Day 8	**Jump Statements & Switch Statements**

Objectives	To understand the purpose and usage of jump statements (break, continue, goto, return) in controlling the flow of a program. To implement and use the switch statement to handle multiple conditions and to practice these statements to manage and control program execution efficiently.
Outcomes	Students will be able to, 1.Effectively use jump statements to control the flow of loops and conditional structures. 2.Enhanced skills in using the switch statement to simplify complex conditional logic.

<u>Jump Statements and Switch Statement</u>

1. Counting Positive and Negative Numbers

A research team is conducting a survey to collect data on people's sentiments about a new policy. Participants are asked to rate their sentiments with a whole number, where positive numbers indicate positive sentiments, and negative numbers indicate negative sentiments. The survey ends when a participant enters 0.

Constraints:

- The integer 0 should not be counted in the positive or negative counts.
- The termination condition occurs when xi=0.

Test case 1:
Input:
5
10
15
0
Output:
Count p: 3
Count n: 0

Test case 2:
Input:
-1
-5
-10
0
Output:
Count p: 0
Count n: 3

Test case 3:
Input:
5
-5
10
10
0
Output:
Count p: 2
Count n: 2

Test case 4:
Input:
8
0
Output:
Count p: 1
Count n: 0

Test case 5:
Input:
-8
0
Output:
Count p: 0
Count n: 1

Test case 6:
Input:
1
2
3
4
5
0
Output:
Count p: 5
Count n: 0

Test case 7:
Input:
10
10
0
Output:
Count p: 1
Count n: 1

Test case 8:
Input:
abc
Output:
Invalid input.Please enter a valid Integer

2. Kaprekar Number Checker :
An educational app is being developed to provide a comprehensive and engaging learning experience in number theory. The app aims to foster a deep understanding of fundamental number properties and their applications. To achieve this, interactive features are being integrated to complement the theoretical content.

Constraints:

- **Input Range**: The input number **n** should be a non-negative integer within the range of 0 to 10^6 to ensure efficient computation and avoid overflow issues.

- **Output**: The program should clearly indicate whether each input number is a Kaprekar number or not, and should handle invalid inputs (like negative numbers) by printing an appropriate error message.
- **Exit Condition**: Use **-1** as a sentinel value to terminate the program, with no additional output required upon exiting.
- **Kaprekar number Constraints:** Calculate the square of **n**, denoted as **n^2.**
  Split **n^2** into two parts: the right part and the left part

  1. The number of digits in the right part should be equal to the number of digits in **n**.
  2. The left part contains the remaining digits.

  If the sum of the left part and the right part equals n, then **n** is a Kaprekar number.

Test case 1:
Input:
Enter a number (or -1 to exit):
123
-1
Output:
123 is not a Kaprekar number

Test case 2:
Input:
Enter a number (or -1 to exit):
297
-1
Output:
297 is a Kaprekar number

Test case 3:
Input:
Enter a number (or -1 to exit):
xy
-1
Output:
Invalid input. Please enter a valid integer.

Test case 4:
Input:
Enter a number (or -1 to exit):
0
Output:
Floating point exception

Test case 5:
Input:
Enter a number (or -1 to exit):
56
99
-1
Output:
56 is not a Kaprekar number
99 is a Kaprekar number

Test case 6:
Input:
Enter a number (or -1 to exit):
2970
-1
Output:
2970 is not Kaprekar number

Test case 7:
Input:
Enter a number (or -1 to exit):
25
10
-1
Output:
25 is not a Kaprekar number
10 is not a Kaprekar number

Test case 8:
Input:
Enter a number (or -1 to exit): -12300
Output:
Invalid input. Number must be non-negative.

Test case 9:
Input:
Enter a number (or -1 to exit):
81
45
7777
-1
Output:
81 is not a Kaprekar number
45 is a Kaprekar number
7777 is a Kaprekar number

3. Skipping Divisibility Checks

A manufacturing company produces batches of products and each product has an identification number. The quality control team needs to check the identification numbers of products to find those that are divisible by 7, as these numbers are tagged for special inspection. They need to process a list of identification numbers provided by the production line. The checking continues until they reach the end of the list, which is signaled by the number -1.

Note:

- Xi be the i-th number entered by the user, where i∈{1,2,…,n}.
- D be the set of numbers divisible by 7.

Constraints:

- A number xi is in D if: xi mod 7=0 where mod denotes the modulo operation.
- The termination condition occurs when xi= −1.
- Valid inputs are integers only
- Xi= -1 represents the end of the list and should not be processed for divisibility.
- Use **continue** to skip numbers that are not divisible by 7.
- Use **break** to exit the loop when the termination condition (-1) is met.

Test case 1:
Input:
Enter identification numbers (end with -1):
7
14
21
28

-1(break)
Output:
7 is divisible by 7.
14 is divisible by 7.
21 is divisible by 7.
28 is divisible by 7.

Test case 2:
Input:
Enter identification numbers (end with -1):
1
2
-1(break)
Output:
1 is not divisible by 7.
2 is not divisible by 7.

Test case 3:
Input:
Enter identification numbers (end with -1):
2
14
16
-1(break)
Output:
2 is not divisible by 7.
14 is divisible by 7.
16 is not divisible by 7.

Test case 4:
Input:
Enter identification numbers (end with -1):
"xyz"
-1
Output:
Invalid input. Please enter a valid integer. (No output)

Test case 5:
Input:
Enter identification numbers (end with -1):
“abc”
25
21
-1
Output:
Invalid input. Please enter a valid integer.
21 is divisible by 7.

Test case 6:
Input:
Enter identification numbers (end with -1):
”xyz”
7
-1
Output:
Invalid input. Please enter a valid integer.
7 is divisible by 7.

Test case 7:
Input:
Enter identification numbers (end with -1):
-7
-14
-21
-1
Output:
-7 is divisible by 7.
-14 is divisible by 7.
-21 is divisible by 7.

Test case 8:
Input:
Enter identification numbers (end with -1):
-21

5
14
-7
-1
Output:
-21 is divisible by 7.
14 is divisible by 7.
-7 is divisible by 7

4. Printing Multiples of 5

In an inventory management system, a warehouse needs to track and print out the serial numbers of items that are multiples of 5 for a specific inventory check. The serial numbers are assigned in a sequential order, and the system needs to print only those serial numbers up to a given maximum number that are multiples of 5. This is useful for categorizing or checking batches of items that have specific serial number patterns.

Note:

- L be the upper limit for serial numbers.
- Si be the i-th serial number where i∈{1,2,…,L}.

Constraints:

- A number Si is a multiple of 5 if:

  Si mod 5=0

  where mod denotes the modulo operation.
- Use **continue** to skip numbers that are not multiples of 5.
- Si > 0 is the valid input.
- Si <5 are invalid inputs.

Test case 1:
Enter the upper limit for serial numbers: 5
Output:
Multiples of 5 up to 5 are:
5

Test case 2:
Enter the upper limit for serial numbers:12
Output:
Multiples of 5 up to 12 are:
5
10

Test case 3:
Enter the upper limit for serial numbers: 3
Output:
Invalid input. Enter input within limit.

Test case 4:
Enter the upper limit for serial numbers: 0
Output:
Invalid input. Enter input within limit.

Test case 5:
Enter the upper limit for serial numbers: -10
Output:
Invalid input. Enter input within limit.

Test case 6:
Enter the upper limit for serial numbers: 50
Output:
Multiples of 5 up to 50 are:
5
10
15
20
25
30
35
40
45
50

Test case 7:
Enter the upper limit for serial numbers: 25
Output:
Multiples of 5 up to 25 are:
5
10
15
20

25

Test case 8:
Enter the upper limit for serial numbers: 26
Output:
Multiples of 5 up to 26 are:
5
10
15
20
25

Test case 9:
Enter the upper limit for serial numbers: abc
Output:
Invalid input. Enter integer input.

Test case 10:
Enter the upper limit for serial numbers: Twenty
Output:
Invalid input. Enter integer input.

5. Zuckerman Number Checker:
Imagine a number validation system for a lottery. The system needs to check if a user-entered number is a "Zuckerman Number" to be eligible for the lottery. A Zuckerman number is divisible by the product of its digits. The system should allow multiple number inputs and display whether each number is a Zuckerman number or not.

Constraints:

- **Input:** The input number **N** should be a positive integer.
- **Output:** A statement indicating whether the number is a Zuckerman number or not.

Test Case 1:
Input:
Enter a number: 115
Output:
115 is a Zuckerman number.

Test Case 2:
Input:
Enter a number: 123
Output:
123 is not a Zuckerman number.

Test Case 3:
Input:
Enter a number: 0
Output:
Invalid input. Please enter a positive integer.

Test Case 4:
Input:
Enter a number: 135
Output:
135 is a Zuckerman number

Test Case 5:
Input:
Enter a number: -98
Output:
Invalid input. Please enter a positive integer.

Test Case 6:
Input:
Enter a number: 999
Output:
999 is a Zuckerman number

Test Case 7:
Input:
Enter a number: 12345
Output:
12345 is not a Zuckerman number

Test Case 8:
Input:
Enter a number: 777

Output:
777 is a Zuckerman number

Test Case 9:
Input:
Enter a number: 54321
Output:
54321 is not a Zuckerman number

Test Case 10:
Input:
Enter a number: 28
Output:
28 is not a Zuckerman number.

6. Prime Number Validator:
Imagine you are developing a utility for a mathematical research institute that needs to validate and verify prime numbers quickly and efficiently. Researchers frequently input various numbers to determine their primality, and they need a tool that can handle multiple checks in a user-friendly manner. The tool should allow them to stop testing whenever they decide, providing results instantly for each number entered.

Constraints:

- Implement a function to check if a number is prime efficiently.
- Handle user input gracefully, allowing them to stop testing seamlessly when input entered is -1.

Test case 1:
Input:
Enter a number to check for primality:
1
-1
Output:
Prime numbers are greater than 1.

Test case 2:
Input:
Enter a number to check for primality:
2

191
-1
Output:
2 is a prime number.
191 is a prime number.

Test case 3:
Input:
Enter a number to check for primality:
3
-2
-1
Output:
3 is a prime number
Prime numbers are greater than 1

Test case 4:
Input:
Enter a number to check for primality:
4
-99
53
-1
Output:
4 is not a prime number.
Prime numbers are greater than 1.
53 is a prime number

Test case 5:
Input:
Enter a number to check for primality:
7
211
-5
-1
Output:
7 is a prime number.
211 is a prime number
Prime numbers are greater than 1

Test case 6:
Input:
Enter a number to check for primality:
997
0
abc
-1
Output:
997 is a prime number.
Prime numbers are greater than 1.
Invalid input. Please enter an integer.

Test case 7:
Input:
Enter a number to check for primality:
1000
525
31
-1
Output:
1000 is not a prime number.
525 is not a prime number.
31 is a prime number

Test case 8:
Input:
Enter a number to check for primality:
-5
111
-1
Output:
Prime numbers are greater than 1.
111 is not a prime number

Test case 9:
Input:
Enter a number to check for primality:
0

1a1
-1
Output:
Prime numbers are greater than 1.
Prime numbers are greater than 1, Invalid input.Please enter an Integer.

Test case 10:
Input:
Enter a number to check for primality :
3.14
-1
Output:
Invalid input. Please enter a number.

7. Fibonacci Series:

A beekeeper observes that the number of bees in his hive seems to increase in a pattern. He suspects a Fibonacci-like growth pattern. How can he model this growth mathematically and predict the future bee population? Implement a C program that can print the Fibonacci series using the goto statement.

Constraints:

- **Input:** The number of terms should be a non-negative integer.
- **Output:** The Fibonacci series will be printed, with each number on a new line.
- **Error handling:** The program should handle invalid input (negative numbers) gracefully.

Test Case 1:
Input:
Enter the number of terms: 5
Output:
Fibonacci Series:
0
1
1
2
3

Test Case 2:
Input:
Enter the number of terms: 8
Output:
Fibonacci Series:
0
1
1
2
3
5
8
13

Test Case 3:
Input:
Enter the number of terms: -5
Output:
Invalid input. Please enter a positive integer

Test Case 4:
Input:
Enter the number of terms: 15
Output:
Fibonacci Series:
0
1
1
2
3
5
8
13
21
34
55
89

144
233
377

Test Case 5:
Input:
Enter the number of terms:
0
Output:
Fibonacci Series:(no output)

Test Case 6:
Input:
Enter the number of terms:
-12
Output:
Invalid input. Please enter a positive integer

Test Case 7:
Input:
Enter the number of terms:
abc
Output:
Invalid input. Please enter a positive integer

Test Case 8:
Input:
Enter the number of terms:
11
Output:
Fibonacci Series:
0
1
1
2
3
5

8
13
21
34
55

Test Case 9:
Input:
Enter the number of terms:
01
Output:
Fibonacci Series:
0

Test Case 10:
Input:
Enter the number of terms:
2
Output:
Fibonacci Series:
0
1

8. Largest of Three Numbers:
While developing a basic data analysis tool for a small-scale research project,one of the initial requirements is to identify the highest value among three experimental data points. To efficiently process this data, develop a C program incorporating a switch statement for optimal performance and clarity.

Constraints:
- **Input:** Integer values within the representable range of an **int** data type.
- Switch statements must be used.
- **Output:** Integer representing the largest input value.

Test Case 1:
Input:
Enter the first number: 10
Enter the second number: 20
Enter the third number: 30

Output:
The largest number is: 30

Test Case 2:
Input:
Enter the first number: -10
Enter the second number: -20
Enter the third number: -30
Output:
The largest number is: -10

Test Case 3:
Input:
Enter the first number: 10
Enter the second number: -20
Enter the third number: 80
Output:
The largest number is: 80

Test case 4:
Input:
Enter the first number: 5
Enter the second number: 5
Enter the third number: 5
Output:
The largest number is: 5

Test case 5:
Input:
Enter the first number: 25
Enter the second number: 25
Enter the third number: 30
Output:
The largest number is: 25

Test case 6:
Input:
Enter the first number: 3

Enter the second number: 2
Enter the third number: 1
Output:
The largest number is: 3

Test case 7:
Input:
Enter the first number: 100
Enter the second number: -50
Enter the third number: 0
Output:
The largest number is: 100

Test case 8:
Input:
Enter the first number: -12
Enter the second number: -23
Enter the third number: -9
Output:
The largest number is: -9

Test case 9:
Input:
Enter the first number: 10
Enter the second number: 30
Enter the third number: -80
Output:
The largest number is: 30

Test case 10:
Input:
Enter the first number: 1000
Enter the second number: 999
Enter the third number: 1001
Output:
The largest number is: 1001

9. Denomination Calculation for Specified Amount:

Working as a cashier at a store, and a customer gives you a specific amount of money to pay for their purchase. You need to return the exact change using the fewest possible currency notes or coins. This program can help you calculate the minimum number of notes or coins required for the given denominations to make up the change accurately using Switch-case statement.

Constraints:

- Input: The amount should be a positive integer between 1 and 10,000.
- Denominations available: 10, 5, 2, 1 (in descending order).
- Output: The total count of denominations must be the minimum number needed to reach the given amount.

Test case 1:
Input:
Enter the amount: 42
Output:
10 rupees: 4
5 rupees: 0
2 rupees: 1
1 rupees: 0
The count of total denominations is 5

Test case 2:
Input:
Enter the amount: 99
Output:
10 rupees: 9
5 rupees: 1
2 rupees: 2
1 rupees: 0
The count of total denominations is 12

Test case 3:
Input:
Enter the amount: -40

Output:
Invalid input. Money cannot be negative. Please check

Test case 4:
Input:
Enter the amount: 120
Output:
10 rupees: 12
5 rupees: 0
2 rupees: 0
1 rupees: 0
The count of total denominations is 12

Test case 5:
Input:
Enter the amount: 55
Output:
10 rupees: 5
5 rupees: 1
2 rupees: 0
1 rupees: 0
The count of total denominations is 6

Test case 6:
Input:
Enter the amount: 7
Output:
10 rupees: 0
5 rupees: 1
2 rupees: 1
1 rupees: 0
The count of total denominations is 2

Test case 7:
Input:
Enter the amount: 89
Output:
10 rupees: 8
5 rupees: 1

2 rupees: 2
1 rupees: 0
The count of total denominations is 11

Test case 8:
Input:
Enter the amount: 14
Output:
10 rupees: 1
5 rupees: 0
2 rupees: 2
1 rupees: 0
The count of total denominations is 3

Test case 9:
Input:
Enter the amount: 53
Output:
10 rupees: 5
5 rupees: 0
2 rupees: 1
1 rupees: 1
The count of total denominations is 7

Test case 10:
Input:
Enter the amount: 0
Output:
10 rupees: 0
5 rupees: 0
2 rupees: 0
1 rupees: 0
The count of total denominations is 0

10. Sum of Even Numbers:
In a company's payroll system, the accountant needs to calculate the total of all even employee salaries for a special budget analysis. The system reads salary amounts in a loop, adds even salaries to the total sum, skips odd salaries, and exits the loop when there are no more salaries to process.

Develop a C program that finds the sum of all even numbers in a list of numbers (salaries), using the **continue** statement to skip odd numbers and the **break** statement to exit the loop when there are no more numbers to process.

Constraints:

- Use the **continue** statement to skip numbers where **Input** is odd.
- Use the **break** statement to exit the loop when there are no more numbers to process.
- Each **Input** should be a non-negative integer.
- Read inputs until -1 is encountered.

Test case 1:
Input:
4 7 10 15 -1
Output: Sum of all even numbers: 14

Test case 2:
Input:
2 4 6 8 10 -1
Output: Sum of all even numbers: 30

Test case 3:
Input:
1 3 5 7 9 -1
Output: Sum of all even numbers: 0

Test case 4:
Input:
8 -1
Output: Sum of all even numbers: 8

Test case 5:
Input:
5 -1
Output: Sum of all even numbers: 0

Test case 6:
Input:
1000000 2000000 3000000 -1
Output: Sum of all even numbers: 6000000

Test case 7:
Input:
12 15 18 21 24 -1
Output: Sum of all even numbers: 54

Test case 8:
Input:
2 999999 1000000 -1
Output: Sum of all even numbers: 1000002

Test case 9:
Input:
1 2 3 4 5 -1
Output: Sum of all even numbers: 6

Test case 10:
Input:
100 200 300 400 -1
Output: Sum of all even numbers: 1000