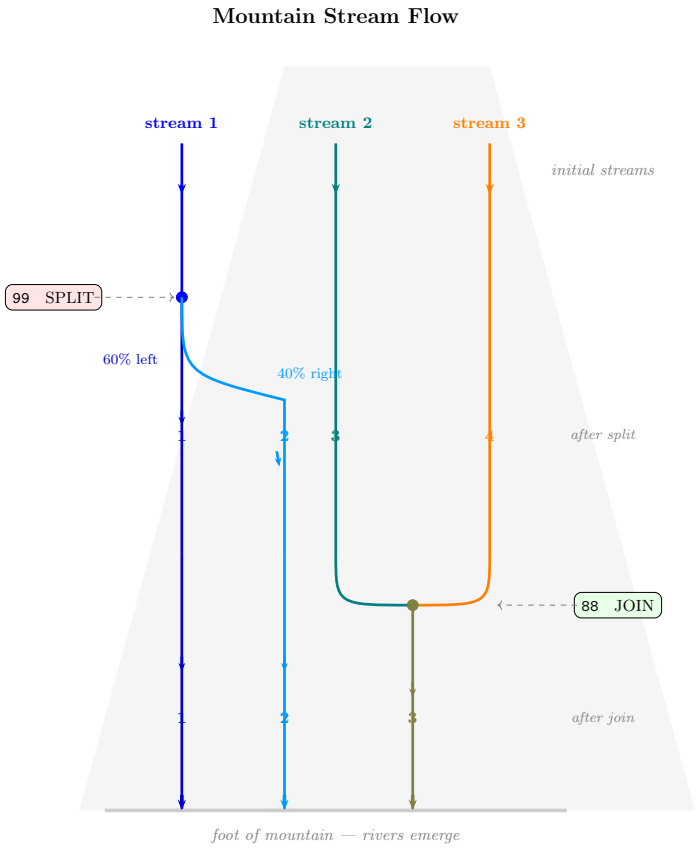
Programming 2: Lab 04

Problem Solving Skills

A Walkthrough Tutorial



Comp 111 — Forman Christian College

Spring 2026

Estimated Time: ∼**3 hours**

How to Use This Lab

This lab is a **walkthrough tutorial**—it is designed to guide you through learning, not to test you. This lab guides you through a structured problem-solving process. You will build a solution in stages, each focusing on a single idea. The goal is to reduce confusion by making each step explicit and testable. Work through each section at your own pace:

- **Read** the short explanations and study the diagrams.
- **Type** every code listing yourself (don't copy-paste!). Muscle memory matters.
- **Run** every example and check that your output matches.
- **Complete** the exercises. They are graded by difficulty:
  - ★ Easy — Immediate practice. Follow the pattern you just saw.
  - ★★ Medium — Requires some thinking. Combines concepts.
  - ★★★ Hard — Stretch goal. It's OK to need help!
- **Ask for help** whenever you're stuck for more than 5 minutes.

Before You Begin

This lab assumes you completed Labs 01–03. You should be comfortable reading problem statements, tracing loops, and running Python files from the **Command Prompt**.

Babbling Brooks (CCC 00 S2)

First read and attempt the problem on your own. Then follow the stages below to build a complete solution. The full problem statement is available at:

https://dmoj.ca/problem/ccc00s2/pdf

At any height on a mountain, there are m streams labeled left-to-right. As water flows down, streams can split or join. You must track the flow of each stream after every change and output the final river flows (rounded to the nearest integer).

Input Codes

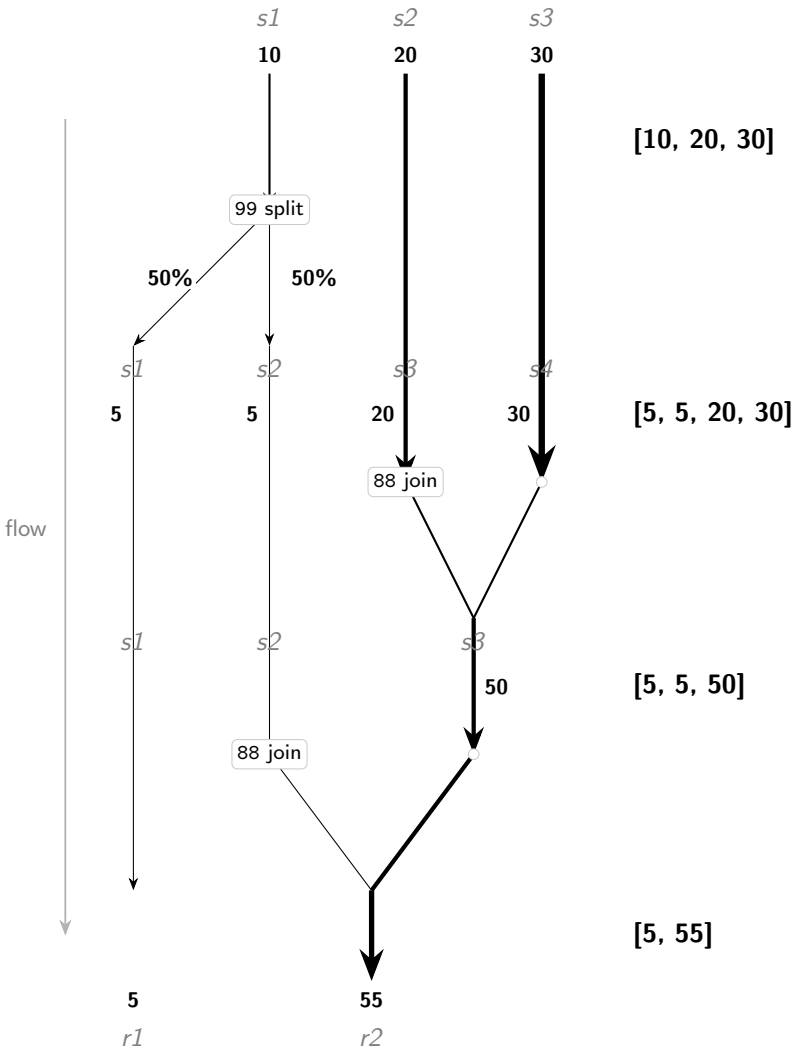
- `99`: split a stream into two forks.
- `88`: join a stream with the one immediately to its right.
- `77`: end of input.

Key Ideas

- The streams are always ordered left-to-right in a list.
- After any split or join, streams are renumbered in order.
- Each command modifies the list by inserting, deleting, or updating values.

Stage 0: Visual Model and Vocabulary

Goal: Understand splits, joins, and renumbering from a picture.



Check yourself: If stream 2 splits, which streams change labels and why?

Micro Exercise

1. ★ Easy A system has 3 streams. Stream 2 splits once. How many streams are there afterward?

Stage 1: Choose a Representation

Goal: Represent the current flows in left-to-right order.

Idea: Use a list of real numbers, where index 0 is stream 1.

Check yourself: If you store streams as `flows = [10, 20, 30]`, what is the flow of stream 2?

Micro Exercise

1. ★ Easy If `flows = [8, 3, 11]`, what is the flow of stream 3?

Stage 2: Simulate a Split

Goal: Replace one stream with two streams based on a percentage.

Prompt: Given stream k and percent p, split it into:

$$\text{left} = \text{flow} \cdot \frac{p}{100}, \quad \text{right} = \text{flow} \cdot \frac{100 - p}{100}$$

Check yourself: If stream 1 has flow 50 and $p = 60$, what are the left and right flows?

Micro Exercise

1. ★★ Medium Stream 2 has flow 90 and splits with $p = 20$. What are the left and right flows?

Stage 3: Simulate a Join

Goal: Merge two adjacent streams into one.

Prompt: Given stream k, replace stream k with the sum of streams k and $k + 1$, then remove stream $k + 1$.

Check yourself: If flows are `[5, 8, 9]` and you join stream 2 with stream 3, what is the new list?

Micro Exercise

1. ★★ Medium Flows are `[6, 12, 5]`. Join stream 2 with stream 3. What is the new list?

Stage 4: Input Loop

Goal: Read commands until `77` and update the list each time.

Prompt: Read a code. If it is `99`, read two more lines (stream index and percent). If it is `88`, read one more line (stream index). If it is `77`, stop.

Check yourself: Why must you renumber after every operation?

Micro Exercise

1. ★★ Medium If the next code is `99`, how many additional input lines do you read, and what do they represent?

Stage 5: Output Formatting

Goal: Print the final list of flows rounded to the nearest integer.

Prompt: Output flows for rivers 1 through m as space-separated integers.

Check yourself: Why is rounding safe here? (Hint: all flows are non-negative.)

Micro Exercise

1. ★ Easy Final flows are `[4.4, 7.6]`. What exact output line should be printed?

Stage 6: Edge Cases

Goal: Avoid common mistakes.

- Split percent can be 0 or 100.
- Join can happen near the end (k = m-1).
- Input stream numbers are 1-based.

Check yourself: What happens if you split a stream with 0 percent to the left?

Micro Exercise

1. ★★★ Hard Flows are `[12, 8, 4]`. Split stream 1 at 0%, then join stream 2 with stream 3. What final list do you get?

Stage 7: Consolidation

Goal: Combine all stages into a full solution.

Sample Input

```
3
10
20
30
99
1
50
88
3
88
2
77
```

Sample Output

```
5 55
```

Final Check: Can you trace each operation and predict the output before running code?

Micro Exercise

1. ★★★ Hard Start with flows `[10, 20]`. Split stream 1 at 50%, then join stream 2 with stream 3. What output line should print?

Spiral Numbers (CCC 2001 J4 / S2)

First read and attempt the problem on your own. Then follow the stages below to build a complete solution. The full problem statement is available at:
https://dmoj.ca/problem/ccc01s2/pdf

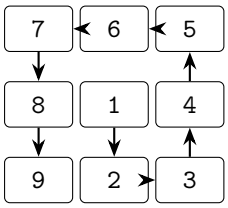
You are given two integers x and y ($1 \leq x \leq y < 100$). Print all numbers from x to y in a counter-clockwise spiral. The spiral starts with x in the center, and the next number is placed **below** it.

Key Ideas

- The spiral is a walk on a grid: each number has a row and column.
- Directions repeat in a cycle: down → right → up → left.
- Step lengths grow as $1, 1, 2, 2, 3, 3, \ldots$.
- Track **min/max row/col** so you can print a tight rectangle.
- Output uses width 2 for each number; blank cells are spaces.

Stage 0: Visual Model and Vocabulary

Goal: See how numbers grow in a spiral.



Check yourself: Which direction comes after moving up?

Micro Exercise

1. ★ Easy If $x = 10$, where is 11 placed relative to 10?

Stage 1: Choose a Coordinate Representation

Goal: Represent each number with a grid position.

Idea: Use `(row, col)` with $(0, 0)$ as the center. Store placements in a dictionary: `positions[(row, col)] = value`.

Check yourself: If you move down from $(0, 0)$, what is the new coordinate?

Micro Exercise

1. ★ Easy From $(0, 0)$, move left once. What is the new (row, col)?

Stage 2: Direction Cycle and Step Lengths

Goal: Walk in the correct spiral pattern.

Idea: Directions cycle in this order:

```
dirs = [(1,0), (0,1), (-1,0), (0,-1)]
         # down   right    up      left
```

The number of steps increases as $1, 1, 2, 2, 3, 3, \ldots$. After every two turns, increase the step length by 1.

Check yourself: How many steps do you take on the third direction in the cycle?

Micro Exercise

1. ★★ Medium The step lengths go $1, 1, 2, 2, 3, 3, \ldots$. What length is used on the 6th direction?

Stage 3: Walk and Place Numbers

Goal: Fill the spiral from x to y.

Prompt: Start at $(0, 0)$ with value x. For each step, move in the current direction, increase the value by 1, and record it in the dictionary. Stop when you place y.

Check yourself: Why do you need a loop that can stop early, even if the spiral ring is not complete?

Micro Exercise

1. ★★ Medium Start at $(0, 0)$ with $x = 5$. After moving down then right, where is 7 placed?

Stage 4: Track Bounds for Printing

Goal: Know how large the output rectangle should be.

Prompt: While walking, update `min_row`, `max_row`, `min_col`, and `max_col`. These bounds define the rectangle you will print.

Check yourself: If your smallest row is -2 and largest row is 1, how many rows will you print?

Micro Exercise

1. ★★ Medium If rows span -1 to 2 and columns span -2 to 1, how many rows and columns do you print?

Stage 5: Output Formatting

Goal: Print the rectangle with correct spacing.

Idea: Each cell is width 2. If a cell has a number, print it right-aligned in width 2. If it is empty, print two spaces. Separate cells with a single space.

Check yourself: What should you print for an empty cell?

Micro Exercise

1. ★ Easy What fixed width should each printed number occupy?

Stage 6: Consolidation

Goal: Combine all stages into a full solution.

Sample Input

```
10
27
```

Sample Output

```
      27 26
16 15 14 25
17 10 13 24
18 11 12 23
19 20 21 22
```

Final Check: Can you trace the first 8 moves and predict the locations of 10 through 17 before running your program?

Micro Exercise

1. ★★★ Hard For $x = 3$ and $y = 7$, list the coordinates of 3 through 7.

Snakes and Ladders (CCC 2003 J3 / S1)

First read and attempt the problem on your own. Then follow the stages below to build a complete solution. The full problem statement is available at:
https://dmoj.ca/problem/ccc03s1/pdf

You will simulate a single player moving on a Snakes and Ladders board. The piece starts on square 1. Each input is a dice total from 2 to 12. After each move, report the square where the piece lands. If the roll would move past 100, the piece does not move. If the piece lands on the bottom of a ladder, it climbs. If it lands on the top of a snake, it slides. Input 0 ends the game.

Key Ideas

- Track exactly one integer position (the current square).
- Apply the overshoot rule before snakes or ladders.
- Use a fixed mapping for the 6 special squares.
- The program ends only on input 0 or when you reach square 100.

Stage 0: Visual Model and Vocabulary

Goal: Connect the board picture to the rules in the prompt.

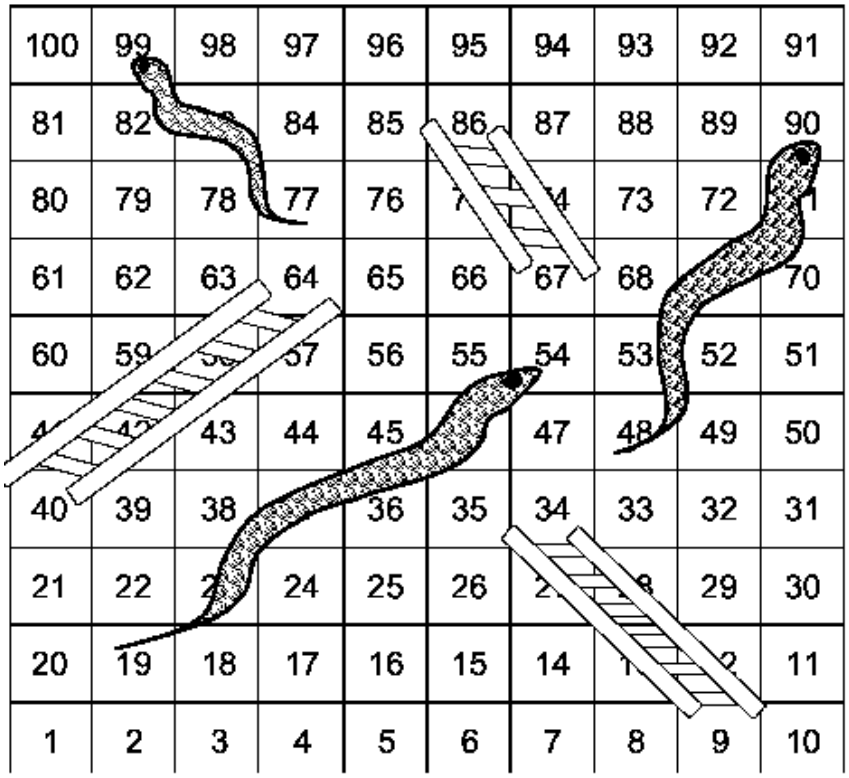


Figure 1: Snakes and Ladders board used in this problem.

Check yourself: Which squares are the bottom of ladders, and which are the top of snakes?

Micro Exercise

1. ★ Easy From the board image, name one ladder bottom and one snake head.

Stage 1: Track the Current Square

Goal: Store the player position as one integer.

Idea: Use `pos = 1` when the program starts. After each valid roll, compute the next square and update `pos`.

Check yourself: If `pos = 98` and the roll is 4, does the piece move?

Micro Exercise

1. ★ Easy If `pos = 1` and the roll is 8, what is the tentative square?

Stage 2: Apply the Overshoot Rule

Goal: Prevent moves that go beyond square 100.

Prompt: If `pos + roll > 100`, then the piece stays where it is. Otherwise, move forward.

Check yourself: If `pos = 96` and the roll is 4, what happens?

Micro Exercise

1. ★★ Medium If `pos = 98` and the roll is 5, what happens?

Stage 3: Apply Snakes and Ladders

Goal: Move the piece after landing on a special square.

Prompt: After a valid move, if `pos` matches a ladder bottom or snake head, replace it using this mapping:

- Ladders: $9 \to 34$, $40 \to 64$, $67 \to 86$
- Snakes: $54 \to 19$, $90 \to 48$, $99 \to 77$

Check yourself: If you land on 67, what square should you report?

Micro Exercise

1. ★ Easy If you land on 90, what square should you report?

Stage 4: Input Loop and Quit Condition

Goal: Keep reading dice totals until the user quits.

Prompt: Repeatedly read an integer. If it is 0, print `You Quit!` and stop. Otherwise apply the move rules and print:

```
You are now on square X
```

Check yourself: Should you print a line after the user enters 0?

Micro Exercise

1. ★★ Medium Input sequence is `4, 0`. What lines are printed?

Stage 5: Win Condition

Goal: Detect when the player reaches square 100.

Prompt: After applying a valid move and any snake/ladder jump, if `pos == 100`, print `You Win!` and stop.

Check yourself: In what order do you check: overshoot, snake/ladder, win?

Micro Exercise

1. ★★ Medium When `pos` becomes 100, what exact line is printed and what happens next?

Stage 6: Edge Cases

Goal: Avoid common mistakes.

- Do not move if the roll overshoots 100.
- Only apply a snake or ladder once per roll.
- Input is guaranteed to be 0 or 2–12.

Check yourself: Why do you not apply snakes or ladders when a move is blocked?

Micro Exercise

1. ★★★ Hard If a roll would overshoot 100 onto a ladder square, do you move? Explain in one sentence.

Stage 7: Consolidation

Goal: Combine all stages into a full solution.

Sample Input

```
9
11
12
7
3
5
10
9
```

Sample Output

```
You are now on square 10
You are now on square 21
You are now on square 33
You are now on square 64
You are now on square 86
```

```
You are now on square 91
You are now on square 91
You are now on square 100
You Win!
```

Final Check: Can you trace each roll and explain every square change before running your program?

Micro Exercise

1. ★★★ Hard Starting at square 1, trace rolls `9, 11, 12`. List each reported square.

Mouse Move (CCC 2005 S2)

First read and attempt the problem on your own. Then follow the stages below to build a complete solution. The full problem statement is available at:
https://dmoj.ca/problem/ccc05s2/pdf

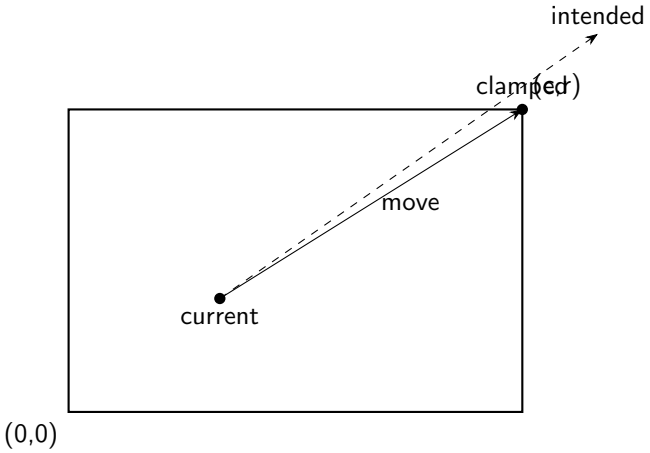
You are given a screen with corners $(0, 0)$ and (c, r). The mouse starts at $(0, 0)$. Each input pair (a, b) is a relative move. Update the cursor position after each move, but clamp it to the screen boundaries. Stop when the move is $(0, 0)$.

Key Ideas

- The mouse move is relative, not absolute.
- The cursor cannot leave the screen: clamp each coordinate to $[0, c]$ and $[0, r]$.
- The first input line is c r; all later lines are moves until 0 0.

Stage 0: Visual Model and Vocabulary

Goal: See a relative move that tries to go off-screen.



Check yourself: If the current position is $(2, 1)$ and the move is $(10, -3)$, where does it end up?

Micro Exercise

1. ★ Easy Screen is $(c, r) = (5, 3)$. Current $(2, 2)$. Move $(4, 1)$. Where does it end?

Stage 1: Track the Current Position

Goal: Store the cursor position as two integers.

Idea: Use `x` and `y`. Start with `x = 0`, `y = 0`.

Check yourself: What should `x` and `y` be right after reading c r?

Micro Exercise

1. ★ Easy What are `x` and `y` before any moves are applied?

Stage 2: Apply a Relative Move

Goal: Compute the tentative new position.

Prompt: For each move (a, b), update `x += a` and `y += b`.

Check yourself: If `(x,y) = (5, 7)` and the move is $(-2, 4)$, what is the tentative position?

Micro Exercise

1. ★★ Medium If `(x,y) = (3,2)` and the move is $(-4, 5)$, what is the tentative position?

Stage 3: Clamp to the Screen

Goal: Keep the cursor inside the rectangle.

Prompt: After applying a move, clamp each coordinate:

$$x = \max(0, \min(x, c)), \quad y = \max(0, \min(y, r)).$$

Check yourself: If `x = -3` and `y = 50` on a screen $(c, r) = (30, 40)$, what are the clamped values?

Micro Exercise

1. ★★ Medium With $(c, r) = (10, 6)$ and tentative $(12, -1)$, what is the clamped position?

Stage 4: Input Loop and Sentinel

Goal: Process moves until $(0, 0)$ ends the input.

Prompt: Read (a, b). If it is $(0, 0)$, stop. Otherwise update, clamp, and print the position.

Check yourself: Should you print anything after reading $(0, 0)$?

Micro Exercise

1. ★★ Medium Screen $(c, r) = (5, 5)$. Moves are $(2, 0)$ then $(0, 0)$. What outputs appear?

Stage 5: Output Format

Goal: Match the required output exactly.

Prompt: After each move (except $(0, 0)$), print:

```
x y
```

Check yourself: Is there any extra text like `Position:` in the output?

Micro Exercise

1. ★ Easy If the current position is $(7, 4)$, what exact line should be printed?

Stage 6: Edge Cases

Goal: Avoid common mistakes.

- Moves can be negative or zero.
- The cursor can hit one boundary but still move in the other direction.
- The screen size can be zero in one direction (e.g., $c = 0$ or $r = 0$).

Check yourself: If $c = 0$, what x-values are possible?

Micro Exercise

1. ★★★ Hard If $c = 0$, what x-values are possible after any move?

Stage 7: Consolidation

Goal: Combine all stages into a full solution.

Sample Input 1

```
100 200
10 40
-5 15
30 -30
0 0
```

Sample Output 1

```
10 40
5 55
35 25
```

Sample Input 2

```
30 40
30 40
-100 -10
0 0
```

Sample Output 2

```
30 40
0 30
```

Final Check: Can you trace each move and predict the printed positions before running your program?

Micro Exercise

1. ★★★ Hard Screen $(c, r) = (5, 5)$. Moves: $(3, 4)$, $(-10, 0)$, $(2, -7)$, $(0, 0)$. List every printed output.

1 CCC Practice Problems

Each problem is on the following pages. Read the problem statement carefully, then write your Python solution in Thonny. **Then submit your solution to the DMOJ online judge to see if it passes all test cases.**

CCC '04 S3 - Spreadsheet

Time limit: 1.0s **Memory limit:** 256M

Canadian Computing Competition: 2004 Stage 1, Senior #3

A spreadsheet consists of a number of "cells" set in a rectangular grid. Each is addressed with one letter from A to J (giving the row) and one number from 1 to 9 (giving the column). Thus the upper-left cell of the spreadsheet is $A1$, and the lower-right cell is $J9$.

Every cell has a value, and this can be specified in one of two ways:

1. as an integer from 0 to 1000
2. as a sum of the values of up to 10 other cells

Cell values may be interdependent (that is, $A1$'s sum may depend on $B6$ which depends on $C9$), but a cell whose value depends on itself, directly or indirectly, is undefined (e.g., $A1$ depending on $G8$ depending on $A1$). Also, a cell whose value depends on an undefined cell is itself undefined. You are given the specification for all the cells in the spreadsheet. Compute and output the values of all of the cells.

Input Specification

Input will consist of 10 lines, one per spreadsheet row. Each line contains 9 descriptions of a cell, which will be either an integer between 0 and $1\,000$, or a sum of 1 to 10 distinct cell names separated by a `+` symbol (e.g. `A1+B5+D3`).

Output Specification

Output 10 lines, with 9 numbers per line, giving the value of every cell in the spreadsheet; if the cell is undefined, print an asterisk (`*`) in place of its value. No cell's final value will exceed $1\,000\,000\,000$.

Sample Input

```
1 2 3 A1+A2+A3 A3+A4 A1+A4+A5 A8+A9 A9 A8
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
```

Sample Output

```
1 2 3 6 9 16 * * *
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
```

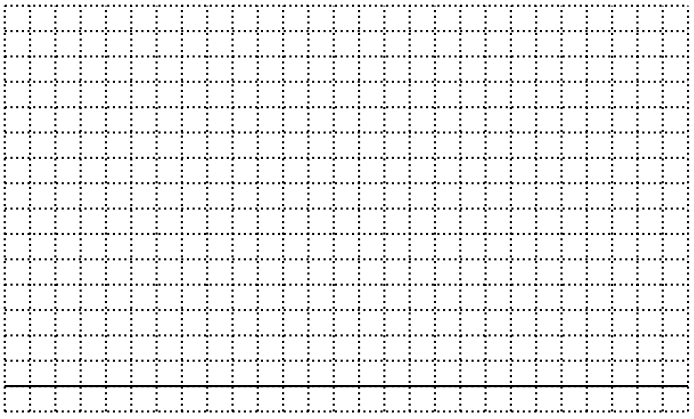
CCC '04 J5 - Fractals

Time limit: 1.0s **Memory limit:** 256M

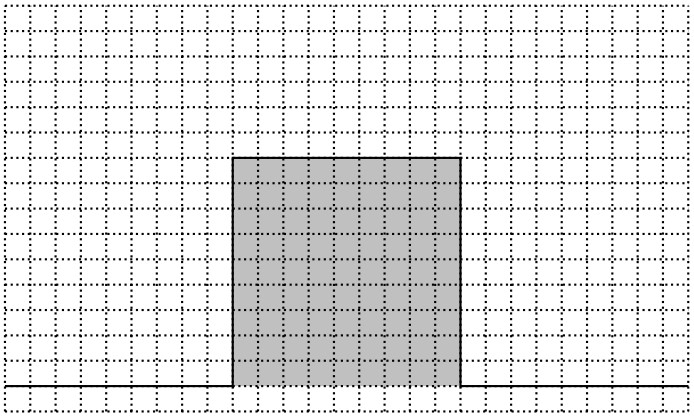
Canadian Computing Competition: 2004 Stage 1, Junior #5

A fractal is a geometric shape where the pattern of the whole shape is self-replicating at each subsection of the shape. A simple "block fractal" is shown below. At each stage of the fractal growth, every straight line in the fractal is divided into three equal parts. The first and last sections stay straight; the middle section contains a square "bump" which has the same height as the width of the middle section. (You will want to consider the four orientations of a line segment within the fractal. Depending upon which line segment is currently being generated, the bump may protrude up, down, left, or right.)

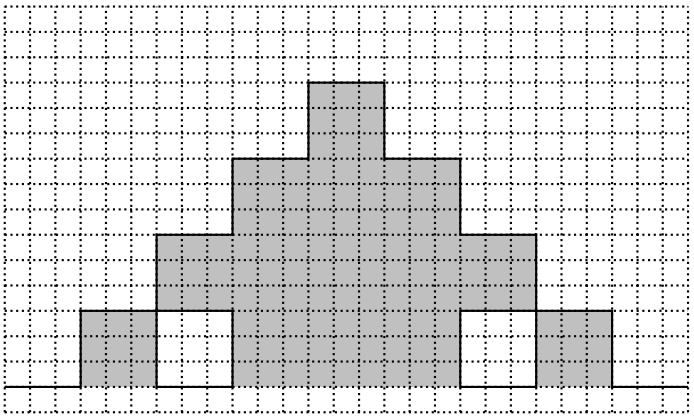
Level 0 of the Fractal



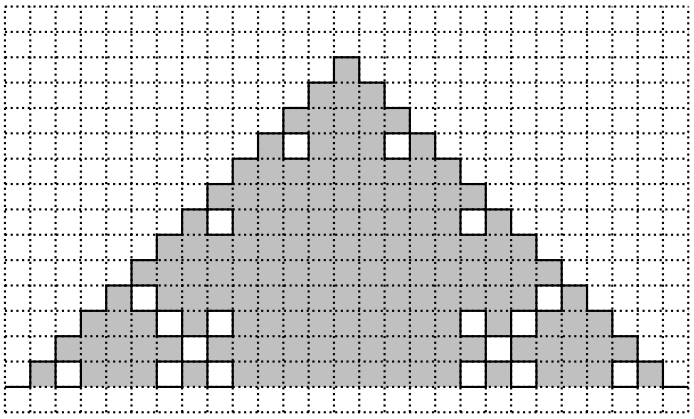
Level 1 of the Fractal



Level 2 of the Fractal



Level 3 of the Fractal



Suppose this fractal is drawn on a Cartesian plane, where $(0, 0)$ is at the bottom left corner. Assume that in the example above, the bottom left point of the fractal is at $(0, 1)$ and the bottom right point of the fractal is at $(27, 1)$. For example the top of the Level 3 fractal is a line from $(13, 14)$ to $(14, 14)$.

Write a program that will keep track of the integer coordinate points of the lines in a similar "block fractal" with its bottom left corner at $(0, 1)$. The program will accept three integers as input: the level of the fractal, the width of the fractal, and an x-coordinate. You may assume that the width of the fractal will be some power of three, and that it will be large enough so that every corner of the fractal will fall on an integer intersection in the Cartesian plane. The width will never be more than 81. The x-coordinate, x, will be in the range 0 – width (inclusive). Your program should output the y-coordinate value(s), y, where lines of the fractal intersect the point (x, y).

You may draw a graphic representation of the fractal for debugging (and interest). However, test cases may ask you to define fractals that are too large to fit on a single screen.

Sample Input 1

```
3 27 5
```

Sample Output 1

```
4 5 6
```

Sample Input 2

```
3 27 18
```

Sample Output 2

```
1 2 3 4 7 8 9 10
```

Sample Input 3

```
2 27 19
```

Sample Output 3

```
1 4 7
```

Sample Input 4

```
4 81 38
```

Sample Output 4

```
37 38 39
```

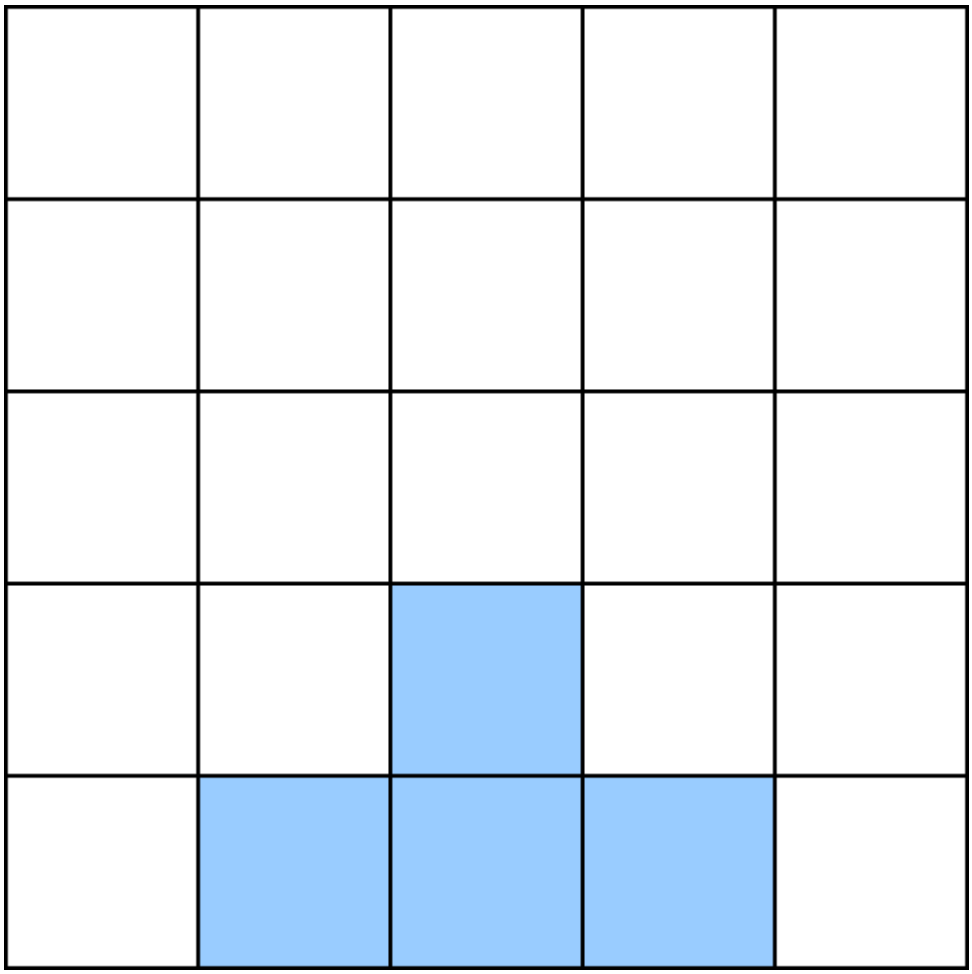
CCC '11 S3 - Alice Through the Looking Glass

Time limit: 1.0s **Memory limit:** 256M

Canadian Computing Competition: 2011 Stage 1, Senior #3

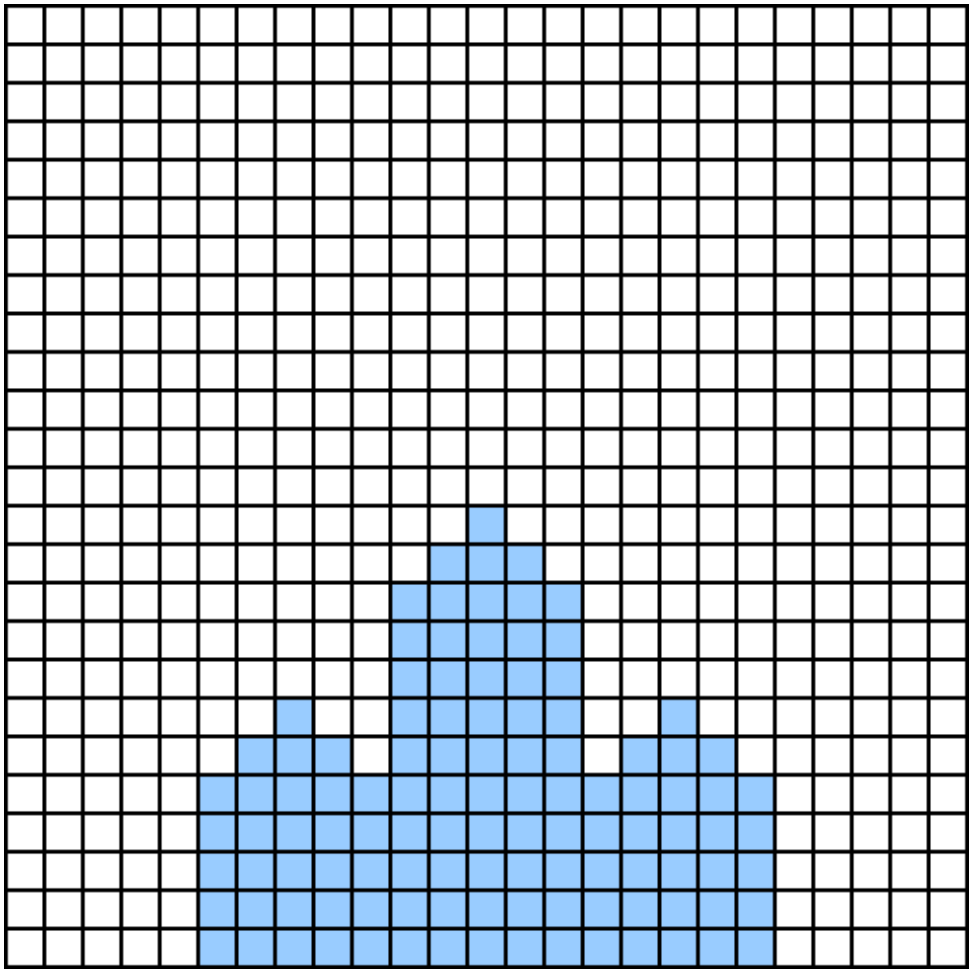
Alice is looking at a crystal through a microscope. Alice's microscope has the interesting feature that it can superimpose grid lines over the image that she is looking at.

At level 1 of magnification, Alice sees the image as follows:



Notice that at level 1, there is a 5×5 grid superimposed over the image.

However, as Alice increases the magnification, the leaf pattern becomes more intricate.



At level 2 of the magnification, Alice sees the image with a 25×25 grid, and notices that three of the four larger squares in the original image have the small four square pattern on top. In fact, for this particular crystal, this self-similarity repeats for each magnification level.

Given that Alice's microscope has up to 13 levels of magnification, she would like to try to quantify the detail of each grid cell at every one of these magnification levels.

Specifically, since there is a $5^m \times 5^m$ grid at magnification level m, Alice will call the bottom-left corner grid cell $(0, 0)$, the bottom-right grid cell $(5^m - 1, 0)$, the top-left grid cell $(0, 5^m - 1)$, and the top-right grid cell $(5^m - 1, 5^m - 1)$.

Given an integer magnification level m ($1 \leq m \leq 13$) and a grid position (x, y) (where $0 \leq x < 5^m$ and $0 \leq y < 5^m$), Alice would like to know if her crystal will fill that grid cell, or if that grid cell will be empty space.

Input Specification

The first line of input will be T ($0 < T \leq 10$) which is the number of test cases. On each of the next T lines there will be three integers: m, the magnification level, followed by x and y, the position of the grid cell that Alice wishes to examine.

Output Specification

The output will be T lines. Each line of output will either be `empty`, if the specified grid cell is empty, or `crystal` if that grid cell contains crystal.

Sample Input

```
4
1 1 1
1 1 0
1 2 1
2 8 5
```

Output for Sample Input

```
empty
crystal
crystal
crystal
```

Note: At least 40% of the test cases will have $m \leq 4$.