Visualizing Processes and Logic

Unit 3: Flowcharts, Decision & Truth Tables, and Execution Tracing

Dr. Gopal Chandra Jana

Dept. of CSE, Sharda University
Fundamentals of Computing and AI (CSAI1101) — Unit 3

September 2, 2026

Course Page:
https://www.gcjana.in/courses/shardauniversity/2601/fundamentals-of-computing-and-artificial-intelligence/

Outline

1. Flowchart Fundamentals
2. Decision Tables & Truth Tables
3. Execution Tracing & Testing
4. Summary & Practice

Learning Outcomes of Unit 3

After this unit, a student will be able to...

1. **Identify** the standard flowchart symbols and their meanings.
2. **Draw** flowcharts using sequence, branching (selection) and loops (iteration) with input–output.
3. **Build** decision tables to capture complex "if–then" rules neatly.
4. **Construct** truth tables for logical conditions using AND, OR and NOT.
5. **Trace** a program by hand using dry runs, state tables and variable tracking, and design good test cases.

Mapping to the Syllabus

A. Flowcharts: symbols, control structures, branching, loops, I/O. **B.** Decision tables and truth tables. **C.** Execution tracing: dry runs, state tables, variable tracking, test cases.

From Unit 2 to Unit 3

Where we are

In **Unit 2** we learnt to *understand* a problem and write an *algorithm* in pseudocode. Now we make that logic **visible** and **testable**.

- **Flowcharts** — draw the logic as a picture.
- **Decision & truth tables** — organise complex conditions.
- **Dry runs** — check the logic by hand *before* coding.

These skills catch mistakes early — when they are cheap to fix.

The theme of this unit

See the process, test the logic, trust the result.

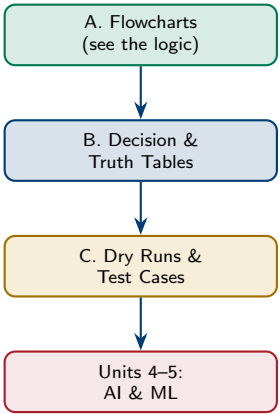


Figure: The three parts of Unit 3.

What is a Flowchart?

A picture of an algorithm

A **flowchart** is a diagram that shows the steps of an algorithm using **standard symbols** joined by arrows (**flowlines**).

- Each symbol shape means a specific kind of step.
- Arrows show the *order* in which steps run.
- Easy to read — you *see* the logic at a glance.

Why draw flowcharts?

- Plan logic *before* coding.
- Spot mistakes and missing steps early.
- Communicate the design to teammates.

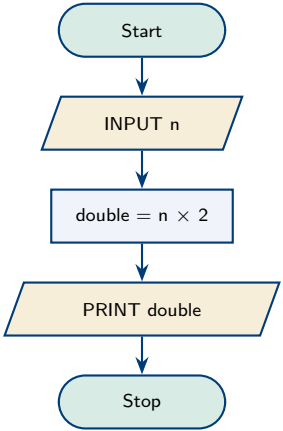


Figure: A flowchart to double a number.

A Short History (Standard Symbols)

Why standard symbols matter

If everyone uses the *same* shapes, any reader anywhere can understand a flowchart — just like traffic signs.

- **ANSI** set the first standard (X3.5) in the 1960s.
- **ISO 5807** made it an international standard (1985).
- The core five symbols are recognised worldwide.

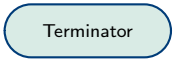
Golden convention

Flowcharts normally flow **top-to-bottom** and **left-to-right**, with one clear *start* and one clear *end*.

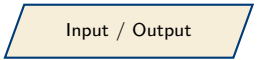
Rules of thumb

- Every flowchart has exactly one **Start** and at least one **Stop**.
- Arrows never cross if it can be avoided.
- A *decision* has **two** exits (Yes / No).
- Keep text inside a symbol short.

The Standard Flowchart Symbols



Oval — Start / Stop of the flowchart.



Parallelogram — read input or print output.

Rectangle — a calculation or action.

Diamond — a Yes/No question; two exits.

Connector — joins parts of the chart.

Flowline — shows direction.

Symbols at a Glance — Reference Table

Symbol	Shape	Meaning / Use
Terminator	Oval	Marks Start and Stop
Input / Output	Parallelogram	Read data in / print results out
Process	Rectangle	A calculation or action step
Decision	Diamond	A Yes/No (True/False) question
Connector	Small circle	Joins flow across breaks/pages
Flowline	Arrow	Direction of flow between steps

Memory aid: *oval* starts, *parallelogram* talks (I/O), *rectangle* works, *diamond* asks, *arrow* walks.

Control Structure 1 — Sequence

One step after another

In a **sequence**, steps run **top to bottom**, each exactly once, in order. No skipping, no repeating.

Example: area of a rectangle

1. Read length l and breadth b.
2. Compute area $= l \times b$.
3. Print the area.

Each step depends on the one before it.

Key point

Sequence is the simplest structure — and most of a program is just sequence.

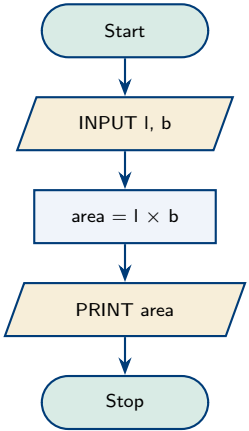


Figure: A pure sequence flowchart.

Control Structure 2 — Selection (Branching)

Choose a path with a decision

Selection (branching) uses a **diamond** to ask a Yes/No question and pick one of two paths.

Example: pass or fail

"If marks ≥ 40 then *Pass*, else *Fail*."

- Condition true $\rightarrow$ one branch.
- Condition false $\rightarrow$ the other branch.
- Both branches *re-join* before the end.

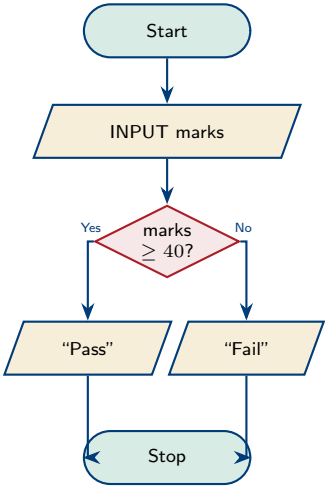


Figure: A branching (if–else) flowchart.

Selection Variants — if, if–else, nested

Three common forms

- **Simple if**: do something only if the condition is true.
- **if–else**: do one thing if true, another if false.
- **Nested if / else-if**: a decision inside a decision (many cases, like grades).

Grade example (else-if ladder)

```
IF m≥90 THEN A
ELSE IF m≥80 THEN B
ELSE IF m≥70 THEN C
ELSE F
```

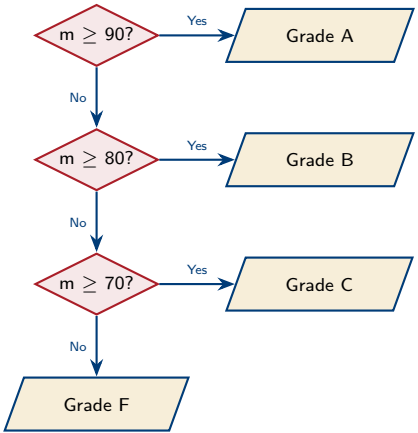


Figure: A nested (else-if) decision ladder.

Control Structure 3 — Iteration (Loops)

Repeat steps without rewriting them

Iteration (a loop) repeats a block of steps while a condition stays true. Two common kinds:

- **while / for loop**: check the condition **first** (may run zero times).
- **do–while loop**: run once, then check the condition **after** (runs at least once).

Beware the infinite loop!

The loop variable *must change* so the condition eventually becomes false — otherwise the loop never stops.

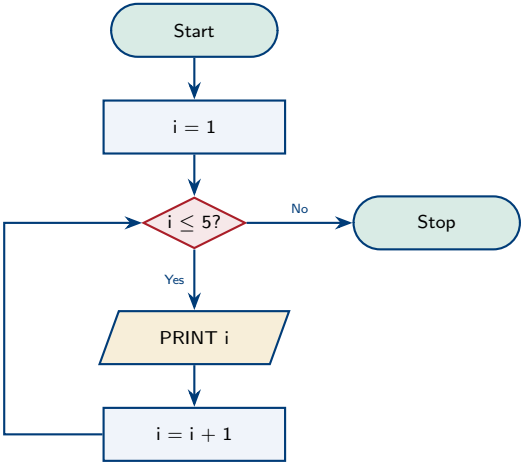


Figure: A loop that prints 1 to 5.

Worked Example — Largest of Two Numbers

The problem

Read two numbers a and b; print the larger one.

The plan

1. Read a and b.
2. Ask: is $a > b$?
3. Yes $\rightarrow$ print a.
4. No $\rightarrow$ print b.

Check the edge case

If $a = b$, the "No" branch prints b — still correct, since both are equal.

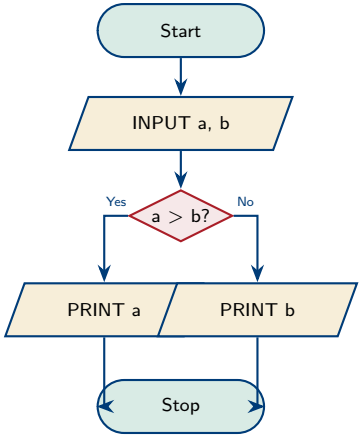


Figure: Flowchart for the larger of two numbers.

Worked Example — Sum of 1 to N (Loop)

The problem

Read n; compute $1 + 2 + \cdots + n$.

The plan

1. Read n; set $sum = 0$, $i = 1$.
2. While $i \leq n$: add i to sum, then $i = i + 1$.
3. When $i > n$, print sum.

Loop essentials

A loop needs three things: **initialise** ($i = 1$), **test** ($i \leq n$), **update** ($i = i + 1$).

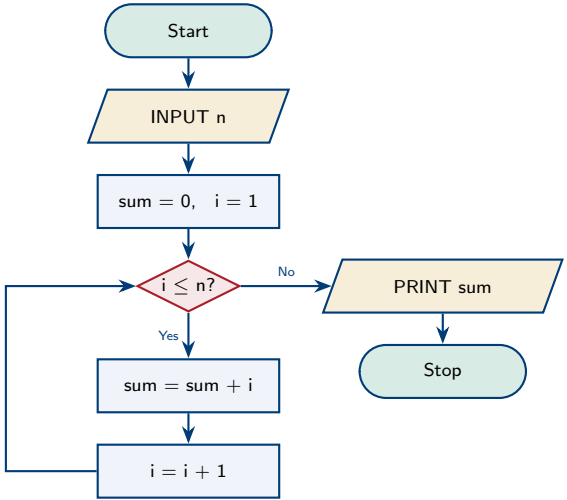


Figure: Flowchart for the sum 1 to n.

Flowchart — Advantages & Common Mistakes

Advantages

- **Clear**: logic is easy to see and follow.
- **Communication**: a shared picture for the whole team.
- **Debugging**: errors in logic show up early.
- **Documentation**: a lasting record of the design.

Limitations

Large programs make huge, messy charts; small changes may need a full redraw.

Common mistakes to avoid

- No *Start* or no *Stop*.
- A decision with only *one* exit (must have two).
- Arrows with no clear direction.
- A loop with no *update* step $\Rightarrow$ infinite loop.
- Using the wrong symbol (e.g. rectangle for a decision).

Quick Check — Round 1 (Flowcharts)

[Q1] Which symbol is used for (a) Start/Stop, (b) a decision, (c) input/output?

[Q2] How many exits must a decision (diamond) symbol have?

[Q3] Name the three control structures used in flowcharts.

[Q4] What happens if a loop has no "update" step for its loop variable?

Think for 60 seconds — answers on the next slide.

Answers — Round 1

[A1]

(a) **Oval** (terminator), (b) **Diamond** (decision), (c) **Parallelogram** (input/output).

[A2]

Two exits — one for *Yes/True* and one for *No/False*.

[A3]

Sequence, **Selection** (branching), **Iteration** (loops).

[A4]

The condition never becomes false, so the loop runs forever — an **infinite loop**. The program never stops.

Why Tables for Logic?

When "if" statements pile up

Some problems have **many conditions** combining in many ways. A tangle of nested "if" statements becomes hard to read and easy to get wrong.

- A **decision table** lists every combination of conditions and the action for each — neatly, in rows and columns.
- A **truth table** shows the result of a logical expression for *every* true/false combination.

Big benefit

Tables make sure you have **not missed** any case — every combination gets a row.

The danger of missed cases

"Give a discount if the customer is a member *and* the bill is over Rs. 1000." What if they are a member but the bill is Rs. 900? A table forces you to decide *every* case.

Truth Values and Logical Operators

The basics of logic

A **condition** is either **True (T / 1)** or **False (F / 0)**.
We combine conditions with three operators:

- **AND**: true only if *both* are true.
- **OR**: true if *at least one* is true.
- **NOT**: reverses — true becomes false.

In plain words

"Wear a raincoat **AND** carry an umbrella" needs both.
"Take a bus **OR** a train" needs just one.

A	B	A AND B	A OR B	NOT A
T	T	T	T	F
T	F	F	T	F
F	T	F	T	T
F	F	F	F	T

Figure: The basic truth table for AND, OR, NOT.

Building a Truth Table — Step by Step

How many rows?

For n conditions there are 2^n rows — one per combination.

- 1 condition $\rightarrow$ 2 rows
- 2 conditions $\rightarrow$ 4 rows
- 3 conditions $\rightarrow$ 8 rows

Filling the input columns

A neat trick: the first column alternates T,T,...then F,F,...; the last column alternates every row. This guarantees every combination appears exactly once.

Example: A AND (NOT B)

A	B	NOT B	A AND (NOT B)
T	T	F	F
T	F	T	**T**
F	T	F	F
F	F	T	F

Work *inside-out*: compute NOT B first, then the AND.

Truth Table — A Three-Variable Example

Expression: $(A$ AND $B)$ OR C

"Approve a loan if (income is high **AND** credit is good) **OR** a guarantor is present."

Reading the table

The result is *True* whenever C is true, or whenever both A and B are true.

A	B	C	A AND B	$(A$ AND $B)$ OR C
T	T	T	T	**T**
T	T	F	T	**T**
T	F	T	F	**T**
T	F	F	F	F
F	T	T	F	**T**
F	T	F	F	F
F	F	T	F	**T**
F	F	F	F	F

Figure: $2^3 = 8$ rows cover every possibility.

What is a Decision Table?

Rules in a compact table

A **decision table** has four parts:

- **Condition stub**: the list of conditions.
- **Condition entries**: Y/N for each rule.
- **Action stub**: the list of possible actions.
- **Action entries**: which action(s) run for each rule.

Each **column** is one *rule*: a combination of conditions and the action to take.

	Rules			
	R1	R2	R3	R4
Conditions				
Member?	Y	Y	N	N
Bill $>$ 1000?	Y	N	Y	N
Actions				
Give discount	**X**			
No discount		X	X	X

Figure: A decision table for a shop discount.

Building a Decision Table — Worked Example

The rules (in words)

A student passes only if attendance $\geq 75\%$ **AND** marks ≥ 40. Otherwise they fail.

Steps to build it

1. List conditions (attendance, marks).
2. 2 conditions $\Rightarrow 2^2 = 4$ rules.
3. Fill Y/N combinations.
4. Mark the action for each rule.

	R1	R2	R3	R4
Attendance $\geq 75\%$?	Y	Y	N	N
Marks ≥ 40?	Y	N	Y	N
Pass	**X**			
Fail		X	X	X

Only R1 (both Yes) leads to Pass — every other case is Fail.

Decision Table vs. Truth Table

Aspect	Truth Table	Decision Table
Purpose	Evaluate a logical expression	Choose actions from business rules
Values	True / False	Yes / No (conditions), X (actions)
Output	One logical result (T/F)	One or more actions to perform
Rows / columns	2^n rows of combinations	Each column is one rule
Used in	Logic, digital circuits	Software design, decision making

In short: a truth table answers "is it true?"; a decision table answers "what should I do?"

Quick Check — Round 2 (Decision & Truth Tables)

[Q1] How many rows does a truth table with 3 conditions have?

[Q2] Complete: $A =$T, $B =$F. What is A AND B? What is A OR B?

[Q3] In a decision table, what does each *column* represent?

[Q4] One line: how does a decision table differ from a truth table?

Answers — Round 2

[A1]

$2^3 = 8$ rows.

[A2]

A AND B = **False** (both must be true); A OR B = **True** (at least one is true).

[A3]

Each column is one **rule** — a specific combination of condition answers and the action(s) to take.

[A4]

A **truth table** gives a True/False *result*; a **decision table** gives the *action* to perform for each rule.

What is a Dry Run?

Being the computer, on paper

A **dry run** (or **hand tracing**) means executing an algorithm *yourself*, step by step, on paper — writing down how each variable changes.

- No computer needed — just pen and paper.
- Done *before* running the real code.
- Reveals logic errors early and cheaply.

Why it is powerful

You truly **understand** your own logic only after you have traced it by hand at least once.

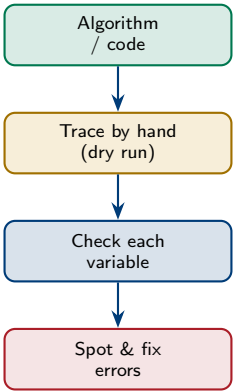


Figure: The dry-run process.

State Tables and Variable Tracking

The tool of a dry run

A **state table** (or **trace table**) has one *column per variable* and one *row per step*. As you execute each line, you write the new value of any variable that changed.

- Columns = the variables to **track**.
- Rows = the steps as they execute.
- The last relevant row shows the *output*.

Tip

Only write a value when it *changes*; carry the others down. This keeps the table clean.

Blank template

Step	var1	var2	Output
1	...	...	
2	...	...	
3	...	...	...

Fill it in as you "become the computer".

Dry Run Example 1 — Swapping Two Variables

Algorithm (using a temp)

1. `a = 5, b = 8`
2. `temp = a`
3. `a = b`
4. `b = temp`

Goal: swap the values of `a` and `b`.

The idea

We need a **temporary** box so we do not lose a's value when we overwrite it.

State table

Step	a	b	temp
1	5	8	–
2	5	8	5
3	**8**	8	5
4	8	**5**	5

Result: a=8, b=5 — **swapped!** ✓

Dry Run Example 2 — Sum of 1 to 4 (Loop Trace)

Algorithm

```
sum = 0
FOR i = 1 TO 4
   sum = sum + i
ENDFOR
PRINT sum
```

Trace each pass

Follow the loop round by round, updating `i` and `sum`.

State table

Pass	i	sum = sum + i	sum
start	–	–	0
1	1	$0+1$	1
2	2	$1+2$	3
3	3	$3+3$	6
4	4	$6+4$	**10**

When i reaches 5, $i > 4$ so the loop stops. Output: **10**.

Dry Run Example 3 — Finding an Error (Debugging)

A buggy algorithm

"Count down from 3."

```
i = 3
WHILE i > 0
   PRINT i
ENDWHILE
```

Can you see the bug before tracing?

Trace reveals the bug

Step	i	i > 0? / Output
1	3	T / print 3
2	3	T / print 3
3	3	T / print 3
...	3	**never ends!**

The fix

We forgot to **update** `i`. Add `i = i - 1` inside the loop. **Dry running caught an infinite loop.**

What is a Test Case?

Checking with chosen inputs

A **test case** is a specific **input** together with the **expected output**. We run the algorithm on it and compare.

- If actual $=$ expected $\rightarrow$ test **passes**.
- If actual $\neq$ expected $\rightarrow$ test **fails** (a bug!).

Good testing uses *several* test cases, not just one.

Example (larger of two numbers)

- Input $(7, 3) \rightarrow$ expected 7.
- Input $(2, 9) \rightarrow$ expected 9.
- Input $(5, 5) \rightarrow$ expected 5.

Test the tricky cases

Always test:

- a **normal** case,
- a **boundary** case (e.g. equal values, zero),
- an **extreme / invalid** case (e.g. negative, empty).

Bugs love to hide at the boundaries!

Designing Good Test Cases — Worked Example

The program

"Read marks (0–100); print *Pass* if marks ≥ 40, else *Fail*."

Which inputs to try?

- Clearly pass and clearly fail.
- The **boundary**: exactly 40.
- Invalid inputs: -5 or 150.

Input	Type	Expected	Reason
85	Normal	Pass	clearly above 40
20	Normal	Fail	clearly below 40
40	Boundary	Pass	exactly the cut-off
39	Boundary	Fail	just below cut-off
-5	Invalid	Error	out of range
150	Invalid	Error	out of range

Figure: A small but thorough set of test cases.

Errors — Syntax, Logic and Runtime

Three kinds of program errors

- **Syntax error**: broken grammar of the language (a missing bracket). The program will not even run.
- **Logic error**: the program runs but gives the *wrong* answer (e.g. used $+$ instead of $-$).
- **Runtime error**: it crashes while running (e.g. divide by zero).

Where each is caught

- Syntax $\rightarrow$ by the *compiler / interpreter*.
- Runtime $\rightarrow$ while the program *runs*.
- Logic $\rightarrow$ often only by **dry runs** and **test cases**!

Why this unit matters

Dry runs and test cases are our best weapons against the sneakiest errors: **logic errors**.

Quick Check — Round 3 (Tracing & Testing)

[Q1] What is a dry run, in one line?

[Q2] In a state table, what do the columns and rows represent?

[Q3] Dry-run this and give the output:

```
x = 2
y = 3
x = x + y
y = x - y
```

[Q4] Name the three kinds of program errors.

Answers — Round 3

[A1]

A **dry run** is executing an algorithm by hand, step by step, on paper, tracking how each variable changes.

[A2]

Columns = the **variables** being tracked; rows = the **steps** as the algorithm executes.

[A4]

Syntax errors, **logic** errors, **runtime** errors.

[A3] Trace

Step	x	y
start	2	3
x=x+y	**5**	3
y=x−y	5	**2**

Final: x=5, y=2.

Summary of Unit 3

What we covered

1. **Flowcharts** — standard symbols (oval, parallelogram, rectangle, diamond, connector, flowline) and the flow of a program.
2. **Control structures** — sequence, selection (branching) and iteration (loops), with input and output.
3. **Truth tables** — AND, OR, NOT; 2^n rows evaluate a logical expression for every case.
4. **Decision tables** — compact rules that map condition combinations to actions.
5. **Execution tracing** — dry runs, state tables and variable tracking to check logic by hand.
6. **Testing** — test cases (normal, boundary, invalid) and the three kinds of errors: syntax, logic, runtime.

See the logic, organise the conditions, and test by hand — before a single line of code runs.

Key Terms to Remember

Flowcharts & logic

- **Oval / Diamond / Parallelogram** = start-stop / decision / I-O
- **Sequence, Selection, Iteration**
- **AND, OR, NOT** truth values
- **Truth table** = 2^n rows
- **Decision table** = rules $\rightarrow$ actions

Tracing & testing

- **Dry run** = trace by hand
- **State / trace table** = variables vs steps
- **Test case** = input + expected output
- **Boundary case** = the tricky edge
- **Syntax / Logic / Runtime** errors

Practice Questions (Exam Oriented)

Short answer (2–5 marks)

1. Draw and name the six standard flowchart symbols.
2. Draw a flowchart to check whether a number is even or odd.
3. Write the truth table for NOT(A AND B).
4. What is a dry run? Why is it useful?
5. Give three types of test cases with one example each.

Long answer (8–10 marks)

6. Draw a flowchart to find the largest of three numbers.
7. Build a decision table for a bank that gives a loan if (salary $\geq$ 30k *AND* age $<$ 60) *OR* a guarantor exists.
8. Dry-run a loop that prints the multiplication table of 5, using a state table.
9. Explain sequence, selection and iteration with a flowchart for each.
10. Differentiate syntax, logic and runtime errors with examples, and say how each is found.

Think & Explore (Take-Home)

Mini tasks

- Draw a flowchart for making a cup of tea (with a decision: "sugar?").
- Build a truth table for $(A$ OR $B)$ AND (NOT $C)$.
- Make a decision table for choosing an outfit from "raining?" and "cold?".
- Dry-run an algorithm that finds the maximum of the list $[4, 9, 2, 7]$ using a state table.

Reading & reflection

- Draw the flowchart of a daily routine you follow.
- Explore a free tool (e.g. draw.io) to make a neat flowchart.
- Prepare for Unit 4: *Introduction to Artificial Intelligence*.

Reminder

Practise dry runs on every algorithm you write — it is the fastest way to become a confident programmer.

References

Textbooks & sources (as per the course page)

1. Peter Norton, *Introduction to Computers*, Tata McGraw Hill, 6th Edition. **[Main text]**
2. R. G. Dromey, *How to Solve it by Computer*, Pearson.
3. ISRD Group, *Introduction to Information Technology*, Tata McGraw Hill.
4. E. Balagurusamy, *Fundamentals of Computers*, McGraw Hill.

Standards & further reading

5. ISO 5807:1985, *Information processing — Documentation symbols and conventions for flowcharts*.
6. M. Morris Mano, *Digital Logic and Computer Design*, Pearson (truth tables).

Course page: https://www.gcjana.in/courses/shardauniversity/2601/fundamentals-of-computing-and-artificial-intelligence/

Thank You

Any Questions?

"Testing shows the presence, not the absence, of bugs —
but a good dry run is where fixing them begins."

Post your doubts on the course page →

https://www.gcjana.in/courses/shardauniversity/2601/fundamentals-of-computing-and-artificial-intelligence/#Post-doubts