Friday, oct 20, 2023.

Programming Fundamentals *Lab*
(BS-IT-F22 Afternoon)
Lab # 5

Instructions:

Attempt the following tasks exactly in the given order. For each task, you must create a separate file in **VSCode.**

You must complete all tasks individually. Absolutely NO collaboration is allowed.
Any traces of plagiarism/cheating would result in an **"F"** grade in this course.

Indent your code properly(There are special marks for this).

Use meaningful variable names. Use the **camelCase** notation to name variables.

Use meaningful prompt lines/labels for all input/output that is performed by your programs.
Don't use an online compiler.

Today's Agenda

- **Dry Running**:
  - Explanation of the concept of dry running in programming.
  - Demonstration of how dry running helps understand code execution.
- **Variable Updation in Loops**:
  - Using variables within loops.
  - Demonstrating how loops can modify variables.
- **Unary Operators in Loops**:
  - Introduction to unary operators (++, --) and their use in loops.
  - Pre-increment vs. post-increment operators.
- **Mathematical Problem Solving**:
  - Practical examples of using variables and loops to solve mathematical problems.
  - Sample problems and step-by-step solutions.
- **Precedence of operators**.

Outcomes:

- Declare and manage variables in your programs effectively.
- Use conditional statements and loops to control program flow.
- Apply a range of operators to manipulate data and make decisions.
- Approach mathematical problem-solving tasks with confidence.
- Independently create, run, and debug C programs.
- Better understand the fundamentals of programming, preparing you for more complex coding tasks in the future.

Write **C programs** for the following tasks:

Task # 1

Write a C program which takes a positive integer ***n*** from the user. Then, your program should take ***n*** numbers from the user and determine both the **smallest** and the **largest** number entered by the user.

Note: Run your program on various inputs, and verify its correctness. For example, (i) provide all negative numbers as input, (ii) provide a mix of positive and negative numbers as input, (iii) repeat one value as input, etc.

Sample Run is given below:

```
Enter the number of values (n): 5

Enter value 1: 42
Enter value 2: -5
Enter value 3: 17
Enter value 4: 99
Enter value 5: 0

The smallest value is: -5
The largest value is: 99
```

Task # 2

Write a C program which takes a positive integer from the user and determines whether the number is a **perfect number** or not.

Note 1: A ***perfect number*** is a positive integer that is equal to the sum of its *proper divisors*, that is, the sum of its divisors excluding the number itself.

Note 2: Two examples of perfect numbers are: **6** (is equal to 1+2+3) and **28** (is equal to 1+2+4+7+14).

Hint:

1. Calculate the sum of the proper divisors of the number (excluding the number itself) by iterating from 1 to num/2.
2. Use the modulus operator (%) to check if a number is a divisor (i.e., if num % divisor == 0).
3. Compare the calculated sum with the original number. If they are equal, the number is a perfect number.

Sample Run:

```
Enter a positive integer: 28
28 is a perfect number.
```

Task # 3

Write a C program that takes a positive integer ***n*** from the user and calculates and displays the
Factorial of ***n***.

Input Validation: *Make sure that* ***n*** *is greater than or equal to zero.*

We can calculate factorial as:

- *factorial of 5 can be calculate as* $5 \times 4 \times 3 \times 2 \times 1 = 120$
- *factorial of 4 can be calculate as* $4 \times 3 \times 2 \times 1 = 24$
- *factorial of 3 can be calculate as* $3 \times 2 \times 1 = 6$

Sample Run:

```
Enter a positive integer: 5
The factorial of 5 is 120
```

Task # 4

Write a C program that inputs two positive integers ***a*** and ***b*** from the user. Then, your program should calculate and display the **Product** of ***a*** and ***b***. You are NOT allowed to use the multiplication operator (*) is your program.

Input Validation: *Your program should make sure the both* ***a*** *and* ***b*** *are positive integers.*

Sample Run:

```
Enter the first positive integer (a): 4
Enter the second positive integer (b): 3
The product of 4 and 3 is 12
```

Task # 5

Write a C program that inputs two positive integers ***a*** and ***b*** from the user. Then, your program should calculate and display the **Addition** of ***a*** and ***b***. You are NOT allowed to use the binary operator(which takes two operands) in your program.

Input Validation: *Your program should make sure the both **a** and **b** are positive integers.*

Sample Run:

```
Enter the first positive integer (a): 5
Enter the second positive integer (b): 3
The addition of 5 and 3 is 8
```

Task # 6

Write a C program to swap two numbers without using any variable.

Task # 7

Write a C program to check whether a number is palindrome or not.

Palindrome numbers are those numbers which after reversing the digits equals the original number. For example, 12321 after reversing is 12321, so it is a palindrome number. But the number 1232 after reversing is 2321, so it is not a palindrome number.

Task # 8

Write a C program to check if a given three digit number is an Armstrong number or not.

The **Armstrong number** can be defined as a number that is equal to the result of the summation of the nth power of each n digit.

Input: *153*

Output: *Yes*

Explanation :

153 is an Armstrong number of 3 digits, since the sum of cubes of each digit is equal to the number itself. As shown below:

*1*1*1 + 5*5*5 + 3*3*3 = 153*

Input: *120*

Output: *No*

Explanation :

120 is not a Armstrong number of 3 digits, the sum of cubes of each digit is equal to 9 but number is 120.

*1*1*1+2*2*2+0*0*0 != 1+8+0 = 9*

Home Task

Task # 9

Solve Task 4 and Task 5 for both positive and negative integers? It will be great fun.

Task # 10

Write a C program that takes a positive integer n from the user and displays the first n terms of the Fibonacci sequence.

Fibonacci sequence: The Fibonacci sequence is a series of numbers where each number is the sum of the two preceding ones. It begins with 0 and 1.

- The first number is 0.
- The second number is 1.
- Each subsequent number is the sum of the two numbers before it.

So, the sequence starts as follows:0, 1, 1, 2, 3, 5, 8, 13, 21, 34, …

Here's how the Fibonacci sequence is generated:

1. Start with the first two numbers, 0 and 1.
2. To find the third number, add the first two numbers: 0 + 1 = 1.
3. To find the fourth number, add the second and third numbers: 1 + 1 = 2.
4. Continue this process, always adding the two most recent numbers to get the next number in the sequence.