CODECHUM

National Programming Challenge 🏆🔥

Preliminary Programming Problems for Group Stage 3:

1. Alternating Character Pattern

Item Description:

You are given a string S. Your task is to determine if all the alphabetic characters within the string follow a strictly alternating pattern of vowels and consonants (or consonants and vowels). Non-alphabetic characters should be ignored when checking the pattern, but they are part of the input string.

The alternating pattern means that if an alphabetic character is a vowel, the next alphabetic character must be a consonant, and vice versa. The pattern can start with either a vowel or a consonant. Case-insensitivity should be applied when determining if a character is a vowel or a consonant. If the string contains no alphabetic characters or only one alphabetic character, it is considered to follow the alternating pattern.

A character is considered a vowel if it is 'a', 'e', 'i', 'o', or 'u' (case-insensitive). All other alphabetic characters are consonants.

Input:

A single line containing the string S. The string may contain spaces and special characters.

Constraints:

- `1 ≤ length(S) ≤ 1000`

Output:

Print "Alternating" if the alphabetic characters follow the alternating pattern, otherwise print "Not Alternating".

Sample Output 1:

```
Enter string: bAnAnA
Result: Alternating
```

Sample Output 2:

```
Enter string: apple
Result: Not Alternating
```

Sample Output 3:

```
Enter string: h3ll0 W0rld!
Result: Not Alternating
```

2. Peak-Valley Difference

Item Description:

You are given a list of numbers. Your task is to identify all the **peaks** and **valleys** in this list.

A **peak** is a number that is strictly greater than both its immediate neighbors (the number to its left and the number to its right).

A **valley** is a number that is strictly smaller than both its immediate neighbors. The first and last numbers in the list cannot be peaks or valleys as they only have one neighbor.

After identifying all peaks and valleys, calculate the sum of all peaks and the sum of all valleys. The final result is the difference between the total sum of the peaks and the total sum of the valleys (`sum_of_peaks - sum_of_valleys`).

Input:

The first line contains an integer `N` (`3 ≤ N ≤ 100`), representing the number of elements in the array.

The second line contains `N` space-separated integers `arr[i]` (`1 ≤ arr[i] ≤ 1000`), representing the elements of the array.

Output:

A single integer representing the final calculated result (`sum_of_peaks - sum_of_valleys`).

Sample Output 1:

```
Enter array size: 7
Enter array elements: 10 50 20 80 30 90 40
Result: 170
```

Sample Output 2:

```
Enter array size: 5
Enter array elements: 1 2 3 4 5
Result: 0
```

Sample Output 3:

```
Enter array size: 5
Enter array elements: 5 4 3 2 1
Result: 0
```

3. Palindromic Prime Counter

Item Description:

A prime number is a natural number greater than 1 that has no positive divisors other than 1 and itself. A palindromic number is a number that reads the same forwards and backward (e.g., 121, 5, 33). Your task is to count how many numbers in a given range `[L, R]` (inclusive) are both prime and palindromic.

Input:

Two integers, `L` and `R`, separated by a space, representing the lower and upper bounds of the range respectively. Constraints: `1 ≤ L ≤ R ≤ 100000`.

Output:

Print the total count of palindromic prime numbers within the specified range. The output should be labeled as `Count: `.

Sample Output 1:

```
Enter L and R: 1 10
Count: 4
```

Sample Output 2:

```
Enter L and R: 11 20
Count: 1
```

Sample Output 3:

```
Enter L and R: 100 200
Count: 5
```

4. Digit Product Sum Series

Item Description:

You are given a positive integer `N`. We define a sequence `A` based on `N` as follows:

- `A[0] = N`.
- For `k > 0`, `A[k]` is calculated as the sum of all unique products of distinct digit pairs from `A[k-1]`.

The sequence terminates when `A[k]` becomes a single-digit number (i.e., a number between 0 and 9 inclusive). Your task is to find the total sum of all numbers in this sequence `A`, including the initial `N` and the final single-digit number.

For example, if `N = 234`:

- `A[0] = 234:`
    - Its distinct digits are 2, 3, 4.
    - The unique products of distinct digit pairs are: `2*3=6`, `2*4=8`, `3*4=12`.
    - The sum of these products is `6 + 8 + 12 = 26`. So, `A[1] = 26`.
- `A[1] = 26:`
    - Its distinct digits are 2, 6.
    - The unique product of distinct digit pairs is: `2*6=12`.
    - The sum of this product is `12`. So, `A[2] = 12`.
- `A[2] = 12:`
    - Its distinct digits are 1, 2.
    - The unique product of distinct digit pairs is: `1*2=2`.
    - The sum of this product is `2`. So, `A[3] = 2`.
- `A[3] = 2:`
    - is a single-digit number, so the sequence terminates.

The sequence is `[234, 26, 12, 2]`. The total sum is `234 + 26 + 12 + 2 = 274`.

Input:

A single positive integer `N`.

Constraints:

- 1 ≤ N ≤ 10

Output:

The total sum of all numbers in the generated sequence.

Sample Output 1:

```
Enter N: 234
Total sum of sequence: 274
```

Sample Output 2:

```
Enter N: 12
Total sum of sequence: 14
```

Sample Output 3:

```
Enter N: 10
Total sum of sequence: 10
```

5. Cyclic Array Shift

Item Description:

You are given an array `A` of `N` integers. Your task is to perform a cyclic shift on this array. A cyclic shift involves moving elements to new positions such that elements 'wrap around' when they reach the end or beginning of the array.

You will be given the size of the array `N`, the number of positions `K` to shift, and the direction of the shift ('L' for left, 'R' for right).

A **left cyclic shift** by `K` positions moves the first `K` elements to the end of the array, and the remaining elements shift left. For example, if `A = [1, 2, 3, 4, 5]` and `K = 2`, a left shift results in `[3, 4, 5, 1, 2]`.

A **right cyclic shift** by `K` positions moves the last `K` elements to the beginning of the array, and the remaining elements shift right. For example, if `A = [1, 2, 3, 4, 5]` and `K = 2`, a right shift results in `[4, 5, 1, 2, 3]`.

Note that `K` can be larger than `N`. In such cases, the effective number of shifts is `K % N`, as shifting by `N` positions brings the array back to its original state.

Input:

- The first line contains an integer `N` ($1 \le N \le 10$), representing the size of the array.
- The second line contains a long long integer `K` ($0 \le K \le 10$), representing the number of positions to shift.
- The third line contains a character, 'L' or 'R', indicating the direction of the shift (Left or Right).
- The fourth line contains `N` space-separated long long integers, representing the elements of the array `A` ($-10 \le A[i] \le 10$).

Output:

Print the elements of the cyclically shifted array, space-separated, followed by a newline.

Sample Output 1:

```
Enter array size: 5
Enter shift amount K: 2
Enter shift direction (L/R): L
Enter array elements: 1 2 3 4 5
Shifted array: 3 4 5 1 2
```

Sample Output 2:

```
Enter array size: 5
Enter shift amount K: 2
Enter shift direction (L/R): R
Enter array elements: 1 2 3 4 5
Shifted array: 4 5 1 2 3
```

Sample Output 3:

```
Enter array size: 7
Enter shift amount K: 10
Enter shift direction (L/R): L
Enter array elements: -5 0 12 34 -99 100 7
Shifted array: 34 -99 100 7 -5 0 12
```

6. Pascal's Pyramid

Item Description:

Pascal's Triangle is a triangular array of binomial coefficients. It starts with 1 at the top, and each number below it is the sum of the two numbers directly above it — one from the previous row in the **same column** and one from the **column to the left**. If there is only one number above, the number below it is the same as the one above. Conventionally, the triangle starts with row 0 (containing a single 1), but for this problem, we will consider the first row to be row 1, containing a single 1.

For example, for `N = 5`, the triangle looks like this:

```
1

1 1

1 2 1

1 3 3 1

1 4 6 4 1
```

Take the number **6** in the 5th row, 3rd column. It is calculated by adding the two numbers above it:

- **3** (from row 3, column 1)
- **3** (from row 3, column 2)

So:

- **6 = 3 + 3**

Your task is to generate and print Pascal's Triangle up to `N` rows, given an integer `N`. You must correctly implement the rules of Pascal's Triangle generation using iterative structures (nested loops) and manage dynamic memory for the triangle.

Input:

An integer `N`, representing the number of rows to generate.

- `1 ≤ N ≤ 30`

Output:

The Pascal's Triangle with `N` rows. Each row should be printed on a new line, and the numbers within each row should be separated by a single space.

Sample Output 1:

```
Enter the number of rows (N): 1
Pascal's Triangle with 1 rows:
1
```

Sample Output 2:

```
Enter the number of rows (N): 3
Pascal's Triangle with 3 rows:
1
1 1
1 2 1
```

Sample Output 3:

```
Enter the number of rows (N): 5
Pascal's Triangle with 5 rows:
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
```

7. Stable Sequence Reduction

Item Description:

In signal processing, a common task is to simplify a sequence of readings by identifying and merging stable patterns. A sequence is reduced by repeatedly finding adjacent elements whose values are very close and replacing them with their average. This process continues until the sequence becomes stable, meaning no more adjacent elements can be merged.

You are given a sequence of integers and a non-negative floating-point threshold `T`. Your task is to implement an algorithm that repeatedly scans the sequence and merges any adjacent pair of numbers `(a, b)` if their absolute difference `|a - b|` is less than or equal to `T`. The pair is replaced by their average `(a + b) / 2`. The process is repeated on the newly formed sequence until no more merges are possible in a full pass. A pass consists of a single scan and merge from left to right.

For example, if the sequence is `[10, 11, 13, 15]` and `T=2`:In the first pass, we check `(10, 11)`. `|10 - 11| = 1 <= 2`, so they merge into `10.5`. The sequence becomes `[10.5, 13, 15]`. The next check is on the rest of the original sequence, `(13, 15)`. `|13-15| = 2 <= 2`, so they merge to `14`. The new sequence after the first pass is `[10.5, 14]`.In the second pass, we check `(10.5, 14)`. `|10.5 - 14| = 3.5 > 2`. No merge occurs.Since no merges occurred in the second pass, the process stops and the final stable sequence is `[10.5, 14]`.

Input:

The first line contains an integer `N`, the number of elements in the sequence, and a floating-point number `T`, the stability threshold.

The second line contains `N` space-separated integers, representing the initial sequence `S`.

Constraints: `1 <= N <= 100`, `0 <= T <= 1000`, `0 <= S[i] <= 10000`.

Output:

Print the final stable sequence. The elements should be space-separated and formatted to two decimal places. The output should end with a newline.

Sample Output 1:

```
Enter the number of elements and the threshold: 4 2.0
Enter the sequence: 10 11 13 15
Final stable sequence: 10.50 14.00
```

Sample Output 2:

```
Enter the number of elements and the threshold: 5 3.0
Enter the sequence: 10 12 20 22 30
Final stable sequence: 11.00 21.00 30.00
```

Sample Output 3:

```
Enter the number of elements and the threshold: 4 5.0
Enter the sequence: 10 20 30 40
Final stable sequence: 10.00 20.00 30.00 40.00
```

8. Maximal Square of Ones

Item Description:

Given an `N × M` binary matrix (a matrix containing only 0s and 1s), your task is to find the side length of the largest square subgrid that is composed entirely of 1s.

For example, consider the matrix below:

```
1 0 1 0 0

1 0 1 1 1

1 1 1 1 1

1 0 0 1 0
```

In this matrix, the largest square subgrid of 1s has a side length of 2. One such square has its top-left corner at row 1, column 2 (0-indexed).

Input:

The first line of input contains two space-separated integers, `N` and `M` (`1 ≤ N, M ≤ 500`), representing the number of rows and columns in the matrix.

The next `N` lines each contain `M` space-separated integers (either 0 or 1), describing the rows of the matrix.

Output:

Print a single integer which is the side length of the largest square of 1s found in the matrix, followed by a newline. The output should be in the format: `Largest square side: [side]`

Sample Output 1:

```
Enter rows and cols: 4 5
Enter the matrix:
1 0 1 1 1
1 1 1 1 1
1 1 1 1 1
1 0 1 1 0
Largest square side: 3
```

Sample Output 2:

```
Enter rows and cols: 3 3
Enter the matrix:
0 0 0
0 0 0
0 0 0
Largest square side: 0
```

Sample Output 3:

```
Enter rows and cols: 1 5
Enter the matrix:
0 1 1 0 1
Largest square side: 1
```