Reporte de la práctica 7 y 8

Práctica: Ciclos, Enum y Operador Condicional en C

Ramirez Gutiérrez Frida
Perea Rojas Giovanni
Zepahua Jimenez Ximena Alejandra

Introducción

En esta práctica aplicaremos las estructuras de control de flujo en lenguaje C para resolver problemas de complejidad intermedia. Reforzaremos los conceptos de los ciclos while, do...while y for, así como el uso de los tipos enumerados (enum) y la tríada condicional (?:).
El objetivo es que logremos diseñar algoritmos eficientes con estructuras anidadas y condiciones múltiples, aplicando lógica de programación y validaciones adecuadas.

Justificación

El dominio de las estructuras de repetición y control condicional es esencial en el desarrollo
de software. A través de los siguientes ejercicios, el estudiante consolidará su capacidad para
analizar y resolver problemas con decisiones y ciclos, fomentando la escritura estructurada y
el análisis lógico de algoritmos.
Además, la práctica busca fortalecer la comprensión de cómo combinar estructuras de
control para modelar comportamientos más complejos en programas reales.

Que es for:

Es un bucle que se repite un número conocido de veces.
Tiene tres partes: inicialización, condición y actualización.
Ejemplo:

```
for(int i = 0; i < 5; i++) {
    printf("%d\n", i);
}
```

Este imprime los números del 0 al 4

Que es while:

Es un bucle que se repite mientras se cumpla una condición.
La condición se evalúa antes de entrar al bucle.
Ejemplo:

```
int i = 0;
while(i < 5) {
    printf("%d\n", i);
    i++;
```

} Imprime igual del 0 al 4 pero la iniciación y actualización se hacen por separado.

Que es do- while:

Similar a while, pero la condición se evalúa después de ejecutar el bloque.
Esto garantiza que el bucle se ejecute al menos una vez.

Ejemplo:

```
int i = 0;
do {
    printf("%d\n", i);
    i++;
} while(i < 5);
```

Que es operador ternario ?:

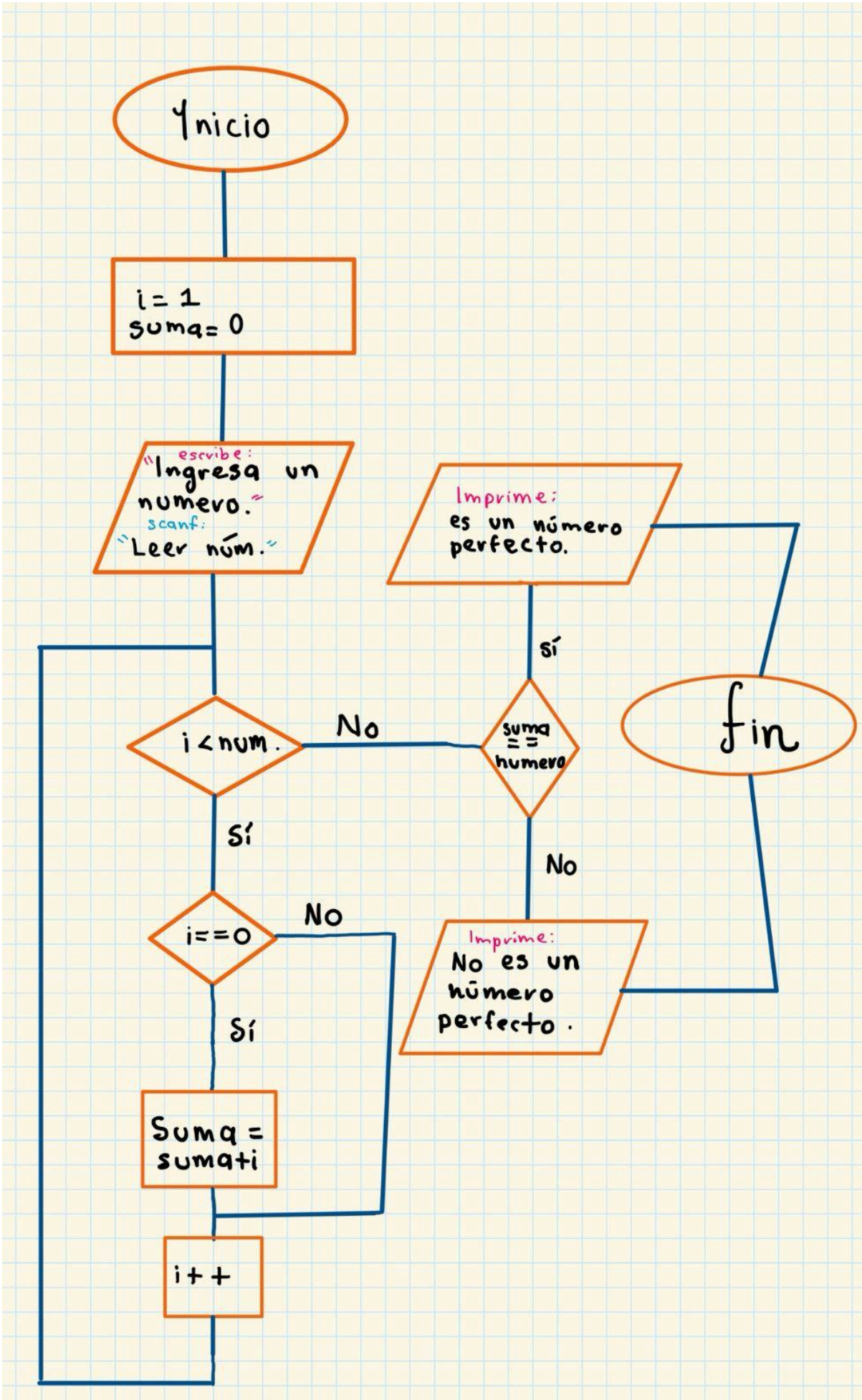
Es una forma corta de un if-else.
condicion ? valor_si_true : valor_si_false;
Permite asignar valores o ejecutar acciones en una sola línea.

Que es enum:

Es una enumeración, una forma de dar nombres simbólicos a números enteros.
Mejora la legibilidad del código.
No hace operaciones por sí mismo, sólo ayuda a recordar categorías.

NÚMERO PERFECTO:

```
#include <stdio.h>  // Biblioteca

int main() {
    int num, i = 1, suma = 0;  // Declaramos las variables:
                               // num = el número ingresado por el usuario
                               // i = contador que recorrerá los posibles divisores
                               // suma = acumula la suma de los divisores

    printf("Ingresa un número entero positivo: ");  // Pedimos un núm al usuario
    scanf("%d", &num);  // Leemos el número y lo guardamos en 'num'

    // Mientras 'i' sea menor que el número, verificamos si 'i' es divisor
    while (i < num) {
        // Si 'num' es divisible entre 'i' (num % i == 0), se suma 'i' a 'suma'
        // De lo contrario, se suma 0 Comprueba si el residuo de dividir num entre i es
igual a 0.
        suma += (num % i == 0) ? i : 0;
        i++;  // Aumentamos 'i' en 1 para probar el siguiente posible divisor
    }

    // Después del ciclo, comparamos si la suma de los divisores es igual al número
    // Si lo es, el número es perfecto. si no no lo es.
    (suma == num) ? printf("%d es un número perfecto.\n", num)
                  : printf("%d no es un número perfecto.\n", num);

    return 0;
}
```


Inicio
i = 1
suma = 0
escribe:
"Ingresa un numero."
scanf:
"Leer núm."
i < num.
No
Sí
i == 0
No
Sí
Suma = sumati
i++
suma == numero
Sí
No
Imprime:
es un número perfecto.
Imprime:
No es un número perfecto.
fin

FUNCIONAMIENTO: Con este programa buscamos clasificar el número que nos da el usuario de dos formas Perfecto y no perfecto, un número perfecto es aquel que la suma de sus divisores exactos (con residuo cero) son iguales al mismo número, excluyendo al propio ya que deben ser menores que el número. Por ejemplo el más pequeño es 6=1+2+3. 1,2 y 3 son menores que 6 y su suma da 6.

```
enum Tipo {PAR, IMPAR, PRIMO};

int main() {
    int num;
    int cuentaPar = 0, cuentaImpar = 0, cuentaPrimo = 0;
    int esPrimo;
    enum Tipo tipo;

    printf("Ingresa números (-999 para terminar):\n");

    while(1) { //pide numeros siempre hasta que ingresemos -999 solo en ese caso termina el mientras
        printf("Número: ");
        scanf("%d", &num);

        if(num == -999) break;

        // Verificar si es primo
        esPrimo = 1; // asumimos que es primo al iniciar hasta que se demuestre lo contrario
        if(num < 2) { // descarta 0 y al 1 pq un num primo es mayor o igual a dos
            esPrimo = 0;
        } else {
            for(int i = 2; i < num; i++){  // Revisa todos los posibles divisores entre 2 y num-1; si encuentra alguno, no es primo
                if(num % i == 0) {
                    esPrimo = 0; //pasa a ser falso
                    break;
                }
            }
        }
        // Clasificar usando enum
        if(num % 2 == 0) {
            tipo = PAR; // guardamos en la variable tipo que este num es par
            cuentaPar++; // aumentamos contador
```

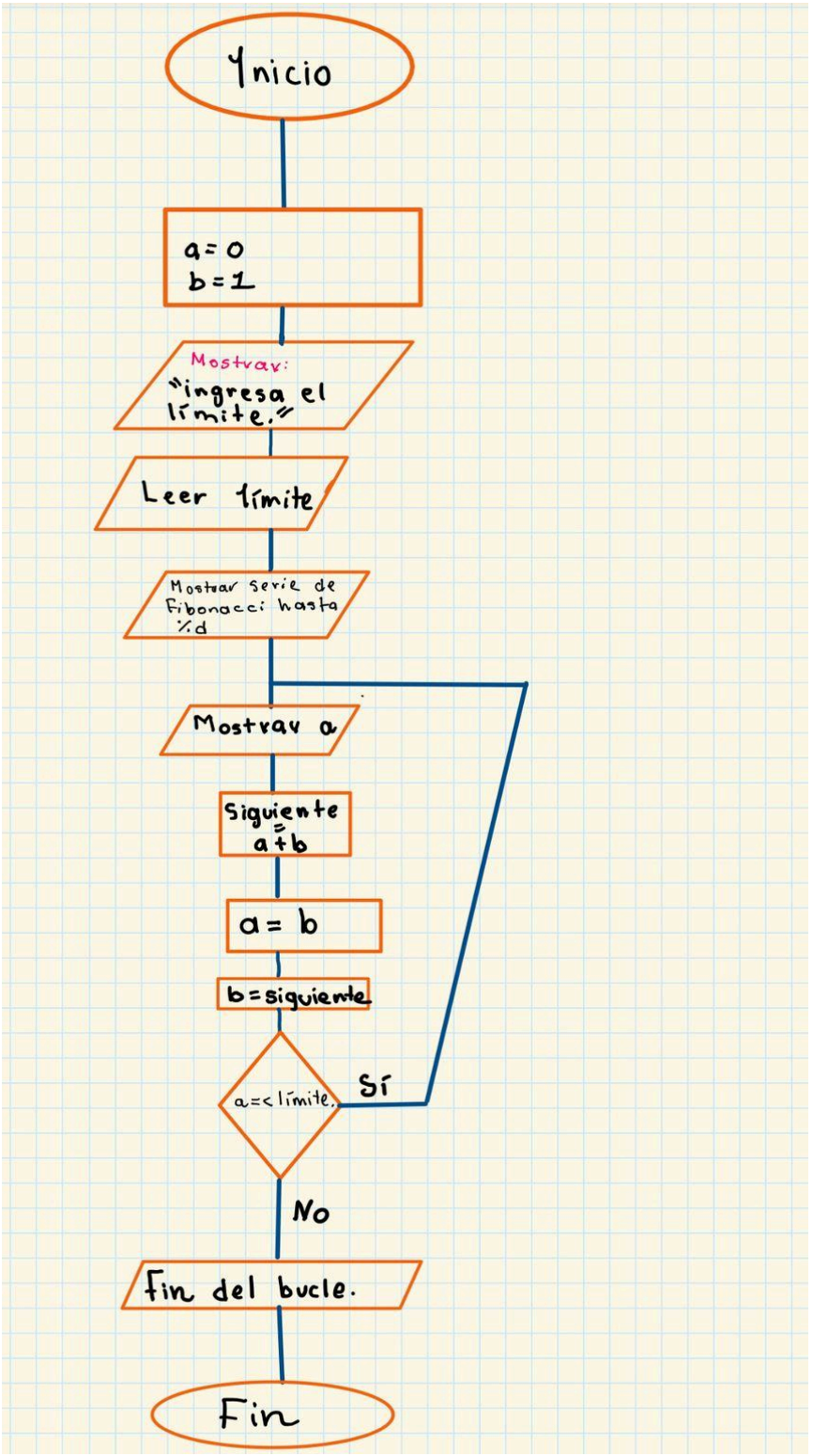
```
        } else {
            tipo = IMPAR; //sino guardamos en tipo que el num es impar
            cuentaImpar++; //aumentamos contador
        }

        if(esPrimo) {
            tipo = PRIMO; // se guarda en en tipo el valor primo del enum
            cuentaPrimo++;//aumenta el contador
        }
    }


    // Resultados
    printf("\nResultados finales:\n");
    printf("Números pares: %d\n", cuentaPar);
    printf("Números impares: %d\n", cuentaImpar);
    printf("Números primos: %d\n", cuentaPrimo);
```

Fibonacci

Serie de Fibonacci con límite. Generar la serie de Fibonacci hasta un límite definido por el usuario utilizando un do...while.

```
#include <stdio.h> //libreriaaa

int main() {
    int limite, a = 0, b = 1, siguiente; //variables limite= hasta que numero, a=0 b=1 primeros num de la serie

    printf("Ingresa el límite de la serie Fibonacci: ");
    scanf("%d", &limite); //se guarda en limite el num que ingresa el usuario

    printf("Serie de Fibonacci hasta %d:\n", limite); //imprimer el limite

    do {
        printf("%d ", a); //imprimer el valor inicial de a=0
        siguiente = a + b; // 0+1 =1
        a = b; //b le pasa el valor a a
        b = siguiente; //b toma el valor de la suma
    } while (a <= limite); // todo se ejecuta mientras a sea menor o igual que el limite

    printf("\n");
    return 0
```

;
}



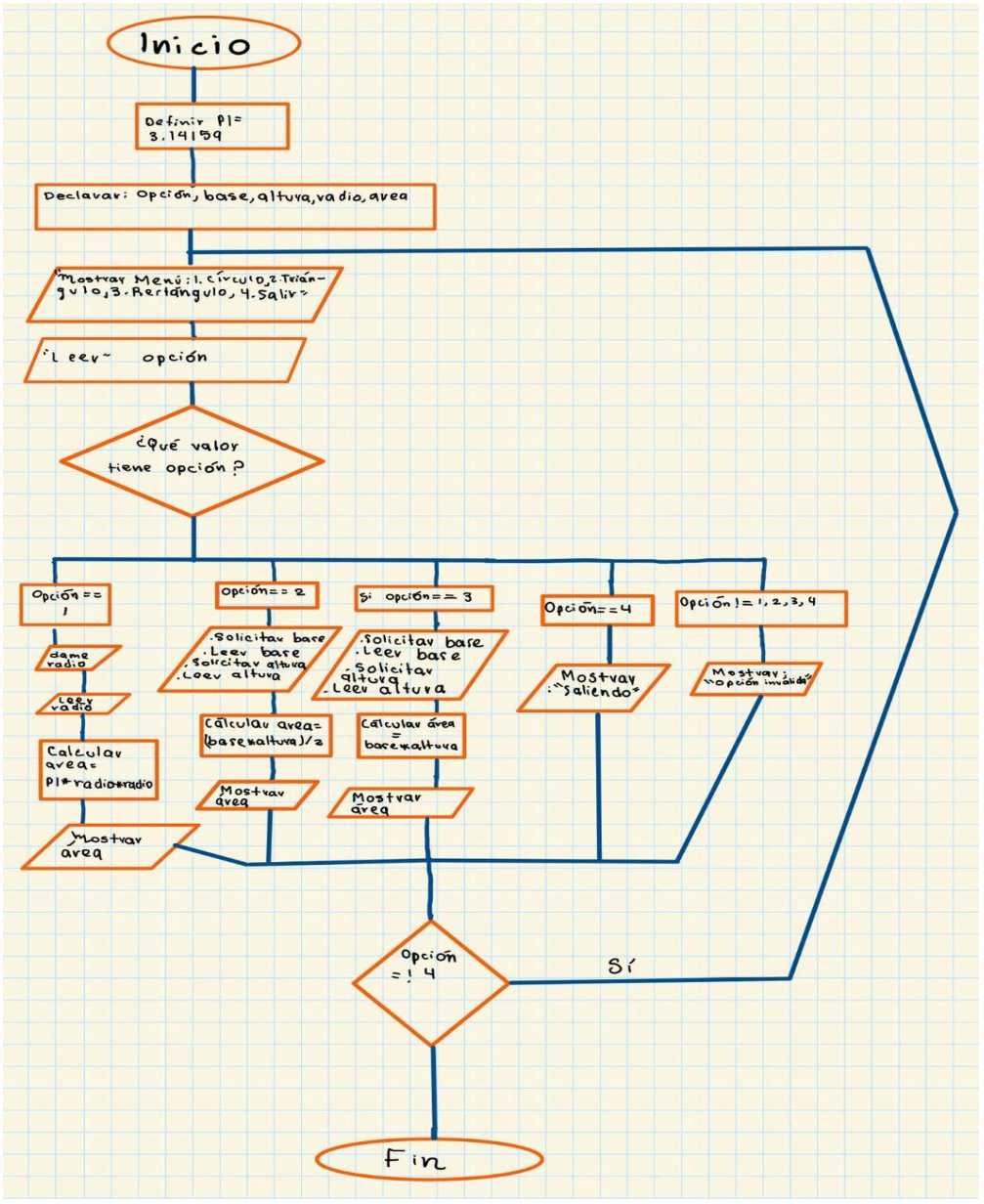
Explicación: Primero que nada la serie de fibonacci es una secuencia de números donde cada número es la suma de los dos anteriores, empezando generalmente con 0 y 1. El programa pide al usuario un límite de hasta dónde llegar y muestra los números de la serie de fibonacci hasta ese límite, empezando con 0 y , sumando los dos últimos números para generar el

siguiente imprime cada número hasta que “a” sea mayor que el límite ingresado.

ENUM menú

```
#include <stdio.h>
#define PI 3.14159  // Definimos PI porque la usaremos despues

int main() {
    enum Opciones {CIRCULO = 1, TRIANGULO, RECTANGULO, SALIR}; // Enum de opciones
    int opcion;
    float base, altura, radio, area; // para abarcar datos mas exactos

    do {
        // Menú
        printf("\n--- Calculadora de Areas ---\n");
        printf("1. Circulo\n");
        printf("2. Triangulo\n");
        printf("3. Rectangulo\n");
        printf("4. Salir\n");
        printf("Elige una opcion: ");
        scanf("%d", &opcion);

        switch(opcion) { //desplegamos el menu y dentro de cada caso sus operaciones
            case CIRCULO:
                printf("Ingresa el radio del circulo: ");
                scanf("%f", &radio);
                area = PI * radio * radio;
                printf("Area del circulo: %.2f\n", area);
                break;

            case TRIANGULO:
                printf("Ingresa la base del triangulo: ");
                scanf("%f", &base);
                printf("Ingresa la altura del triangulo: ");
                scanf("%f", &altura);
                area = (base * altura) / 2;
                printf("Area del triangulo: %.2f\n", area);
                break;

            case RECTANGULO:
                printf("Ingresa la base del rectangulo: ");
```

```
            scanf("%f", &base);
            printf("Ingresa la altura del rectangulo: ");
            scanf("%f", &altura);
            area = base * altura;
            printf("Area del rectangulo: %.2f\n", area);
            break;

        case SALIR:
            printf("Saliendo del programa...\n");
            break;

        default:
            printf("Opcion invalida. Intenta de nuevo.\n");
    }

} while(opcion != SALIR);

return 0;
}
```

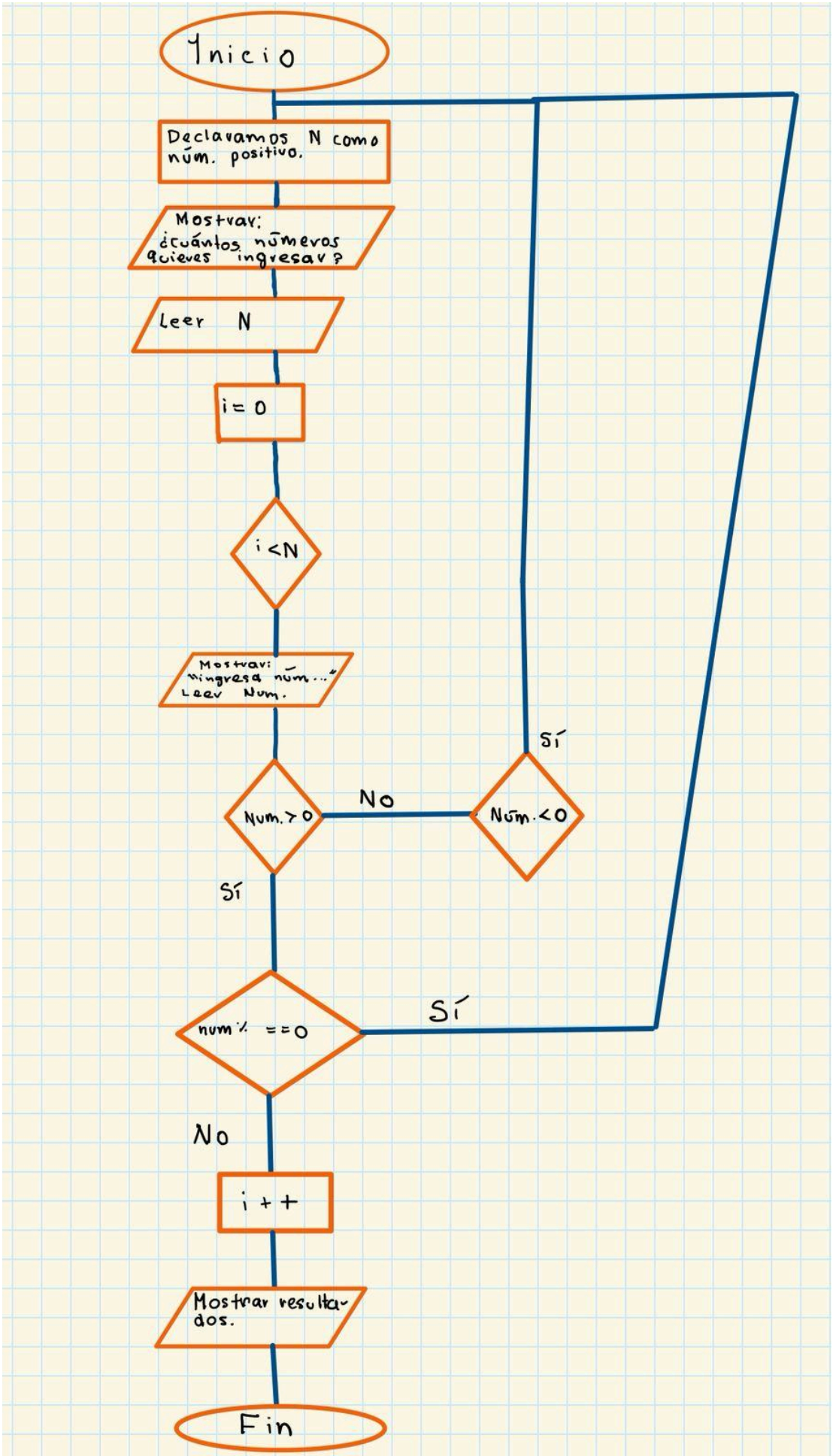


Explicación: Queremos una calculadora de áreas entonces lo hacemos con un switch case y en cada caso dependiendo de la figura ocupamos pedir datos como por ejemplo en el círculo pedimos el radio, y solo aplicamos una fórmula (pi*radio*radio) en el triángulo pedimos la base y la altura (base*altura)/2 y en el rectángulo que pedimos base y altura también (base*altura). Aquí el enum nos sirve para que cambiemos los números por nombres y se nos sea más fácil de leer. Círculo=1, triangulo=2, rectángulo=3 y salir=4.

Estadísticas de números.

```
#include <stdio.h>

int main() {
    int N, num;
    int positivos = 0, negativos = 0, pares = 0;

    printf("¿Cuántos números quieres ingresar? ");
    scanf("%d", &N);

    for(int i = 0; i < N; i++) {
        printf("Ingresa el número %d: ", i + 1); //pide todos los num que haya querido
ingresar
        scanf("%d", &num);

        // Contar positivos o negativos
        (num > 0) ? positivos++ : (num < 0 ? negativos++ : 0); /* Se evalua el num y se
agrega un "punto"
         dependiendo que en que concicion entre */

        // Contar pares
        (num % 2 == 0) ? pares++ : 0; // si es par se añade un "punto " a pares siempre
y cuando el residuo sea 0 al dividir entre dos
    }

    printf("\nResultados:\n");
    printf("Positivos: %d\n", positivos);
    printf("Negativos: %d\n", negativos);
    printf("Pares: %d\n", pares);
```

```
    return 0;+
}
```

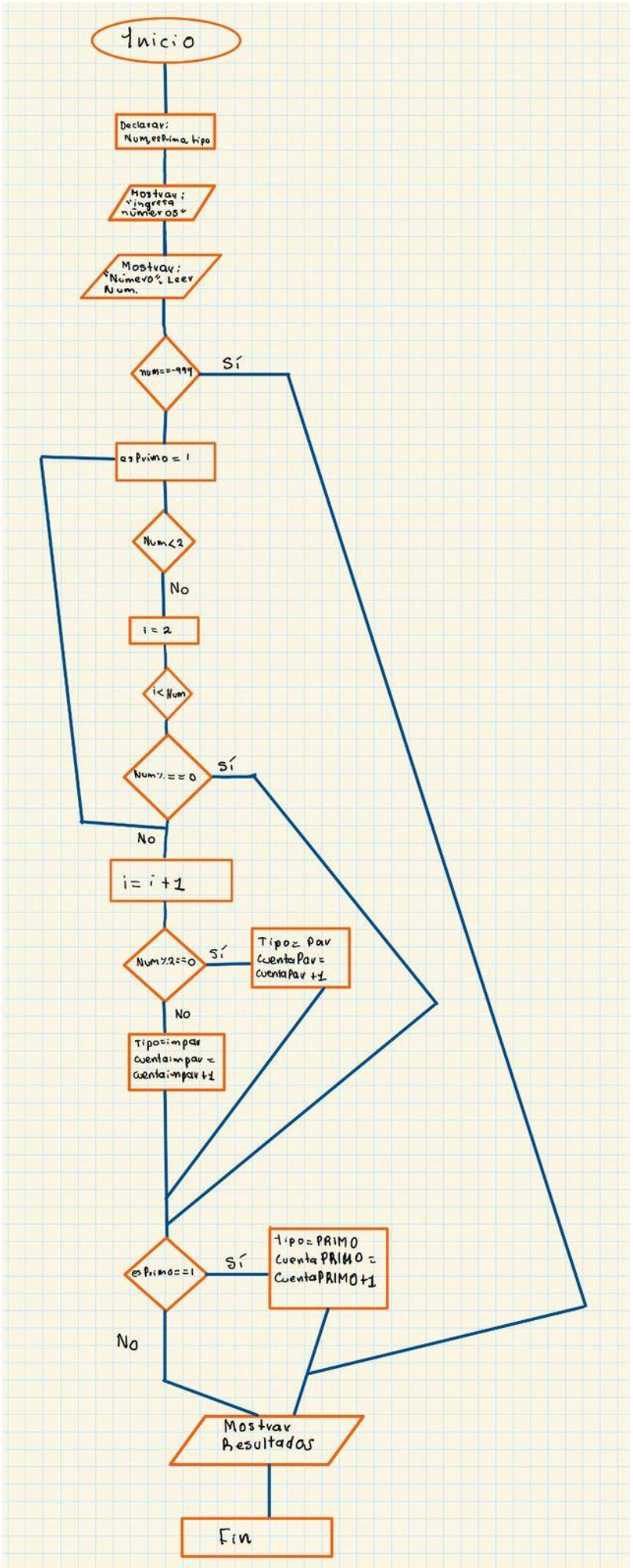

Inicio
Declaramos N como núm. positivo.
Mostrar: ¿cuántos números quieres ingresar?
Leer N
i = 0
i < N
Mostrar: "ingresa núm..."
Leer Num.
Num. > 0
No
Sí
Num. < 0
Sí
num % == 0
Sí
No
i ++
Mostrar resultados.
Fin

Explicación: Se pide la cantidad de números que el usuario va a meter después con un for pide todos, con una ternaria se evalúa si los números entran en una condición u otra y dependiendo de eso se le va sumando en el contador de positivos num>0 o negativos num<0 , lo mismo con los pares si al dividir entre 2 el residuo es cero se le suma al contador pares. De esta forma se clasifican los números de una lista que el usuario proporcione.

Clasificación por tipo numérico. Leer números hasta que se introduzca -999, clasificar cada uno como par, impar o primo (usando enum), y mostrar las sumas por categoría.

```
#include <stdio.h>

enum Tipo {PAR, IMPAR, PRIMO};

int main() {
    int num;
    int cuentaPar = 0, cuentaImpar = 0, cuentaPrimo = 0;
    int esPrimo;
    enum Tipo tipo;

    printf("Ingresa números (-999 para terminar):\n");

    while(1) { //pide numeros siempre hasta que ingresemos -999 solo en ese caso termina el mientras
        printf("Número: ");
        scanf("%d", &num);

        if(num == -999) break;

        // Verificar si es primo
        esPrimo = 1; // asumimos que es primo al iniciar hasta que se demuestre lo contrario
        if(num < 2) { // descarta 0 y al 1 pq un num primo es mayor o igual a dos
            esPrimo = 0;
        } else {
            for(int i = 2; i < num; i++){  // Revisa todos los posibles divisores entre 2 y num-1; si encuentra alguno, no es primo
                if(num % i == 0) {
                    esPrimo = 0; //pasa a ser falso
                    break;
                }
            }
        }
        // Clasificar usando enum
        if(num % 2 == 0) {
            tipo = PAR; // guardamos en la variable tipo que este num es par
            cuentaPar++; // aumentamos contador
        } else {
            tipo = IMPAR; //sino guardamos en tipo que el num es impar
```

```
            cuentaImpar++; //aumentamos contador
        }

        if(esPrimo) {
            tipo = PRIMO; // se guarda en en tipo el valor primo del enum
            cuentaPrimo++;//aumenta el contador
        }
    }


    // Resultados
    printf("\nResultados finales:\n");
    printf("Números pares: %d\n", cuentaPar);
    printf("Números impares: %d\n", cuentaImpar);
    printf("Números primos: %d\n", cuentaPrimo);

    return 0;
}
```


Inicio
Declarar: NumesPrima, tipo
Mostrar: "ingresa números"
Mostrar: "Número" % Leer Num.
num==-999
Sí
esPrimo = 1
Num<2
No
i = 2
i<Num
Num%i==0
Sí
No
i= i +1
Num%2==0
Sí
Tipo= Par
CuentaPar= CuentaPar +1
No
Tipo=impar
cuentaimpar = cuentaimpar+1
esPrimo==1
Sí
Tipo=PRIMO
CuentaPRIMO = CuentaPRIMO+1
No
Mostrar Resultados
Fin

FUNCIONAMIENTO:

El programa pide números al usuario hasta que ingrese -999, hay tres clasificaciones PAR, IMPAR Y PRIMO, aumenta el contador correspondiente dependiendo de donde se clasifique el número, al final muestra cuantos numero ingresaste de cada tipo el enum sirve para identificar el tipo del último número leído. Un número par se clasifica si es divisible entre 2, en caso contrario se clasifica como impar y un número primo si sus divisores son unicamente 1

CONCLUSIONES:

XIMENA ZEPAHUA:

NÚMERO PERFECTO:

Aprendí antes que nada el concepto de que es un número perfecto, reforcé el uso del while y también comprendí mejor el uso de la ternaria porque es como un if else donde si no pasa lo de la izquierda se cumple lo de la derecha pero en una sola línea, de dificultades me surgió a la hora de pasar la idea a código, aun me falta practicar varias cosas porque no suelo recordar todo entonces debo investigar cómo se pone X cosa. Una de las dificultades que tuve fue pasar la idea al código, pues todavía necesito practicar más para recordar ciertas estructuras y a veces debo investigar cómo escribir determinadas cosas.

En cuanto a posibles mejoras, considero que lo principal es seguir practicando para poder llevar mis ideas al código con mayor facilidad.

FIBONACCI:

Comprendí lo que es la serie de Fibonacci que es una secuencias de números donde cada número es la suma de los dos anteriores, lleve a cabo el uso del Do while donde en el do ejecuta las operaciones al menos una vez y, después, el while evalúa la condición para determinar si el ciclo se repite. Si la condición no se cumple, el programa sale del ciclo y no vuelve a ejecutar el do. Así que reforcé esta estructura en la serie de Fibonacci, de dificultades me encontré con pocas que solucionamos al momento de investigar conceptos en internet sobre lo aplicado otra alternativa pudo ser usar el while común.

MENÚ:

Aquí aprendí a usar el enum que cambia los números por las palabras que yo le otorgue haciéndolo más entendible. El switch era donde estaba puesto cada caso Círculo, Triángulo y Rectángulo que nos ayuda a manejar varias opciones. La verdad este si estuvo muy sencillo porque habíamos hecho algo parecido en Pseint con una calculadora y dependiendo el caso que el usuario quiera es el área de la figura. No encontré gran dificultad al realizar este programa ya que casi todo dentro de los casos eran las fórmulas del área. En cuanto a mejoras podría ser que usaramos funciones para las áreas de cada figura y en los case imprimirlas.

ESTADÍSTICAS DE NÚMEROS:

Aquí usamos el for principalmente porque teníamos que repetir el pedir números hasta llegar al límite que diera el usuario. Me sirvió bastante el ejercicio ya que comprendí que el for sigue hasta que i<límite también usamos aquí la ternaria en lugar de un if else para clasificar a los números en positivo, negativos o pares haciendo que los contadores aumentan si se agrega uno a alguna categoría.

En cuanto a las dificultades fue principalmente al poner la idea en código porque tuvimos que consultar datos en internet para hacerlo de la mejor manera posible. En cuanto a mejoras propondría clasificar al cero como un contador porque no se clasifica ni como positivo ni negativo.

CLASIFICACIÓN POR TIPO NUMÉRICO:

Aquí volvimos a usar el enum para asignarle a números palabras PAR, IMPAR o PRIMO y recordar con más facilidad cada caso. Usamos un while para pedir números hasta que el usuario meta -999 hicimos uso de for para encontrar divisores de los números ingresados donde sí había varios divisores era no primo y otro for para evaluar si era par el número. El enum aquí registra la categoría a la que pertenece el número para que el código sea más entendible. Como mejora podríamos importar otra librería que tenga booleanos, eso podría ser otra alternativa. No tuvimos gran inconveniente en este programa quizá el mismo problema de llevar la idea al código pero eso se irá mejorando con la práctica.

FRIDA RAMÍREZ

Número perfecto: Con este ejercicio comprendí el uso del ciclo **while** para realizar repeticiones controladas y la aplicación en otras situaciones. También comprendí un poco más cómo obtener y sumar los divisores propios de un número para determinar si es perfecto.

Serie de Fibonacci: Con este ejercicio aprendí un poco más sobre cómo funciona el ciclo do… while y cómo se puede usar para generar la serie de Fibonacci hasta un cierto límite. Me ayudó a entender mejor cómo se van sumando los números anteriores para formar el siguiente.

Menú interactivo: Con este ejercicio empecé a conocer cómo se usa el ***enum*** y el ***switch*** para hacer un menú con diferentes opciones. Sigue siendo confuso, pero poco a poco lo voy entendiendo.

Estadísticas de números: Acá aprendí un poco más sobre cómo usar el ciclo **for** para repetir instrucciones varias veces y también cómo usar las expresiones condicionales *? :* para decidir si un número es positivo, negativo o par.

Clasificación por tipo numérico: Este ejercicio me sirvió para intentar entender cómo se pueden clasificar los números como pares, impares o primos usando ***enum***. Aprendí un poco sobre cómo hacer que el programa se repita hasta que se escriba el número -999. En general, me cuesta un poco seguir la lógica del programa, pero con más práctica espero poder entenderlo mejor.

GIOVANNI PEREA

Número perfecto:

Al desarrollar el programa para determinar si un número es perfecto utilizando un ciclo *while* y la triada condicional, pude reforzar el entendimiento de cómo recorrer un rango de valores y acumular resultados según una condición lógica. Comprendí mejor la relación entre la lógica matemática y su representación en código, y cómo las decisiones secuenciales dentro de un bucle conducen a un resultado final correcto. En general, el ejercicio me ayudó a afianzar el razonamiento algorítmico y la aplicación práctica de estructuras básicas de control en la programación.

Fibonacci:

Al realizar el programa para generar la serie de Fibonacci con un límite definido por el usuario, utilizando la estructura *do...while*, pude comprender mejor cómo controlar la ejecución de un ciclo que se repite al menos una vez. La lógica de ir sumando los dos números anteriores para obtener el siguiente término me ayudó a fortalecer mi razonamiento secuencial. También me permitió ver que el proceso se ejecutara correctamente antes de verificar la condición de límite. En general, este ejercicio reforzó mi comprensión sobre la generación de series numéricas y el manejo de ciclos controlados por condiciones lógicas en programación.

Menú interactivo:

Aquí pude comprender mejor cómo organizar las opciones de un programa de manera clara y estructurada. El uso del **enum** *nos* facilitó la asignación de valores simbólicos a cada figura geométrica, haciendo el código más legible y fácil de mantener. Con el *switch*, logramos manejar las distintas operaciones para calcular las áreas según la opción elegida por el usuario, aplicando fórmulas diferentes de manera ordenada.

Estadísticas:
Al realizar el programa para leer N números y determinar cuántos son positivos, negativos y pares usando un ciclo ***for*** y expresiones condicionales, pude practicar cómo recorrer una serie de datos y aplicar condiciones para clasificarlos. Aunque en general comprendí cómo funcionan los operadores ternarios (`?:`) para simplificar las decisiones dentro del código, todavía me surge algo de duda sobre cómo combinarlos cuando hay varias condiciones al mismo tiempo. Este ejercicio me ayudó a entender mejor la relación entre ciclos, condiciones y acumuladores, y me permitió ver cómo se puede organizar el código para contar distintos tipos de números sin necesidad de muchas líneas.

Clasificación por tipo numérico:

Al hacer el programa para clasificar números como par, impar o primo usando ***enum***, comprendí cómo se puede organizar la información y contar valores por

categoría. Entiendo la lógica de identificar cada tipo de número y acumular sus sumas, todavía me cuesta un poco calcular primos de manera fácil. Este ejercicio me ayudó a practicar ciclos con condición de parada y el uso de *enum* para que el código sea más claro.