🔕

CS1010

Programming Methodology

Lecture 1

14 August 2018

Admin Matters

Unit 1: **What is a Program?**

Unit 2: **Computational Problems & Algorithms**

Ooi **Wei Tsang**

ooiwt@comp.nus.edu.sg
AS6 05-15

CS1010 Office Hours:
Wed 4-5pm

What will you learn?

- Comfortable with reading and developing small programs to solve a given problem

- Become proficient with C syntax and language and associated programming tools (editors, debuggers)

What do you need to know to be here?

- Nothing about writing programs
- Comfortable with using a computer
- Comfortable with secondary school math and notations

Weekly Activities

- **2-hour lecture**: Tues 4-6pm, weekly
- **2-hour tutorial / lab**: up to 15 students per lab, starting Week 3

UDLs and TAs

- **UDLs**: Your seniors who return to help your learn in CS1010
- **TAs**: PhD students who teach part-time / full-time

Open Book Exams

Not a memorisation-based module

Any APIs needed will be given

Module Homepage

`https://nus-cs1010.github.io/1819-s1/`

lecture notes
assignments
guides/manuals
etc.

piazza Q&A

https://piazza.com/class/jkqlna92ju045j

Important Dates

October 2, 2018 (Midterm)

November 27, 2018 (Final)

Important Dates

(tentative)

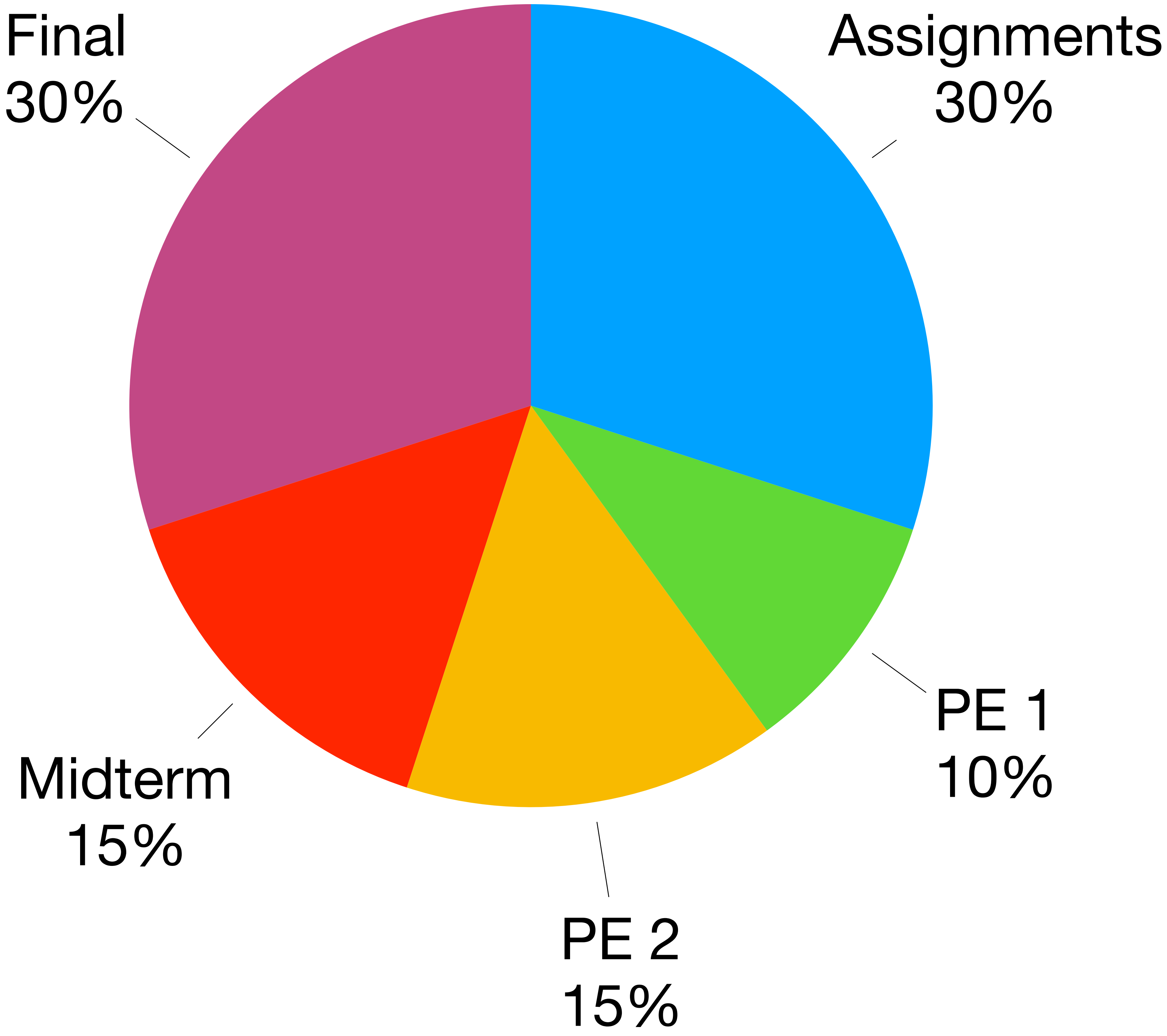
October 6, 2018 (PE1)

November 10, 2018 (PE2)

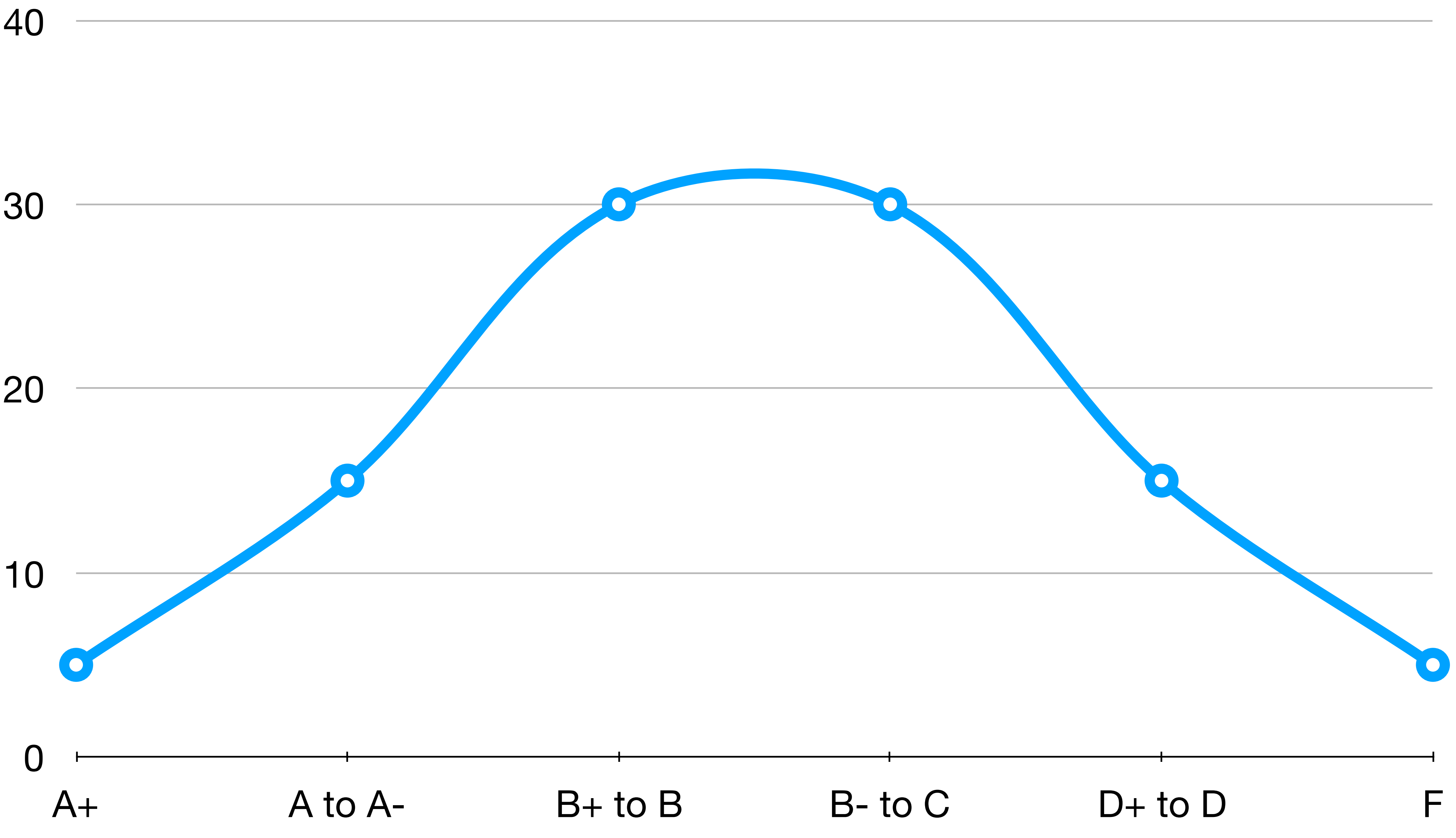
Open Book

Bring any materials you wish

Nothing to memorise


Final
30%
Assignments
30%
PE 1
10%
PE 2
15%
Midterm
15%

CS1010 is not graded on a bell curve



NOT the actual NUS bell curve

In CS1010, you will get the grade you deserved, irrespective of other students in the class.

Strict policy on plagiarism

Disciplinary action will be taken :(

Accounts

- Register for an SoC UNIX account if you do not have one at https://mysoc.nus.edu.sg/~newacct/

- Register for a GitHub account if you do not have one at https://www.github.com/

- Activate your Piazza account (link emailed to you)

Policies

- Read about other CS1010 policies at the module website
  - Using email and Piazza
  - Attendance
  - Late submission
  - Slides / screencast availability
  - …

Some words of advice

Work hard.
Very very hard.

At NUS SoC, learning is very different from your prior education

less rote learning;
more open-ended problems;
more self-learning

don’t optimise for grades and CAPs

optimise **your skills, knowledge and experience** instead

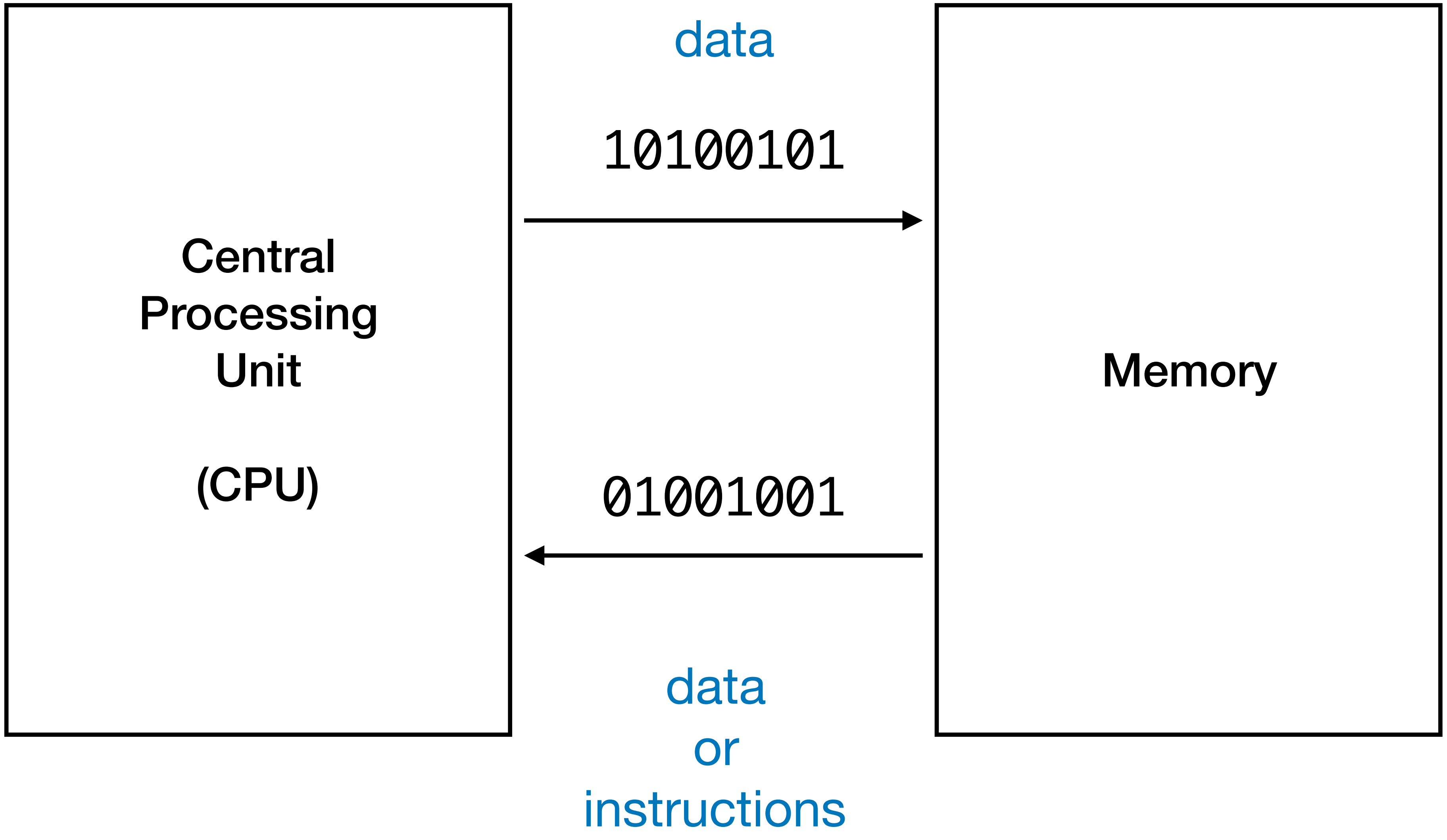
invest time to master the tools

(e.g., bash, vim, git, etc)
think long term benefits,
not how to get things done now

What is a program?
What is a computational problem?
What is an algorithm?

What is a program?

A Simplified View of a Computer



Machine Code

- Instructions that can be interpreted by a CPU

- Different CPU architecture may use different instruction sets

```
1001010001011110101001010101010100101…
```

Assembly Language

- Uses mnemonic to represent the instructions such as `incr`, `decr`, `store`, etc., in a more human readable way.

- Still difficult to write: operation that is conceptually simple still requires multiple instructions to be issued.

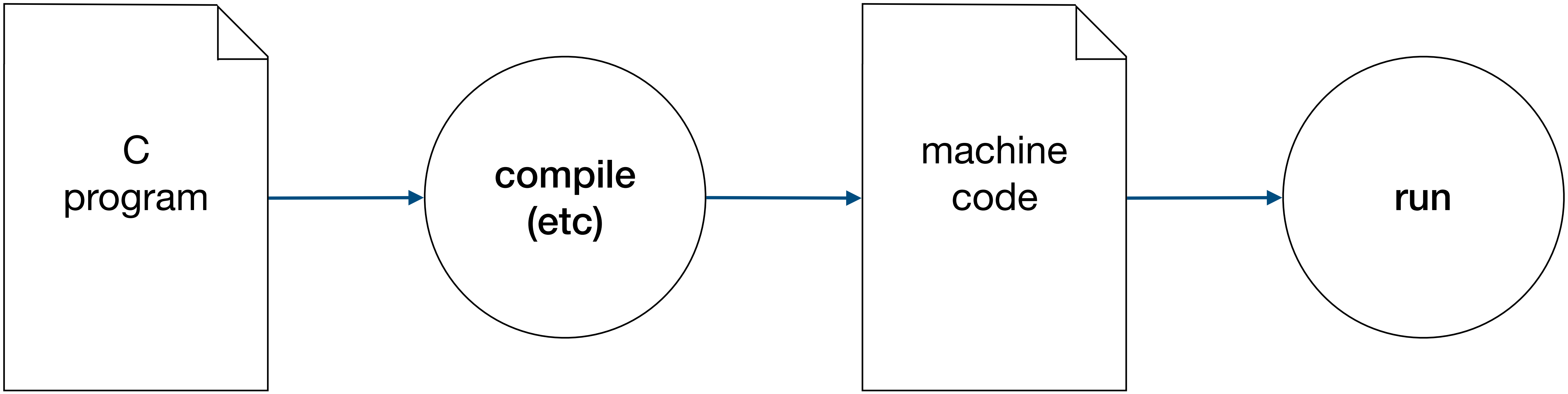
Higher-Level Language

- Languages that allow programmers to express an operation closer to their intention
- e.g., `x = x + y if (y < 0)`

Higher-Level Languages

- FORTRAN (first popular PL, 1950s)
- C (1972, still influential today)
- Many many others

Compiling High-Level Program to Machine Code



What will you learn?

How to write C

How a C program behaves

Tools and techniques to help write good and correct C programs

How to use C to solve computational problems

Learning to write a program that does what you want it to do is actually not difficult.

Knowing what you want your program to do is the more challenging part!

What is a computational problem?

Computational Problems

- Problems that can be solved step-by-step by a computer

- Have well-defined inputs, outputs, and constraints

Is 1933091 prime?

Is there a route from a point A to point B?

Find a route from a point A to point B

How many different routes are there from a point A to point B?

Find the fastest route
from a point A to point B

What is the meaning of life?

Do I look good in this outfit?

Computational Problems

- Problems that can be solved step-by-step by a computer

- Have well-defined inputs, outputs, and constraints

Find the maximum number

5 9 8

1 3 2

Find the maximum

12	20	11	20	14	2	24	36	43	16	27	6	11	7
10	15	38	10	22	7	16	26	32	30	11	9	30	6
20	6	27	19	26	10	27	6	19	34	5	9	3	22
10	4	13	1	10	22	43	36	39	29	41	12	13	25
17	8	30	31	29	38	2	42	7	45	24	33	9	40
34	29	37	2	26	17	19	34	6	7	34	22	21	41
38	5	15	13	9	1	42	39	5	29	38	4	22	29
41	10	26	32	30	26	16	16	18	22	32	34	14	10
5	17	25	16	19	6	31	16	3	13	8	42	41	0
13	30	44	1	41	14	5	39	40	38	6	37	38	9

How to Solve It?

- Scan the integers one by one
- Keep track of the “max so far”
- When we are done scanning, the “max so far” will be the answer.

Problem Formulation

- **Input:** $L = \{l_0, l_1, \ldots, l_{k-1}\}$
- **Output:** m
- **Constraint:** $m \in L$ and $m \geq l_i$ for all i.

“Scan one-by-one”

- l_i : the number being scanned
- i = 0, 1, … k-1

“Maximum so far”

- m: max so far
- $m = l_0$ initially

Keeping Track..

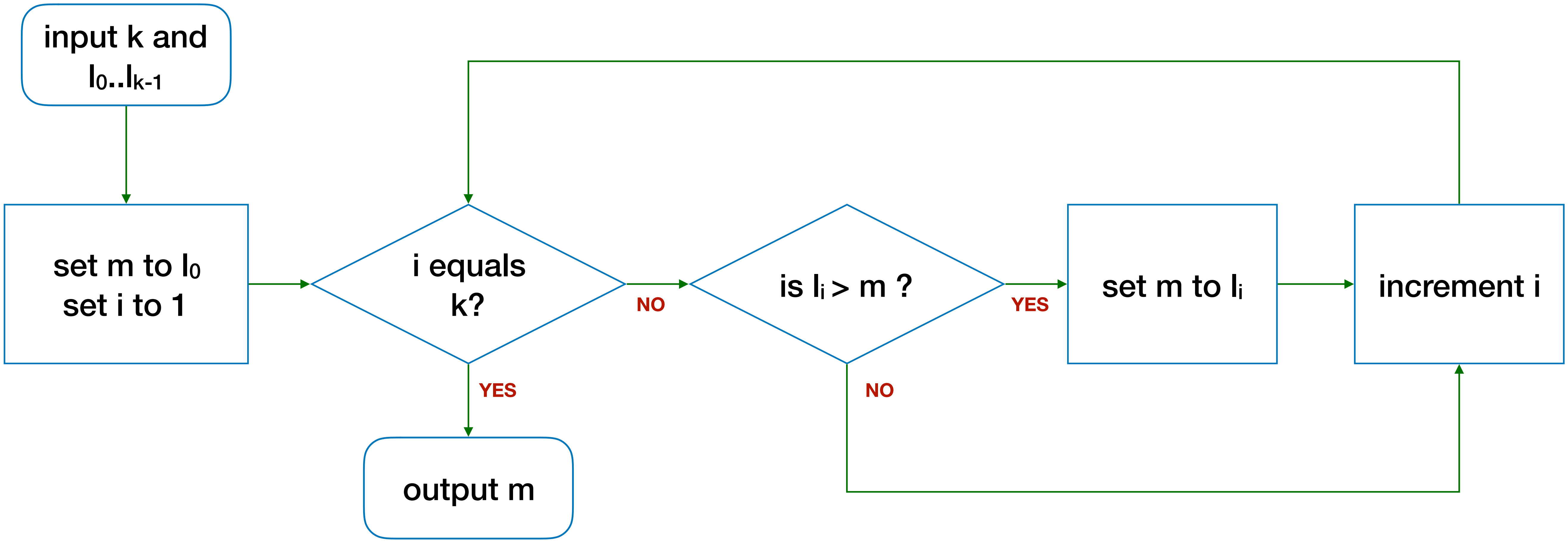
- Compare m with l_i
- If $m < l_i$, then update m with value of l_i

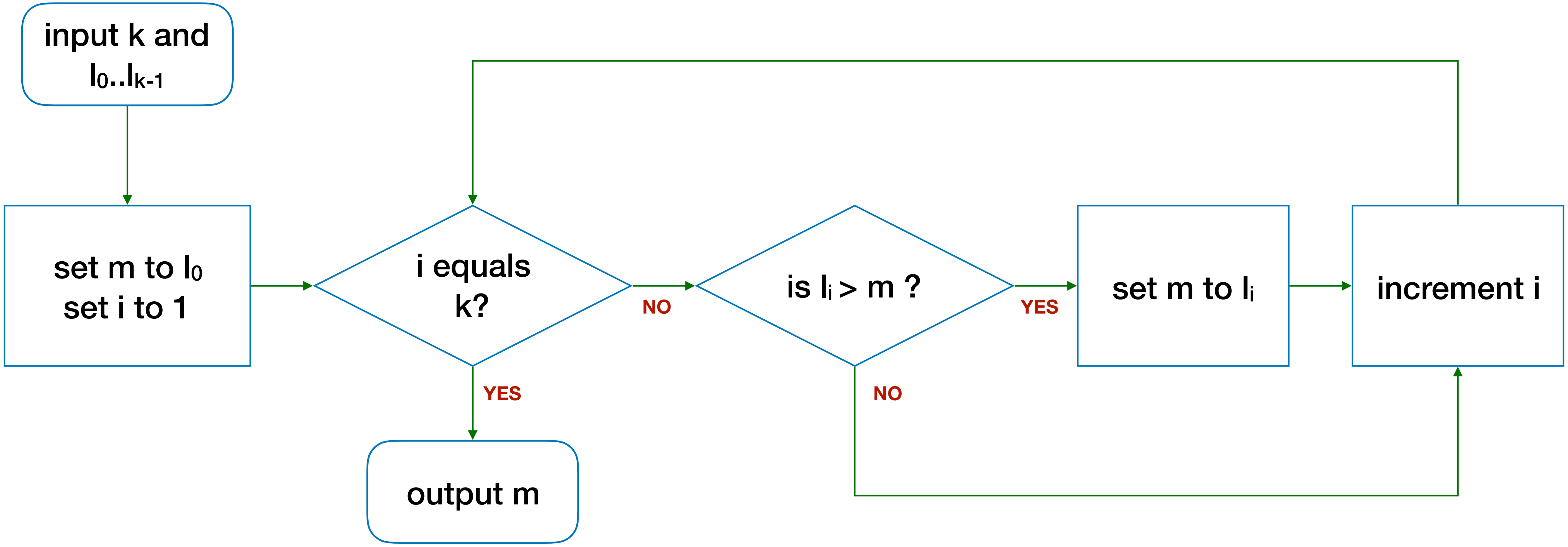
Done Scanning?

- When we have scanned l_{k-1}

The steps to solve a computational problem are called “algorithm”

Flowchart


input k and $l_0..l_{k-1}$
set m to l_0 set i to 1
i equals k?
NO
YES
output m
is $l_i > m$?
YES
NO
set m to l_i
increment i



input k and $l_0..l_{k-1}$

set m to l_0
set i to 1

i equals k?

NO

is $l_i > m$?

YES

set m to l_i

increment i

YES

NO

output m

Input: $L = \{ 0, 2, 1\}$, $k = 3$

k	**i**	**l_i**	**m**
3	1	2	0
3	2	1	2
3	3	?	2
3			

Variables

- *i*, *m*, and *k* are called variables
- They are location in memory that holds a value
- You can read a value from a variable
- You can write a value to a variable

Assignment

- We can assign the value of a variable to a constant (e.g., set *i* to 1)
- or to the value of another variable (e.g., set *m* to l_i)

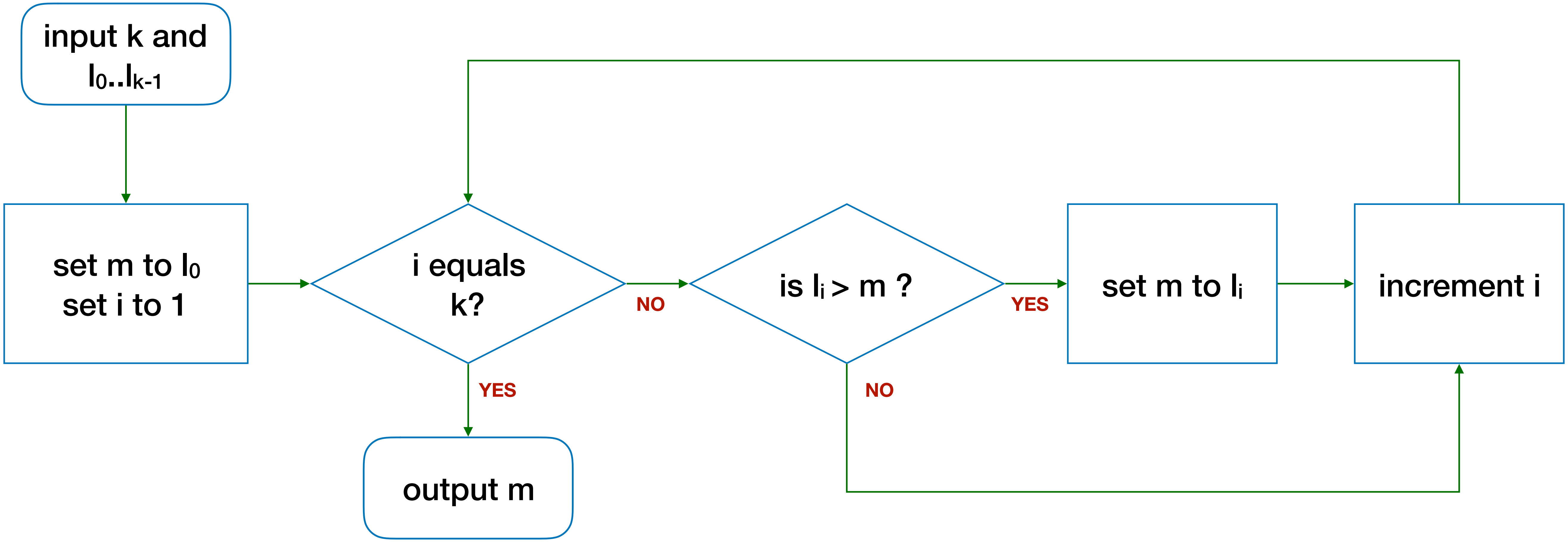
Comparison

- We can compare the values of two variables
- E.g., i equals k?, $li > m$?

Arithmetic Operations

- We can perform arithmetic operations (+, -, etc)
- E.g., set *i* to *i* + 1

Does this algorithm always find the maximum value correctly?



Correctness of Algorithm

- If an algorithm produces the correct output for one instance of input, it is not necessary correct
- To be correct, it has to produce the correct output for all possible inputs!

How to check for correctness?

- Carefully choose a set of test cases to try
- Argue formally about the properties of the algorithms

Homework

- Try out Problem Set 1.1 to 1.3
- We will discuss in Week 3 !