LeetCode Problem: Triangle Pattern Generator

Problem Statement

Given an integer `n`, implement three functions to print different triangle patterns:

1. **Right-Aligned Triangle**: Triangle aligned to the right with `~` as padding and `*` as the pattern
2. **Left-Aligned Triangle**: Simple left-aligned triangle with `*` characters
3. **Hollow Left-Aligned Triangle**: Left-aligned triangle that is hollow inside (borders are `*`, interior is `.`)

Examples

Example 1:

```
Input: n = 4

Right-Aligned Triangle:
~~~*
~~**
~***
****

Left-Aligned Triangle:
*
**
***
****

Hollow Left-Aligned Triangle:
*
**
*.*
****
```

Example 2:

```
Input: n = 3

Right-Aligned Triangle:
~~*
~**
***

Left-Aligned Triangle:
*
**
***

Hollow Left-Aligned Triangle:
*
**
***
```

Example 3:

```
Input: n = 5

Right-Aligned Triangle:
~~~~*
~~~**
~~***
~****
*****

Left-Aligned Triangle:
*
**
***
****
*****

Hollow Left-Aligned Triangle:
*
**
*.*
*..*
*****
```

Constraints

- `1 ≤ n ≤ 100`
- Print each pattern with appropriate spacing and newlines
- Use `*` for the main pattern, `~` for right-alignment padding, and `.` for hollow interior

---

Solution

```
c
```

```c
#include <stdio.h>

void printRightAlignedTriangle(int n) {
    for (int i = n; i > 0; i--) {
        for (int j = i - 1; j > 0; j--)
            printf("~");
        for (int k = i - (i - 1); k <= n - (i - 1); k++)
            printf("*");
        printf("\n");
    }
}

void printLeftAlignedTriangle(int n) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j <= i; j++) {
            printf("*");
        }
        printf("\n");
    }
}

void printHollowLeftTriangle(int n) {
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= i; j++) {
            if (j == 1 || j == i || i == 1 || i == n)
                printf("*");
            else
                printf(".");
        }
        printf("\n");
    }
}

int main() {
    int n;
    scanf("%d", &n);

    printf("Right-Aligned Triangle:\n");
    printRightAlignedTriangle(n);

    printf("\nLeft-Aligned Triangle:\n");
    printLeftAlignedTriangle(n);
    printf("\nHollow Left-Aligned Triangle:\n");
    printHollowLeftTriangle(n);

    return 0;
}
```

Algorithm Explanation

1. Right-Aligned Triangle (`printRightAlignedTriangle`)

Logic:

- Loop from `n` down to `1` (variable `i`)
- For each row `i`, print `(i-1)` padding characters (`~`)
- Then print `(n-i+1)` star characters (`*`)

Key insight: The number of padding characters decreases as we go down, while the number of stars increases.

Mathematical relationship:

- Row `i`: `(i-1)` tildes + `(n-i+1)` stars = `n` total characters

2. Left-Aligned Triangle (`printLeftAlignedTriangle`)

Logic:

- Loop from `0` to `n-1` (variable `i`)
- For each row `i`, print `(i+1)` star characters

Key insight: Simple incremental pattern where row number directly corresponds to star count.

3. Hollow Left-Aligned Triangle (`printHollowLeftTriangle`)

Logic:

- Loop from `1` to `n` (variable `i`)
- For each position `j` in row `i`:
  - Print `*` if it's the first position (`j == 1`)
  - Print `*` if it's the last position (`j == i`)
  - Print `*` if it's the first row (`i == 1`)
  - Print `*` if it's the last row (`i == n`)

- Otherwise, print `.` for hollow interior

Key insight: Border detection using conditional logic to create hollow effect.

Complexity Analysis

Time Complexity: O(n²)

- Each function contains nested loops
- Outer loop runs `n` times
- Inner loops run up to `n` times in total
- Overall: `O(n²)` for all three patterns combined

Space Complexity: O(1)

- Only using loop variables and constants
- No additional data structures
- Direct output to console

Test Cases

Test Case 1:

```
Input: 1
Expected Output:
Right-Aligned Triangle:
*


Left-Aligned Triangle:
*


Hollow Left-Aligned Triangle:
*
```

Test Case 2:

```
Input: 6
Expected Output:
Right-Aligned Triangle:
~~~~~*
~~~~**
~~~***
~~****
~*****
******

Left-Aligned Triangle:
*
**
***
****
*****
******

Hollow Left-Aligned Triangle:
*
**
*.*
*..*
*...*
******
```

Key Programming Concepts

1. **Nested Loop Patterns**: Understanding how to use multiple loops to create 2D patterns
2. **Mathematical Relationships**: Each pattern follows specific formulas relating row and column positions
3. **Conditional Logic**: Hollow triangle demonstrates border detection algorithms
4. **Character-by-Character Output**: Efficient printing without string concatenation
5. **Pattern Recognition**: Identifying mathematical sequences in visual patterns

---

Difficulty Level

Medium - Requires understanding of:

- Nested loop control

- Mathematical pattern recognition
- Conditional logic for complex shapes
- Index manipulation and boundary conditions

Topics

- Arrays & Strings
- Pattern Recognition
- Mathematical Logic
- Loop Control
- Conditional Statements

Follow-up Questions

1. How would you modify the code to use different characters for each pattern?
2. Can you create an inverted version of these triangles?
3. What changes would be needed to generate diamond patterns instead?
4. How would you optimize the solution for very large values of `n`?

Problem ID: 1234
Difficulty: Medium
Tags: Pattern, Loops, Mathematics