Topic 1
On Programming

CS 1MD3 • Introduction to Programming
Winter 2018

Dr. Douglas Stebila
with modifications by Nicholas Moore

McMaster University

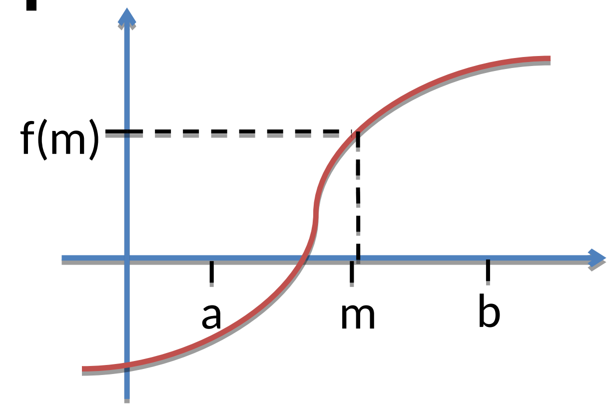
Formulating a problem

- What is the square root of 25?

- What is the square root of y?
  - "The square root of y is a number x such that x * x = y."

Formulating a problem

- What is the x-intersect of a function f?



An **x-intersect** of function f is an x such that f(x) = 0.

Suppose f is monotonically increasing on [a, b] and:
f(a) ≤ 0, f(b) ≥ 0
To determine the **x-intersect** with precision ε > 0:

1. as long as b – a > ε, do 2. – 4.
2. calculate m = (a + b)/2
3. if f(m) ≤ 0, set a to m, or
4. if f(m) ≥ 0, set b to m

The result is (a, b) such that
f(a) ≤ 0, f(b) ≥ 0, b – a ≤ ε

Declarative vs Imperative Descriptions

Declarative descriptions

- Declarative descriptions state the **properties** of the result. They describe what the result is.
- They are typically short, but cannot be followed.
  - In the case of x-intersect, there could be several possible results or there could be none

Imperative descriptions

- Imperative descriptions tell **how** the result is obtained by given a **sequence of instructions**.
- Each step is simple enough that it can be followed.
  - In the case of x-intersect, restrictions on the input are imposed (f monotonic), additional input is needed (a, b, ε), and the result is only approximate (with ε). There is repetition (line 1) and selection (lines 3, 4).

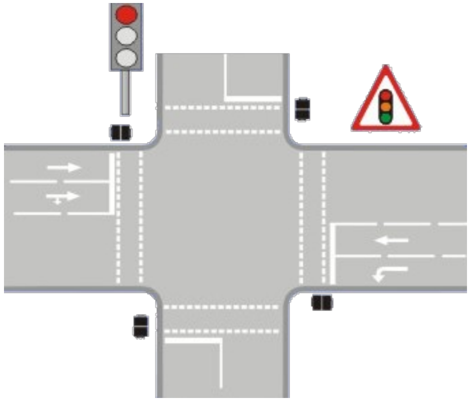
Algorithms

- An algorithm specifies a **sequence of instructions** to be executed by a **machine** that, when provided with **input**, will eventually stop and produce some **output**.

Algorithms

A cookbook recipe is an algorithm to be executed by a human.

- **Inputs**: ingredients.
- **Instructions:** recipe instructions.
- **Outputs:** yummy food.



A traffic light follows an algorithm for switching the light.

- **Inputs:** time, car presence sensors
- **Instructions:** conditions on which lights should be on
- **Outputs:** electrical signals to lightbulbs

Algorithms

- The **computation** prescribed by an algorithm goes through a sequence of **states**, starting from an **initial state**, and each instruction leading to a new state.
- The **trace** is the resulting sequence of states.

9th-century Persian mathematician Al-Khwārizmī published an arithmetic treatise on calculating certain values.

The Latin translation was called *Algoritmi de numero Indorum*, "Algoritmi on the numbers of the Indians".

Properties of Algorithms

Algorithms are supposed to be executed **mechanically**:

- Algorithms must be **precise**: each instruction and the next possible instruction to be taken must be unambiguous.

- Algorithms must be **effective**: each instruction must be executable by the underlying machine.

Computing the x-intersect

Input:

- a range [a, b]
- a function f that is monotonically increasing on [a, b] such that $f(a) \le 0$ and $f(b) \ge 0$
- a precision $\varepsilon > 0$:

Instructions:

1. As long as $b - a > \varepsilon$, do steps 2–4
2. Calculate $m = (a + b)/2$
3. If $f(m) \le 0$, set a to m
4. Otherwise, if $f(m) \ge 0$, set b to m

Output:

- Values (a, b) such that $f(a) \le 0$, $f(b) \ge 0$, and $b - a \le \varepsilon$

Question: what happens if $f(m) = 0$ after instruction 2?

A. The algorithm is not precise because it is ambiguous

B. The algorithm is precise (unambiguous), but the inputs which lead to $f(m) = 0$ are not allowed

C. Either instruction 3 or 4 is executed, it doesn't matter

Computing the x-intersect

Input:

- a range [a, b]
- a function f that is monotonically increasing on [a, b] such that $f(a) \leq 0$ and $f(b) \geq 0$
- a precision $\varepsilon > 0$:

Instructions:

1. As long as $b - a > \varepsilon$, do steps 2–4
2. Calculate $m = (a + b)/2$
3. If $f(m) \leq 0$, set a to m
4. Otherwise, if $f(m) \geq 0$, set b to m

Output:

- Values (a, b) such that $f(a) \leq 0$, $f(b) \geq 0$, and $b - a \leq \varepsilon$

Question: is the output of the algorithm always unique?

A. Yes

B. No

Fundamental questions about algorithms

Correctness

- What output is produced for each input?
- Is it what we "intended"?

Efficiency

- How long does it take to produce the output for a particular input?
  - Measured in terms of time
  - Measured in terms of instructions
  - In the worst case?
  - On average?
- How many resources (like memory) are needed in the process?

Computing the x-intersect

Input:

- a range [a, b]
- a function f that is monotonically increasing on [a, b] such that $f(a) \leq 0$ and $f(b) \geq 0$
- a precision $\varepsilon > 0$:

Instructions:

1. As long as $b - a > \varepsilon$, do steps 2–4
2. Calculate $m = (a + b)/2$
3. If $f(m) \leq 0$, set a to m
4. Otherwise, if $f(m) \geq 0$, set b to m

Output:

- Values (a, b) such that $f(a) \leq 0$, $f(b) \geq 0$, and $b - a \leq \varepsilon$

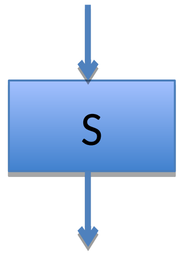
Question: can we compute the exact x-intersect by taking $\varepsilon = 0$?

A. Yes

B. No

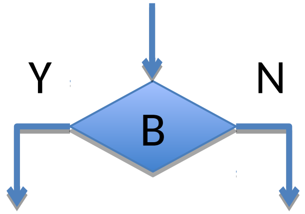
Algorithms as flowcharts

- Algorithms can be formulated textually or as flowcharts. **Flowcharts** connect instructions and conditions:



Instruction (statement, command, action), e.g.

- turn green light on
- continue straight ahead
- subtract v from u



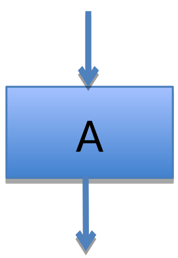
Condition (Boolean expression, test), e.g.

- drum is full
- water boils
- u is not equal to v

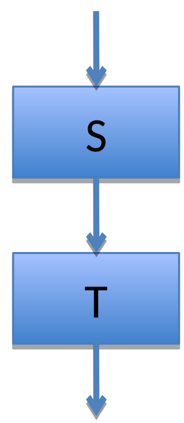
Structured Flowcharts

Hierarchically structured flowcharts composed of:

Basic instruction:

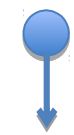


Sequence of instructions:

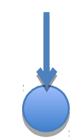


"S then T"

Start (optional):



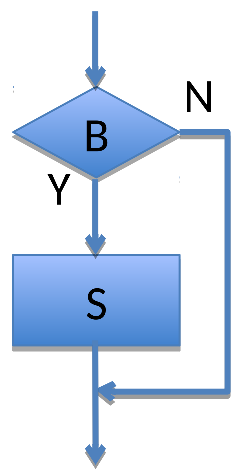
End (optional):



Structured Flowcharts

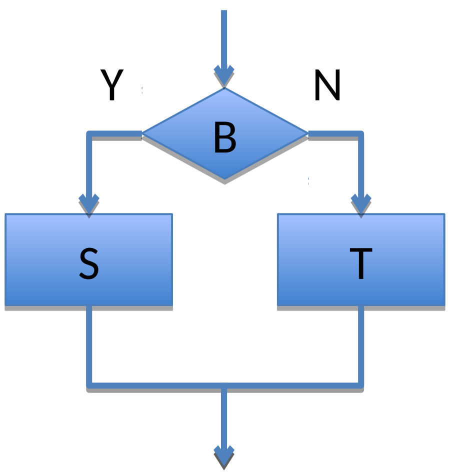
Hierarchically structured flowcharts composed of:

Selection:



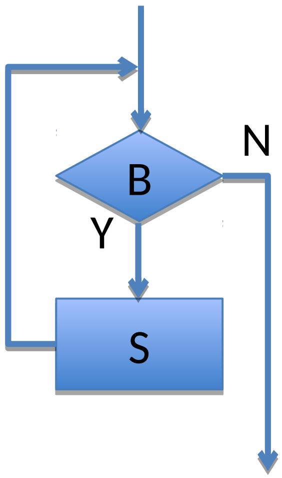
"if B then S"

Selection:



"if B then S else T"

Repetition:



"while B do S"

Variables and Memory

- The most basic way to track data in a computer program is the ***variable***
- Variables are named locations in memory where data is stored.
- Unlike variables in algebra, variables in computer programs always have a ***concrete value***.
- All program instructions read to or write from some type of memory (CPU registers, RAM, ROM, etc.)

Assignment Instructions

Assignment changes the value of variables in memory.

Mathematical Notation:

$$x := e$$

In Python:

x = e

“x becomes e,” where e is any syntactically correct expression

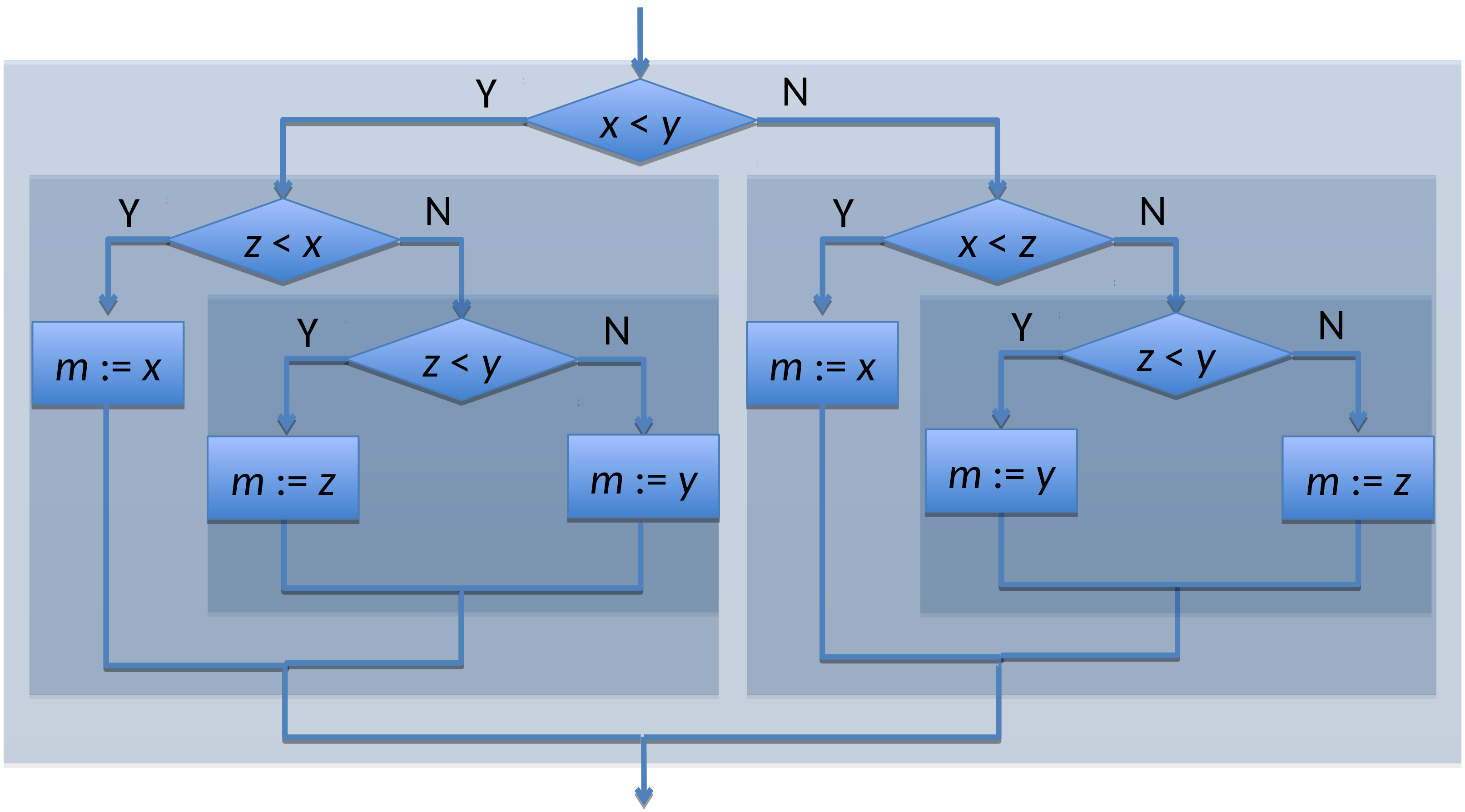
$$x := (7 + (y \div 2))^2$$

In Python:

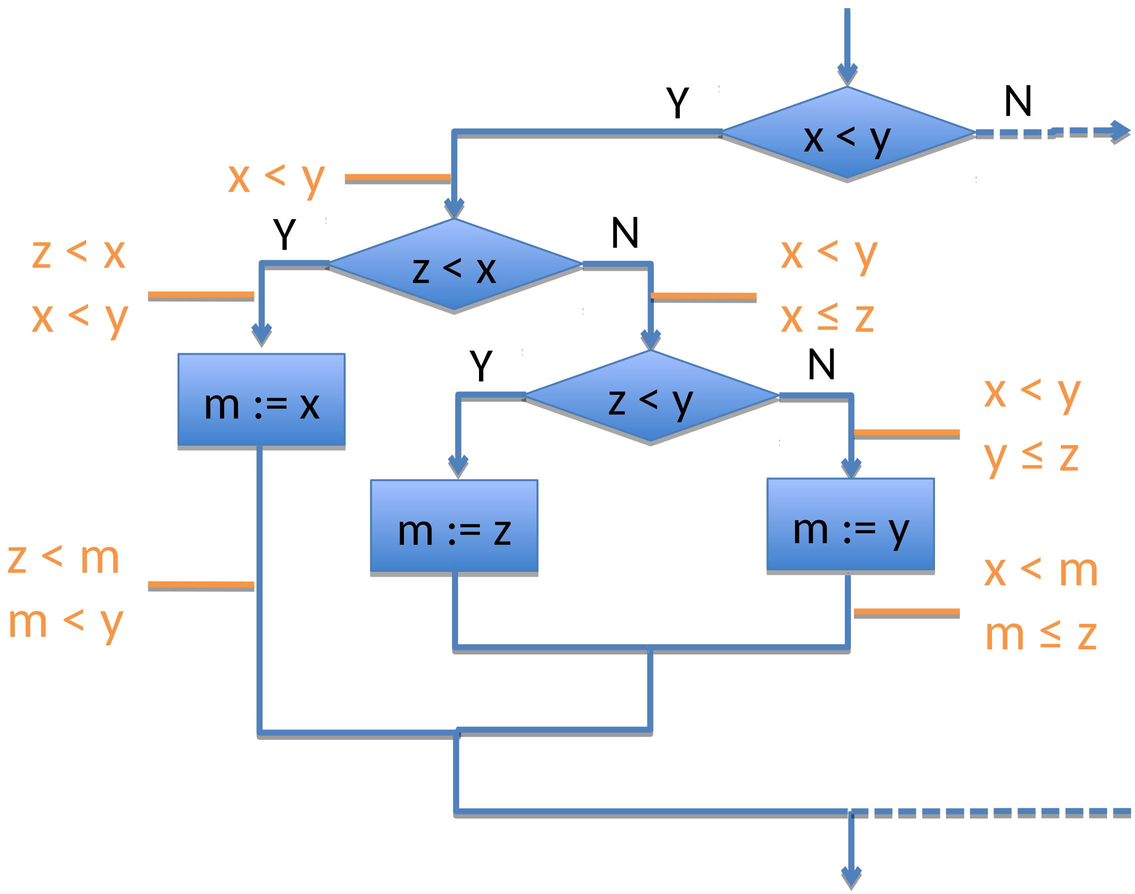
```
X = ( 7 + ( y / 2 ) ) * * 2
```

Flowchart for median of three numbers

The **median** of *x*, *y*, *z*, is the “middle” number. The median of 3, 7, 2 is 3. The median of 3, 7, 3 is also 3. Assign the median of *x*, *y*, *z* to *m*!



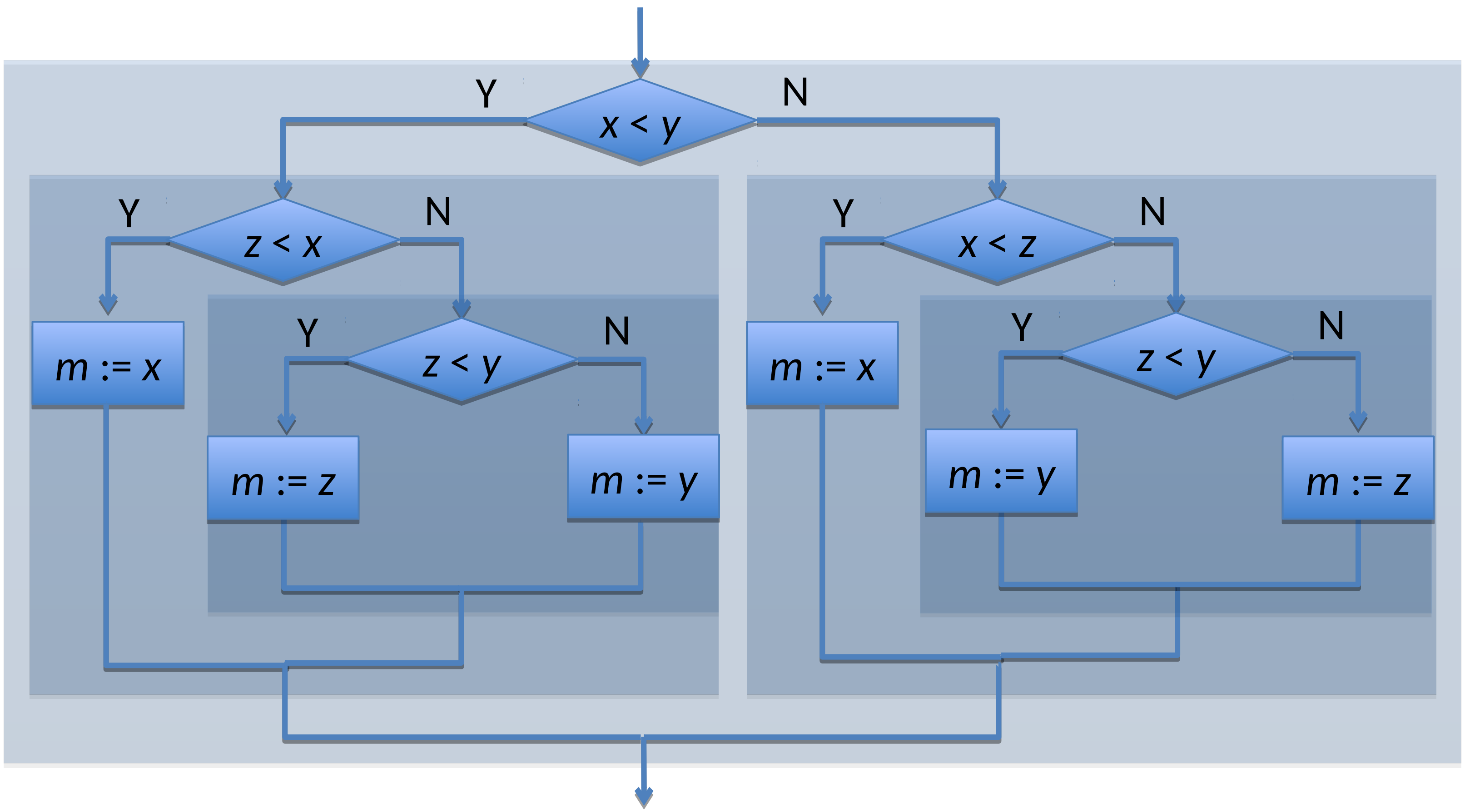
Annotating a flowchart



- **Annotations express what holds at particular points**
- They are declarative descriptions
- They help understanding algorithms and arguing their correctness
- It is good practice to explicitly write the main annotations
- There is a danger of over-annotating!

Flowchart for median of three numbers

What is the maximum number of operations required to compute the median of 3 numbers? 4



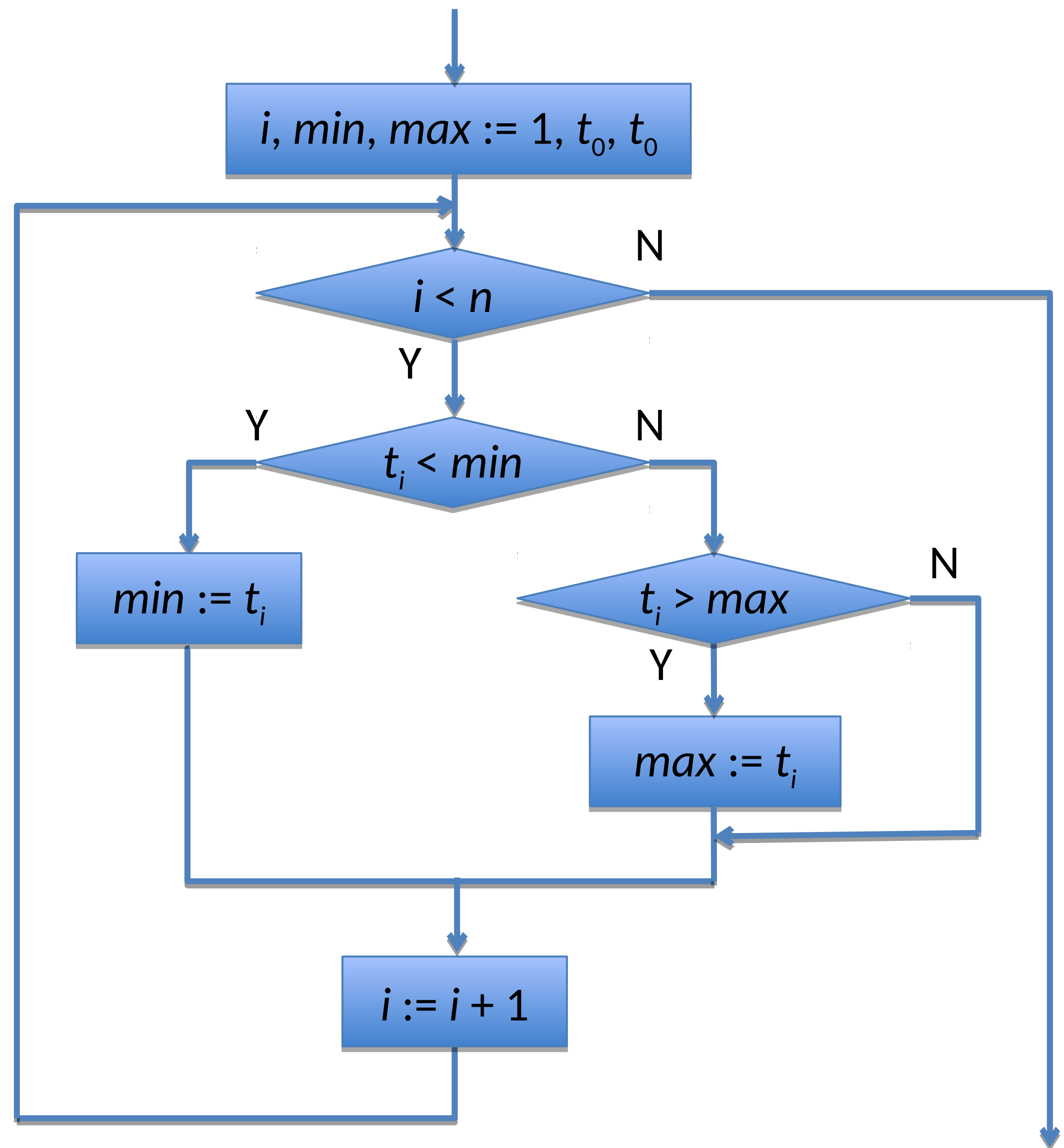
Flowchart for Min and Max Temperature

Inputs:

- integers t_i for $0 \leq i < n$, a series of n temperature readings
- integer $n > 0$

Output:

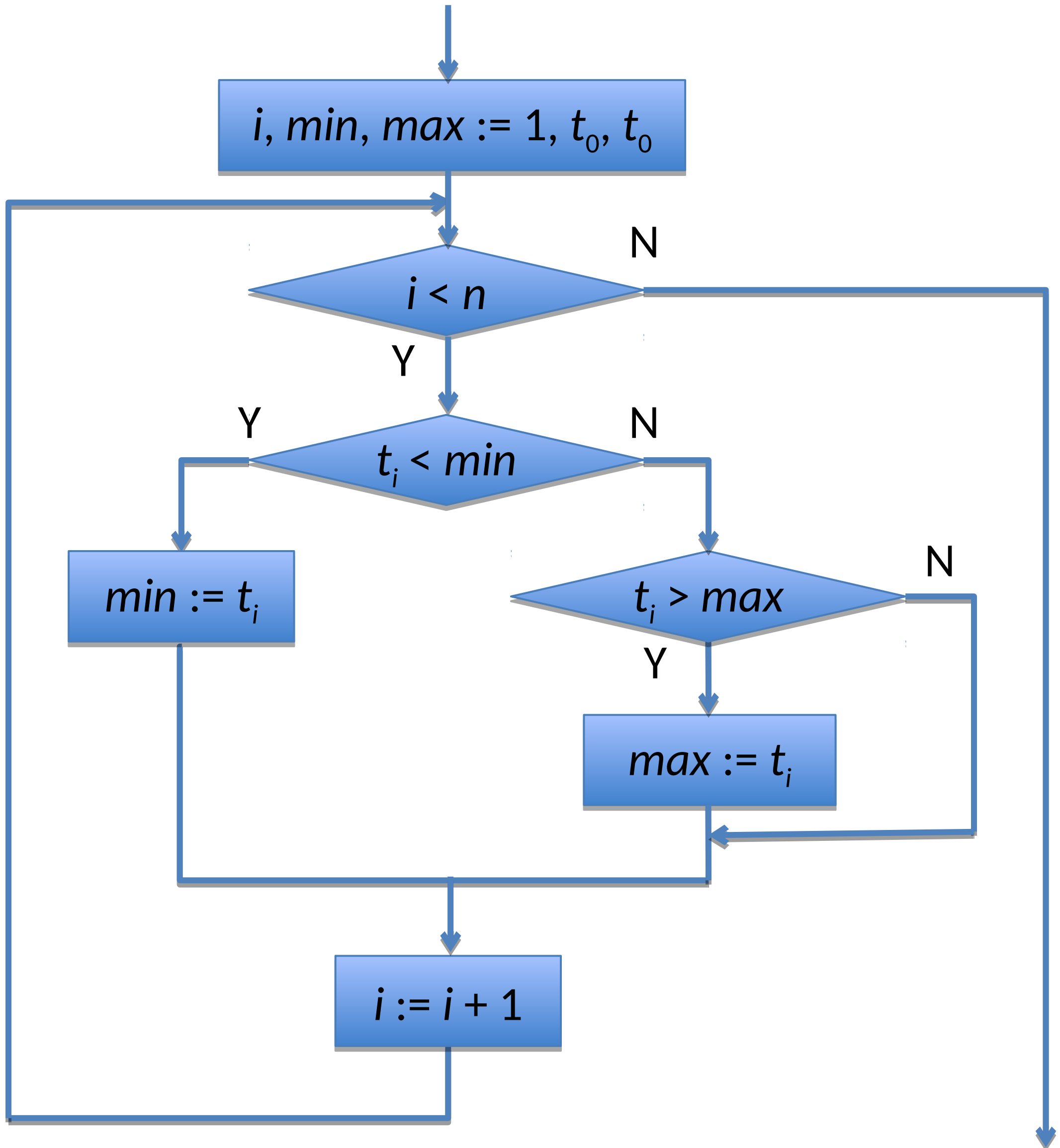
- minimum temperature *min*
- maximum temperature *max*



Flowchart for Min and Max Temperature

Inputs:
- $t_0 = 3$
- $t_1 = 7$
- $t_2 = 4$
- $n = 3$

- min = 3, max = 3
- i = 1
  - i < n
  - t_i not < min
  - t_i > max
  - max = 7
- i = 2
  - i < n
  - t_i not < min
  - t_i not > max
- i = 3
  - i not < n



Steps for Min and Max Temperature

What is the **minimum** number of comparisons with t_i when n = 3?

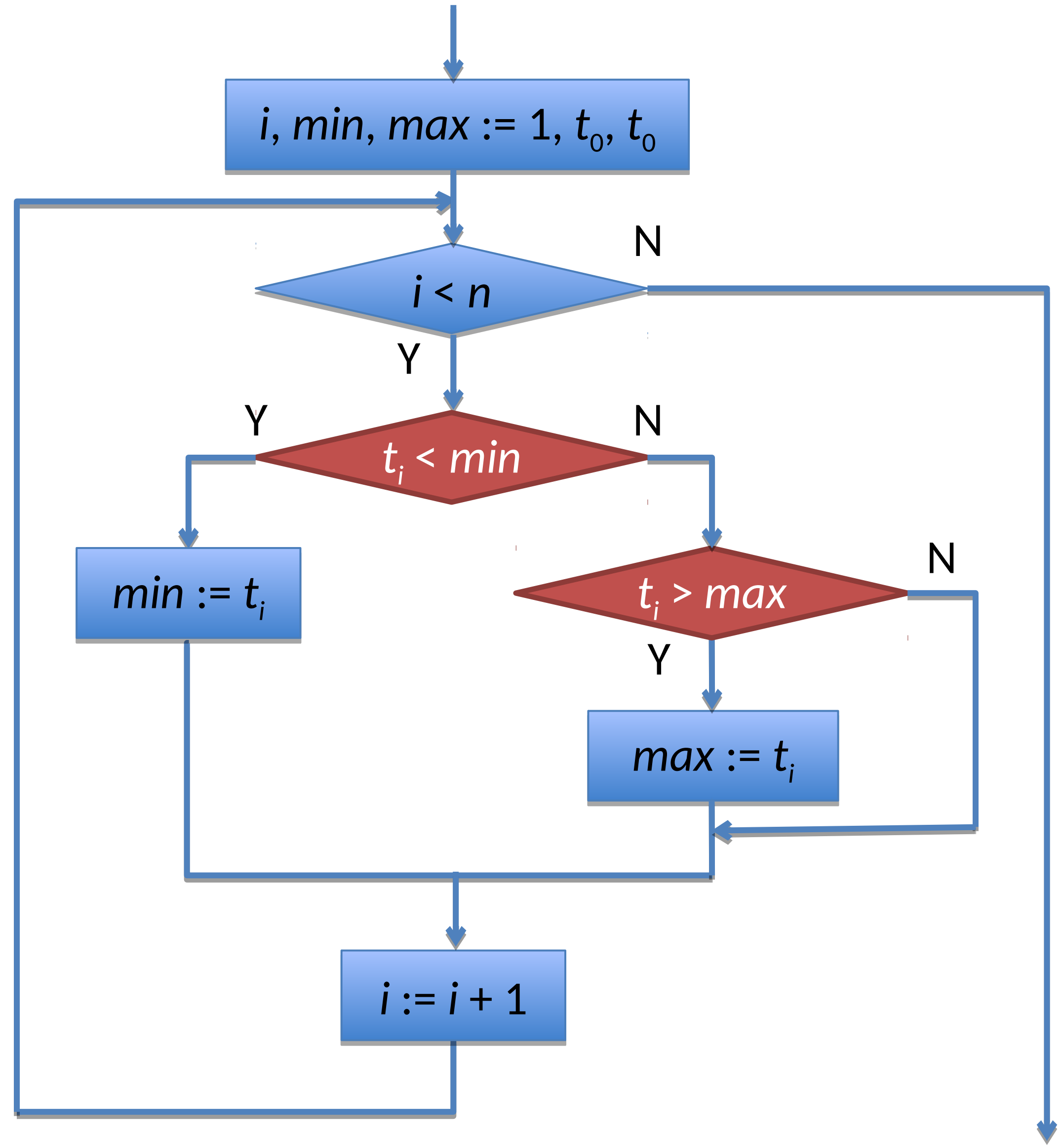
A. 0

B. 1

C. 2

D. 3

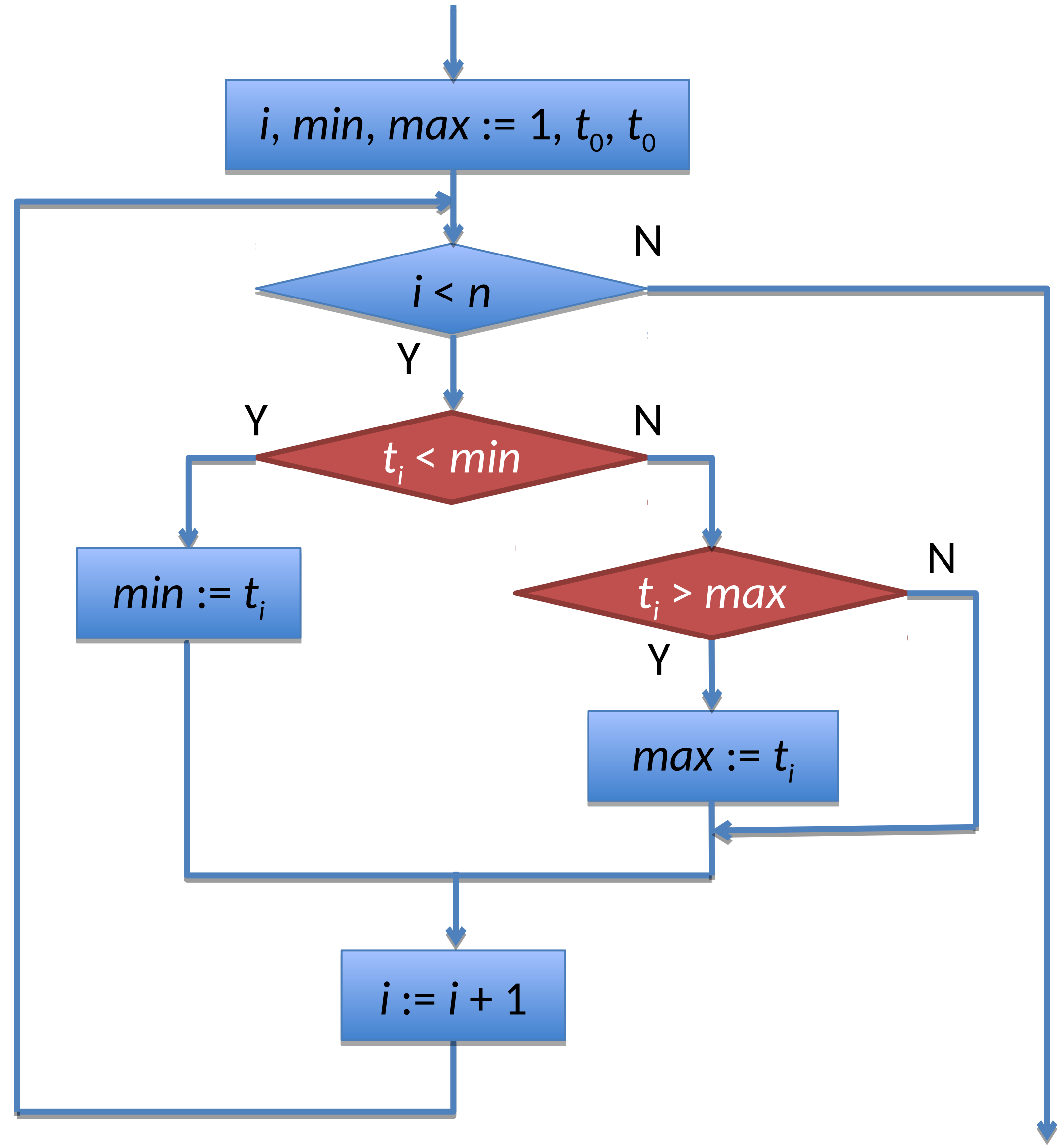
E. 4



Steps for Min and Max Temperature

What is the **maximum** number of comparisons with t_i when n = 3?

A. 0
B. 1
C. 2
D. 3
E. 4



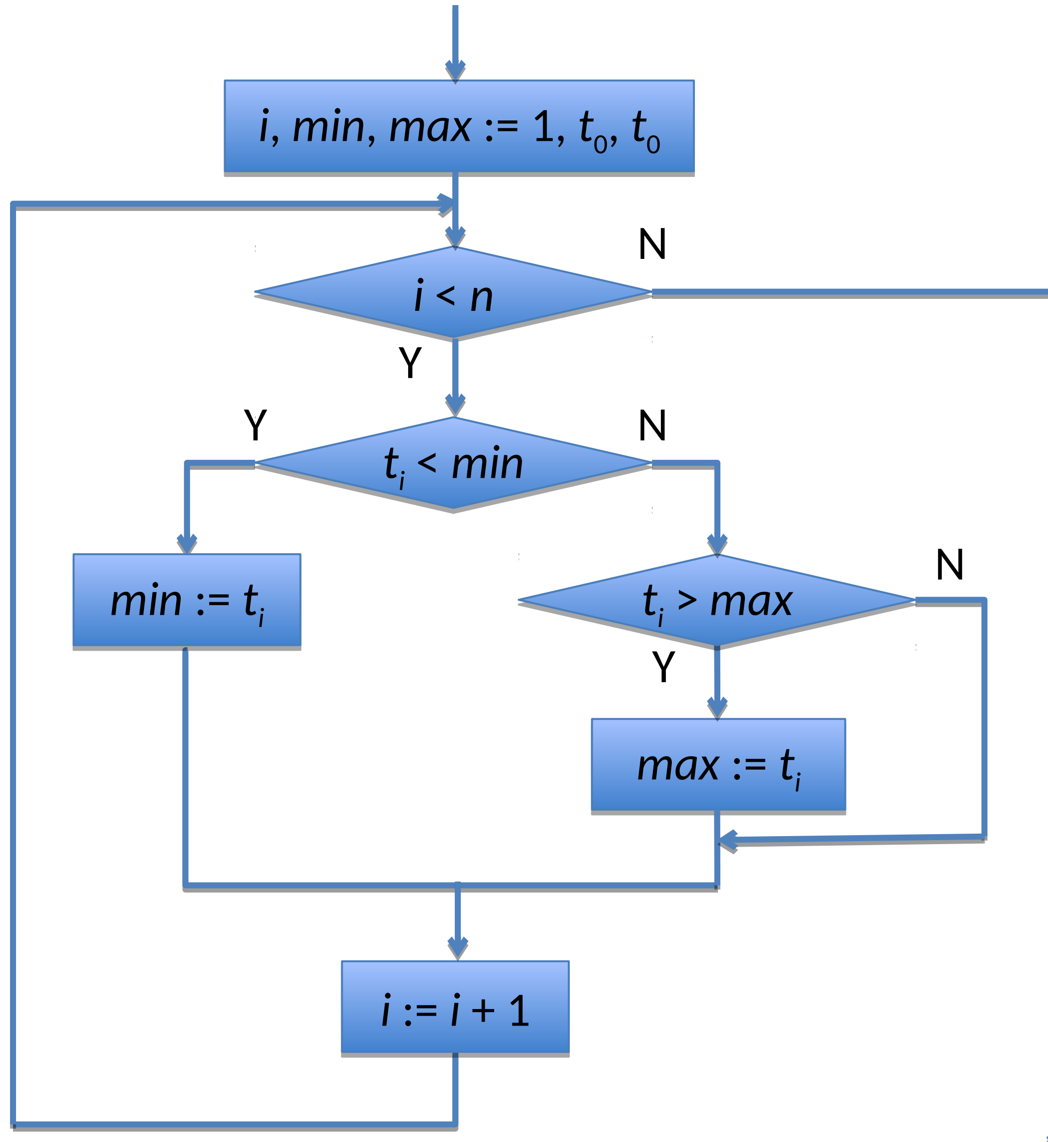
Steps for Min and Max Temperature

How many comparisons with t_i will be made in the best and worst case?

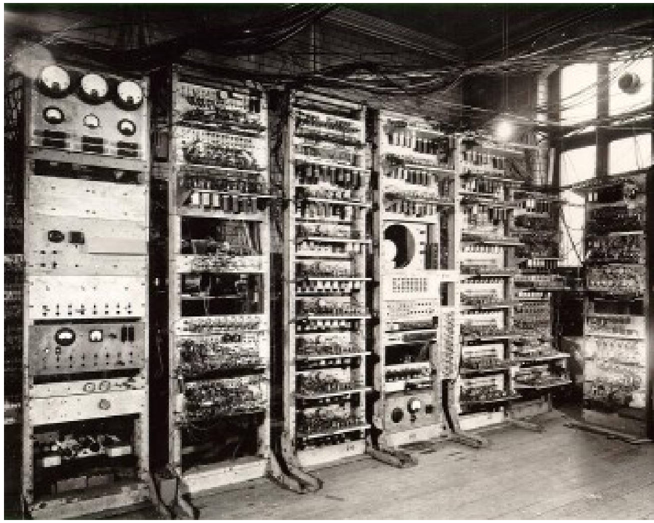
- n – 1 comparisons
  if t is strictly decreasing
- 2 x (n – 1) comparisons if t is increasing

How many assignments to *min* and *max* will be made in the best and worst case?

- 1 to *min*, 1 to *max* if all t_i are the same
- 1 + n in total if t strictly increasing



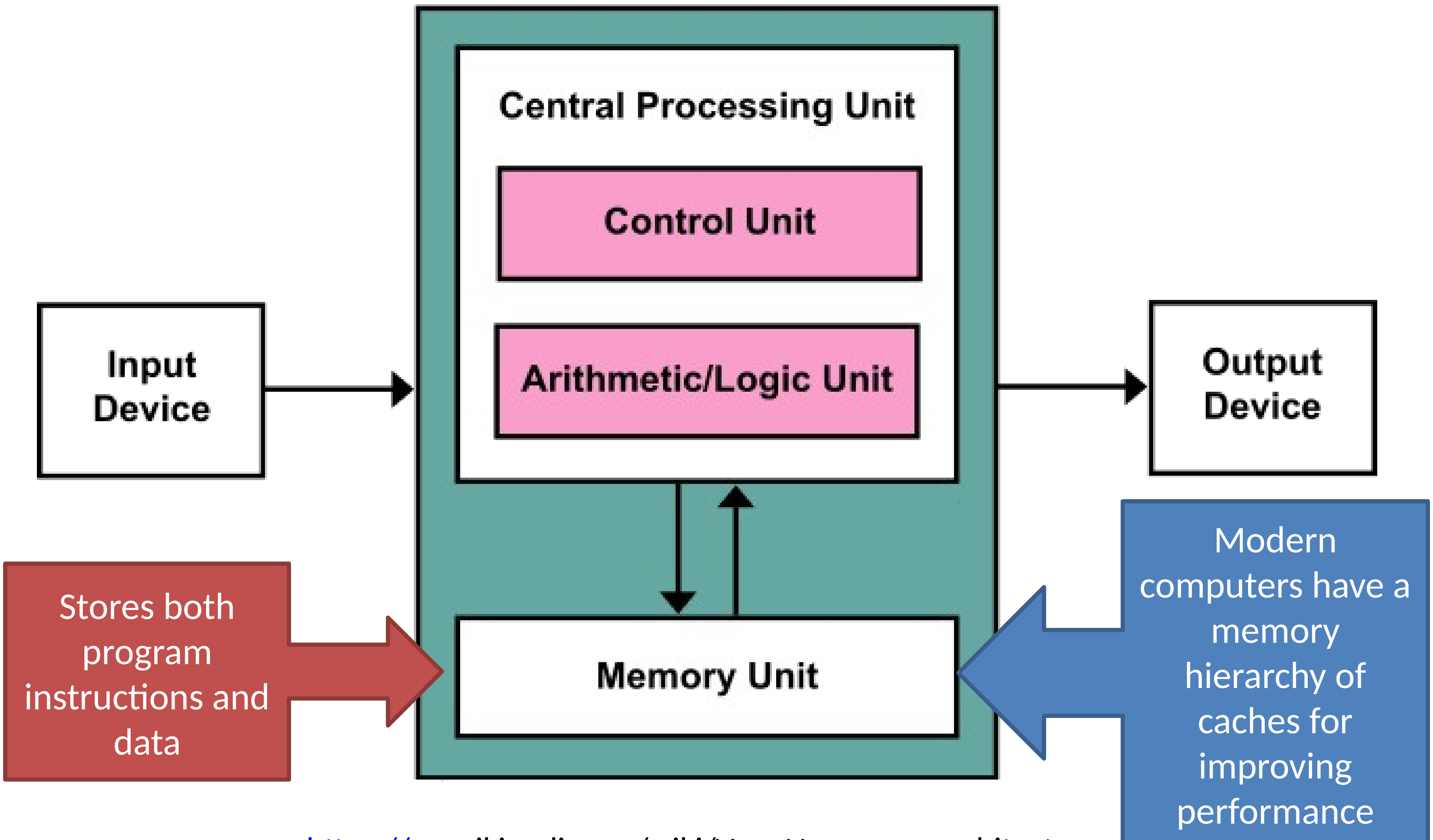
Programs



- **Programs** are algorithms that can be executed automatically by a **computer**
- **Fixed-program computers** serve only one purpose (some early computers, digital clock, washing machine)
- **Stored-program computers** store a program's sequence of instructions and has components that execute any sequence of instructions
  - One of the first modern stored-program computers was the Manchester Mark-1 from 1949
- The program can be changed in a stored-program computer
- It achieved universality by storing instructions the same way as data, allowing programs to be changed – the universality of “machines” was anticipated by Alan Turing in 1936

http://en.wikipedia.org/wiki/Manchester_Mark_1

von Neumann computer architecture



https://en.wikipedia.org/wiki/Von_Neumann_architecture

CPUs and machine language

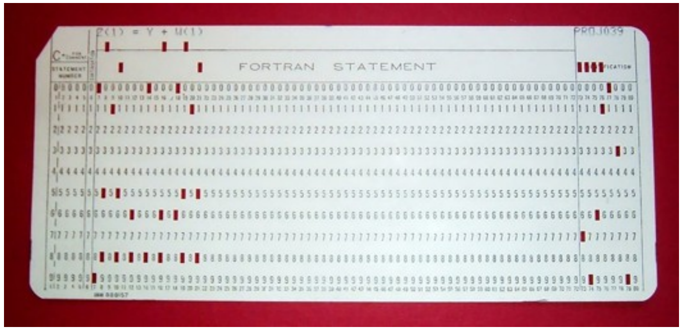
- CPUs operate on bits (0 and 1), grouped into **words**, typically 32 or 64-bit words
- CPUs contain their own very limited memory called **registers**
- Main operations:
  - **Load** a piece of data from memory into a **register**
  - Do an **arithmetic operation** on some data in registers
  - **Store** data from a register into memory
  - **Jump** to another instruction based on the value of data in a register

Programming Languages

- Fortran, an early high-level language from 1958 was a “formula translator” for mathematical expressions into machine language

- Starting with Algol, Pascal in the 60’s and 70’s, programming languages are closer to algorithmic notation.

- While graphical programming languages have dedicated use, textual languages are dominant for large programs.



Fortran “arithmetic if”:
  IF (X) 10, 20, 30
Jump to card 10, 20 , 30
if X < 0, X = 0, X > 0

Compilers

- Compilers translate high-level programs to machine language

- High level language:

  u := u − v

  Machine language:

  load R1, u move u to register 1

  load R2, v move v to register 2

  sub R1, R1, R2 subtract register 2 from 1 and put result in R1

  store u, R1 move register 1 to u

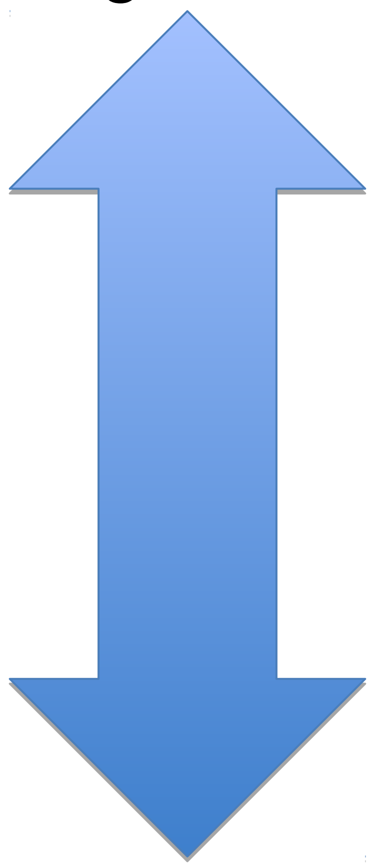
Differences in programming languages

Python
R
Haskell

VisualBasic
Java

C
Assembly

High level



Low level

- Provides programmer lots of abstractions so formulate complex mathematical expressions easily

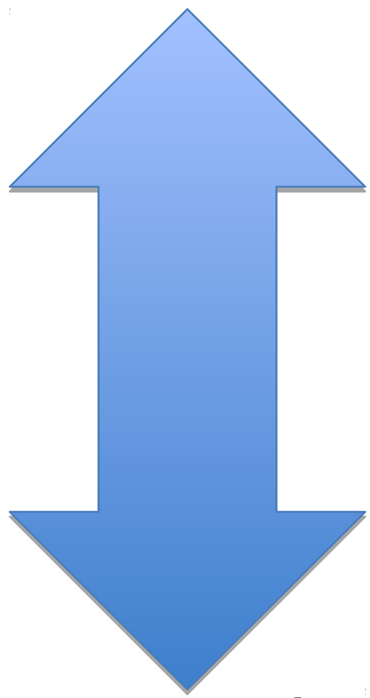
- More direct access to memory and hardware but complex operations require more programmer effort

Differences in programming languages

SQL (databases)
HTML (webpages)
R (statistics)

Application-specific

- Intended for use for a very specific purpose
- Sometimes called DSL (Domain Specific Language)



Python
C
Haskell

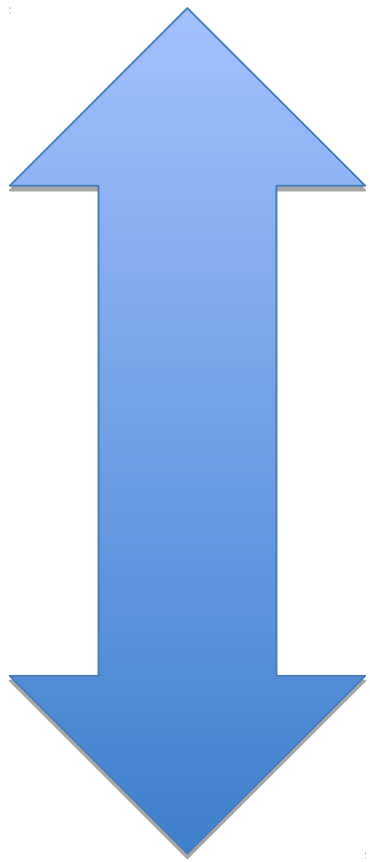
General purpose

- Can be used for many different applications

Differences in programming languages

Python
R

Interpreted



Compiled

Java
C

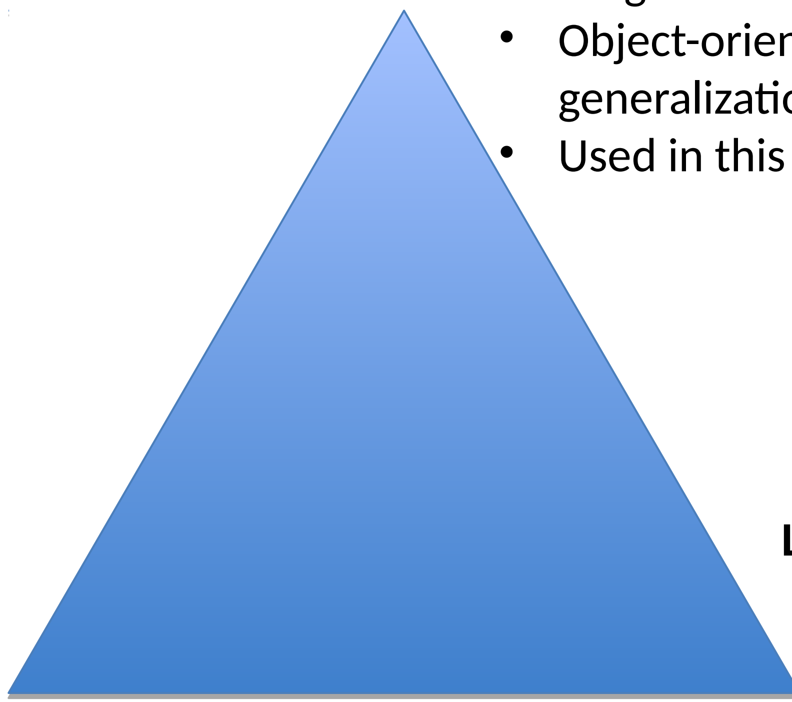
- Program source code is translated into machine language at runtime (sometimes line by line)
  - Easier to debug

- Program source code is translated into machine language in advance
  - Faster to run

Differences in programming languages

Imperative / procedural languages

- Programs are sequences of operations
- Object-oriented languages are a generalization of imperative languages
- Used in this course



Declarative / functional languages

- Programs focus on functions and relationships between functions
- Used in COMP SCI 1JC3

Logic languages

- Programs focus on automated reasoning about logical statements

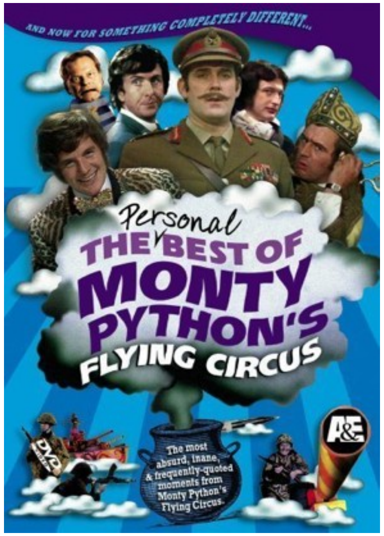
Difference in Programming Languages

- Notation:
  - <> or != for $\neq$
  - begin … end or {…} or indentation for grouping statements
- Safety:
  - type checking (Java: static, Python: dynamic, C: little)
- Support for large programs:
  - classes (Java, Python: yes, C: no)
  - exceptions (Java, Python: yes, C: no)
- Instructions and standard libraries:
  - data types like lists, sets, dictionaries (Python: yes)
  - interaction, graphics (mixed)
- Portability to other platforms

Python

- Began in 1989
- Created by Guido van Rossum (Netherlands)
- Open language

- Python 2
  - Released in October 2000
  - End-of-life 2020
- Python 3
  - Released in December 2008



The language is named after the British comedians Monty Python

Python

- General purpose
- Primarily imperative/procedural language
  - Can be object-oriented or functional, making it “Multi-paradigm"
- High-level
  - Supports many mathematical data types, allowing programs to be abstract and easy to understand
  - Lots of standard libraries
- Interpreted
  - Compiles to an intermediate language and interprets that
  - Makes playing around in the Python environment quick, allowing easy exploration
- Dynamically typed
  - We’ll get to what that means later…
- Compact and intuitive notation
  - Whitespace is important!
- Widely used in industry