I.E.S.T.P
"Andrés Avelino Cáceres Dorregaray"

ESTRUCTURAS REPETITIVAS

Docente: Mg. Ing. Raúl Fernández Bejarano
Alumno: Brayan Stiven Quispe Huamán
Curso: Programación de Aplicaciones Web y Móviles
Semestre: VI

Diseño y Programación Web

Ejercicio 1: Sumatoria de números primos en un rango

Enunciado:

Escribe un programa que solicite dos números y calcule la sumatoria de los números primos que existen entre esos dos valores. Utiliza un bucle for o while para recorrer los números en el rango y verifica si son primos.

Requerimientos funcionales:

1. El programa debe solicitar dos números enteros como rango.
2. Verificar si los números en ese rango son primos.
3. Calcular la sumatoria de los números primos en el rango.
4. Mostrar el resultado de la suma.

Código:

```
import 'dart:io';

bool esPrimo(int num) {
  if (num < 2) return false;
  for (int i = 2; i <= num ~/ 2; i++) {
    if (num % i == 0) return false;
  }
  return true;
}

void main() {
  print('Ingrese el número inicial del rango:');
  int inicio = int.parse(stdin.readLineSync()!);

  print('Ingrese el número final del rango:');
  int fin = int.parse(stdin.readLineSync()!);

  int sumatoriaPrimos = 0;

  for (int i = inicio; i <= fin; i++) {
    if (esPrimo(i)) {
      sumatoriaPrimos += i;
    }
  }

  print('La sumatoria de los números primos en el rango es: $sumatoriaPrimos');
}
```

Ejercicio 2: Números de Fibonacci hasta N términos

Enunciado:

Implementa un programa que genere la secuencia de Fibonacci hasta un número n de términos ingresado por el usuario. Utiliza un bucle while o for para ir generando los números de la secuencia.

Requerimientos funcionales:

1. El programa debe solicitar al usuario el número de términos.
2. Generar la secuencia de Fibonacci hasta el número de términos especificado.
3. Mostrar la secuencia generada.

Código:

```
import 'dart:io';

void main() {
  print('Ingrese el número de términos de la secuencia de Fibonacci:');
  int n = int.parse(stdin.readLineSync()!);

  int a = 0, b = 1, temp;

  print('Secuencia de Fibonacci:');
  for (int i = 1; i <= n; i++) {
    print(a);
    temp = a + b;
    a = b;
    b = temp;
  }
}
```

Ejercicio 3: Factorial de números grandes

Enunciado:

Escribe un programa que calcule el factorial de un número grande (por ejemplo, 100) utilizando estructuras repetitivas y el tipo de datos BigInt para manejar grandes números.

Requerimientos funcionales:

1. El programa debe solicitar un número entero.
2. Calcular el factorial del número utilizando bucles.
3. Manejar números grandes con BigInt.
4. Mostrar el resultado del factorial.

Código:

```
import 'dart:io';


void main() {
  print('Ingrese un número para calcular su factorial:');
  int num = int.parse(stdin.readLineSync()!);

  BigInt factorial = BigInt.from(1);

  for (int i = 1; i <= num; i++) {
    factorial *= BigInt.from(i);
  }

  print('El factorial de $num es: $factorial');
}
```

Ejercicio 4: Inversión de un número

Enunciado:

Crea un programa que invierta los dígitos de un número entero ingresado por el usuario, utilizando un bucle while para extraer y reordenar los dígitos.

Requerimientos funcionales:

1. El programa debe solicitar un número entero.
2. Invertir los dígitos del número.
3. Mostrar el número invertido.

Código:

```
import 'dart:io';

void main() {
  print('Ingrese un número entero para invertir:');
  int num = int.parse(stdin.readLineSync()!);

  int invertido = 0;

  while (num != 0) {
    int digito = num % 10;
    invertido = invertido * 10 + digito;
    num ~/= 10;
  }

  print('El número invertido es: $invertido');
}
```

Ejercicio 5: Suma de matrices NxN

Enunciado:

Escribe un programa que solicite dos matrices de tamaño N x N (donde N es proporcionado por el usuario) y luego realice la suma de las dos matrices utilizando bucles anidados for.

Requerimientos funcionales:

1. El programa debe solicitar el tamaño de las matrices.
2. Pedir los valores de ambas matrices.
3. Sumar las matrices.
4. Mostrar la matriz resultante.

Código:

```
import 'dart:io';

void main() {
  print('Ingrese el tamaño de las matrices NxN:');
  int n = int.parse(stdin.readLineSync()!);

  List<List<int>> matriz1 = List.generate(n, (_) => List.filled(n, 0));
  List<List<int>> matriz2 = List.generate(n, (_) => List.filled(n, 0));
  List<List<int>> suma = List.generate(n, (_) => List.filled(n, 0));

  print('Ingrese los valores de la primera matriz:');
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
      matriz1[i][j] = int.parse(stdin.readLineSync()!);
    }
  }

  print('Ingrese los valores de la segunda matriz:');
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
```

```
    matriz2[i][j] = int.parse(stdin.readLineSync()!);
   }
  }

  for (int i = 0; i < n; i++) {
   for (int j = 0; j < n; j++) {
    suma[i][j] = matriz1[i][j] + matriz2[i][j];
   }
  }

  print('La suma de las matrices es:');
  for (int i = 0; i < n; i++) {
   print(suma[i]);
  }
}
```

Ejercicio 6: Número perfecto

Enunciado:

Implementa un programa que encuentre y muestre todos los números perfectos entre 1 y 10,000. Un número perfecto es aquel que es igual a la suma de sus divisores propios.

Requerimientos funcionales:

1. El programa debe recorrer los números entre 1 y 10,000.
2. Verificar si un número es perfecto.
3. Mostrar los números perfectos.

Código:

```
import 'dart:io';

bool esPerfecto(int num) {
  int suma = 0;
  for (int i = 1; i <= num ~/ 2; i++) {
   if (num % i == 0) {
    suma += i;
   }
  }
  return suma == num;
}

void main() {
  print('Números perfectos entre 1 y 10,000:');
  for (int i = 1; i <= 10000; i++) {
   if (esPerfecto(i)) {
    print(i);
   }
  }
}
```

Ejercicio 7: Matriz de espiral

Enunciado:

Crea un programa que imprima una matriz cuadrada de tamaño n x n en forma de espiral. Utiliza bucles anidados para recorrer las posiciones de la matriz en el orden adecuado.

Requerimientos funcionales:

1. El programa debe solicitar el tamaño `n` de la matriz.
2. Llenar la matriz en forma de espiral con números del 1 al n^2.
3. Mostrar la matriz resultante en forma de espiral.

Código:

```
import 'dart:io';

void main() {
  print('Ingrese el tamaño de la matriz (n):');
  int n = int.parse(stdin.readLineSync()!);

  List<List<int>> matriz = List.generate(n, (_) => List.filled(n, 0));

  int valor = 1;
  int inicioFila = 0, finFila = n - 1;
  int inicioCol = 0, finCol = n - 1;

  while (inicioFila <= finFila && inicioCol <= finCol) {
    // Llenar la fila superior
    for (int i = inicioCol; i <= finCol; i++) {
      matriz[inicioFila][i] = valor++;
    }
    inicioFila++;

    for (int i = inicioFila; i <= finFila; i++) {
      matriz[i][finCol] = valor++;
    }
    finCol--;

    if (inicioFila <= finFila) {
      for (int i = finCol; i >= inicioCol; i--) {
        matriz[finFila][i] = valor++;
      }
      finFila--;
    }

    if (inicioCol <= finCol) {
      for (int i = finFila; i >= inicioFila; i--) {
        matriz[i][inicioCol] = valor++;
      }
      inicioCol++;
    }
  }

  print('Matriz en forma de espiral:');
  for (List<int> fila in matriz) {
    print(fila);
  }
}
```

Ejercicio 8: Verificación de un número Armstrong

Enunciado:

Escribe un programa que verifique si un número de `n` dígitos ingresado por el usuario es un número de Armstrong (o narcisista). Utiliza un bucle `for` para separar y elevar cada dígito a la potencia correspondiente.

Requerimientos funcionales:

1. El programa debe solicitar un número.
2. Verificar si el número es Armstrong.
3. Mostrar el resultado.

Código:

```
import 'dart:io';

import 'dart:math';

bool esArmstrong(int num) {
  int sum = 0;
  int original = num;
  int n = num.toString().length;

  while (num != 0) {
    int digito = num % 10;
    sum += pow(digito, n).toInt();
    num ~/= 10;
  }

  return sum == original;
}

void main() {
  print('Ingrese un número para verificar si es un número Armstrong:');
  int num = int.parse(stdin.readLineSync()!);

  if (esArmstrong(num)) {
    print('$num es un número Armstrong.');
  } else {
    print('$num no es un número Armstrong.');
  }
}
```

Ejercicio 9: Cálculo de potencias usando multiplicación repetida

Enunciado:

Crea un programa que calcule la potencia de un número usando multiplicación repetida, es decir, sin utilizar la función Math.pow(). El programa debe solicitar una base y un exponente, y luego calcular la potencia utilizando un bucle while o for.

Requerimientos funcionales:

1. El programa debe solicitar una base y un exponente.
2. Calcular la potencia utilizando multiplicación repetida.
3. Mostrar el resultado.

Código:

```
import 'dart:io';


void main() {
  print('Ingrese la base:');
  int base = int.parse(stdin.readLineSync()!);
```

```
  print('Ingrese el exponente:');
  int exponente = int.parse(stdin.readLineSync()!);

  int resultado = 1;

  for (int i = 1; i <= exponente; i++) {
    resultado *= base;
  }

  print('El resultado de $base^$exponente es: $resultado');
}
```