

Table of Contents

Introducción a la algoritmia	2
Algebra Booleana	4
Pseudocódigo	7
Instrucciones de los algoritmos	10
Operadores de los algoritmos	13
Diagramas de flujo	15
Elementos de un diagrama de flujo	17
La condicional	21
Ejemplo de uso de Condicionales	26
En caso de (switch)	31
Ejemplo de en caso de (switch)	33
Estructuras Repetitivas	38
Ejemplos de Bucles	43

Introducción a la algoritmia

La algoritmia es una disciplina que estudia los algoritmos, es decir, los procedimientos sistemáticos para resolver problemas. Un algoritmo es una secuencia finita de pasos que, partiendo de unos datos de entrada, produce unos datos de salida. Los algoritmos son la base de la informática y de la programación, y son fundamentales en la resolución de problemas en la vida cotidiana.

Características de los algoritmos

Los algoritmos tienen las siguientes características:

- **Preciso:** Los algoritmos deben ser precisos y no ambiguos, es decir, deben describir claramente los pasos a seguir para resolver un problema.
- **Finito:** Los algoritmos deben ser finitos, es decir, deben terminar en un número finito de pasos.
- **Definido:** Los algoritmos deben ser definidos, es decir, deben tener un número finito de pasos bien definidos.
- **Entrada:** Los algoritmos deben tener unos datos de entrada, que son los datos con los que se inicia el proceso.
- **Salida:** Los algoritmos deben tener unos datos de salida, que son los resultados obtenidos al final del proceso.

Elementos de un algoritmo

Los elementos básicos de un algoritmo son los siguientes:

- **Entrada de datos:** Los datos de entrada son los valores que se utilizan para iniciar el proceso. Por ejemplo, si queremos calcular el área de un círculo, el radio sería un dato de entrada.
- **Proceso:** El proceso es la secuencia de pasos que se deben seguir para resolver el problema. Por ejemplo, para calcular el área de un círculo, se debe multiplicar el radio por sí mismo y por el valor de pi.

- ** Salida de datos **: Los datos de salida son los resultados obtenidos al final del proceso. Por ejemplo, el área calculada del círculo sería un dato de salida.

Ejemplo de algoritmo

A continuación, se muestra un ejemplo de un algoritmo que calcula el área de un círculo a partir de su radio:

Nombre del algoritmo: Calcular área de un círculo

Definición de constantes:

Real: PI = 3.1416

Definición de variables:

Real: radio, area

Algoritmo:

1. Inicio
2. Escribir "Ingrese el radio del círculo:"
3. Leer radio
4. Hacer area = PI * radio * radio
5. Escribir "El área del círculo es:", area
6. Fin

En este ejemplo, PI es una constante que representa el valor de pi, radio es la variable que almacena el radio del círculo y area es la variable que almacena el área calculada. Las instrucciones Escribir y Leer se utilizan para mostrar mensajes en pantalla y leer valores del usuario, respectivamente.

Algebra Booleana

La álgebra booleana es una rama de la matemática y la lógica que trata de las operaciones lógicas y aritméticas en los valores binarios 0 y 1. Fue inventada por George Boole en el siglo XIX como un sistema de lógica para modelar el razonamiento humano. La álgebra booleana es fundamental para la informática y la electrónica digital, ya que se utiliza para diseñar circuitos digitales y programación de computadoras.

Operaciones básicas

Las operaciones básicas de la álgebra booleana son las siguientes:

Operación	Descripción	Símbolo
Negación	Invierte el valor de una variable	!
Conjunción	Devuelve verdadero si ambas variables son verdaderas	&&
Disyunción	Devuelve verdadero si al menos una de las variables es verdadera	

Leyes de la álgebra booleana

Las leyes del álgebra booleana son reglas que se utilizan para simplificar y manipular expresiones booleanas. Las principales leyes del álgebra booleana son las siguientes:

1. Ley de la identidad: $A + 0 = A$ y $A * 1 = A$
2. Ley de la dominancia: $A + 1 = 1$ y $A * 0 = 0$
3. Ley de la idempotencia: $A + A = A$ y $A * A = A$
4. Ley de la complementación: $A + !A = 1$ y $A * !A = 0$
5. Ley de la absorción: $A + A * B = A$ y $A * (A + B) = A$

6. Ley de la distribución: $A * (B + C) = A * B + A * C$ y $A + B * C = (A + B) * (A + C)$

7. Ley de De Morgan: $!(A * B) = !A + !B$ y $!(A + B) = !A * !B$

Tablas de verdad

Una tabla de verdad es una representación visual de todas las posibles combinaciones de valores de las variables en una expresión booleana y el resultado de la operación.

La tabla de verdad general es la siguiente:

A	B	A && B	A B	!A
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	0

Aplicación en algoritmos

La álgebra booleana se utiliza en algoritmos para realizar operaciones lógicas y de comparación. Por ejemplo, en un algoritmo de búsqueda se puede utilizar la operación de conjunción para verificar si dos condiciones son verdaderas

1. Inicio
2. Leer x, y
3. Si $x > 0 \&\& y > 0$
4. Escribir "Ambos números son positivos"
5. Sino
6. Escribir "Al menos uno de los números es negativo"
7. Fin

En este ejemplo, la operación $x > 0 \&\& y > 0$ verifica si tanto x como y son mayores que cero, y en caso afirmativo muestra un mensaje en pantalla. Si al menos uno de los

números es negativo, se muestra otro mensaje.

El álgebra booleana es una herramienta poderosa para la programación y la lógica, ya que permite realizar operaciones lógicas y de comparación de manera eficiente y precisa.

Pseudocódigo

El pseudocódigo es una forma de representar algoritmos de manera que sean fáciles de entender y de traducir a un lenguaje de programación. Aunque no es un lenguaje de programación en sí mismo, el pseudocódigo es una herramienta muy útil para planificar y diseñar programas antes de escribir el código en un lenguaje de programación específico.

Estructura del pseudocódigo

El pseudocódigo se compone de una serie de instrucciones que describen paso a paso lo que debe hacer un programa. Estas instrucciones se organizan en bloques de código que representan las diferentes partes del algoritmo. Cada bloque de código se inicia con una palabra clave que indica el tipo de instrucción que se va a realizar, seguida de una descripción detallada de la operación que se va a llevar a cabo.

La estructura que usaremos y que debe de seguir el pseudocódigo es la siguiente:

Nombre del algoritmo: <<Nombre del algoritmo>>

Definición de constantes:

 Tipo de Dato: <<Nombre de la constante>> = <<Valor de la constante>>

Definición de variables:

 Tipo de Dato: <<Nombre de la variable>>

1. Inicio

2. <<Instrucciones>>

.

.

n. Fin

Tipos de datos

Los tipos de datos que podemos utilizar en el pseudocódigo son los mismos que en la mayoría de los lenguajes de programación, y nos permiten almacenar diferentes tipos de información, como números, texto, fechas, etc. Algunos de los tipos de datos más comunes son:

Tipo de Dato	Descripción
Entero	Números enteros, positivos o negativos
Real	Números con decimales
Carácter	Letras, números o símbolos
Cadena	Conjunto de caracteres
Lógico	Valores verdadero o falso

Ejemplo de pseudocódigo

A continuación, se muestra un ejemplo de pseudocódigo que calcula el área de un círculo a partir de su radio:

Nombre del algoritmo: Calcular área de un círculo

Definición de constantes:

Real: PI = 3.1416

Definición de variables:

Real: radio, area

1. Inicio
2. Escribir "Ingrese el radio del círculo:"
3. Leer radio
4. Hcer area = PI * radio * radio
5. Escribir "El área del círculo es:", area
6. Fin



Nota: En este ejemplo, PI es una constante que representa el valor de pi, radio es la variable que almacena el radio del círculo y area es la variable que almacena el área calculada. Las instrucciones Escribir y Leer se utilizan para mostrar mensajes en pantalla y leer valores del usuario, respectivamente.

⚠ Nota: Para efectos de las constantes, estas siempre se escribirán en mayúsculas y se les asignará un valor que no cambiará durante la ejecución del algoritmo. Por su parte, las variables se escribirán en minúsculas y se les asignará un valor que puede cambiar durante la ejecución del algoritmo.

Instrucciones de los algoritmos

Para los algoritmos tenemos las siguientes instrucciones:

1. **Inicio/Fin:** Marca el inicio y el fin del algoritmo.
2. **Leer:** Permite leer algún dato y almacenarlo dentro de una variable para su posterior uso.
3. **Escribir:** Permite mostrar en pantalla algún dato.
4. **Hacer:** Permite asignar un valor a una variable, ya sea simple o mediante algún cálculo.

Ejemplo 1

Nombre del algoritmo: SumaDeNumeros

Definición de variables:

Entero: num1, num2, suma

Algoritmo:

1. Inicio
2. Leer num1
3. Leer num2
4. Hacer suma = num1 + num2
5. Escribir suma
6. Fin

Otro ejemplo sería el siguiente:

Nombre del algoritmo: AreaDeUnCírculo

Definición de constantes:

Real: PI = 3.1416

Definición de variables:

Real: radio, area

Algoritmo:

1. Inicio
2. Leer radio
3. Hacer area = PI * radio * radio

4. Escribir area
5. Fin

Leer y Escribir

La instrucción Leer se utiliza para obtener un valor del usuario y almacenarlo en una variable, mientras que la instrucción Escribir se utiliza para mostrar un valor en pantalla. Estas instrucciones son fundamentales para la interacción con el usuario y para la visualización de resultados.

Ambas instrucciones pueden usarse para leer o escribir más de una variable o texto, separándolos por comas. Por ejemplo:

```
Leer a, b, c  
Escribir "El resultado es:", resultado
```

En este caso, se leen tres valores y se almacenan en las variables a, b y c, y luego se muestra en pantalla el mensaje "El resultado es:" seguido del valor de la variable resultado.

⚠ Nota: Es importante tener en cuenta que las instrucciones Leer y Escribir son específicas de pseudocódigo y pueden variar en otros lenguajes de programación.

⚠ Nota: Ambas instrucciones dependen de la definición de variables antes de poder usarlas, en otras palabras, las variables deben ser declaradas antes de poder ser utilizadas en las instrucciones Leer y Escribir.

Hacer

La instrucción Hacer se utiliza para asignar un valor a una variable, ya sea un valor simple o el resultado de una operación matemática. Por ejemplo:

```
Hacer suma = num1 + num2
```

En este caso, se asigna a la variable `suma` la suma de los valores almacenados en las variables `num1` y `num2`.

La instrucción `Hacer` es fundamental para realizar cálculos y operaciones en un algoritmo, ya que permite manipular los valores almacenados en las variables y realizar operaciones matemáticas con ellos.

⚠ Nota: La instrucción `Hacer` puede utilizarse para asignar valores a variables de diferentes tipos de datos, como enteros, reales, caracteres, etc.

⚠ Nota: La instrucción `Hacer` también puede utilizarse para asignar valores arbitrarios a variables, como por ejemplo `Hacer x = 10`, donde se asigna el valor 10 a la variable `x`.

Ejemplo 2

A continuación, se muestra un ejemplo de un algoritmo que calcula el área de un triángulo a partir de su base y altura:

Nombre del algoritmo: `CalcularAreaTriangulo`

Definición de variables:

Real: `base`, `altura`, `area`

Algoritmo:

1. Inicio
2. Leer `base`
3. Leer `altura`
4. Hacer `area = (base * altura) / 2`
5. Escribir "El área del triángulo es:", `area`
6. Fin

Operadores de los algoritmos

Los operadores son símbolos que permiten realizar operaciones matemáticas, lógicas y de comparación. En los algoritmos se utilizan los siguientes operadores:

1. **Aritméticos:** Permiten realizar operaciones matemáticas.
2. **Relacionales:** Permiten comparar dos valores.
3. **Lógicos:** Permiten realizar operaciones lógicas.

Operadores aritméticos

Los operadores aritméticos son los siguientes:

Operador	Descripción	Ejemplo
<code>+</code>	Suma	<code>a + b</code>
<code>-</code>	Resta	<code>a - b</code>
<code>*</code>	Multiplicación	<code>a * b</code>
<code>/</code>	División	<code>a / b</code>
<code>%</code>	Módulo	<code>a % b</code>
<code>++</code>	Incremento	<code>a++</code>
<code>--</code>	Decremento	<code>a--</code>
<code>^</code>	Potencia	<code>a ^ b</code>

Operadores relacionales

Los operadores relacionales son los siguientes:

Operador	Descripción	Ejemplo
<code>==</code>	Igual a	<code>a == b</code>
<code>!= o <></code>	Diferente de	<code>a != b</code>
<code>></code>	Mayor que	<code>'a > b</code>
<code><</code>	Menor que	<code>a < b</code>
<code>>=</code>	Mayor o igual	<code>a >= b</code>
<code><=</code>	Menor o igual	<code>a <= b</code>
<code>!</code>	Negación	<code>!a</code>

Operadores lógicos

Los operadores lógicos son los siguientes:

Operador	Descripción	Ejemplo
<code>&&</code>	Y	<code>a && b</code>
<code> </code>	O	<code>a b</code>

Diagramas de flujo

Los diagramas de flujo son una herramienta visual que se utiliza para representar algoritmos y procesos de forma gráfica. Los diagramas de flujo son una forma eficaz de comunicar ideas y procesos complejos de una manera clara y concisa.

Elementos de un diagrama de flujo

Los diagramas de flujo están compuestos por una serie de elementos que representan diferentes partes de un algoritmo o proceso. Algunos de los elementos más comunes de un diagrama de flujo son:

1. Inicio/Fin:

- Representa el inicio o fin de un algoritmo o proceso.
- Se representa con un óvalo.

2. Proceso:

- Representa una operación o acción que se realiza en el algoritmo.
- Se representa con un rectángulo.
- Usualmente, se usa para asignar valores a variables, realizar cálculos o ejecutar instrucciones.

3. Decisión:

- Representa una condición o pregunta que se evalúa en el algoritmo.
- Se representa con un rombo.
- Usualmente, se usa para tomar decisiones basadas en una condición.

4. Conector:

- Representa la conexión entre diferentes partes de un diagrama de flujo.
- Se representa con un círculo.

5. Flecha:

- Representa la dirección del flujo de un proceso en el diagrama.
- Se representa con una flecha.

6. Entrada/Salida:

- Representa la entrada o salida de datos en el algoritmo.
- Se representa con un paralelogramo.
- Usualmente, se usa para leer o escribir datos desde o hacia una fuente externa.

Elementos de un diagrama de flujo

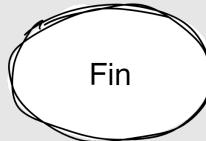
Los diagramas de flujo están compuestos por una serie de elementos que representan diferentes partes de un algoritmo o proceso. Algunos de los elementos más comunes de un diagrama de flujo son:

Inicio/Fin

Representa el inicio o fin de un algoritmo o proceso. Se representa con un óvalo.



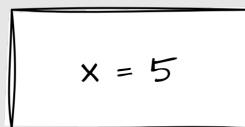
inicio.png



fin.png

Proceso

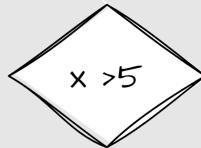
Representa una operación o acción que se realiza en el algoritmo. Se representa con un rectángulo.



hacer.png

Decisión

Representa una condición o pregunta que se evalúa en el algoritmo. Se representa con un rombo.



decision.png

Conektor

Representa la conexión entre diferentes partes de un diagrama de flujo. Se representa con un círculo pequeño.



Flecha

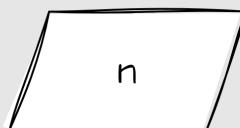
Representa la dirección del flujo de un proceso en el diagrama. Se representa con una flecha.



flecha.png

Entrada

Representa la entrada de datos en el algoritmo. Se representa con un paralelogramo.



entrada.png

Salida

Representa la salida de datos en el algoritmo. Se representa con un documento.

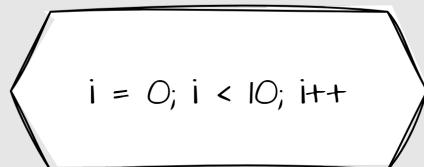


salida.png

Para

Representa un bucle o ciclo en el algoritmo. Se representa con un hexágono.

En dónde se deberá indicar el número de veces que se repetirá el proceso.



para.png

Subproceso

Representa un proceso o algoritmo que se ejecuta de forma independiente en el diagrama de flujo. Se representa con un rectángulo con dos líneas verticales.



subproceso.png

Estos elementos son fundamentales para la creación y comprensión de un diagrama de flujo, ya que permiten representar de forma clara y concisa la lógica y el flujo de

ejecución de un algoritmo o proceso. Al utilizar símbolos gráficos y conectarlos mediante flechas, se crea una representación visual que facilita la visualización y el análisis de un algoritmo, lo que permite diseñar, analizar y documentar procesos de una manera efectiva y eficiente.

La condicional

Dentro de algoritmia podemos utilizar la condicional para tomar decisiones en nuestro código. La condicional es una estructura de control que nos permite evaluar una expresión y ejecutar un bloque de código si la expresión es verdadera.

Para hacer esto usamos la instrucción **Si**, que se escribe de la siguiente manera:

```
n. Si <expresión> Entonces  
    Inicio  
        <bloque de código>  
    Fin  
n+1. <><Otras instrucciones>>
```

A Nota: Las instrucciones **Inicio** y **Fin** son solo para indicar el inicio y fin del bloque de código que se ejecuta si la expresión es verdadera. No son instrucciones que se escriban en el código, y sirven para delimitar las instrucciones que se ejecutan si la expresión es verdadera.

Donde **<expresión>** es una expresión lógica que se evalúa a verdadero o falso, y **<bloque de código>** es un conjunto de instrucciones que se ejecutan si la expresión es verdadera.

Por ejemplo, si queremos imprimir un mensaje si un número es mayor que 5, podemos hacerlo de la siguiente manera:

```
1. Inicio  
2. Leer n  
3. Si n > 5 Entonces  
    Escribir "El número es mayor que 5"  
4. Fin
```

En este caso, si el número ingresado es mayor que 5, se imprimirá el mensaje "El número es mayor que 5".

Condicional doble

Además de la condicional simple, también podemos utilizar la condicional doble, que nos permite ejecutar un bloque de código si la expresión es verdadera, y otro bloque de código si la expresión es falsa.

La condicional doble se escribe de la siguiente manera:

```
n. Si <expresión> Entonces
    Inicio
        <bloque de código 1>
    Fin
    Otro caso
    Inicio
        <bloque de código 2>
    Fin
n+1. <>Otras instrucciones>>
```

Donde **<expresión>** es una expresión lógica que se evalúa a verdadero o falso, **<bloque de código 1>** es un conjunto de instrucciones que se ejecutan si la expresión es verdadera, y **<bloque de código 2>** es un conjunto de instrucciones que se ejecutan si la expresión es falsa.

Por ejemplo, si queremos imprimir un mensaje si un número es mayor que 5 y otro mensaje si no lo es, podemos hacerlo de la siguiente manera:

```
1. Inicio
2. Leer n
3. Si n > 5 Entonces
    Escribir "El número es mayor que 5"
    Otro caso
    Escribir "El número no es mayor que 5"
4. Fin
```

En este caso, si el número ingresado es mayor que 5, se imprimirá el mensaje "El número es mayor que 5", y si no lo es, se imprimirá el mensaje "El número no es mayor que 5".

Condicional múltiple

Además de la condicional simple y doble, también podemos utilizar la condicional múltiple, que nos permite evaluar varias expresiones y ejecutar un bloque de código

dependiendo de cuál de ellas sea verdadera.

La condicional múltiple se escribe de la siguiente manera:

```
n. Si <expresión 1> Entonces
    Inicio
        <bloque de código 1>
    Fin
    Otro caso
        Si <expresión 2> Entonces
            Inicio
                <bloque de código 2>
            Fin
        Otro caso
            Si <expresión 3> Entonces
                Inicio
                    <bloque de código 3>
                Fin
                ...
            Otro caso
                Inicio
                    <bloque de código n>
                Fin
n+1. <>Otras instrucciones>>
```

Donde <expresión 1>, <expresión 2>, <expresión 3>, ..., <expresión n> son expresiones lógicas que se evalúan a verdadero o falso, <bloque de código 1>, <bloque de código 2>, <bloque de código 3>, ..., <bloque de código n> son conjuntos de instrucciones que se ejecutan si la expresión correspondiente es verdadera.

Por ejemplo, si queremos imprimir un mensaje dependiendo de si un número es mayor que 5, igual a 5 o menor que 5, podemos hacerlo de la siguiente manera:

```
1. Inicio
2. Leer n
3. Si n > 5 Entonces
    Escribir "El número es mayor que 5"
    Otro caso
        Si n = 5 Entonces
```

```
    Escribir "El número es igual a 5"
    Otro caso
        Escribir "El número es menor que 5"
4. Fin
```

En este caso, si el número ingresado es mayor que 5, se imprimirá el mensaje "El número es mayor que 5", si es igual a 5, se imprimirá el mensaje "El número es igual a 5", y si es menor que 5, se imprimirá el mensaje "El número es menor que 5".

Condicional anidada

Además de la condicional simple, doble y múltiple, también podemos utilizar la condicional anidada, que nos permite anidar condicionales dentro de otros condicionales.

La condicional anidada se escribe de la siguiente manera:

```
n. Si <expresión 1> Entonces
    Inicio
        Si <expresión 2> Entonces
            Inicio
                <bloque de código 1>
            Fin
        Otro caso
            Inicio
                <bloque de código 2>
            Fin
        Fin
    Otro caso
        Inicio
            <bloque de código 3>
        Fin
n+1. <>Otras instrucciones>>
```

Donde <expresión 1> y <expresión 2> son expresiones lógicas que se evalúan a verdadero o falso, <bloque de código 1>, <bloque de código 2>, <bloque de código 3>, ...,

<bloque de código n> son conjuntos de instrucciones que se ejecutan si la expresión correspondiente es verdadera.

Por ejemplo, si queremos imprimir un mensaje dependiendo de si un número es mayor que 5 y par, mayor que 5 e impar, o menor que 5, podemos hacerlo de la siguiente manera:

```
1. Inicio
2. Leer n
3. Si n > 5 Entonces
    Si n % 2 = 0 Entonces
        Escribir "El número es mayor que 5 y par"
    Otro caso
        Escribir "El número es mayor que 5 e impar"
    Otro caso
        Escribir "El número es menor que 5"
4. Fin
```

En este caso, si el número ingresado es mayor que 5 y par, se imprimirá el mensaje "El número es mayor que 5 y par", si es mayor que 5 e impar, se imprimirá el mensaje "El número es mayor que 5 e impar", y si es menor que 5, se imprimirá el mensaje "El número es menor que 5".

Conclusiones

En resumen, la condicional es una estructura de control que nos permite tomar decisiones en nuestro código. Podemos utilizar la condicional simple, doble, múltiple y anidada para evaluar expresiones lógicas y ejecutar bloques de código dependiendo de si la expresión es verdadera o falsa. Esto nos permite crear programas más complejos y con un mayor grado de control sobre su ejecución.

Ejemplo de uso de Condicionales

El Problema

Fábricas “El cometa” produce artículos con claves (1, 2, 3, 4, 5 y 6). Se requiere un algoritmo para calcular los precios de venta, para esto hay que considerar lo siguiente:

- Costo de producción = materia prima + mano de obra + gastos de fabricación.
- Precio de venta = costo de producción + 45 % de costo de producción.

El costo de la mano de obra se obtiene de la siguiente forma: para los productos con clave 3 o 4 se carga 75 % del costo de la materia prima; para los que tienen clave 1 y 5 se carga 80 %, y para los que tienen clave 2 o 6, 85 %.

Para calcular el gasto de fabricación se considera que si el artículo que se va a producir tiene claves 2 o 5, este gasto representa 30 % sobre el costo de la materia prima; si las claves son 3 o 6, representa 35 %; si las claves son 1 o 4, representa 28 %. La materia prima tiene el mismo costo para cualquier clave.

Definición de Variables

Para efectos del problema planteado se tiene lo siguiente:

Variable	Descripción	Tipo de Dato
C	Clave del artículo a producir.	Entero
MP	Costo de la materia prima.	Real
MO	Costo de la mano de obra.	Real
GF	Costo de los gastos de fabricación.	Real
CP	Costo de producción.	Real
PV	Precio de venta.	Real

EL Algoritmo

Nombre del Algoritmo: CostosFabrica

Declaración de variables:

Entero: C

Real: MP, MO, GF, CP, PV

Algoritmo:

1. Inicio

2. Escribir "Ingrese el código del producto a fabricar"

3. Leer C

4. Escribir "Ingrese el costo de la materia prima"

5. Leer MP

6. Si C == 4 || C == 3 Entonces

Hacer MO= MP * 0.75

Otro caso

Si C == 1 || C == 5 Entonces

Hacer MO = MP * 0.80

Otro caso

Hacer MO = MP * 0.85

7. Si C == 2 || C == 5 Entonces

Hacer GF = MP * 0.30

Otro caso

Si C == 3 || C == 6 Entonces

Hacer GF = MP * 0.35

Otro caso

Hacer GF = MP * 0.28

8. Hacer CP = MP + MO + GF

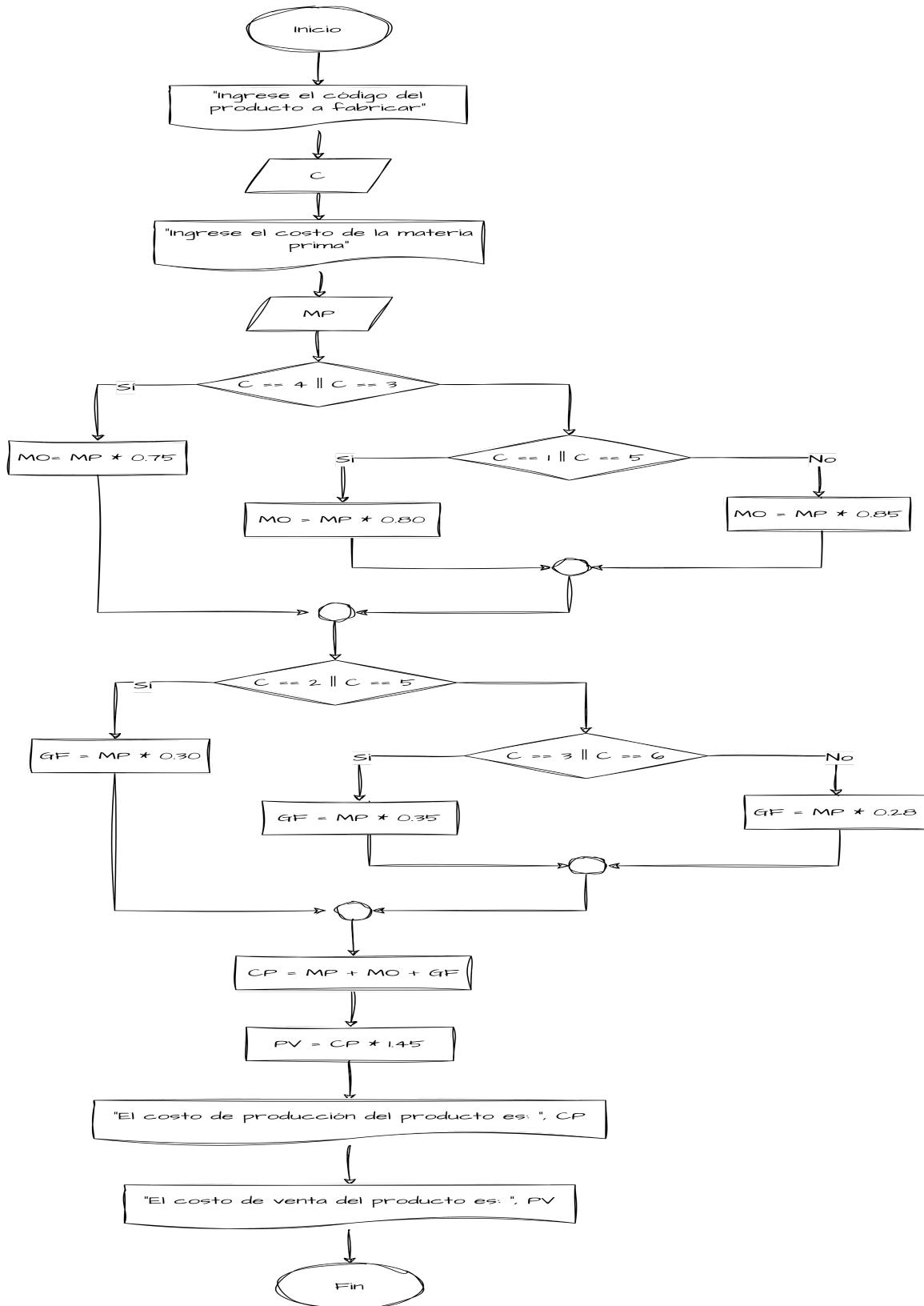
9. Hacer PV = CP * 1.45 => CP + CP * 0.45

10. Escribir "El costo de producción del producto es: ", CP

11. Escribir "El costo de venta del producto es: ", PV

12. Fin

El diagrama de flujo



condicional_1.png

Prueba del Algoritmo

Para probar el algoritmo se ingresan los siguientes datos:

- Clave del producto: 3
- Costo de la materia prima: 100

Al ejecutar el algoritmo se obtiene lo siguiente:

```
Ingrese el código del producto a fabricar
```

```
3
```

```
Ingrese el costo de la materia prima
```

```
100
```

```
El costo de producción del producto es: 210
```

```
El costo de venta del producto es: 304.5
```

Conclusiones

El uso de condicionales en un algoritmo permite tomar decisiones basadas en el valor de una expresión lógica. En este caso, se utilizó una condicional para determinar el costo de la mano de obra y los gastos de fabricación en función de la clave del producto a fabricar. Esto permitió calcular el costo de producción y el precio de venta del producto de forma automática, simplificando el proceso de cálculo y evitando errores humanos.

Referencias

- Estructuras de control en programación
(https://es.wikipedia.org/wiki/Estructura_de_control)
- Algoritmos y diagramas de flujo (https://es.wikipedia.org/wiki/Diagrama_de_flujo)
- Introducción a la programación (<https://es.wikipedia.org/wiki/Programación>)
- Lenguajes de programación
(https://es.wikipedia.org/wiki/Lenguaje_de_programación)

En caso de (switch)

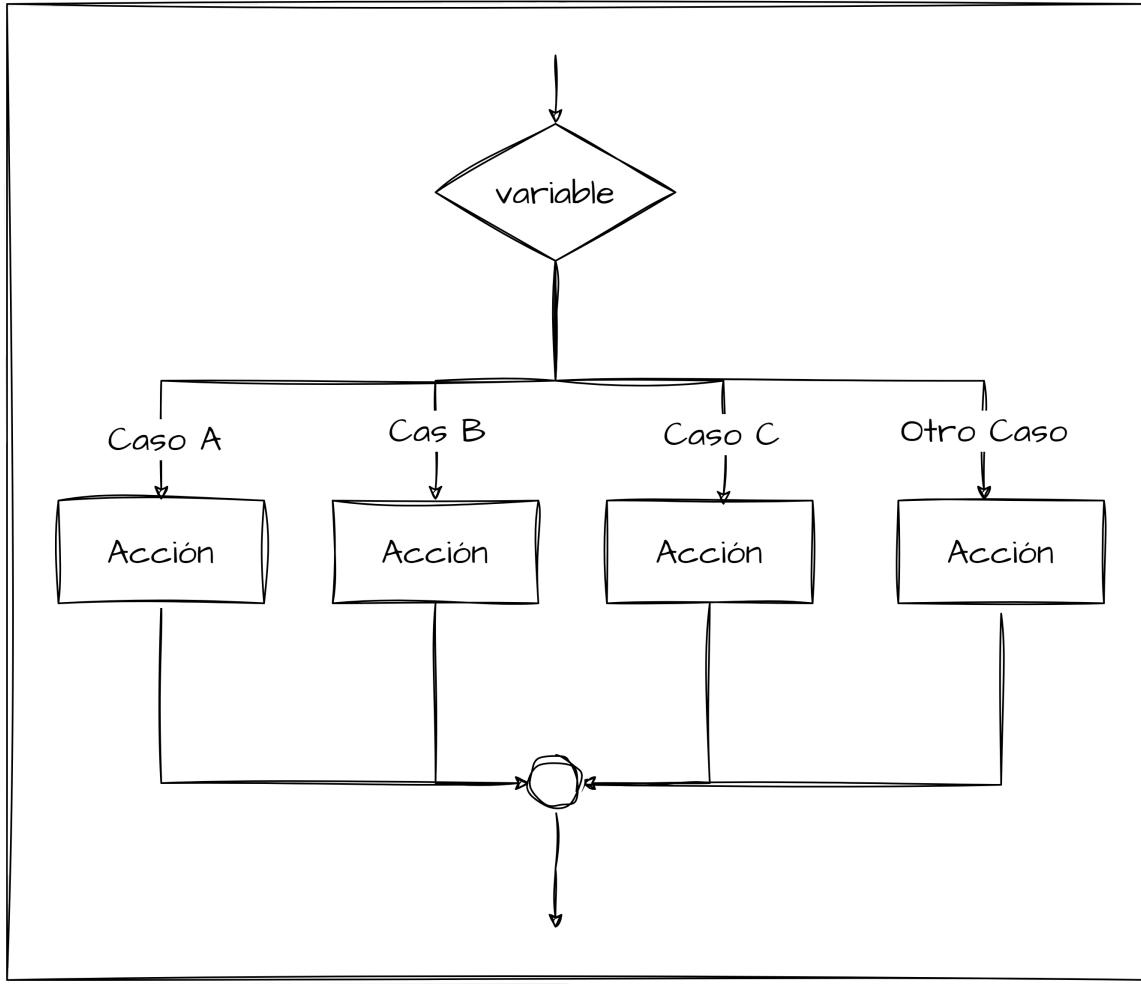
La estructura de decisión switch es una estructura de control que permite evaluar una expresión y ejecutar un bloque de código dependiendo del valor de la expresión. La estructura switch en pseudocódigo es la siguiente:

```
n.- En caso de [expresión]
    Caso [valor1]
        Inicio
        [Acción1]
        Fin
    Caso [valor2]
        Inicio
        [Acción2]
        Fin
    ...
    Caso [valorN]
        Inicio
        [AcciónN]
        Fin
    Otro Caso
        Inicio
        [Acción por defecto]
        Fin
```

Como se puede observar, la estructura switch consta de varios bloques de código, cada uno asociado a un valor específico de la expresión. Si el valor de la expresión coincide con alguno de los valores especificados, se ejecuta el bloque de código correspondiente. Si no se encuentra un valor coincidente, se ejecuta el bloque de código asociado a Otro Caso.

Es importante tener en cuenta que la estructura switch no permite evaluar expresiones complejas, como rangos de valores o expresiones booleanas. Solo se pueden evaluar valores concretos.

Diagrama de flujo



switch_flujo.png

En el diagrama de flujo, la estructura de decisión `switch` se representa con un rombo que contiene la variable o expresión a evaluar. Cada flecha que sale del rombo representa un caso posible, con una etiqueta que indica el valor asociado al caso. Si la expresión coincide con alguno de los valores especificados, se sigue la flecha correspondiente al caso. Si no se encuentra un valor coincidente, se sigue la flecha correspondiente a `Otro Caso`.

Ejemplo de en caso de (switch)

Caso 1

Una compañía de paquetería internacional tiene servicio en algunos países de América del Norte, América Central, América del Sur, Europa y Asia. El costo por el servicio de paquetería se basa en el peso del paquete y la zona a la que va dirigido. Lo anterior se muestra en la tabla siguiente:

Zona	Ubicación	Costo por Kg
1	América del Norte	24.00
2	América Central	20.00
3	América del Sur	21.00
4	Europa	10.00
5	Asia	18.00

Se requiere un algoritmo para determinar cuánto se debe cobrar por el servicio de paquetería, considerando que si el peso del paquete excede los 5 kg, se debe cobrar un costo adicional de 5.00 por cada kilo adicional.

Definición de Variables

Para efectos del problema planteado se tiene lo siguiente:

Variable	Descripción	Tipo de Dato
Z	Zona a la que va dirigido.	Entero
P	Peso del paquete.	Real
C	Costo del servicio.	Real

El Algoritmo

Nombre del Algoritmo: CostoPaqueteria

Declaración de variables:

Entero: Z

Real: P, C

Algoritmo:

1. Inicio

2. Escribir "Seleccione la zona a la que va dirigido el paquete:

- 1. América del Norte
- 2. América Central
- 3. América del Sur
- 4. Europa
- 5. Asia"

3. Leer Z

4. Escribir "Ingrese el peso del paquete"

5. Leer P

6. En caso de Z Hacer

Caso 1

 Inicio

 Hacer C = P * 24.00

 Fin

Caso 2

 Hacer C = P * 20.00

Caso 3

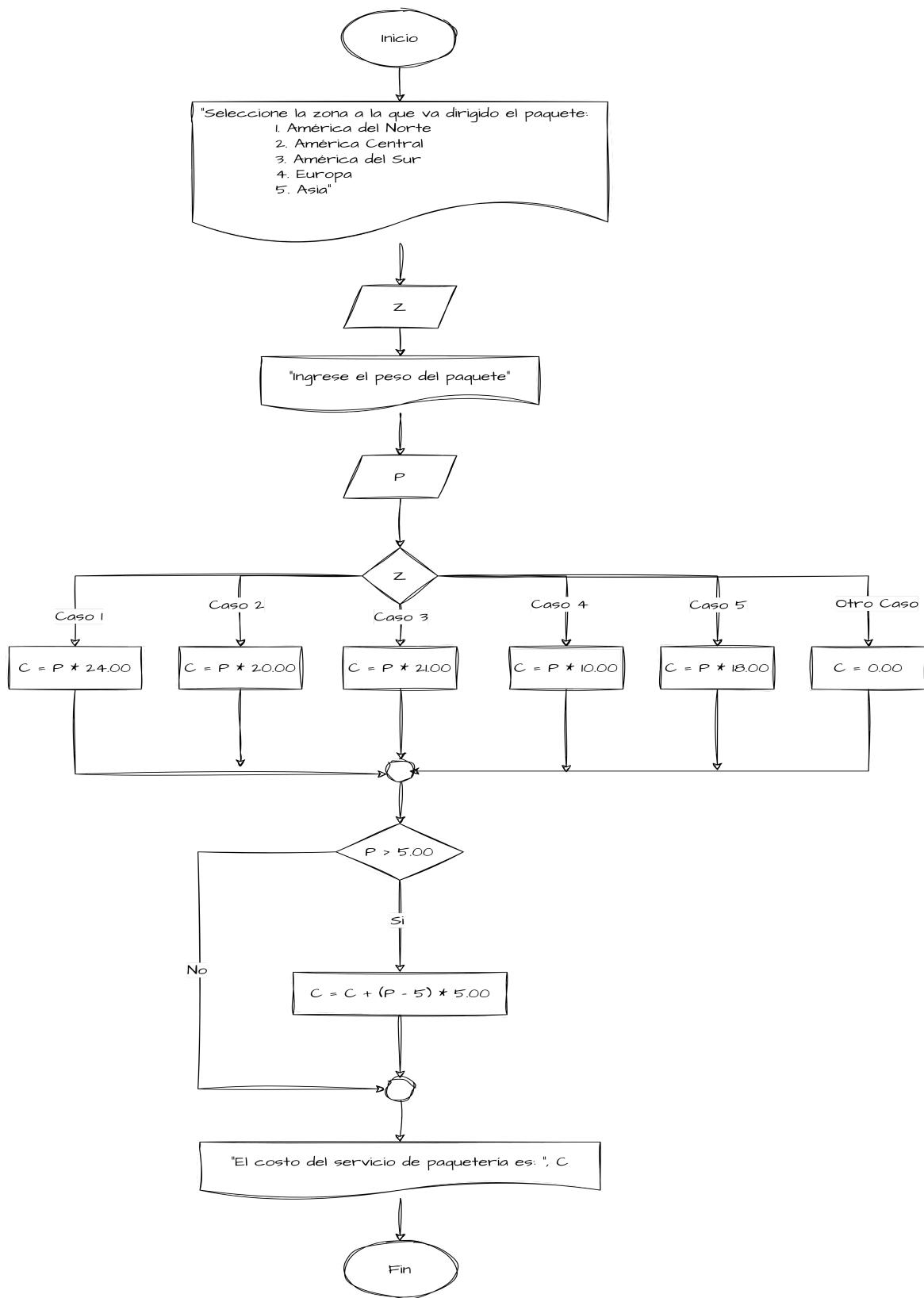
 Inicio

 Hacer C = P * 21.00

 Fin

```
Caso 4
    Inicio
        Hacer C = P * 10.00
    Fin
Caso 5
    Inicio
        Hacer C = P * 18.00
    Fin
Otro Caso
    Inicio
        Hacer C = 0.00
    Fin
7. Si P > 5 Entonces
    Hacer C = C + (P - 5) * 5.00
8. Escribir "El costo del servicio de paquetería es: ", C
9. Fin
```

El diagrama de flujo



switch_1.png

Prueba del Algoritmo

Para probar el algoritmo se ingresan los siguientes datos:

- Zona: 3
- Peso del paquete: 7

Al ejecutar el algoritmo se obtiene lo siguiente:

Seleccione la zona a la que va dirigido el paquete:

1. América del Norte
2. América Central
3. América del Sur
4. Europa
5. Asia

3

Ingrese el peso del paquete

7

El costo del servicio de paquetería es: 147.00

Estructuras Repetitivas

Introducción

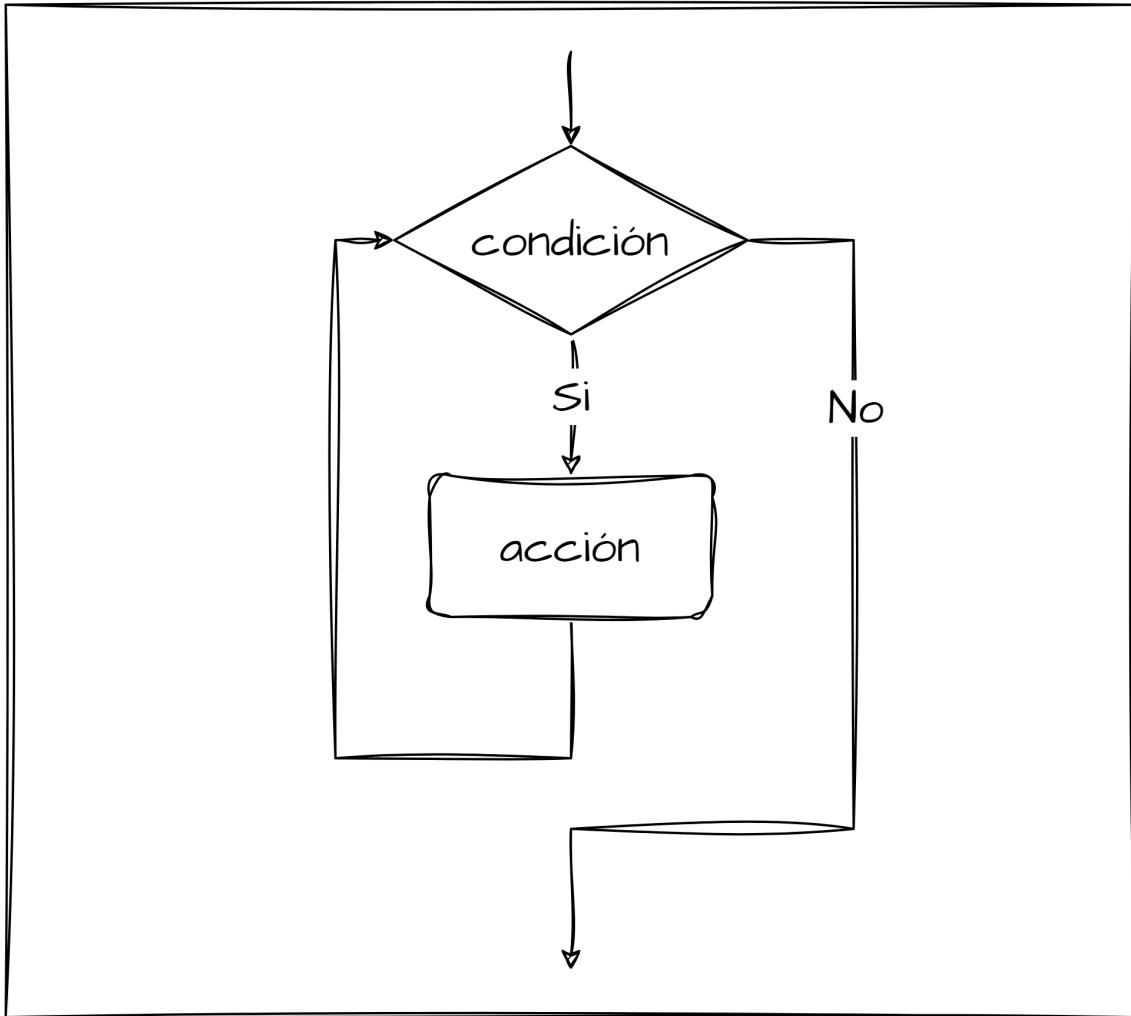
Las estructuras repetitivas son estructuras de control que permiten ejecutar un bloque de código múltiples veces. Estas estructuras son útiles cuando se necesita realizar una tarea repetitiva, como sumar una lista de números o imprimir una secuencia de caracteres.

Las estructuras repetitivas más comunes son el bucle `for`, el bucle `while` y el bucle `do while`. El bucle `for` se utiliza cuando se conoce el número de iteraciones que se deben realizar, mientras que el bucle `while` se utiliza cuando se desconoce el número de iteraciones.

Mientras que (`while`)

La estructura repetitiva `while` es una estructura de control que permite ejecutar un bloque de código mientras se cumpla una condición. La estructura `while` en pseudocódigo es la siguiente:

```
n.- Mientras que [condición]
    Inicio
        [Acción]
    Fin
```



while.png

Como se puede observar, la estructura `while` consta de un bloque de código que se ejecuta mientras la condición sea verdadera. Si la condición es falsa, el bloque de código no se ejecuta.

Es importante tener en cuenta que si la condición es siempre verdadera, el bucle `while` se ejecutará indefinidamente,

Hacer hasta que (do while)

La estructura repetitiva `do while` es una estructura de control que permite ejecutar un bloque de código al menos una vez y luego repetirlo mientras no se cumpla una condición. La estructura `do while` en pseudocódigo es la siguiente:

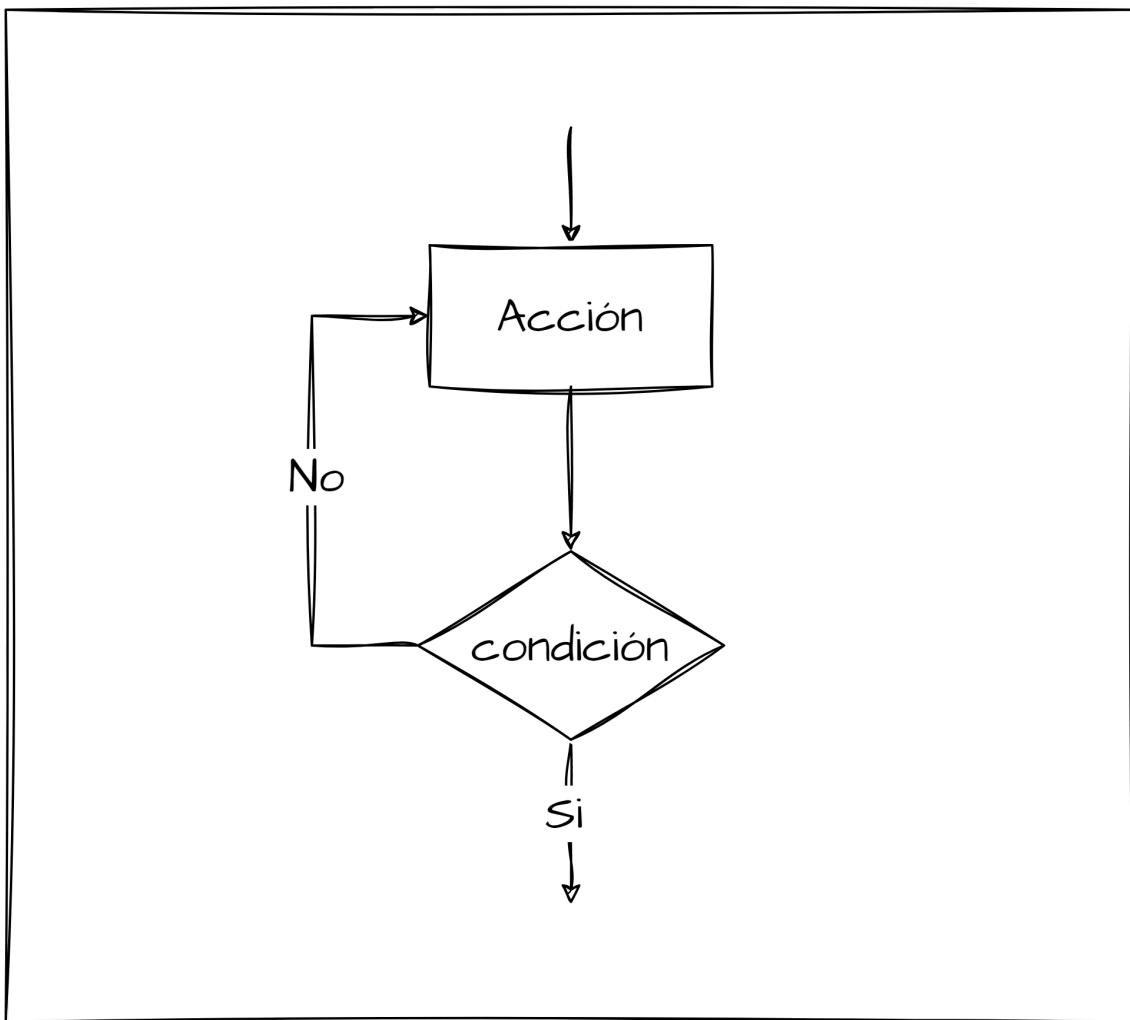
n.- Hacer

 Inicio

 [Acción]

 Fin

Hasta que [condición]



dowhile.png

Como se puede observar, la estructura `do while` consta de un bloque de código que se ejecuta al menos una vez y luego se repite mientras la condición sea falsa. Si la condición es verdadera, se continuará con el flujo del programa.

Al igual que con el bucle `while`, si la condición es siempre falsa, el bucle `do while` se ejecutará indefinidamente.

Para (for)

La estructura repetitiva `for` es una estructura de control que permite ejecutar un bloque de código un número fijo de veces. La estructura `for` en pseudocódigo es la siguiente:

```
n.- Para [variable] = [inicio] Hasta [fin] Con Paso [paso] Hacer  
    Inicio  
    [Acción]  
    Fin
```

Como se puede observar, la estructura `for` consta de una variable de control que se inicializa con un valor inicial y luego se incrementa o decrementa en cada iteración. El bucle se ejecuta mientras la variable de control esté dentro del rango especificado.

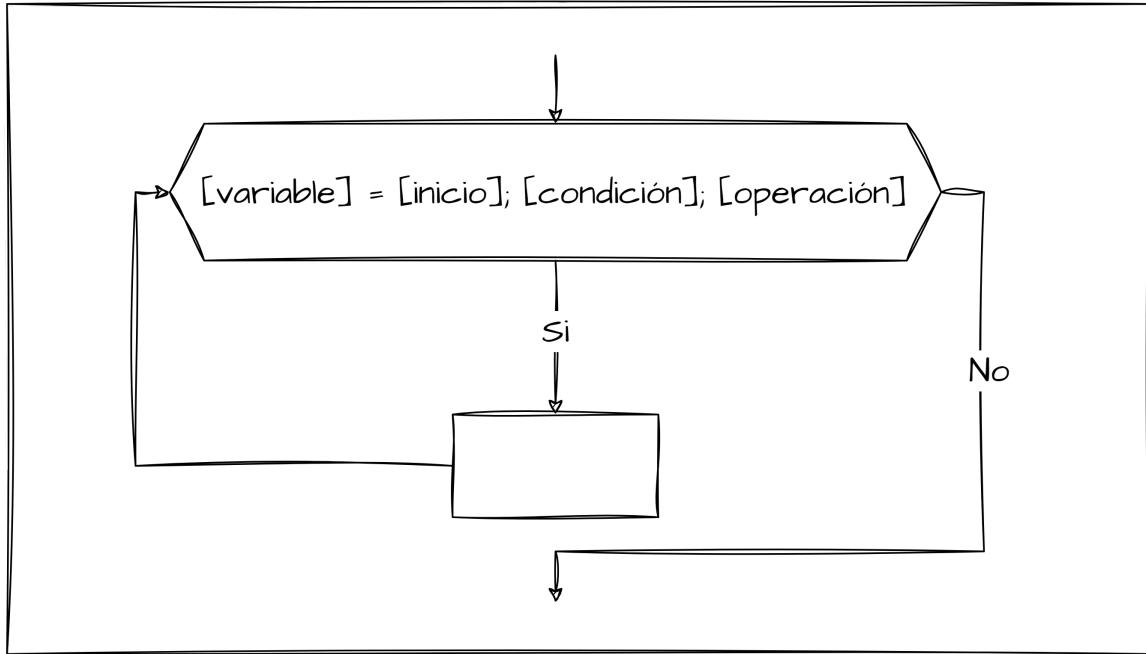
Sin embargo, la estructura `for` también puede no declarar la variable de control en la misma línea. En este caso, la variable de control se declara e inicializa fuera del bucle y se incrementa o decrementa dentro del bucle. La estructura `for` en pseudocódigo sin declarar la variable de control es la siguiente:

```
n.- [variable] = [inicio]  
m.- Para Hasta [condición] Con Paso [paso] Hacer  
    Inicio  
    [Acción]  
    Fin
```

Como se puede observar, la estructura `for` sin declarar la variable de control consta de una condición que se evalúa en cada iteración. El bucle se ejecuta mientras la condición sea verdadera.

Por último, el segmento `Con Paso [paso]` es opcional y se utiliza para especificar el incremento o decremento de la variable de control en cada iteración. Si no se especifica, la variable de control se incrementa o decrementa en una unidad por defecto. Además de esto, el segmento `Con Paso [paso]` también puede ser negativo, lo que permite decrementar la variable de control en cada iteración. Aunque también se puede emplear para realizar algún tipo de operación en la variable de control.

Diagrama de flujo



for.png

Ejemplos de Bucles

Ejercicio 1

Se requiere un algoritmo para obtener la suma de diez cantidades. Realice el diagrama de flujo y el pseudocódigo.

Con base en lo que se requiere determinar se puede establecer que las variables requeridas para la solución del problema son las mostradas en la tabla:

Variable	Tipo de Dato	Descripción
C	Entero	Contador
VA	Real	Valor por Sumar
SU	Real	Suma Acumulada

Resolviendo con el ciclo Mientras

Nombre del Algoritmo: Suma de 10 Cantidadades

Definición de variables

Entero: C

Real: VA, SU

Algoritmo:

1. Inicio
2. Hacer C=1
3. Hacer SU=0
4. Mientras C<=10

 Inicio

 Escribir "Ingrese el valor a sumar"

 Leer VA

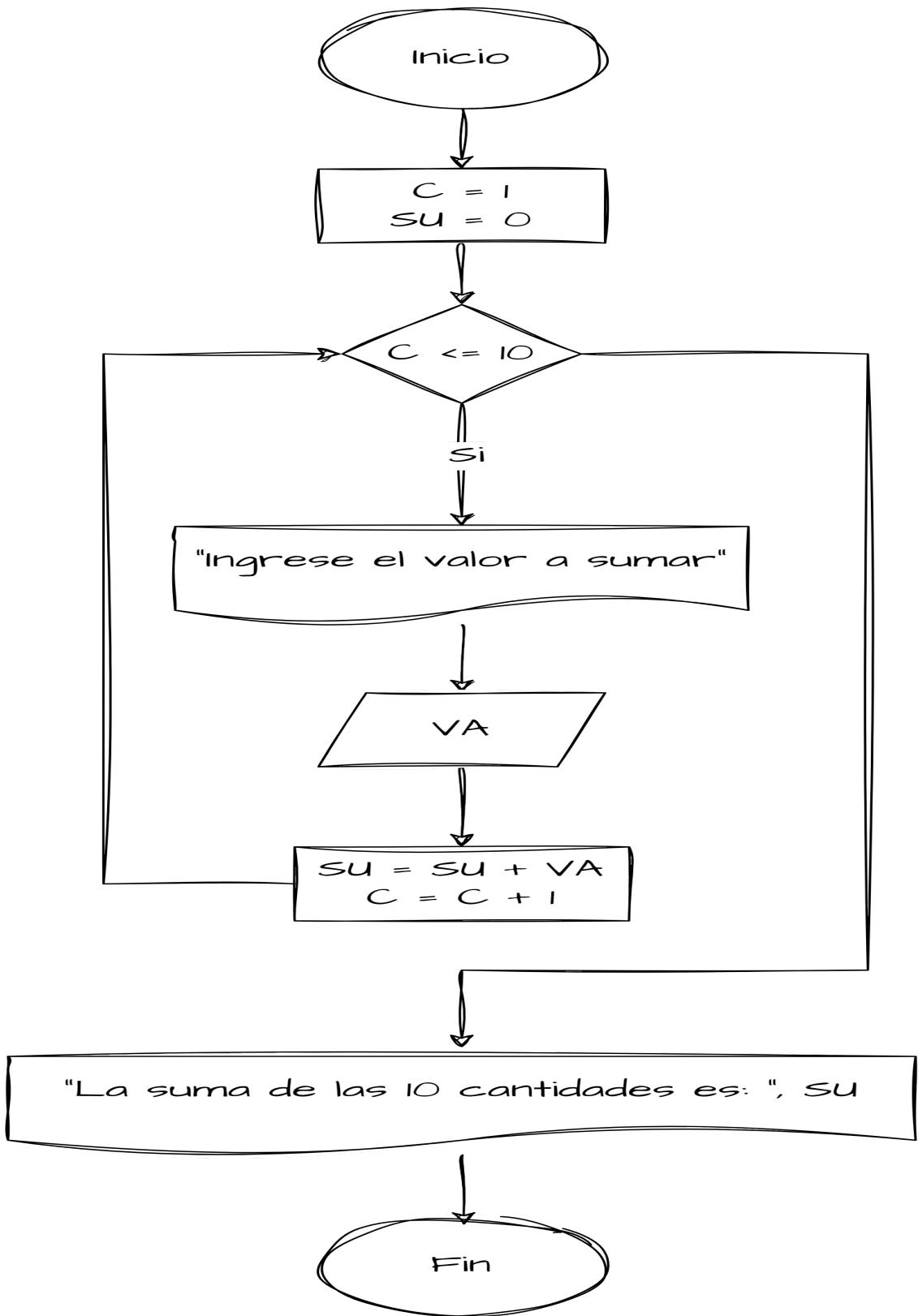
 Hacer SU=SU+VA

 Hacer C=C+1

 Fin

5. Escribir "La suma de las 10 cantidades es: ", SU

6. Fin



while_1.png

Resolviendo con el ciclo Hasta Que

Nombre del Algoritmo: Suma de 10 Cantidades

Definición de variables

Entero: C

Real: VA, SU

Algoritmo:

1. Inicio
2. Hacer $C=1$
3. Hacer $SU=0$
4. Hacer

 Inicio

 Escribir "Ingrese el valor a sumar"

 Leer VA

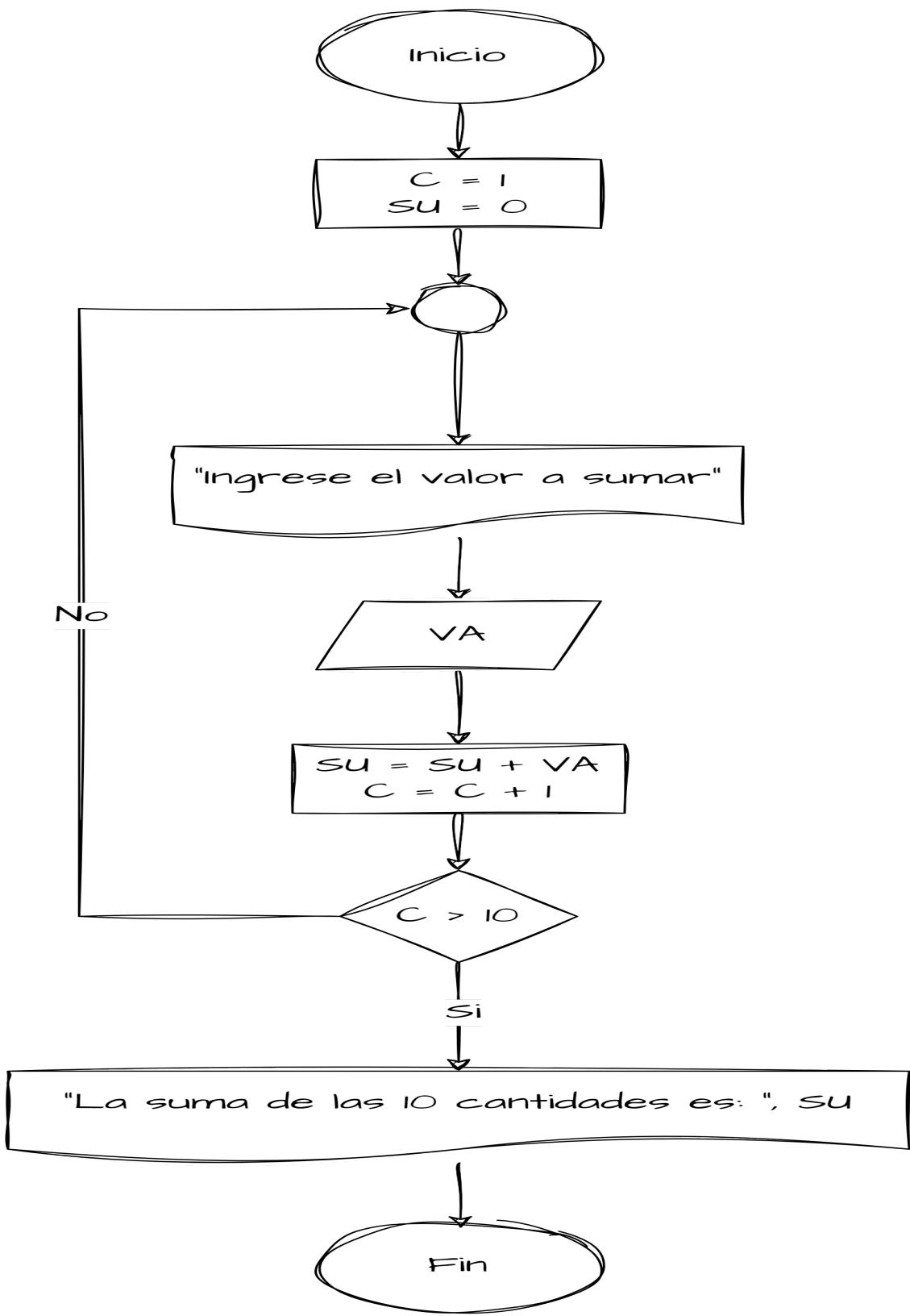
 Hacer $SU=SU+VA$

 Hacer $C=C+1$

 Fin

 Hasta Que $C>10$

5. Escribir "La suma de las 10 cantidades es: ", SU
6. Fin



do_while_1.png

Resolviendo con el ciclo Para

Nombre del Algoritmo: Suma de 10 Cantidadess

Definición de variables

Entero: C

Real: VA, SU

Algoritmo:

1. Inicio

2. Hacer SU=0

3. Para C=1 Hasta 10 // [Opcional] Con Paso 1

 Inicio

 Escribir "Ingrese el valor a sumar"

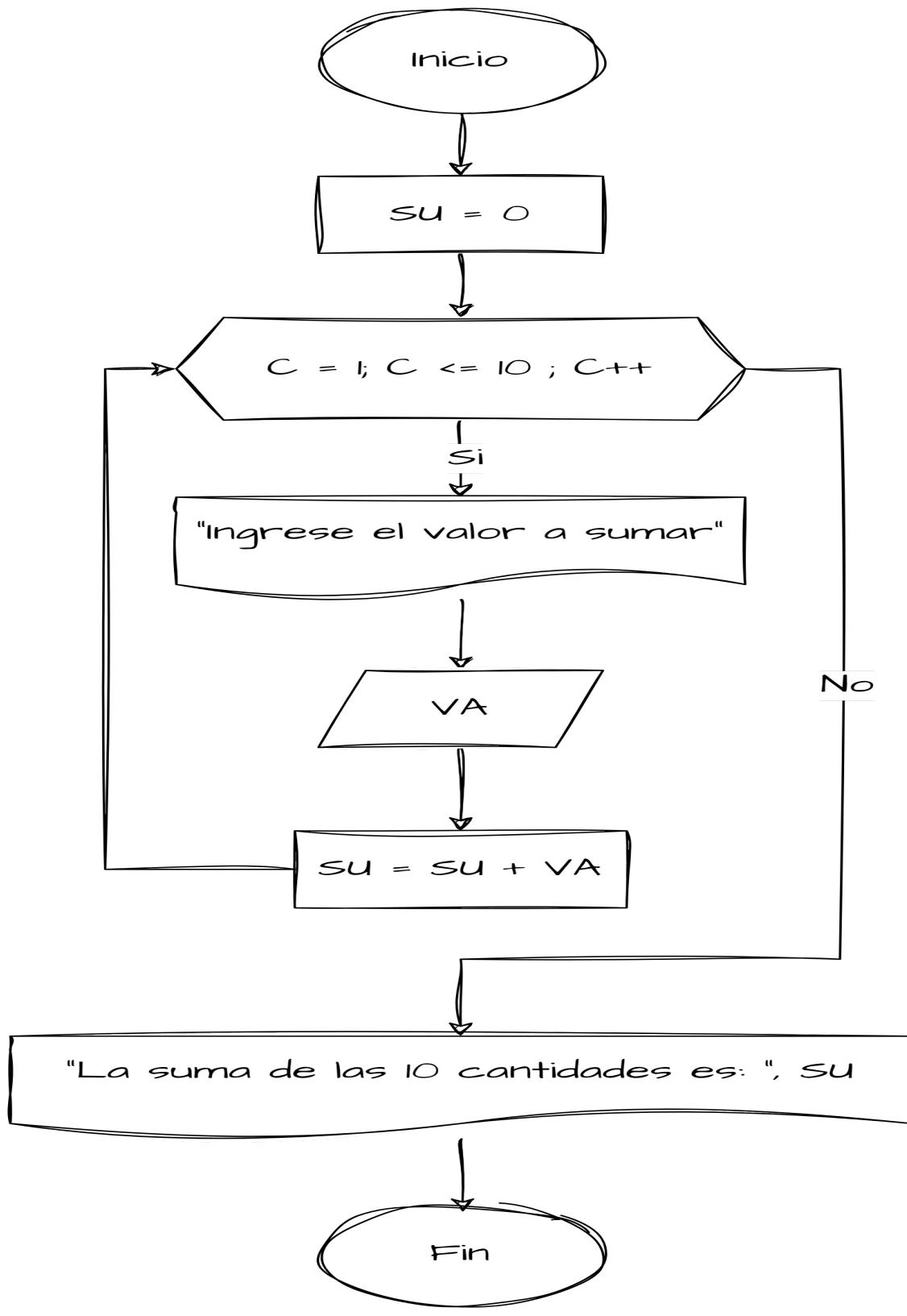
 Leer VA

 Hacer SU=SU+VA

 Fin

4. Escribir "La suma de las 10 cantidades es: ", SU

5. Fin



for_1.png

De estas maneras es como se puede resolver el problema planteado, utilizando los diferentes tipos de ciclos que se pueden encontrar en un lenguaje de programación.

Ejercicio 2

Una empresa les paga a sus empleados con base en las horas trabajadas en la semana. Realice un algoritmo para determinar el sueldo semanal de N trabajadores y, además, calcule cuánto pagó la empresa por los N empleados.

Con base en lo que se requiere determinar se puede establecer que las variables requeridas para la solución del problema son las mostradas en la tabla:

Variáble	Tipo de Dato	Descripción
N	Entero	Número de Empleados
HT	Real	Horas Trabajadas
PH	Real	Pago por Hora
SS	Real	Sueldo Semanal
ST	Real	Sueldo Total
C	Entero	Contador

Resolviendo con el ciclo Mientras 2

Nombre del Algoritmo: Sueldo Semanal de N Trabajadores

Definición de variables

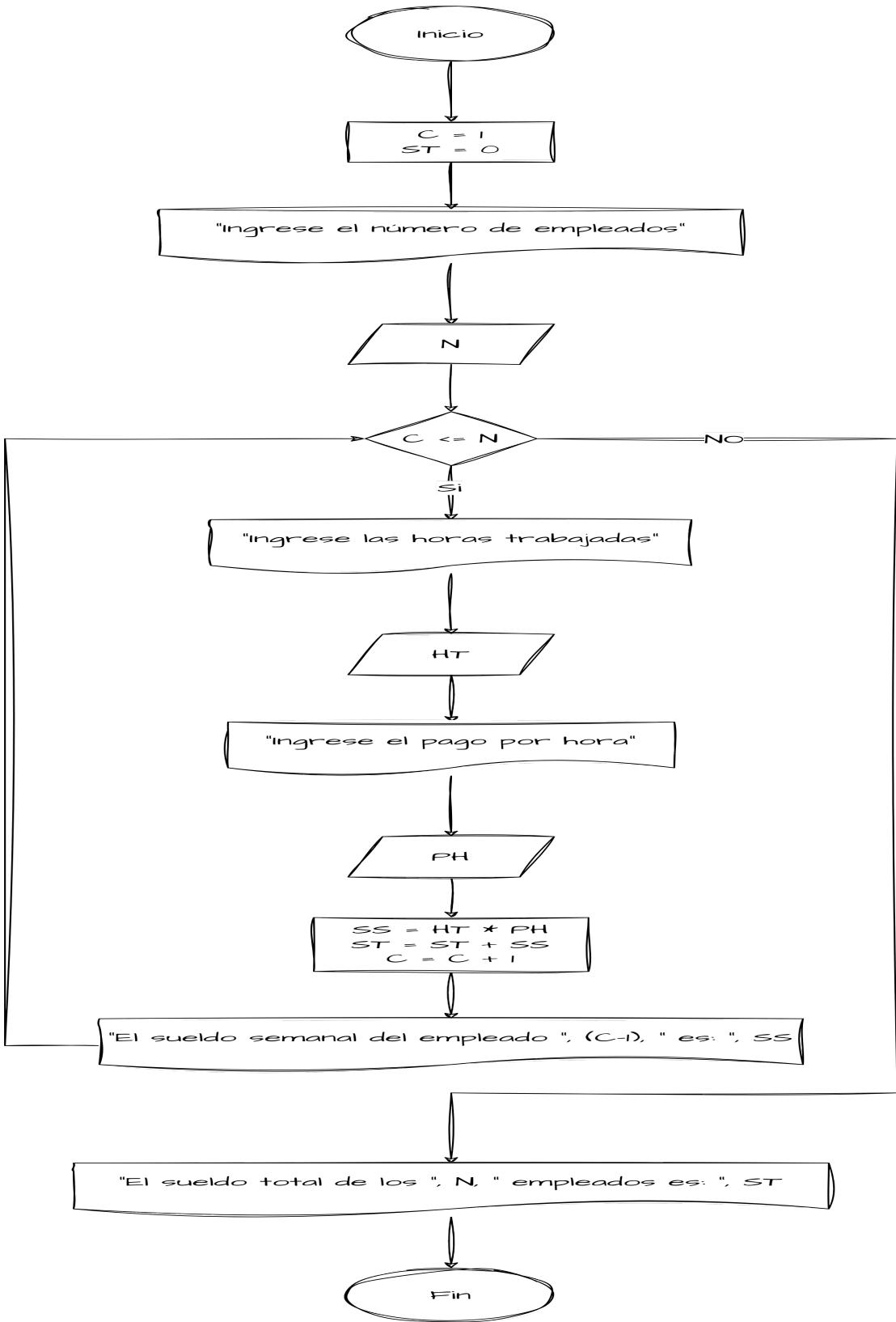
Entero: N, C

Real: HT, PH, SS, ST

Algoritmo:

1. Inicio
2. Hacer C=1
3. Hacer ST=0
4. Escribir "Ingrrese el número de empleados"

```
5. Leer N
6. Mientras C<=N Entonces
    Inicio
        Escribir "Ingrese las horas trabajadas"
        Leer HT
        Escribir "Ingrese el pago por hora"
        Leer PH
        Hacer SS=HT*PH
        Hacer ST=ST+SS
        Escribir "El sueldo semanal del empleado ", C, " es: ", SS
        Hacer C=C+1
    Fin
7. Escribir "El sueldo total de los ", N, " empleados es: ", ST
8. Fin
```



while_2.png

Resolviendo con el ciclo Hasta Que 2

Nombre del Algoritmo: Sueldo Semanal de N Trabajadores

Definición de variables

Entero: N, C

Real: HT, PH, SS, ST

Algoritmo:

1. Inicio
2. Hacer C=1
3. Hacer ST=0
4. Escribir "Ingrese el número de empleados"
5. Leer N
6. Hacer
 Inicio
 Escribir "Ingrese las horas trabajadas"
 Leer HT
 Escribir "Ingrese el pago por hora"
 Leer PH
 Hacer SS=HT*PH
 Hacer ST=ST+SS
 Escribir "El sueldo semanal del empleado ", C, " es: ", SS
 Hacer C=C+1
 Fin
 Hasta Que C>N
7. Escribir "El sueldo total de los ", N, " empleados es: ", ST
8. Fin

Resolviendo con el ciclo Para 2

Nombre del Algoritmo: Sueldo Semanal de N Trabajadores

Definición de variables

Entero: N, C

Real: HT, PH, SS, ST

Algoritmo:

1. Inicio
2. Hacer ST=0
3. Escribir "Ingrese el número de empleados"

```
4. Leer N
5. Para C=1 Hasta N Hacer // [Opcional] Con Paso 1
    Inicio
        Escribir "Ingrese las horas trabajadas"
        Leer HT
        Escribir "Ingrese el pago por hora"
        Leer PH
        Hacer SS=HT*PH
        Hacer ST=ST+SS
        Escribir "El sueldo semanal del empleado ", C, " es: ", SS
    Fin
6. Escribir "El sueldo total de los ", N, " empleados es: ", ST
7. Fin
```