UNIVERSIDAD DE LAS FUERZAS ARMADAS ESPE

Nombre: Guerrero Jair **NRC:** 29583

Fecha: 10/01/2026

Tema: Ejercicios de Cap. 3

1. EJERCICIO

Un número perfecto es un número natural que es igual a la suma de sus divisores propios positivos, sin incluirse el mismo. Por ejemplo, el 6 es un número perfecto, porque sus divisores propios son 1,2 y 3; y 6 = 1+2+3. El siguiente perfecto es 28 = 1+2+4+7+14.

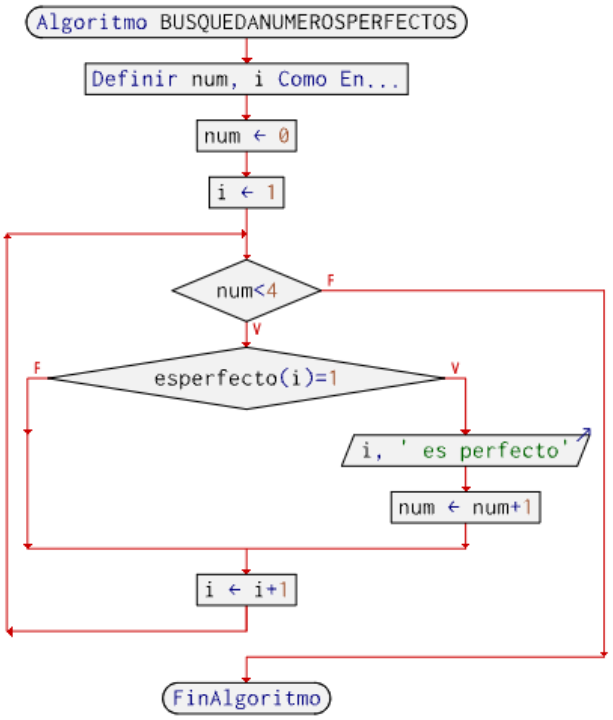
Se pide:

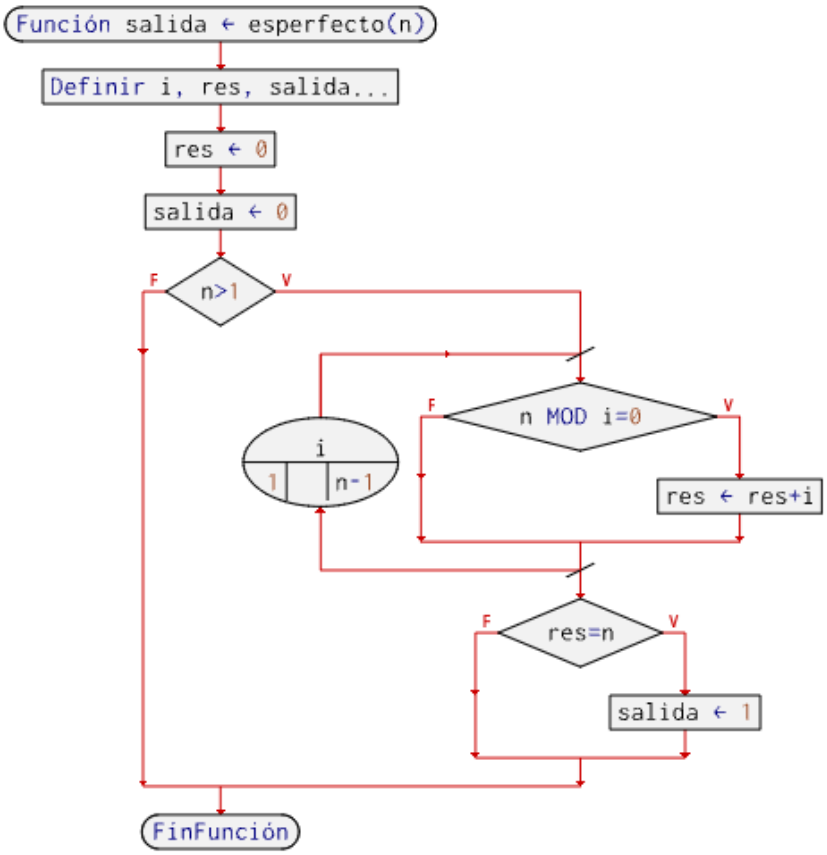
1. Programe la función **esperfecto**, que recibe como argumento un número natural y devuelve un valor 1 si el número es perfecto, y 0 en caso contrario.
    a Desarrolle la función principal del programa que calcule los 4 primeros números perfectos y los muestre por pantalla.

2. Tabla de objetos

Objeto	Nombre	Valor	Tipo
Cantidad de Perfectos	num	Variable	Entero
Iterador	i	Variable	Entero
Número a evaluar	n	Variable	Entero
Suma de divisores	res	Variable	Entero
Indicador de salida	salida	Variable	Entero
Función verificación	esperfecto	Función	Entero
Límite de perfectos	4	Constante	Entero
Valor Verdadero	1	Constante	Entero
Valor Falso	0	Constante	Entero

3. Diagrama de flujo en PSEINT


Algoritmo BUSQUEDANUMEROSPERFECTOS
Definir num, i Como En...
num ← 0
i ← 1
num<4
F
V
esperfecto(i)=1
F
V
i, ' es perfecto'
num ← num+1
i ← i+1
FinAlgoritmo

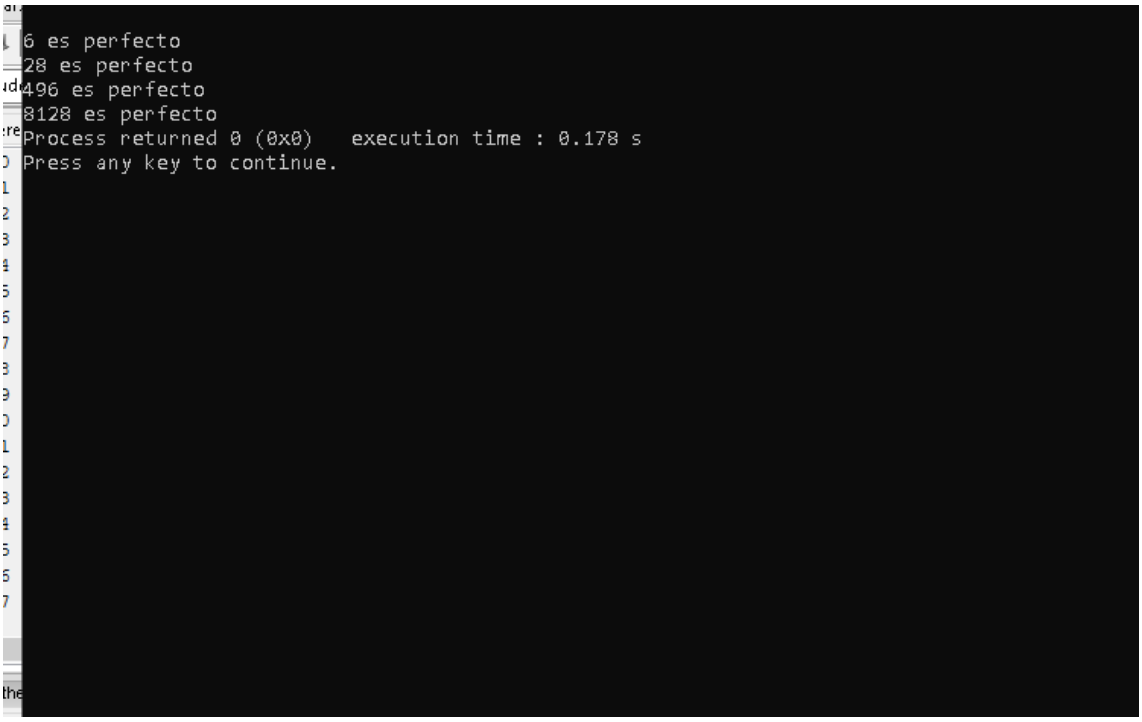

Función salida ← esperfecto(n)
Definir i, res, salida...
res ← 0
salida ← 0
n>1
F
V
i
1
n-1
n MOD i=0
F
V
res ← res+i
res=n
F
V
salida ← 1
FinFunción

4. Código en CODEBLOCKS

```
#include <stdio.h>

int es_numero_perfecto(int numero);

int main(void)
{
   int encontrados = 0;
   int candidato = 1;

   while (encontrados < 4)
   {
      if (es_numero_perfecto(candidato))
      {
         printf("%d es perfecto\n", candidato);
         encontrados++;
      }
      candidato++;
   }

   return 0;
}

int es_numero_perfecto(int n)
{
   int divisor, suma_divisores = 0;
```

```
    for (divisor = 1; divisor < n; divisor++)
    {
        if (n % divisor == 0)
        {
            suma_divisores += divisor;
        }
    }

    return (suma_divisores == n);
}
```

5. **Pantallazos de la ejecución**



6. **Conclusiones y recomendaciones**

- **Conclusión:** El uso de funciones resulta fundamental para segmentar el código, logrando que la operación matemática (como la validación de propiedades del número) sea independiente de la gestión de flujo en el cuerpo principal del programa.

- **Recomendación:** Es aconsejable implementar una variable de control o contador (por ejemplo, `num`) que permita finalizar automáticamente la ejecución del algoritmo en el momento exacto en que se alcancen los cuatro resultados requeridos.

2. EJERCICIO

1. Enunciado del ejercicio

Problema 3.1.2 Números primos.

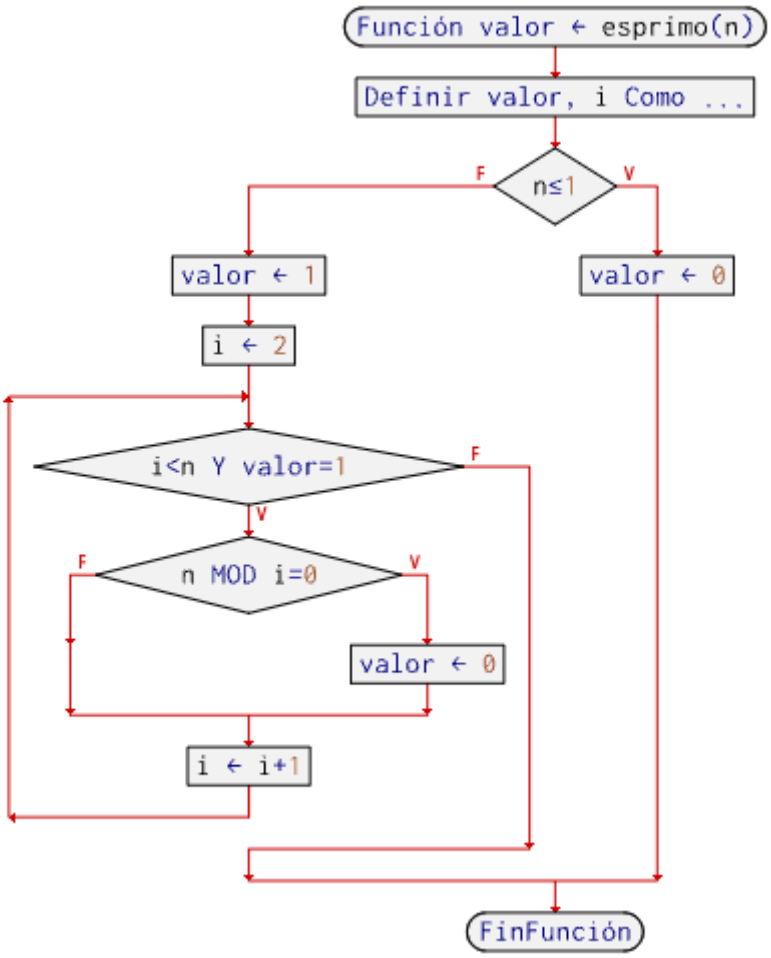
Realice un programa que resuelva adecuadamente los siguientes apartados:

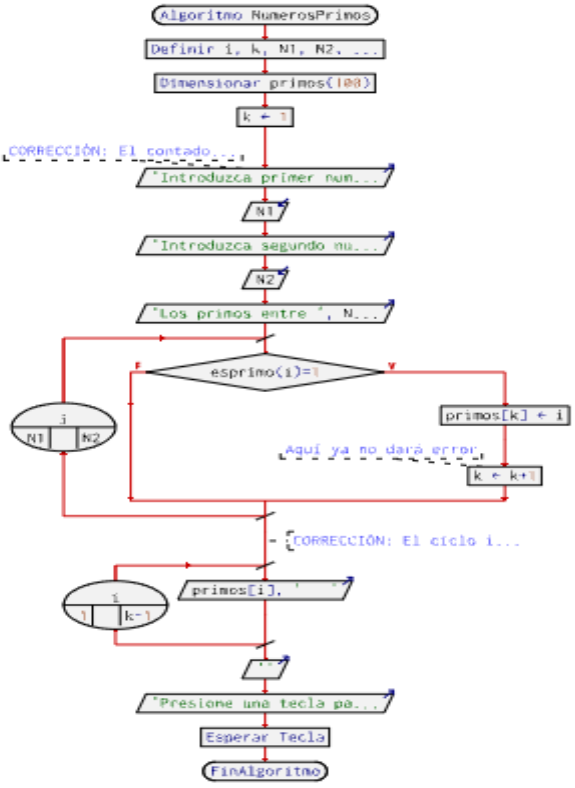
1. Programe la función esprimo, que recibe como argumento un número entero, y devuelve un valor 1 si el número es primo y 0 en caso contrario.
2. Almacene en un vector todos los números primos comprendidos entre dos números introducidos por teclado y luego imprima dicho vector.

2. Tabla de objetos

Objeto	Nombre	Valor	Tipo
Número a evaluar	n	Variable	Entero
Límite inferior	inicio	Variable	Entero
Límite superior	fin	Variable	Entero
Contenedor de primos	vector_primos	Primo	Arreglo (Enteros)

3. Diagrama de flujo en PSEINT

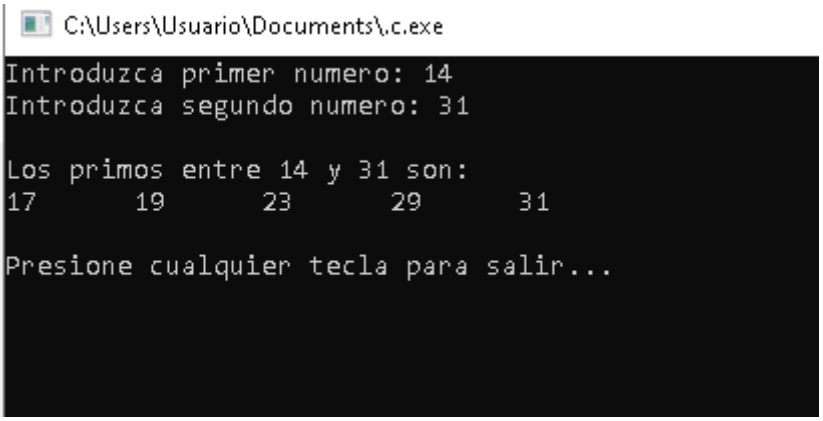

Función valor ← esprimo(n)
Definir valor, i Como ...
n≤1
F
V
valor ← 1
valor ← 0
i ← 2
i<n Y valor=1
F
V
n MOD i=0
F
V
valor ← 0
i ← i+1
FinFunción


Algoritmo NumerosPrimos
Definir i, k, N1, N2, ...
Dimensionar primos(100)
k ← 1
CORRECCIÓN: El contado...
"Introduzca primer num...
N1
"Introduzca segundo nu...
N2
"Los primos entre ", N...
esprimo(i)=1
F
V
primos[k] ← i
Aquí ya no dará error
k ← k+1
CORRECCIÓN: El ciclo i...
primos[i], " "
" "
"Presione una tecla pa...
Esperar Tecla
FinAlgoritmo

4. **Código en CODEBLOCKS**

```
#include <stdio.h>
#include <conio.h>

int esprimo(int n);

int main(void) {
   int i, k = 0, N1, N2, primo, primos[100];

   printf("Introduzca primer numero: "); scanf("%d", &N1);
   printf("Introduzca segundo numero: "); scanf("%d", &N2);
   printf("\nLos primos entre %d y %d son:\n", N1, N2);

   for(i = N1; i <= N2; i++)
      if(esprimo(i)) primos[k++] = i;

   for(i = 0; i < k; i++) printf("%d \t", primos[i]);

   printf("\n\nPresione cualquier tecla para salir...");
   getch(); return 0;
}

int esprimo(int n) {
   if (n <= 1) return 0;
   int primo = 1, i;
   for(i = 2; i < n && primo == 1; i++)
      if(n % i == 0) primo = 0;
   return primo;
}
```

5. **Pantallazos de la ejecución**



6. Conclusiones y recomendaciones

Conclusión: El empleo de funciones es fundamental para aislar el cálculo matemático (como verificar si un número cumple una propiedad) de la gestión operativa del programa, como llevar la cuenta de los hallazgos.

Recomendación: En funciones de comprobación, se sugiere integrar una variable lógica o "bandera" que detenga la búsqueda inmediatamente al encontrar una condición que invalide el resultado, evitando así ciclos innecesarios.

3.EJERCICIO

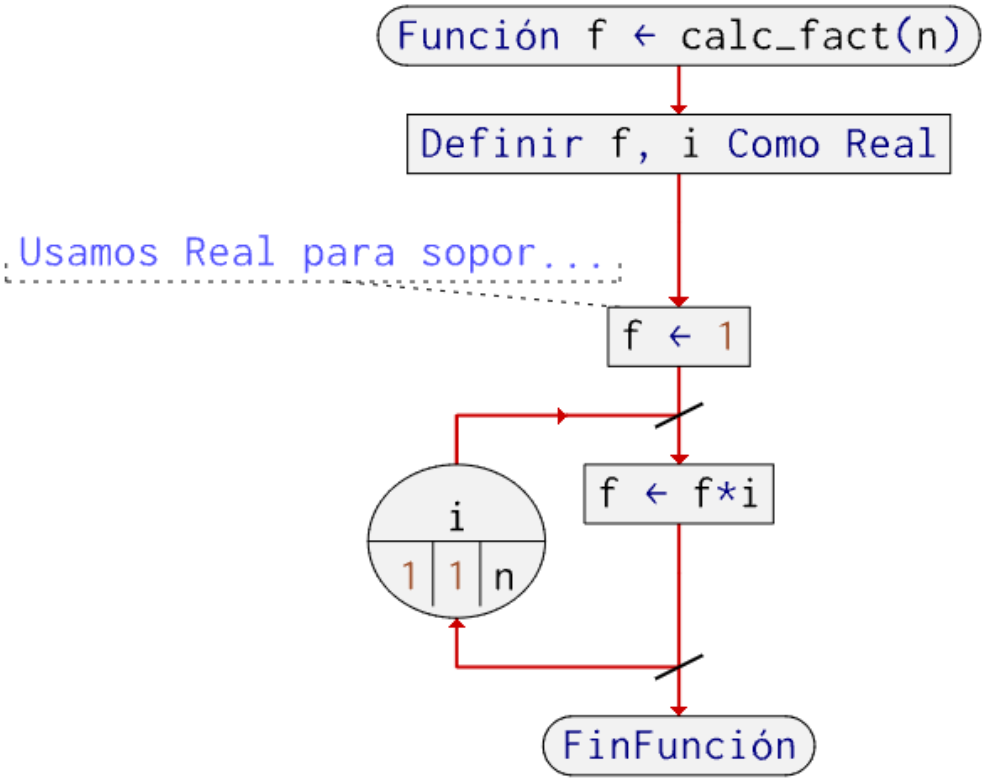
1. Enunciado del ejercicio

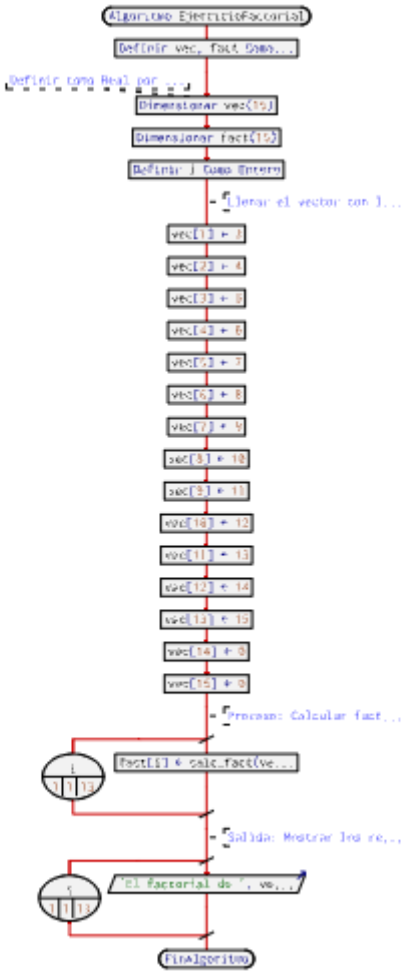
Programe la función calc_fact, que recibe como argumento un número entero, y devuelva el valor de la factorial. A continuación, use esta función en un programa que, dado un vector de 15 números enteros vec, calcule un vector fact con sus factoriales y lo muestre por pantalla.

2. Tabla de objetos

Objeto	Nombre	Valor	Tipo
Tamaño del vector	N	Constante	Entero
Índice de recorrido	I	Variable	Entero
Índice de multiplicación	J	Variable	Entero
Vector de números	Vec	Variable	Entero

3. Diagrama de flujo en PSEINT

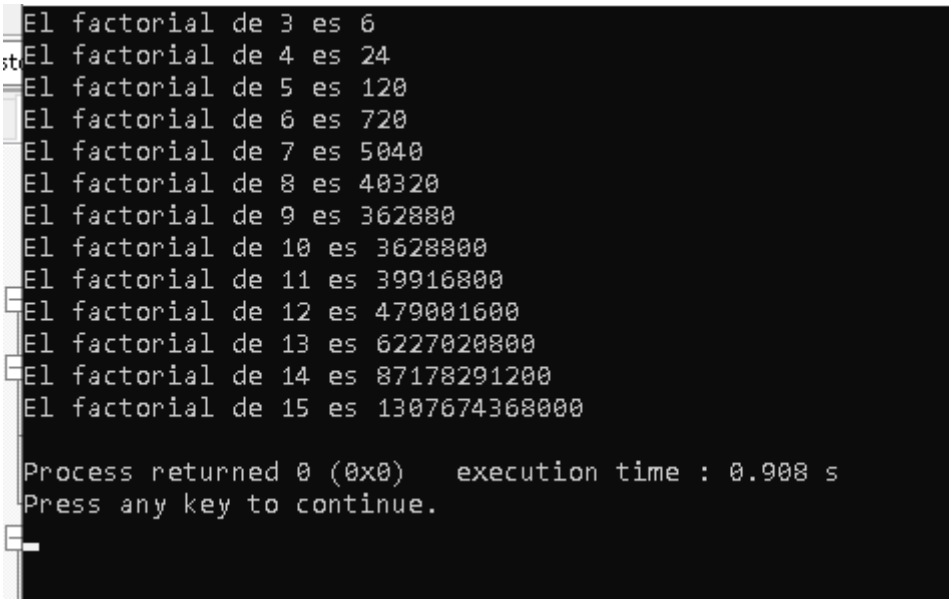

Función f ← calc_fact(n)
Definir f, i Como Real
Usamos Real para sopor...
f ← 1
i
1 1 n
f ← f*i
FinFunción


Algoritmo EjercicioFactorial
Definir vec, fact Como...
Definir como Real por ...
Dimensionar vec(15)
Dimensionar fact(15)
Definir i Como Entero
Llenar el vector con 1...
vec[1] ← 3
vec[2] ← 4
vec[3] ← 5
vec[4] ← 6
vec[5] ← 7
vec[6] ← 8
vec[7] ← 9
vec[8] ← 10
vec[9] ← 11
vec[10] ← 12
vec[11] ← 13
vec[12] ← 14
vec[13] ← 15
vec[14] ← 0
vec[15] ← 0
Proceso: Calcular fact...
fact[i] ← calc_fact(ve...
Salida: Mostrar los re...
"El factorial de ", ve...
FinAlgoritmo

4. Código en CODEBLOCKS

```
#include <iostream>

using namespace std;

long long calc_fact(int n) {
    long long factorial = 1;
    for(int i = 1; i <= n; i++) {
        factorial *= i;
    }
    return factorial;
}
int main() {
    int vec[15] = {3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 0, 0};
    long long fact[15];

    for(int i = 0; i < 13; i++) {
        fact[i] = calc_fact(vec[i]);
    }
    for(int i = 0; i < 13; i++) {
        cout << "El factorial de " << vec[i] << " es " << fact[i] << endl;
    }

    return 0;
}
```

5. Pantallazos de la ejecución



6. Conclusiones y recomendaciones

- **Conclusión:** Almacenar la información en estructuras tipo vector permite centralizar los hallazgos para desplegarlos de forma conjunta y ordenada al concluir el proceso.

- **Recomendación:** Para optimizar el rendimiento en algoritmos de búsqueda, es preferible emplear una variable lógica o "bandera" que interrumpa la ejecución apenas se confirme que un número no cumple con la condición, ahorrando ciclos de procesamiento.

4.EJERCICIO

1. Enunciado del ejercicio

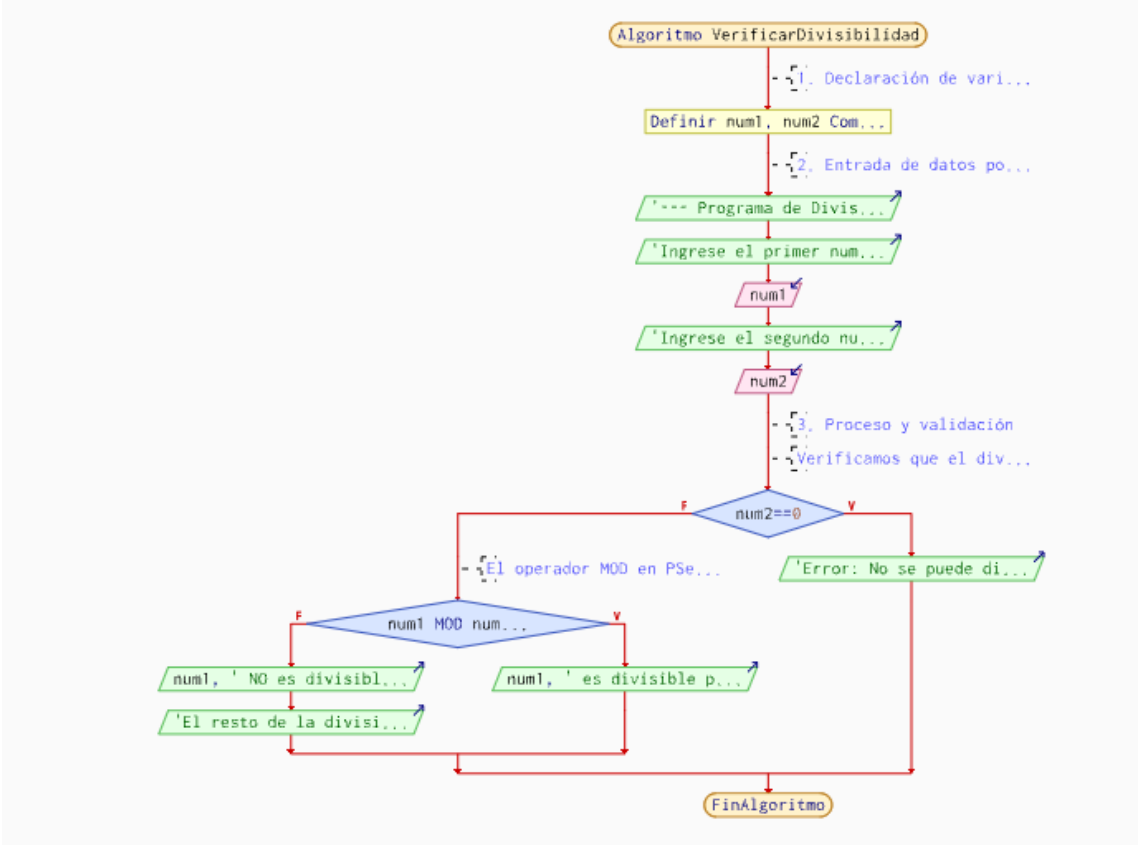
Desarrolle un programa que lea dos números enteros por teclado y determine si el primero de ello es divisible por el Segundo, Se mostrará en pantalla el resultado. Utilice el operador modulo % que devuelve el resto de la división.

2. Tabla de objetos

Objeto	Nombre	Valor (Naturaleza)	Tipo de Dato
Primer número	num1	Variable	Entero
Segundo número	num2	Variable	Entero

Operador de resto	%	Operador aritmético	Operación

3. Diagrama de flujo en PSEINT

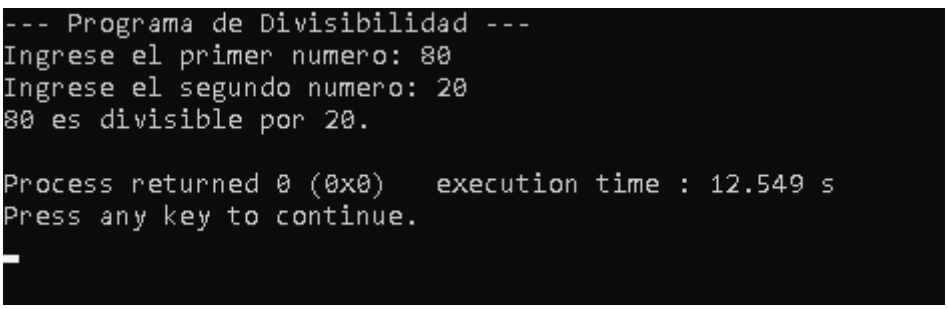


4. Código en CODEBLOCKS

```
#include <iostream>

using namespace std;

int main() {
    int num1, num2;

    cout << "--- Programa de Divisibilidad ---" << endl;
```

```
    cout << "Ingrese el primer numero: ";
    cin >> num1;
    cout << "Ingrese el segundo numero: ";
    cin >> num2;

    if (num2 == 0) {
        cout << "Error: No se puede dividir por cero." << endl;
    } else {
        // El operador % es la clave. Si el resultado es 0, es divisible.
        if (num1 % num2 == 0) {
            cout << num1 << " es divisible por " << num2 << "." << endl;
        } else {
            cout << num1 << " NO es divisible por " << num2 << "." << endl;
            cout << "El resto de la division es: " << (num1 % num2) << endl;
        }
    }

    return 0;
}
```

5. **Pantallazos de la ejecución**



6. **Conclusiones y recomendaciones**

- **Conclusión:** El empleo de funciones es vital para aislar los cálculos matemáticos complejos de la gestión operativa del flujo principal.

- **Recomendación:** Se sugiere integrar un contador de control específico para automatizar la detención del proceso una vez se alcance la cantidad exacta de resultados requeridos.