



Universidad Nacional Autónoma de México
Facultad de Ingeniería
Computación para Ingenieros
Tema 5
FUNDAMENTOS DE ALGORITMOS

5. Fundamentos de algoritmos

Objetivo: Explicar la importancia de llevar un método formal para resolver problemas en la computadora; así mismo, aplicar dicho método en la resolución de problemas.

5. Fundamentos de algoritmos

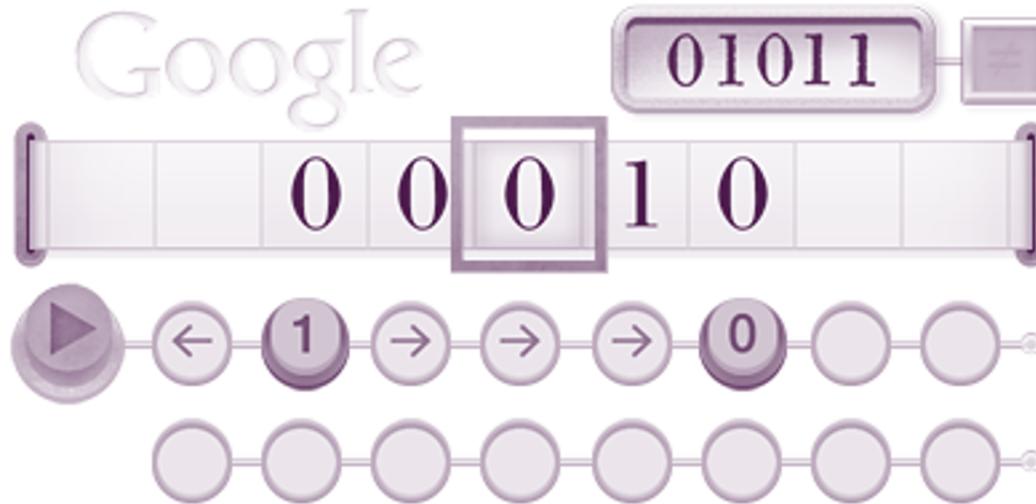
**5.1 La Computabilidad y Concepto de algoritmo:
Máquina de Turing**

5.2 Elementos de los algoritmos y tipos de datos

5.3 Representación de los algoritmos (diagrama de flujo y pseudocódigo)

5.4 Estructuras básicas (secuencia, condicional e iteración)

5.5 Resolución de problemas básicos de ingeniería



5.1 La computabilidad y el concepto de algoritmo: Máquina de Turing

5.1 La computabilidad y el concepto de algoritmo: Máquina de Turing

La teoría de la computabilidad, también denominada teoría de la recursión, es una de las cuatro partes que constituyen la lógica matemática (junto con la teoría de conjuntos, la teoría de modelos y la teoría de la demostración).

La teoría de la computabilidad se ocupa del estudio y clasificación de las relaciones y aplicaciones computables.

Además, la teoría de la computabilidad, junto con la teoría de autómatas, lenguajes y máquinas, es el fundamento de la informática teórica y ésta, a su vez, de la industria de las computadoras.

Problemas computables

Un problema matemático es computable si éste puede ser resuelto, en principio, por un dispositivo computacional. Algunos sinónimos comunes para el término computable son resoluble, decidible o recursivo.

No todos los problemas matemáticos son resolubles y, por ende, no todos los problemas son computables.

Alrededor de la década de 1930, antes de las computadoras, matemáticos de todo el mundo pronunciaron diferentes definiciones de lo que significaba computable.

La inquietud de demostrar qué podía ser o no computable provino de la creencia de David Hilbert de que todo en las matemáticas puede ser axiomatizado. Hilbert dejó plasmada esta interrogante en el **Entscheidungsproblem**.

El entscheidungsproblem (problema de decisión en alemán) es un reto que David Hilbert lanzó en 1928. El problema consistía en encontrar un algoritmo que teniendo como entrada la descripción de un lenguaje formal y una declaración matemática, produjera como salida “Verdadero” o “Falso” dependiendo de si la declaración se cumplía o no.

For the mathematician there is no Ignorabimus, and, in my opinion, not at all for natural science either. ... The true reason why [no one] has succeeded in finding an unsolvable problem is, in my opinion, that there is no unsolvable problem.

En 1936 Alonzo Church y en 1937 Alan Turing publicaron artículos independientes demostrando que es imposible decidir algorítmicamente si una declaración aritmética es verdadera o falsa y, por lo tanto, encontrar una solución general para el entscheidungsproblem era imposible.

Los estudios que dan como conclusión lo irresoluble del problema planteado por Hilbert son conocidos como el Teorema de Church y la Máquina de Turing.

Por lo tanto, la Teoría de la computabilidad es la parte de la computación que estudia los problemas de decisión que pueden ser resueltos con un algoritmo o, de manera equivalente, con una máquina de Turing.

Los problemas que pueden ser resueltos por un algoritmo y/o la máquina de Turing, se les conoce como *sistemas computables*.

Definición de algoritmo

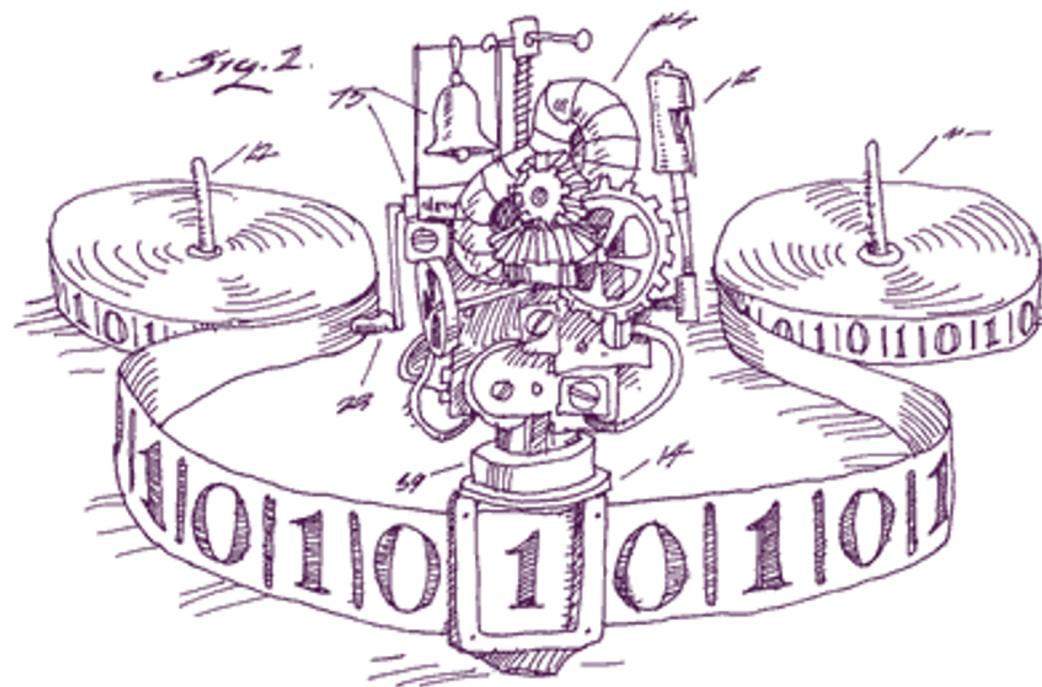
Un algoritmo es un conjunto de reglas, expresadas en un lenguaje específico, para realizar alguna tarea en general. Lo que hace especial a un algoritmo es que sus reglas pueden ser aplicadas un número ilimitado de veces sobre una situación particular.

Máquina de Turing

La máquina de Turing, descrita por Alan Turing en 1937, es un dispositivo computacional abstracto cuyo propósito es indagar el alcance y las limitaciones de los problemas que son computables.

La máquina de Turing no es un objeto físico ni uno matemático. Su arquitectura es simple así como las acciones que ésta puede llevar a cabo.

Una máquina de Turing es un tipo de máquina de estados. Posee una cinta infinita de una dimensión dividida en celdas. Cada celda puede contener un valor.



Programa contador

- La máquina inicia con el número 1011_2 (11_{10}).
- La cabeza lectora/escritora se mueve a la derecha buscando el bit menos significativo.
- Si encuentra un uno, lo cambia por cero y se mueve a la celda izquierda.
- Si lee un cero lo cambia por uno y sigue leyendo a la derecha buscando al bit menos significativo.
- El programa se detiene cuando encuentra una celda en blanco a la izquierda de una celda que contenga un cero (bit más significativo), entonces escribe 1 y el programa termina.

Una máquina de Turing debe poseer un conjunto finito de estados con, por lo menos, dos (inicial y final).

La cabeza de lectura/escritura puede moverse hacia la derecha o hacia la izquierda.

La máquina de Turing está conformada por los siguientes datos:

$$M = (Q, \Sigma, \beta, \sigma, s, F, \delta)$$

donde:

Q: se refiere al conjunto finitos de datos.

Σ : es el alfabeto de entrada.

β : es un símbolo en blanco que no está en Σ .

σ : es el alfabeto de la cinta

s: es el estado inicial.

F: es el estado final.

δ : es el conjunto de transiciones.

Por ejemplo, para el caso del programa contador se tiene:

$$Q = \{q, qvi, qvd\}$$

$$\Sigma = \{0, 1\}$$

$$\beta = \{ \}$$

$$\sigma = \{0, 1, \beta\}$$

$$s = qvd$$

$$F = qvi$$

Las transiciones δ están dadas por los estados:

$$\delta(q, 0) = (qvd, 1, \text{der})$$

$$\delta(q, 1) = (q, 0, \text{izq})$$

$$\delta(qvd, \beta) = (q, \beta, \text{izq})$$

$$\delta(qvi, \beta) = (qvi, 1, \text{alto})$$

Las transiciones de la máquina están dadas por la fórmula:

$$\delta = Q \times \Sigma \rightarrow Q' \times \Sigma \times \text{Mov}$$

donde

$$\text{Mov} = \{\text{der, izq, alto}\}$$

La fórmula de transiciones se lee como sigue: Si se está en un estado Q y se lee un símbolo que pertenece a Σ , luego entonces, en el estado Q se escribe el símbolo que pertenece a Σ , se pasa al estado Q' y se realiza el movimiento Mov.

Para el caso de la primera transición descrita para el programa contador se tiene:

$$\delta(q, 0) = (q_{vd}, 1, \text{der})$$

La transición anterior se lee como: Si se está en el estado q y se lee el símbolo 0, luego entonces, en el estado q se escribe el símbolo 1, y se mueve la cabeza hasta el estado qvd a la derecha.

La transición $\delta(q, 0) = (q_{vd}, l, \text{der})$ puede escribirse como:

$$\delta(q, 0 ; q_{vd}, l, \text{der})$$

con lo cual la máquina de Turing queda especificada por su lista de quintuplas.

Ejemplo 5.1

Se necesita comprobar que, dada una cadena de paréntesis, ésta se encuentre equilibrada y bien escrita.

Para tener una cadena de paréntesis equilibrada, por cada símbolo ')' debe tener un correspondiente símbolo '(' que lo anteceda.

Ejemplo 5.1

Para reconocer cadenas equilibradas se deben tener las siguiente consideraciones:

1. **Se busca el símbolo ')' de izquierda a derecha.**
2. **Cuando se encuentre ')', se marca con B y después:**
 - a. **Se busca hacia la izquierda el símbolo '(' y se marca con A. Se repite paso 1.**
 - b. **Si se llega al final y no se encuentra '(', la cadena está desequilibrada.**
3. **Si quedan '(' o ')' no marcados (A o B), la cadena está desequilibrada.**

Ejemplo 5.1

Las transiciones para el estado uno de la máquina de Turing se muestran a continuación:

Transición	Descripción
$q_1, x ; q_1, x, \text{der}$	Con cualquier símbolo x , $x = \{(), A, B\}$, se deja igual y se avanza a la derecha.
$q_1,) ; q_2, B, \text{izq}$	Si se encuentra el símbolo ')', se cambia por el símbolo 'B' y se mueve a la izquierda.
$q_1, \beta ; q_3, \beta, \text{izq}$	Si se termina de recorrer la cadena se revisa que todo haya sido marcado.

Ejemplo 5.1

Las transiciones para el estado dos de la máquina de Turing se muestran a continuación:

Transición	Descripción
$q2, y ; q2, y, izq$	Con cualquier símbolo y , $y = \{ \), A, B\}$, se deja igual y se avanza a la izquierda.
$q2, (; q1, A, der$	Si se encuentra el símbolo ‘(’, se cambia por el símbolo ‘A’ y se mueve a la derecha.
$q2, \beta ; q4, err, alto$	Si se termina la cadena a la izquierda sin encontrar ‘(‘, se termina el proceso. No hay equilibrio.

Ejemplo 5.1

Las transiciones para el estado tres de la máquina de Turing se muestran a continuación:

Transición	Descripción
$q_3, z ; q_3, z, \text{izq}$	Revisa hacia la izquierda ignorando z, con $z = \{A, B\}$.
$q_3, (; q_3, \text{err}, \text{alto}$	Si, en estado 3, queda algún símbolo '(', termina el proceso. No hay correspondencia.
$q_3, \beta ; q_3, q_4, \text{alto}$	Si, en estado 3, se termina la cadena entonces todo se marcó. La cadena está en equilibrio y termina el proceso.

Ejemplo 5.1

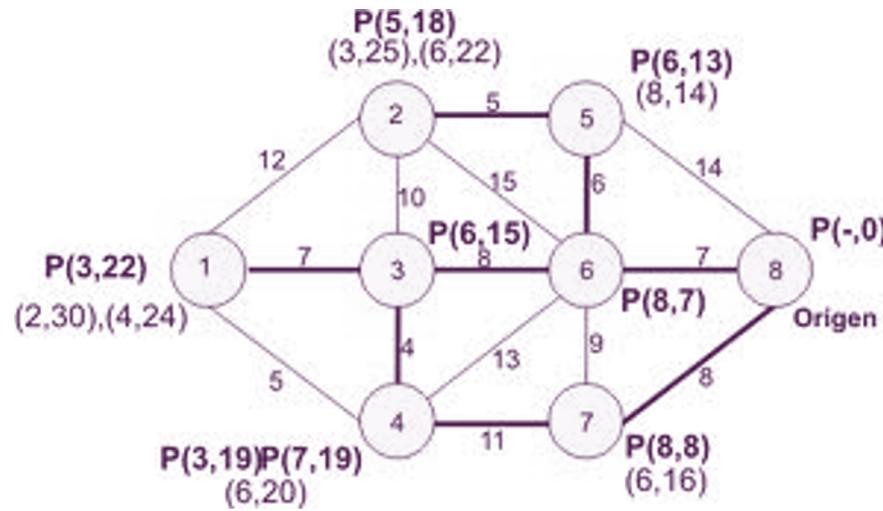
La máquina posee 4 estados. Los estados tienen la siguiente interpretación:

- **q1: estado inicial. Avanza a la derecha buscando ')'.**
- **q2: avanza a la izquierda buscando '('.**
- **q3: revisa que todo haya sido marcado.**
- **q4: estado final. Termina el proceso.**

Ejercicio 5.1

Utilizando las transiciones descritas para una máquina de Turing, comprobar si las siguientes cadenas de paréntesis están balanceadas.

Equipo	Cadena
1	((())())()
2	((())(())
3	((())(())
4	((())((())
5	(((((()))))
6	(((((()))))
7	(()()()()()
8	((())()())



5.2 Elementos de los algoritmos y tipos de datos

5.2 Elementos de los algoritmos y tipos de datos

Como ya se mencionó en el tema pasado, un algoritmo es un conjunto de reglas, expresadas en un lenguaje específico, para realizar alguna tarea en general, es decir, un conjunto de pasos, procedimientos o acciones que permiten alcanzar un resultado o resolver un problema.

Algoritmo



En las ciencias de la computación se le llama problema computable a aquella abstracción de la realidad que tiene una representación algorítmica.

Ejercicio 5.2

Figura de papel (Origami)

Problema:

- Describir una serie de instrucciones que permitan realizar la figura de papel que se encuentra descrita en el video proporcionado.

Ejercicio 5.2

Restricciones:

- Las instrucciones pueden ser tan largas como deseen pero no deben contener ningún dibujo.
- La hoja no debe hacer alusión en ningún momento a la forma de la figura final.
- No debe tener nombre.
- Se tiene un tiempo máximo de 15 minutos.

Entregables:

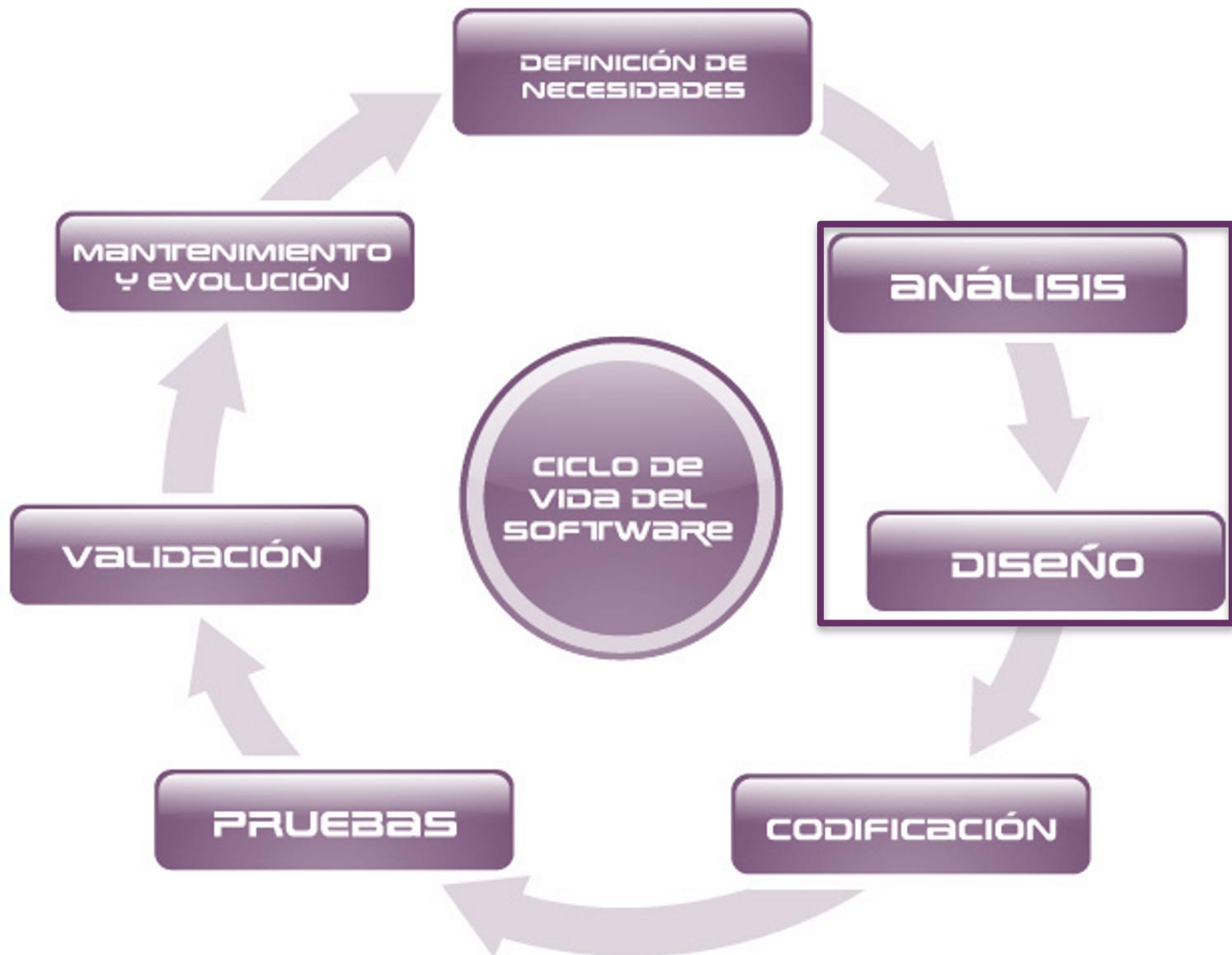
- Al final de la actividad se debe entregar la figura de papel terminada junto con las instrucciones para realizarla.

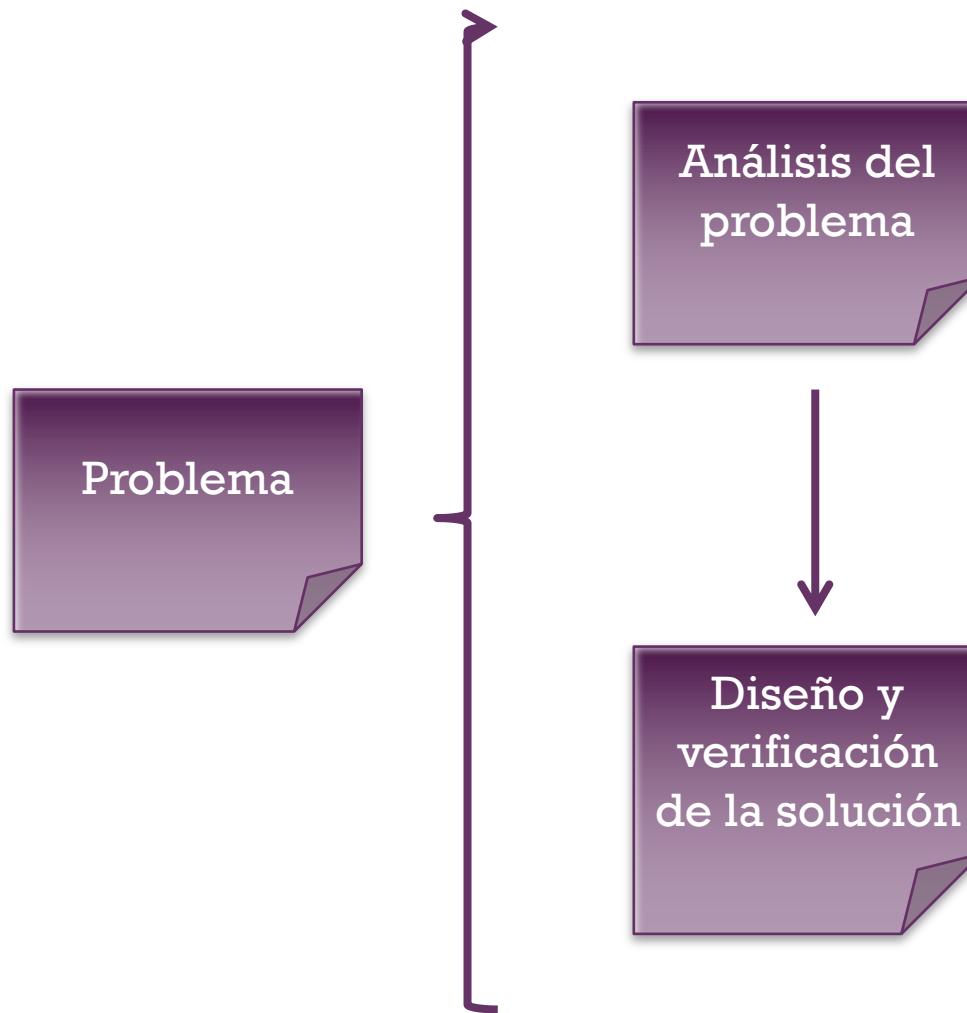
Características de un algoritmo

Un algoritmo debe ser preciso, es decir, llegar a la solución en el menor tiempo posible y sin ambigüedades.

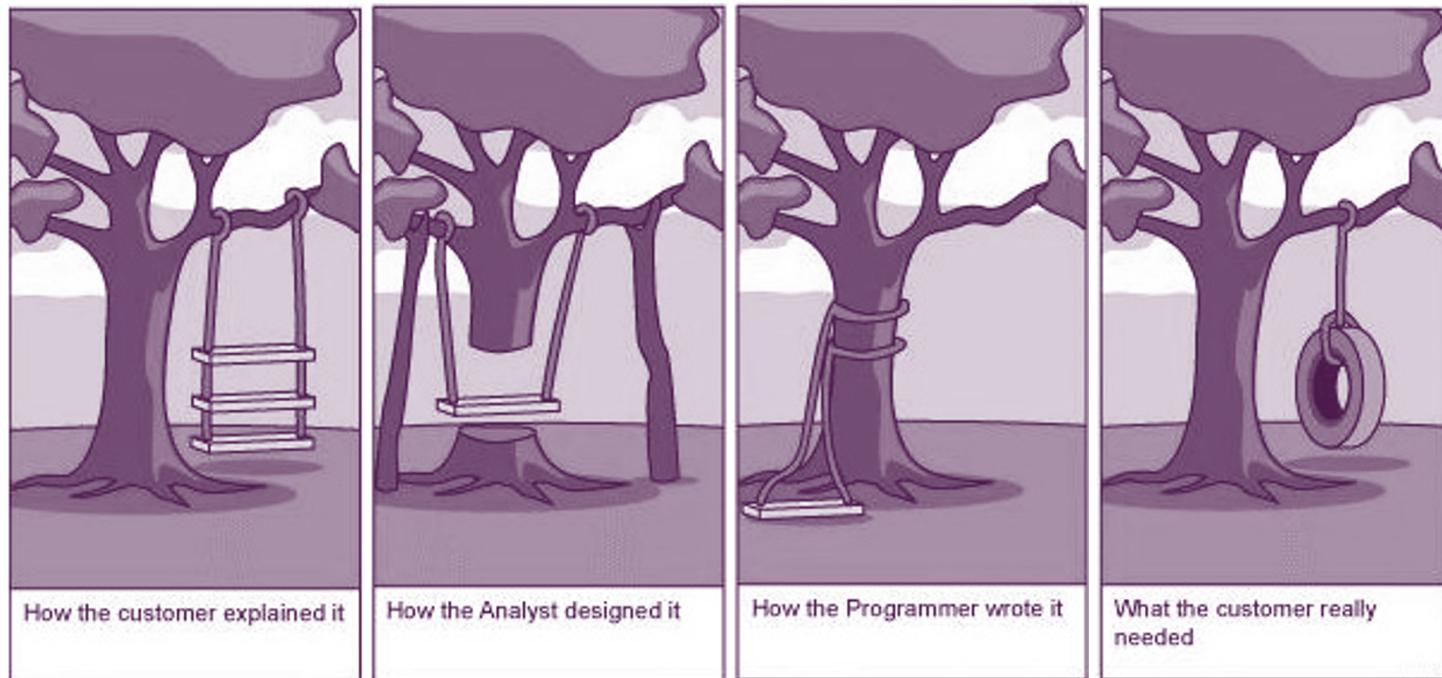
También debe ser determinista, es decir, a partir de un conjunto de datos idénticos de entrada, debe arrojar siempre los mismos resultados.

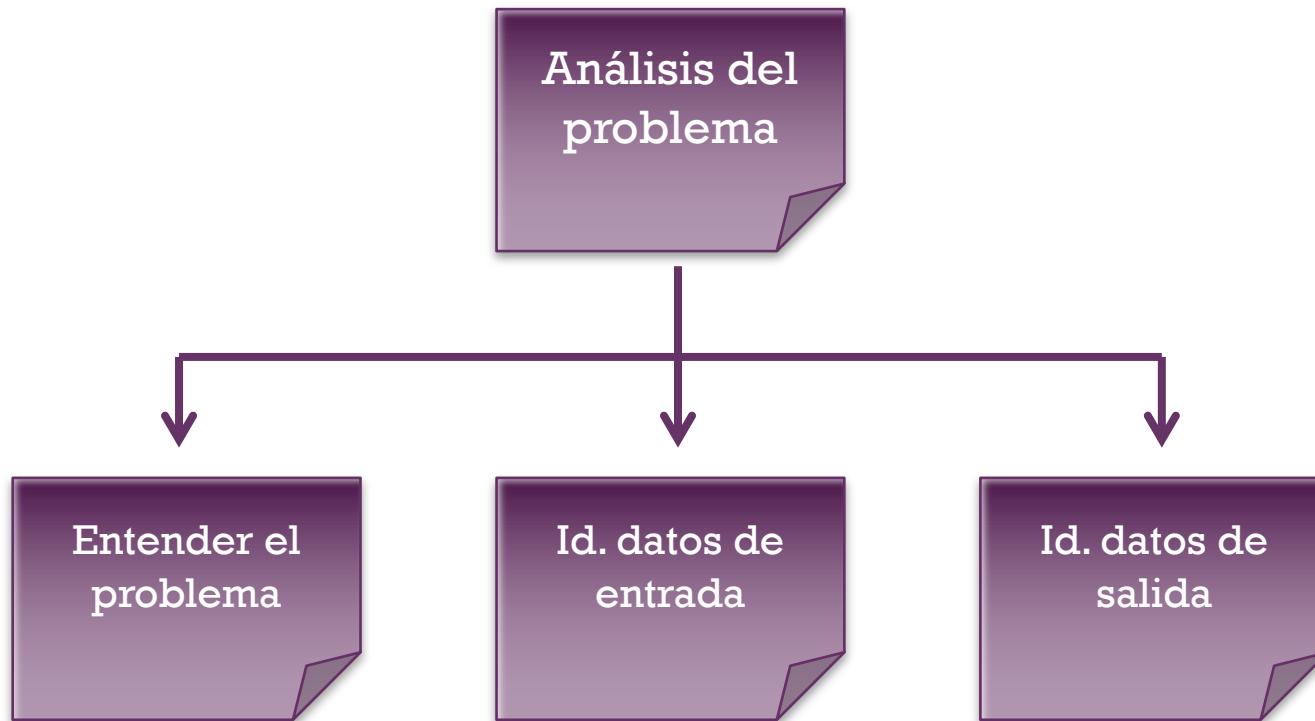
El que un proceso sea computable implica que, en algún momento, el proceso va a llegar a su fin. Un algoritmo debe ser finito, es decir, en algún momento dado debe terminar.

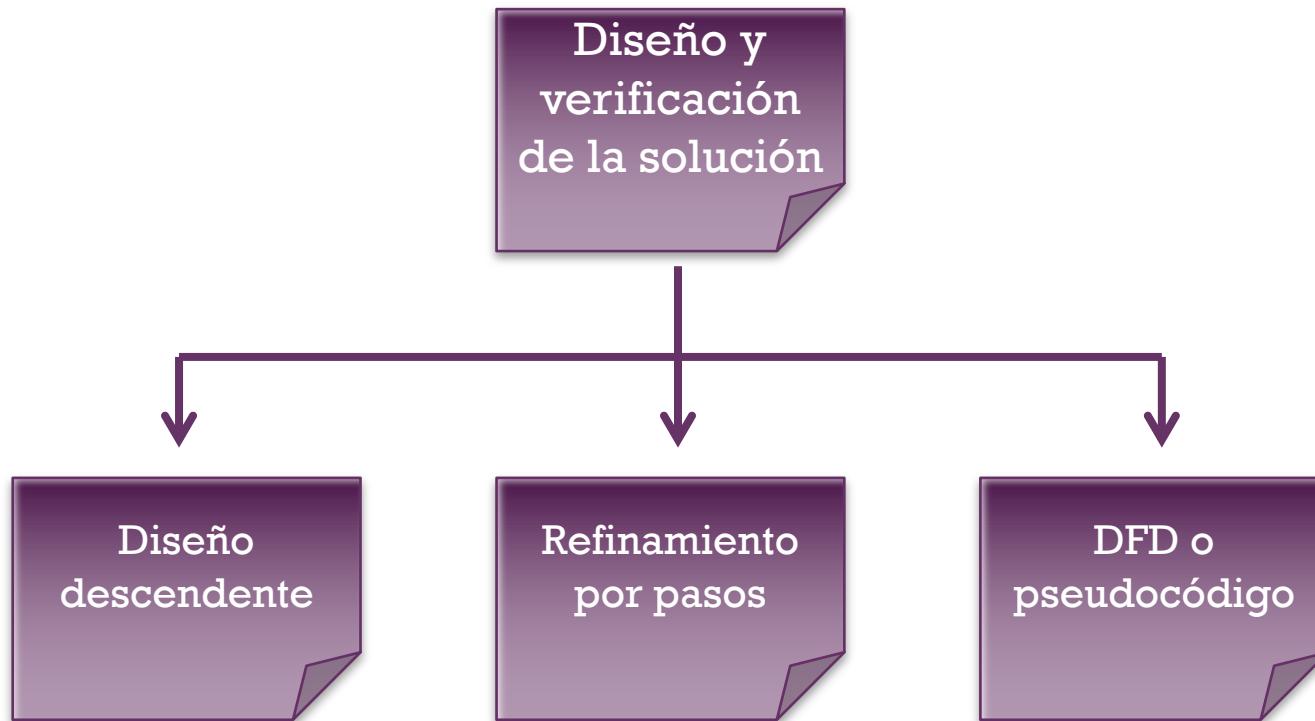




Dado un problema a resolver ...





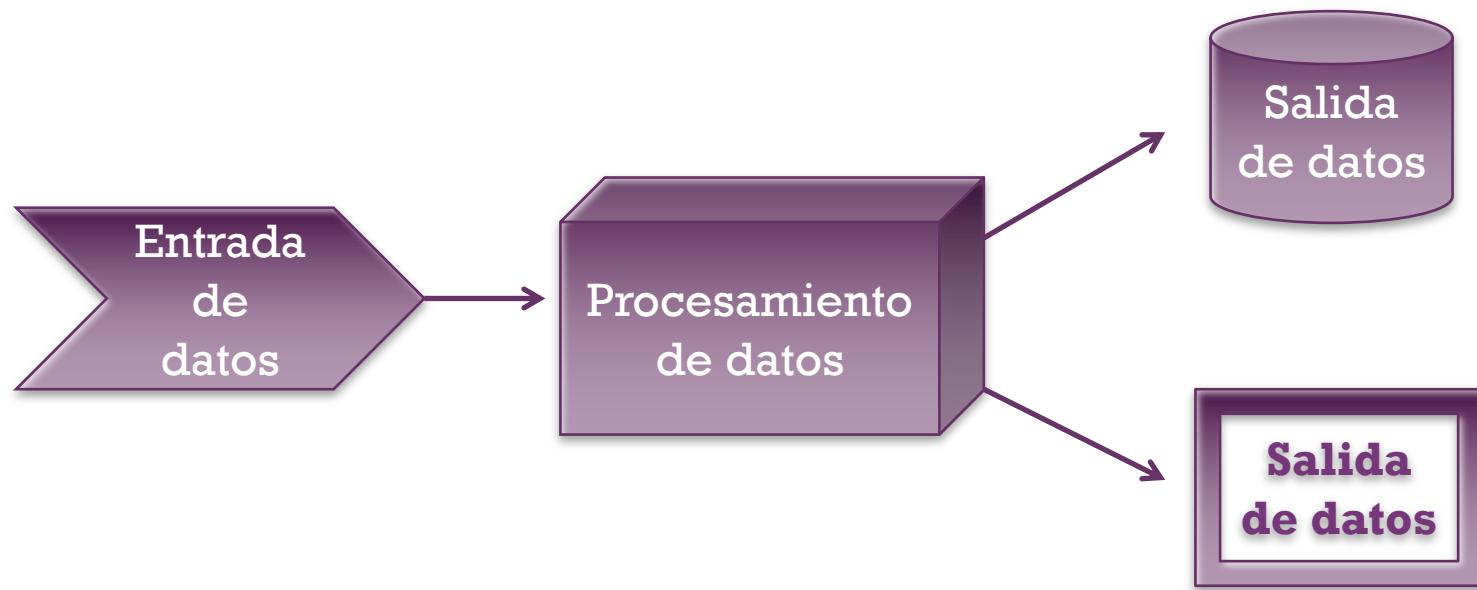


El diseño descendente hace referencia a la división del problema en problemas más sencillos de tal manera que el conjunto de soluciones forme la solución general. Este diseño se conoce como divide y vencerás.

El refinamiento por pasos se refiere a la comprobación de cada uno de los módulos en los que se dividió el problema en el diseño descendente.

El diseño del algoritmo se debe expresar en un herramienta que el programador pueda entender. Existen dos lenguajes o representaciones algorítmicas básicos: Diagramas de flujo y pseudocódigo.

Un algoritmo consta de 3 módulos básicos: módulo de entrada, módulo de procesamiento y módulo de salida.





El módulo de entrada representa la manera en la que el se obtienen los datos para resolver el problema. Se pueden solicitar al usuario, leer de un archivo, consultar una base de datos, etc.



El módulo de procesamiento representa las operaciones necesarias para obtener un resultado a partir de los datos de entrada.



El módulo de salida permiten mostrar los resultados obtenidos en el módulo de procesamiento. Los resultados pueden mostrarse en diversos sitios: pantalla, en archivo, etc.

Algoritmo para crear un algoritmo

PROGRAMA desarrollar_algoritmo (problema)

correcto = falso;

MIENTRAS (NO correcto) O (NO rápido(tiempo_ejec))

algoritmo = idear_algoritmo(problema)

correcto = analizar_exactitud(algoritmo)

tiempo_ejec= analizar_eficiencia(algoritmo)

FIN_MIENTRAS

RETURN algoritmo

FIN_PROGRAMA

Ejemplo 5.2

PROBLEMA: Determinar si un número dado es par o impar.

SOLUCIÓN:

1. Iniciar el programa.
2. Definir X.
3. Iniciar X igual a 0.
4. HACER
5. Solicitar un número entero al usuario.
6. Almacenar número en X.
7. MIENTRAS X sea igual 0
8. Obtener el módulo del número entero y guardarlo en M.
9. SI M es igual a 0.
10. IMPRIMIR “X es par”
11. DE LO CONTRARIO
12. IMPRIMIR “X es impar”.
11. Terminar el programa.

Ejemplo 5.2**Prueba de escritorio**

iteración	X	M	Imprime
0	5	1	X es impar

iteración	X	M	Imprime
0	4	0	X es par

iteración	X	M	Imprime
0	-3	1	X es impar

iteración	X	M	Imprime
0	-10	0	X es par

Ejemplo 5.2**Prueba de escritorio**

iteración	X	M	Imprime
0	0	-	-
1	0	-	-
2	13	1	X es impar

Ejercicio 5.3

Realizar un programa que multiplique dos números (equis y ye) sin utilizar el operador multiplicación (*), es decir, no se puede especificar en el algoritmo la operación:

equis * ye

La solución se tiene que proporcionar como un conjunto de pasos a seguir numerados. Al final se tiene que verificar el algoritmo (pruebas de escritorio).

Ejercicio 5.3

SOLUCIÓN:

- 1. Inicia el programa.**
- 2. Definir X, Y y Z.**
- 3. Declarar Z igual a 0.**
- 4. Solicitar un número entero al usuario.**
- 5. Almacer número en X.**
- 6. Solicitar un número entero al usuario.**
- 7. Almacer número en Y.**
- 8. MIENTRAS que X sea mayor a 0**
 - 9. Hacer Z igual a Z más Y**
 - 10. Hacer X igual a X menos 1**
 - 11. TERMINA_MIENTRAS**
 - 12. Imprimir el contenido de Z**
 - 13. Termina el programa.**

Ejercicio 5.3**Prueba de escritorio**

iteración	X	Y	Z
0	5	7	7
1	4	7	14
2	3	7	21
3	2	7	28
4	1	7	35
5	0	7	35

El algoritmo anterior cumple con el objetivo del problema planteado, empero se puede eficientar el tiempo de ejecución.

Una vez que ya se hayan recibido ambos valores (X y Y), se comprueba cuál es el menor y se le asigna a X. Debido a que X determina cuantas veces se va a realizar el ciclo, entre más pequeña sea X menos iteraciones se realizan. Esto se puede hacer debido a la propiedad de la multiplicación que dicta que “el orden de los factores no altera el producto”.

Ejercicio 5.3**SOLUCIÓN 2:**

1. Inicia el programa.
2. Definir X, Y, Z y TMP.
3. Declarar Z igual a 0.
4. Solicitar un número entero al usuario.
5. Almacer el número en X.
6. Solicitar un número entero al usuario.
7. Almacer el número en Y.
8. **Si Y < X**
 9. **X Hacer TMP igual X**
 10. **Hacer X igual a Y**
 11. **Hacer Y igual a TMP**
12. **TERMINA_SI**
13. MIENTRAS X sea mayor a 0
14. Hacer Z igual a Z más Y
15. Hacer X igual a X menos 1
16. **TERMINA_MIENTRAS**
17. Imprimir el contenido de Z
18. Termina el programa.

El análisis puede llegar a ser tan complicado o refinado como el analista desee o experiencia tenga.

Existe un algoritmo más eficiente para resolver una multiplicación en menos pasos.

Para entender el siguiente ejemplo es importante comprender las operaciones que realizan los operandos >> y <<:

N₁₀	N₂	Operando	Resultado
10	1010	>>	0101
7	111	<<	1110

Ejemplo 5.3**SOLUCIÓN MEJORADA:**

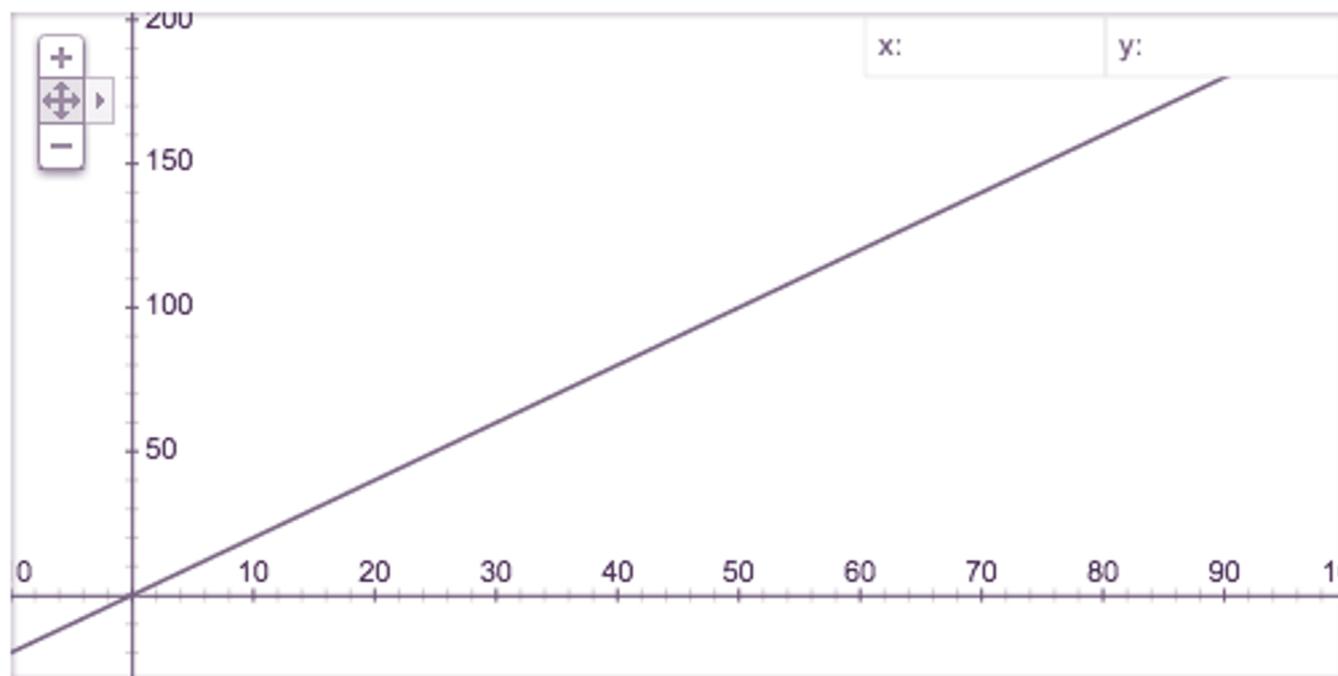
1. Inicia el programa.
2. Definir X, Y y Z.
3. Declarar Z igual a 0.
4. Solicitar un número entero al usuario.
5. Almacer número en X.
6. Solicitar un número entero al usuario.
7. Almacer número en Y.
8. MIENTRAS X sea mayor a 0
9. Si X modulo 2 es igual a 1
 Hacer Z igual a Z más Y
10. Hacer un corrimiento de bits a la izquierda de Y
11. Hacer un corrimiento de bits a la derecha de X
12. FIN_MIENTRAS
13. Imprimir el valor de Z
14. Termina el programa.

Ejemplo 5.3**Prueba de escritorio**

iteración	X	Y	Z
0	14	11	0
1	7	22	0
2	3	44	22
3	1	88	66
4	0	176	154

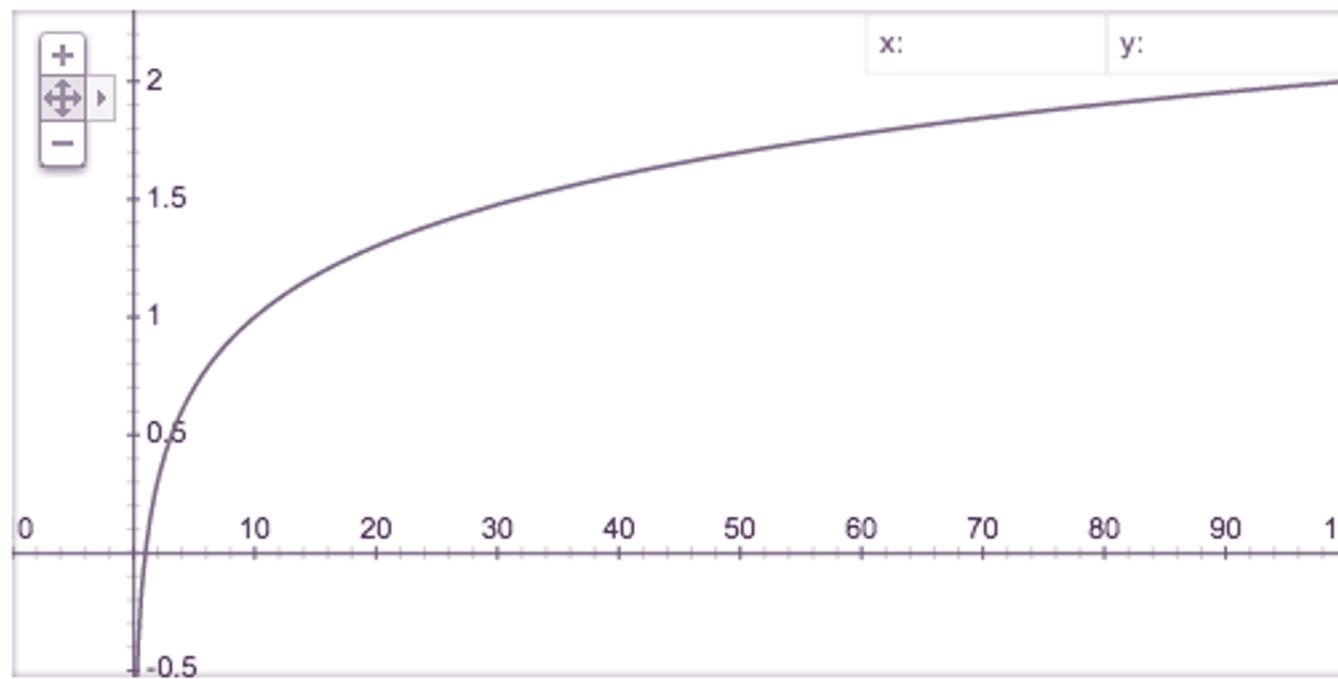
El tiempo que le toma al algoritmo del ejercicio 5.2 en realizar la multiplicación es lineal:

$$t = 2 * X$$



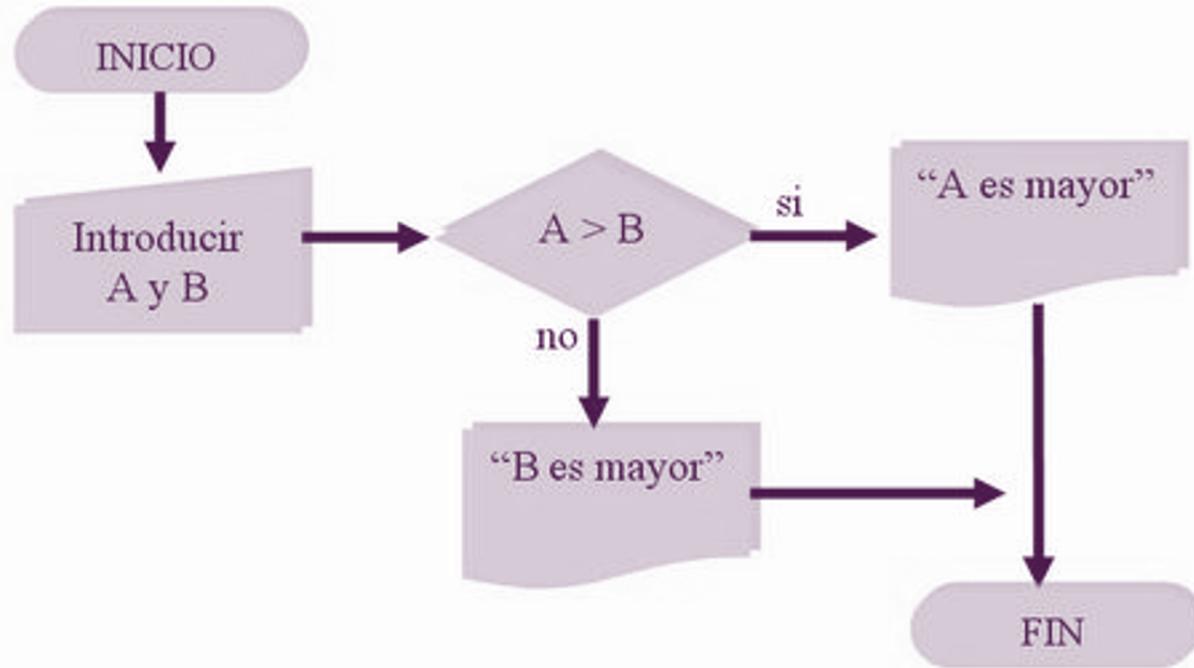
El tiempo que le toma al algoritmo del ejemplo 5.3 en realizar la multiplicación es logarítmico:

$$\lceil \log_2 X \rceil + 1$$



Por lo tanto, a pesar de que ambos algoritmos cumplen con la solución del problema planteado, el segundo es más eficiente.

Al segundo método se le conoce como algoritmo de los campesinos rusos (Russian peasant algorithm) para resolver una multiplicación.



5.3 Representación de los algoritmos (diagrama de flujo y pseudocódigo).

5.3 Representación de los algoritmos (diagrama de flujo y pseudocódigo)

Una vez que un problema dado ha sido analizado (se obtienen los datos de entrada y la salida deseada) y se ha diseñado un algoritmo que lo resuelva de manera eficiente (procesamiento de datos), se debe proceder a la etapa de codificación del algoritmo.

La codificación se refiere a la operación de escribir la solución del problema (algoritmo), en una serie de instrucciones detalladas, en un código que pueda ser reconocido por la computadora.

La serie de instrucciones se conoce como código fuente, el cual se escribe en un lenguaje de programación de medio o alto nivel.

Para que la solución de un problema (algoritmo) pueda ser codificada, se debe presentar en lo que se conoce como representación de un algoritmo.

Las representaciones de un algoritmo estructurado pueden ser dos:

- **Diagrama de flujo**
- **Pseudocódigo**

Conceptos fundamentales para diagramas de flujo y pseudocódigos

➤ Tipos de datos

Los tipos de datos que se manejan para programas estructurados son simples o compuestos.

Dentro de los tipos de datos simples se cuentan con los siguientes:

- **Enteros:** -10, 5, 100.
- **Reales:** 0.0001, 10.5.
- **Caracteres:** 'l', 'a', 'Z'.
- **Booleanos:** true o false.

Dentro de los tipos de datos estructurados se encuentran los siguientes:

- **Arreglos: unidimensionales o multidimensionales.**
- **Enumeraciones: varios tipos de dato constantes.**
- **Estructuras: tipos de datos compuestos por la unión de otros tipos de datos simples o compuestos: enteros, arreglos, estructuras, etc.**

➤ Identificador

Un identificador es el nombre con el que se va a almacenar en memoria un tipo de dato. Sigue las siguientes reglas:

- **Debe iniciar con una letra [a-z].**
- **Puede contener letras [a-z], números [0-9] y el carácter guión bajo (_).**

➤ Constante

Un tipo de dato constante no cambia su valor durante la ejecución del programa. Las constantes pueden ser de cualquier tipo de dato simple (carácter, entero o real) e incluso arreglos.

Por definición, los valores dentro de un tipo de dato enumerador son constantes.

➤ Variable

Un tipo de dato variable puede cambiar su valor durante la ejecución del programa. Las variables pueden ser de cualquier tipo de dato simple (carácter, entero o real) e incluso arreglos.

➤ Asignación

Para determinar el valor de una variable se utiliza el símbolo de asignación. La asignación es una operación destructiva debido a que el valor que tiene una variable es borrado al momento de realizar una asignación, adquiriendo el nuevo valor dado.

En un diagrama de flujo el símbolo de asignación está representado por \leftarrow .

var \leftarrow expresión o valor

En pseudocódigo el símbolo de asignación está representado por $:=$

var $:=$ expresión o valor

➤ Operaciones aritméticas

Las siguiente tabla muestra las operaciones aritméticas básicas así como su nomenclatura para la representación de algoritmos.

Operador	Operación	Uso	Resultado
+	Suma	$125.78+62.5$	188.28
-	Resta	$65.3-32.33$	32.97
*	Multiplicación	$8.27*7$	57.75
/	División	$15/4$	3.75
**	Potencia	$4^{**}3$	64

Operador r	Operación	Uso	Resultado
mod	Módulo (residuo)	15 mod 2	1
div	División entera	17 div 3	5
>>	Corrimiento de bits a la derecha	8 >> 2	2
<<	Corrimiento de bits a la izquierda	8 << 1	16

Al evaluar expresiones con operaciones aritméticas se debe tener en cuenta la jerarquía de los mismos, es decir, el orden con el que se van a ejecutar:

Operador	Jerarquía	Operación
**	Mayor	Potencia
*, /, mod, div		Multiplicación, división, módulo, división entera.
+,-	↓	Suma, resta.
<<, >>	Menor	Corrimiento de bits a la izquierda, a la derecha.

Ejemplo 5.4

1) $7 + 5 - 6 :=$
 $12 - 6 :=$
 6

2) $9 + 8 * 7 - 36 / 5 \leftarrow$
 $9 + 56 - 36 / 5 \leftarrow$
 $9 + 56 - 7.2 \leftarrow$
 $65 - 7.2 \leftarrow$
 57.8

3) $7 * 5 ** 3 / 4 \text{ div } 3 :=$
 $7 * 125 / 4 \text{ div } 3 :=$
 $875 / 4 \text{ div } 3 :=$
 $218.75 \text{ div } 3 :=$
 72

➤ Expresiones lógicas

Las expresiones lógicas están constituidas por números, caracteres, constantes o variables que están relacionados entre sí por operadores lógicos. Una expresión lógica puede tomar únicamente los valores verdadero o falso.

Los operadores relacionales permiten comparar elementos numéricos, alfanuméricos, constantes o variables.

Operador	Operación	Uso	Resultado
=	Igual que	'h' = 'H'	Falso
< >	Diferente a	'a' <> 'b'	Verdadero
<	Menor que	7 < 15	Verdadero
>	Mayor que	11 > 22	Falso
< =	Menor o igual que	15 <= 22	Verdadero
> =	Mayor o igual que	20 >= 35	Falso

NOTA. Si se utilizan operadores de relación con operadores lógicos, falso es menor que verdadero.

Ejemplo 5.5

1) **A := 5**
B := 11
(A ** 2) > (B * 2)
 25 > 22
Verdadero

2) **X ← 6**
B ← 7.8
(X * 5 + B ** 3 / 4) <= (X ** 3 div B)
(X * 5 + 474.552 / 4) <= (X ** 3 div B)
(30 + 474.552 / 4) <= (X ** 3 div B)
(30 + 118.638) <= (X ** 3 div B)
(148.638) <= (X ** 3 div B)
(148.638) <= (216 div B)
(148.638) <= (27)
Falso

Los operadores lógicos permiten formular condiciones complejas a partir de condiciones simples.

Operador	Jerarquía	Expresión lógica
NO	Mayor	NO p
Y	Intermedia	p Y q
O	Menor	p O q

La jerarquía completa de los operadores aritméticos, relacionales y lógicos es la siguiente:

Operador	Jerarquía
()	Mayor
**	
*, /, mod, div	
+, -	
<<, >>	
NO	
Y	
O	Menor

Ejemplo 5.6

i : ENTERO
j : REAL
k: BOOLEANO

i ← 0
i ← i + 1

j ← 5 ** 2 div 3

k ← (8 > 5) Y (15 < 2 ** 3)

Pseudocódigo

Un pseudocódigo es la representación escrita de un algoritmo, es decir, muestra en forma de texto los pasos a seguir para solucionar un problema.

Los pseudocódigo poseen una sintaxis propia para poder generar la solución de un problema.

Sintaxis de pseudocódigo

- **Alcance del programa:** Todo pseudocódigo está limitado por las etiquetas de **INICIO** y **FIN**. Dentro de estas etiquetas se deben escribir todas las instrucciones del programa.
- **Lectura / escritura:** Para indicar lectura de datos se utiliza la etiqueta **LEER**. Para indicar escritura de datos se utiliza la etiqueta **ESCRIBIR**.

- Declaración de variables: la declaración de variables la definen un identificador, seguido de dos puntos, seguido del tipo de dato:

contador: ENTERO

producto: REAL

continuar: BOOLEANO

NUM_MAX: REAL, CONSTANTE

- **Operadores aritméticos:** Se tiene la posibilidad de utilizar operadores aritméticos y lógicos:

Operadores aritméticos : suma (+), resta (-), multiplicación (*), división (/), división entera (div), módulo (mod), exponenciación (^), asignación (:=).

Operadores lógicos : igualdad (=), y-lógica (&), o-lógica (|), negación (!), relaciones de orden (<, >, <=, >=).

Ejemplo 5.7

Realizar la suma de dos números reales. Los números deben ser proporcionados por el usuario.

INICIO

uno, dos : REAL

ESCRIBIR “Programa que suma dos números reales”

ESCRIBIR “Ingrese el primer número”

LERR uno

ESCRIBIR “Ingrese el segundo número”

LEER dos

res : REAL

res := uno + dos

ESCRIBIR “El resultado es ” res

FIN

Ejemplo 5.8

Obtener el valor de la función $f(x)$ para una ‘x’ dada por el usuario. $f(x) = x^2 + 2$

INICIO

x : REAL

ESCRIBIR “Programa que resuelve la función”

ESCRIBIR “ $f(x) = x^2 + 2$ ”

ESCRIBIR “Ingrese el valor de x”

LERR x

y : REAL

y := x2 + 2**

ESCRIBIR “El valor de la función en ” x

ESCRIBIR “es ” y

FIN

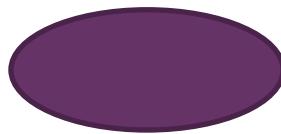
Diagramas de flujo

Un diagrama de flujo es una representación gráfica de un algoritmo, es decir, muestra gráficamente los pasos a seguir para solucionar un problema.

La correcta construcción de estos diagramas es fundamental para la etapa de codificación, ya que, a partir del diagrama de flujo se escribe un programa en algún lenguaje de programación.

Los diagramas de flujo poseen símbolos que permiten estructurar la solución de un problema de manera gráfica. Por tanto es fundamental conocer los elementos que conforman este lenguaje gráfico.

Elementos de los diagramas de flujo



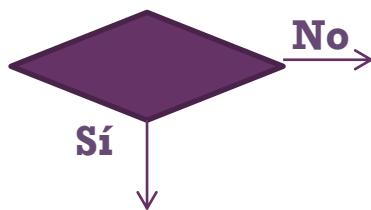
Representa el inicio o el fin del diagrama de flujo.



Datos de entrada. Expresa lectura.



Proceso. En su interior se expresan asignaciones u operaciones.



Decisión. Valida una condición y ejecuta uno u otro camino.



Escritura. Impresión en pantalla.



Dirección de flujo del diagrama.



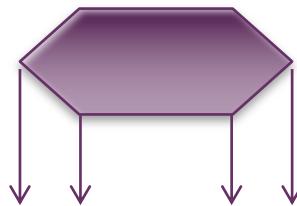
Conexión dentro de la misma página.



Conexión entre diferentes páginas.

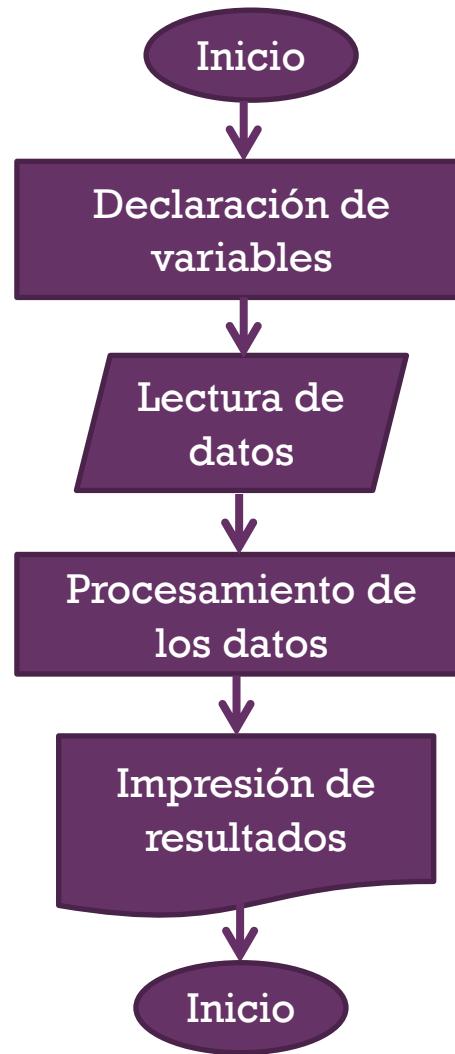


Módulo de un problema. Llamadas a otros módulos o funciones



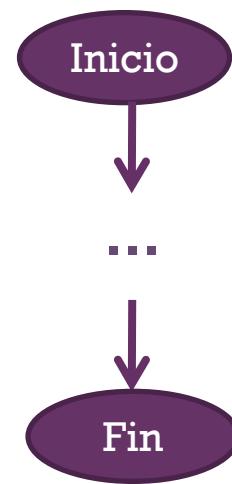
Decisión múltiple.
Almacena un selector que determina la rama por la que sigue el flujo.

Construcción de un diagrama de flujo

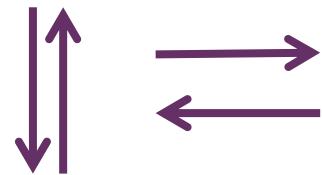


Los diagramas de flujo poseen reglas que hay que seguir al momento de desarrollar la solución de un problema.

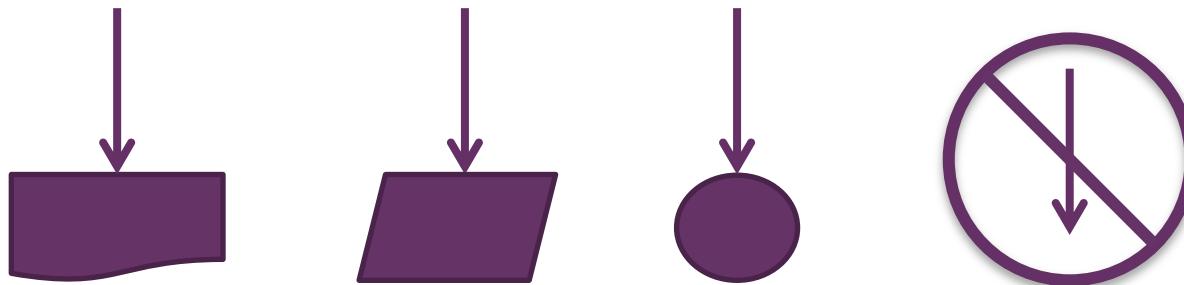
1. Todo diagrama de flujo debe tener un inicio y un fin.



2. Las líneas utilizadas para indicar la dirección del flujo del diagrama deben ser rectas, verticales u horizontales, exclusivamente.



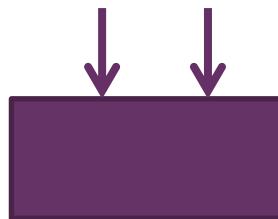
3. Todas las líneas utilizadas para indicar la dirección del flujo del diagrama deben estar conectadas a un símbolo.



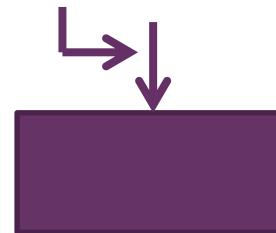
- 4. El diagrama debe ser construido de arriba hacia abajo (top-down) de izquierda a derecha (left to right).**
- 5. La notación utilizada en el diagrama de flujo debe ser independiente del lenguaje de programación en el que se va a codificar la solución.**
- 6. Se recomienda poner comentarios que expresen o ayuden a entender un bloque de símbolos.**
- 7. Si la extensión de un diagrama de flujo ocupa más de una página, es necesario utilizar y numerar los símbolos adecuados.**

8. A cada símbolo solo le puede llegar una línea de dirección de flujo.

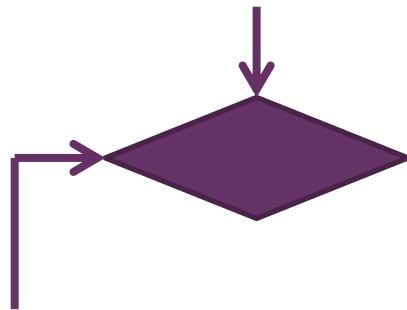
Inválido



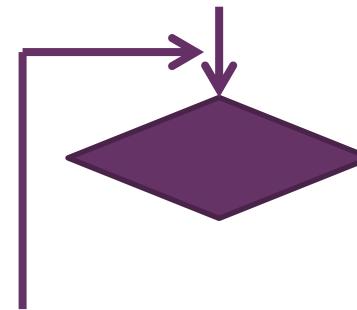
Válido



Inválido

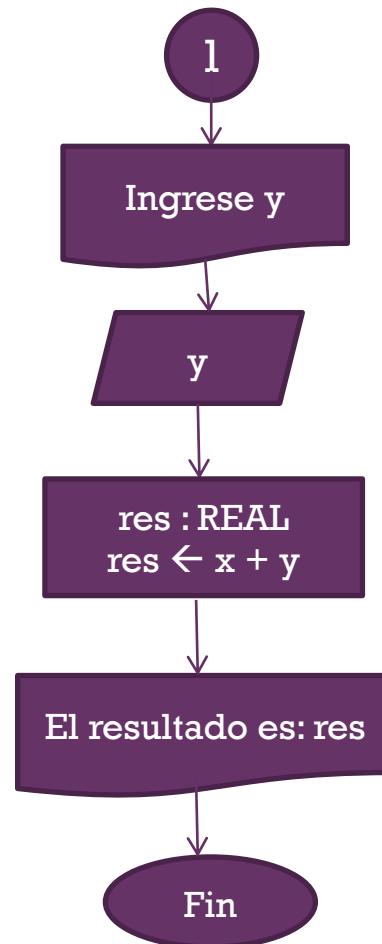


Válido



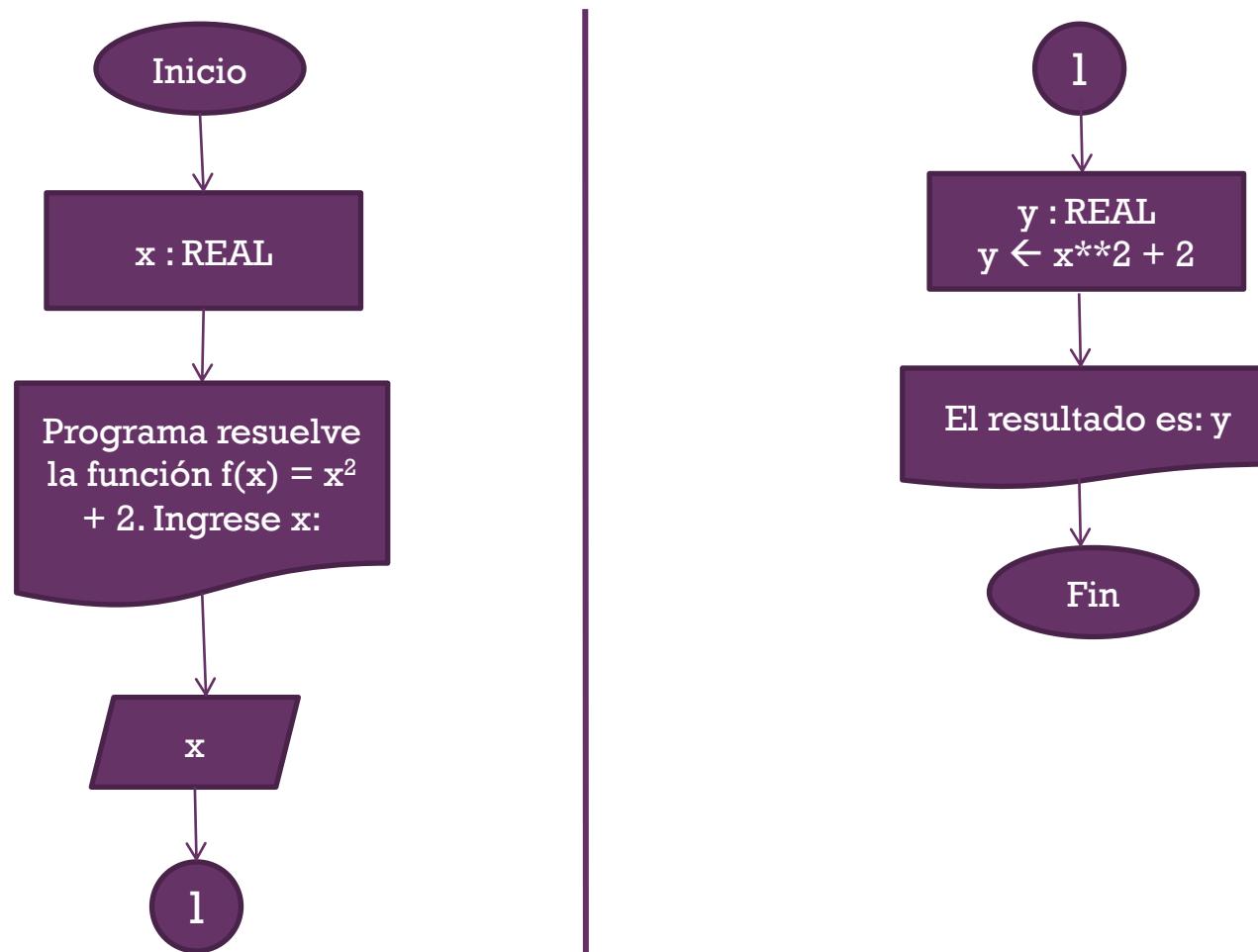
Ejemplo 5.10

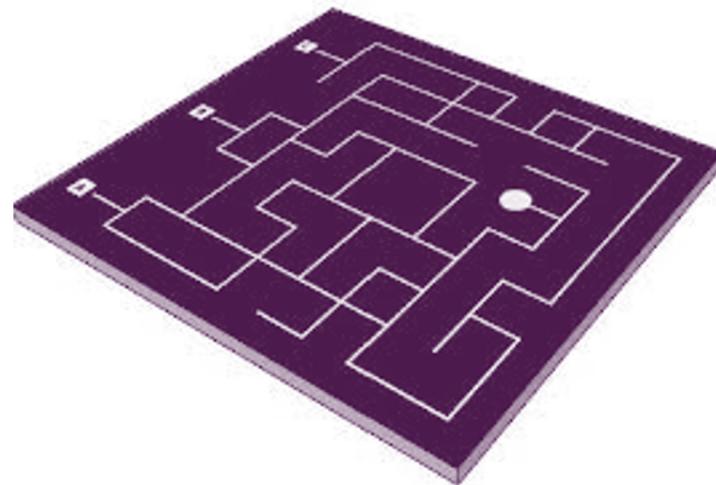
Realizar la suma de dos números reales. Los números deben ser proporcionados por el usuario.



Ejemplo 5.11

Obtener el valor de la función $f(x)$ para una ‘x’ dada por el usuario. $f(x) = x^2 + 2$





5.4 Estructuras básicas (secuencia, condicional e iteración)

5.4 Estructuras básicas (secuencia, condicional e iteración)

Las estructuras de control de flujo permiten la ejecución condicional y la repetición de un conjunto de instrucciones.

Existen 3 estructuras de control: secuencial, condicional y repetitivas o iterativas.

Pseudocódigo

Las estructuras de control secuenciales son las sentencias o declaraciones que se realizan una a continuación de otra en el orden en el que están escritas.

x: REAL

x := 5.8

x := x * 2

Las estructuras de control condicionales permiten evaluar una expresión lógica (condición que puede ser verdadera o falsa) y dependiendo del resultado se realiza uno u otro flujo de instrucciones.

Las estructuras de control repetitivas o iterativas (Bucles o Ciclos), permiten repetir un conjunto de instrucciones mientras se cumpla una condición (expresión lógica).

Estructuras condicionales

La estructura de control de flujo más simple es la estructura condicional SI, su sintaxis es la siguiente:

```
SI cond_booleana ENTONCES
    // Instrucciones
FIN_SI
```

Se valúa la expresión lógica y si se cumple (si la condición es verdadera) se ejecutan las instrucciones del bloque. Si no se cumple la condición se sigue con el flujo normal del programa.

La estructura de control de flujo condicional SI completa es el SI-DE_LO_CONTRARIO:

```
SI cond_booleana ENTONCES
    // Instrucciones
FIN_SI
DE_LO CONTRARIO
    // Instrucciones
FIN_DLC
```

Se valúa la expresión lógica y si se cumple (si la condición es verdadera) se ejecutan las instrucciones del bloque SI. Si no se cumple la condición se ejecutan las instrucciones del bloque DE_LO CONTRARIO. Al final el programa sigue su flujo normal.

Ejemplo 5.12**INICIO****a, b, c: ENTERO****a := 5****b := 6****c := 0****SI a > b ENTONCES****c = a - b****FIN_SI****DE_LO CONTRARIO****c = b - a****FIN_DE_LO CONTRARIO****ESCRIBIR c****FIN**

La estructura de control de flujo selección de casos valida el valor de la variable que está entre paréntesis y comprueba si es igual a las variables que están definidas en cada caso. Si la variable no está definida en algún caso se va a la instrucción por defecto:

SELECCIONAR (exp_entera)

CASO 1-> // Instrucciones

CASO 2-> // Instrucciones

CASO 3-> // Instrucciones

DEFECTO -> // Instrucciones

FIN_SELECCIONAR

Ejemplo 5.13

INICIO

```
var: ENTERO  
ESCRIBIR Seleccione la opción que desea  
ESCRIBIR 1) Iniciar sesión  
ESCRIBIR 2) Registrarse  
ESCRIBIR 3) Salir  
LEER var
```

SELECCIONAR (var)

CASO 1->

ESCRIBIR Bienvenido...

CASO 2->

ESCRIBIR De nombre usuario...

CASO 3->

ESCRIBIR Vuela pronto.

DEFECTO ->

ESCRIBIR Opción no válida.

FIN_SELECCIONAR

FIN

Ejemplo 5.13 b

INICIO

var: CARÁCTER

ESCRIBIR Seleccione la opción que desea

ESCRIBIR c) Iniciar sesión

ESCRIBIR b) Registrarse

ESCRIBIR a) Salir

LEER var

SELECCIONAR (var)

CASO 'a'->

ESCRIBIR Bienvenido...

CASO 'b'->

ESCRIBIR De nombre usuario...

CASO 'c'->

ESCRIBIR Vuela pronto.

DEFECTO ->

ESCRIBIR Opción no válida.

FIN_SELECCIONAR

FIN

Estructuras iterativas

Las estructuras de control de flujo cíclicas permiten ejecutar una serie de instrucciones hasta que se cumpla la expresión lógica. Existen dos tipos de expresiones cíclicas MIENTRAS y REPETIR-MIENTRAS.

La estructura MIENTRAS primero valida la expresión lógica y si ésta es verdadera procede a ejecutar el bloque de instrucciones de la estructura, de lo contrario rompe el ciclo y continúa el flujo normal del programa.

```
MIENTRAS exp_booleana HACER
    // Bloque de
    // instrucciones
    // a ejecutar
FIN_MIENTRAS
```

El final de la estructura lo determina la etiqueta **FIN_MIENTRAS**.

La estructura REALIZAR - MIENTRAS primero ejecuta las instrucciones descritas en la estructura y al final valida la expresión lógica.

```
REALIZAR
    // Bloque de
    // instrucciones
    // a ejecutar
MIENTRAS exp_booleana
```

Si la expresión lógica se cumple vuelve a ejecutar las instrucciones de la estructura, de lo contrario rompe el ciclo y sigue el flujo del programa.

Ejemplo 5.14**INICIO**

```
contra: CADENA_CARACTERES
negar: BOOLEANO
contador: ENTERO
contra := ""
seguir := verdadero
contador := 0
MIENTRAS (seguir) Y (contador < 3) HACER
    ESCRIBIR "Ingrese contraseña."
    LEER contra
    SI contra = "root" ENTONCES
        seguir:= false
    FIN_SI
    DE_LO CONTRARIO
        contador := contador + 1
    FIN_DE_LO CONTRARIO
FIN_MIENTRAS
FIN
```

Ejemplo 5.15

INICIO

usr, pas: CADENA_CARACTERES

REALIZAR

ESCRIBIR Escriba nombre usuario

LEER usr

ESCRIBIR Escriba contraseña

LEER pas

MIENTRAS (usr = “admin”) O (pas = “istrador”)

ESCRIBIR ;Bienvenido administrador!

FIN

Funciones

Cuando la solución de un problema es muy compleja se suele ocupar el diseño descendente (divide y vencerás). Este diseño implica la división de un problema en varios subprocessos más sencillos que juntos forman la solución completa.

Cada subprocesso realiza, de manera independiente, una serie de pasos para completar su tarea. Estos subprocessos se suelen llamar funciones y permiten dividir la solución del problema para que sea más fácil entenderlo y depurarlo.

Una función está constituida por un identificador de función (nombre), de cero a n parámetros de entrada y un valor de retorno:

```
FUNC nombre (var:TipoDato,..., var:TipoDato) DEV TipoDato  
    // Bloque de  
    // instrucciones  
    // a ejecutar  
FIN_FUNC
```

Las funciones son subprocessos independientes.

Ejemplo 5.16**INICIO****FUNC principal (vacío) DEV vacío****a, b, c: ENTERO****a := 5;****b := 24****c := suma(a, b)****ESCRIBIR c****FIN_FUNC****FIN****INICIO****** Función que suma dos números enteros****FUNC suma (uno:ENTERO, dos:ENTERO) DEV ENTERO****tres: ENTERO****tres := uno + dos****DEV tres****FIN_FUNC****FIN**

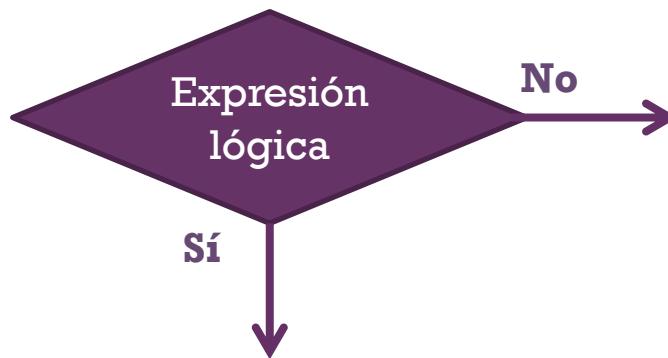
Diagramas de flujo

Las estructuras de control secuenciales son las sentencias o declaraciones que se realizan una a continuación de otra en el orden en el que están escritas.

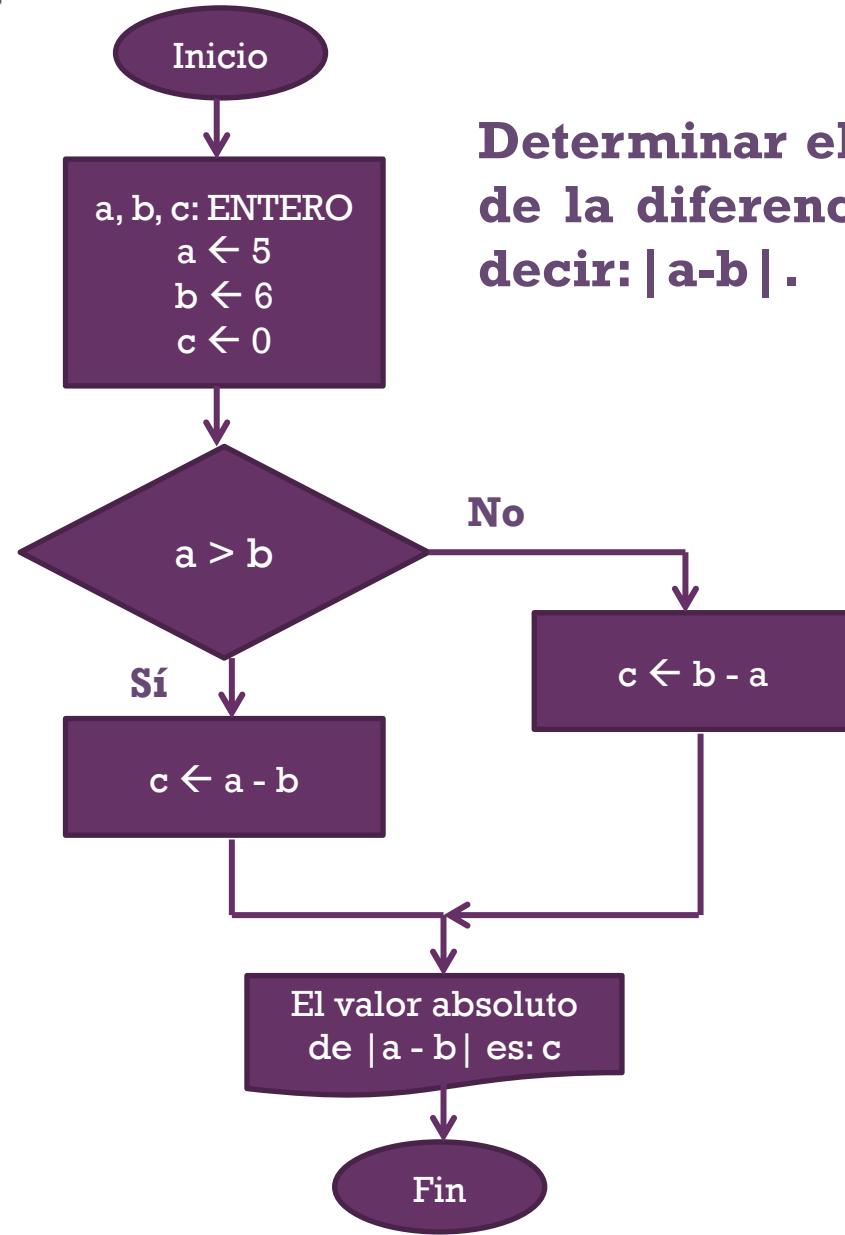
```
x: REAL  
x := 5.8  
x := x * 2
```

Estructuras condicionales

La estructura de control de flujo más simple es la estructura condicional SI, su grafo es el siguiente:

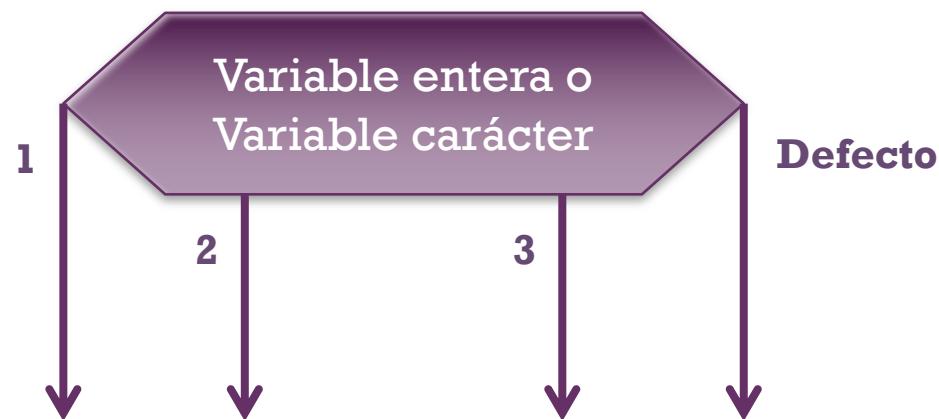


Este mismo elemento puede simbolizar la estructura SI - DE_LO CONTRARIO.

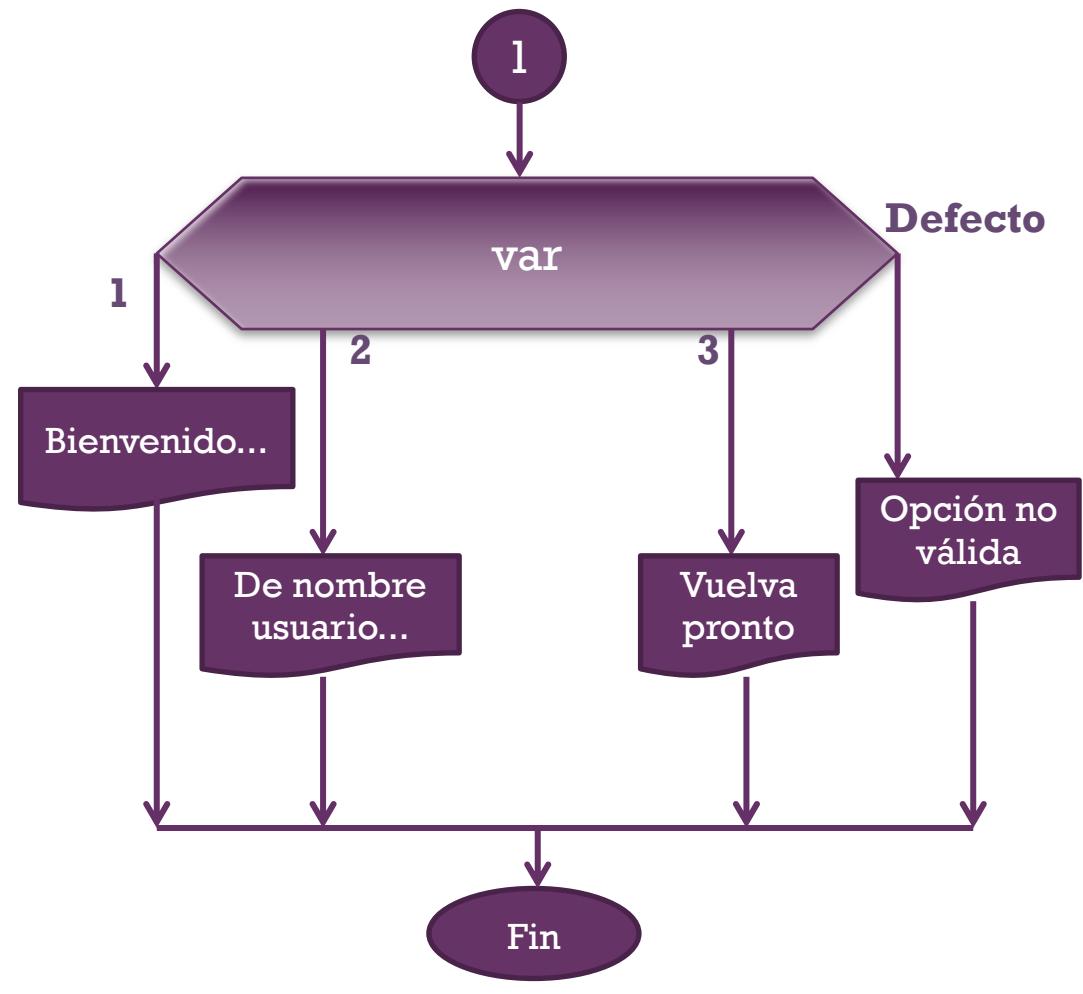
Ejemplo 5.17

Determinar el valor absoluto de la diferencia de a y b , es decir: $|a-b|$.

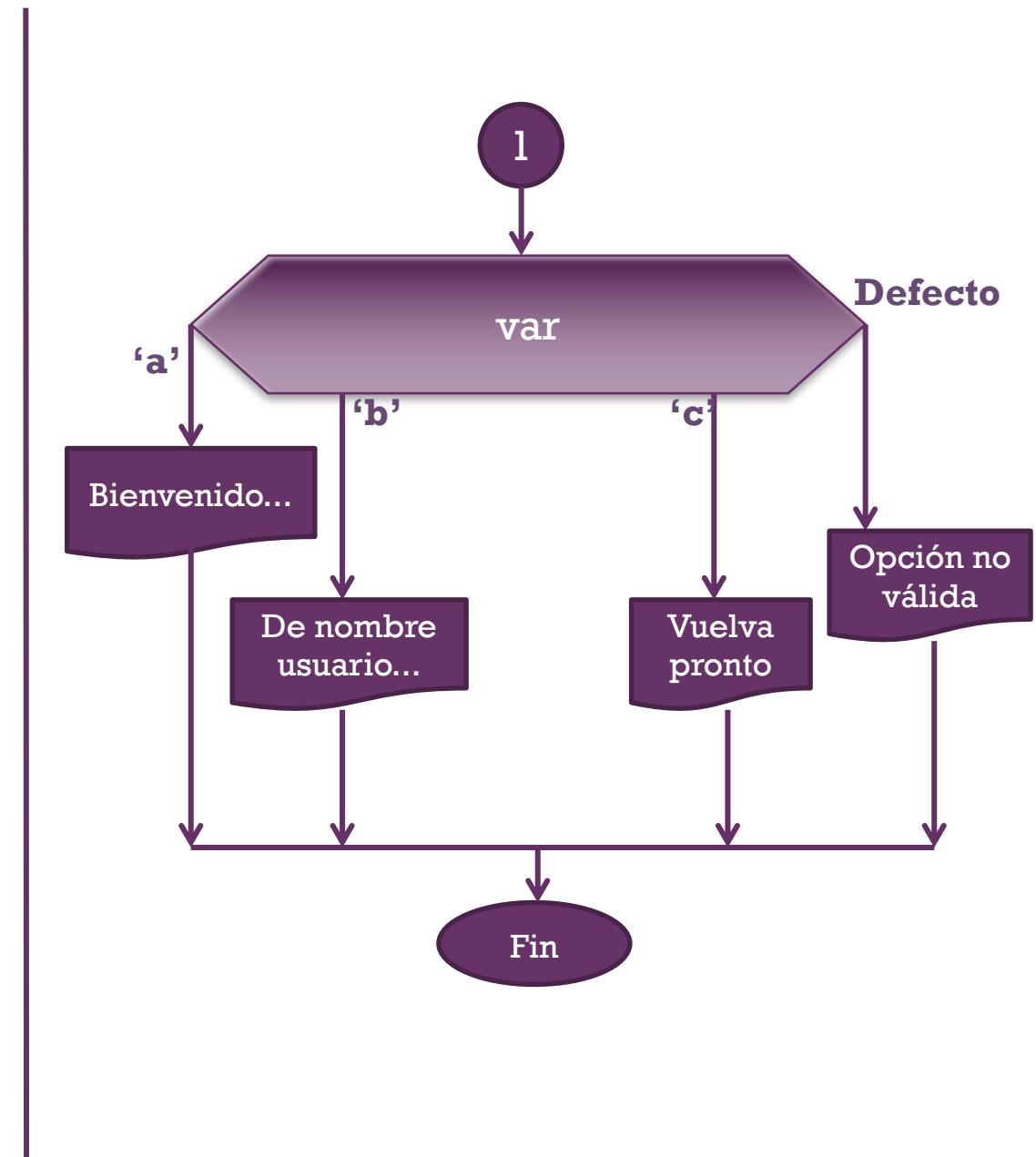
La estructura de control de flujo selección de casos valida el valor de la variable que está dentro de la figura y comprueba si es igual a las variables que están definidas en cada uno de las líneas. Si la variable no está definida en algún caso se va a la línea por defecto:



Ejemplo 5.18

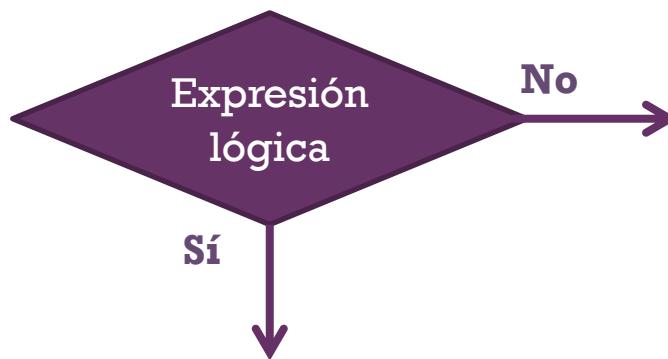


Ejemplo 5.18 b



Estructuras iterativas

Las estructuras de control de flujo cíclicas MIENTRAS y REPETIR- MIENTRAS se representan con el mismo grafo, la única diferencia es la colocación del mismo. El elemento es el siguiente:

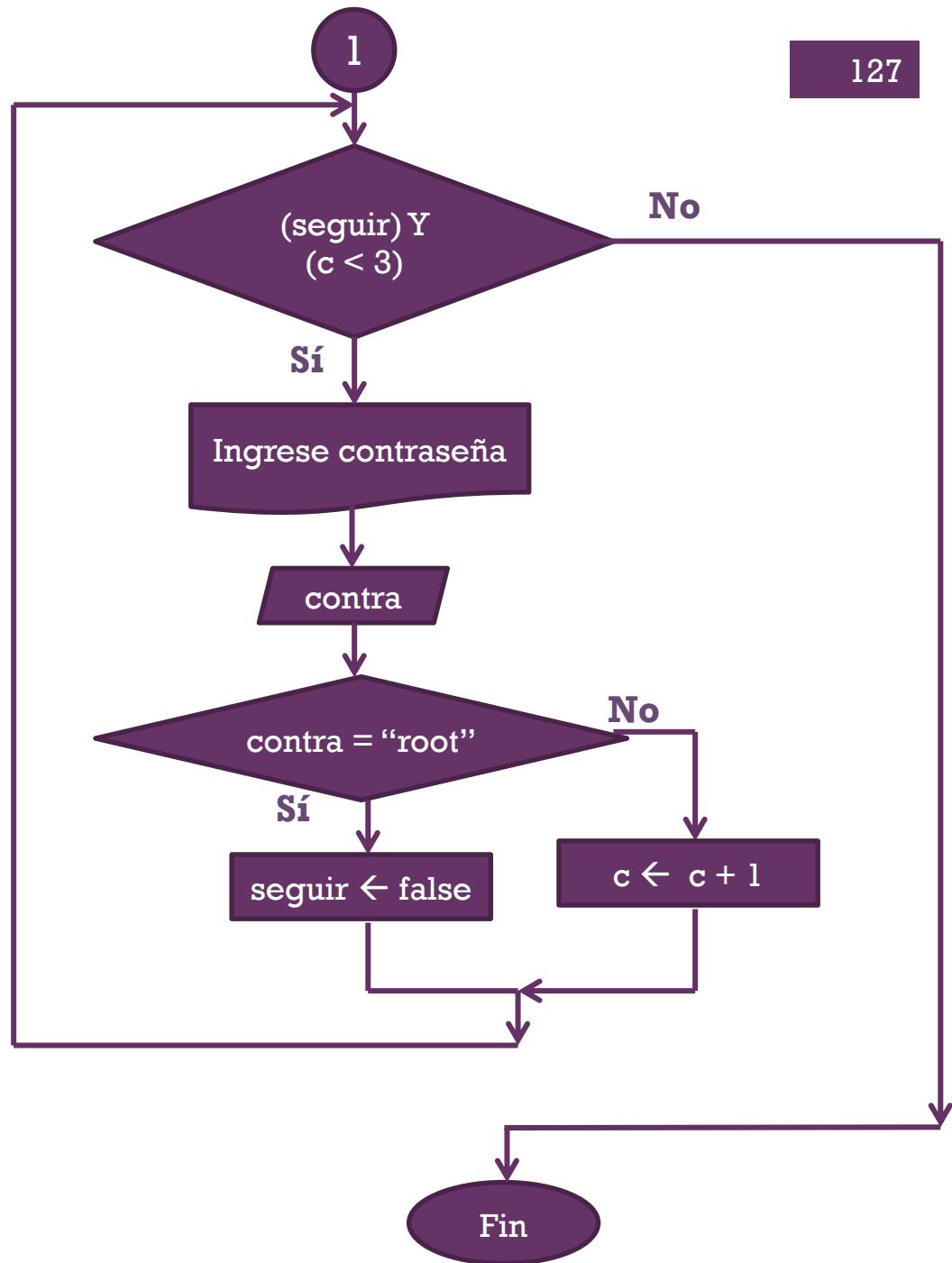


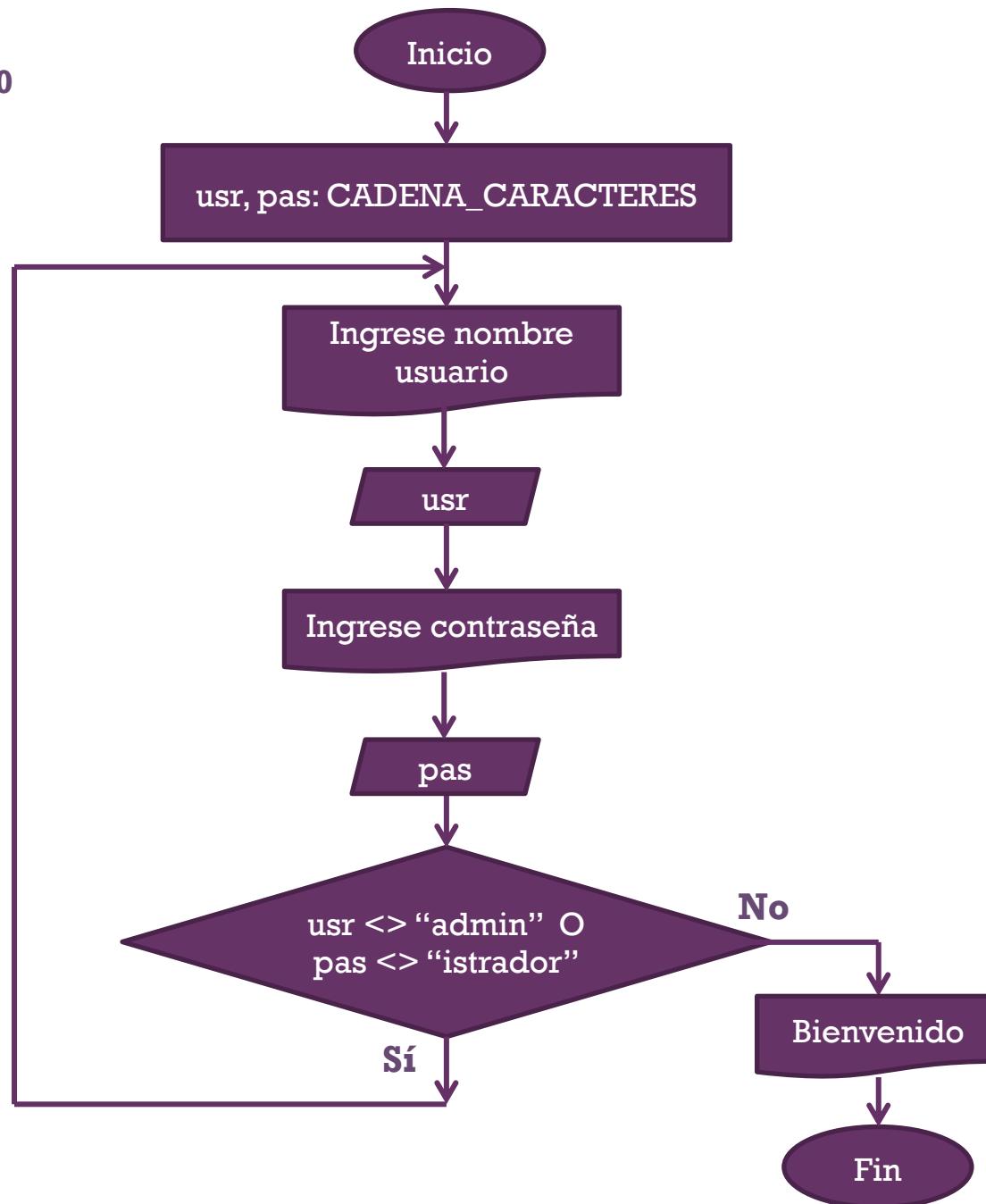
Ejemplo 5.19



```

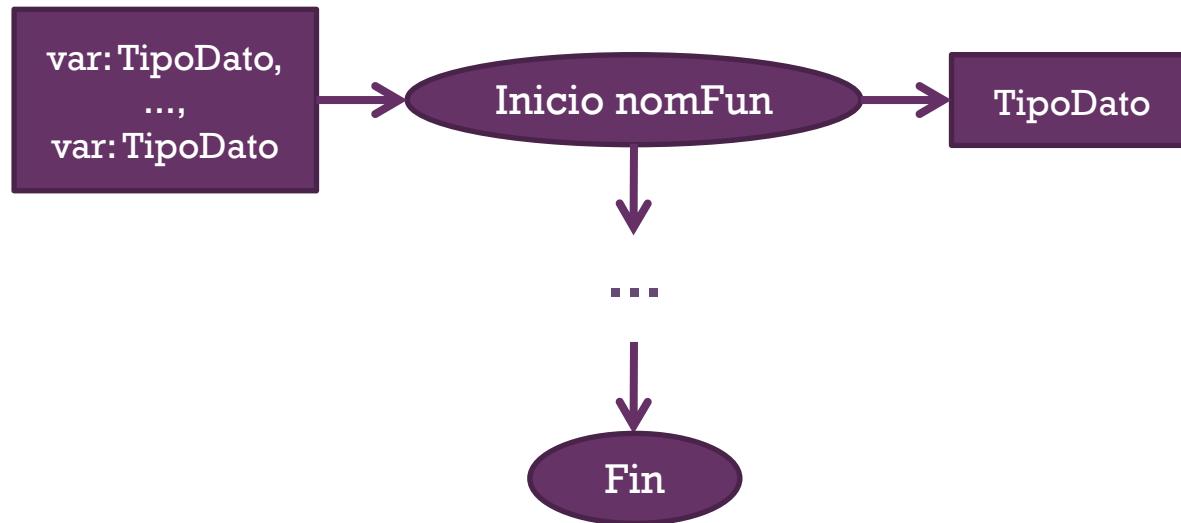
contra: CADENA_CARACTERES
seguir: BOOLEANO
c: ENTERO
contra := ""
seguir := verdadero
c := 0
  
```

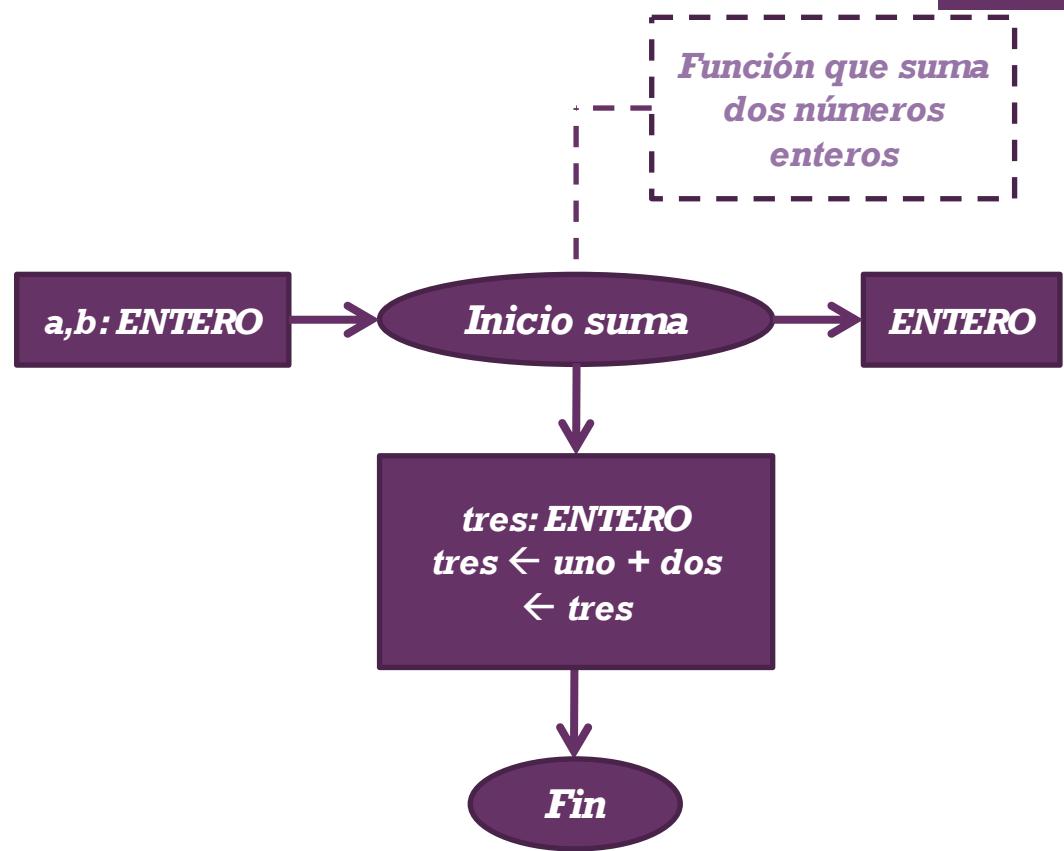
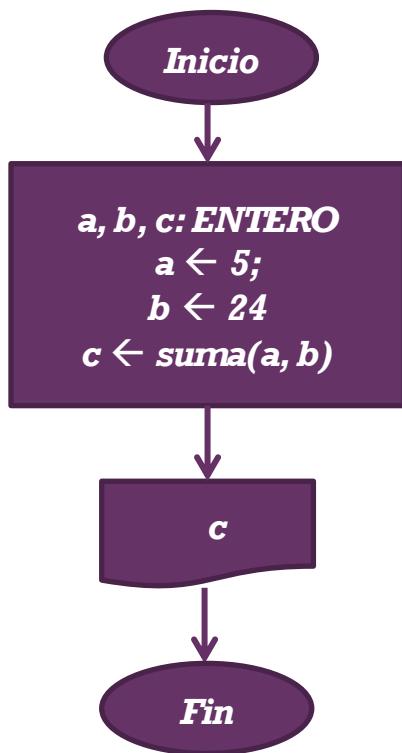


Ejemplo 5.20

Funciones

En los diagramas de flujo también es posible dibujar funciones.



Ejemplo 5.21

Arreglos

Se denomina arreglo a un conjunto ordenados de datos del mismo tipo y con un tamaño fijo.

Por ejemplo, si se tiene un grupo con 5 alumnos, se pueden manejar sus calificaciones finales en un arreglo:

Alum 0	Alum 1	Alum 2	Alum 3	Alum 4
10	8	5	8	7

La sintaxis para definir un arreglo en un pseudocódigo es la siguiente:

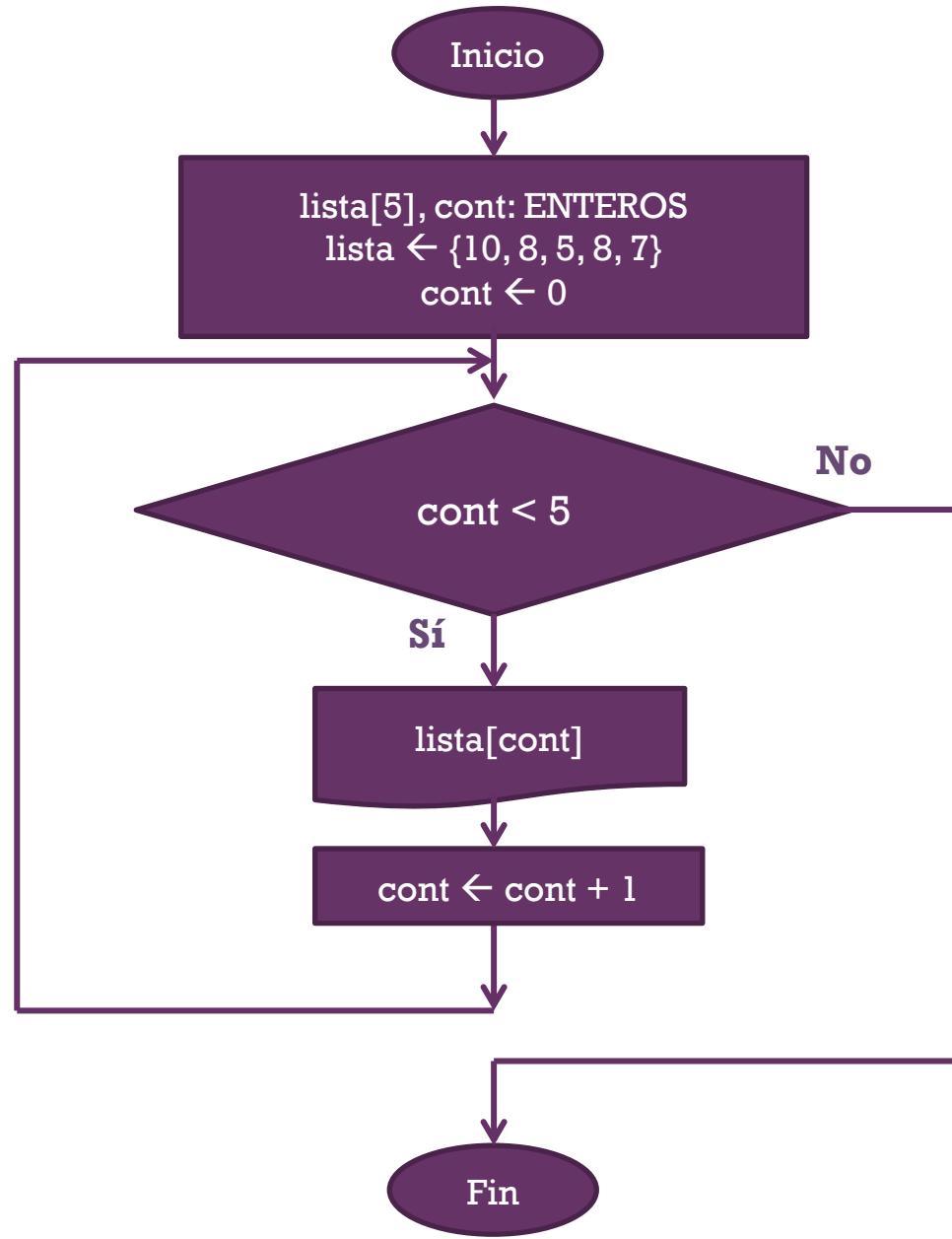
lista[tamaño] : ENTERO

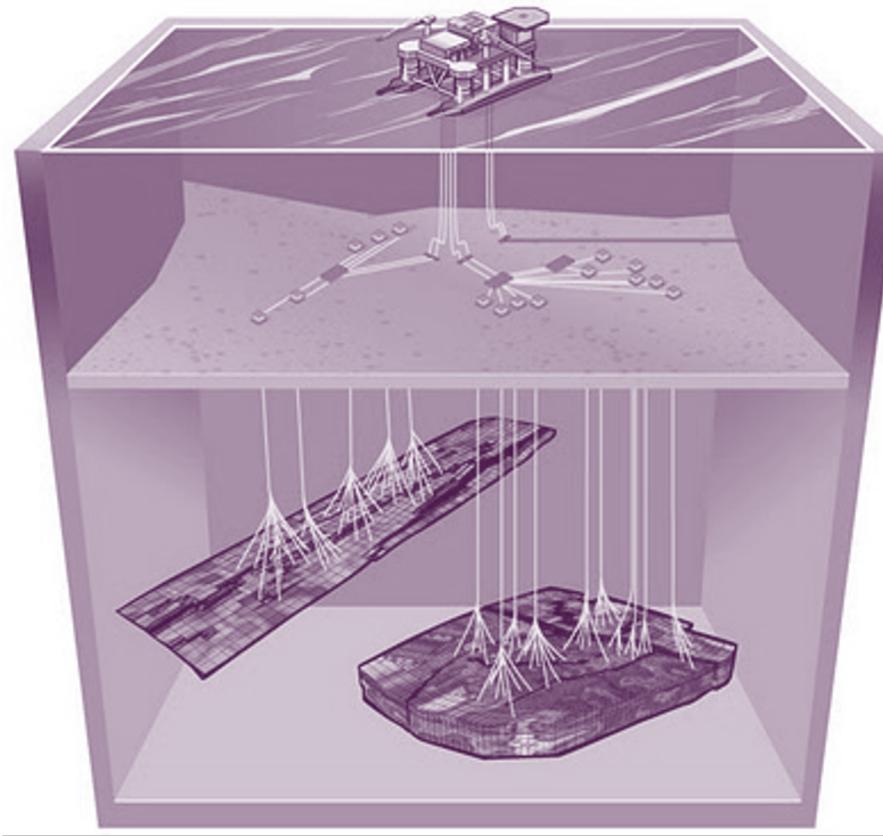
La sintaxis para definir un arreglo dentro de un grafo de proceso en un diagrama de flujo es la siguiente:

lista[tamaño] : REAL

Donde lista se refiere al nombre del arreglo, tamaño es un número entero y define el número máximo de elementos que puede contener el arreglo.

Ejemplo 5.22**INICIO****lista[5], cont: ENTERO****cont := 0****lista := {10, 8, 5, 8, 7}****MIENTRAS cont < 5 HACER****ESCRIBIR lista[cont]****cont := cont + 1****FIN_MIENTRAS****FIN**

Ejemplo 5.23



5.5 Resolución de problemas básicos de ingeniería

5.5 Resolución de problemas básicos de ingeniería

En la actualidad, cualquier problema que pueda ser computable puede ser sistematizado.

La sistematización lleva implícita el análisis del proceso y el desarrollo de una solución, los pilares de la definición de algoritmo.

Ejercicio 5.4

Realizar el diagrama de flujo o pseudocódigo por equipo según corresponda del problema dado.

Es necesario realizar el análisis (datos de entrada y datos de salida) así como el diseño de la solución plasmado en un algoritmo.

Todos los problemas implican la búsqueda de usuarios en la base de datos de una RRSS, realizar el algoritmo de dicha actividad como una función externa.

Ejercicio 5.4

Equip o	Representación	Problema
1	Diagrama de flujo	Inicio de sesión en una RRSS.
2	Pseudocódigo	
3	Diagrama de flujo	Registro de usuario en una RRSS.
4	Pseudocódigo	
5	Diagrama de flujo	Envío de un mensaje en una RRSS.
6	Pseudocódigo	
7	Diagrama de flujo	Aregar amigo en una RRSS.
8	Pseudocódigo	

5. Fundamentos de algoritmos

Objetivo: Explicar la importancia de llevar un método formal para resolver problemas en la computadora; así mismo aplicará dicho método en la resolución de problemas matemáticos sencillos.

**5.1 La Computabilidad y Concepto de algoritmo:
Máquina de Turing**

5.2 Elementos de los algoritmos y tipos de datos

5.3 Representación de los algoritmos (diagrama de flujo y pseudocódigo)

5.4 Estructuras básicas (secuencia, condicional e iteración)

5.5 Resolución de problemas básicos de ingeniería