

Tema 2. Diseño de algoritmos

Septiembre 2025

Índice

1	Objetivo	2
2	Definición, planteamiento y modelado del problema.	2
2.1	Ciclo de vida de un programa	2
2.1.1	Etapas del ciclo de vida de un programa	2
3	Desarrollo de algoritmos	4
3.1	Algoritmo	4
3.1.1	Representacion de un algoritmo	4
3.2	Conceptos relevantes en la construcción de algoritmos:	6
3.2.1	Operadores	6
3.2.2	Jerarquía	7
3.2.3	Variables	7
3.2.4	Operadores lógicos y relacionales	7
3.2.5	Jerarquía de operación en forma global	8
4	Diagramas de flujo y Pseudocodigo	8
5	Bibliografía	10

1 Objetivo

El alumno construirá algoritmos para resolver problemas.

2 Definición, planteamiento y modelado del problema.

A continuación se abordarán los aspectos previos más relevantes, los cuales permitirán introducirnos a los conceptos esenciales de los algoritmos.

2.1 Ciclo de vida de un programa

El proceso de resolución de un problema con una computadora conduce a la escritura de un programa y a su ejecución. Aunque el proceso de diseñar programas es, esencialmente, un proceso creativo, se puede considerar una serie de fases o pasos comunes que generalmente deben seguir todos los programadores.

2.1.1 Etapas del ciclo de vida de un programa

Análisis del problema :

Esta fase requiere una clara definición, donde se contemple exactamente lo que debe hacer el programa y el resultado o solución deseada. El problema se analiza teniendo presente la especificación de los requisitos dados por el cliente de la empresa o por la persona que encarga el programa. En esta etapa se define claramente los datos de entrada, el proceso de cálculo de la información y los resultados de salida, es por ello que es fundamental conocer los siguientes conceptos:

- **Datos de entrada:** Es la información requerida para resolver el problema. Se debe considerar el medio por el cuál se obtendrán los datos (teclado, disco, pantalla, etc.), las

variables necesarias para leer la información, el tipo de datos (numérico, texto, carácter, booleano), el orden y formato de lectura de la información (si se manda un mensaje antes de leer el dato o no).

- **Proceso:** Aquí se debe considerar principalmente la mejor manera en que se puede calcular la información de salida
- **Resultados de salida:** Es la información que se obtendrá al resolver el problema. Aquí se debe considerar la manera en que se presentará dicho resultado (texto, datos numéricos enteros o con una parte fraccionaria, ordenados alfabéticamente, de mayor a menor, etc.). Se deberá establecer claramente lo que se quiere y la forma en que se presentará

Para poder identificar y definir bien un problema es conveniente responder a las siguientes preguntas:

1. ¿Qué entradas se requieren? (tipo de datos con los cuales se trabaja y cantidad).
2. ¿Cuál es la salida deseada? (tipo de datos de los resultados y cantidad).
3. ¿Qué método produce la salida deseada?
4. Requisitos o requerimientos adicionales y restricciones a la solución.

Ejemplo

Escribir un programa para leer la base y altura de un rectángulo y calcular su área.

Indicar y explicar cuáles son:

1. Los datos de entrada
2. El proceso
3. Los resultados de salida

Respuesta:

- *Datos de entrada:* Los datos de entrada son base y altura, son datos numéricos, números reales porque tienen parte entera y fraccionaria, los datos los proporciona el usuario al momento de ejecutar el programa, el orden de lectura de los mismos es primero la base y después la altura (aunque en este caso no afecta si primero se lee la altura y después la base).
- *Proceso:* Los datos de entrada se leen por medio del teclado al momento de correr el programa ya que no se indica una forma especial de lectura, se sabe que: $area = base \times altura$.
- *Salida:* En este caso particular es el resultado de multiplicar la base por la altura y se presentará con un dato numérico que tiene parte entera y decimal.

Diseño del algoritmo :

En la etapa de análisis del problema se determina qué hace el programa; en la etapa de diseño, cómo lo hace.

Una vez analizado el problema, se diseña una solución que conducirá a un algoritmo que resuelva el problema. Los métodos más eficaces para el proceso de diseño se basan en el conocido divide y vencerás. Es decir, la resolución de un problema complejo se realiza dividiendo el problema en subproblemas y a continuación dividiendo estos subproblemas en otros de nivel más bajo, hasta que pueda ser implementada una solución en la computadora.

Codificación :

En la etapa de codificación se realiza la escritura en un lenguaje de programación de la representación del algoritmo desarrollada en las etapas precedentes. Dado que el diseño de un algoritmo es independiente del lenguaje de programación utilizado para su implementación,

puede ser escrito con igual facilidad en un lenguaje o en otro.

La solución se escribe en la sintaxis del lenguaje de alto nivel (por ejemplo, C/C++) y se obtiene un programa fuente que se compila a continuación.

De tal manera que, se implementa el algoritmo en un código escrito en un lenguaje de programación, reflejando las ideas desarrolladas en las fases de análisis y diseño.

Ejecución, verificación y depuración :

La verificación de un programa es el proceso de ejecución del programa con una amplia variedad de datos de entrada, llamados datos de test o prueba, que determinarán si el programa tiene o no errores (denominados bugs, en inglés). Para realizar la verificación se debe desarrollar una amplia gama de datos de prueba: valores normales de entrada, valores extremos de entrada que comprueben los límites del programa y valores de entrada que comprueben aspectos especiales del programa. El programador busca errores de las etapas anteriores y los elimina. Comprobará que mientras más tiempo se gaste en la fase de análisis y diseño, menos se gastará en la depuración del programa.

La depuración es el proceso de encontrar los errores del programa y corregir o eliminar dichos errores. Cuando se ejecuta un programa pueden producirse tres tipos de errores:

- *Errores de compilación:* Se producen normalmente por un uso incorrecto de las reglas del lenguaje de programación y suelen ser errores de sintaxis. Si existe un error de sintaxis, la computadora no puede comprender la instrucción, en consecuencia se imprimen una lista de todos los errores encontrados durante la compilación.
- *Errores de ejecución:* Se producen por instrucciones que la computado-

ra puede comprender pero no ejecutar. Ejemplos típicos son: división por cero y raíces cuadradas de números negativos. En estos casos se detiene la ejecución del programa y se imprime un mensaje de error.

- *Errores lógicos*: Se producen en la lógica del programa y la fuente del error suele ser el diseño del algoritmo. Estos errores son los más difíciles de detectar, ya que el programa puede funcionar y no producir errores de compilación ni de ejecución, y sólo puede advertirse el error por la obtención de resultados incorrectos. En este caso se debe volver a la fase de diseño del algoritmo, modificar el algoritmo, cambiar el programa fuente y compilar y ejecutar una vez más.

Mantenimiento :

El programa se actualiza y modifica cada vez que sea necesario, de modo que se cumplan todas las necesidades de cambio de sus usuarios.

Documentación :

Escritura de las diferentes fases del ciclo de vida del software, esencialmente el análisis, diseño y codificación, unidos a manuales de usuario y de referencia, así como normas para el mantenimiento.

3 Desarrollo de algoritmos

3.1 Algoritmo

La palabra algoritmo se deriva de la traducción al latín de la palabra Alkhôwarîzmi, nombre de un matemático y astrónomo árabe que escribió un tratado sobre manipulación de números y ecuaciones en el siglo IX.

Un algoritmo es el conjunto de pasos ordenados en forma lógica que se ejecutan para llevar a cabo una actividad o resolver un problema.

Todos los días se usan algoritmos para realizar tareas cotidianas en donde los pasos del algoritmo se ejecutan sin reparar en ellas, porque son algoritmos aprendidos, integrados por actividades que se realizan a diario. Es posible tener algoritmos para resolver problemas complejos o problemas a los que nunca nos hemos enfrentado, en cuyo caso es necesario analizar el problema, determinar los elementos que se tienen y los resultados a los que se quiere llegar, pasando por los posibles estados no previstos en donde se deben tomar decisiones de acuerdo a la situación que se presente.

Características de un algoritmo:

- Preciso (indica el orden de realización en cada paso).
- Definido (si se sigue dos veces, obtiene el mismo resultado cada vez).
- Finito (tiene fin, un número determinado de pasos).

Ejemplos de algoritmos son: instrucciones para montar en una bicicleta, hacer una receta de cocina, obtener el máximo común divisor de dos números, etcétera.

3.1.1 Representación de un algoritmo

Existen varios métodos para la representación de algoritmos entre los cuales se pueden mencionar:

Descripción narrada :

Este método de representación de algoritmos utiliza el lenguaje natural hablado o escrito para describir los pasos que se deben seguir al resolver el problema. Cada una de las instrucciones puede ser una frase u oración narrada en forma concreta de lo que se debe hacer.

Ejemplo

Realiza la descripción narrada de las instrucciones para leer la base y altura de un rectángulo y calcular su área.

```

inicio
leer la base
leer la altura
calcular el area
(multiplicando base por altura)
imprimir el area.
fin

```

Diagrama de flujo :

Es una manera gráfica de representar los algoritmos, usando símbolos geométricos para cada uno de los pasos que integran el algoritmo. Por medio del diagrama de flujo se ilustra de manera gráfica la secuencia de operaciones que se realizan para resolver el problema. Un diagrama de flujo solamente tiene un principio y un final. Los símbolos más utilizados se muestran a continuación:

Símbolos básicos de los diagramas de flujo			
Símbolo	Significado	Símbolo	Significado
	Inicio o Fin de diagrama		Dirección de flujo
	Actividad		Entrada de datos
	Decisión		Conector para unir el flujo a otra parte del diagrama.
			Impresión o desplegado de información

Figura 1: Símbolos básicos empleados en un diagrama de flujo y su descripción.

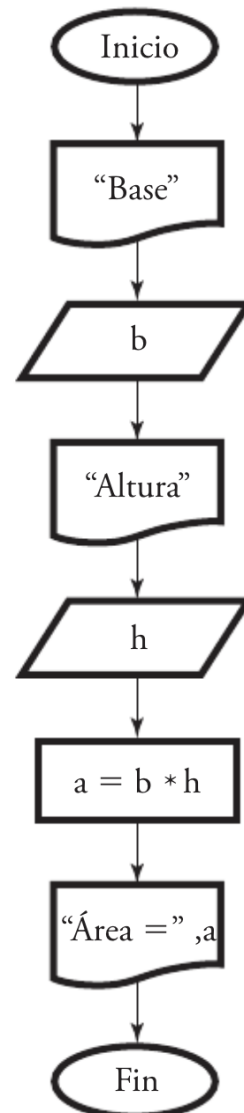
Recomendaciones para elaborar un diagrama de flujo:

- Comienzo del diagrama en la parte superior
- Fin del diagrama parte inferior
- El flujo de las secuencias debe ir de arriba - abajo
- Evitar cruces de líneas de flujo
- Símbolo de inicio y fin deben de estar solo una vez

Ejemplo

Siguiendo el ejemplo anterior, escribir un programa para leer la base y altura de un rectángulo y calcular su área.

Realiza su diagrama de flujo:



Observar que los mensajes se deben encerrar entre comillas y que es posible imprimir mensajes y el valor de variables en una misma línea, separándolos por coma.

Pseudocódigo :

El pseudocódigo (también, seudocódigo) es una herramienta de programación en la que las instrucciones se escriben en palabras similares al inglés o español, que facilitan tanto la escritura como la lectu-

ra de programas. En esencia, el pseudocódigo se puede definir como un lenguaje de especificaciones de algoritmos.

Se puede decir que es la manera aproximada de representar algoritmos usando para ello palabras de un idioma natural para simular un lenguaje de programación, indicando los pasos que integran dicho algoritmo. El pseudocódigo está hecho para que las personas representen o comprendan de una manera clara y sencilla el funcionamiento del algoritmo. Aunque las computadoras no pueden entender esta forma de representación porque no es un lenguaje de programación.

El pseudocódigo no obedece a reglas sintácticas de ningún idioma en particular y puede cambiar de una fuente de información a otra porque no es una representación estandarizada, ya que una persona puede representar los pasos del algoritmo de una manera muy cercana al lenguaje de programación que utilizará al momento de codificar el programa, o bien representar esos pasos del algoritmo apegado a una narración de lo que se hace, con lenguaje natural y apoyándose con algunos símbolos matemáticos.

El pseudocódigo facilita la programación y solución de problemas, es fácil de utilizar y manejar, es independiente del lenguaje de programación que se usará para la representación del problema, lo cual lo hace transportable, los algoritmos representados con pseudocódigo son compactos ya que a diferencia de los diagramas de flujo no requieren de mucho espacio. Permite la representación de algoritmos de manera clara, concisa y comprensible para todas las personas, ya que no se requiere conocer algún lenguaje de programación para entender los algoritmos representados con pseudocódigo.

Ejemplo

Siguiendo el ejemplo anterior, escribir el pseudocódigo para leer la base y altura de un rectángulo y calcular su área:

```
INICIO
ESCRIBIR 'Ingrese la base del
rectangulo:'
LEER base
ESCRIBIR 'Ingrese la altura
del rectangulo:'
LEER altura

area <- base * altura

ESCRIBIR 'El area del
rectangulo es:', area
FIN
```

3.2 Conceptos relevantes en la construcción de algoritmos:

Para la construcción y representación de algoritmos es fundamental tener claro los siguientes conceptos:

3.2.1 Operadores

- Paréntesis $\rightarrow ()$
- Potencia $\rightarrow ^$
- Multiplicación $\rightarrow *$
- División $\rightarrow /$
- Modulo (residuo) $\rightarrow \%$
- Suma $\rightarrow +$
- Resta $\rightarrow -$
- Operador de asignación $\rightarrow =$

En la siguiente tabla mostramos aquellos operadores mas relevantes y su descripción:

Tabla 2.1			
Operador	Operación	Ejemplo	Significado de la operación
+	Suma	$c = a + b$	Suma el valor de a con b y asigna el resultado a c .
-	Resta	$c = a - b$	Al valor de a le resta el valor de b y el resultado de la operación se lo asigna a c .
*	Multiplicación	$c = a * b$	Multiplifica a por b y asigna el resultado a c .
/	División	$c = a/b$	Divide el valor de la variable a entre el valor de b y asigna el resultado a la variable c .
^	Potenciación	$c = a ^ b$	Eleva a la potencia b el valor de a (a^b) y se asigna el resultado a c . Ejemplo si $a=5$ y $b=3$ entonces $c = 5 ^ 3 = 5^3 = 125$.
mod	Módulo o residuo	$c = a \text{ mod } b$	El valor de c es el residuo de dividir a entre b . Ejemplos: $c = 5 \text{ mod } 3 = 2$; $c = 13 \text{ mod } 8 = 5$

Figura 2: Operadores más relevantes.

3.2.2 Jerarquía

Las expresiones matemáticas no necesariamente se evalúan de corrido de izquierda a derecha, sino que los operadores aritméticos tienen diferente prioridad de evaluación y dependiendo del operador aritmético, la colocación de los mismos y la forma en que esté agrupada la información, marcará el orden de operación y por lo tanto el resultado obtenido. La tabla siguiente muestra la jerarquía de operación de los principales operadores matemáticos:

Tabla 2.2	
Operador	Jerarquía de operación
()	1a.
^	2a.
*, /, mod	3a.
+, -	4a.

Figura 3: Jerarquía de operaciones.

Lo que implica que primeramente se evalúa la información que está encerrada entre paréntesis, posteriormente la potenciación, después tienen la misma jerarquía de operación la multiplicación, división y módulo para finalmente evaluar la suma y la resta

3.2.3 Variables

En los distintos lenguajes de programación se deben definir las variables como enteros, reales, caracter, cadenas, booleanas, etc. El resultado

de una operación dependerá de los tipos de datos de las variables. Por ejemplo si se definen a las variables: a , b y c como enteras, donde $a = 7$ y $b = 3$, entonces $c = a/b = 2$, pero si c es real, entonces $c = a/b = 2,5$.

También el símbolo para indicar las operaciones matemáticas puede cambiar de un lenguaje a otro (por ejemplo, la palabra “*mod*” o el símbolo para indicar la potenciación no se representan igual en C++, que en Java).

Principales tipos de datos :

- Numericos
 - Enteros
 - Reales
- Alfanuméricos
 - Caracteres
- Datos lógicos
 - Booleanos

Contador y acumulador :

- Contador: Lleva la cuenta de determinadas acciones que se pueden solicitar durante la resolución de un problema (numero de repeticiones o eventos).
- Acumulador: Almacena la suma de valores de manera progresiva (total de una suma acumulada).

3.2.4 Operadores lógicos y relacionales

- **Operadores relacionales:** Se usan para comparar el valor de dos variables, el resultado de la comparación puede ser falso o verdadero. Por lo general este tipo de operadores se utilizan en las proposiciones de condiciones o para controlar las veces que se ejecutan las instrucciones que se encuentran en ciclos.

El símbolo del operador no es el mismo en los diferentes lenguajes de programación.

Operador	Significado	Ejemplo
>	Mayor que	$x > y$
<	Menor que	$x < y$
>=	Mayor o igual que	$x \geq y$
<=	Menor o igual que	$x \leq y$
=	Igual a	$x = y$
!=	Diferente de	$x \neq y$

Figura 4: Operadores relacionales.

- **Operadores lógicos:** Se utilizan en instrucciones condicionales o en ciclos en donde se requiere determinar si son falsas o verdaderas más de una proposición. Los operadores *and* y *or* son binarios, lo cual significa que se utilizan para relacionar dos proposiciones y el operador *not* es unario y se utiliza para negar o complementar una proposición. La siguiente tabla muestra los operadores lógicos básicos.

Operador	Significado	Ejemplo
and	y	$(x > y) \text{ and } (m == n)$
or	o	$(x > y) \text{ or } (m == n)$
not	no	$\text{not } (m == n)$

Figura 5: Operadores lógicos.

3.2.5 Jerarquía de operación en forma global

Operador	Jerarquía de operación
()	1a.
^	2a.
*, /, mod	3a.
+, -	4a.
>, >=, <, <=	5a.
=, !=	6a.
not	7a.
and	8a.
or	9a.

Figura 6: Jerarquía de operaciones completa.

4 Diagramas de flujo y Pseudocódigo

Los algoritmos también pueden representarse usando condicionales, ciclos, procedimientos y funciones. Existe representación grafica usada en los diagramas de flujo para indicar cada una de estas instrucciones compuestas como se muestra a continuación.

Para diagramas de flujo:

Decisiones (condición simple) :

Si la condición se cumple se ejecuta el conjunto de Sentencias A y sino se cumple no se ejecuta ninguna.

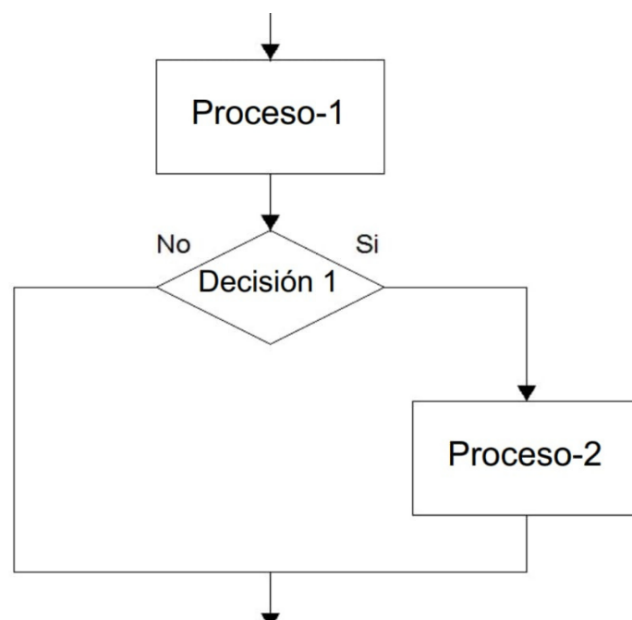


Figura 7: Diagrama de flujo condición simple.

Decisiones (condición doble) :

Si la condición se cumple se ejecuta el conjunto de Sentencias A y sino se cumple se ejecutan las sentencias B.

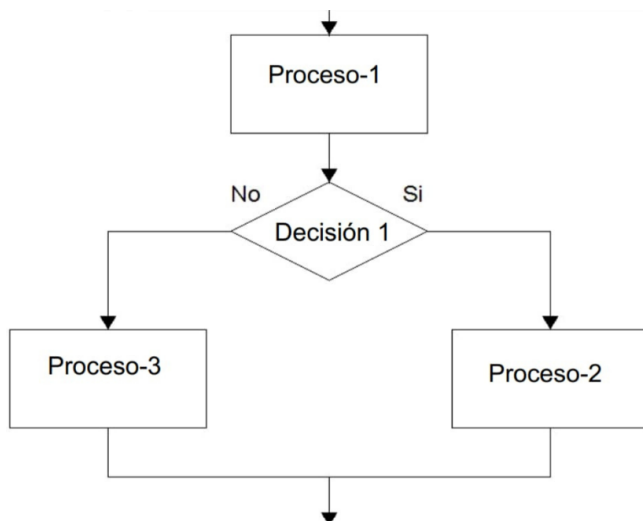


Figura 8: Diagrama de flujo condición doble.

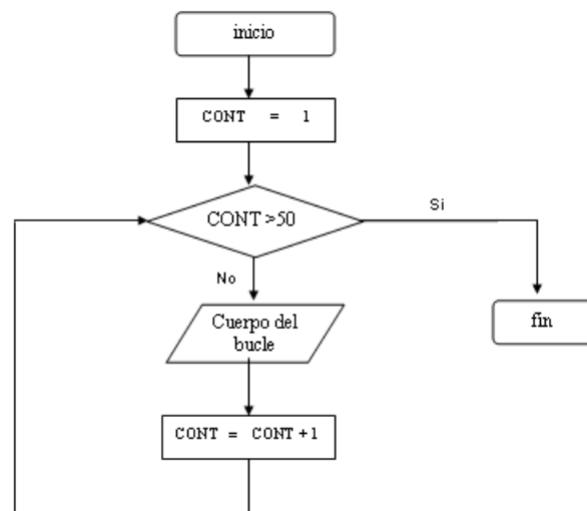


Figura 10: Diagrama de flujo estructura de repetición.

Estructuras de selección múltiple :

Dependiendo del valor de la expresión de selección se ejecutará el bloque de instrucciones del caso 1, 2, ... ó n. Cuando el valor de la expresión de selección no está dentro de las opciones contempladas, podría no ejecutarse ninguna instrucción o bien otro bloque de instrucciones alternas.

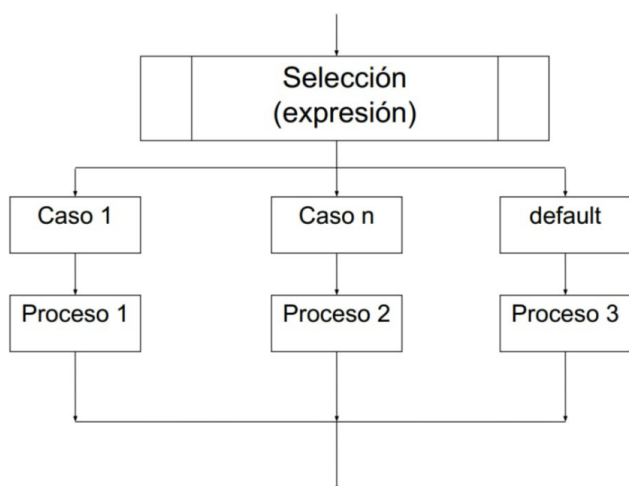


Figura 9: Diagrama de flujo estructuras de selección múltiple.

Estructuras de repetición :

Es un ciclo que ejecuta una serie de sentencias (o instrucciones) mientras se cumpla la condición indicada en el rombo, una vez que deje de cumplirse la condición, se sale del ciclo.

Para pseudocódigo:

Decisiones (condición simple) :

Tomar un camino según una condición dada (verdadero / falso). Si la condición se cumple se ejecuta el conjunto de Sentencias A y sino se cumple no se ejecuta ninguna.

ASIGNAR PROCESO 1
SI DECISIÓN 1 (ES VERDADERA) ENTONCES
ASIGNAR PROCESO 2
FIN DE DECISION

Decisiones (condición doble) :

Tomar distintos caminos según una condición dada (verdadero / falso). Si la condición se cumple se ejecuta el conjunto de Sentencias A y sino se cumple se ejecutan las sentencias B

ASIGNAR PROCESO 1
SI DECISIÓN 1 (ES VERDADERA) ENTONCES
ASIGNAR PROCESO 2
SI NO
ASIGNAR PROCESO 3
FIN DE DECISION

Estructuras de selección múltiple :

Variable puede tomar varios valores y según el caso se ejecuta una acción distinta

SI SELECTOR IGUAL
VALOR 1: HACER ACCIÓN 1
VALOR 2: HACER ACCIÓN 2
VALOR 3: HACER ACCIÓN 3
VALOR 4: HACER ACCIÓN 4
DE OTRA FORMA: HACER ACCIÓN N
FIN DEL CONDICIONAL (SELECTOR)

Estructuras de repetición :

Ejecutar un bloque de código varias veces
dependiendo de una condición o contador

Nombre de algoritmo: ciclo1

Variables: X, entero

Inicio

asignar $X \leftarrow 1$

mientras $X \leq 10$ repetir

mostrar X

asignar $X \leftarrow X + 1$

fin ciclo

Fin

5 Bibliografía

- Metodología de la Programación, Algoritmos, Diagramas de Flujo y Programas Osvaldo Cairó Alfaomega
- Fundamentos de programación: algoritmos, estructuras de datos y objetos / Luis Joyanes Aguilar (Quinta edición). (2020).
- Jiménez Hernández, E. M., & Alvarado Zamora, L. N. (2015). Fundamentos de programación: diagramas de flujo, diagramas N-S, Pseudocódigo y Java / José Alfredo Jiménez Murillo, Eréndida Miriam Jiménez Hernández, Laura Nelly Alvarado Zamora. Alfaomega.