INSTRUCTIONS :

1. Complete all questions in your designated project group.
2. All members must contribute to writing the codes. (i.e. 1 question = 1 person, and share the workload if there's an additional question relative to the actual number of members in your team (i.e. 5)). Ensure that all members must understand and explain codes from any of the questions.
3. During viva, all students in each team will be randomly asked to describe, answer and edit any of the answers provided. Marks will be given to your ability to present the answers.

Lab Report

Prepare a report to solve the above problems. The report should contain all the sections as below for each question:

Section	Description
1. Problem	Description on the problem
2. Solution	Explanation on how to solve the problems below
3. Sample Input & Output	A few sets of input and output (snapshot)
4. Source code	Java source code

Requirements

1. Group Assignment (Grouping is the same as your project group)
2. Cover page that includes all student matric number and full name.
3. Font: Times New Roman 12, Line Spacing: 1 ½ Spacing
4. Submit to Spectrum according to your OCC. **Deadline** : Before your viva session (W6).

Question 1: Digital Signal Processing

Problem statement:

You are working on a project related to digital signal processing. One requirement is to process a number and reduce it to its *digital root*, which means summing the digits of a number repeatedly until a single-digit number is obtained. This concept is helpful in error detection algorithms, where reducing a number to its single-digit form helps in quick verification. To accomplish this, you need to write a program that can handle such calculations efficiently, using different loop control structures.

Write a Java program that accepts a positive integer and sums its digits repeatedly until the result is a single-digit number. The program should demonstrate the use of different looping structures to achieve the task.

Input Format :

The program should prompt the user to enter a number that consists of several digits.

Sample Output:

```
Enter a number: 9876
Sum of digits until single digit: 3
```

Question 2 : Custom Number Sequence

Problem Statement:
You are given three integers nnn, aaa, and bbb. Your task is to generate a sequence of numbers according to the following rules:

1. Start with the number nnn.
2. In each step, you can perform one of the following operations:
    a. Subtract a.
    b. Divide by b (only if the number is divisible by b).
3. Your goal is to reach the number 1 using the fewest possible steps. If it's impossible to reach 1, return -1.

Constraints:

- $1 \leq n \leq 10^9$
- $1 \leq a \leq n$
- $2 \leq b \leq 10^5$

Input:
The input consists of a single line containing three integers n, a, and b.

Output:
Print the minimum number of steps required to reduce nnn to 1 using the above rules. If it is impossible, print -1.

Sample Output:

Input	Output
10 2 2	4
15 5 3	-1

Tips:

- n=10, divisible by 2:
    - Divide by 2, n=5, steps = 1.

- n=5, not divisible by 2:
    - Subtract 2, n=3, steps = 2.
- n=3, not divisible by 2:
    - Subtract 2, n=1, steps = 3.

Question 3: Number Analysis Tool

Problem Statement

Imagine a mathematics research team developing a comprehensive tool for analyzing positive integers. This tool is designed to support mathematicians in understanding the properties of numbers by providing detailed analyses. The tool can identify whether a number is prime, list its factors, and highlight other special properties like being a perfect number. Additionally, the tool will give a list of all prime numbers between 2 and the input number for deeper mathematical insight.

A mathematics professor is working on a research paper involving number properties and needs a quick analysis of various numbers. For instance, when analyzing the number 96, the professor wants to know:

- Whether it is prime.
- Its factors and how many there are.
- The sum and product of its factors.
- If it is a perfect number.
- All prime numbers less than 96 for comparison in research.

The professor uses the tool to:

1. Check if 96 is prime (it's not).
2. See that 96 has 12 factors: `1, 2, 3, 4, 6, 8, 12, 16, 24, 32, 48, 96`.
3. Get the sum of the factors (252) and identify that the product is too large to display due to size.
4. Discover that 96 is not a perfect number.
5. View a list of all prime numbers up to 96 for additional context.

Constraints:

- The input should be a positive integer greater than 1.
- The program should handle numbers up to the maximum value of `int` in Java `(2,147,483,647)`, but calculations like the product of factors may overflow for large numbers, so overflow handling should be considered.
- The program must use only flow control constructs (no arrays, methods, or external libraries).

- If the product of factors exceeds `Long.MAX_VALUE`, it should be indicated without displaying an incorrect result.

Input:

- A single positive integer provided by the user.

Output:

- A message indicating whether the input integer is prime.
- If not prime, display:
  - The number of factors and a list of all factors.
  - The sum of the factors.
  - The product of the factors (with overflow indication if necessary).
- A message indicating whether the number is a perfect number.
- A list of all prime numbers between 2 and the given number (if the input number is not prime).

Sample Output

```
Integer is not a prime number, it has 12 factors
The factors of this integer are:
1, 2, 3, 4, 6, 8, 12, 16, 24, 32, 48, 96
The sum of the factors is 252
The product of the factors is too large to display
96 is not a perfect number.
Prime numbers between 2 and 96: 2, 3, 5, 7, 11, 13, 17, 19,
23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83,
89
```

Question 4: Bill Calculation

Problem Statement

You are a programmer hired by Mr. Maroni, owner of the famous pizza shop Maroni's Pizza. You are tasked with creating a program to calculate the bills of customers based on their order.

The program consists of a main menu and three sub-menus.

```
Welcome to Maroni's Pizza!
1. Pizza
2. Drinks
3. Dessert
4. CHECKOUT

Pick an option:
```

Each sub-menu consists of items and the pricing for each.

```
PIZZA
1 Chicken Peperoni - RM15
2 Chicken Supreme - RM18
3 Vegan Indulgence - RM12
4 Beef Delight- RM22
5 Margherita - RM9
6 BACK TO MAIN MENU

Pick an option:
```

```
DRINKS
1 Strawberry Smoothie - RM8
2 Banana Smoothie - RM8
3 Mocktail - RM12
4 Soft Drink - RM5
5 Mineral Water - RM3
6 BACK TO MAIN MENU
```

```
 DESSERT
1 Tiramisu - RM7
2 Strawberry Shortcake - RM10
3 Green Jello - RM4
4 Crème Brûlée - RM15
5 Raspberry Pie - RM20
6 BACK TO MAIN MENU
```

Once you pick an item in a *sub-menu*, it is added to your cart and you are navigated back to the same sub-menu to pick more items. The program also keeps track of your current total.

```
PIZZA
1 Chicken Peperoni - RM15
2 Chicken Supreme - RM18
3 Vegan Indulgence - RM12
4 Beef Delight- RM22
5 Margherita - RM9
6 BACK TO MAIN MENU

Pick an option: 2
Added Chicken Supreme
Current total : RM18.0
```

You can go back to the main menu by choosing the 6th option in any of the sub-menus. From there you can checkout, displaying your total bill and ending the program.

```
Welcome to Maroni's Pizza!
1. Pizza
2. Drinks
3. Dessert
4. CHECKOUT

Pick an option: 4

Your total is RM36.0!
Have a nice day!
```

To encourage customers to purchase more, Maroni offers a 20% discount to all customers who order at least 1 item from each sub-menu. In case a customer orders at least one of each, the final checkout screen would look like this:

```
Welcome to Maroni's Pizza!
1. Pizza
2. Drinks
3. Dessert
4. CHECKOUT

Pick an option: 4

Your total is RM27.0!
You've availed the One-of-each offer. You get a 20% discount!
your new total is RM21.6!

Have a nice day!
```

Hint: you may use infinite *while(true)* loops to keep the program running, and use *break* and *continue* with the appropriate loop labels to help navigate the menus.

Question 5: DJ REMIX

Problem Statement

Alex works as a DJ in the best club in his city, and he often uses remix music in his performances. Recently, he has decided to take a couple of old songs and create remixes from them.

Let's assume that a song consists of some number of words. To make a DJ remix of this song, Alex inserts a certain number of words "REMIX" before the first word of the song (the number may be zero), after the last word (the number may be zero), and between words (at least one between any pair of neighboring words), and then he glues together all the words, including "REMIX", in one string and plays the song at the club.

For example, a song with words "I LOVE MUSIC" can transform into a DJ remix as "REMIXREMIXIREMIXLOVEREMIXMUSIC" and cannot transform into "REMIXREMIXILOVEMUSIC".

Recently, Sam has heard Alex's new remix track, but since he isn't into modern music, he decided to find out what the initial song that Alex remixed was. Help Sam restore the original song.

Input:
The input consists of a single non-empty string, consisting only of uppercase English letters, and the string's length doesn't exceed 200 characters. It is guaranteed that before Alex remixed the song, no word contained the substring "REMIX" in it; Alex didn't change the word order. It is also guaranteed that initially the song had at least one word.

Output:
Print the words of the initial song that Alex used to make a DJ remix. Separate the words with a space.

Sample Output:

Input	Output
REMIXREMIXABCREMIX	ABC
REMIXHELLOREMIXWORLDREMIXISREMIXBEAUTIFUL	HELLO WORLD IS BEAUTIFUL

Tips:

- In the first sample: "REMIXREMIXABCREMIX" = "REMIX" + "REMIX" + "ABC" + "REMIX". That means that the song originally consisted of a single word "ABC", and all words "REMIX" were added by Alex.

- In the second sample, Alex added a single word "REMIX" between all neighboring words, in the beginning, and at the end.

Question 6: Sports Tournament Scoring System

Problem Statement

Imagine a local sports tournament where players compete in various games, and their scores are recorded. The tournament organizers want a program to analyze the scores at the end of each round to provide useful insights. Players' scores are entered into the system one by one. The data entry ends when the score `0` is entered, indicating the end of the round. The program should then:

1. Identify the highest score among all the scores entered and count how many times it appears.
2. Determine the second-highest score and its frequency if it exists.
3. Calculate the total sum of all scores recorded in the round to get an idea of the scoring intensity.
4. Check and indicate if any negative scores were entered, which could represent penalties or deductions.

5. Provide a summary that includes these insights so that the tournament organizers can make data-driven decisions for future matches or detect outlier performances.

Example: During a fast-paced basketball shooting competition, players scored the following: 3, 5, 2, 5, -3, 5, 5, 0. The program should report:

- The highest score as 5, which occurred 4 times.
- The second-highest score as 3, which occurred 1 time.
- The total sum of all the scores (22).
- A note whether negative scores (e.g., penalties) were present in the data.

Sample output:

```
Enter numbers: 3 5 2 5 -3 5 5 0
The largest number is 5
The occurrence count of the largest number is 4
The second-largest number is 3
The occurrence count of the second-largest number is 1
The total sum of all numbers is 22
Negative numbers were entered.
```