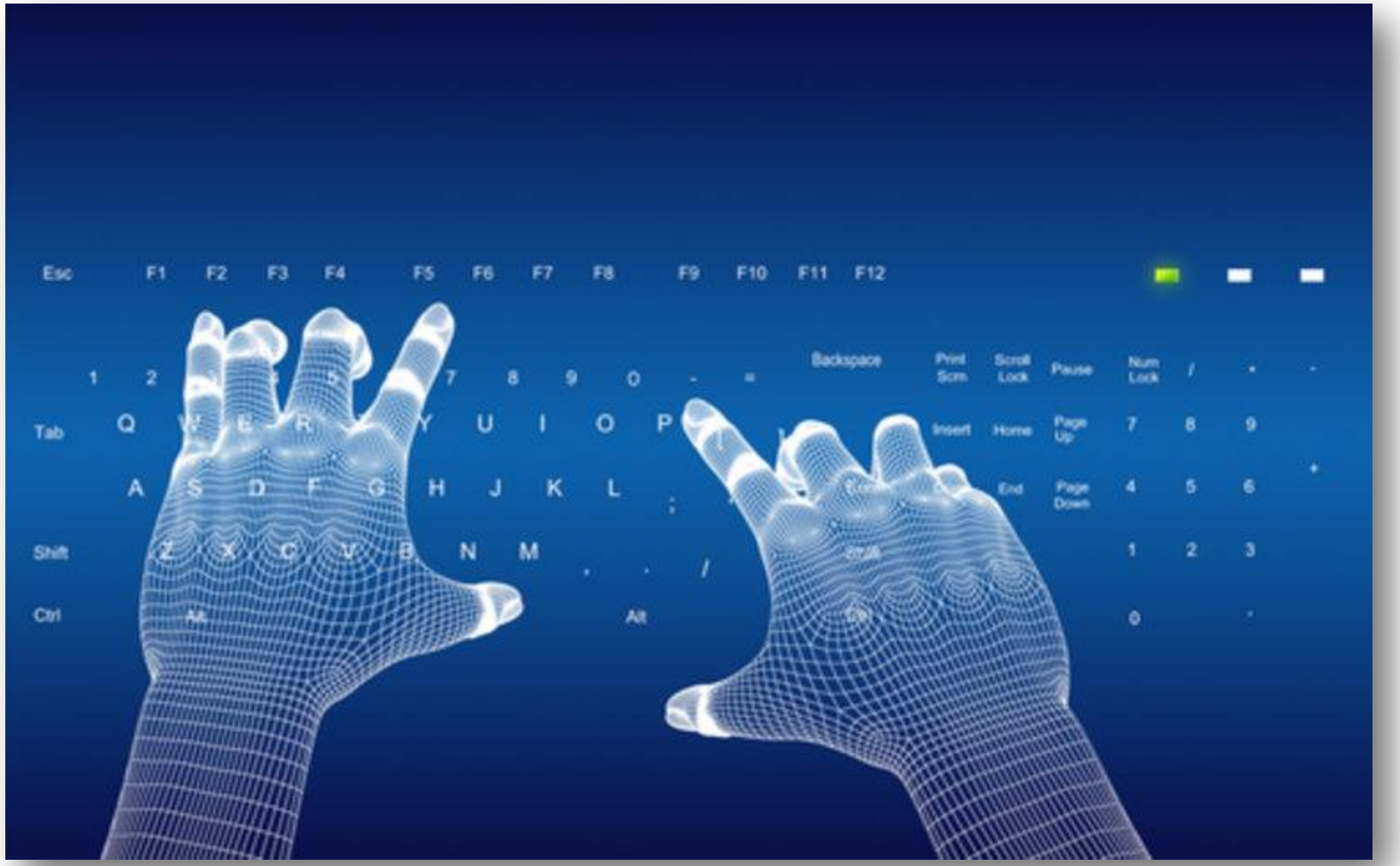
Computer Programs

- A computer program tells a computer the **sequence of steps** needed to complete a specific task
  - The program consists of a (very) large number of primitive (simple) instructions
- Computers can carry out a wide range of tasks because they can **execute** different programs
  - Each program is designed to direct the computer to work on a specific task
- **Programming:**
  - The act (and the **art**) of designing, implementing, and testing computer programs

Executing a Program

- Program instructions and data (such as text, numbers, audio, or video) are **stored in digital format (→ as bits)**
- To execute a program, it must be **brought (loaded) into memory**, where the CPU can read it.
- Then, the CPU executes the program **one instruction at a time**.
  - The program may react to input from the user.
- The **sequence of instructions** and the user input determine the program execution
  - The CPU reads data (including user input), modifies it, and writes it back to memory, the screen, or secondary storage (e.g. hard disk).

Programming



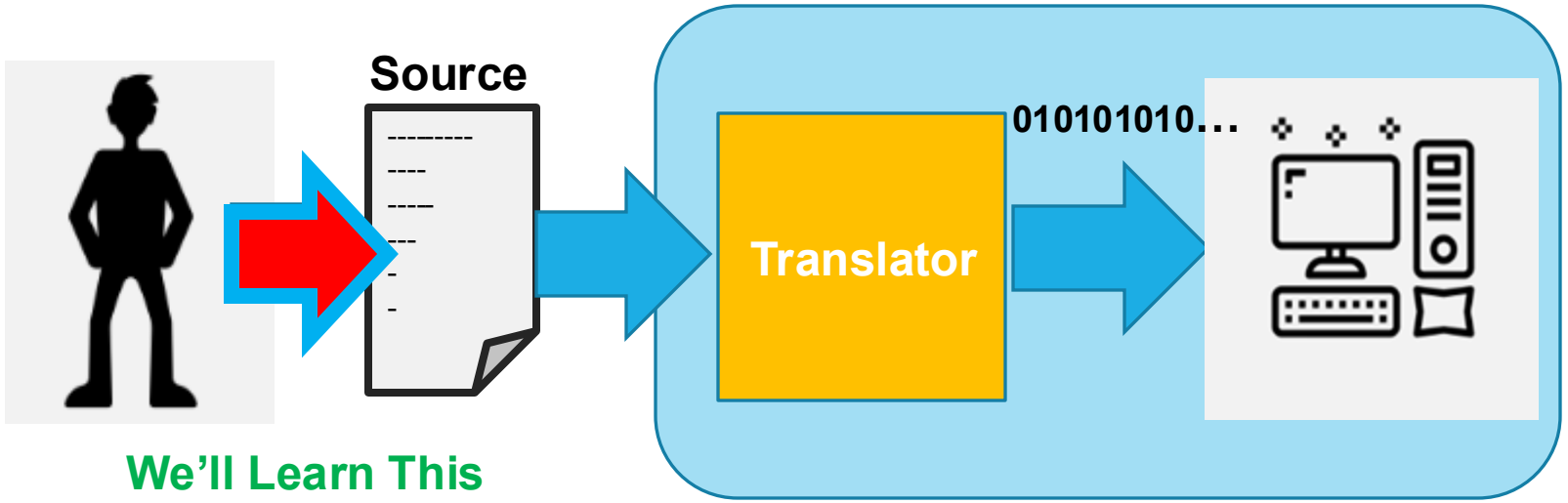
What does ”programming” mean?

- **Programming = designing the solution to a problem in a way that can be solved by a computer**

- To better understand the challenges of programming, it is important to understand the process of **building a program**

Building a Program

1. **Writing a Program**
   - **”Source” code**
   - Written in some programming language
2. **Translation** of the program **in a format that the computer can understand**
   - **”Executable” code**
   - Done automatically by a program called generically **translator**.



What does “Programming” mean?

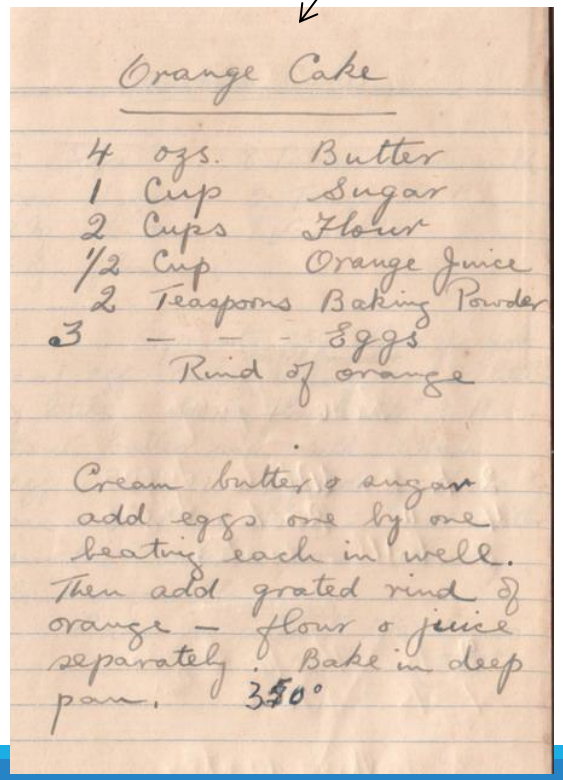
- Programming consists in writing a «document» (**source file**) which **describes the solution** to the considered problem in terms of a **sequence of instructions operating on data**

- In general, “**the**” solution to a problem does **not** exist
  - Programming consists of finding the *most efficient and effective solution* (according to appropriate metrics) for the problem

Cooking vs Programming

Hardware

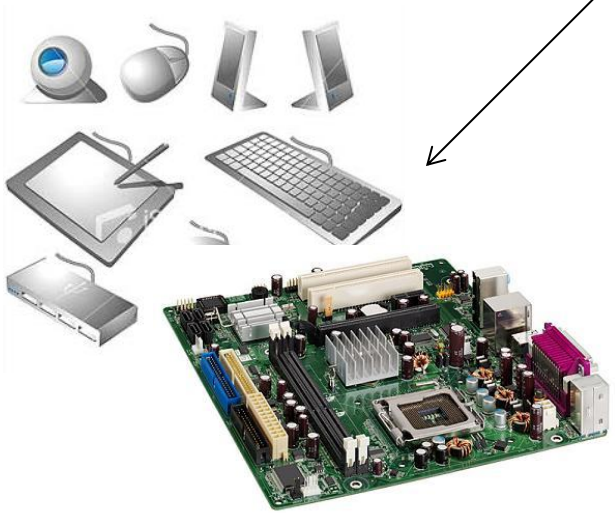
Software



Cooking vs Programming

Hardware

Software

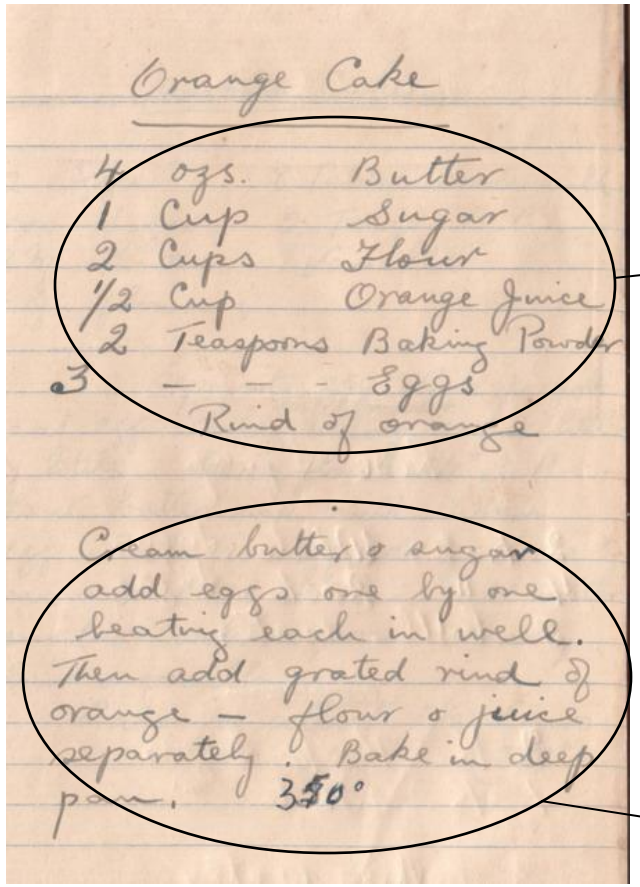


```
void CruiseControl_init(_C_CruiseControl * _C_)
{
    CruiseSpeedMgt_init(&(_C_->_C0_CruiseSpeedMgt));
    CruiseStateMgt_init(&(_C_->_C3_CruiseStateMgt));
    (_C_->_M_condact_0) = true;
    ThrottleCmd_init(&(_C_->_C4_ThrottleCmd));
    (_C_->_M_init) = true;
}

/* ========= */
/* MAIN NODE */
/* ========= */

void CruiseControl(_C_CruiseControl * _C_)
{
    bool BrakePressed;
    bool AcceleratorPressed;
    bool SpeedOutOffLimits;
    bool _L19;
/*#code for node CruiseControl */
/* call to node not expanded DetectPedalsPressed */
    (_C_->_Cn_DetectPedalsPressed._I0_Brake) = (_C_->_I
    (_C_->_Cn_DetectPedalsPressed._I1_Accelerator) = (_
    DetectPedalsPressed(&(_C_->_Cn_DetectPedalsPressed)
    BrakePressed = (_C_->_Cn_DetectPedalsPressed._O0_Br
    AcceleratorPressed =
        (_C_->_Cn_DetectPedalsPressed._O1_AcceleratorPr
/* call to node not expanded DetectSpeedLimits */
    (_C_->_Cn_DetectSpeedLimits._I0_speed) = (_C_->_I8_
    DetectSpeedLimits(&(_C_->_Cn_DetectSpeedLimits));
    SpeedOutOffLimits = (_C_->_Cn_DetectSpeedLimits._O0_
/* call to node not expanded CruiseStateMgt */
    (_C_->_C3_CruiseStateMgt._I0_BrakePressed) = BrakeP
```

Cooking vs Programming



Ingredients/Data

```
/* ========= */
/* MAIN NODE */
/* ========= */

void CruiseControl(_C_CruiseControl * _C_)
{
    bool BrakePressed;
    bool AcceleratorPressed;
    bool SpeedOutOffLimits;
    bool _L19;
/*#code for node CruiseControl */
/* call to node not expanded DetectPedalsPressed */
    (_C_->_Cn_DetectPedalsPressed._I0_Brake) = (_C_->_I
    (_C_->_Cn_DetectPedalsPressed._I1_Accelerator) = (_
    DetectPedalsPressed(&(_C_->_Cn_DetectPedalsPressed))
    BrakePressed = (_C_->_Cn_DetectPedalsPressed._O0_Br
    AcceleratorPressed =
        (_C_->_Cn_DetectPedalsPressed._O1_AcceleratorPr
/* call to node not expanded DetectSpeedLimits */
    (_C_->_Cn_DetectSpeedLimits._I0_speed) = (_C_->_I8_
    DetectSpeedLimits(&(_C_->_Cn_DetectSpeedLimits));
    SpeedOutOffLimits = (_C_->_Cn_DetectSpeedLimits._O0_
/* call to node not expanded CruiseStateMgt */
    (_C_->_C8_CruiseStateMgt._I0_BrakePressed) = BrakeP
```

Instructions/Code

What is “programming”?

- Programming is a “**creative**” task!
  - Every problem is **different** from every other problem
  - No silver bullets (**universal solutions**)
  - (Almost) **no systematic/analytic** solutions

- Programming is a complex operation
  - A “direct” approach (from the problem directly to the final source code) is almost impossible
  - The problem needs to be decomposed in sub tasks
  - Recognize known “patterns” (don’t reinvent the wheel)
  - Usually, organized in several stages (refinements)

Algorithms

1.7

Our First Definition

- **Algorithm**
- An algorithm is a **step-by-step, formal description** of **how to solve a problem**
- A “systematic procedure that produces — in a finite number of steps — the answer to a question or the solution of a problem”
- A program is an implementation of an algorithm in a certain language.

The name derives from Al-Khwārizmī (c. 780 — c. 850), Persian mathematician and astronomer

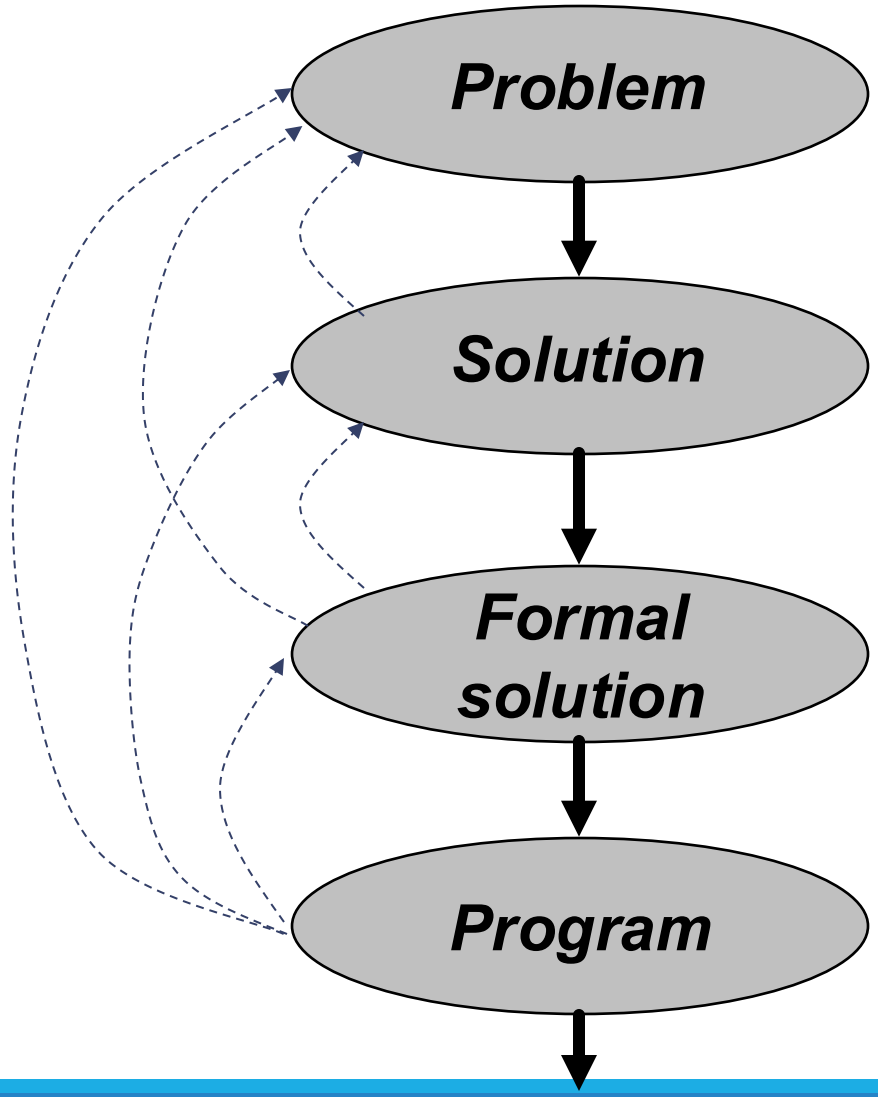
Find the biggest number

9345

1302

5901

Designing a Program

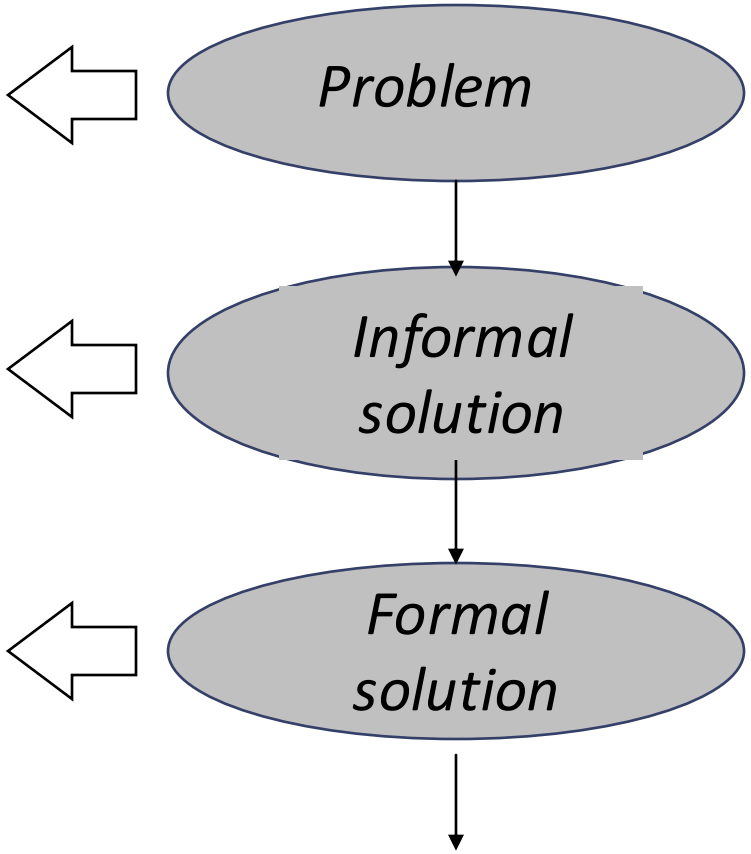


Algorithm!

Example of problem solving

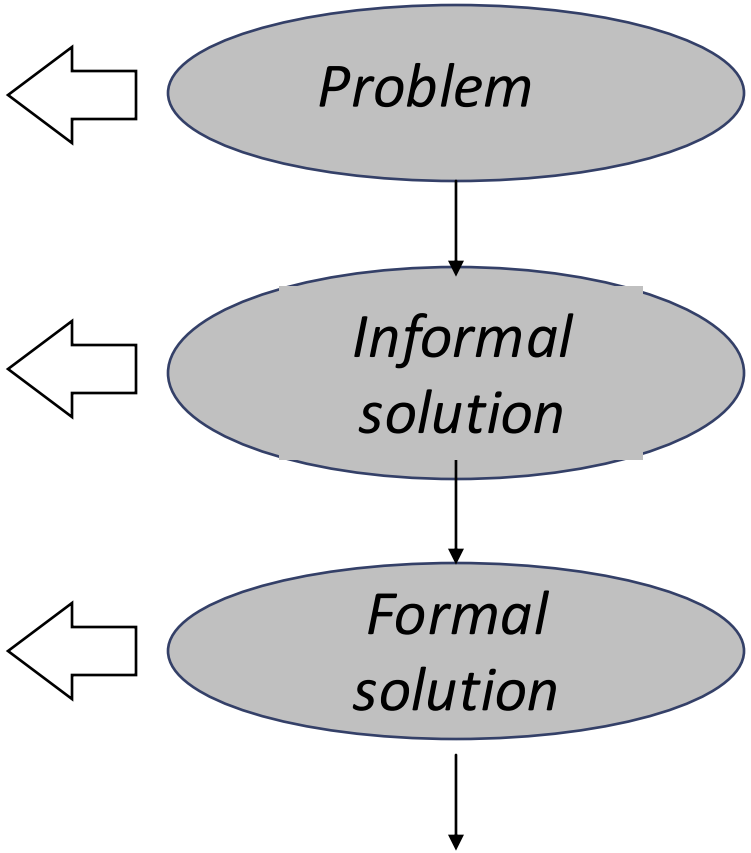
- Problem: Compute the maximum among A, B and C
- Solution: The maximum is the largest number between A, B and C…
- Formal solution:

  ??



Example of problem solving

- Problem: Compute the maximum among A, B and C
- Solution: The maximum is the largest number between A, B and C…
- Formal solution:
  1. if A >= B and A >= C then MAX = A;
  2. Else if B >= A and B >= C then MAX = B;
  3. Else if C >= A and C >= B then MAX = C;



Find the biggest number

```
1256 6765 6829 3847 3284 4798 3848
0185 3721 3689 4574 3985 4389 5743
4395 7239 8473 2854 3785 6648 0101
9840 7231 5672 1419 4385 6318 9910
3532 5498 6043 7542 8957 2149 3285
4903 2750 3019 2754 8018 3605 2389
2740 8993 7416 5415 9857 1624 8935
7318 7561 3746 5327 5432 5436 5078
5612 3465 7984 8764 1264 1547 8144
2194 1948 1282 5821 8975 4276 5219
```

Algorithms in everyday life

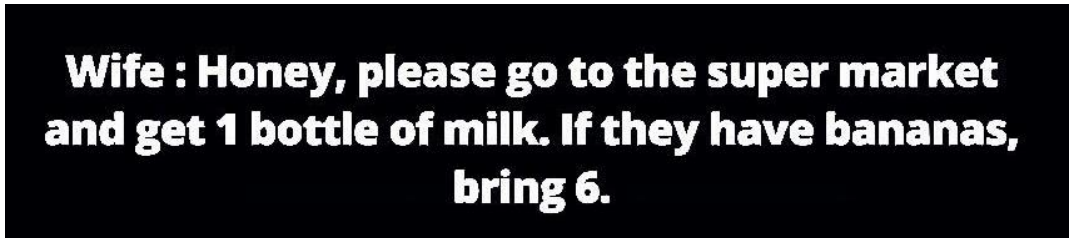
1. Pour some water
2. Light the fire
3. Wait
4. If water is not boiling, go back to n.3
5. Add pasta
6. Wait a little
7. Taste
8. If it's still raw go back to 6
9. Remove water

IS THERE AN ERROR IN THIS ALGORITHM?

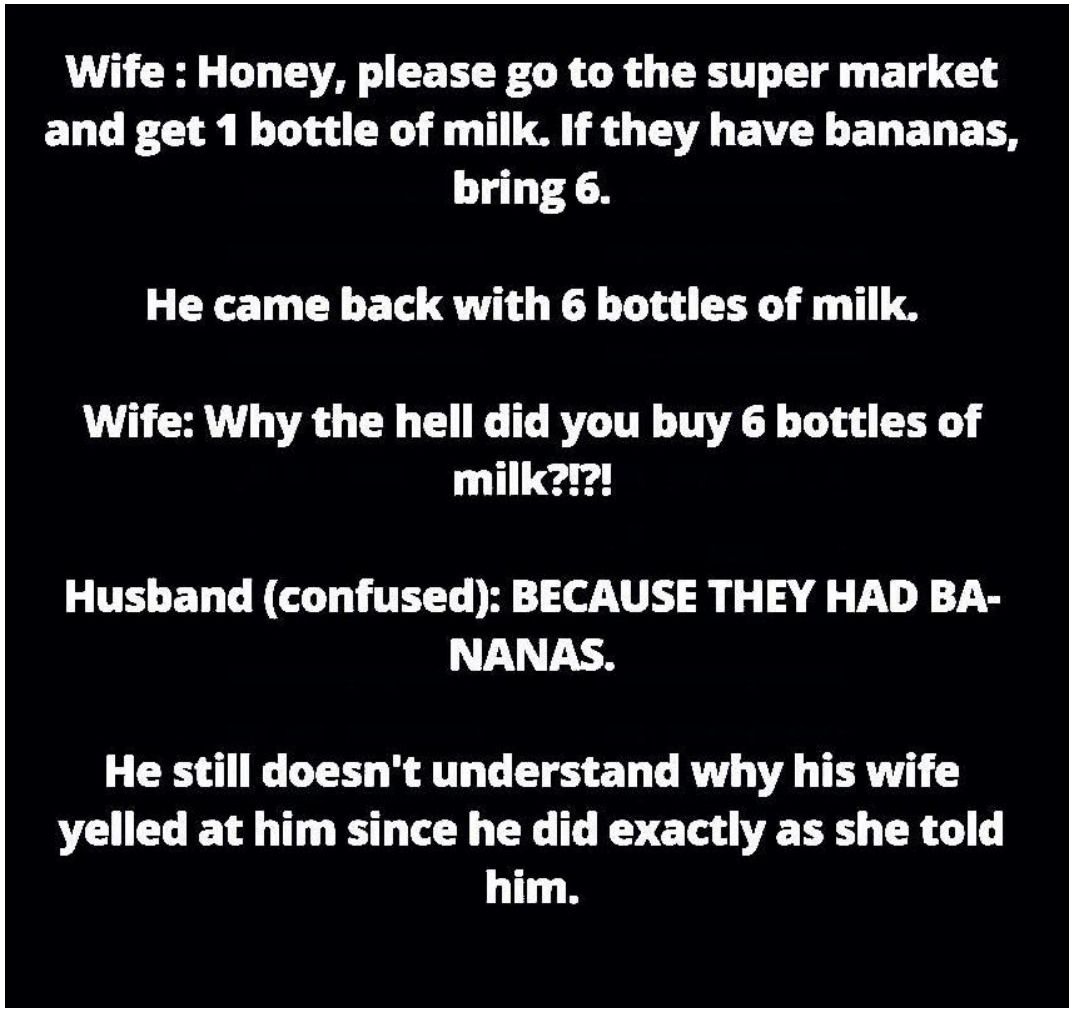
Algorithm: Formal Definition

- An algorithm describes a sequence of steps that is:
- **Unambiguous**
  - No “assumptions” are required to execute the algorithm
  - No “common sense” is assumed
  - The algorithm uses precise instructions
- **Simple**
  - Each step can be carried out in practice
- **Terminating**
  - The algorithm will eventually come to an end

Thinking Like a Programmer



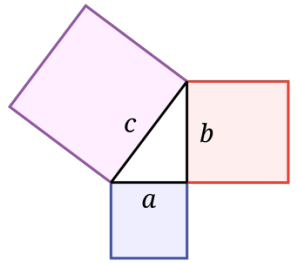
Thinking Like a Programmer



Ambiguous Algorithm!

Problem Solving: Algorithm Design

- We are already familiar with many algorithms
  - Calculate the area of a circle
  - Verify that a triangle is a right triangle
  - Solve a quadratic equation $ax^2 + bx + c = 0$



- Some problems are more complex and require to go through a variable number of steps
  - Find the **greatest common divisor** of two numbers

Example: Selecting a Car

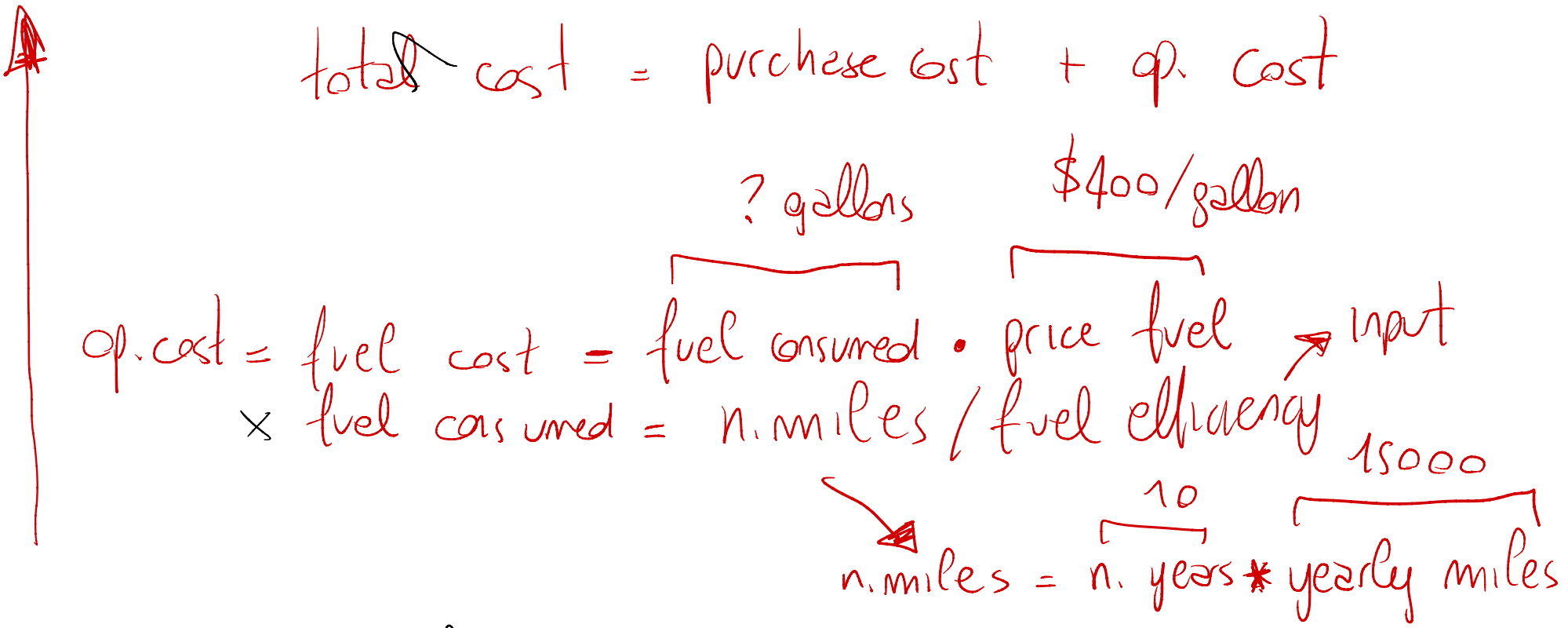
- Problem Statement:
  - You have the choice of buying two cars
  - One is more fuel efficient than the other, but also more expensive
  - You know the **price and fuel efficiency** (in miles per gallon, mpg) of both cars
  - You know the price of fuel per gallon **($4.00 / gallon)**
  - You know that you drive **15000 miles per year**
  - You plan to **keep the car for ten years**
  - Which car is the better deal?

Developing the Algorithm

- Determine the inputs and outputs
- From the problem statement we know:
  - Car 1: Purchase price, Fuel Efficiency
  - Car 2: Purchase price, Fuel Efficiency
  - Price per gallon = $4.00
  - Annual miles driven= 15,000
  - Length of time = 10 years
- For each car we need to calculate:
  - Annual fuel consumed for each car
  - Operating cost for each car
  - Total cost of each Car
- Then we select the car with the lowest total cost

Formalizing the Algorithm

- Break down the problem into smaller tasks
  - ‘Calculate total cost’ for each car
  - To calculate the total cost for each year we need to calculate the operating cost
  - The operating cost depends on the annual fuel cost
  - The annual fuel cost is the price per gallon times the annual fuel consumed
  - The annual fuel consumed is the annual miles drive divided by fuel efficiency
- Describe each subtask
  - total cost = purchase price + operating cost
  - etc.



yearly miles = 15000

n.years = 10

fuel efficiency1 = ...

fuel efficiency2 = ...

price fuel = 4.00

purchase1 = ...

purchase 2 = ...

n.miles = n.years * yearly miles

fuel_consumed1 = n.miles/fuel efficiency1.

fuel_consumed2 - n.miles/fuel efficiency 2

op cost1 = fuel consumed1 • price fuel

op.cost2 = fuel consumed2 • price fuel

tot cost1 = op.cost1 + purchase 1

tot cost2 = op cost2 + purchase 2

```
If tot cost1 < tot cost2 :
    print ("car 1")
else:
    print ("car 2")
```

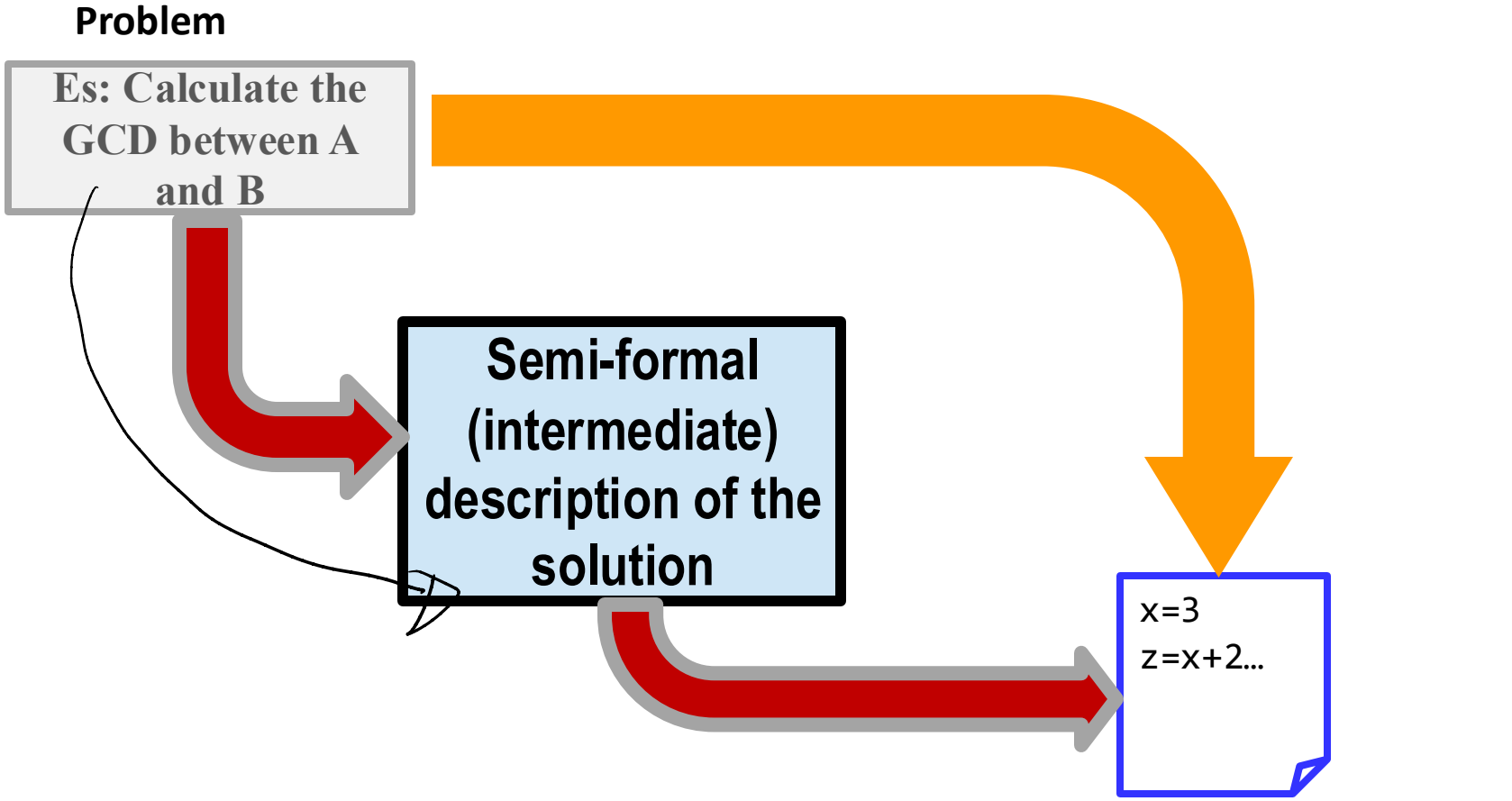
Intermediate Step

- The program is the **final representation of an algorithm**/solution

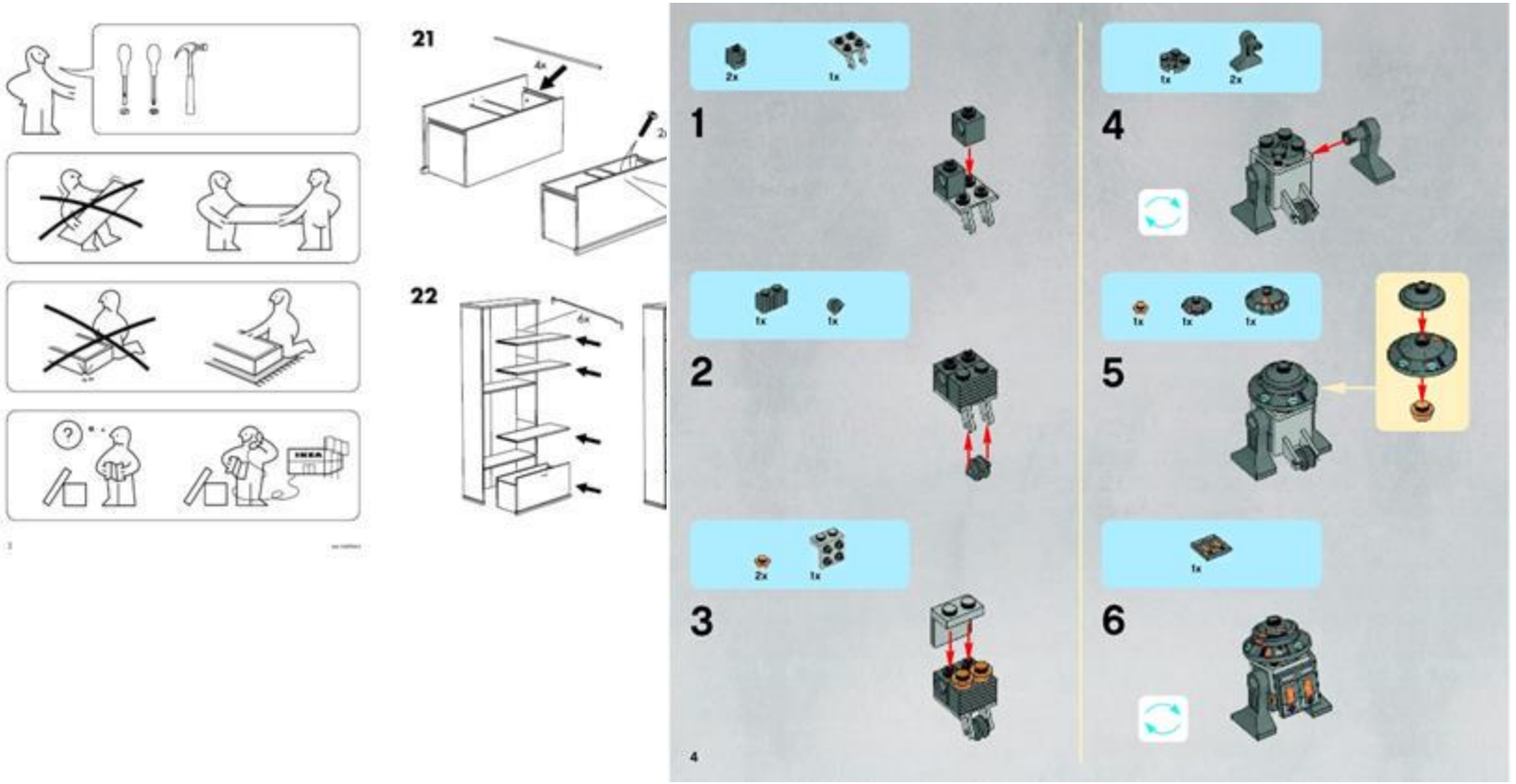
- Except in very simple cases, a direct approach (from the problem to the source program) is not feasible

- So let's decouple problem of "**finding a solution**" from the problem of "**writing the program**", using a **"semi-formal" intermediate solution**

From Problem to Program



Intermediate Representation: Inspiration



- Structured visual language
- Sequential operations
- Sub-operations
- Repetitions

Intermediate Representation of a Solution

- For a computer algorithm, we need a formalism that is:
  - Unambiguous
  - Aware of the operations that the computer can perform
  - Easy to interpret

Intermediate Representation of a Solution

Two main options:

- **Pseudo-code**
  - Half-way between natural language and a programming language
- **Flow Charts**
  - Intuitive graphical formalism
  - …Struggle to represent complex or more abstract operations

Possible Pseudo-code for the “car problem”

- For each Car, compute the total cost
  - annual fuel consumed = annual miles driven / fuel efficiency
  - annual fuel cost = price per gallon * annual fuel consumed
  - operating cost = length of time * annual fuel cost
  - total cost = purchase price + operating cost
- If total cost1 < total cost2
  - Choose Car1
- Else
  - Choose Car2

Bank Account Example

- You put $10,000 into a bank account that earns 5 percent interest per year. How many years does it take for the account balance to be double the original?

Bank Account Example

- How would you solve it?
  - Make a table
  - Add lines until done
- Use a spreadsheet!

year	balance
0	10000
1	10000.00 x 1.05 = 10500.00
2	10500.00 x 1.05 = 11025.00
3	11025.00 x 1.05 = 11576.25
4	11576.25 x 1.05 = 12155.06

y 1

B 10500

I 0.05

B = B · (1 + I)

y = y + 1

B_inital = 10.000

B = B_inital

while B ≤ 2 · B_initial:

B = B · (1 + I)

y = y + 1

print (y)

Developing the Algorithm

- You put $10,000 into a bank account that earns 5 percent interest per year. How many years does it take for the account balance to be double the original?

year	balance
0	10000

- Break it into steps
  - Start with a year value of 0 and a balance of $10,000
  - Repeat the following while the balance is less than $20,000
    - Add 1 to the year value
    - Multiply the balance by 1.05
    - (5% increase)

year	balance
0	10000
1	10500
14	19799.32
15	20789.28

  - Report the final year value as the answer

Formalizing the Algorithm: Rules

- I must solve the problem, and imagine I have one sheet of paper (memory) and a pen, only

- Every information that I need to remember must be written on the paper

Formalized Algorithm and Pseudo-code

- Set the year value of 0
- Set the balance to $10,000
- While the balance is less than $20,000
  - Add 1 to the year value
  - Multiply the balance by 1.05
- Report the final year value as the answer

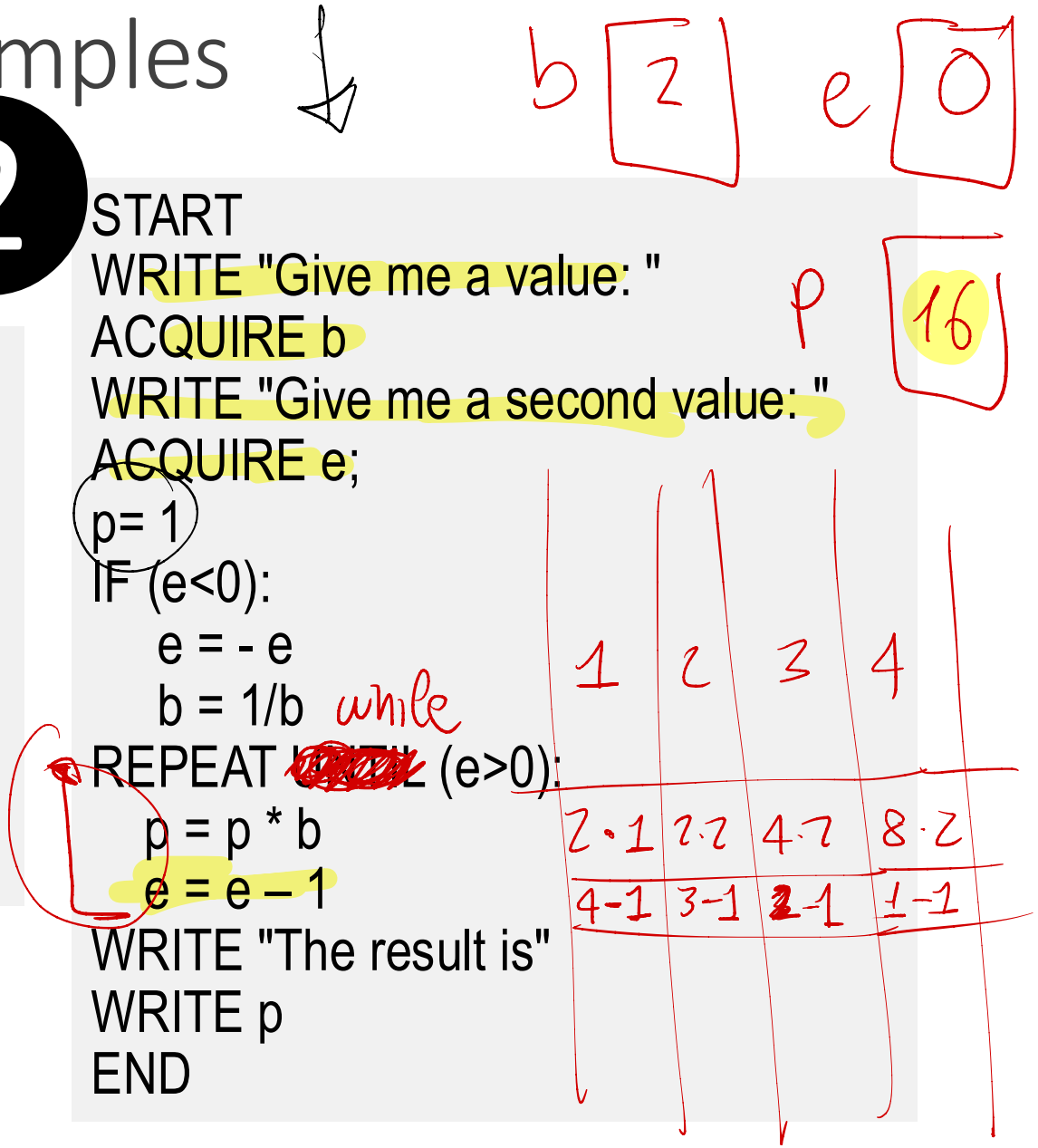
Pseudo-code: other examples

1

```
START
Write: "Enter a number n"
ACQUIRE n from user
If the remainder of the division by 2 is 0:
        Write: "The number is even"
Otherwise:
        Write: "The number is odd".
END
```

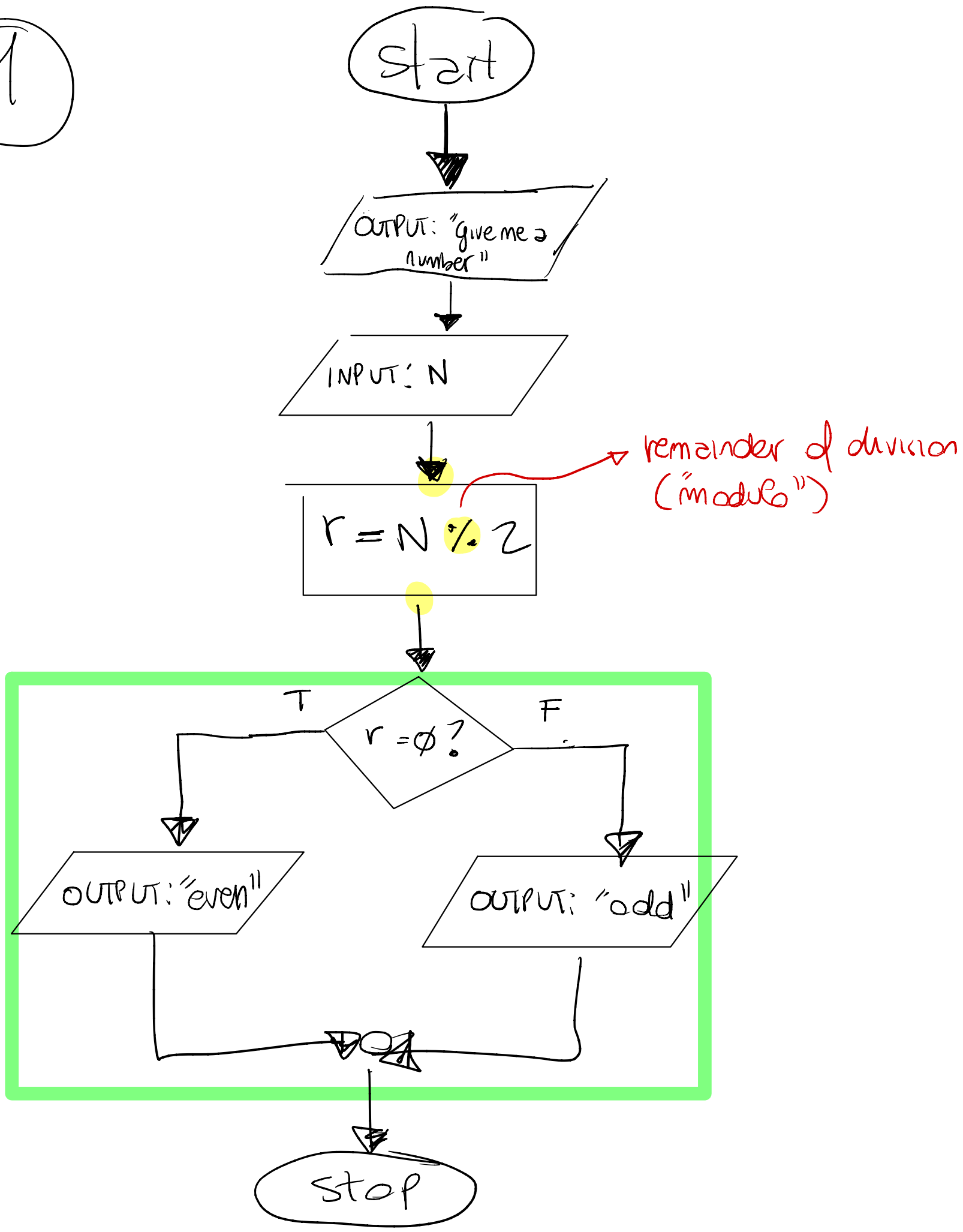
2

```
START
WRITE "Give me a value: "
ACQUIRE b
WRITE "Give me a second value: "
ACQUIRE e;
p= 1
IF (e<0):
    e = - e
    b = 1/b
REPEAT UNTIL (e>0):
    p = p * b
    e = e – 1
WRITE "The result is"
WRITE p
END
```

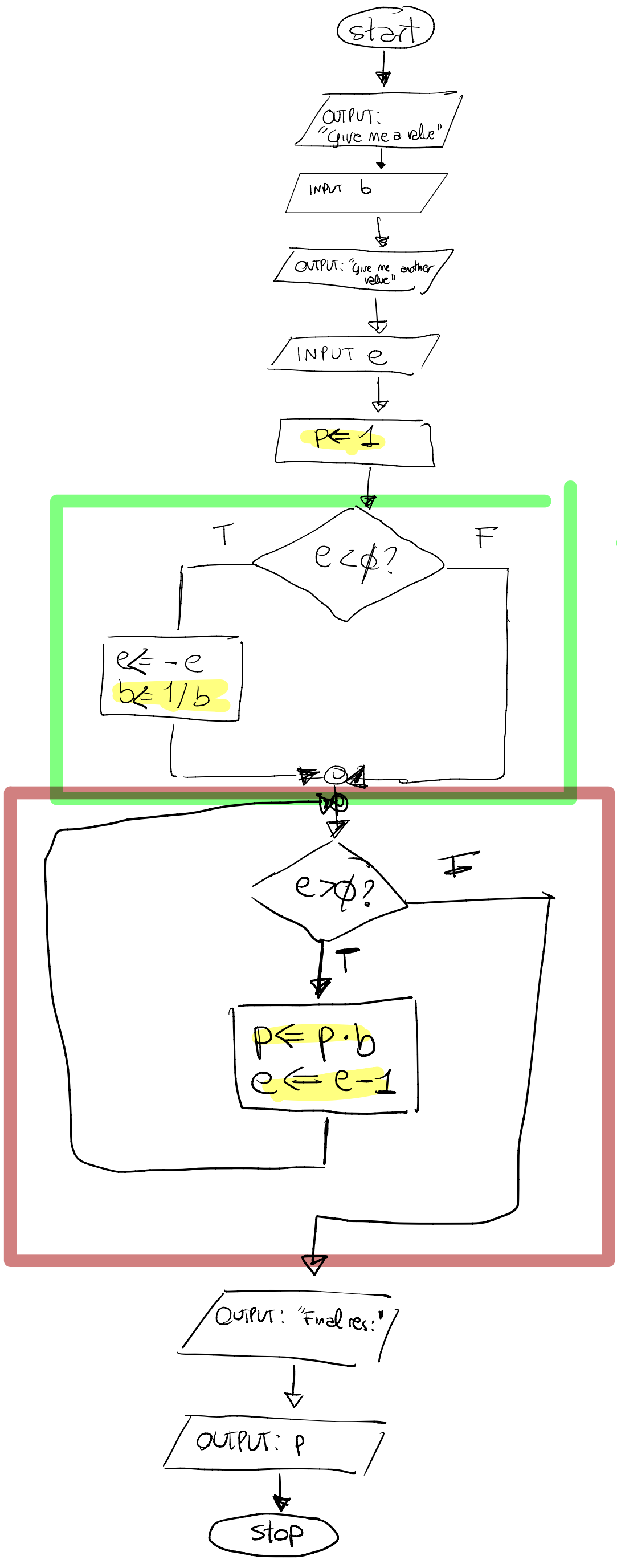


$$2^{-4} = \frac{1}{2^4}$$

1


start
OUTPUT: "give me a number"
INPUT: N
r = N % 2
remainder of division ("modulo")
r = ∅ ?
T
F
OUTPUT: "even"
OUTPUT: "odd"
stop

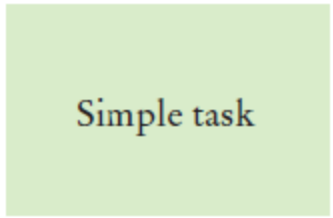
2

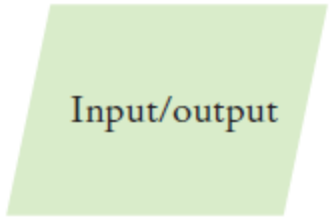

start
OUTPUT: "Give me a value"
INPUT b
OUTPUT: "Give me another value"
INPUT e
p ← 1
T
F
e < 0 ?
e ← -e
b ← 1/b
e > 0 ?
F
T
p ← p · b
e ← e - 1
OUTPUT: "Final res:"
OUTPUT: p
stop
if/else
loop

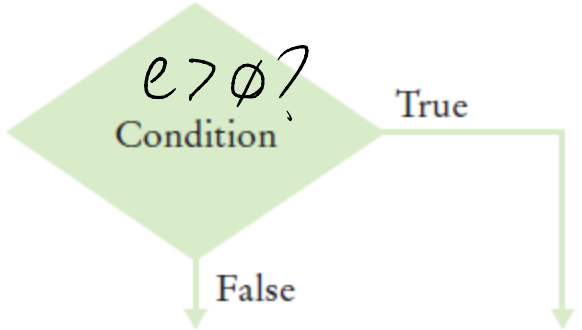
Flowcharts

Problem Solving: Flowcharts

- A flowchart shows the structure of decisions and tasks to solve a problem
- Basic flowchart elements:



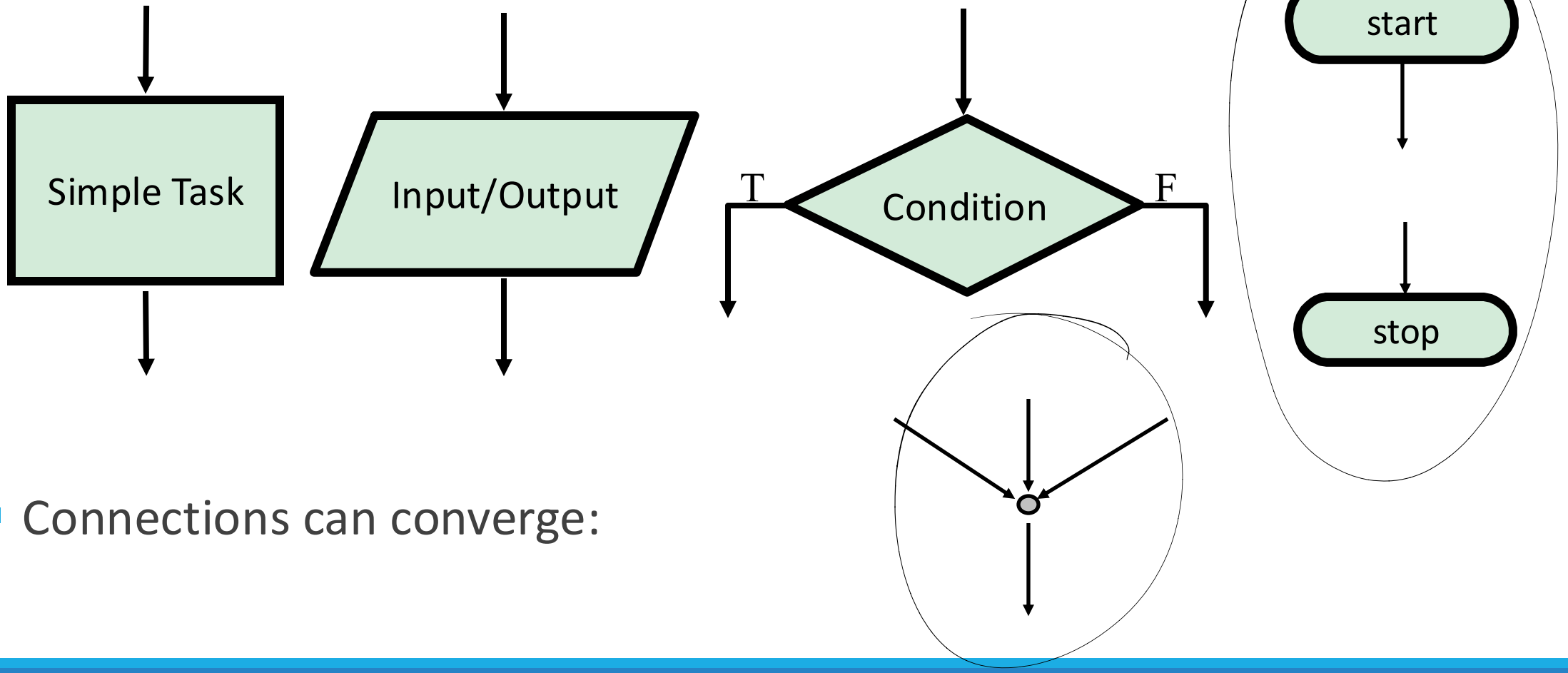




- Connect them with arrows
- Each branch of a decision can contain tasks and further decisions

Flow Charts

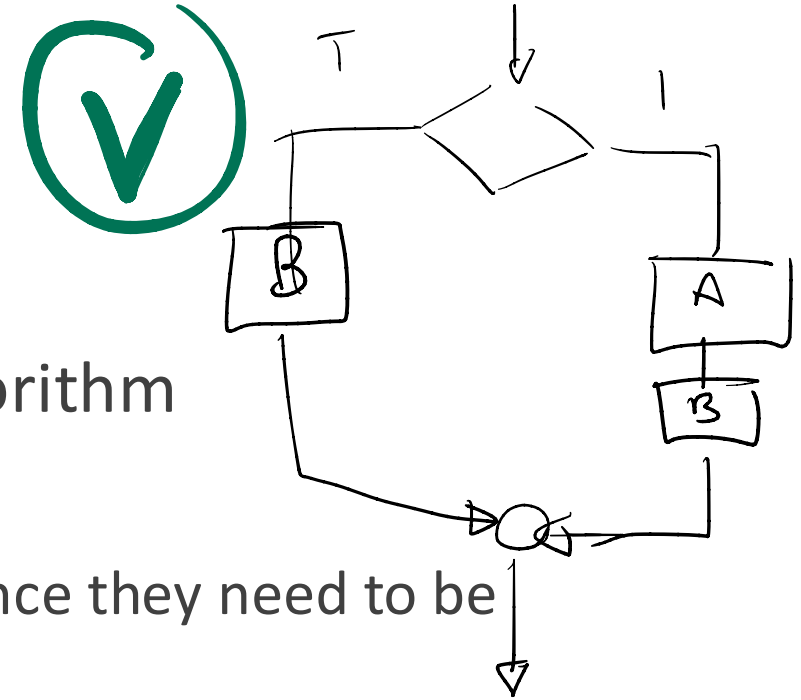
- Connection rules between blocks

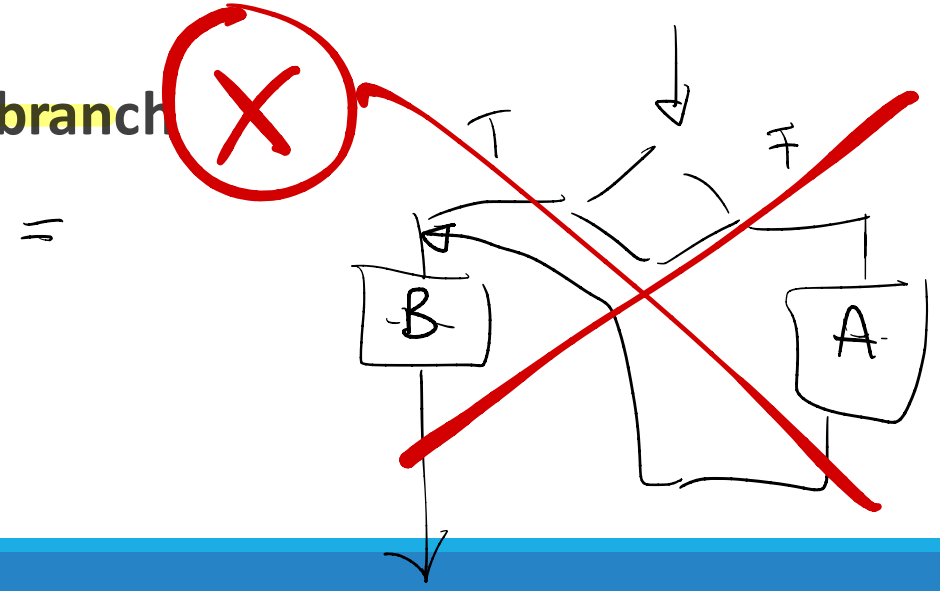


- Connections can converge:

Using Flowcharts

- Flowcharts are an excellent tool
- They can help you visualize the flow of your algorithm
- Building the flowchart
  - Link your tasks and input / output boxes in the sequence they need to be executed
  - When you need to make a decision use the diamond (a conditional statement) with two outcomes
  - **Never point an arrow inside another branch**





Flowcharts: Abstraction Layer

- What can I write inside the blocks (especially action blocks)?

- More precisely, what level of complexity can the operations I write inside the blocks have?

Flowcharts: Abstraction Layer

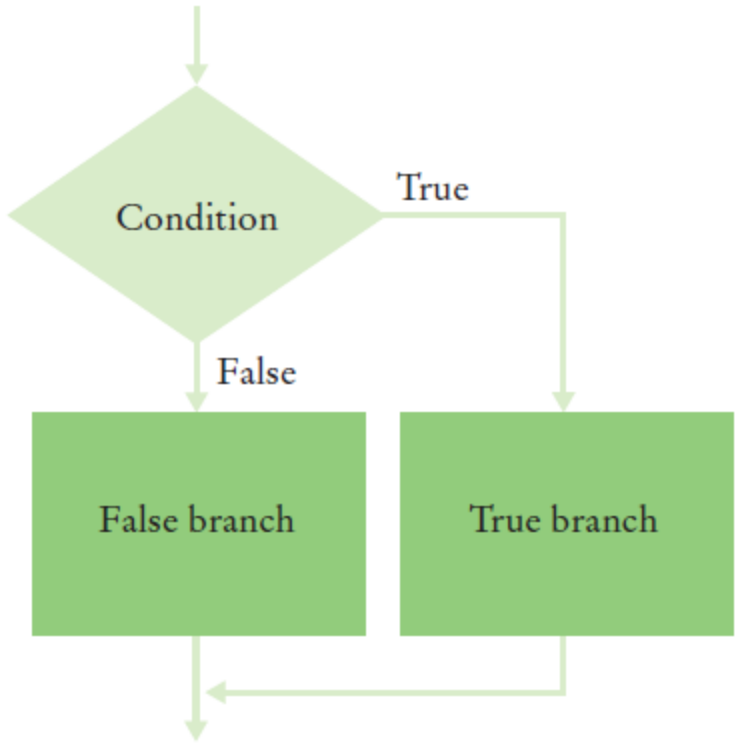
- Action blocks should contain actions corresponding to the **operations performed by a computer (microprocessor)**
  - Micro-operations
- Low level of abstraction

  - **Arithmetic operations between numeric values**
  - **Moving data**
  - **Acquisition (input) and printing (output)**
  - **Comparison of numerical data**
  - **Logical operations between logical conditions (union, intersection)**

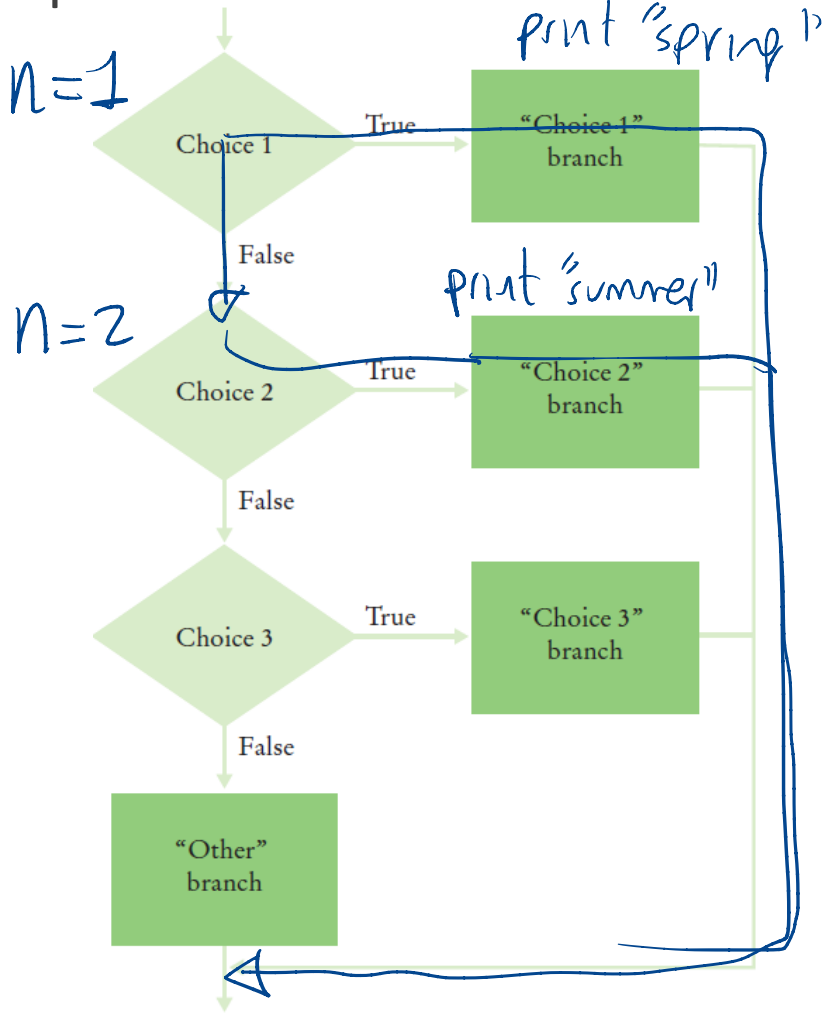
- These are the operations that the computer circuits actually implement (as we will see later)

Conditional Flowcharts

- Two Outcomes

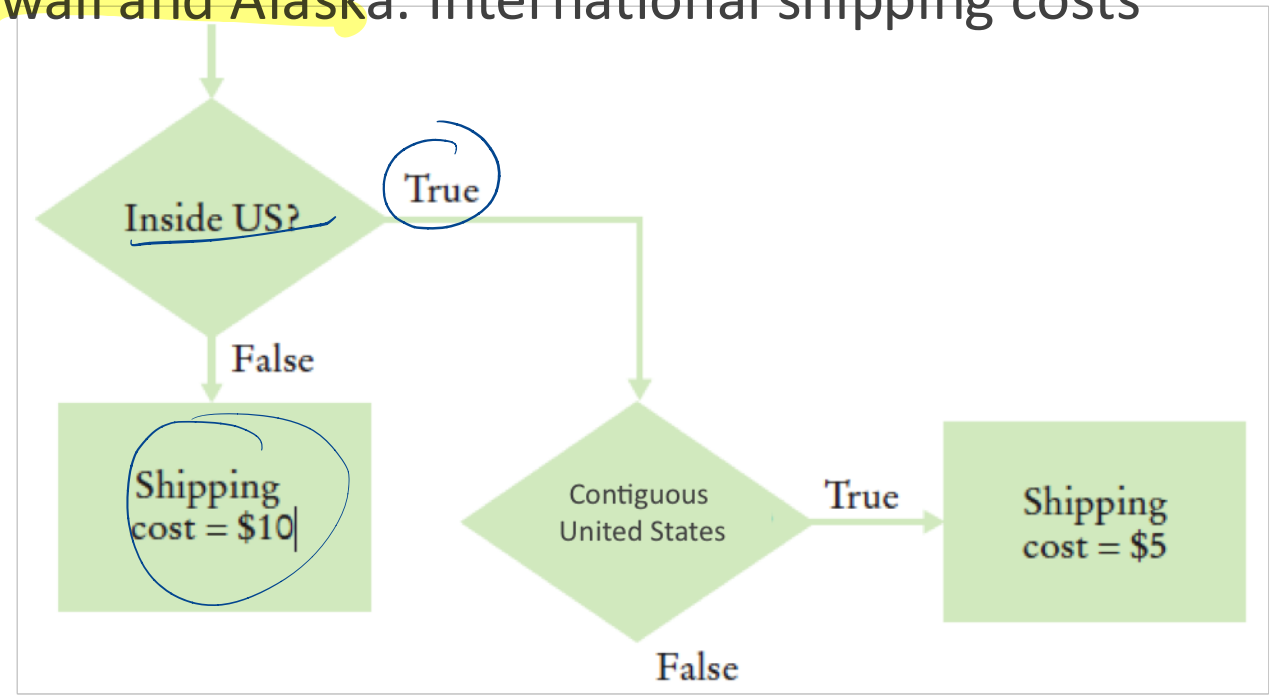


- Multiple Outcomes



Shipping Cost flowchart

- **Problem: we want to compute how much we will pay to ship a packet to a friend**
- Available data: Shipping costs are $5 inside the contiguous United States (Lower 48 states), and $10 to Hawaii and Alaska. International shipping costs are also $10.
- Three Branches:



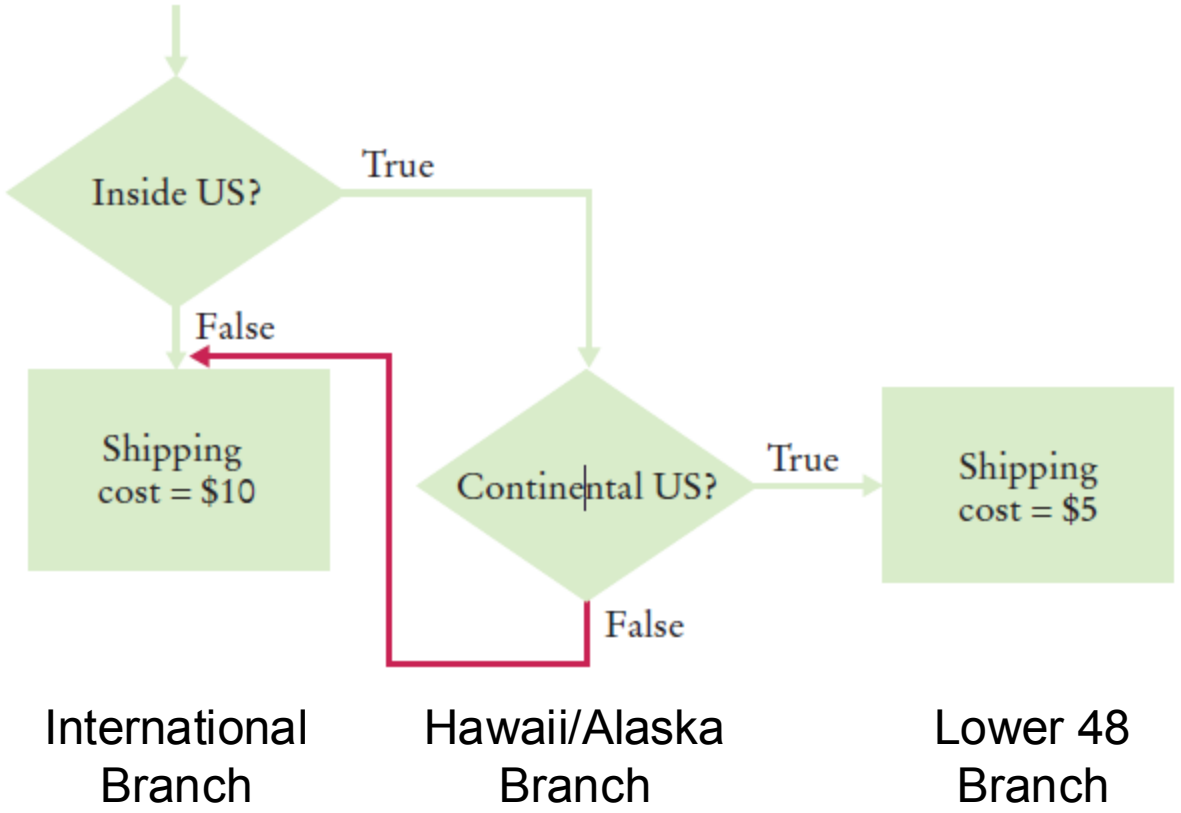
International Branch

Hawaii/Alaska Branch

Lower 48 Branch

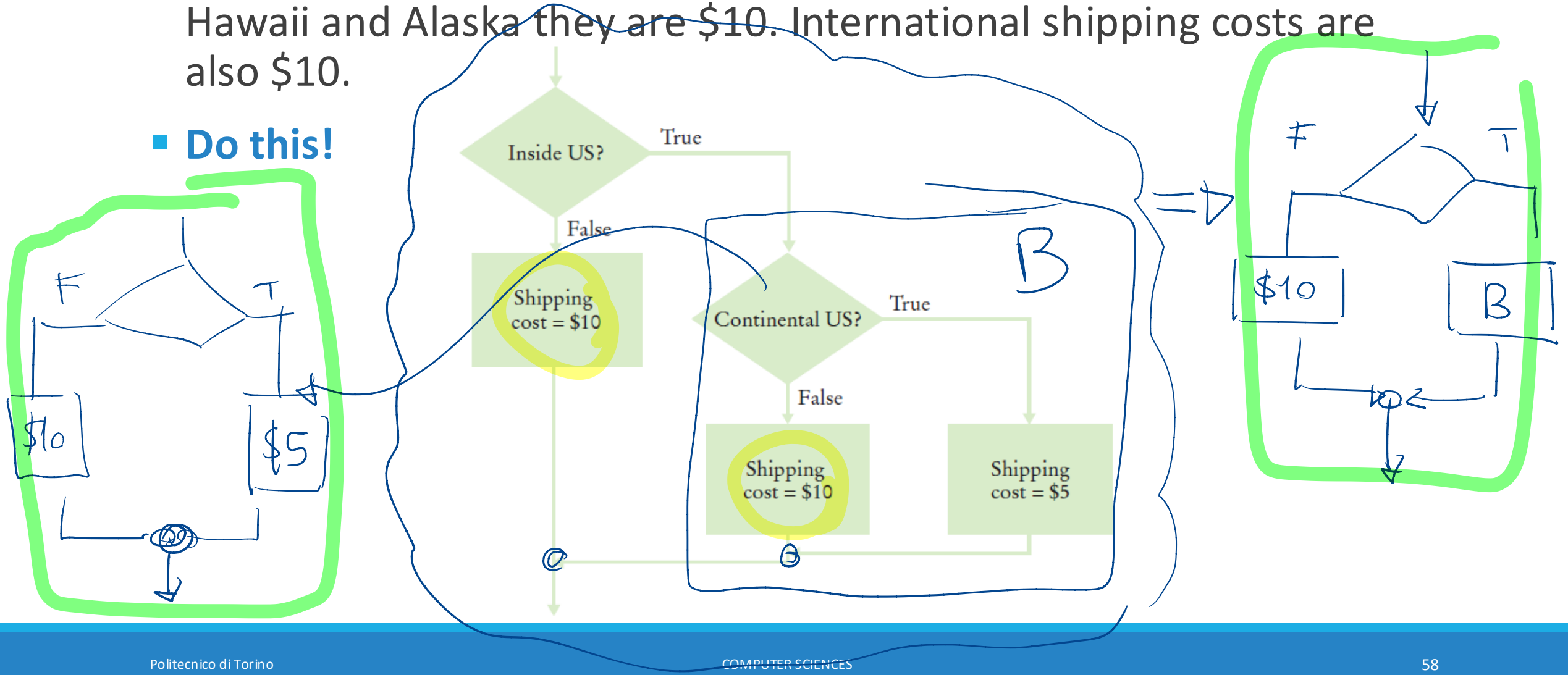
Don’t Connect Branches!

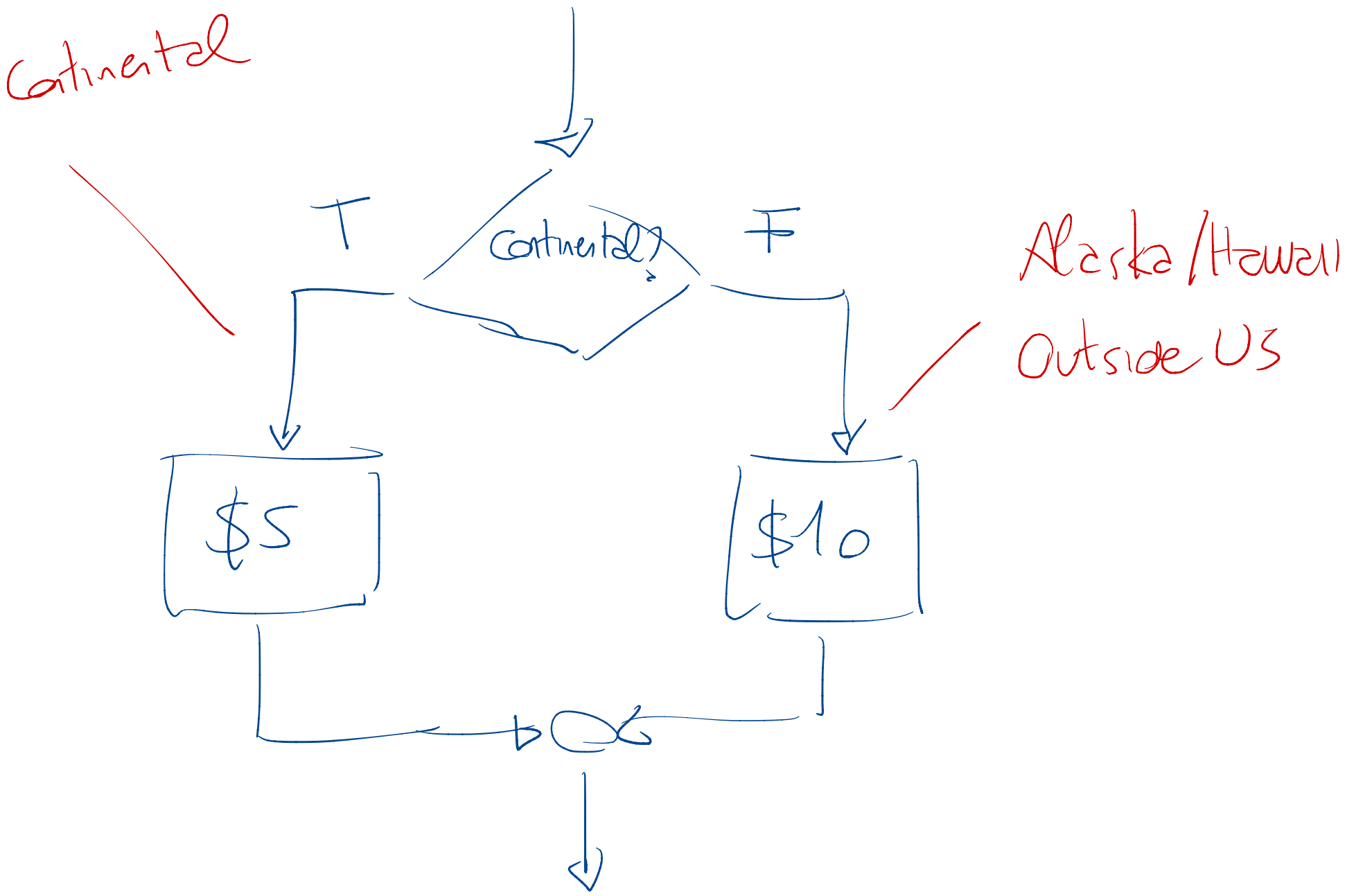
- Shipping costs are $5 inside the United States, except that to Hawaii and Alaska they are $10. International shipping costs are also $10.
- Don’t do this!



Shipping Cost Flowchart

- Shipping costs are $5 inside the United States, except that to Hawaii and Alaska they are $10. International shipping costs are also $10.
- **Do this!**




Continental
T
Continental?
F
Alaska/Hawaii
Outside US
$5
$10

Exercise

- Fred sells bunches of flowers at the local shopping center.

- One day Fred's boss, Joe, tells Fred that at any time during the day he (Joe) will need to know:
  - how many bunches of flowers have been sold
  - what was the value of the most expensive bunch sold
  - what was the value of the least expensive bunch sold
  - what is the average value of bunches sold

We use -1 as "special" value to know when there are no more bunches

start

bunches = ∅
highest = ∅
lowest = 100 → assumes all prices are < 100
average = ∅
total = ∅

INPUT: b → price of the new bunch

b≠-1? (b)

F

AVG = TOTAL SUM / BUNCHES

T

bunches = bunches + 1

b > highest?

T

highest = b

F

b < lowest

T

lowest = b

F

total = total + b

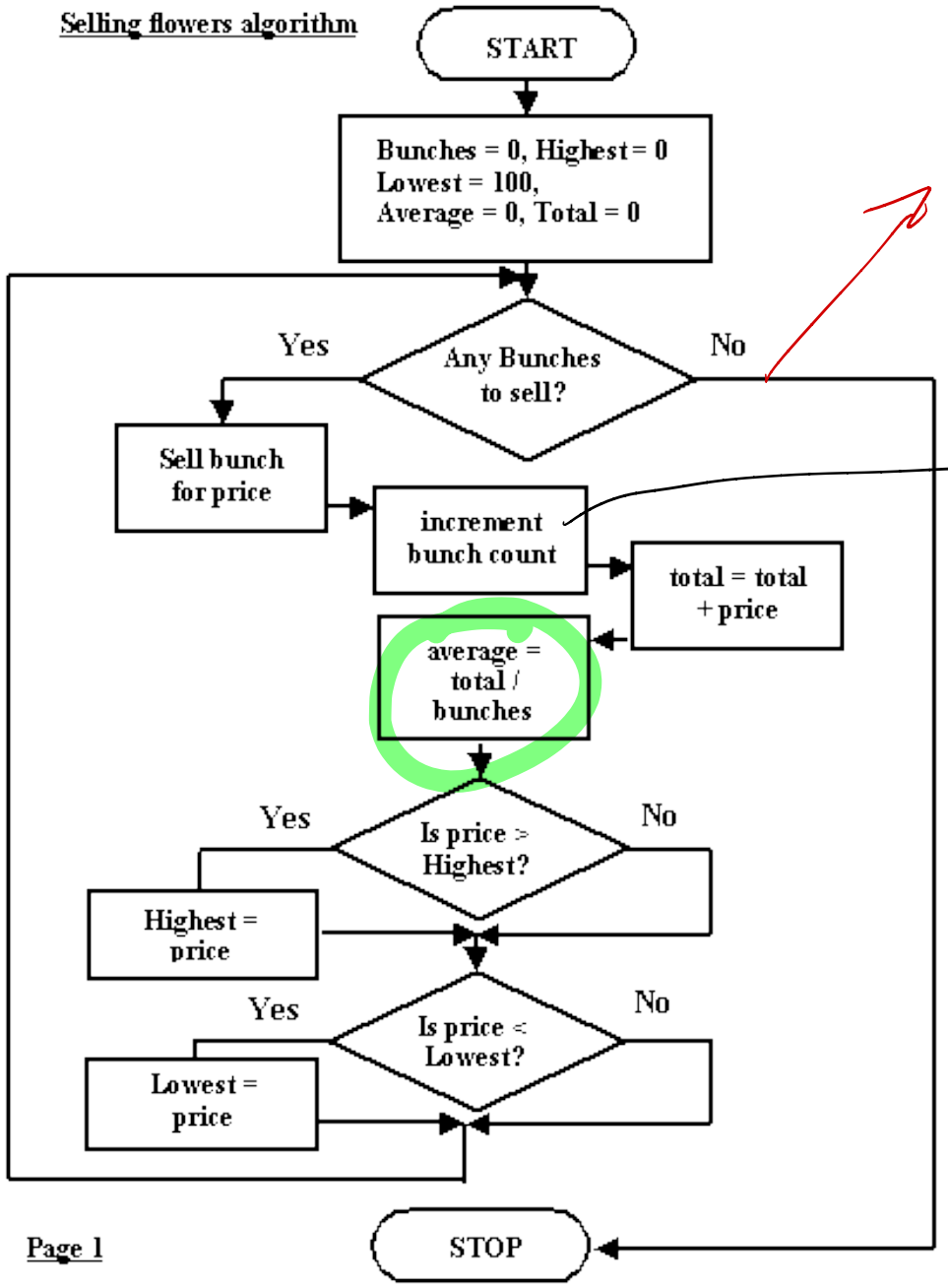
INPUT: b

average = total / bunches

OUTPUT: bunches
OUTPUT: highest
OUTPUT: lowest
OUTPUT: average

stop

Exercise

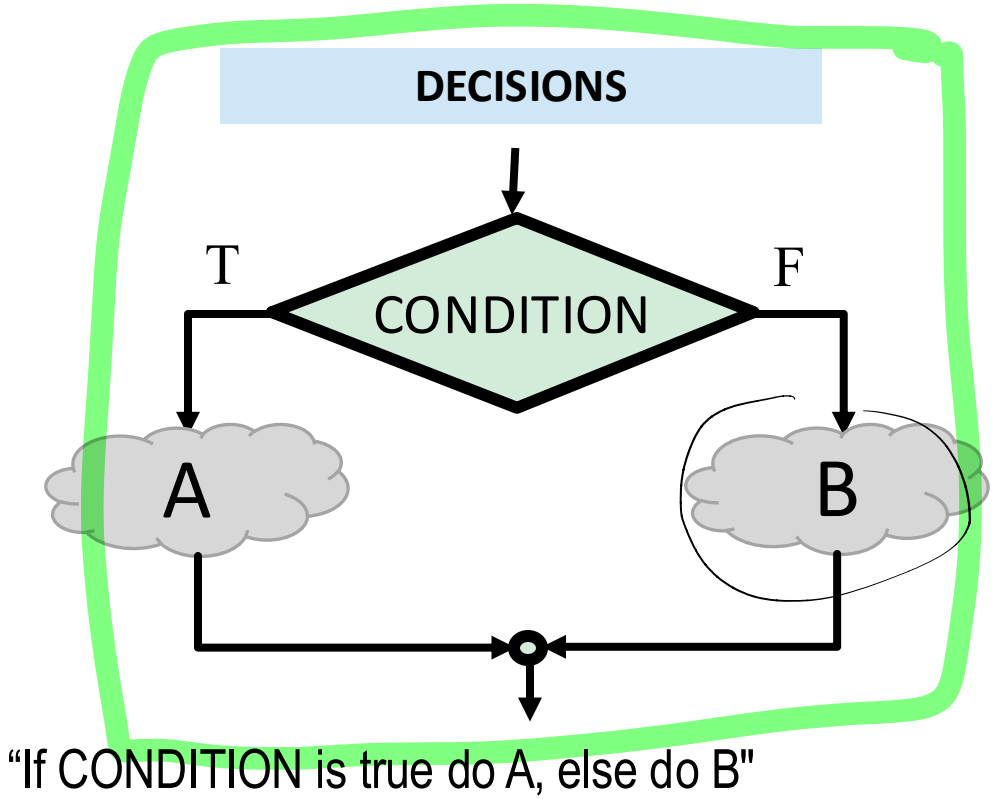


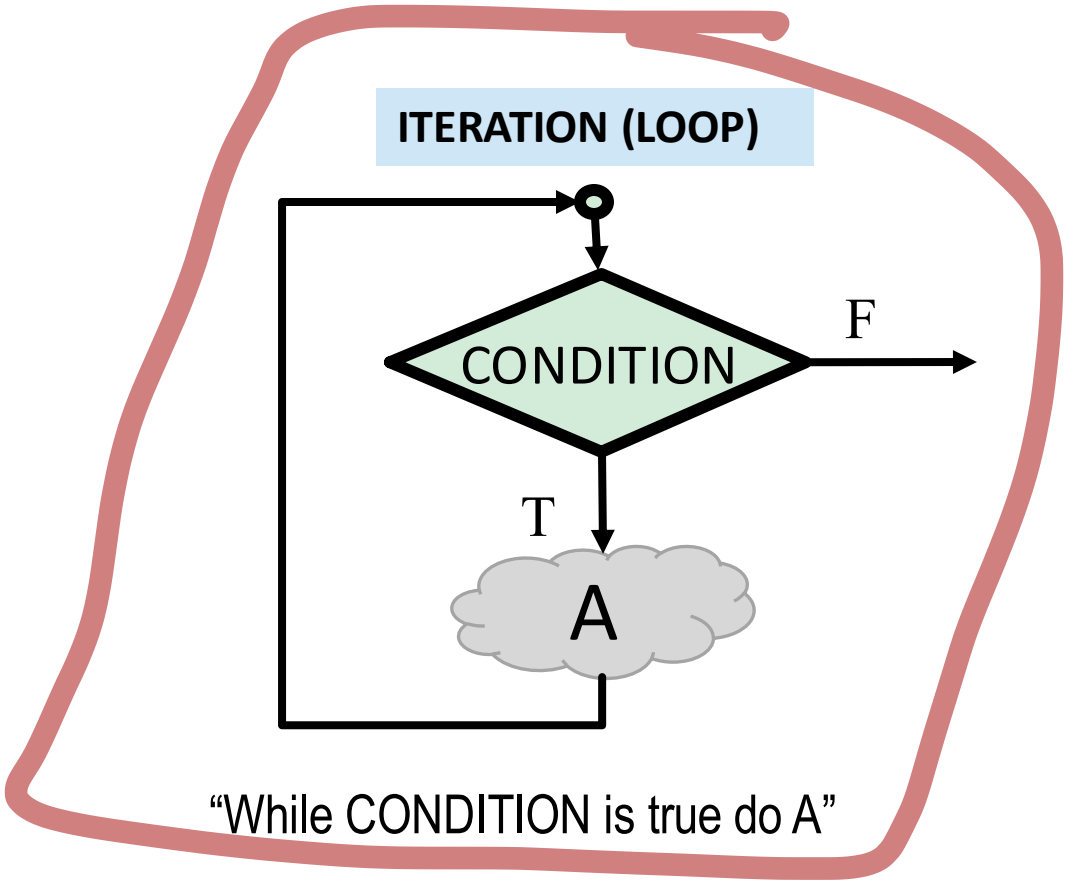
Hand-written Version more clear! (previous page)

bunches = bunches + 1

Try to think at a higher level

- Don't think just in terms of **SINGLE** blocks, but also in terms of **multi-block structures**
  - Two fundamental "structures"





Flowchart Exercises

GCD(A,B)

- Let's build a simple algorithm to find the GCD between two numbers
- Informal description

  For each value ***i*** between ***min(A,B)*** and 2 (in decreasing order), verify if both ***A*** and ***B*** are divisible by ***i.***

  If both conditions are true, then i is the GCD(A,B) and we can terminate

- Let's turn it into a flowchart.

⇒ Advanced: EUCLID's Algorithm

Start

INPUT : A

INPUT : B

FOUND = FALSE

A < B

T

m = A

F

m = B

m ≥ 2 AND FOUND = FALSE

T

"are both A and B divisible by M?"

A % m = 0? and B % m = 0?

F

FOUND = TRUE

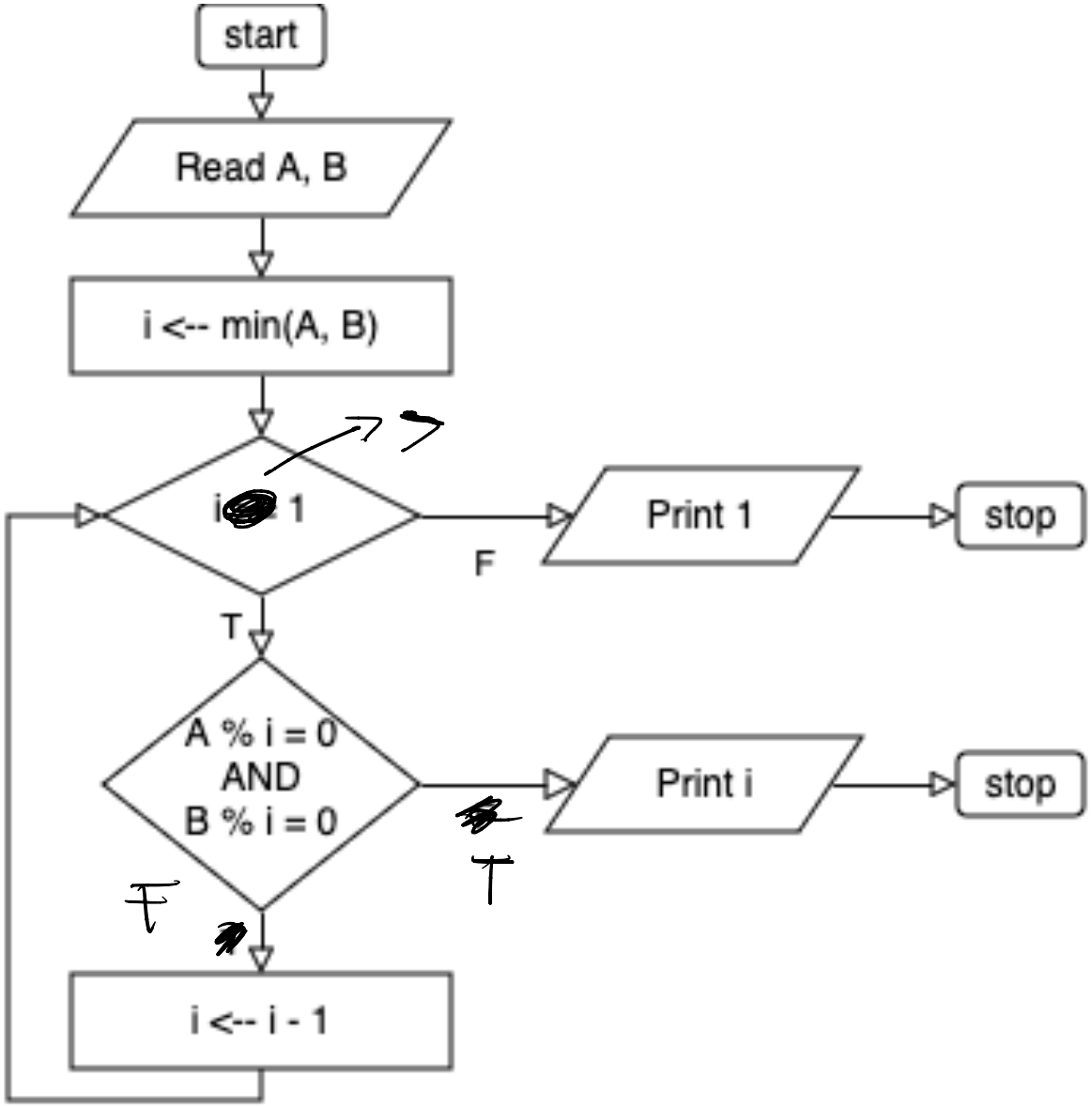
m = m - 1

F

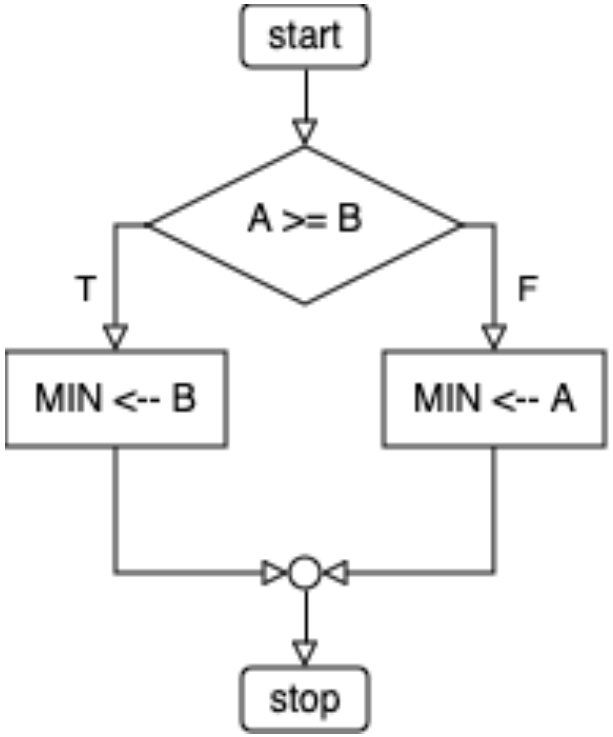
Print m

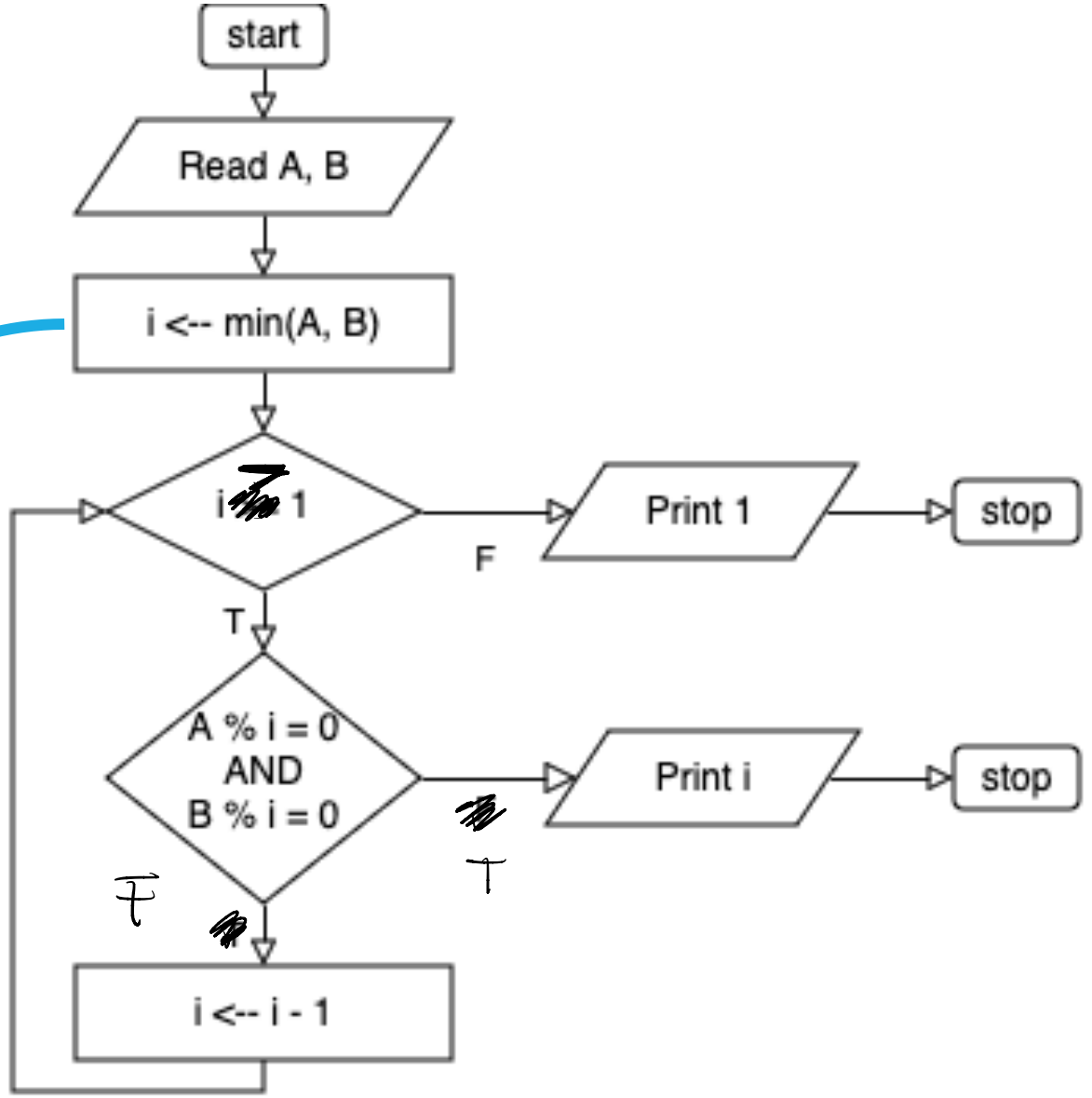
Stop

GCD (A, B)


start
Read A, B
i <-- min(A, B)
i = 1
F
Print 1
stop
T
A % i = 0
AND
B % i = 0
Print i
stop
i <-- i - 1

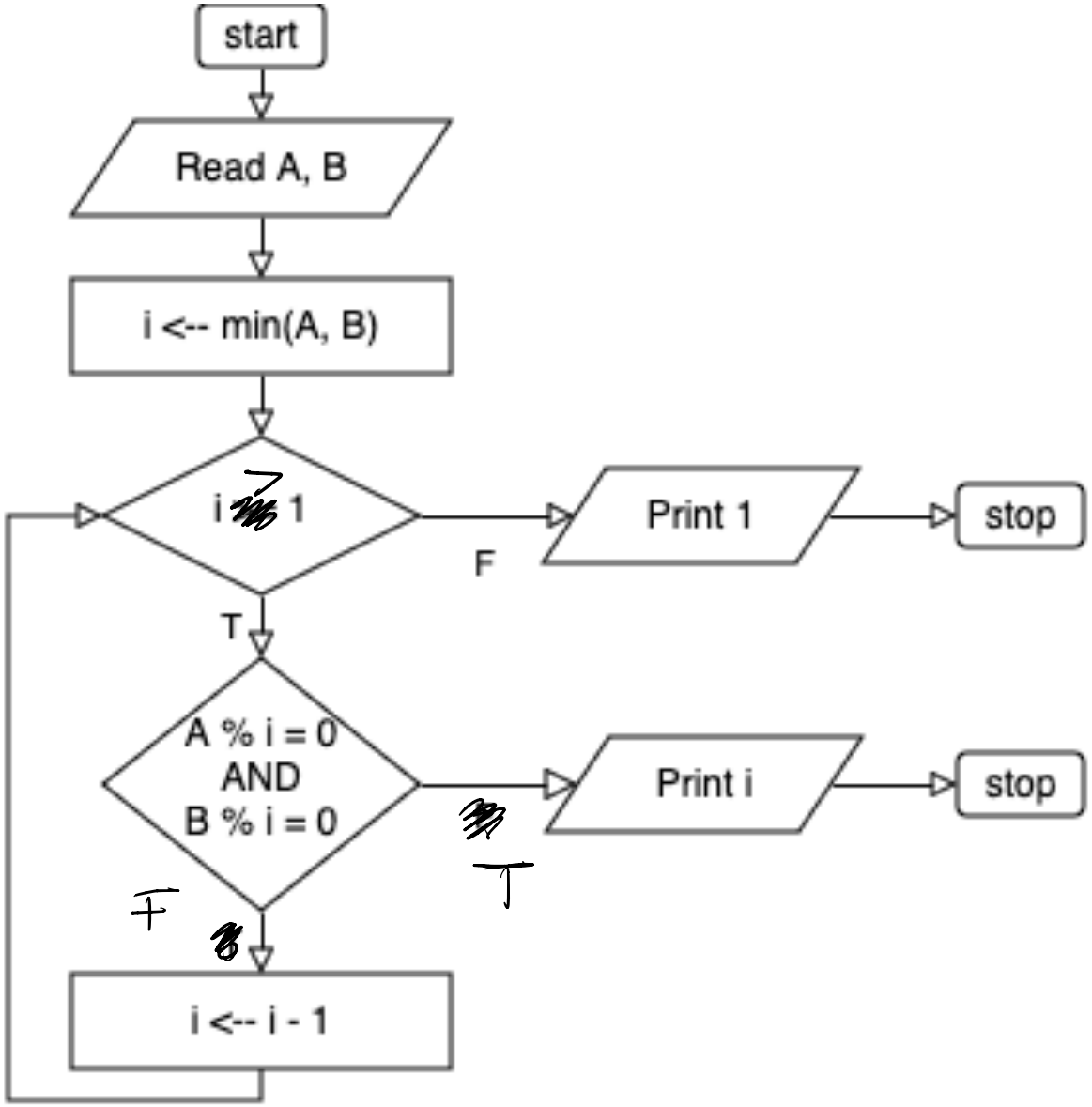
GCD (A, B)


start
A >= B
T
F
MIN <-- B
MIN <-- A
stop


start
Read A, B
i <-- min(A, B)
i >= 1
F
Print 1
stop
T
A % i = 0
AND
B % i = 0
Print i
stop
i <-- i - 1

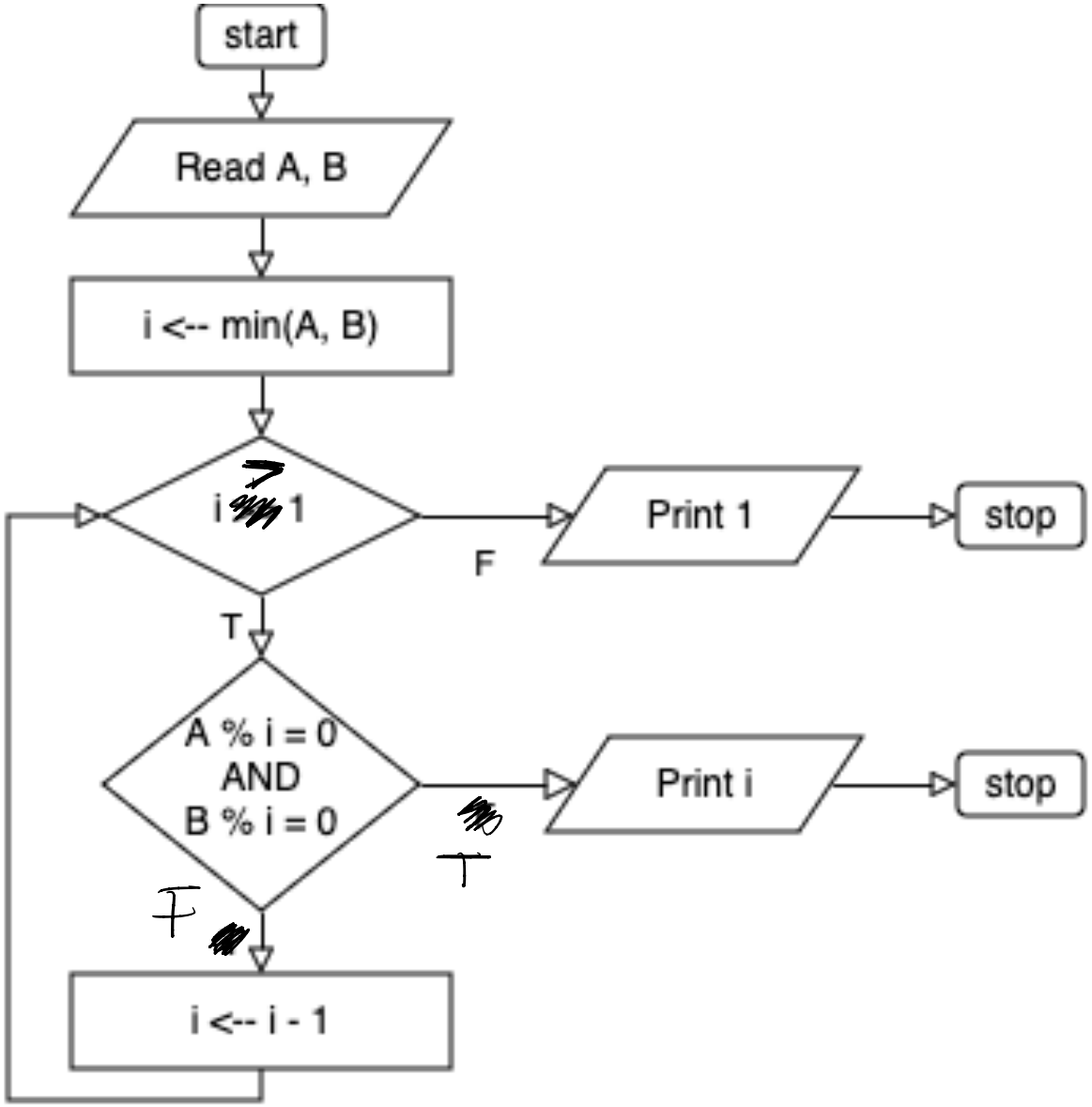
GCD (A, B)

- Example: **A=12, B=8**
- **Initialization: i ← min(A, B) = 8**
- **i >= 1? TRUE**
- **i divides A? FALSE | i divides B? TRUE**
- **i <-- 7**
- **i >= 1? TRUE**
- **i divides A? FALSE | i divides B? FALSE**
- **i <-- 6**
- **(…continues…)**



GCD (A, B)

- Example: **A=12, B=8**
- **(…continues…)**
- **i >= 1? TRUE**
- **i divides A? TRUE | i divides B? FALSE**
- **i <-- 5**
- **i >= 1? TRUE**
- **i divides A? FALSE | i divides B? FALSE**
- **i <-- 4**
- **i >= 1? TRUE**
- **i divides A? TRUE | i divides B? TRUE**
- **Print ”i” → Print “4”**
- **Stop**



Exercise

- Given an integer number **N** entered by the user, determine if it is a prime number
- Let's use the definition: "**divisible only by itself and 1**"
- Said otherwise: "**not divisible by any number between 2 and N-1**"

Exercise

- Solution?

More Exercises

You will find a video lecture on the course website with some exercises on flowcharts.