

Algorithms, Decisions and Loops

Prof. Dr. Mehmet Göktürk
Gebze Technical University · Fall 2026

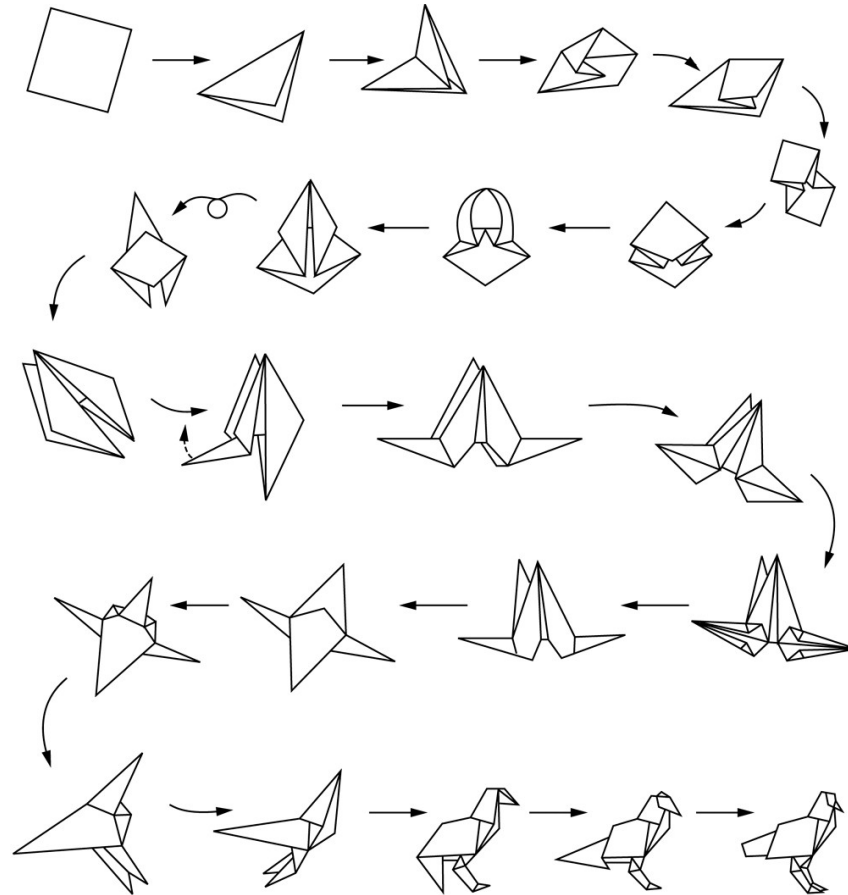
Instructions that choose and repeat

Begin with a problem: what must go in, and what result must come out?

Describe an algorithm with sequence, selection and repetition.

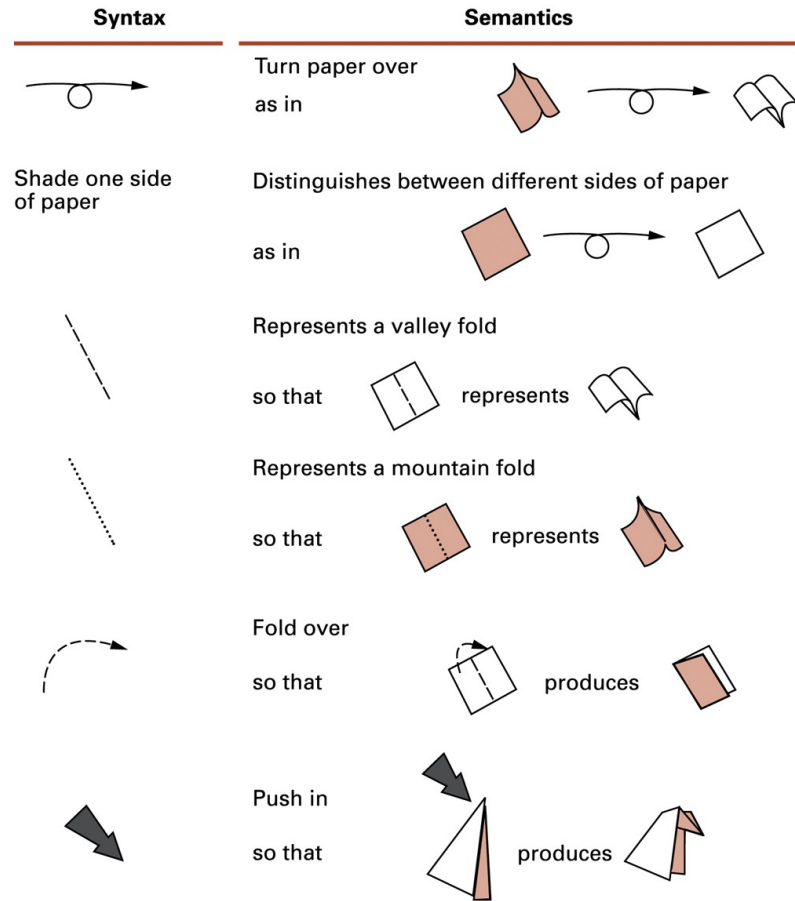
Then connect the flowchart and its trace to a VizMachine program.

An algorithm for folding a bird



The steps have an order. Can each step be understood and carried out?

The meaning of each instruction



Turn the paper over.

Make a valley fold.

Make a mountain fold.

Each symbol has an agreed meaning.

A shared set of basic operations makes the instructions precise.

An algorithm describes precise steps for solving a problem.

A program expresses those steps in instructions a machine can execute.

Sequence, selection and repetition organize those instructions.

For our finite tasks, we must explain how execution reaches the end.

Tea-making: a repeated check

Fill the kettle and start heating.
Check whether the water has boiled.
If it has not, wait briefly and check again.
When it has boiled, continue preparing the tea.



The condition is checked again after waiting. The environment changes the water.

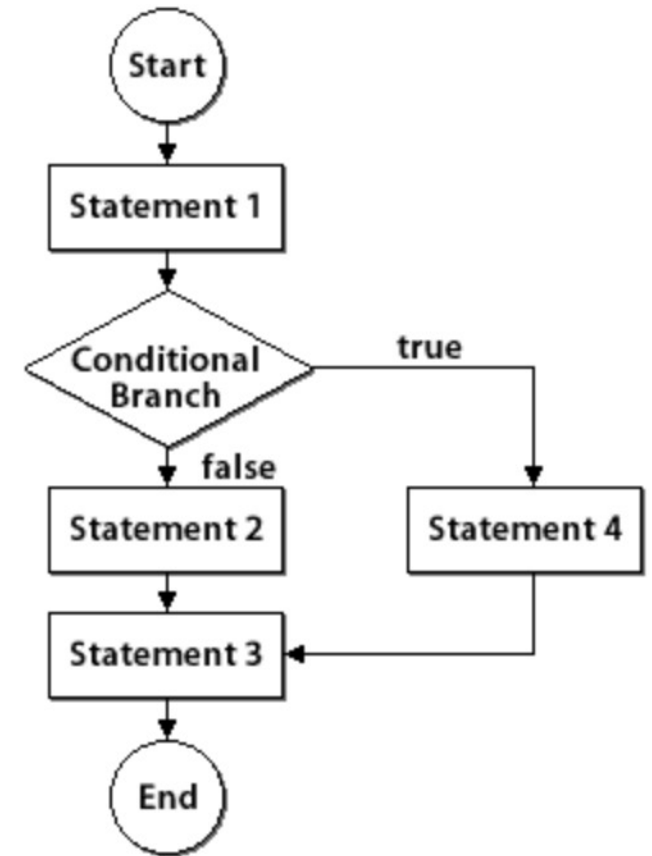
Three control-flow patterns

Sequence: perform the next step in order.

Selection: choose a path according to a condition.

Repetition: return to an earlier step while more work is needed.

The program counter tells the CPU where to fetch next.



Define the problem before the steps

Task: prepare and serve the requested number of cups.

Input: a nonnegative whole number n and each person's sugar preference.

Result: exactly n cups served, each matching its request.

Assume cups, tea and sugar are available. The procedure must finish.

Precise instructions

Too vague	A more precise instruction
Make enough tea	Prepare exactly n cups
Add some sugar	Add one spoon if sugar is requested
Repeat a few times	Repeat until served equals n
Keep track of progress	Increase served by 1 after each completed cup

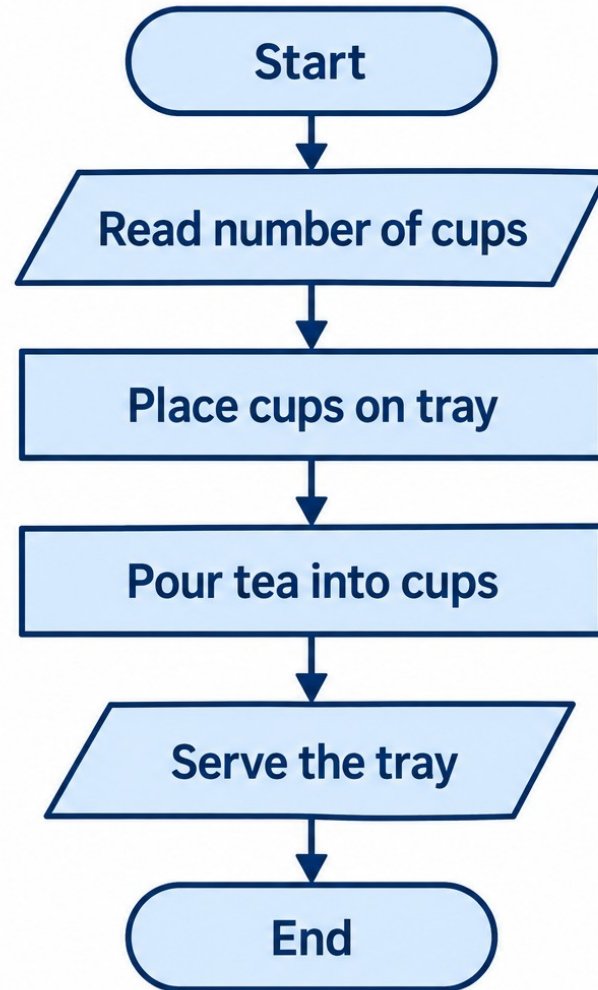
Different people should be able to follow the steps and agree on the result.

Reading a flowchart

Visual element	How to read it
Rounded terminal	Where the procedure begins or ends
Rectangle	Carry out an action
Diamond	Evaluate a condition and follow Yes or No
Arrow	Continue in the direction of its arrowhead
Parallelogram	Receive input or produce output

Follow one path at a time. Read each condition before choosing its outgoing arrow.

Sequence: the order matters



Each action follows the previous action.

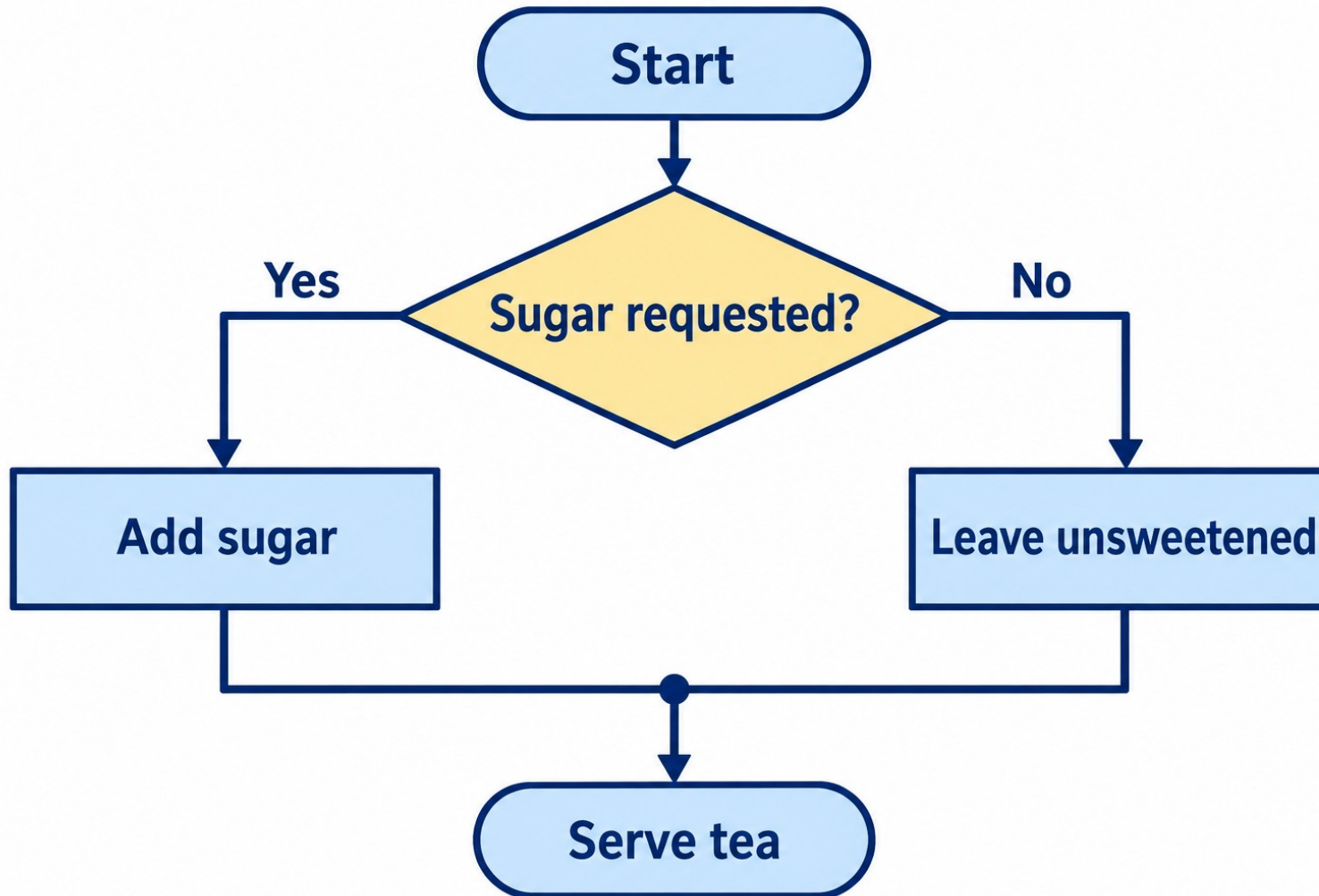
Place the cups before pouring into them.

A box can summarize a task we explain in more detail later.



This overview treats filling the cups as one action; the repetition inside it comes next.

Selection: choose one path



Evaluate the request once for this cup.

Yes: add sugar. No: leave it unsweetened.

Both paths rejoin before the tea is served.

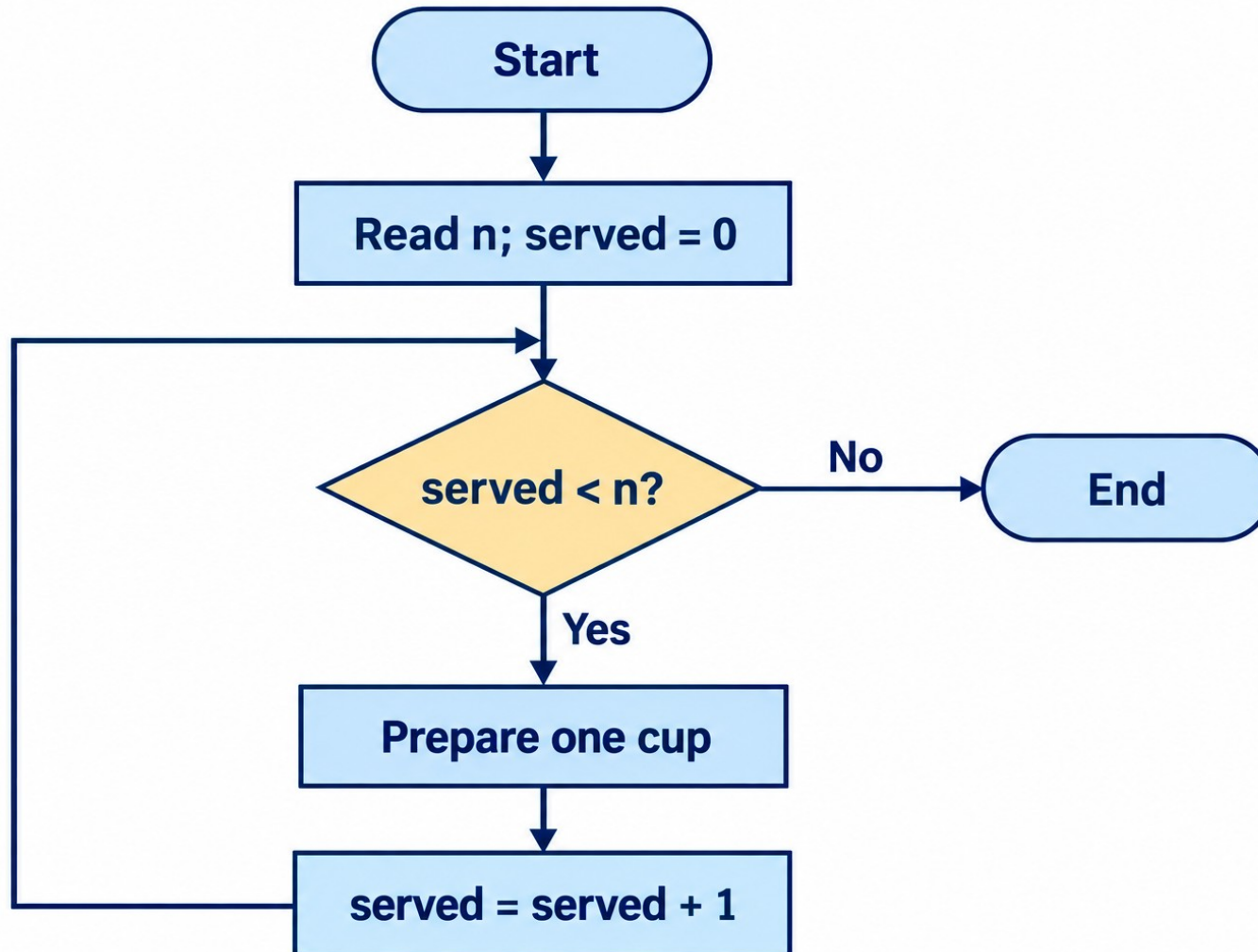
For this example, adding sugar means adding one spoon. Only one alternative executes.

Trace both sides of a decision

Request	Selected action	Shared continuation
Sugar: yes	Add one spoon	Serve tea
Sugar: no	Leave unsweetened	Serve tea

The decision chooses an action; it does not repeat it. The other path is skipped.

Repetition: return to the test



Initialize served once, before the loop.

If $\text{served} < n$, prepare one cup and increase served.

Return to the condition. If it is false, finish.

Here “prepare one cup” includes handing it to the customer. n is a nonnegative whole number.

Trace the loop with $n = 3$

served before test	served < 3?	Action	served afterward
0	Yes	Prepare cup 1	1
1	Yes	Prepare cup 2	2
2	Yes	Prepare cup 3	3
3	No	Finish	3

The condition is tested four times; the body executes three times. These are decimal counts.

Zero repetitions and termination

If $n = 0$, the first test is false. No cup is prepared.

For $n > 0$, each completed cup increases served by 1.

The remaining work, $n - \text{served}$, decreases until it reaches zero.

Without the update, the condition can remain true forever.

A choice inside a repetition

Repeat “prepare one cup” for each customer.

Inside that task, check this customer’s sugar preference.

Select one alternative, serve the cup, then count it as completed.

The loop repeats the whole task. The decision controls part of each repetition.

Check the combined algorithm

n = 3; preferences: yes, no, yes	Answer
How many cups are served?	3
How many times is sugar added?	2
How many times is served < n tested?	4
Does the sugar decision repeat on its own?	No; the surrounding loop repeats it

Change the preferences to no, no, no: three cups still get served; sugar is never added.

One algorithm, several representations

A flowchart shows the paths and the return to a condition.

Pseudocode describes those same actions and conditions in words.

A program expresses them using a language's exact rules.

Next we will use the same loop structure to change a display character.

Recall: a value, an address, an instruction

Instruction	Assembly	Effect
2130	LOAD R1, #30	R1 gets literal 30
1130	LOAD R1, \$30	R1 gets the contents of memory cell 30
31E0	STORE R1, \$E0	Memory cell E0 gets R1
2201	LOAD R2, #01	R2 gets literal 01
5112	ADD R1, R1, R2	R1 gets $R1 + R2$

#30 means the value 30; \$30 means memory address 30. All values and addresses are hexadecimal.

Recall: instruction addresses

Address	Instruction	Assembly	Next PC after execution
A0	2130	LOAD R1, #30	A2
A2	31E0	STORE R1, \$E0	A4
A4	C000	HALT	A6, then halted

Each instruction occupies two bytes. A0 and A1 hold the first instruction.

Execution state in VizMachine

VizMachine exposes one small teaching machine's state.

We can inspect registers, memory and the program counter directly.

It runs one program; operating-system scheduling comes in Week 4.

We will trace a single program and observe its output.

Instructions for the exercise

Encoding	Assembly	Meaning
2RXY	LOAD R, #XY	Load literal XY into R
1RXY	LOAD R, \$XY	Load memory[XY] into R
3RXY	STORE R, \$XY	Store R into memory[XY]
5RST	ADD R, S, T	Add registers S and T into R
C000	HALT	Stop execution

Each instruction occupies two bytes. All machine values here are hexadecimal.

The display reads memory

Open VizMachine on the course website; select the Screen tab.

First row: memory addresses E0–EF. Second row: F0–FF.

The display interprets each stored byte as a character code.

Writing a byte to this region changes the displayed character.

www.mehmetgokturk.com/cse101/vizmachine/

Bytes and characters

Character	Decimal code	Hexadecimal byte
H	72	48
I	73	49
O	79	4F
K	75	4B
Space	32	20

The same byte can be viewed as a number or interpreted as a character.

Exercise: display HI

Goal: write H to E0 and I to E1, then halt.

Place the program at A0. Keep its code away from the display region.

Reset memory/registers; set PC = A0 before execution.

Predict when each character appears, then single-step the program.

Display HI: the complete program

Address	Instruction	Assembly	Effect
A0	2148	LOAD R1, #48	R1 gets 48 (H)
A2	31E0	STORE R1, \$E0	M[E0] gets R1
A4	2149	LOAD R1, #49	R1 gets 49 (I)
A6	31E1	STORE R1, \$E1	M[E1] gets R1
A8	C000	HALT	Halt

Enter consecutive bytes: 21 48 31 E0 21 49 31 E1 C0 00. Set PC to A0.

Display H: execution trace

After	Next PC	R1	M[E0]	M[E1]
2148	A2	48	00	00
31E0	A4	48	48	00
2149	A6	49	48	00
31E1	A8	49	48	49
C000	AA	49	48	49

H appears after the first STORE; I after the second. This core advances PC before HALT.

Why 48 and not H?

Memory stores a byte, not the idea of a letter.

The display interprets hexadecimal 48 as the character H.

Loading R1 alone does not change the display; STORE changes memory.

The CPU writes data. The display determines how that data is shown.

Modify the program

Change HI to OK without changing the STORE addresses.

Move the message to the second display row.

Predict the bytes and addresses before running each version.

Check both the memory cells and the displayed characters.

Modification answers

Goal	Instruction sequence	Assembly
OK on first row	214F 31E0 214B 31E1 C000	LOAD R1, #4F; STORE R1, \$E0 LOAD R1, #4B; STORE R1, \$E1; HALT
HI on second row	2148 31F0 2149 31F1 C000	LOAD R1, #48; STORE R1, \$F0 LOAD R1, #49; STORE R1, \$F1; HALT

Character changes affect literal LOADs. Position changes affect STORE destinations.

A useful debugging check

Suppose 2148 is accidentally entered as 1148.

2148 uses the literal byte 48; 1148 reads memory cell 48.

After a memory reset, $M[48]$ is 00, so this version will not display H.

Check the opcode, then the source value, then the destination address.

Optional: calculate the next character

Load R1 = 48 and R2 = 01. ADD them into R3.

Store R3 at E0. Predict the displayed character.

```
LOAD R1, #48; LOAD R2, #01; ADD R3, R1, R2  
STORE R3, $E0; HALT (at A0)
```

Answer: $48 + 01 = 49$, so the display shows I.

A branch changes the next address

Normally, execution continues at the next two-byte instruction.

A taken branch puts its target address into the program counter.

An untaken branch leaves the sequential next address in place.

The target must be the start of the intended instruction.

VizMachine conditional jump

Encoding	Assembly	Meaning
BRXY	JUMP R, \$XY	If register R equals R0, set PC to XY
B1AE	JUMP R1, \$AE	If R1 equals R0, continue at AE
B0A6	JUMP R0, \$A6	R0 equals itself, so always continue at A6

B is the opcode. R is a register number. XY is a memory address, not a count.

Taken and untaken branches

Before B1AE at A6	Assembly	Condition	Next PC
R0 = 33, R1 = 33	JUMP R1, \$AE	Equal	AE
R0 = 33, R1 = 31	JUMP R1, \$AE	Not equal	A8

All values are hexadecimal. The comparison is with R0; R0 need not contain zero.

Check: choose the next instruction

At A6, the instruction is B1AE. R0 = 30 and R1 = 30.

What is the next PC? What changes if R1 = 31?

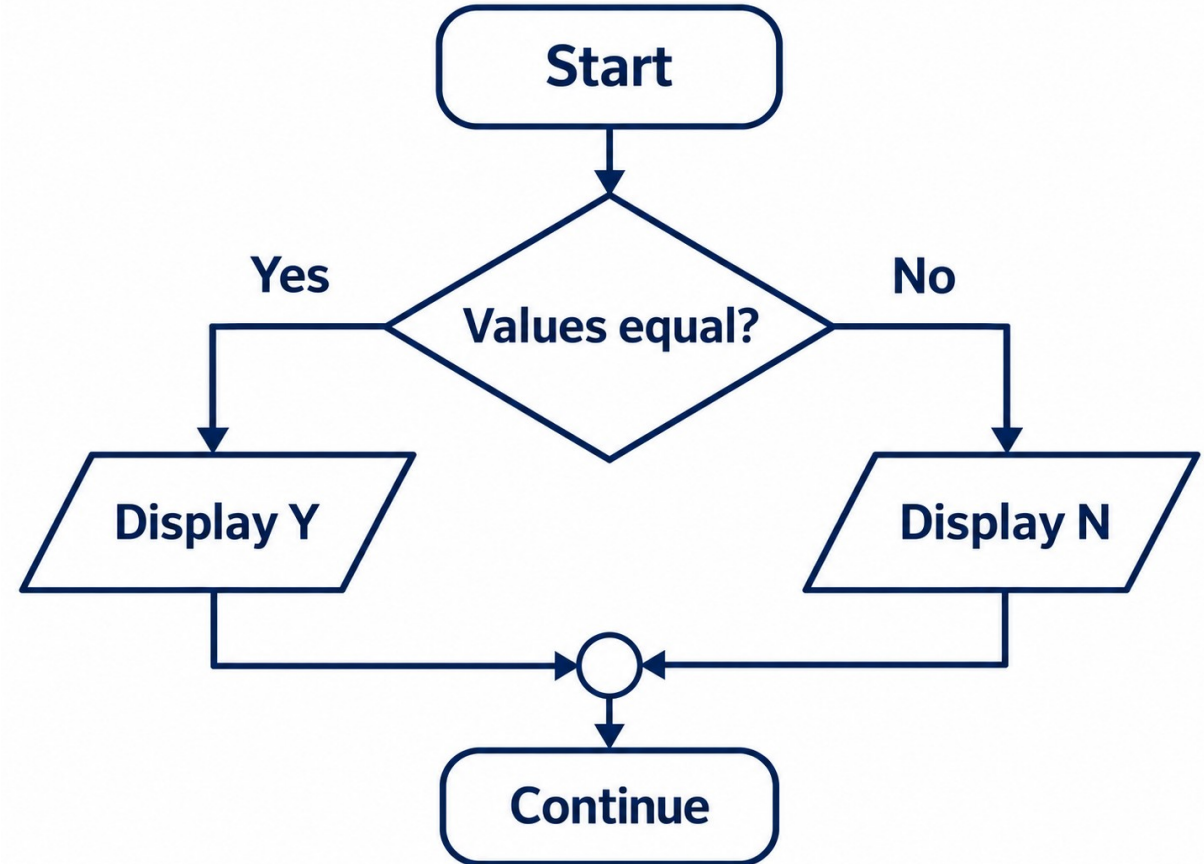
What would B0A6 do, regardless of R0's value?

Answers: AE; A8; always branch to A6.

A decision with two possible outcomes

If the stored values are equal:
display Y
Otherwise:
display N
Continue after the choice.

Only one output executes.
Both paths rejoin before continuing.



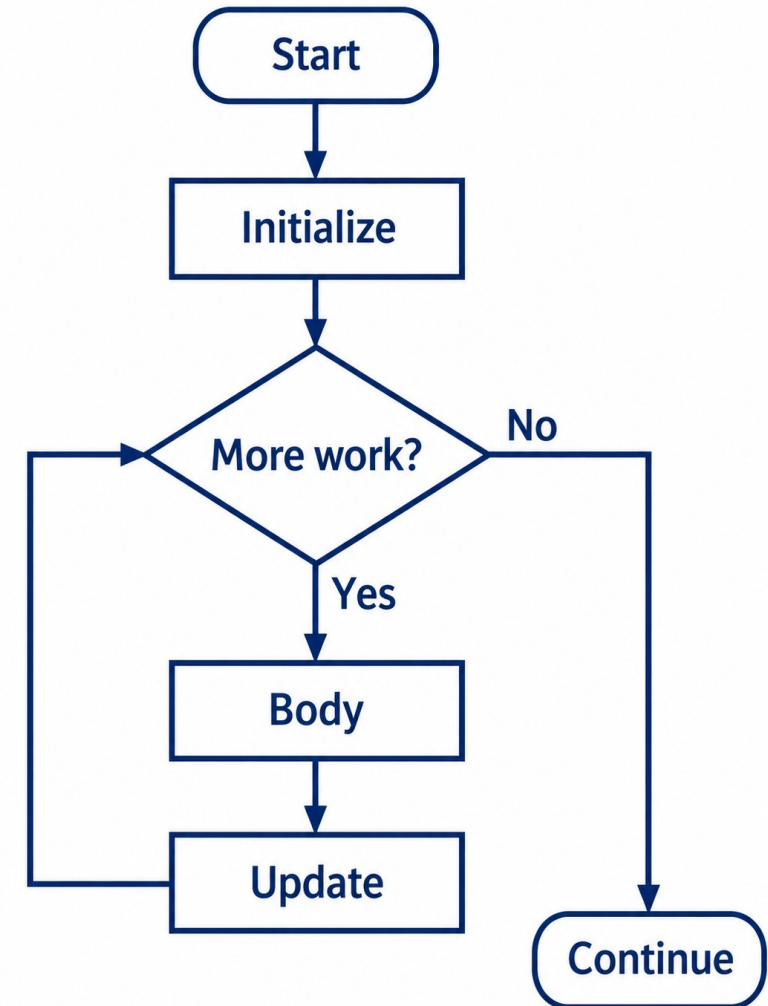
The parts of a loop

Initialize: set the first value and stopping value.

Test: decide whether work remains.

Body: perform one repetition.

Update: change the state and return to the test.



Pseudocode: count on the display

```
character = code for 0
stop = code immediately after 9
while character is not equal to stop:
    write character to the first display cell
    character = character + 1
Halt.
```

Testing before the body allows zero repetitions.

Character codes for the counter

Character	Hexadecimal byte
0	30
1	31
2	32
9	39
:	3A

3A is the stopping value. The program checks it before STORE, so : is not displayed.

Exercise: a changing display cell

Write 0, then 1, up to 9 into memory cell E0, then halt.

Each new character replaces the previous one in the same cell.

Use single-step execution to see the changes. A full run may appear instant.

Keep the program at A0–AF, away from display memory E0–FF.

Registers used by the loop

Register / cell	Purpose	Initial byte
R0	Stop before writing this character	3A
R1	Current character	30
R2	Increment	01
M[E0]	First display cell	00 after reset

R0 is the comparison register. R1 changes each time around the loop.

The complete counter program

Address	Word	Assembly	Effect
A0	203A	LOAD R0, #3A	R0 = 3A
A2	2130	LOAD R1, #30	R1 = 30
A4	2201	LOAD R2, #01	R2 = 01
A6	B1AE	JUMP R1, \$AE	If R1 = R0, branch to AE
A8	31E0	STORE R1, \$E0	Display R1 at E0
AA	5112	ADD R1, R1, R2	R1 = R1 + R2
AC	B0A6	JUMP R0, \$A6	Return to the test
AE	C000	HALT	Halt

Readable notation: # = immediate value; \$ = memory address. Menu input: LOAD2 + operand 03A.

Entering and running the program

Reset memory and registers before entering the program.

Enter the eight words consecutively from A0. Set PC = A0.

Single-step and inspect PC, R1 and E0. Use the Screen tab for output.

Do not reset memory after entering the code. Resetting would erase it.

Program bytes and memory locations

A0: 20 3A 21 30 22 01 B1 AE

A8: 31 E0 51 12 B0 A6 C0 00



The address labels are not input bytes. Each row here contains eight bytes.

The first repetition

Execute at	Assembly	What happens	Next PC
A6	JUMP R1, \$AE	30 differs from 3A; continue	A8
A8	STORE R1, \$E0	E0 gets 30; display shows 0	AA
AA	ADD R1, R1, R2	R1 becomes 31	AC
AC	JUMP R0, \$A6	Always branch back	A6

The display still shows 0 after ADD. The next STORE makes 1 visible.

The last repetition and exit

Execute at	Assembly	State change or decision	Next PC
A6	JUMP R1, \$AE	R1 = 39 differs from 3A	A8
A8	STORE R1, \$E0	E0 gets 39; display shows 9	AA
AA	ADD R1, R1, R2	R1 becomes 3A	AC
AC	JUMP R0, \$A6	Return to the test	A6
A6	JUMP R1, \$AE	R1 equals R0; branch taken	AE
AE	HALT	Halt	B0

The final screen shows 9. R1 contains 3A. The core advances PC before HALT.

Why does this loop terminate?

R1 starts at 30 and increases by 01 after each STORE.

The test is reached again after every update.

After writing 39, the next update produces 3A, equal to R0.

The branch then skips STORE and reaches HALT.

Predict before running

Change to the original	Assembly	Prediction to make
203A becomes 2033	LOAD R0, #3A LOAD R0, #33	Which characters will appear?
203A becomes 2030	LOAD R0, #3A LOAD R0, #30	How many STOREs execute?
2130 becomes 2137	LOAD R1, #30 LOAD R1, #37	What are the first and last characters?
31E0 becomes 31F0	STORE R1, \$E0 STORE R1, \$F0	Where will output appear?

Each Assembly cell lists the original, then its replacement. Change one instruction at a time.

Modification answers

Change	Observed sequence / location
R0 = 33	0, 1, 2; three STOREs
R0 = 30	No STORE; zero repetitions
R1 starts at 37	7, 8, 9; three STOREs
STORE at F0	First cell of the second row

For the zero-repetition case, reset display memory first to avoid stale output.

Debugging: a loop that makes no progress

Change the increment at A4 from 2201 to 2200.

R1 remains 30 after every ADD, so it never equals 3A.

The same 0 is written repeatedly. The loop does not reach HALT.

Stop execution, inspect R2 and restore 2201.

Debugging: returning to the wrong place

Change the branch at AC from B0A6 to B0A0.

The branch now repeats initialization as well as the loop body.

R1 is set back to 30 each time, so the screen repeatedly receives 0.

The return address should be the test, after initialization.

Equality needs a reachable stopping value

Our loop advances by 01 and reaches its chosen stopping byte.

Changing the increment can skip the stopping value.

Starting beyond the stop may cause a long path through byte wraparound.

Do not treat “not equal” as interchangeable with “less than.”

Tests that explain the behavior

Case	Expected result
Original program	Ten STOREs; last character 9
Stop byte equals starting byte	Zero STOREs
Start = 39, stop = 3A	One STORE; character 9
Increment = 00	Repeated 0; no HALT

A result is stronger evidence when it agrees with a predicted state trace.

The same repetition in C

```
int digit = 0;
while (digit < 10) {
    printf("%d", digit);
    digit = digit + 1;
}
```

This preview prints 0123456789 in sequence. printf does not overwrite VizMachine E0.

A complete C preview

```
#include <stdio.h>

int main(void) {
    int digit = 0;
    while (digit < 10) {
        printf("%d", digit);
        digit = digit + 1;
    }
    printf("\n");
    return 0;
}
```

Compile: `cc counter.c -o counter` Run: `./counter`

Machine program and C program

Idea	VizMachine example	C preview
Working value	Register R1	Variable digit
Update	ADD instruction	digit = digit + 1
Condition	Compare R1 with R0	digit < 10
Repetition	Jump back to A6	while statement

A compiler translates C into machine instructions for its target CPU, not necessarily VizMachine.

Flowcharts support algorithm development

Problem solving: separate the inputs, required result and smaller tasks.

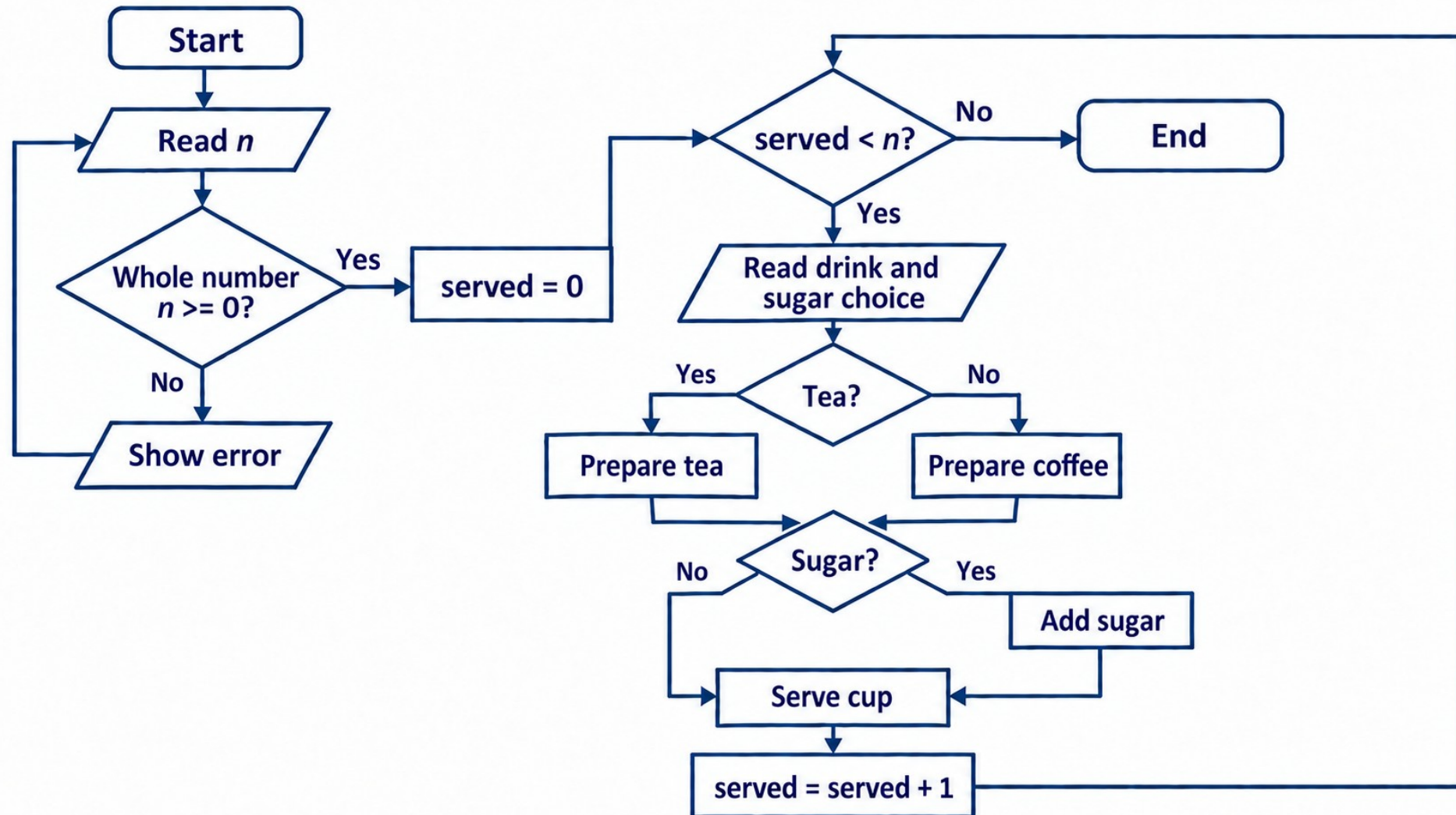
Mental tracing: follow one path and track how the state changes.

Precise description: show both decision outcomes and where paths rejoin.

Logic checks: find missing cases, updates and exits before writing code.

A clear flowchart helps reasoning. Correctness still needs traces, tests and justified assumptions.

A complete order: choices inside a loop



Assume each drink choice is tea or coffee, and supplies are available. Trace $n = 0$, then $n = 2$.

Practice and next steps

Trace one taken and one untaken branch, including the next PC.

Modify a working loop and explain the new output before running it.

Test the initial condition and the final repetition, not just the middle.

Next: operating systems in Week 4; C variables and expressions in Week 5.

Algorithm efficiency: the amount of work

Task: find the sum of the whole numbers from 1 to n.

Repeated addition

$$1 + 2 + 3 + \dots + 10 = 55$$

Start with 1. Add each remaining number.

A formula

$$10 \times (10 + 1) \div 2 = 55$$

In general: $n \times (n + 1) \div 2$

Input size	Repeated addition	Formula
n = 10	9 additions	3 arithmetic operations
n = 1,000	999 additions	3 arithmetic operations

Both are correct. How does the work grow when the input grows?

We count arithmetic operations here. Assume the numbers fit in the chosen numeric type.