CS1002-Programming Fundamentals (CS-A,B,C,D,E,F,G,H,J)

FALL 2025

Assignment 2

The aim of this assignment is to give you practice with **arithmetic operations, decision structures, and bitwise operators**. Zero marks will be awarded if topics not taught in class are used, and use of loops (any sort) are **strictly prohibited** except for the bonus question.

Instructions for submission:

I. Outline a draft of your code logic on paper before attempting the assignment, and bring those drafts with you for demos. If the drafts do not match your code, marks will be deducted.

II. Displayed output should be well mannered and well presented. Use appropriate comments and indentation in every code segment; anyone caught using AI-generated comments shall receive 0 marks in that question.

III. Combine each question into a single switch statement such that running the .exe file should first ask for input to decide which question shall be run. If a question is attempted on a separate .cpp file, marks will be deducted.

IV. Run and test your program on a lab machine before submission.

V. Make a single .cpp file and name it as ROLL- NUM_SECTION.cpp e.g (23i-0001_A.cpp). The file should contain your name and id on the top of the file in comments.

VI. Make a scanned pdf of your drafts and name it ROLL- NUM_SECTION_DRAFT.pdf

VII. Make a scanned pdf of the flowchart in question 5 and name it as ROLL-NUM_SECTION_Q5.pdf

VIII. Combine all your work into one folder named ROLL-NUM_SECTION, with only .cpp and .pdf files. Compress it into a **.zip** file, not **.rar**, named ROLL-NUM_SECTION.zip

IX. Submit the .zip file on Google Classroom within the deadline.

X. Submission other than Google classroom (e.g. email etc.) will not be accepted.

XI. The student is solely responsible to check the final zip files for issues like corrupt file, virus in the file, mistakenly exe sent. If we cannot download the file from Google classroom due to any reason it will lead to zero marks in the assignment.

Deadline:

Deadline to submit assignment is **October 22,2025 11:59 PM**. You are supposed to submit your assignment on GOOGLE CLASSROOM (CLASSROOM TAB not lab). Only ".ZIP" files are acceptable. Other formats will be directly given **0**. Correct and timely submission of the

assignment is the responsibility of every student, hence no relaxation will be given to anyone. Late Submission policy will be applied as described in course outline.

Tip: For timely completion of the assignment, start as early as possible.

Plagiarism: Plagiarism is not allowed. If found plagiarized, you will be awarded zero marks in the assignment

Note: Follow the given instructions to the letter, failing to do so will result in a zero.

Good Luck :P

Q1: Campus Print Shop [30 Marks]

FAST's Campus Print Shop bills students per job. Each student also has a monthly free quota that can partially cover a job. Admins warned that submissions with wrong options must be rejected.

Inputs

1. `char size` in `{ 'A','B','C' }` where A=A4, B=A3, C=A5
2. `char mode` in `{ 'C','G' }` Color or Grayscale
3. `int pages` positive
4. `int duplex` in `{0,1}` where 1 means both sides
5. `int quotaLeft` remaining free pages for the month

Pricing rules

- Base per page by `size` via switch: A4=7, A3=12, A5=5
- Color adds +4 per page, Grayscale adds 0
- Duplex prints 2 pages per sheet but billing is per page
- If `pages >= 100` apply 10 percent bulk discount on the page-rate subtotal
- Quota reduces payable pages: `chargeable = max(0, pages - quotaLeft)`
- Environmental fee: add PKR 30 if `pages > 150`
- Rush guard: if `mode=='C'` and `pages>300`, print `Job Rejected: Color limit exceeded`

Output

If invalid, print `Invalid input`. Otherwise print exactly:

```
ChargeablePages=...
PerPage=...
Subtotal=...
Fees=...
Total=...
```

Constraints

Use `switch` for size mapping and `if` for checks. No arrays, no loops, no functions.

Q2 : Final Checkout [50 Marks]

The street is swarming with walkers. Shutters rattle against the cold air. This is the last store before the highway, your final shot at survival.

Your task: compute the checkout total, how much your wallet covers, and whether you escape alive.
All conditional logic must use ternary operators only.
Use integer arithmetic only throughout.

Input Order

- role {'S','M','E'} — Scavenger, Medic, Engineer
- timeLeft [0..59] — Minutes to dawn
- noise [0..100] — Current store noise level
- wounds [0..3]
- tempC [-30..50] — Ambient temperature
- rain {0,1} — 1 = raining
- powerOn {0,1} — 1 = store mains active
- barricade {0,1} — 1 = intact
- takeFood {0,1} — Taking food
- foodKg [0..50] — Food weight
- takeMeds {0,1} — Taking medical kits
- medKits [0..20] — Number of med kits
- takeTools {0,1} — Taking tools
- toolKg [0..80] — Tools weight
- takeAmmo {0,1} — Taking ammo
- ammoBoxes [0..50] — Ammo box count
- cartKg [0..200] — Cart total weight
- morale [0..100] — Morale level
- zCount [0..500] — Zombies nearby
- promo {0,1,2,3} — 0=None, 1=RATION10, 2=SILENCE25, 3=FIRSTAID40
- wallet [0..5000] — Wallet cash
- keys {0,1} — Vehicle keys
- fuelL [0..60] — Fuel in liters
- companion {0,1} — 1 = you're not alone

Constants and Base Prices

Food 120 per kg
Med Kit 500 each
Tools 200 per kg
Ammo Box 900 each

Role Carry Capacity Base:

S=35, M=30, E=40
If companion==1 add 15.

Role Perks (Apply Before Surge or Discounts)

Scavenger: Food & Tools −10%
Medic: Med kits −20%
Engineer: Tools −15%
If role=='E' and powerOn==0, reduce Server Fee by 15 before waive (min 0).
Percentages floor after each step.

Noise Adjustments

Start noiseAdj = noise
+2*toolKg if takeTools==1
+1*ammoBoxes if takeAmmo==1
+5 if powerOn==1
Clamp to max 100.

Threat Multiplier H (Hundredths)

Start H=100
Modify in order, clamp to [80..220]
1. Horde size (+40/+20/+10)
2. Noise pressure +10 if noiseAdj>60
3. Gate +15 if barricade==0
4. Weather −10 if rain==1
5. Temp ±10 for <0 or >40.

Rush Fee and Server Fee

Rush Fee: 200 if <10, else 120 if <20, else 0
Server Fee: default 80, Engineer no-power reduces by 15 (min 0).

Carry Penalty

carryCap = baseCap(role)+(companion==1?15:0)
carryPenalty = 30*max(0,cartKg−carryCap)

Item Costs (Before Surge)

Use role-adjusted prices.
FoodCost, MedCost, ToolCost, AmmoCost as per conditions.
If foodKg≥25 reduce FoodCost by 12%.

Subtotal Before Promos

itemsPreSurge = sum of item costs
itemsAfterSurge = itemsPreSurge * H / 100
subtotalBeforePromos = itemsAfterSurge + rushFee + serverFee + carryPenalty

Stealth Waive

If takeAmmo==0 & noiseAdj<=40 & barricade==1, remove server fee.

Credits

Teamwork Credit −70 if companion==1
Focus Credit −50 if wounds==0 & morale≥80

Injury Taxes

+90 if wounds≥2, +60 more if wounds==3 (part of Fees)

Promos (Exactly One, After Credits & Injury Taxes)

foodAfterSurge=(FoodCost*H)/100, medAfterSurge=(MedCost*H)/100
1=RATION10: 10% food (cap 800)
2=SILENCE25: 25% of (rushFee+serverFee) if noiseAdj≥50 & takeAmmo==0 (cap 300)
3=FIRSTAID40: 40% of med if wounds≥2 (cap 1000)
0=None

Debt Pressure and Minimum Checkout

If subtotal>wallet add DebtFee=100, else 0.
If total<150 set total=150.
DebtFee adds to Fees.

Wallet Usage

WalletUsed = min(wallet, total)
Payable = total − WalletUsed

Escape Chance & Outcome

Start 30
+2*ammoBoxes if takeAmmo==1
+1*medKits if takeMeds==1
+min(20,max(0,fuelL−5))
+10 if keys==1 & fuelL≥5 else −20
−max(0,cartKg−carryCap)
−H/3
+morale/5

−10 if wounds≥2
−15 if DebtFee>0
Clamp [0..100]
Outcome=Survived if ≥50 else Caught

Output Format (Exact Order & Spelling)

FoodCost=...
MedCost=...
ToolCost=...
AmmoCost=...
H=... (as X.YY)
RushFee=...
ServerFee=...
CarryPenalty=...
SubtotalBeforePromos=...
PromoDiscount=...
Fees=...
Total=...
WalletUsed=...
Payable=...
EscapeChance=...
Outcome=...

Note:
No use of if, else, or switch is allowed.
No loop-based branching us allowed.
Use Integer math only.
Follow order as mentioned exactly.
Clamp H & EscapeChance.
No extra inputs or hidden states are allowed.

Q3: Holocron of Malachor [30 Marks]

In the forgotten archives of Malachor V lies a Sith Holocron said to hold secrets powerful enough to sway the entire galaxy. This Holocron is unlike any other—it does not speak in clear words, but in fragments of the Force, bound into 16 shards. Each shard may either shine with the light **(1)** or sink into the darkness **(0)**. Together they form Holocron's essence, a number whispered to contain both creation and destruction. To test a worthy apprentice, the Holocron demands answers to ancient riddles. And though the Sith tried to bury their meaning in riddles, the Jedi archivists left faint clues. You must decipher them to awaken the Holocron. Also, some manuscripts mention

that inverting Holocron is the path of the Sith where every Light becomes Dark, every Dark becomes Light, but beware of this forbidden path, for it corrupts all shards within.

The Riddles of the Holocron:

1. “Gaze into the mirror, and all things appear reversed. The first shard becomes the last, and the last the first, until the Holocron itself is inverted in the reflection.”
2. “The Force always flows forward, a river without pause. What is cast away returns to the beginning, and thus the shards march ever forward.”
3. “Count the flames that burn with brilliance, for every shard that shines adds to the strength of the Jedi. Only in knowing how many lights remain can the Holocron’s glow be measured.”
4. “At the edge of all things lies the heaviest shard. If the weight of Darkness rests upon it, the Seal is broken, and chaos unfurls. If Light endures, the Seal remains intact.”

Thus, the keeper of the Holocron must enter the number and choose which verse to enact. Only then does the Holocron reveal its truth.

Note:

Use of Loops, If-Else, Ternary, Logical Operators and (/,*) Operators is not allowed.

Q4: Formulaic Engima [30 Marks]

Part A

Scientists at the Computational Climate Analysis Unit (CCAU) of FAST are investigating how rising greenhouse gas emissions contribute to oceanic expansion. Their simulation accounts for multiple environmental parameters such as time progression, mean surface temperature deviation, and carbon dioxide levels in the atmosphere. The objective is to forecast future fluctuations in ocean volume and assist in developing adaptive environmental policies.

Using the current model, determine the estimated ocean expansion using the formula below:

$$\Delta O = P \cdot e^{-\frac{(y-y_0)}{\mu}} + Q \cdot \sin(\kappa y) + \eta \cdot \ln\left(\frac{G+G_0}{\theta}\right) + \lambda \cdot \sqrt{X - X_0}$$

Where:

ΔO: Ocean expansion (in meters)

y: Years since 2010

G: Mean surface temperature deviation (in °C)

X: Atmospheric CO_2 levels (in ppm)

Constants:

P = 0.12, Q = 0.07

$\mu = 40$, $\kappa = 0.45$, $\eta = 1.3$

$\theta = 25$, $\lambda = 0.9$

$y_0 = 20$, $G_0 = 0.8$, $X_0 = 380$

Use the cmath library in C++ for all trigonometric and logarithmic computations.

YOU ARE ONLY ALLOWED THESE LIBRARIES' FUNCTIONS IN THIS QUESTION

Part B

You have recently joined a cyber defense firm responsible for developing a secure communication framework for IoT-based devices. Your task is to design an encryption and decryption process that ensures all transmitted numerical data remains confidential even if intercepted by external agents.

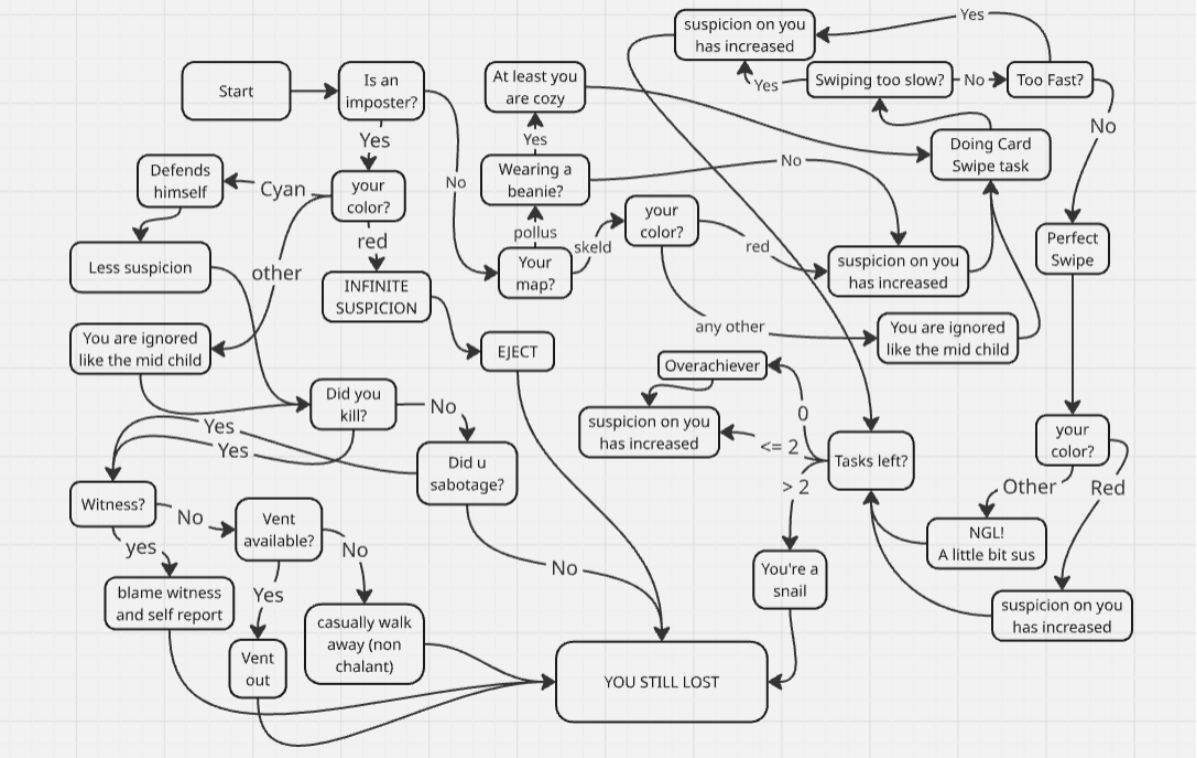
1. Let the input value be m.
2. Use the last four digits of your roll number (e.g., 24I-5678) and compute its remainder when divided by 11.
3. Add 150 to m.
4. Multiply the resulting number by the value of m.
5. Divide the new result by the remainder obtained in step 2.
6. Multiply it by 13.
7. Subtract 8 from the result.
8. Divide it by the remainder of the last four digits of your roll number when divided by 9.
9. Add 4 to the final value.

Part C

Based on the encryption sequence above, create a decryption algorithm that can accurately reverse the encryption process when the encrypted number is received at the target device.

Q5: Among Us [30 Marks]

A random guy once said, “26 mein to duniya khatam hai”. Guess what? It didn’t end. You are far into the future. In 2077 to be exact. Civilizations as we know it have been destroyed by AI’s takeover. You and your friend were spared because you always said “thank you” to every LLM after using it. In your free time, you decide to make an algorithm that plays the game **Among Us** for you. Your friend isn’t a CS major so he, just vaguely, gives you a structure and asks you to implement it.



Your task:

PART I:
First, you’ll have to construct a proper flowchart using this vague structure and flow with proper symbols.
PART II:
Implementation of a C++ program that mimics the exact behavior of this structure. Note that for this implementation you’re only allowed to use the switch-case structure from all the decision structures.

NOTE: Flowchart should be made by hand on A4 size paper (1 or more) and scanned copy should be submitted. It should be kept saved for it’ll be required during the demos.

Q6: The Challenge of Arithmetic Operation [30Marks]

In the realm of computer science, there exists a fascinating constraint that has challenged programmers throughout the digital age: the restrictive boundaries of standard integer data types and their inherent limitations in representing the full spectrum of numerical values we encounter in sophisticated applications. Picture yourself developing a cryptographic system or engineering a scientific calculator that must manipulate substantial seven-digit numbers. These are values that, while not infinite, push against the comfortable margins of typical integer operations and demand meticulous handling to prevent overflow catastrophes and precision degradation.

Your Mission:

Your task is to architect an advanced arithmetic engine capable of managing numbers that stretch to the absolute limits of ten-digit magnitude, operating across multiple numeral systems with unwavering accuracy. These are problems worthy of requiring sophisticated algorithmic approaches that traditional programming shortcuts simply cannot accommodate.

Input Requirements:

Your program must commence by soliciting two formidable numbers from the user. These are values that teeter on the precipice of what conventional integer types can reliably process. These numerical behemoths won't necessarily manifest in the standard decimal notation we casually experience in our daily life..

Your system must:

- Ask the user which numeral system they want to work in
- Accept any base from 2 through 16 (binary, ternary, quaternary, quinary, senary, septenary, octal, nonary, decimal, undecimal, duodecimal, tridecimal, tetradecimal, pentadecimal, or hexadecimal)
- Handle input numbers that can be up to ten digits long in their specified base

You are gonna build an all in one system that encompasses all. Binary dominates in digital logic and low-level programming. Hexadecimal appears everywhere in memory addresses and color representations. Octal still surfaces in Unix file permissions. Decimal remains intuitive for human comprehension; other bases have as such uses for deciphering internal patterns.

- Your algorithm must accommodate the full ten-digit range with mathematical exactitude.
- You cannot resort to using string datatype or any of its library functions
- Usage of loops is a forbidden no go area.

- You must work with actual numeric types and manage the digits through proper algorithmic techniques

Required Operations:

The nucleus of functionality revolves around implementing four arithmetic operations:

- Addition
- Subtraction
- Multiplication
- Division

Your task is to make each of these operations work flawlessly with numbers spanning the complete ten-digit spectrum.

Output Requirements:

Your program must render the results of all four operations back to the user in the identical numeral system they initially specified.

This is a proper test of your algorithmic thinking and your ability to work within the genuine constraints that real systems face.

Q7: Five Nights at NASCON [100 Marks]

Note: What is Five Nights at Freddy's? | Overview, history, and more!

NASCON has gone dark. You are the PF TA trapped overnight in the FAST campus control room. Four faulty animatronics, each themed after a legendary course, roam the venue. Your only tools are a battery, a debug terminal, and a locked-down panel that exposes a few 8-bit registers. You must survive five nights by reacting to incidents, managing power, and keeping the office safe.

Registers

You maintain these 8-bit integers throughout the program:

1. SYS system-state bitmask
   Bit 0 Cameras
   Bit 1 LeftDoor
   Bit 2 RightDoor
   Bit 3 VentGate
   Bit 4 StageLights
   Bit 5 VentFans
   Bit 6 Generator
   Bit 7 PanicAlarm

A bit value 1 means that subsystem is ON or CLOSED where applicable.

2. `L0G` action-log bitmask
    - Initially set to 0
    - When your program changes a `SYS` bit or auto-adjusts a subsystem due to a rule, set the same bit in `L0G`.

3. Animatronic position masks, one per bot, each with exactly one room bit set at any time:

    - `A1` DeanBoy
    - `A2` APChika
    - `A3` CalculusFoxy
    - `A4` PFazzBear

4. Room bits shared by all animatronics:
   Bit 0 Stage
   Bit 1 HallL
   Bit 2 HallR
   Bit 3 Vent
   Bit 4 Labs
   Bit 5 Library
   Bit 6 Courtyard
   Bit 7 Office

   At all times each `A*` must have exactly one 1-bit.

Movement graph for rooms

Animatronics can only move along these allowed edges. Movements that violate this graph are invalid.

- Stage <-> HallL
- Stage <-> HallR
- HallL <-> Office
- HallR <-> Office
- Vent <-> Office
- Labs <-> HallL
- Library <-> HallR
- Courtyard <-> Vent

You must enforce this adjacency when processing a move. Use a `switch` or equivalent logic to validate the source room and its legal destinations.

Nights, incidents, and loop

You will process exactly five nights. Each night has exactly one incident line. Process them in order using one loop that iterates five times. After each incident, apply safety rules, then drain power, then check office safety. If power hits zero you still continue to process remaining nights with whatever state remains.

Incident decoding

Each incident line starts with an integer `evt` and one argument. Use a `switch` on `evt` to decode and handle it.

1. `evt = 1` Move
   Format: `1 a t`
   - a in `{1,2,3,4}` selects which animatronic.
   - `t` in `[0..7]` is the destination room bit index.
     Steps:
   - Determine the animatronic's current room index from its mask.
   - If `t` is not adjacent to that current room by the movement graph, print `InvalidMove` and ignore.
   - Otherwise set its mask so only bit `t` is 1. No change to LOG.

2. `evt = 2` Jam
   Format: `2 X`
   - X in `{ 'L','R','V' }` jams LeftDoor, RightDoor, or VentGate.
     Steps:
   - Force the corresponding SYS bit to 1 meaning closed for doors and 1 meaning closed for the vent gate.
   - Print `DoorSealed` for L or R, or `VentSealed` for V.
   - Set the same bit in LOG if your program changed its value.

3. `evt = 3` Brownout
   Format: `3 load`
   - `load` in `[0..200]`.
     Steps:
   - If `load > 120` and Generator is 0, print `Outage` then force StageLights and VentFans to 0. Set bits 4 and 5 in LOG if they changed.
   - If `load > 120` and Generator is 1, print `Stable`. No SYS change.
   - Otherwise no message and no SYS change.

4. `evt = 4` Override
   Format: `4 X`

   - `X` in `{ 'C','L','F','G','A' }` maps to Cameras, StageLights, VentFans, Generator, PanicAlarm.
     Steps:
   - Flip the mapped bit in `SYS`.
   - Set the same bit in `LOG`.
   - Exception for Alarm: if any of LeftDoor, RightDoor, or VentGate is 0 at the time you try to turn Alarm on, print `CallSecurity`, then immediately clear Alarm to 0 and set bit 7 in `LOG`.

Validation for incidents:

- If `evt` is not in `{1,2,3,4}`, print `CorruptEvent` and ignore the line.
- If a is outside `{1,2,3,4}` or `t` outside `[0..7]`, print `InvalidTarget` and ignore the move.
- If `X` is not one of the allowed letters for its incident, print `InvalidCode` and ignore.
- If `load` is outside `[0..200]`, print `InvalidLoad` and ignore.

Safety rules applied after each incident

Apply these in the order written. For each rule, print at most one message per night when it triggers.

1. Energy saver
   - If StageLights is 1 while Generator is 0, print `DimMode` then set StageLights to 0. Set bit 4 in `LOG` if it changed.
2. Printer trap prevention
   - If Cameras is 0 and both LeftDoor and RightDoor are 1, print `BlindLock` then set RightDoor to 0. Set bit 2 in `LOG`.
3. Vent evacuation
   - If PanicAlarm is 1 and VentFans is 1, print `Evacuate` then set VentFans to 0. Set bit 5 in `LOG`.

Power drain after safety

You maintain an integer `P` as battery percent. After safety rules for that night:

1. Let k be the number of 1-bits in `SYS`. Reduce `P` by `2*k`. Clamp `P` to a minimum of 0.
2. If `P == 0`, print `LimpMode` and force Cameras, StageLights, and VentFans to 0. Set their bits in `LOG` if any changed. Doors, Generator, and Alarm remain unchanged.

Office safety check

After power drain each night, check if any animatronic is in the Office:

- If A* has bit 7 set and the corresponding entry is open, you are caught.
  Entry rules:
    - If it came from HallL, LeftDoor must be 1 to block.
    - If it came from HallR, RightDoor must be 1 to block.
    - If it came from Vent, VentGate must be 1 to block.

- If an animatronic is in Office and the matching barrier is 0, print `Caught` and immediately push that animatronic back to Stage by setting its mask to only bit 0. Set bit 0 in LOG. The night continues.

To decide the matching entry, use the animatronic's room before it moved into Office during that night if it moved by `evt = 1`. If it was already in Office at the start of the night, treat the entry as Vent by default.

Closing report after Night 5

Print exactly three lines:

```
SYS=xxxxxxxx
LOG=xxxxxxxx
Score=S
```

- Show masks from bit 7 to bit 0 as 0 or 1 characters.

- Score S is computed as:
  Start at 50
  Add the number of 1-bits in LOG
  Add 10 if Generator is 1 at the end
  Subtract 15 if Cameras is 0 at the end
  Subtract 5 for each of LeftDoor and RightDoor that is 1 at the end while Cameras is 0

Program I O contract

Input

Line 1: integer SYS in `[0..255]` initial panel state
Line 2: integer LOG initialised by you to 0 inside the program. The line is not provided in input.
Line 2 of input: integer P in `[0..100]` initial battery percent
Lines 3..7: exactly 5 incident lines, one per night, following the formats above

Processing constraints

- Use exactly one loop that runs five iterations.
- Use one `switch` on `evt`.
- Use bitwise operators for all mask edits.
- Do not use arrays, vectors, or user-defined functions.

Output

- Print any messages as they occur, each on its own line.
  Possible messages: `InvalidMove`, `DoorSealed`, `VentSealed`, `Outage`, `Stable`, `CorruptEvent`, `InvalidTarget`, `InvalidCode`, `InvalidLoad`, `CallSecurity`, `DimMode`, `BlindLock`, `Evacuate`, `LimpMode`, `Caught`

- After Night 5, print the three-line closing report exactly as specified.

Sample

Input

```
5
40
1 2 7
2 L
3 150
4 G
4 A
```

Possible Output

```
InvalidMove
DoorSealed
Outage
LimpMode
CallSecurity
SYS=01000010
LOG=11110011
Score=60
```

Story notes for the sample

SYS starts as 00000101 so Cameras and StageLights are on. Night 1 tries to jump from HallR to Office without legal adjacency from Stage, so `InvalidMove`. Night 2 jams LeftDoor closed. Night 3 brownout with Generator off cuts Lights and Fans, then power drains to zero and triggers `LimpMode`. Night 4 flips Generator. Night 5 tries to arm Alarm while an entry is open which prints `CallSecurity` and clears it. The final report shows masks and score.

BONUS QUESTION

Q8 : Cryptographic Serpentine

The Mission

Three days ago, your intelligence unit intercepted an encrypted data packet from a hostile research facility deep in foreign territory. The transmission was brief, just ten 64-bit integers, but the metadata suggested something significant. Building schematics. Floor plans. Structural data that could reveal weak points, security gaps, ventilation systems.

Your crypto-analysis team worked around the clock and finally cracked the encoding scheme. The blueprints aren't stored as images or coordinates. They're hidden in the binary structure itself, woven into the bit patterns of intercepted data. Clever! Almost invisible.

But here's where it gets complicated.

The enemy's network monitoring system is sophisticated. Really sophisticated. It tracks computational patterns in real time, looking for suspicious activity. Loops show up like searchlights. Array accesses create memory patterns that their intrusion detection picks up immediately. Even recursion leaves a recognizable call stack signature. Everything that one can think of could trigger their security monitoring systems.

All but Direct bit manipulation. Your team discovered something interesting about their monitoring system. It samples processor activity at fixed intervals, checking for pattern recognition in the instruction pipeline. But bitwise operations are atomic. They complete in a single machine cycle. One clock tick. Below the sampling threshold. By the time their sensors sweep the next cycle, the operation is already done. Invisible.

So you need to write a decoder that works entirely through bitwise logic and explicit computation. No loops that would create repetitive instruction patterns. No arrays that would generate predictable memory access sequences. No recursion that would build a telltale call stack. Just pure, single-cycle bitwise operations, carefully chained together.

The decoded blueprints could be the difference between a successful operation and a disaster. Your team is counting on you.

Time to get to work.

Understanding the Data

The Input

Your program will read exactly 10 standard `long long integers` in C++. Let's call them `A1, A2, A3, A4, A5, A6, A7, A8, A9, A10`.

How Floors Map to Bits

Each of these 10 integers contains 64 bits, which means each integer holds data for 16 different floors. Here's how it works:

- Each floor uses exactly 4 consecutive bit positions
- Floor F (where F ranges from 1 to 16) uses bit positions starting at

  `p0 = 4 × (F - 1)`

- The four positions for floor F are: `p0`, `p1 = p0 + 1`, `p2 = p0 + 2`, and `p3 = p0 + 3`

Example

If the user wants floor 3:

- `p0 = 4 × (3 - 1) = 8`
- `p1 = 9`, `p2 = 10`, `p3 = 11`

You'll be reading bits 8, 9, 10, and 11 from all 10 integers.

Building the Bit Stream

Once you know which bit positions to read, you need to construct a 40-bit sequence. This is where things get interesting. The data isn't stored linearly. It's stored in a serpentine (snake-like) pattern.

The Serpentine Pattern

For the four bit positions of floor F:

1. **At position p0:** Read the bit from A1, then A2, then A3, ... up to A10 (forward direction)
2. **At position p1:** Read the bit from A10, then A9, then A8, ... down to A1 (backward direction)
3. **At position p2:** Read the bit from A1, then A2, then A3, ... up to A10 (forward direction)

4. **At position p3:** Read the bit from A10, then A9, then A8, ... down to A1 (backward direction)

This gives you exactly 40 bits total (10 bits per position × 4 positions). Let us just call this sequence `B`, where `B(0)` is the first bit you collected and `B(39)` is the last.

Creating Overlapping Tokens

Now comes the clever part. You're going to slice this 40-bit stream into 20 overlapping tokens. Each token is 3 bits wide, but you move forward by only 2 bits each time (that's the overlap).

Token Extraction

For token index `i` (where i ranges from 0 to 19):

- Token `T_i` consists of three bits: `(L_i, V_i, R_i)`
- `L_i = B(2×i)` (does this cell *want* to connect to its left neighbor?)
- `V_i = B(2×i + 1)` (does this cell *want* to connect vertically ?)
- `R_i = B(2×i + 2)` (does this cell *want* to connect to its right neighbor?)

It must also be noted that the sequence wraps around circularly. When your index exceeds 39, you need to wrap back to the beginning of the 40-bit stream. This is how the overlapping works at the edges.

What These Bits Mean

Think of each token as representing a single cell in your floor plan:

- `L_i`: suggests whether there's a connection going left
- `V_i`: suggests whether there's a vertical connection
- `R_i`: suggests whether there's a connection going right

Laying Out the Grid

You now have 20 tokens. These get arranged into a rectangular grid that's 5 columns wide and 4 rows tall.

Grid Arrangement

- Row 0 contains tokens T0, T1, T2, T3, T4
- Row 1 contains tokens T5, T6, T7, T8, T9
- Row 2 contains tokens T10, T11, T12, T13, T14
- Row 3 contains tokens T15, T16, T17, T18, T19

For any cell at row `r` and column `c`, its token index is `i = r × 5 + c`.

The Magic of Auto-Joining Edges

Here's where the design gets elegant. Each cell doesn't just use its own token bits. It also looks at its neighbors to create automatic connections. This is how corridors link up and corners form up in the grid.

The Joining Rules

For each cell `i` at position (row r, column c), compute four final connection values:

Left connection:

- If c > 0: `L(i) = L_i OR R_(i-1)` (merge with neighbor's right)
- Otherwise: `L(i) = L_i`

Right connection:

- If c < 4: `R(i) = R_i OR L_(i+1)` (merge with neighbor's left)
- Otherwise: `R(i) = R_i`

Up connection:

- If r > 0: `U(i) = V_i OR V_(i-5)` (merge with neighbor above's vertical)
- Otherwise: `U(i) = V_i`

Down connection:

- If r < 3: `D(i) = V_i OR V_(i+5)` (merge with neighbor below's vertical)
- Otherwise: `D(i) = V_i`

Drawing the Tiles

Each cell now has four boolean values: `L(i)`, `U(i)`, `R(i)`, `D(i)`. These form a 4-bit pattern that determines what character to print for that cell.

ASCII Tile Legend

Use these ASCII characters for the blueprint:

- `0000` (no connections): space `' '`
- `0101` (up and down): `'|'`
- `1010` (left and right): `'-'`
- `0011` (right and down): `'J'`
- `0110` (up and right): `'r'`
- `1001` (left and down): `'L'`
- `1100` (left and up): `'7'`
- `1110` (left, up, and right): `'T'`
- `1101` (left, down, and right): `'U'`
- `0111` (up, right, and down): `'>'`
- `1011` (left, up, and down): `'<'`
- `1111` (all four directions): `'+'`

Each character is chosen to visually suggest the connection pattern where possible.

Printing the Floor Plan

You must print a 5×4 grid with borders, but you cannot use any loops.

Output Requirements

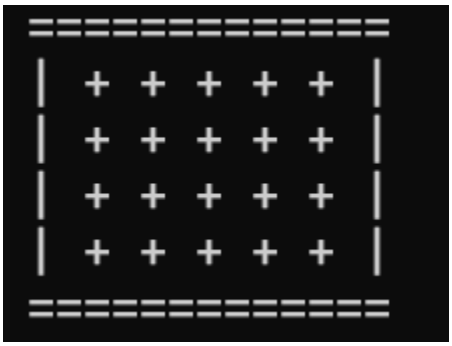
1. **Read the floor number F** (valid range: 1 to 16)
    - If F is outside this range, print `INVALID FLOOR` and exit
2. **Print top border:** A line of '=' characters
3. **Print each of the 4 rows:**
    - Start with | (left border)
    - Print each of the 5 cells, using `setw(2)` so each cell takes 2 characters (the tile plus a space)

    - End with | (right border)
4. **Print bottom border:** A line of = characters

Example:

For input
15
15
15
15
15
15
15
15
15
15

The First Floor:



The Second Floor:

Scalable Blueprints

The enemy facilities vary in size. Sometimes you need a detailed view, other times you need to see patterns at a larger scale. Add a scaling feature to your decoder.

Additional Input

After reading the floor number F, read two more integers:

- `scale_width`: horizontal scaling factor (1 or greater)
- `scale_height`: vertical scaling factor (1 or greater)

Scaling Rules

A 1×1 scale renders the original 5×4 grid normally (as described above). A 2×2 scale doubles both dimensions:

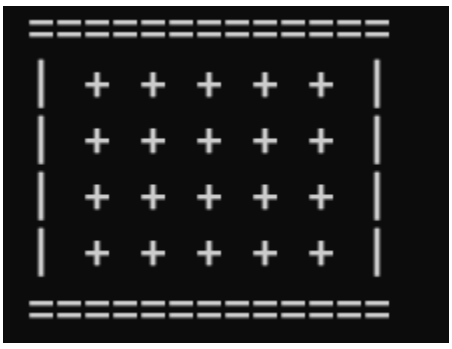
Width scaling: Each cell's character is printed `scale_width` times horizontally **Height scaling:** Each row is printed `scale_height` times vertically

The border characters (= and |) must also scale appropriately

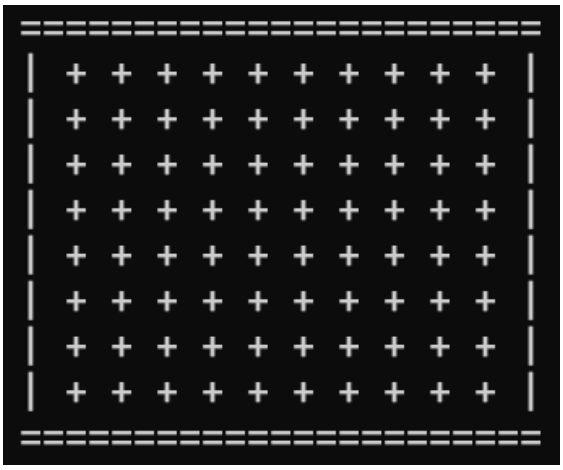
Max supportable scaling is 3x3.

Example

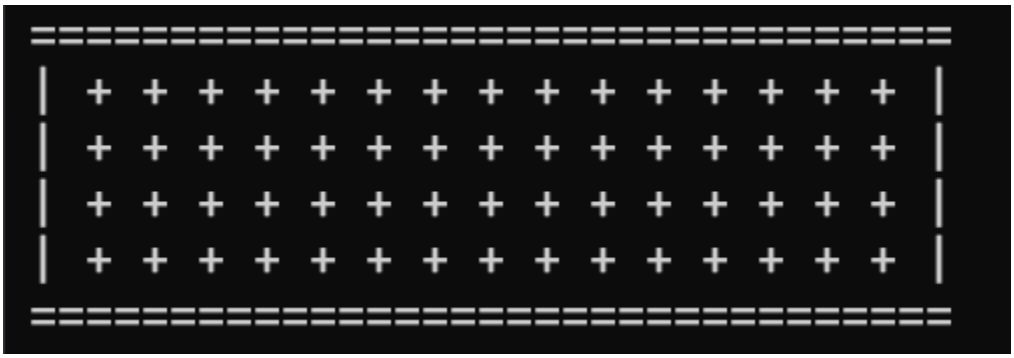
Original 1×1 output:



With 2×2 scaling:



With 3×1 scaling :



Good luck, you'll need it!