Day 7	**Loops**

Objectives	Understand the basics of loops and their purpose in programming. Learn the syntax and use cases of different types of loops (for loop, while loop, do-while loop). Practice writing and controlling simple and nested loops for repetitive tasks.
Outcomes	Students will be able to 1.Design and implement different types of loops according to the application. 2.Enhance the problem-solving skills through the application of loops. 3.Manage loop behavior with multiple initialization and handle complex tasks with nested loops.

1. Sum of Two Numbers:

A customer places two items on the conveyor belt. The POS system scans each item, capturing their prices. These prices are then added together to determine the total cost of the purchase. One of the functionalities of the POS system is to calculate the total amount to be paid by the customer based on the items they purchase. As items are scanned or selected, their prices are added together to compute the total bill [**using Do While loop].**

Constraints:

Input :

Prices: Typically, the prices of items should be positive floating-point numbers. For simplicity, we assume:

- **0 <= price 1 <= 10^6 (i.e., prices can range from 0 to 1 million dollars)**
- **0 <= price 2 <= 10^6**

Output :

Total Amount: The total amount should be displayed with exactly two decimal places to represent the currency accurately.

TEST CASE 1:
Input:
Enter the number of items: 2
Enter price of item 1: 15.75
Enter price of item 2: 9.99
Output:

Total amount to be paid: $25.74

TEST CASE 2:
Input:
Enter the number of items: 3
Enter price of item 1: 0.000
Enter price of item 2: 0.011
Enter price of item 3: 0.111
Output:
Total amount to be paid: $0.12

TEST CASE 3:
Input:
Enter the number of items: 2
Enter price of item 1: 999.99
Enter price of item 2: 499.99
Output:
Total amount to be paid: $1499.98

TEST CASE 4:
Input:
Enter the number of items: 4
Enter price of item 1: 12.34
Enter price of item 2: 56.78
Enter price of item 3: 95.00
Enter price of item 4: 11.11
Output:
Total amount to be paid: $175.23

TEST CASE 5:
Input:
Enter the number of items: 2
Enter price of item 1: 7.50
Enter price of item 2: 3.25
Output:
Total amount to be paid: $10.75

TEST CASE 6:
Input:

Enter the number of items: 3
Enter price of item 1: 19.99
Enter price of item 2: 0.01
Enter price of item 3: 0.00
Output:
Total amount to be paid: $20.00

TEST CASE 7:
Input:
Enter the number of items: 3
Enter price of item 1: 123.45
Enter price of item 2: 678.90
Enter price of item 3: 999.45
Output:
Total amount to be paid: $1801.80

TEST CASE 8:
Input:
Enter the number of items: 0
Output:
Invalid number of items.

TEST CASE 9:
Input:
Enter the number of items: 2
Enter price of item 1: -50
Invalid price.
Enter price of item 1: 10
Enter price of item 2: 20
Output:
Total amount to be paid: $30.00

TEST CASE 10:
Input:
Enter the number of items: 2
Enter price of item 1: 50
Enter price of item 2: -20
Invalid price.
Enter price of item 2: 20

Output:
Total amount to be paid: $70.00

2. Factorial Calculation:
A project manager needs to determine the number of possible orderings for a set of tasks using permutations. Factorials can be used to calculate the number of permutations (different orderings) for a given number of tasks. Generate C program for the same.

Constraints:
Input
- Range: 0 <= num <= 20
- Type: Non-negative integer

Output
- Type: Integer
- Range: Factorial values fit within long long (up to 20! which is 2,432,902,008,176,640,000)

TEST CASE 1:
Input:
Enter a number: 0
Output:
Factorial of 0 is 1

TEST CASE 2:
Input:
Enter a number: -5
Output:
Factorial is not defined for negative numbers.

TEST CASE 3:
Input:
Enter a number: 3
Output:
Factorial of 3 is 6

TEST CASE 4:
Input:
Enter a number: 7
Output:

Factorial of 7 is 5040

TEST CASE 5:
Input:
Enter a number: 12
Output:
Factorial of 12 is 479001600

TEST CASE 6:
Input:
Enter a number: 06

Output:
Factorial of 6 is 720

TEST CASE 7:
Input:
Enter a number: 8
Output:
Factorial of 8 is 40320

TEST CASE 8:
Input:
Enter a number: 30
Output:
Number of tasks exceeds the limit.

TEST CASE 9:
Input:
Enter a number: 20
Output:
Factorial of 20 is 2432902008176640000

TEST CASE 10:
Input:
Enter a number: 21
Output:
Number of tasks exceeds the limit.

3. Palindrome Check:

Credit card numbers are often masked for security purposes. However, for internal processing, the full number is required. To ensure data integrity, it's crucial to verify that the unmasked number is correct. A palindrome check can be a part of this validation process.**[Using For Loop]**

Constraints:

- **Input Range**: The number can be any non-negative integer within the range of 0 to 10^9.
- Output Format:
  if the number is a palindrome, **True**
  Otherwise, **False**

TEST CASE 1:
Input:
Enter an integer: 12321
Output:
12321 is a palindrome.

TEST CASE 2:
Input:
Enter an integer: 12345
Output:
12345 is not a palindrome.

TEST CASE 3:
Input:
Enter an integer: 1234321
Output:
1234321 is a palindrome.

TEST CASE 4:
Input:
Enter an integer: 121
Output:
121 is a palindrome.

TEST CASE 5:
Input:
Enter an integer: 5
Output:
5 is a palindrome.

TEST CASE 6:
Input:
Enter an integer: 2002
Output:
2002 is a palindrome.

TEST CASE 7:
Input:
Enter an integer: 889
Output:
889 is not a palindrome.

TEST CASE 8:
Input:
Enter an integer: -101
Output:
Invalid number

TEST CASE 9:
Input:
Enter an integer: -751
Output:
Invalid number

TEST CASE 10:
Input:
Enter an integer: 0
Output:
0 is a palindrome.

4. Fibonacci Series:
Students are designing a network topology where the number of connections between nodes follows a Fibonacci sequence. This helps optimize the network structure for scalability and efficiency. By leveraging the Fibonacci sequence, you can ensure that each node connects to a number of other nodes that follows a predictable pattern, which can help in balancing the load and enhancing fault tolerance in the network. **[Using For Loop]**

Constraints:
Input: An integer n representing the number of terms in the Fibonacci series to generate.
Output: The Fibonacci series up to n terms that extends upto 40.

TEST CASE 1:
Input:
Enter the number of terms: 5
Output:
The Fibonacci series up to 5 terms is 0, 1, 1, 2, 3

TEST CASE 2:
Input:
Enter the number of terms:15
Output:
The Fibonacci series up to 15 terms is 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377

TEST CASE 3:
Input:
Enter the number of terms:10
Output:
The Fibonacci series up to 10 terms is 0, 1, 1, 2, 3, 5, 8, 13, 21, 34

TEST CASE 4:
Input:
Enter the number of terms: 2
Output:
The Fibonacci series up to 2 terms is 0, 1

TEST CASE 5:
Input:
Enter the number of terms: 25
Output:
The Fibonacci series up to 10 terms is 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, 6765, 10946, 17711, 28657, 46368

TEST CASE 6:
Input:
Enter the number of terms: 12
Output:
The Fibonacci series up to 10 terms is 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89

TEST CASE 7:
Input:
Enter the number of terms: -11
Output:
Please enter a positive integer greater than 0.

TEST CASE 8:
Input:
Enter the number of terms: 35
Output:
The Fibonacci series up to 10 terms is 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, 6765, 10946, 17711, 28657, 46368, 75025, 121393, 196418, 317811, 514229, 832040, 1346269, 2178309, 3524578, 5702887

TEST CASE 9:
Input:
Enter the number of terms: 41
Output:
Limit exceeded

TEST CASE 10:
Input:
Enter the number of terms: 40
Output:
The Fibonacci series up to 40 terms is: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, 6765, 10946, 17711, 28657, 46368, 75025, 121393, 196418, 317811, 514229, 832040, 1346269, 2178309, 3524578, 5702887, 9227465, 14930352, 24157817, 39088169, 63245986

5. Prime Number Checks:
In cryptography and security systems, prime numbers play a crucial role, especially in algorithms like RSA (Rivest–Shamir–Adleman), which are widely used for secure data transmission. One practical application is in generating large prime numbers, which are essential for ensuring the security and integrity of encrypted data.

Constraints:
Input: An integer **num** to be checked for primality.

Output: A message indicating whether **num** is a prime number or not.

TEST CASE 1:
Input:
Enter a number: 13
Output:
13 is a prime number.

TEST CASE 2:
Input:
Enter a number: 25
Output:
25 is not a prime number

TEST CASE 3:
Input:
Enter a number: 97
Output:
97 is a prime number.

TEST CASE 4:
Input:
Enter a number: -101
Output:
-101 is not a prime number.

TEST CASE 5:
Input:
Enter a number: abc
Output:
32764 is not a prime number.

TEST CASE 6:
Input:
Enter a number: 31
Output:
31 is a prime number.

TEST CASE 7:
Input:
Enter a number: 44
Output:
44 is not a prime number.

TEST CASE 8:
Input:
Enter a number: - 17
Output:
17 is not a prime number.

TEST CASE 9:
Input:
Enter a number: 101
Output:
101 is not a prime number.

TEST CASE 10:
Input:
Enter a number: xyz
Output:
32767 is not a prime number.

6. Reverse Numbers:
Data entry systems or applications where numeric input is crucial, such as in financial transactions or inventory management, it is often necessary to reverse the digits of a number. This can be useful for tasks like error checking or displaying numbers in a different format for clarity.

Constraints:
Input Range:
- The range is from 1 to 987654321.

Output Format:
- Return the reversed number.

TEST CASE 1:
Input:
Enter a number to reverse: 987654321
Output:
Reversed number: 123456789

TEST CASE 2:
Input:
Enter a number to reverse: -987
Output:
Number not in range. Please check

TEST CASE 3:
Input:
Enter a number to reverse: 997654321
Output:
Number not in range. Please check

TEST CASE 4:
Input:
Enter a number to reverse:120
Output:
Reversed number: 21

TEST CASE 5:
Input:
Enter a number to reverse:5000
Output:
Reversed number: 5

TEST CASE 6:
Input:
Enter a number to reverse: 1001
Output:
Reversed number: 1001

TEST CASE 7:
Input:
Enter a number to reverse: 0
Output:
Reversed number: 0

TEST CASE 8:
Input:
Enter a number to reverse: -123456789

Output:
Number not in range.Please check

TEST CASE 9:
Input:
Enter a number to reverse: 100000000
Output:
Reversed number: 1

TEST CASE 10:
Input:
Enter a number to reverse: 1
Output:
Reversed number: 1

7. Printing the Even Numbers:
The teacher wants a simple tool to demonstrate even numbers to the students. Program takes a number from the teacher as input and prints all even numbers from 1 up to the desired number.
Constraints:
Input:

- A single integer, num.
- The value of num can be negative, zero, or positive.

Output:

- Print all even numbers from 1 up to and including num if num is positive.
- Print 0 if num is zero.
- Print an error message "Invalid input. Please enter a positive number." if num is negative.

TEST CASE 1:
Input:
Enter a number: 20
Output:
Even numbers from 1 to 20:
2 4 6 8 10 12 14 16 18 20

TEST CASE 2:
Input:
Enter a number: 0

Output:
0

TEST CASE 3:
Input:
Enter a number: -5
Output:
Invalid input. Please enter a positive number.

TEST CASE 3:
Input:
Enter a number: 120
Output:
Even numbers from 1 to 120:
2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48 50 52 54 56 58 60 62 64 66 68 70 72 74 76 78 80 82 84 86 88 90 92 94 96 98 100 102 104 106 108 110 112 114 116 118 120

TEST CASE 4:
Input:
Enter a number: 012
Output:
Even numbers from 1 to 12:
2 4 6 8 10 12

TEST CASE 5:
Input:
Enter a number: 27
Output:
Even numbers from 1 to 27:
2 4 6 8 10 12 14 16 18 20 22 24 26

TEST CASE 6:
Input:
Enter a number: -40
Output:
Invalid input. Please enter a positive number.

TEST CASE 7:
Input:
Enter a number: 013
Output:
Even numbers from 1 to 13:
2 4 6 8 10 12

TEST CASE 8:
Input:
Enter a number: -55
Output:
Invalid input. Please enter a positive number.

8. Armstrong number:

Contestants are presented with a series of complex number-based challenges. One particular round requires participants to identify Armstrong numbers within a given time limit. The program should aim to help contestants quickly verify if a number meets the Armstrong criteria.

Constraints:

Input Range:

- **Positive Integers:** The program is designed to handle positive integers. For example, typical values could range from 0 to large numbers like 9999. However, extremely large numbers might be impractical due to computational limits.
- **Negative and Non-integer Inputs:** Invalid Inputs: The program should handle non-positive integers by providing an appropriate error message, as it is assumed that only positive integers are valid.

Output Format:

- If the number is an Armstrong number:
  Output the number followed by "is an Armstrong number."

- If the number is not an Armstrong number:
  Output the number followed by "is not an Armstrong number."

TEST CASE 1:
Input:
Input a number: 153
Output:
153 is an Armstrong number.

TEST CASE 2:
Input:
Input a number: 123
Output:
123 is not an Armstrong number.

TEST CASE 3:
Input:
Input a number: 0
Output:
0 is an Armstrong number.

TEST CASE 4:
Input:
Input a number: 407
Output:
407 is an Armstrong number.

TEST CASE 5:
Input:
Input a number: 370
Output:
370 is an Armstrong number.

TEST CASE 6:
Input:
Input a number: 121
Output:
121 is not an Armstrong number.

TEST CASE 7:
Input:
Input a number: 1
Output:
1 is an Armstrong number.

TEST CASE 8:
Input:
Input a number: 371

Output:
371 is an Armstrong number.

TEST CASE 9:
Input:
Input a number: -371
Output:
Invalid input

TEST CASE 10:
Input:
Input a number: 10000
Output:
Invalid input

9. Number Pattern:
Imagine that the teachers are tasked with developing a program that prints a specific number pattern to help students learn about loops and nested loops in C programming. The pattern needs to generate consecutive numbers in an incremental triangular format, where each row contains one more number than the previous row.
Constraints:
Input Range:

- **Positive Integer:** The input should be a positive integer (n), which determines the number of rows in the pattern. Typically, n should be within a reasonable range for practical output, such as 1 to 100, to avoid excessively large outputs that may be impractical to handle.

- **Invalid Input Handling:**
  Non-positive Integers: The program should handle non-positive integers (n <= 0) by providing an error message indicating "Invalid Input."

Output Format:
For a valid positive integer input n:

The output should display n rows of numbers in a triangular format. Each row **i** contains **i** numbers, starting from 1 and incrementing consecutively.

TEST CASE 1:
Input:
Enter a positive integer: 5
Output:
1
2 3
4 5 6
7 8 9 10
11 12 13 14 15

TEST CASE 2:
Input:
Enter a positive integer: 3
Output:
1
2 3
4 5 6

TEST CASE 3:
Input:
Enter a positive integer: -5
Output:
Invalid Input

TEST CASE 4:
Input:
Enter a positive integer: 2
Output:
1
2 3

TEST CASE 5:
Input:
Enter a positive integer: 101
Output:
Invalid input

TEST CASE 6:
Input:
Enter a positive integer: 0
Output:
Invalid Input

TEST CASE 7:
Input:
Enter a positive integer: 1
Output:
1

TEST CASE 8:
Input:
Enter a positive integer: 7
Output:
1
2 3
4 5 6
7 8 9 10
11 12 13 14 15
16 17 18 19 20 21
22 23 24 25 26 27 28

10. LCM of Two Numbers:

Consider a scenario where a software application needs to determine the frequency of events that occur independently. For example, a traffic light controller might need to calculate the least common multiple of the green light durations for two intersecting roads to synchronize their cycles. In this case, finding the LCM of the green light intervals would ensure that both lights turn green simultaneously at regular intervals, optimizing traffic flow.

Constraints:

Input Range:

- Input values of green light intervals must be positive integers.

Output:

- Prints the calculated LCM in the format "LCM = value".
- Provides an error message for invalid input.

TEST CASE 1:
Input:

Enter two positive integers:
72
120
Output:
LCM = 360

TEST CASE 2:
Input:
Enter two positive integers:
15
25
Output:
LCM = 75

TEST CASE 3:
Input:
Enter two positive integers:
7
5
Output:
LCM = 35

TEST CASE 4:
Input:
Enter two positive integers:
0
0

Output:
Floating point exception

TEST CASE 5:
Input:
Enter two positive integers:
8
12
Output:
LCM = 24

TEST CASE 6:
Input:
Enter two positive integers:
1
100
Output:
LCM = 100

TEST CASE 7:
Input:
Enter two positive integers:
50
50
Output:
LCM = 50

TEST CASE 8:
Input:
Enter two positive integers:
9
28
Output:
LCM = 252

TEST CASE 9:
Input:
Enter two positive integers:
-12
6
Output:
ERROR!
Error: Please enter positive integers.

TEST CASE 10:
Input:
Enter two positive integers: 6
0
Output:
Floating point exception