IESTP

“ANDRÉS AVELINO CÁCERES DORREGARAY”

DISEÑO Y PROGRAMACIÓN WEB

Actividad 05

CURSO : Programación de Aplicaciones Web y Móviles

TURNO : Diurno - VI

PROFESOR : Raúl Fernandez

ALUMNO : Quispe Osorio Luis Alberto

2024

DESARROLLO DE EJERCICIOS

Ejercicio 1: Sumatoria de números primos en un rango

1. Enunciado

Desarrollar un programa que solicite dos números y calcule la sumatoria de los números primos que existen entre esos dos valores.

2. Requerimientos Funcionales

2.1 Entrada de Datos

- **Primer Número:**
    - El usuario debe ingresar el primer número del rango.
    - Condición: Debe ser un valor numérico entero.
- **Segundo Número:**
    - El usuario debe ingresar el segundo número del rango.
    - Condición: Debe ser un valor numérico entero mayor que el primero.

2.2 Proceso/Cálculo

- **Identificación de Números Primos:**
    - Verificar si cada número en el rango es primo.
    - Sumar los números primos encontrados en el rango.

2.3 Salida de Datos

- **Sumatoria de Números Primos:**
    - Mostrar la suma de todos los números primos encontrados en el rango.

3. Validaciones

- **Rango de Números:**
    - Ambos números deben ser enteros.
    - El segundo número debe ser mayor que el primero.
    - Si se ingresan valores inválidos, mostrar un mensaje de error y solicitar nuevamente.

Ejercicio 2: Números de Fibonacci hasta N términos

1. Enunciado

Implementar un programa que genere la secuencia de Fibonacci hasta un número nnn de términos ingresado por el usuario.

2. Requerimientos Funcionales

2.1 Entrada de Datos

- **Número de Términos:**
    - El usuario debe ingresar un número que representa cuántos términos de Fibonacci quiere generar.
    - Condición: Debe ser un valor numérico entero positivo.

2.2 Proceso/Cálculo

- **Generación de la Secuencia de Fibonacci:**
    - Utilizar un bucle para calcular los términos de Fibonacci hasta llegar al número de términos especificado.

2.3 Salida de Datos

- **Secuencia de Fibonacci:**
    - Mostrar los números de la secuencia generada.

3. Validaciones

- **Número de Términos:**
    - Debe ser un entero positivo.
    - Si se ingresa un valor inválido, mostrar un mensaje de error y solicitar nuevamente.

Ejercicio 3: Factorial de números grandes

1. Enunciado

Escribir un programa que calcule el factorial de un número grande (por ejemplo, 100) utilizando estructuras repetitivas.

2. Requerimientos Funcionales

2.1 Entrada de Datos

- **Número para Calcular Factorial:**
    - El usuario debe ingresar un número entero positivo para el cual desea calcular el factorial.

2.2 Proceso/Cálculo

- **Cálculo del Factorial:**
    - Utilizar un bucle para multiplicar todos los números desde 1 hasta el número ingresado.
    - Utilizar el tipo de datos BigInt para manejar grandes números.

2.3 Salida de Datos

- **Resultado del Factorial:**
    - Mostrar el resultado del cálculo del factorial.

3. Validaciones

- **Número para Factorial:**
    - Debe ser un entero positivo.
    - Si se ingresa un valor inválido, mostrar un mensaje de error y solicitar nuevamente.

Ejercicio 4: Inversión de un número

1. Enunciado

Crear un programa que invierta los dígitos de un número entero ingresado por el usuario.

2. Requerimientos Funcionales

2.1 Entrada de Datos

- **Número a Invertir:**
    - El usuario debe ingresar un número entero.

2.2 Proceso/Cálculo

- **Inversión de Dígitos:**
    - Utilizar un bucle para extraer y reordenar los dígitos del número.

2.3 Salida de Datos

- **Número Invertido:**
    - Mostrar el número con sus dígitos en orden inverso.

3. Validaciones

- **Número a Invertir:**
    - Debe ser un entero.
    - Si se ingresa un valor inválido, mostrar un mensaje de error y solicitar nuevamente.

Ejercicio 5: Suma de matrices NxN

1. Enunciado

Escribir un programa que solicite dos matrices de tamaño N×NN \times NN×N y realice la suma de las dos matrices.

2. Requerimientos Funcionales

2.1 Entrada de Datos

- **Tamaño de la Matriz:**
    - El usuario debe ingresar el tamaño NNN de la matriz.
- **Matrices:**
    - El usuario debe ingresar los elementos de las dos matrices.

2.2 Proceso/Cálculo

- **Suma de Matrices:**
    - Utilizar bucles anidados para sumar los elementos de las dos matrices.

2.3 Salida de Datos

- **Matriz Resultado:**
    - Mostrar la matriz resultante de la suma.

3. Validaciones

- **Tamaño y Elementos de la Matriz:**
    - Debe ser un número entero positivo.
    - Los elementos deben ser numéricos.
    - Si se ingresan valores inválidos, mostrar un mensaje de error y solicitar nuevamente.

Ejercicio 6: Número perfecto

1. Enunciado

Implementar un programa que encuentre y muestre todos los números perfectos entre 1 y 10,000.

2. Requerimientos Funcionales

2.1 Proceso/Cálculo

- **Identificación de Números Perfectos:**
    - Utilizar un bucle para iterar y otro para encontrar los divisores de cada número.
    - Verificar si un número es perfecto (igual a la suma de sus divisores).

2.2 Salida de Datos

- **Números Perfectos Encontrados:**
    - Mostrar todos los números perfectos encontrados.

3. Validaciones

- **Rango de Búsqueda:**
    - Los números deben estar entre 1 y 10,000.
    - Si no se encuentran números perfectos, mostrar un mensaje informando.

Ejercicio 7: Matriz de espiral

1. Enunciado

Crear un programa que imprima una matriz cuadrada de tamaño n×nn \times nn×n en forma de espiral.

2. Requerimientos Funcionales

2.1 Entrada de Datos

- **Tamaño de la Matriz:**
    - El usuario debe ingresar el tamaño nnn de la matriz.

2.2 Proceso/Cálculo

- **Generación de la Matriz en Espiral:**
    - Utilizar bucles anidados para llenar la matriz en orden espiral.

2.3 Salida de Datos

- **Matriz en Espiral:**
    - Mostrar la matriz resultante.

3. Validaciones

- **Tamaño de la Matriz:**
    - Debe ser un número entero positivo.
    - Si se ingresa un valor inválido, mostrar un mensaje de error y solicitar nuevamente.

Ejercicio 8: Verificación de un número Armstrong

1. Enunciado

Escribir un programa que verifique si un número de nnn dígitos ingresado por el usuario es un número de Armstrong.

2. Requerimientos Funcionales

2.1 Entrada de Datos

- **Número a Verificar:**
    - El usuario debe ingresar un número entero.

2.2 Proceso/Cálculo

- **Verificación de Número de Armstrong:**
    - Utilizar un bucle para separar cada dígito, elevarlo a la potencia nnn y sumarlo.

2.3 Salida de Datos

- **Resultado de la Verificación:**
    - Mostrar si el número es un número de Armstrong o no.

3. Validaciones

- **Número a Verificar:**
    - Debe ser un entero positivo.
    - Si se ingresa un valor inválido, mostrar un mensaje de error y solicitar nuevamente.

Ejercicio 9: Cálculo de potencias usando multiplicación repetida

1. Enunciado

Crear un programa que calcule la potencia de un número usando multiplicación repetida.

2. Requerimientos Funcionales

2.1 Entrada de Datos

- **Base y Exponente:**
    - El usuario debe ingresar una base y un exponente, ambos valores enteros.

2.2 Proceso/Cálculo

- **Cálculo de Potencia:**
    - Utilizar un bucle para multiplicar la base por sí misma tantas veces como el exponente indique.

2.3 Salida de Datos

- **Resultado de la Potencia:**
    - Mostrar el resultado del cálculo de la potencia.

3. Validaciones

- **Base y Exponente:**
    - Ambos deben ser enteros.
    - El exponente debe ser un número no negativo.
    - Si se ingresan valores inválidos, mostrar un mensaje de error y solicitar nuevamente.