
# UNIVERSIDAD TECNOLÓGICA DE PANAMÁ

## SEDE VICTOR LEVI SASSO


## ESTRUCTURA Y REPRESENTACIÓN DE DATOS

## INCLUYE PRUEBAS SUMATIVAS Y PRESENTACIONES DEL CONTENIDO

**ELABORADO POR:**

**DR. CARLOS A. ROVETTO**

**JULIO 2021**

# Contenido

# Índice de figuras

# Introducción

El mejoramiento en los procesos de información depende de una buena disposición de los datos y el manejo adecuado del espacio de memoria, lo que se consigue con un buen diseño y utilización de las estructuras de datos.

Las estructuras y representación de datos constituyen herramientas muy útiles para la evaluación de soluciones optimas de problemas y necesidades en el complejo mundo de la informática. Estas son la base para el diseño de aplicaciones importantes en distintas áreas de sistemas como lo son Bases de datos, Sistemas operativos, Compiladores, Redes, etc.

Esta asignatura consta de dos módulos. En el módulo I, denominado estructura de datos fundamentales y lineales, se desarrollarán temas relacionados con las estructuras de datos primitivas y simples y las estructuras de datos lineales y recursividad. En el primer tema se explican las distintas clasificaciones de las estructuras de datos, así como también los tipos de datos primitivas y simples. En el segundo tema relacionado con las estructuras de datos lineales y recursividad se explican las estructuras lineales tales como pila y colas, además de la recursividad.

En el módulo II se desarrollarán temas relacionados con la estructura de datos dinámicas lineales tales como las listas enlazadas. De este tema se explican las listas simples, doblemente enlazadas y las listas enlazadas circulares. Además de explicar las estructuras de datos dinámicas no lineales, las cuales incluyen los árboles y los grafos. En la sección de los árboles veremos su representación y sus formas de recorridos. En el tópico de grafos se definirá qué es un grafo, se explicarán los tipos de grafos, su representación y sus recorridos.

# Capítulo I: Estructura de datos fundamentales y lineales

Existen dos conceptos que son importantes conocer antes de iniciar este capítulo, estos son la definición de datos y de estructura de datos. El dominio de estas definiciones, les permitirán comprender la forma en la cual se desarrollarán los problemas relacionados con los mismos.

a) **Datos:** Es la mínima unidad de información significativa para alguien. Es la materia prima para la obtención de la información.
b) **Estructura de datos:** Es una clase de datos que se puede caracterizar por su organización y operaciones definidas sobre de ella. Algunas veces a estas estructuras se les llama tipos de datos.

Las estructuras de datos se clasifican de la siguiente manera:

- **Estructuras lógicas de datos:** En un programa, cada variable pertenece a alguna estructura de datos la cual determina el conjunto de operaciones válidas para ella. (Lógicamente se decide cual va a ser el tipo de datos que se le va a asignar a la variable). Cada estructura de datos lógica puede tener varias representaciones físicas diferentes para sus almacenamientos posibles.
- **Estructuras primitivas y simples:** Las estructuras primitivas no están compuestas por otras estructuras de datos, por ejemplo: enteros, booleanos y caracteres. Otras estructuras de datos se pueden construir de una o más primitivas. Las estructuras de datos simples se construyen a partir de estructuras primitivas, estas son: cadenas, arreglos y registros.
- **Estructuras lineales y no-lineales:** Las estructuras de datos simples se pueden combinar de varias maneras para formar estructuras más complejas. Las dos clases principales de estructuras de datos complejas son las lineales y las no-lineales, dependiendo de la complejidad de las relaciones lógicas que representan. Las estructuras de datos lineales incluyen pilas, colas y listas enlazadas. Las estructuras de datos no-lineales incluyen grafos y árboles.

## *1.1. Estructura de datos primitivas y simples*

En esta sección del Capítulo I se describirán los distintos tipos de estructuras de datos primitivas y simples.

### 1.1.1. Primitivas

A continuación se describen los tipos de estructuras de datos primitivas:

#### 1.1.1.1. Enteros

Un entero es un miembro del siguiente conjunto de números: {..., -(n+1), -n, …, -2, -1, 0, 1, 2, …., n, (n+1), …} . Las operaciones fundamentales sobre enteros son muy conocidas: suma, resta, multiplicación, división, exponenciación y otras.

#### 1.1.1.2. Reales

Son aquellos valores que poseen dígitos enteros y decimales. Pueden ser valores positivos o negativos. Por ejemplo 12.47, 0.12, 6x $10^{-3}$, 7x $10^{5}$.

#### 1.1.1.3. Carácter

Es un elemento tomado de un conjunto de símbolos, en el cual se incluyen dígitos, los caracteres del alfabeto y algunos caracteres especiales.

#### 1.1.1.4. Booleanos

A esta estructura de datos primitiva también se le llama lógico. Un dato booleano es un elemento que puede tener uno de dos valores: verdadero ó falso. Los tres operadores booleanos básicos son not (no), and (y) y or(o).

### 1.1.2. Simples

Los distintos tipos de estructuras de datos simples se describen a continuación:

#### 1.1.2.1. Cadenas

Una cadena (string en inglés) de caracteres es una sucesión de caracteres que se encuentran delimitados por una comilla o dobles comillas, según el tipo de lenguaje de programación.

#### 1.1.2.2. Arreglos

Un arreglo es un conjunto finito ordenado de elementos homogéneos. La propiedad de ordenación significa que es posible identificar el primero, segundo, tercero, … y el enésimo elemento del arreglo. Los elementos del arreglo son homogéneos porque todos son del mismo tipo de datos. Un arreglo puede ser un conjunto de elementos de tipo cadena en tanto que otro puede ser de tipo entero. Los elementos de un arreglo pueden ser, a su vez, otro arreglo. A estos arreglos por lo general se les llama tablas.

Los arreglos se clasifican de la siguiente manera:

**a) Arreglos unidimensionales**

La forma más simple de un arreglo es el arreglo unidimensional, conocido como vector.

Un arreglo unidimensional contiene un nombre y una cantidad N de elementos y se puede representar de forma horizontal o vertical como lo podemos observar en el siguiente ejemplo (**Figura 1**) del arreglo llamado A.

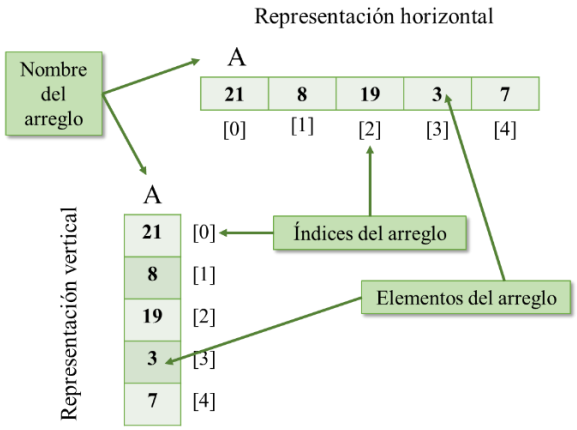



*Figura 1. Ejemplo de arreglos unidimensionales. Elaboración propia.*

Los **índices**, llamados también subíndices, de un elemento designan su posición en el arreglo. Estos pueden observarse en la **Figura 2**.

Un **elemento** en particular se define con el nombre del arreglo seguido por el subíndice del elemento entre paréntesis. En la siguiente figura se muestran los subíndices del arreglo

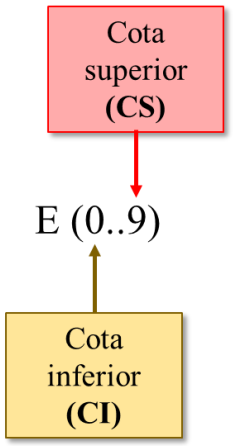



*Figura 2. Subíndices del arreglo. Elaboración propia.*

En algunas ocasiones la cuota inferior no se declara. Esto sucede cuando el subíndice inferior del arreglo tiene el valor de uno.

Para conseguir la implementación de cualquier estructura de datos se debe:

1. Reservar posiciones de memoria para la estructura de datos, y
2. Codificar las funciones de acceso.

En el caso de un arreglo unidimensional, las sentencias de declaración le dicen al compilador cuántas celdas se necesitan para representar el arreglo. El nombre del arreglo se asocia entonces con unas características del mismo.

Estas características incluyen:

1. La cota superior del rango índice (**CS**)
2. La cota inferior del rango índice (**CI**)
3. La posición de memoria de la primera celda del arreglo llamada la dirección base del arreglo (Base)
4. El número de posiciones de memoria necesitadas para cada elemento del arreglo (**Tamaño**)

La información sobre las características se almacena en una tabla llamada vector *dope*. Cuando accedemos a un elemento del arreglo, la función de acceso utiliza esta información para determinar la posición del elemento deseado.

Las fórmulas para utilizar son las siguientes:

1. Para calcular el **número de celdas** necesitadas:

   **Celdas = (CS - CI + 1) * tamaño**

2. El **tamaño** de los elementos es:

   **Entero = 1**

   **Real = 2**

   **Carácter = 1**

3. La **función de acceso** que nos da la posición de un elemento en un arreglo unidimensional asociado con la expresión índice es:

   **Dirección (índice) = Base + Desplazamiento por índice**

   En donde,

   **Desplazamiento por índice = (índice - CI) * tamaño**

   Así la función de acceso quedará de la siguiente forma:

   **Dirección (índice) = Base + [(índice - CI) * tamaño]**

**Ejemplo:** Dado el siguiente arreglo

***Dato [1..10]***

***tipo: entero***

***base = 100***

Encuentre:

a. El número de celdas necesitadas
b. Hacer la representación en memoria
c. Desarrollar la función de acceso en:
    1. Dato [5]
    2. Dato [7]

- **Solución:** Se busca la información necesaria para calcular el número de celdas necesitadas por el arreglo

| | |
|---|---|
| **Cota superior (CS)** | **10** |
| **Cota inferior (CI)** | **1** |
| **Tamaño** | **1** |

**a. Cálculo del número de celdas:**

**# Celdas = (CS - CI + 1) * tamaño**
# Celdas = (10 – 1 +1 ) * 1
# Celdas = 10

**b. Representación en memoria**

El compilador asigna memoria a las variables en orden secuencial a partir del valor base. En este arreglo la base es 100, a partir de ese valor se empieza a asignar los elementos. Recuerde que el tamaño de cada elemento es de 1 por ser de tipo entero.

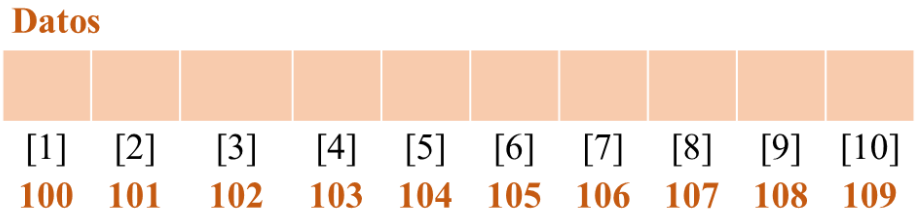



*Figura 3. Representación en memoria del arreglo Datos. Elaboración propia.*

**c. Buscar la función de acceso para el arreglo en:**

**Dato [5]**
**Dato [7]**

Usamos la siguiente fórmula:
**Dirección (índice) = Base + ((índice - CI) * tamaño)**

**Dato [5]**
Dirección [Dato (5)] = 100 + ((5 - 1) * 1)

Dirección [Dato (5)] = 100 + 4 = 104
Dirección [Dato (5)] = 104

**Dato [7]**
Dirección [Dato (7)] = 100 + ((7 - 1) * 1)
Dirección [Dato (7)] = 100 + 6
Dirección [Dato (7)] = 106

Los índices del arreglo pueden tener valores negativos como en el siguiente ejemplo.

**Ejemplo:** Dado el siguiente arreglo

**Valores [ -3..2]**

**tipo: real**

**base= 110**

Encuentre:

a. El número de celdas necesitadas
b. Hacer la representación en memoria
c. Desarrollar la función de acceso en:
    1. Valores [0]
    2. Valores [1]

- **Solución:** Se busca la información necesaria para calcular el número de celdas necesitadas por el arreglo

**Cota superior (CS)** **2**

**Cota inferior (CI)** **-3**

**Tamaño** **2**

**a. Cálculo del número de celdas:**

**# Celdas = (CS - CI + 1) * tamaño**
# Celdas = (2 - (-3) + 1) * 2
# Celdas = 12

**b. Representación en memoria**

El compilador asigna memoria a las variables en orden secuencial a partir del valor base. En este arreglo la base es 110, a partir de ese valor se empieza a asignar los elementos. Recuerde que el tamaño de cada elemento es de 2 por ser de tipo real.

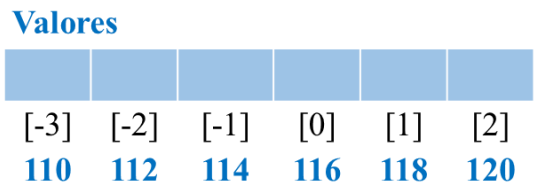



*Figura 4. Representación en memoria del arreglo Valores. Elaboración propia.*

**c. Buscar la función de acceso para el arreglo en:**

**Valores [0]**
**Valores [1]**

**Dirección (índice) = Base + ((índice - CI) * tamaño)**

**Valores [0]**
Dirección [Valores (0)] = 110 + (0 - (- 3) * 2)
Dirección [Valores (0)] = 110 + 6
Dirección [Valores (0)] = 116

**Valores [1]**
Dirección [valores (1)] = 110 + (1 - (- 3) * 2)
Dirección [valores (1)] = 110 + 8
Dirección [valores (1)] = 118

### b) Arreglos bidimensionales

Es un tipo de datos estructurado formado por una colección finita, de tamaño fijo de elementos homogéneos ordenados. Un par de índices específica el elemento deseado dando su posición relativa.

Un arreglo bidimensional es una forma natural de representar datos que puede verse de forma lógica como una tabla con filas y columnas como se puede observar en la **Figura 5**.

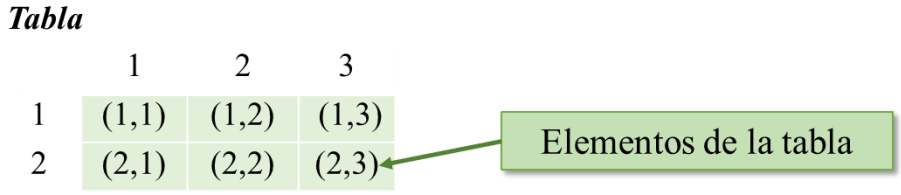



*Figura 5. Representación del arreglo bidimensional. Elaboración propia.*

La función de acceso para el arreglo bidimensional es: ***Nombre (fila, columna)***

Las fórmulas para utilizar son las siguientes:

1. Para calcular el número de celdas necesitadas
   ***# Celdas = [(CS fila - CI fila + 1) * (CS columna - CI columna + 1)] * tamaño***

2. El tamaño de los elementos es:
   **Entero = 1**
   **Real = 2**
   **Carácter = 1**

3. La función de acceso que nos da la posición de un elemento en un arreglo bidimensional asociado con la expresión índice (fila, columna). El arreglo puede estar ordenado en una de las siguientes formas:
   - **Ordenado en orden principal por fila es:**
     ***Dirección [elemento (fila, columna)] = Base + tamaño * [(CS columna - CI Columna + 1) * (fila - CI fila) + (columna - CI columna)]***

   - **Ordenado en orden principal por columna es:**
     ***Dirección [elemento (fila, columna)] = Base + tamaño * [(columna - CI columna) * (CS fila - CI fila + 1) + (fila - CI fila)]***

La memoria de la computadora es una secuencia larga de celdas a las que se accede por una dirección que comienza en cero. Por tanto, un arreglo bidimensional se almacena con todos los elementos de una **fila (**o columna), y así sucesivamente. Si los elementos de una **fila** se almacenan a continuación de los de otra, se dice que el arreglo está almacenado por **orden principal de fila**. Si los elementos de una columna se almacenan a continuación de los de otra, el arreglo está en **orden principal de columna**.

Cuando en un arreglo no aparece la cota inferior se asume que su valor es 1, como por ejemplo A(4,3) en cuyo caso la cota inferior de la fila es 1, la cota superior de la fila es 4; la cota inferior de la columna es 1 y la cota superior de la columna es 3.

Cuando las cotas inferiores no son iguales a 1 entonces se especifican los arreglos de la siguiente forma: B[0..7, 2..5]. Donde,

CI fila = 0 CS fila = 7

CI columna = 2 CS columna = *5*

**Ejemplo:** Dado el siguiente arreglo

**Ventas (3,2)**

**tipo: real**

**base= 200**

Encuentre:

a. El número de celdas necesitadas
b. Hacer la representación en memoria en orden principal de:
   1. Filas
   2. Columnas
c. Para cada una de las representaciones anteriores, desarrollar la función de acceso en:
   - Ventas (2,1)
   - Ventas (3,1)

- **Solución:** Se busca la información necesaria para calcular el número de celdas necesitadas por el arreglo:

| | |
|---|---|
| **Cota superior fila (CS fila)** | **3** |
| **Cota inferior fila (CI fila)** | **1** |
| **Cota superior columna (CS columna)** | **2** |
| **Cota inferior columna (CI columna)** | **1** |
| **Tamaño** | **2** |

**a. Cálculo del número de celdas:**

El compilador asigna memoria a las variables en orden secuencial a partir del valor base. En este arreglo la base es 200, a partir de ese valor se empieza a asignar los elementos. Recuerde que el tamaño de cada elemento es de 2 por ser de tipo real.

**1. Representación en memoria en orden principal de fila**

Ventas

| | | | | | |
|---|---|---|---|---|---|
| [1,1] | [1,2] | [2,1] | [2,2] | [3,1] | [3,2] |
| 200 | 202 | 204 | 206 | 208 | 210 |

*Figura 6. Representación en memoria en orden principal de fila del arreglo Ventas. Elaboración propia.*

**Función de acceso por orden principal de filas:**

Usamos la siguiente fórmula:

**Dirección [elemento (fila, columna)] = Base + tamaño * [(CS columna - CI Columna + 1) * (fila - CI fila) + (columna - CI columna)]**

Dirección [Ventas (2,1)] = 200 + 2 * [(2 - 1 + 1) * (2 - 1) + (1 – 1)]

= 200 + 2 * 2

= 204

Dirección [Ventas (3,1)] = 200 + 2 *[(2 - 1 + 1) * (3 - 1) + (1 – 1)]

= 200 + 2 * 4

= 208

2. **Representación en memoria en orden principal de columna**

Ventas

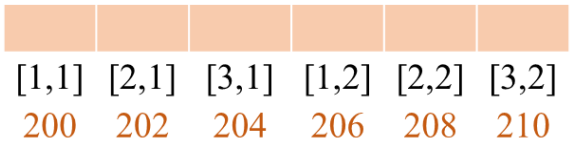



*Figura 7. Representación en memoria en orden principal de columna del arreglo Ventas. Elaboración propia.*

**Función de acceso por orden principal de columnas:**
Usamos la siguiente fórmula:

**Dirección [elemento (fila, columna)] = Base + tamaño * [(columna - CI columna) * (CS fila - CI fila + 1) + (fila - CI fila)]**

Dirección [Ventas (3,1)] = 200 + 2 * [(1 – 1) * (3 - 1 + 1) + (3 – 1)]

= 200 + 2 * 2

= 204

Dirección [Ventas (1,2)] = 200 + 2 * [(2 – 1) * (3 - 1 + 1) + (1 – 1)]

= 200 + 2 * 3

= 206

### 1.1.2.3. Registros

Un registro es una colección finita y ordenada de elementos, posiblemente heterogéneos, que se tratan como una unidad. Un registro se distingue de un arreglo en el hecho de que todos los elementos de un arreglo deben tener la misma estructura, a diferencia de los elementos componentes del registro que pueden tener diferentes estructuras de datos. Un registro se menciona algunas veces sólo como una estructura.

Los elementos de un registro son comúnmente llamados campos. Un campo es un área específica de un registro utilizada para una clase particular de información. A continuación, se muestra un ejemplo de un registro.

```
Empleado =    registro
              Nombre: arreglo [1..30] carácter
              Cedula: entero
              Dirección: arreglo [1..15] carácter
              Teléfono: entero
              Salario: real
Fin empleado
```

Este registro se puede representar de la siguiente forma:

Registro - Empleado

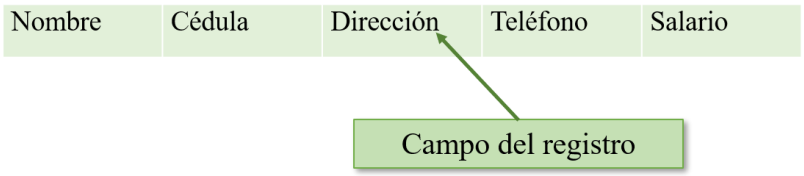



*Figura 8. Representación del registro Empleado. Elaboración propia.*

- **Campo**

El acceso se hace directamente mediante un conjunto de selectores de campos con nombres.

Sintaxis de la función de acceso: *Variable registro.selector de campo*.

Por ejemplo: ***Empleado.cedula***. La sintaxis de la función de acceso es el nombre de la variable de registro; seguida de un punto y del selector de campo de la componente en la que está interesado.

Si esta expresión es la parte derecha de una sentencia de asignación, se extrae la información de esa posición, por ejemplo: nombre = Empleado.nombre.

Si está en la parte izquierda, se almacena un valor a ese campo del registro, por ejemplo: Empleado.salario = *500.00*

Un registro ocupa un bloque de celdas consecutivas de memoria. El nombre del registro se asocia con una posición de memoria. Para acceder a cada campo, necesitamos simplemente calcular cuánto hay que saltar del registro para obtener el campo deseado. Esto se hace calculando cuántas celdas necesita cada campo.

**Ejemplo:**

Dado el registro Empleado

```
Empleado =      registro
        Nombre: arreglo [1..30] carácter
        Cedula: entero
        Dirección: arreglo [1..15] carácter
        Teléfono: entero
        Salario: real
Fin empleado
```

Encontrar:

a. Longitud y dirección de inicio de cada campo, el registro tiene una dirección inicial de 200
b. Calcular la dirección de los campos:
   1. Cédula
   2. Teléfono

- **Solución:**
  **a. Longitud y dirección de inicio de cada campo**

| **Campo** | Nombre | Cédula | Dirección | Teléfono | Salario |
|---|---|---|---|---|---|
| **Longitud** | 30 | 1 | 15 | 1 | 2 |
| **Dirección del campo** | 200 | 230 | 231 | 246 | 247 |

**b. Dirección de los campos**

Para calcular la dirección del campo se utilizará la función:

**Dirección = Base + Longitud de los campos a saltar**

Empleado.cedula = 200 + 30 = 230

Empleado.teléfono = 200 + 30 + 1 + *15* = 246

## *1.2. Estructura de datos lineales y recursividad*

En esta sección del Capítulo I se describirán los distintos tipos de estructuras de datos lineales y la recursividad.

### 1.2.1 Introducción

Las estructuras lineales de datos se caracterizan porque sus elementos están en secuencia, relacionados en forma lineal, uno luego del otro. Cada elemento de la

estructura puede estar conformado por uno o varios subelementos o campos que pueden pertenecer a cualquier tipo de dato, pero que normalmente son tipos básicos.

### 1.2.2 Pila

Una pila es una lista de elementos en la que se pueden insertar y eliminar elementos sólo por uno de los extremos. Como consecuencia, los elementos de una pila serán eliminados en orden inverso al que se insertaron. Es decir, el último elemento que se metió a la pila será el primero en salir de ella. Debido al orden en que se insertan y eliminan los elementos en una pila, a ésta también se le conoce como estructura LIFO (Last In, First Out: último en entrar, primero en salir).

En la vida cotidiana existen muchos ejemplos de pilas, una pila de platos en una alacena, una pila de latas en un supermercado, una pila de papeles sobre un escritorio, etc.

#### 1.2.2.1 Operaciones sobre pila

Una pila es una estructura dinámica; cambia conforme se añaden y quitan elementos de ella. La operación que añade un elemento por la cabeza de la pila se llama ***Meter*** y la operación que toma el elemento de la cabeza de la pila y lo quita es denominada ***Sacar***. Cuando empezamos usando una pila, debe estar vacía, por lo que es necesario operaciones para crear una pila vacía (una pila que no contiene elementos). A esta operación la llamaremos ***LimpiarPila***. Debemos también poder decir si una pila contiene algún elemento que podamos sacar, por lo que necesitamos una operación booleana que denominaremos ***PilaVacía***. Cuando se van a añadir elementos a una pila, antes de meter el elemento, es necesario verificar si una pila está llena. A esta operación booleana se le llama ***PilaLlena***.

Resumiendo, las distintas operaciones sobre pilas tenemos:

- **Meter:** Operación que añade un elemento por la cabeza de la pila.
- **Sacar:** Operación que toma el elemento de la cabeza de la pila y lo quita.
- **LimpiarPila:** Operación para crear una pila vacía (una pila que no contiene elementos).
- **PilaVacía:** Operación para verificar si una pila contiene algún elemento que podamos sacar.
- **PilaLlena:** Operación para verificar si una pila está llena antes de meter un elemento.

#### 1.2.2.2 Implementación de pilas

Consideremos ahora la implementación de nuestra estructura de datos pila abstracta. Debido a que todos los elementos de una pila son del mismo tipo, un arreglo parece una estructura razonable para contener la pila. Podemos poner los elementos de forma secuencial en el arreglo, colocando el primer elemento en la primera posición del arreglo, el segundo en la segunda posición, y así sucesivamente. Para declarar la variable arreglo

que contenga la pila, debemos decidir el tamaño máximo de la misma (***MaxPila***). Por lo tanto, declaramos nuestra estructura de la siguiente forma:

**Pila = arreglo [1..MaxPila] entero**

Podemos implementar la pila como un arreglo solo debemos recordar que aunque podemos acceder a cualquier elemento del arreglo directamente, estamos de acuerdo en utilizar la función de acceso ***"último en entrar, primero en salir"*** para una pila. Por tanto, accederemos a los elementos de la pila sólo a través de la cabeza, no por el fondo o por la mitad. Reconocer esta diferencia desde el principio es importante.

Hay varias formas de conocer la posición cabeza. Podríamos utilizar alguna variable entera a la que llamaremos ***Cabeza*** que indicaría el índice a la posición actual de la cabeza en el arreglo. Sin embargo, este esquema exigiría pasar Cabeza como un parámetro adicional en nuestras operaciones sobre pila. Sería mejor encontrar una forma de vincular la colección de los elementos y el indicador cabeza una entidad simple, Pila. Este objetivo puede conseguirse extendiendo el arreglo de forma que incluya una posición más, ***Pila[0],*** en el cual almacenaremos el índice del elemento cabeza actual.

Ahora podemos comenzar a escribir las funciones y procedimientos para implementar nuestras operaciones de pila

**Procedimiento LimpiarPila**

```
Inicio
    Pila[cabeza] = 0
Fin
```

**Función PilaLlena**

```
Inicio
Si (Pila[cabeza] = maxPila)
  Entonces imprimir Pila llena
  Sino imprimir la pila no está llena
Fin si
Fin
```

**Función PilaVacía**

Inicio

**Si** (Pila[cabeza] = 0)

**Entonces** imprimir la pila está vacía

**Sino** imprimir la pila no está vacía

**Fin si**

Fin

### Procedimiento Meter

Inicio

Pila[cabeza] = Pila[cabeza] + 1

Pila[cabeza] = Nuevoelemento

Fin

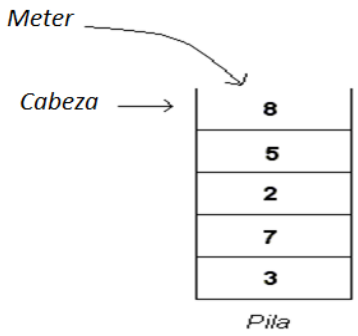



*Figura 9. Procedimiento Meter.*

### Procedimiento Sacar

Inicio

Elementosacado = Pila[Pila[cabeza] ]

Pila[cabeza] = Pila[cabeza] - 1

Fin

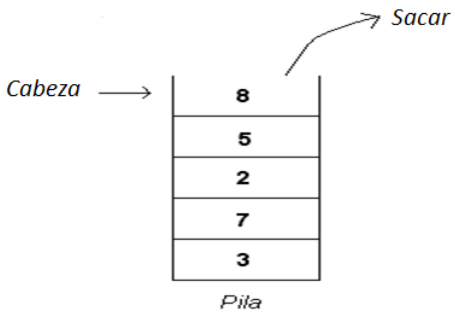



*Figura 10. Procedimiento Sacar.*

### 1.2.2.3 Ejemplo

Desarrolle las siguientes instrucciones utilizando pilas.

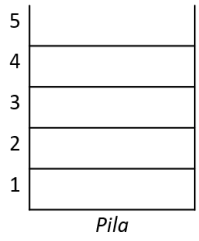
Limpiarpila (Pila) maxpila =5
X = 1 Y = 3
Z = Y * 2
Meter (Pila, Z)
Z = Y + X
Meter (Pila, Z)
Y = Z * 3 + X
Meter (Pila, Y)
Imprimir (Pila Contiene)
Mientras no-pilavacia (Pila) hacer
  Sacar (Pila, Y)
  Imprimir (Y)
Fin – Mientras

**Solución:**

**PASO 1**

Limpiarpila (Pila) maxpila =5

*Cabeza = 0*




**Procedimiento LimpiarPila**
Inicio
Pila[cabeza] = 0
Fin

**PASO 2**

Limpiarpila ( Pila ) maxpila =5

**Z = 3 * 2 = 6**

| X | Y | Z |
|---|---|---|
| 1 | 3 | 6 |

X = 1  Y = 3
Z = Y * 2

## PASO 3

Limpiarpila ( Pila )  maxpila =5
X = 1  Y = 3
Z = Y * 2
Meter (Pila, Z)

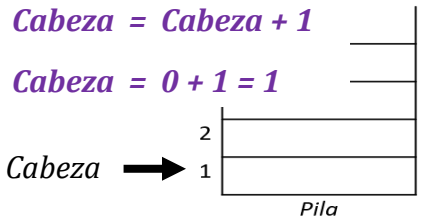
**Antes de meter un elemento a la pila hay que verificar que la misma no esté llena**

```
Función PilaLlena
Inicio
Si (Pila[cabeza] = maxPila)
   Entonces imprimir Pila llena
   Sino imprimir la pila no está llena
Fin si
Fin
```

*Cabeza = maxPila*
*0 = 5 FALSO*
La pila no está llena
Podemos meter el nuevo elemento

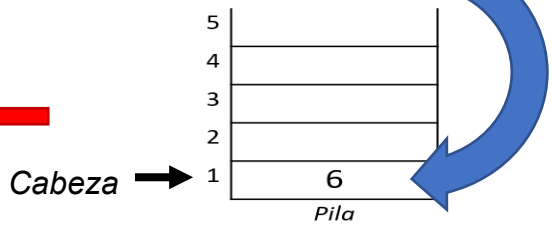
```
Procedimiento Meter
Inicio
        Pila[cabeza] = Pila[cabeza] + 1
        Pila[cabeza] = Nuevoelemento
Fin
```

*Cabeza = Cabeza + 1*
*Cabeza = 0 + 1 = 1*




```
Procedimiento Meter
Inicio
        Pila[cabeza] = Pila[cabeza] + 1
        Pila[cabeza] = Nuevoelemento
Fin
```

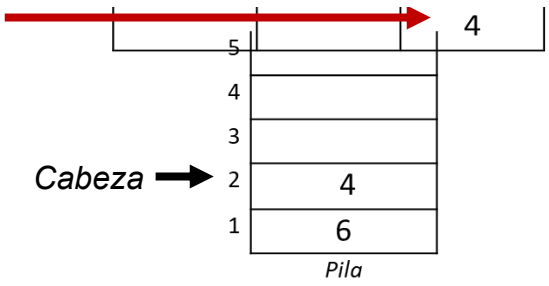
| X | Y | Z |
|---|---|---|
| 1 | 3 | 6 |



## PASO 4

Limpiarpila (Pila)  maxpila =5
X = 1  Y = 3
Z = Y * 2
Meter (Pila, Z)
Z = Y + X  **Y = 3 + 1 = 4**

***Cabeza = 5***
***1 = 5 FALSO***
La pila no está llena
Podemos meter el nuevo elemento
***Cabeza = 1 + 1 = 2***




## PASO 5

Limpiarpila (Pila)  maxpila =5
X = 1  Y = 3
Z = Y * 2
Meter (Pila, Z)
Z = Y + X
Meter (Pila, Z)

## PASO 6

Limpiarpila (Pila)  maxpila =5
X = 1  Y = 3
Z = Y * 2
Meter (Pila, Z)
Z = Y + X
Meter (Pila, Z)
Y = Z * 3 + X  **Y = 4 * 3 + 1 = 13**

| X | Y | Z |
|---|---|---|
| 1 | 3 | 6 |
| | | 4 |
| | 13 | |

## PASO 7

Limpiarpila (Pila)  maxpila =5
X = 1  Y = 3
Z = Y * 2
Meter (Pila, Z)
Z = Y + X
Meter (Pila, Z)
Y = Z * 3 + X
Meter (Pila, Y)

## PASO 8

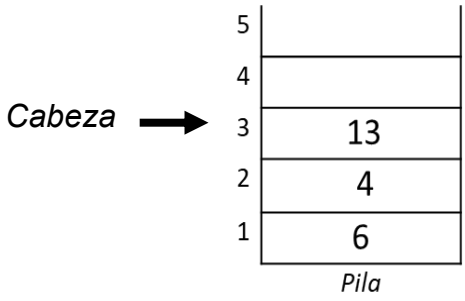
Limpiarpila (Pila)  maxpila =5
X = 1  Y = 3
Z = Y * 2
Meter (Pila, Z)
Z = Y + X
Meter (Pila, Z)
Y = Z * 3 + X
Meter (Pila, Y)
Imprimir (Pila Contiene)

**Función PilaVacía**
Inicio
**Si** (Pila[cabeza] = 0)
  **Entonces** imprimir la pila está vacía
  **Sino** imprimir la pila no está vacía
**Fin si**
Fin

***Cabeza = 0***
***3 = 0 FALSO***
La pila no está vacía
Podemos sacar el elemento de la pila
Podemos meter el nuevo elemento
***Cabeza = 2 + 1 = 3***

## PASO 9

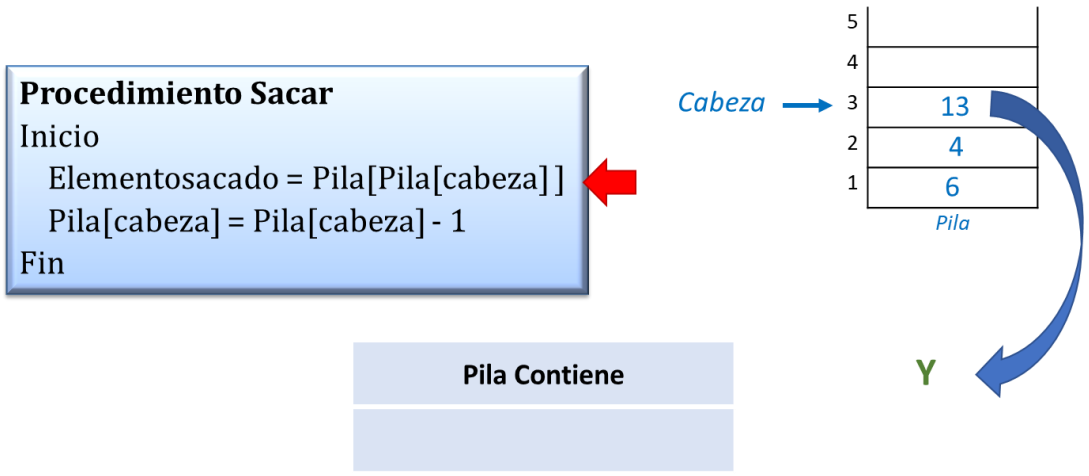
Limpiarpila (Pila)  maxpila =5
X = 1  Y = 3
Z = Y * 2
Meter (Pila, Z)
Z = Y + X
Meter (Pila, Z)



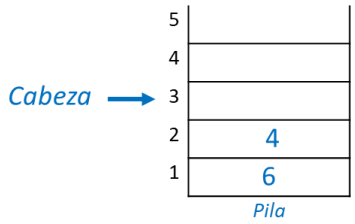
Y = Z * 3 + X
Meter (Pila, Y)
Imprimir (Pila Contiene)
Mientras no-pilavacia (Pila) hacer

## PASO 10

Limpiarpila (Pila) maxpila =5
X = 1 Y = 3
Z = Y * 2
Meter (Pila, Z)
Z = Y + X
Meter (Pila, Z)
Y = Z * 3 + X
Meter (Pila, Y)
Imprimir (Pila Contiene)
Mientras no-pilavacia (Pila) hacer
Sacar (Pila, Y)



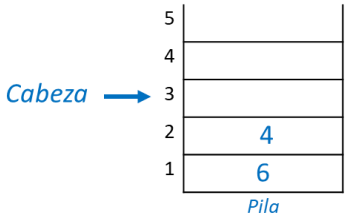
```
Procedimiento Sacar
Inicio
   Elementosacado = Pila[Pila[cabeza] ]
   Pila[cabeza] = Pila[cabeza] - 1
Fin
```




| Pila Contiene |
| --- |
| |

Y
13

```
Procedimiento Sacar
Inicio
   Elementosacado = Pila[Pila[cabeza] ]
   Pila[cabeza] = Pila[cabeza] - 1
Fin
```

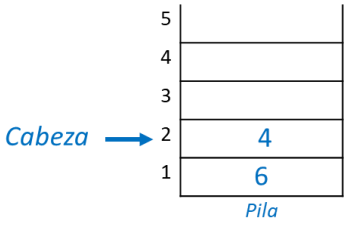



*Cabeza = 3 - 1 = 2*

| Pila Contiene |
| --- |
| |

Y
13

```
Procedimiento Sacar
Inicio
   Elementosacado = Pila[Pila[cabeza] ]
   Pila[cabeza] = Pila[cabeza] - 1
Fin
```




| Pila Contiene |
| --- |
| |

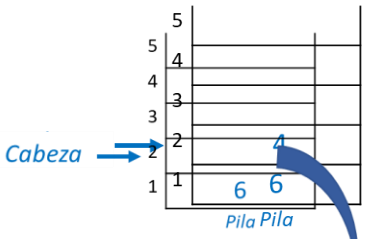
Y
13

## PASO 11

Limpiarpila (Pila)  maxpila =5
X = 1  Y = 3
Z = Y * 2
Meter (Pila, Z)
Z = Y + X
Meter (Pila, Z)
Y = Z * 3 + X
Meter (Pila, Y)
Imprimir (Pila Contiene)
Mientras  no-pilavacia (Pila) hacer
  Sacar (Pila,  Y)
  Imprimir (Y)
Fin – Mientras

## PASO 12

Limpiarpila (Pila)  maxpila =5
X = 1  Y = 3
Z = Y * 2
Meter (Pila, Z)
Z = Y + X
Meter (Pila, Z)
Y = Z * 3 + X
Meter (Pila, Y)
Imprimir (Pila Contiene)
Mientras  no-pilavacia (Pila) hacer

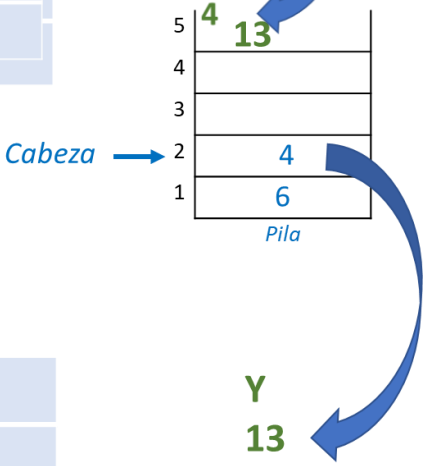



| Pila Contiene |
| --- |
| 13 |

Y
4
Y
13

*2 = 0*  ***FALSO***

La pila no está vacía
Podemos sacar el
elemento de la pila




| Pila Contiene |
| --- |
| 13 |

## PASO 13

Limpiarpila (Pila)  maxpila =5
X = 1  Y = 3
Z = Y * 2
Meter (Pila, Z)
Z = Y + X
Meter (Pila, Z)
Y = Z * 3 + X
Meter (Pila, Y)
Imprimir (Pila Contiene)
Mientras  no-pilavacia (Pila) hacer
  Sacar (Pila,  Y)


***Cabeza = 2 - 1 = 1***

| Pila Contiene |
| --- |
| 13 |

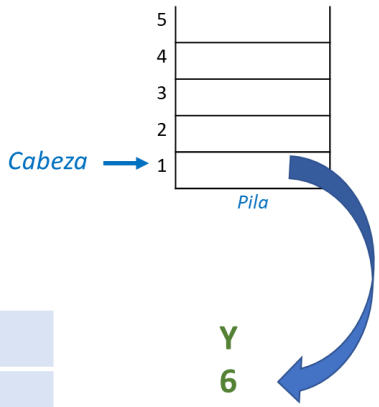
**Y**
**4**


| Pila Contiene |
| --- |
| 13 |

**Y**
**4**

**PASO 14**

Limpiarpila (Pila)  maxpila =5
X = 1  Y = 3
Z = Y * 2
Meter (Pila, Z)
Z = Y + X
Meter (Pila, Z)
Y = Z * 3 + X
Meter (Pila, Y)
Imprimir (Pila Contiene)
Mientras  no-pilavacia (Pila) hacer
  Sacar (Pila,  Y)
  Imprimir (Y)
Fin – Mientras




| Pila Contiene |
|---|
| 13 |
| 4 |

## PASO 15

Limpiarpila (Pila)  maxpila =5
X = 1  Y = 3
Z = Y * 2
Meter (Pila, Z)
Z = Y + X
Meter (Pila, Z)
Y = Z * 3 + X
Meter (Pila, Y)
Imprimir (Pila Contiene)
Mientras  no-pilavacia (Pila) hacer

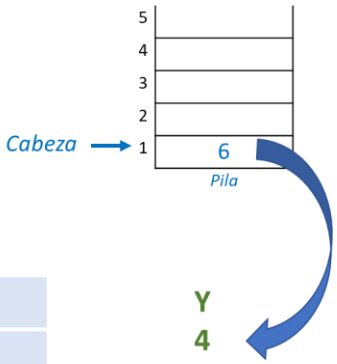

| Pila Contiene |
|---|
| 13 |
| 4 |

Y
4

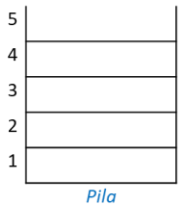
## PASO 16

Limpiarpila (Pila)  maxpila =5
X = 1  Y = 3
Z = Y * 2
Meter (Pila, Z)
Z = Y + X
Meter (Pila, Z)
Y = Z * 3 + X
Meter (Pila, Y)
Imprimir (Pila Contiene)
Mientras  no-pilavacia (Pila) hacer
  Sacar (Pila,  Y)

*1 = 0*  ***FALSO***
La pila no está vacía
Podemos sacar el
elemento de la pila



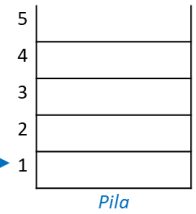

| Pila Contiene |
|---|
| 13 |
| 4 |




*Cabeza = 0*

| Pila Contiene |
|---|
| 13 |
| 4 |
| 6 |

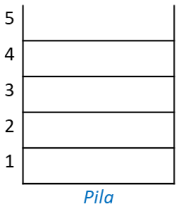
Y
6




***Cabeza = 1 - 1 = 0***

| Pila Contiene |
|---|
| 13 |
| 4 |

Y
6




*Cabeza = 0*

| Pila Contiene |
|---|
| 13 |
| 4 |

Y
6

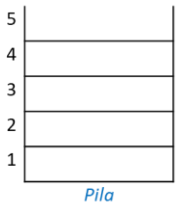
**PASO 17**

Limpiarpila (Pila)  maxpila =5
X = 1  Y = 3
Z = Y * 2
Meter (Pila, Z)
Z = Y + X
Meter (Pila, Z)
Y = Z * 3 + X
Meter (Pila, Y)
Imprimir (Pila Contiene)
Mientras  no-pilavacia (Pila) hacer
  Sacar (Pila,  Y)
  Imprimir (Y)
Fin – Mientras

**PASO 18**

Limpiarpila (Pila)  maxpila =5
X = 1  Y = 3
Z = Y * 2
Meter (Pila, Z)
Z = Y + X
Meter (Pila, Z)
Y = Z * 3 + X
Meter (Pila, Y)
Imprimir (Pila Contiene)
Mientras  no-pilavacia (Pila) hacer
  Sacar (Pila,  Y)
  Imprimir (Y)
Fin – Mientras

*0 = 0* ***CIERTO***
La pila está vacía




| Pila Contiene |
| --- |
| 13 |
| 4 |
| 6 |

Y
6

### 1.2.2.3 Evaluación de expresiones

Una de las implementaciones más comunes de las pilas es en la evaluación de expresiones aritméticas formadas por operandos y operadores. En donde, los operandos son variables que pueden tomar valores enteros o reales. Por otro lado, los operadores son aquellos con los que estamos acostumbrados a trabajar: suma, resta, división, multiplicación y exponenciación y, en donde, el orden de ejecución depende del orden de precedencia que existe entre ellos.

Una forma de garantizar que se realizará la evaluación correctamente es usar una pareja de paréntesis para cada operador. Los paréntesis dictan el orden de las operaciones evitando la ambigüedad y sin tener que recordar las reglas de precedencia. A la expresión resultante se le denomina expresión completamente agrupada. Esta es la forma natural que usamos para escribir las expresiones aritméticas y corresponde a lo que se denomina notación infija. Esta notación sitúa los operadores entre los operandos, por ejemplo: A – B * (C + D). Sin embargo, este método resulta difícil de implementar por la computadora.

Una de las formas de conseguir evaluar las expresiones aritméticas mediante la computadora es implementando la estructura tipo pila. En donde se utiliza una pila para almacenar los operadores y otra para almacenar los operandos para las evaluaciones y las conversiones de las expresiones. A continuación, se explica la conversión de expresiones infijas a prefijas y postfijas para luego explicar la evaluación de expresiones prefijas y postfijas.

Es importante recordar que antes de realizar la conversión o la evaluación de las expresiones **se debe inicializar las pilas de operandos y operadores.** Este paso se ha omitido para que solo nos enfoquemos en el trabajo requerido en cada caso

**a) Conversión de una expresión infija a prefija o postfija**

La diferencia al evaluar una expresión infija a prefija o postfija es la forma en la cual se escribe la expresión, esto es, debemos recordar que en una expresión prefija los operadores se encuentran al principio de la expresión mientras que en una expresión postfija los operadores se encuentran al final de la expresión.

Existen tres tipos expresiones:

- Expresión infija: ((A + 2) * (B + 4)) – 1
- Expresión prefija: - * + A 2 + B 4  1
- Expresión postfija: A 2 + B 4 + * 1 -

El orden de los operandos y el operador en cada una de estas expresiones es el siguiente:

- expresión infija: operando1 operador operando2
- expresión prefija: operador operando1 operando2
- expresión postfija: operando1 operando2 operador

Para poder hacer la evaluación de la expresión es necesario identificar correctamente al operando. Estos corresponden a:

operando2: primer operando que se saca de la pila de operandos

operando1: siguiente operando que se saca de la pila de operandos

**Pasos para la conversión de expresiones infijas a prefijas o postfijas**

1. Leer un carácter de la expresión hasta que no haya caracteres en la expresión de entrada.

   **1.1.** Si es un operador se coloca en la pila de operadores. **Ir al paso 1.**

   **1.2.** Si es un operando se coloca en la pila de operandos

   **1.3.** Si hay un operador en la pila de operadores, verificar
   - **Si hay dos o más operandos en la pila de operandos: ir al paso 1.5.**
   - **Sino: Ir al paso 1.**

   **1.4.** Si es un paréntesis:
   - **Si es izquierdo:** se mete en la pila de operandos. **Ir al paso 1.**
   - **Si es derecho: ir al paso 1.5.**

   **1.5.** Sacar los dos últimos operandos de la pila de operandos
   - **Si el operando1 es igual al paréntesis izquierdo, verificar:**
     - **Si se ha leído un paréntesis derecho:** meter el operando2 en la pila de operandos. **Ir al paso 1.3.**
     - **De lo contrario:** meter los operandos en la pila, en el mismo orden en que fueron sacados de la pila. **Ir al paso 1.**
   - **De lo contrario:** sacar el operador de la pila de operadores y evaluar la expresión en función de la conversión solicitada. Meter el resultado a la pila de operandos.

**Ejemplo 1:**
Convierta la siguiente expresión infija en prefija
**A + 3 * (B - 2)**

- **Solución:**

## PASO 1

A + 3 * (B - 2)

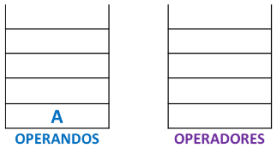
Caracteres leídos: A

Leer un carácter de la expresión de entrada

Es un operando, se coloca en la pila de operandos

Hay un operador en la pila de operadores

FALSO



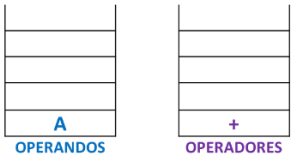

## PASO 2

A + 3 * (B - 2)

Caracteres leídos: A +

Leer un carácter de la expresión

Es un operador, se coloca en la pila de operadores




## PASO 3

A + 3 * (B - 2)

Caracteres leídos: A + 3

Leer un carácter de la expresión

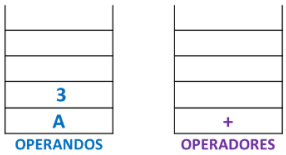
Es un operando, se coloca en la pila de operandos

Hay un operador en la pila de operadores

CIERTO

Hay dos o más operandos en la pila de operandos

CIERTO

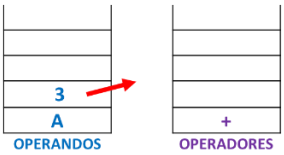



## PASO 4

Sacar los dos últimos operandos de la pila de operandos

A + 3 * (B - 2)

Caracteres leídos: A + 3




## PASO 5

Sacar los dos últimos operandos de la pila de operandos

operando1 es igual al paréntesis izquierdo

FALSO

A + 3 * (B - 2)

Caracteres leídos: A + 3


OPERANDO 2: 3

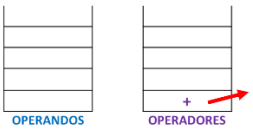
OPERANDO 1: A

## PASO 6

Sacar el operador de la pila de operadores y evaluar la expresión

A + 3 * (B - 2)

Caracteres leídos: A + 3




OPERANDO 2: 3

OPERANDO 1: A

OPERADOR: +

*Expresión prefija: operador operando1 operando2*

***+A3***

## PASO 7

Meter el resultado a la pila de operandos

A + 3 * (B - 2)

Caracteres leídos: A + 3


## PASO 8

Leer un carácter de la expresión de entrada

Es un operador, se coloca en la pila de operadores

Leer un carácter de la expresión de entrada

Si es un paréntesis izquierdo: se mete en la pila de operandos

CIERTO

Leer un carácter de la expresión de entrada

Es un operando se coloca en la pila de operandos

A + 3 * (B - 2)

Caracteres leídos: A + 3 * ( B

## PASO 9

Hay un operador en la pila de operadores

➡ CIERTO

Hay dos o más operandos en la pila de operandos

➡ CIERTO

**A + 3 * (B - 2)**

**Caracteres leídos: A + 3 * ( B**


## PASO 10

Sacar los dos últimos operandos de la pila de operandos

Si el operando1 es igual al paréntesis izquierdo

➡ CIERTO

**A + 3 * (B - 2)**

**Caracteres leídos: A + 3 * ( B**


OPERANDO 2: B

OPERANDO 1: (

## PASO 11

Meter los operandos en la pila, en el mismo orden en que fueron sacados de la pila

**A + 3 * (B - 2)**

**Caracteres leídos: A + 3 * ( B**


## PASO 12

Leer un carácter de la expresión de entrada

Es un operador, se coloca en la pila de operadores

Leer un carácter de la expresión de entrada

Es un operando se coloca en la pila de operandos

Hay un operador en la pila de operadores ➡ CIERTO

Hay dos o más operandos en la pila de operandos ➡ CIERTO

**A + 3 * (B - 2)**

**Caracteres leídos: A + 3 * ( B – 2**


## PASO 13

Sacar los dos últimos operandos de la pila de operandos

¿operando1 es igual al paréntesis izquierdo? ➡ FALSO

Sacar el operador de la pila de operadores y evaluar la expresión

Meter el resultado a la pila de operandos

**A + 3 * (B - 2)**

**Caracteres leídos: A + 3 * ( B – 2**


OPERANDO 2: 2

OPERANDO 1: B

OPERADOR: -

*Expresión prefija: operador operando1 operando2*

***-B2***

## PASO 14

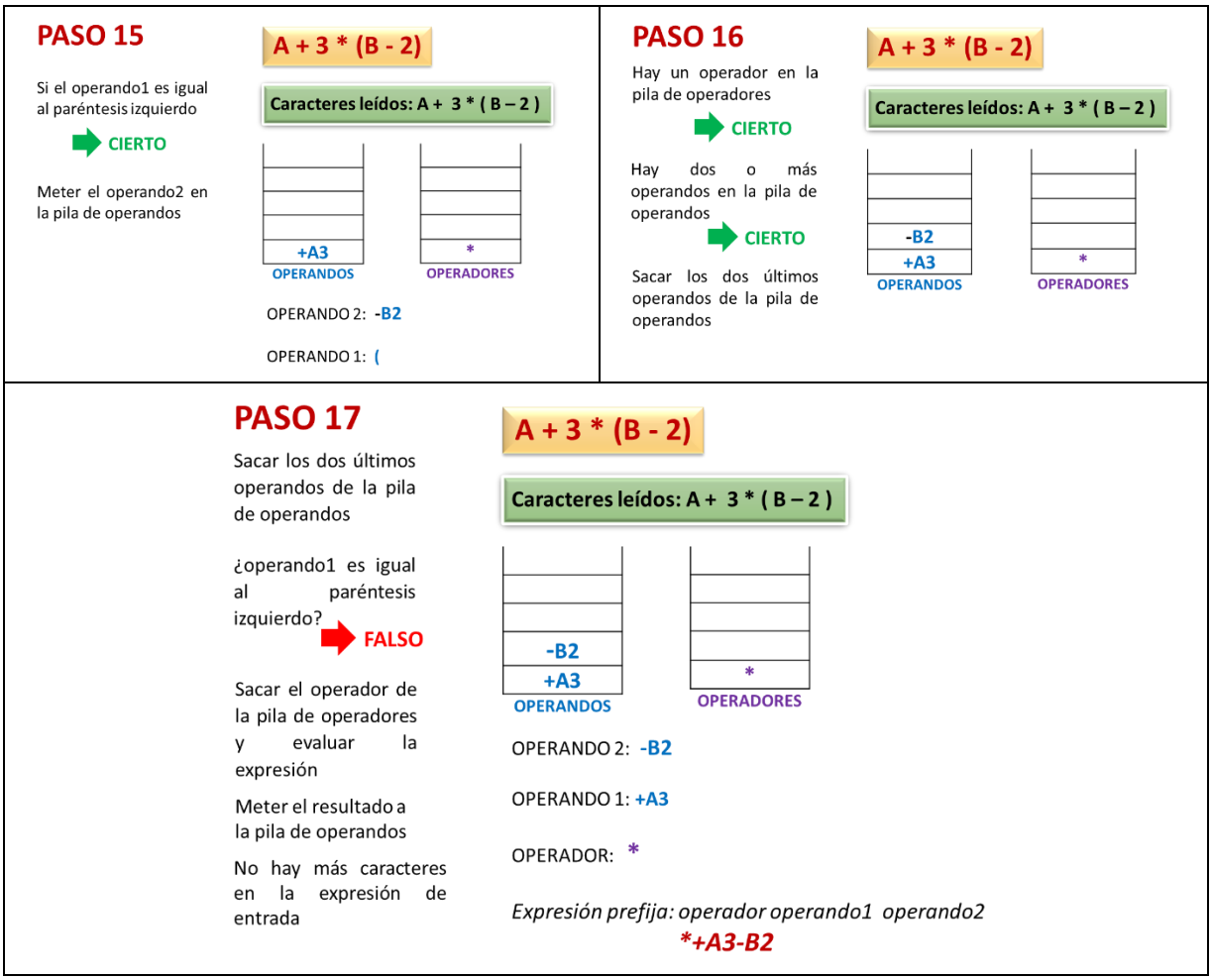
Leer un carácter de la expresión de entrada

¿Es un paréntesis derecho? ➡ CIERTO

Sacar los dos últimos operandos de la pila de operandos

**A + 3 * (B - 2)**

**Caracteres leídos: A + 3 * ( B – 2 )**

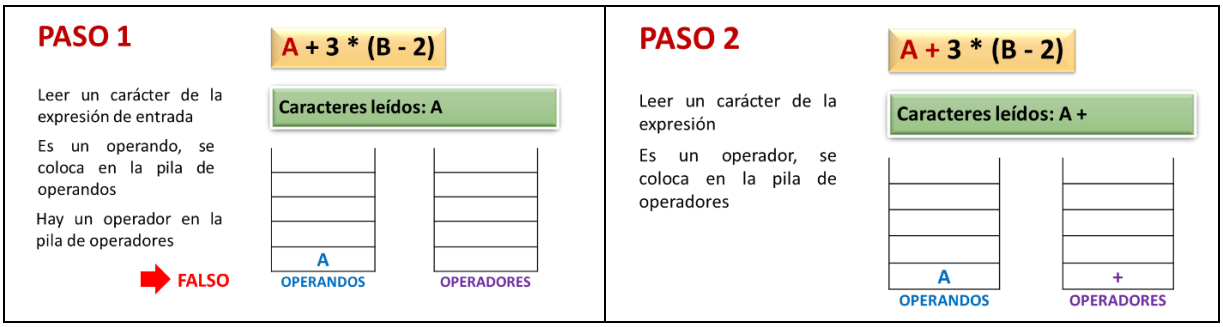



## Ejemplo 2:

Convierta la siguiente expresión infija en postfija

**A + 3 * (B - 2)**

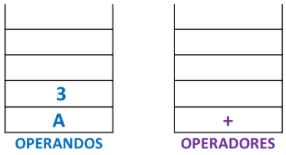
- **Solución:**



## PASO 3

A + 3 * (B - 2)

Caracteres leídos: A + 3




Leer un carácter de la expresión

Es un operando, se coloca en la pila de operandos

Hay un operador en la pila de operadores

CIERTO

Hay dos o más operandos en la pila de operandos

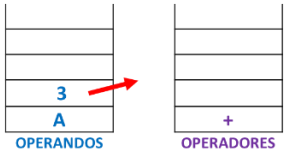
CIERTO

## PASO 4

Sacar los dos últimos operandos de la pila de operandos

A + 3 * (B - 2)

Caracteres leídos: A + 3




## PASO 5

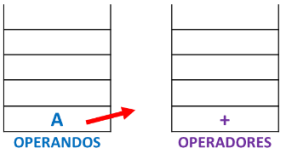
Sacar los dos últimos operandos de la pila de operandos

operando1 es igual al paréntesis izquierdo

FALSO

A + 3 * (B - 2)

Caracteres leídos: A + 3




OPERANDO 2: 3

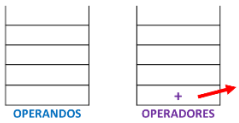
OPERANDO 1: A

## PASO 6

Sacar el operador de la pila de operadores y evaluar la expresión

A + 3 * (B - 2)

Caracteres leídos: A + 3




OPERANDO 2: 3

OPERANDO 1: A

OPERADOR: +

*Expresión postfija: operando1 operando2 operador*

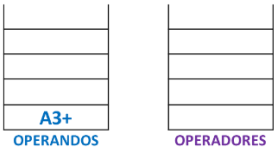
*A3+*

## PASO 7

Meter el resultado a la pila de operandos

A + 3 * (B - 2)

Caracteres leídos: A + 3




## PASO 8

Leer un carácter de la expresión de entrada

Es un operador, se coloca en la pila de operadores

Leer un carácter de la expresión de entrada

Si es un paréntesis izquierdo: se mete en la pila de operandos

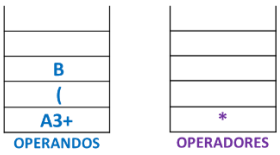
CIERTO

Leer un carácter de la expresión de entrada

Es un operando se coloca en la pila de operandos

A + 3 * (B - 2)

Caracteres leídos: A + 3 * ( B



## PASO 9

Hay un operador en la pila de operadores

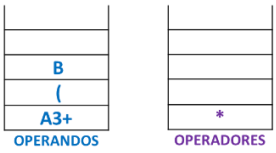
➡ CIERTO

Hay dos o más operandos en la pila de operandos

➡ CIERTO

**A + 3 * (B - 2)**

**Caracteres leídos: A + 3 * ( B**




## PASO 10

Sacar los dos últimos operandos de la pila de operandos

Si el operando1 es igual al paréntesis izquierdo

➡ CIERTO

**A + 3 * (B - 2)**

**Caracteres leídos: A + 3 * ( B**


OPERANDO 2: B

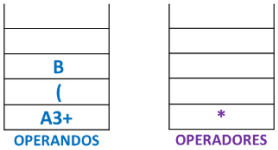
OPERANDO 1: (

## PASO 11

Meter los operandos en la pila, en el mismo orden en que fueron sacados de la pila

**A + 3 * (B - 2)**

**Caracteres leídos: A + 3 * ( B**




## PASO 12

Leer un carácter de la expresión de entrada

Es un operador, se coloca en la pila de operadores

Leer un carácter de la expresión de entrada

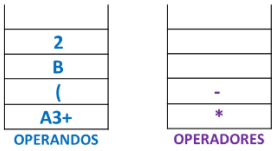
Es un operando se coloca en la pila de operandos

Hay un operador en la pila de operadores ➡ CIERTO

Hay dos o más operandos en la pila de operandos ➡ CIERTO

**A + 3 * (B - 2)**

**Caracteres leídos: A + 3 * ( B – 2**




## PASO 13

Sacar los dos últimos operandos de la pila de operandos

¿operando1 es igual al paréntesis izquierdo? ➡ FALSO

Sacar el operador de la pila de operadores y evaluar la expresión

Meter el resultado a la pila de operandos

**A + 3 * (B - 2)**

**Caracteres leídos: A + 3 * ( B – 2**


OPERANDO 2: 2

OPERANDO 1: B

OPERADOR: -

*Expresión postfija: operando1 operando2 operador*

***B2 -***

## PASO 14

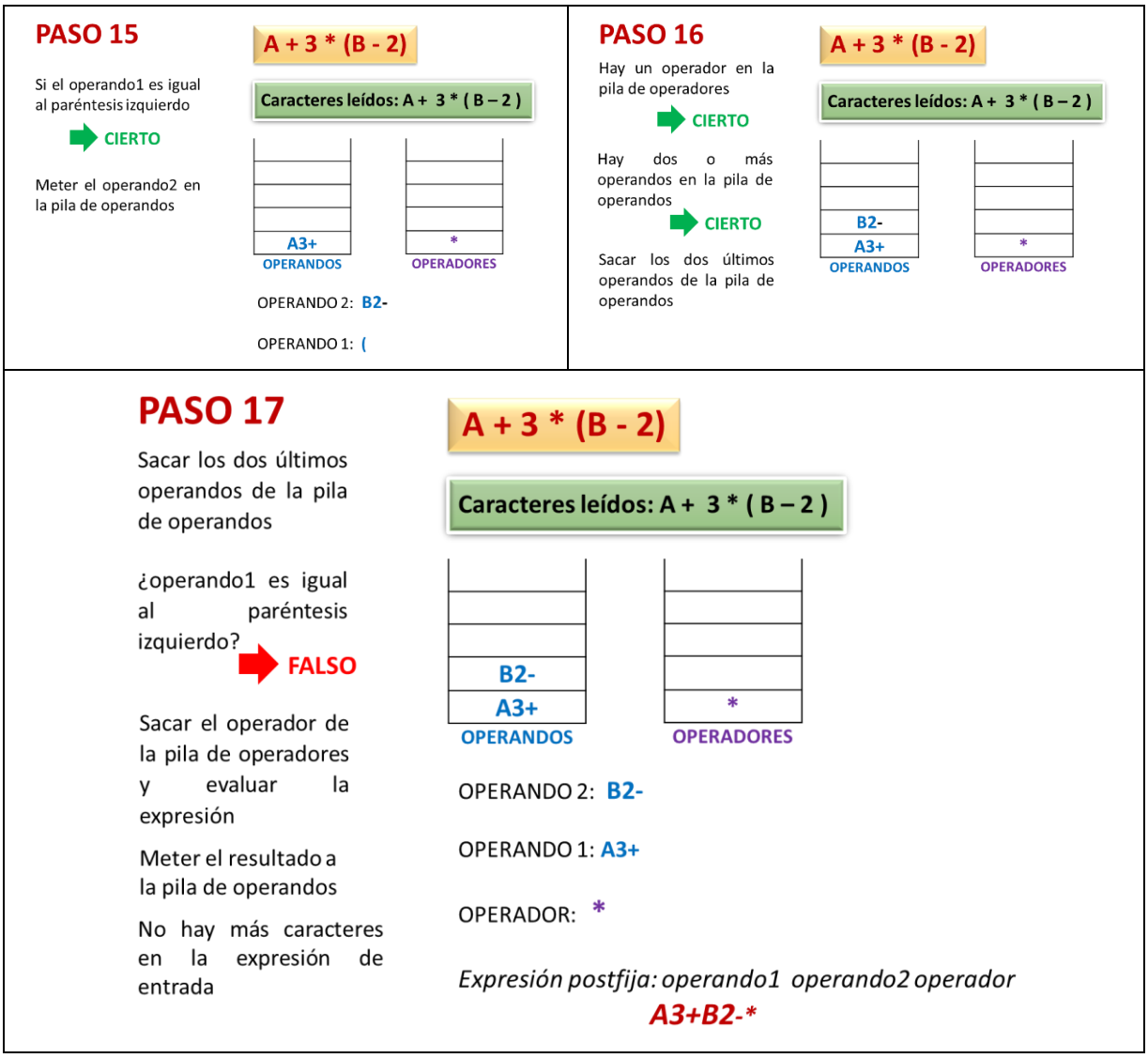
Leer un carácter de la expresión de entrada

¿Es un paréntesis derecho? ➡ CIERTO

Sacar los dos últimos operandos de la pila de operandos

**A + 3 * (B - 2)**

**Caracteres leídos: A + 3 * ( B – 2 )**




### b) Evaluación de una expresión prefija

**Pasos para la evaluación de expresiones prefijas**

1. Leer un carácter de la expresión hasta que no haya caracteres en la expresión de entrada.

   **1.1.** Si es un operador se coloca en la pila de operadores. **Ir al paso 1.**

   **1.2.** Si es un operando se coloca en la pila de operandos, verificar

   - **Si hay dos operandos precedidos por un operador, entonces:** sacar los dos últimos operandos de la pila de operandos y el operador de la pila de operadores. Evaluar la expresión de la siguiente forma:

operando1 operador operando2. Meter el resultado a la pila de operandos. **Ir al paso 1.**

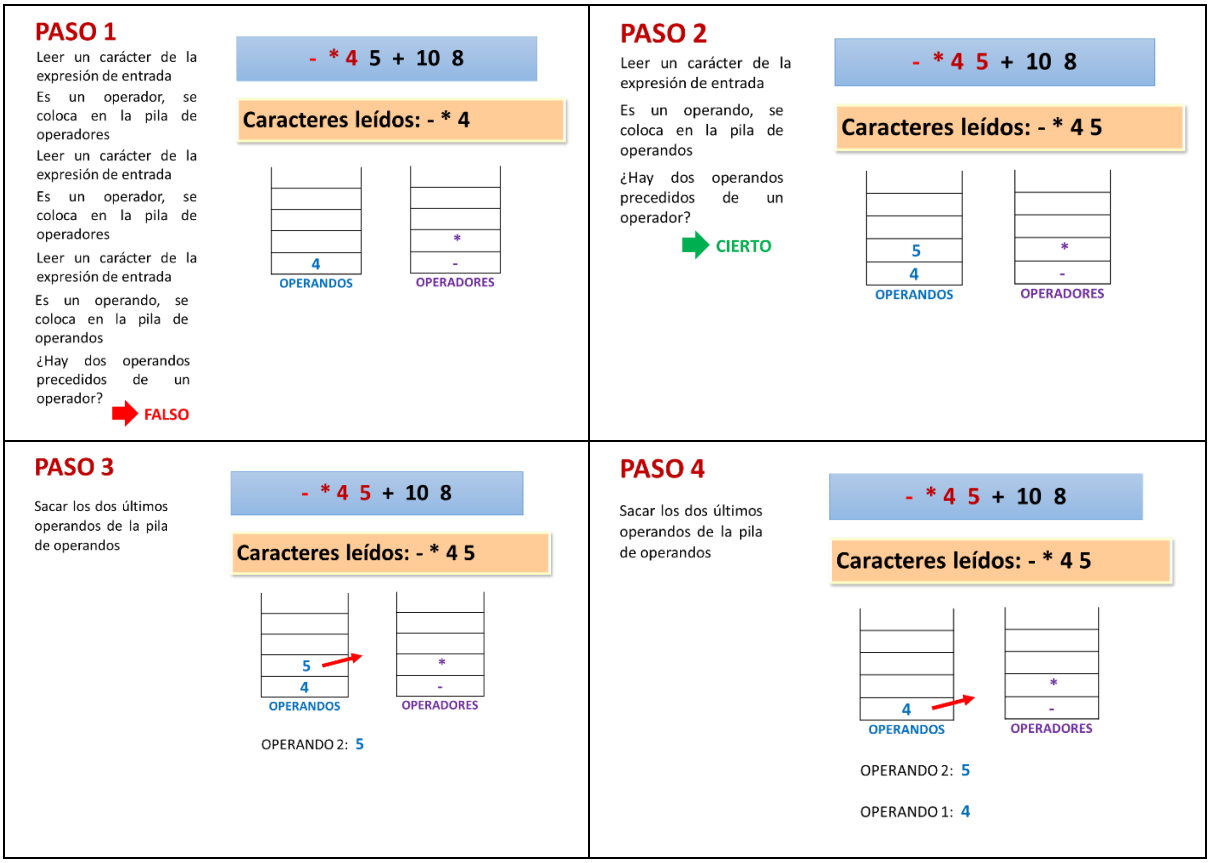
- **Sino: Ir al paso 1.**

**Ejemplo:**

Evaluar la siguiente expresión prefija

**- * 4 5 + 10 8**

- **Solución:**



## PASO 5

Sacar el operador de la pila de operadores

Evaluar la expresión

**- * 4 5 + 10 8**

**Caracteres leídos: - * 4 5**


OPERANDO 2: 5

OPERANDO 1: 4

OPERADOR: *

*Expresión prefija: operando1 operador operando2*

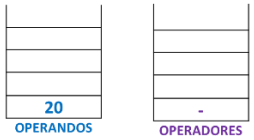
***4 * 5 = 20***

## PASO 6

Meter el resultado a la pila de operandos

**- * 4 5 + 10 8**

**Caracteres leídos: - * 4 5**




## PASO 7

Leer un carácter de la expresión de entrada

Es un operador, se coloca en la pila de operadores

Leer un carácter de la expresión de entrada

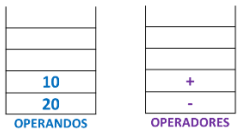
Es un operando, se coloca en la pila de operandos

¿Hay dos operandos precedidos de un operador?

➡ FALSO

**- * 4 5 + 10 8**

**Caracteres leídos: - * 4 5 + 10**




## PASO 8

Leer un carácter de la expresión de entrada

Es un operando, se coloca en la pila de operandos

¿Hay dos operandos precedidos de un operador?

➡ CIERTO

**- * 4 5 + 10 8**

**Caracteres leídos: - * 4 5 + 10 8**


## PASO 9

Sacar los dos últimos operandos de la pila de operandos

**- * 4 5 + 10 8**

**Caracteres leídos: - * 4 5 + 10 8**


OPERANDO 2: 8

## PASO 10

Sacar los dos últimos operandos de la pila de operandos

**- * 4 5 + 10 8**

**Caracteres leídos: - * 4 5 + 10 8**


OPERANDO 2: 8

OPERANDO 1: 10

## PASO 11

Sacar el operador de la pila de operadores

Evaluar la expresión

**- * 4 5 + 10 8**

**Caracteres leídos: - * 4 5 + 10 8**


OPERANDO 2: 8

OPERANDO 1: 10

OPERADOR: +

*Expresión prefija: operando1 operador operando2*

***10 + 8 = 18***

## PASO 12

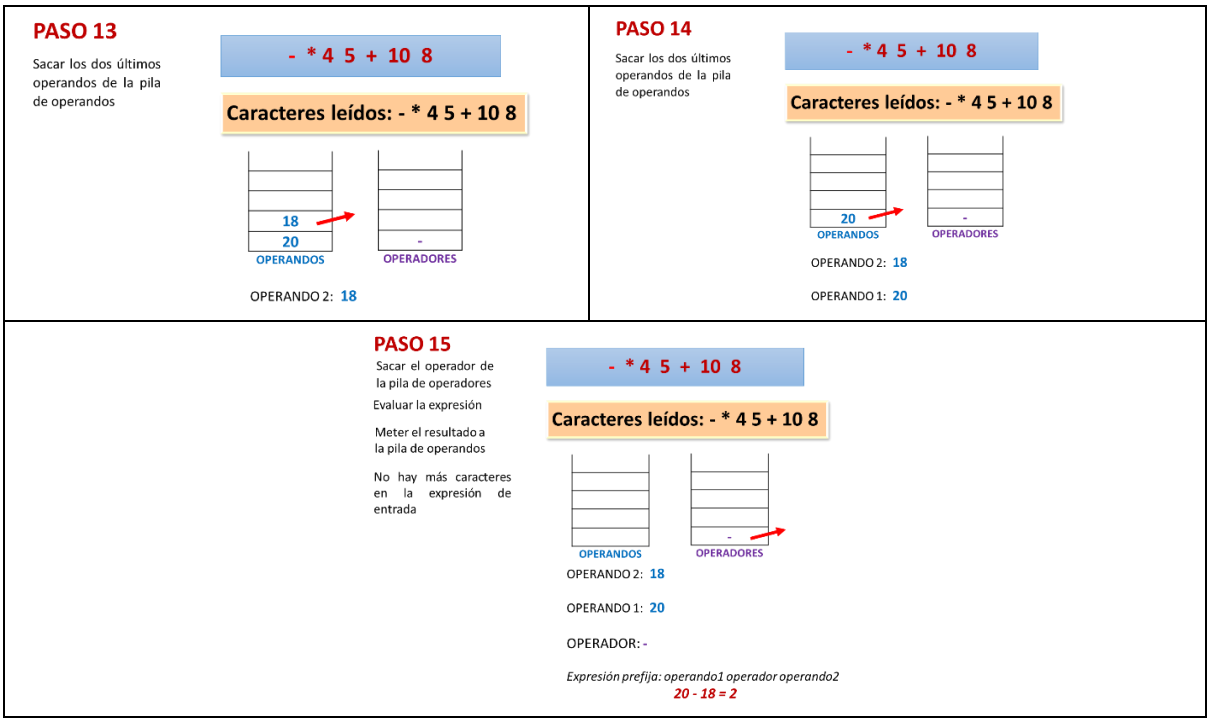
Meter el resultado a la pila de operandos

¿Hay dos operandos precedidos de un operador?

➡ CIERTO

**- * 4 5 + 10 8**

**Caracteres leídos: - * 4 5 + 10 8**




### c) Evaluación de una expresión postfija

## Pasos para la evaluación de expresiones postfijas

1. Leer un carácter de la expresión hasta que no haya caracteres en la expresión de entrada.

   **1.1.** Si es un operando se coloca en la pila de operandos. **Ir al paso 1.**

   **1.2.** i es un operador se coloca en la pila de operadores, verificar

   - **Si hay dos operandos seguidos de un operador, entonces:** sacar los dos últimos operandos de la pila de operandos y el operador de la pila de operadores. Evaluar la expresión de la siguiente forma: **operando1 operador operando2**. Meter el resultado a la pila de operandos. **Ir al paso 1.**
   - **Sino: Ir al paso 1.**

## Ejemplo:

Evaluar la siguiente expresión postfija
**4 5 * 10 8 + -**

- **Solución:**

**PASO 1**

Leer un carácter de la expresión de entrada

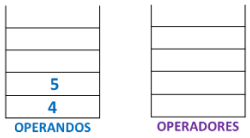
Es un operando, se coloca en la pila de operandos

Leer un carácter de la expresión de entrada

Es un operando, se coloca en la pila de operandos

4 5 * 10 8 + -

Caracteres leídos: 4 5




**PASO 2**

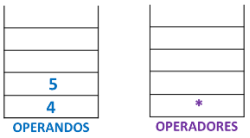
Leer un carácter de la expresión de entrada

Es un operador, se coloca en la pila de operadores

¿Hay dos operandos seguidos de un operador? CIERTO

4 5 * 10 8 + -

Caracteres leídos: 4 5 *

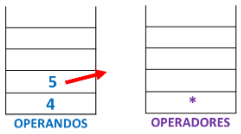



**PASO 3**

Sacar los dos últimos operandos de la pila de operandos

4 5 * 10 8 + -

Caracteres leídos: 4 5 *




OPERANDO 2: 5

**PASO 4**

Sacar los dos últimos operandos de la pila de operandos

4 5 * 10 8 + -

Caracteres leídos: 4 5 *


OPERANDO 2: 5

OPERANDO 1: 4

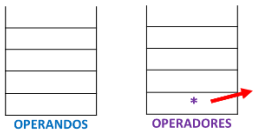
**PASO 5**

Sacar el operador de la pila de operadores

Evaluar la expresión

4 5 * 10 8 + -

Caracteres leídos: 4 5 *




OPERANDO 2: 5

OPERANDO 1: 4

OPERADOR: *

*Expresión prefija: operando1 operador operando2*

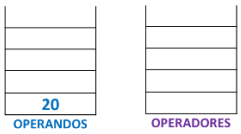
*4 * 5 = 20*

**PASO 6**

Meter el resultado a la pila de operandos

4 5 * 10 8 + -

Caracteres leídos: 4 5 *




**PASO 7**

Leer un carácter de la expresión de entrada

Es un operando, se coloca en la pila de operandos

Leer un carácter de la expresión de entrada

Es un operando, se coloca en la pila de operandos

4 5 * 10 8 + -

Caracteres leídos: 4 5 * 10 8


**PASO 8**

Leer un carácter de la expresión de entrada

Es un operador, se coloca en la pila de operadores

¿Hay dos operandos seguidos de un operador? CIERTO

4 5 * 10 8 + -

Caracteres leídos: 4 5 * 10 8 +

## PASO 9

Sacar los dos últimos operandos de la pila de operandos

**4 5 * 10 8 + -**

**Caracteres leídos: 4 5 * 10 8 +**


OPERANDO 2: 8

## PASO 10

Sacar los dos últimos operandos de la pila de operandos

**4 5 * 10 8 + -**

**Caracteres leídos: 4 5 * 10 8 +**


OPERANDO 2: 8

OPERANDO 1: 10

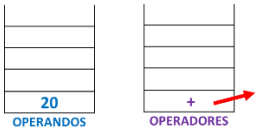
## PASO 11

Sacar el operador de la pila de operadores

Evaluar la expresión

**4 5 * 10 8 + -**

**Caracteres leídos: 4 5 * 10 8 +**




OPERANDO 2: 8

OPERANDO 1: 10

OPERADOR: +

*Expresión prefija: operando1 operador operando2*

***10 + 8 = 18***

## PASO 12

Meter el resultado a la pila de operandos

**4 5 * 10 8 + -**

**Caracteres leídos: 4 5 * 10 8 +**


## PASO 13

Leer un carácter de la expresión de entrada

Es un operador, se coloca en la pila de operadores

¿Hay dos operandos seguidos de un operador? CIERTO

**4 5 * 10 8 + -**

**Caracteres leídos: 4 5 * 10 8 + -**


## PASO 14

Sacar los dos últimos operandos de la pila de operandos

**4 5 * 10 8 + -**

**Caracteres leídos: 4 5 * 10 8 + -**


OPERANDO 2: 18

## PASO 15

Sacar los dos últimos operandos de la pila de operandos

**4 5 * 10 8 + -**

**Caracteres leídos: 4 5 * 10 8 + -**


OPERANDO 2: 18

OPERANDO 1: 20

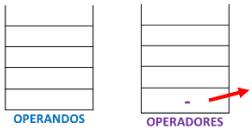
## PASO 16

Sacar el operador de la pila de operadores

Evaluar la expresión

**4 5 * 10 8 + -**

**Caracteres leídos: 4 5 * 10 8 + -**



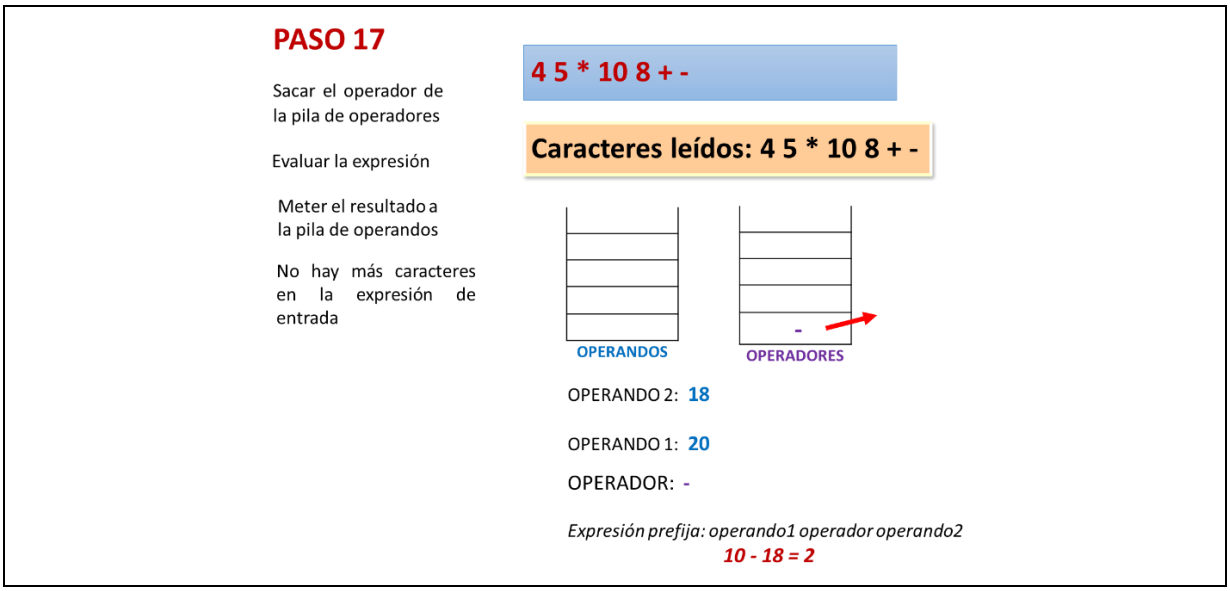

OPERANDO 2: 18

OPERANDO 1: 20

OPERADOR: -

*Expresión prefija: operando1 operador operando2*

***10 - 18 = 2***




### 1.2.3 Colas

Una cola es un grupo ordenado de elementos homogéneos en el que los nuevos elementos se añaden por un extremo de la cola (el **final**) y se quitan por el otro extremo de la cola (el **frente**). Una representación de una cola se muestra en la **Figura 11**.


*Figura 11. Funcionamiento de una cola. Freepik.es.(CCBY)*

Una cola es una estructura **"primero en entrar, primero en salir"**, se suele llamar **FIFO** (First Input First Output).

A diferencia de las pilas, una cola está abierta en ambos extremos. Un extremo siempre se usa para insertar datos (poner en cola) y el otro se usa para eliminar datos (quitar de cola).

Ponemos un elemento en la cola y luego cuando lo necesitamos lo quitamos de la cola. Si queremos cambiar el valor de un elemento, sacamos ese elemento de la cola, cambiamos su valor y luego lo añadimos a la cola. No podemos manipular directamente los valores de los elementos que estén en la cola.

### 1.2.3.1 Operaciones sobre cola

Las operaciones que pueden realizarse en una cola son:

- Pueden añadirse nuevos elementos a la cola a través de una operación que se llamará **InsCola**. InsCola(Cola, NuevoElemento) significa "añadir NuevoElemento al final de Cola". Las inserciones se llevarán a cabo por el Final de la cola. Para insertar un elemento debemos primero incrementar Final para que así contengan el índice al siguiente elemento libre del arreglo. Podemos entonces insertar el nuevo elemento en este espacio.

- Podemos también quitar elementos del frente de la cola a través de una operación que llamaremos **SupCola**. SupCola(Cola, Elemento) significa "quitar el dato del principio de Cola y devolver su contenido en Elemento". Las eliminaciones se harán por el Frente de la cola.

- Otra operación útil sobre la cola es ver si la cola está vacía. **ColaVacía**(Cola) devuelve True si la Cola está vacía y False en los demás casos. Podemos suprimir de la cola (SupCola) cuando la cola no está vacía.

- Antes de añadir un nuevo elemento a la cola debemos verificar que la cola no se encuentre llena. Llamaremos a esta función como **ColaLlena**. Si la cola no está llena podremos ir al procedimiento InsCola para insertar un nuevo elemento en la cola.

- También necesitamos una operación para inicializar una cola a un estado vacío, la cual se llamará **LimpiarCola**.

A continuación se resumen de las operaciones que se pueden ejecutar sobre las colas.

- ***InsCola(Cola, NuevoElemento):*** operación para añadir nuevos elementos al final de la cola.
- ***SupCola. SupCola(Cola, Elemento):*** operación para quitar elementos del frente de la cola.
- ***ColaVacía(Cola):*** operación para verificar si la cola está vacía.
- ***ColaLlena (Cola):*** operación para verificar si la cola está llena.
- ***LimpiarCola(Cola):*** operación para inicializar una cola a un estado vacío.

### 1.2.3.2 Implementación de cola

Por último, antes de mostrar los algoritmos que nos permiten trabajar con las colas, debemos recordar que estas implementaciones que se darán hacen uso eficiente de la memoria disponible al tratar a la cola como una estructura circular. Es decir, el elemento anterior al primero es el último.

**Algoritmos**

**Procedimiento LimpiarCola**

```
Inicio
        Cola.Frente = MaxCola
        Cola.Final = MaxCola
Fin
```

**Función ColaVacía**

```
Inicio

Si (Cola.Final = Cola.Frente)

    Entonces Imprimir (“La cola está vacía”)

    Sino Imprimir (“La cola no está vacía”)

Fin si

Fin
```

**Función ColaLlena**

```
/* Encuentra la siguiente posición disponible del final  */

Inicio

Si (Cola.Final = MaxCola)

        Entonces SigFinal = 1

        Sino SigFinal = Cola.Final + 1

Fin si


/* La cola está llena si la siguiente posición disponible del final es igual al frente de la cola
*/

Si (SigFinal = Cola.Frente)
```

```
        Entonces Imprimir (“La cola está llena”)
        Sino Imprimir (“La cola no está llena”)
Fin si
Fin
```

**Procedimiento InsCola**

```
/* Añade NuevoElemento al final de la cola. Supone que la cola no está llena */
Inicio
Si (Cola.Final = MaxCola)
        Entonces Cola.Final = 1
        Sino Cola.Final = Cola.Final + 1
Fin si

/* Añade un nuevo elemento al final de la cola */
Cola.Elementos[Cola.Final] = NuevoElemento
Fin
```

**Procedimiento SupCola**

```
/* Quita el elemento frente de Cola y lo devuelve en ElemSuprimido. Supone que la cola
no está vacía */
Inicio
Si (Cola.Frente = MaxCola)
        Entonces Cola.Frente = 1
        Sino Cola.Frente = Cola.Frente + 1
Fin si

/* Asigna el elemento frente de la cola a ElemSuprimido */
SupColaElemento = Cola.Elementos[Cola.Frente]
```

Fin

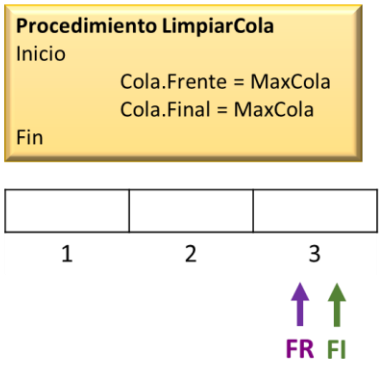
**Ejemplo:**
Desarrolle el siguiente algoritmo usando colas.

Limpiarcola (Cola)  maxcola =3
X = 0  Y = 1
R = X + Y
Mientras  (R < 10) Y (No Colallena)
    Inscola (Cola, R)
        X = Y
    Y = R
    R = X + Y
Fin – Mientras
Imprimir ( Cola Contiene)
Mientras  no-colavacia (Cola )  hacer
    Supcola (Cola, Y)
    Imprimir ( Y )
Fin – Mientras

- **Solución:**

**PASO 1**

Limpiarcola (Cola)  maxcola =3




**PASO 2**

Limpiarcola (Cola)  maxcola =5
X = 0  Y = 1
R = X + Y

| X | Y | R |
|---|---|---|
| 0 | 1 | 1 |

## PASO 3

Limpiarcola (Cola)  maxcola =5
X = 0  Y = 1
R = X + Y
Mientras  (R < 10) Y (No Colallena)

CIERTO  CIERTO

| X | Y | R |
|---|---|---|
| 0 | 1 | 1 |

*VERIFICAMOS QUE LA COLA NO ESTÉ LLENA*


SigFI ≠ FR
LA COLA NO ESTÁ LLENA

**Función ColaLlena**
/* Encuentra la siguiente posición disponible del final */
Inicio
1 → Si (Cola.Final = MaxCola) → CIERTO
2 → Entonces SigFinal = 1
Sino SigFinal = Cola.Final + 1
Fin si

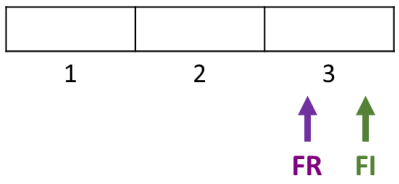
/* La cola está llena si la siguiente posición disponible del final es igual al frente de la cola */

3 → Si (SigFinal = Cola.Frente) → FALSO
Entonces Imprimir ("La cola está llena")
4 → Sino Imprimir ("La cola no está llena")
Fin si
Fin

## PASO 4

Limpiarcola (Cola)  maxcola =5
X = 0  Y = 1  R = X + Y
Mientras  (R < 10) Y (No Colallena)
Inscola (Cola, R)

| X | Y | R |
|---|---|---|
| 0 | 1 | 1 |




**Procedimiento InsCola**
/* Añade NuevoElemento al final de la cola. Supone que la cola no está llena */

Inicio
Si (Cola.Final = MaxCola) → CIERTO
Entonces Cola.Final = 1
Sino Cola.Final = Cola.Final + 1
Fin si

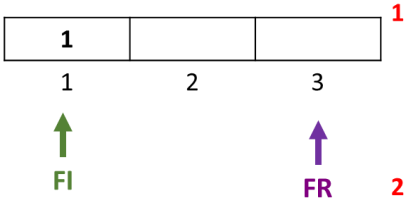
/* Añade un nuevo elemento al final de la cola */

Cola.Elementos[Cola.Final] = NuevoElemento
Fin

## PASO 4

Limpiarcola (Cola)  maxcola =5
X = 0  Y = 1  R = X + Y
Mientras (R < 10) Y (No Colallena)
Inscola (Cola, R)

| X | Y | R |
|---|---|---|
| 0 | 1 | 1 |




**Procedimiento InsCola**
/* Añade NuevoElemento al final de la cola. Supone que la cola no está llena */

Inicio
Si (Cola.Final = MaxCola)
  Entonces Cola.Final = 1
  Sino Cola.Final = Cola.Final + 1
Fin si

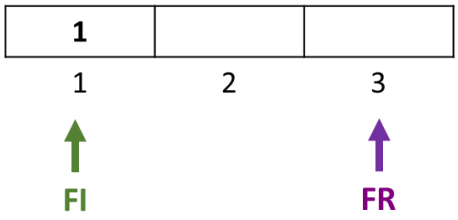
/* Añade un nuevo elemento al final de la cola */

Cola.Elementos[Cola.Final] = NuevoElemento
Fin

## PASO 5

Limpiarcola (Cola)  maxcola =5
X = 0  Y = 1  R = X + Y
Mientras (R < 10) Y (No Colallena)
Inscola (Cola, R)
X = Y
Y = R
R = X + Y
Fin – Mientras

| X | Y | R |
|---|---|---|
| 0 | 1 | 1 |
| 1 | 1 | 2 |



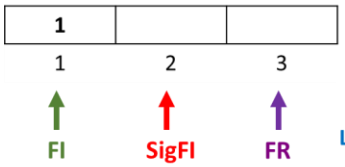
## PASO 6

Limpiarcola (Cola)  maxcola =5
X = 0  Y = 1  R = X + Y
Mientras (R < 10) Y (No Colallena)

CIERTO  CIERTO

| X | Y | R |
|---|---|---|
| 0 | 1 | 1 |
| 1 | 1 | 2 |

*VERIFICAMOS QUE LA COLA NO ESTÉ LLENA*




SigFI ≠ FR

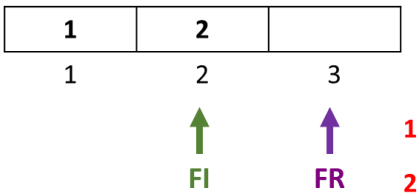
LA COLA NO ESTÁ LLENA

```
Función ColaLlena
/* Encuentra la siguiente posición disponible del final */
Inicio
1  Si (Cola.Final = MaxCola)  → FALSO
                Entonces SigFinal = 1
2               Sino SigFinal = Cola.Final + 1
Fin si

/* La cola está llena si la siguiente posición disponible del final
es igual al frente de la cola */

3  Si (SigFinal = Cola.Frente)  → FALSO
                Entonces Imprimir ("La cola está llena")
4               Sino Imprimir ("La cola no está llena")
Fin si
Fin
```

## PASO 7

Limpiarcola (Cola)  maxcola =5
X = 0  Y = 1  R = X + Y
Mientras (R < 10) Y (No Colallena)
Inscola (Cola, R)

| X | Y | R |
|---|---|---|
| 0 | 1 | 1 |
| 1 | 1 | 2 |

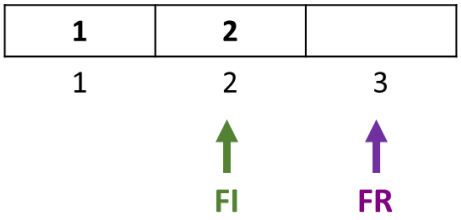



```
Procedimiento InsCola
/* Añade NuevoElemento al final de la cola. Supone que la
cola no está llena */

Inicio
1  Si (Cola.Final = MaxCola)  → FALSO
                Entonces Cola.Final = 1
2               Sino Cola.Final = Cola.Final + 1
Fin si

/* Añade un nuevo elemento al final de la cola */

3  Cola.Elementos[Cola.Final] = NuevoElemento
Fin
```

## PASO 8

Limpiarcola (Cola)  maxcola =5
X = 0  Y = 1  R = X + Y
Mientras (R < 10) Y (No Colallena)
Inscola (Cola, R)
X = Y
Y = R
R = X + Y
Fin – Mientras

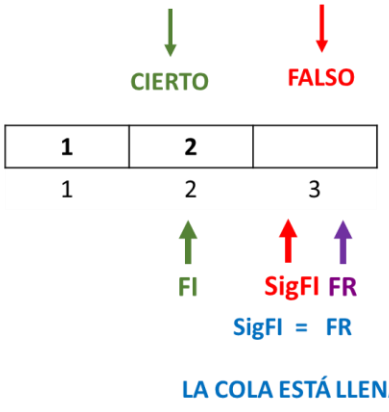
| X | Y | R |
|---|---|---|
| 0 | 1 | 1 |
| 1 | 1 | 2 |
| 1 | 2 | 3 |




## PASO 9

Limpiarcola (Cola)  maxcola =5
X = 0  Y = 1  R = X + Y
Mientras (R < 10) Y (No Colallena)

| X | Y | R |
|---|---|---|
| 0 | 1 | 1 |
| 1 | 1 | 2 |
| 1 | 2 | 3 |



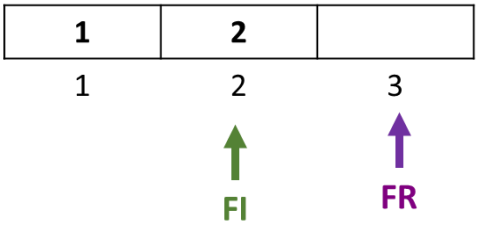

**Función ColaLlena**
/* Encuentra la siguiente posición disponible del final */
Inicio
1 → Si (Cola.Final = MaxCola) → FALSO
Entonces SigFinal = 1
2 → Sino SigFinal = Cola.Final + 1
Fin si

/* La cola está llena si la siguiente posición disponible del final es igual al frente de la cola */

3 → Si (SigFinal = Cola.Frente) → CIERTO
4 → Entonces Imprimir ("La cola está llena")
Sino Imprimir ("La cola no está llena")
Fin si
Fin

## PASO 10

Limpiarcola (Cola)  maxcola =3
X = 0  Y = 1  R = X + Y
Mientras  (R < 10) Y (No Colallena)
Inicio
Inscola (Cola, R)
X = Y
Y = R
R = X + Y
Fin – Mientras
**Imprimir ( Cola Contiene)**




| COLA CONTIENE |
|---|
| |

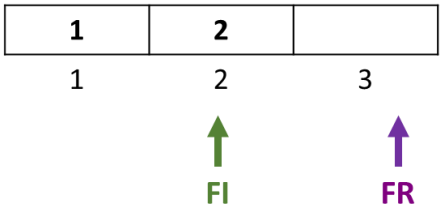
## PASO 11

Limpiarcola (Cola)  maxcola =3
X = 0  Y = 1  R = X + Y
Mientras  (R < 10) Y (No Colallena)
Inicio
Inscola (Cola, R)
X = Y
Y = R
R = X + Y
Fin – Mientras
Imprimir ( Cola Contiene)
**Mientras  no-colavacia (Cola )  hacer**

**FI ≠ FR**

**LA COLA NO ESTÁ VACÍA**

| COLA CONTIENE |
|---|
| |



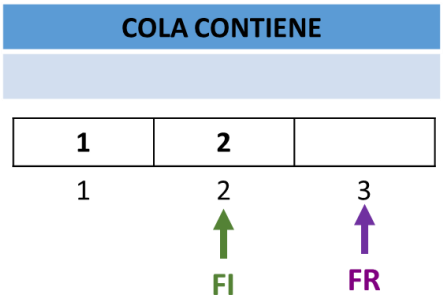

**1** →

**2** →

**Función ColaVacía**
Inicio
Si (Cola.Final = Cola.Frente) → **FALSO**
Entonces Imprimir (“La cola está vacía”)
Sino Imprimir (“La cola no está vacía”)
Fin si
Fin

## PASO 12

Limpiarcola (Cola)  maxcola =3
X = 0  Y = 1  R = X + Y
Mientras (R < 10) Y (No Colallena)
Inicio
  Inscola (Cola, R)
  X = Y
  Y = R
  R = X + Y
Fin – Mientras
Imprimir ( Cola Contiene)
Mientras no-colavacia (Cola ) hacer
  **Supcola (Cola, Y)**




**Procedimiento SupCola**
/* Quita el elemento frente de Cola y lo devuelve en ElemSuprimido. Supone que la cola no está vacía */

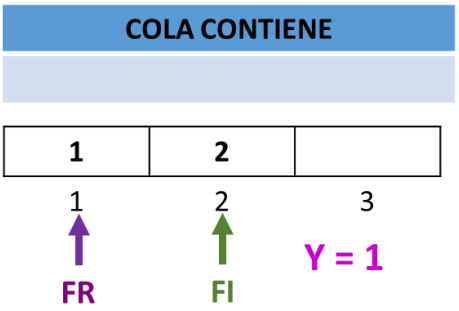
Inicio
1 → Si (Cola.Frente = MaxCola) → CIERTO
  Entonces Cola.Frente = 1
  Sino Cola.Frente = Cola.Frente + 1
2 → Fin si

/* Asigna el elemento frente de la cola a ElemSuprimido */

SupColaElemento = Cola.Elementos[Cola.Frente]
Fin

## PASO 12

Limpiarcola (Cola)  maxcola =3
X = 0  Y = 1  R = X + Y
Mientras  (R < 10) Y (No Colallena)
Inicio
  Inscola (Cola, R)
  X = Y
  Y = R
  R = X + Y
Fin – Mientras
Imprimir ( Cola Contiene)
Mientras  no-colavacia (Cola )  hacer
  **Supcola (Cola, Y)**




**Procedimiento SupCola**
/* Quita el elemento frente de Cola y lo devuelve en ElemSuprimido. Supone que la cola no está vacía */

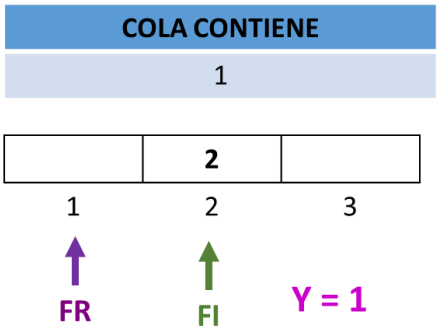
Inicio
Si (Cola.Frente = MaxCola)
  Entonces Cola.Frente = 1
  Sino Cola.Frente = Cola.Frente + 1
Fin si

/* Asigna el elemento frente de la cola a ElemSuprimido */

SupColaElemento = Cola.Elementos[Cola.Frente]
Fin

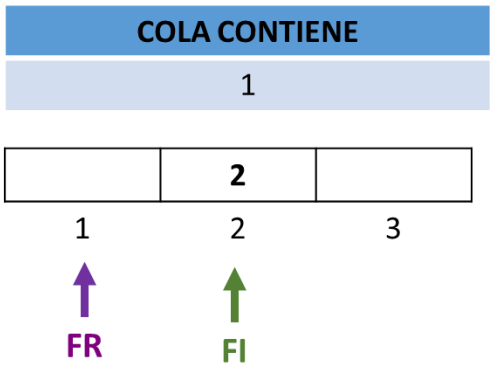
## PASO 13

Limpiarcola (Cola)  maxcola =3
X = 0  Y = 1  R = X + Y
Mientras  (R < 10) Y (No Colallena)
Inicio
  Inscola (Cola, R)
  X = Y
  Y = R
  R = X + Y
Fin – Mientras
Imprimir ( Cola Contiene)
Mientras  no-colavacia (Cola )  hacer
  Supcola (Cola, Y)
  **Imprimir ( Y )**
Fin – Mientras



## PASO 14

Limpiarcola (Cola)  maxcola =3
X = 0  Y = 1  R = X + Y
Mientras  (R < 10) Y (No Colallena)
Inicio
  Inscola (Cola, R)
  X = Y
  Y = R
  R = X + Y
Fin – Mientras
Imprimir ( Cola Contiene)
Mientras  no-colavacia (Cola )  hacer
  FI ≠ FR
  LA COLA NO ESTÁ VACÍA

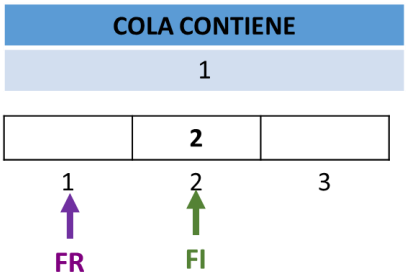



1 → 2 →

**Función ColaVacía**
Inicio
Si (Cola.Final = Cola.Frente) → FALSO
  Entonces Imprimir ("La cola está vacía")
  Sino Imprimir ("La cola no está vacía")
Fin si
Fin

## PASO 15

Limpiarcola (Cola)  maxcola =3
X = 0  Y = 1  R = X + Y
Mientras  (R < 10) Y (No Colallena)
Inicio
  Inscola (Cola, R)
  X = Y
  Y = R
  R = X + Y
Fin – Mientras
Imprimir ( Cola Contiene)
Mientras  no-colavacia (Cola )  hacer
  Supcola (Cola, Y)




1 → 2 →

**Procedimiento SupCola**
/* Quita el elemento frente de Cola y lo devuelve en ElemSuprimido. Supone que la cola no está vacía */

Inicio
Si (Cola.Frente = MaxCola) → FALSO
  Entonces Cola.Frente = 1
  Sino Cola.Frente = Cola.Frente + 1
Fin si

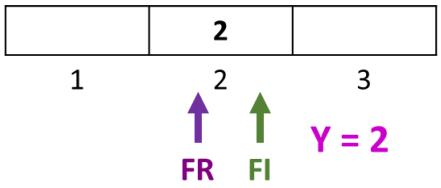
/* Asigna el elemento frente de la cola a ElemSuprimido */

SupColaElemento = Cola.Elementos[Cola.Frente]
Fin

## PASO 15

Limpiarcola (Cola)  maxcola =3
X = 0  Y = 1  R = X + Y
Mientras  (R < 10) Y (No Colallena)
Inicio
  Inscola (Cola, R)
  X = Y
  Y = R
  R = X + Y
Fin – Mientras
Imprimir ( Cola Contiene)
Mientras  no-colavacia (Cola )  hacer
  **Supcola (Cola, Y)**

| COLA CONTIENE |
|---|
| 1 |




**Procedimiento SupCola**
/* Quita el elemento frente de Cola y lo devuelve en ElemSuprimido. Supone que la cola no está vacía */

Inicio
Si (Cola.Frente = MaxCola)
  Entonces Cola.Frente = 1
  Sino Cola.Frente = Cola.Frente + 1
Fin si

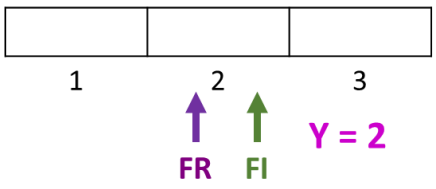
/* Asigna el elemento frente de la cola a ElemSuprimido */

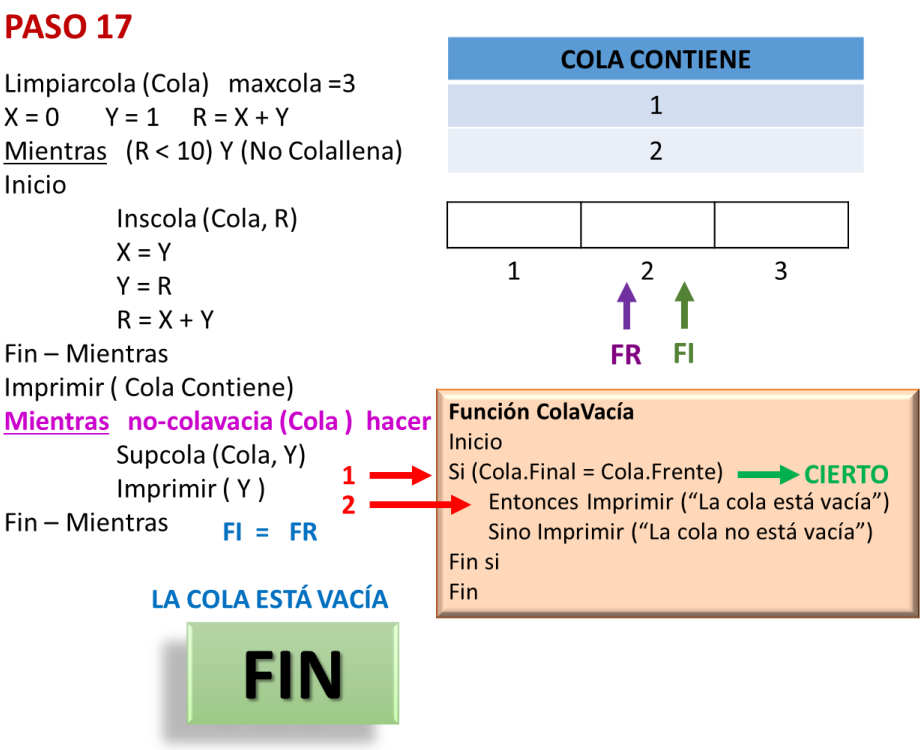
SupColaElemento = Cola.Elementos[Cola.Frente]
Fin

## PASO 16

Limpiarcola (Cola)  maxcola =3
X = 0  Y = 1  R = X + Y
Mientras  (R < 10) Y (No Colallena)
Inicio
  Inscola (Cola, R)
  X = Y
  Y = R
  R = X + Y
Fin – Mientras
Imprimir ( Cola Contiene)
Mientras  no-colavacia (Cola )  hacer
  Supcola (Cola, Y)
  **Imprimir ( Y )**
Fin – Mientras

| COLA CONTIENE |
|---|
| 1 |
| 2 |






## 1.2.4 Recursividad

### 1.2.4.1 Introducción

En las secciones anteriores de este capítulo, hemos visto formas de estructurar mediante funciones datos y programas. Estas funciones se resolvían por sí mismas o podían llamar a otras funciones que terminaban solucionando el problema. Sin embargo, hay veces en que una función sólo se puede resolver volviéndose a llamar a sí mismas. Este tipo de funciones son conocidas como funciones recursivas.

Esta técnica puede ser utilizada en lugar de la iteración dando excelentes resultados en especial en programación. Las soluciones generadas para los problemas de gran complejidad, en esta técnica, son bien estructuradas.

Hay que tener presente que en esta técnica al invocarse a sí misma las funciones, se generan variables y valores en los parámetros en cada una de las llamadas a cada función y estos se guardan en la pila junto con la dirección de la siguiente línea de código que debe ser ejecutada cuando finaliza la ejecución de la función invocada. Esta acción provoca que la pila crezca en cada una de las llamadas a cada función, por lo que, existe el riesgo de que se agote la memoria de la pila cuando las llamadas recursivas son realizadas en un número muy grande de veces, lo que puede causar la terminación abrupta del programa.

**Tipos de recursividad**

- **Directa:** un subprograma se llama a sí mismo una o más veces directamente.
- **Indirecta:** se definen una serie de subprogramas que se llaman unos a otros.

### 1.2.4.2 Procedimiento recursivo

Un algoritmo recursivo consta de una parte recursiva, otra iterativa o no recursiva y una condición de terminación. La parte recursiva y la condición de terminación siempre existen. En cambio, la parte no recursiva puede coincidir con la condición de terminación.

Algo muy importante para tener en cuenta cuando se usa la recursividad es que es necesario asegurarnos que llega un momento en que no hacemos más llamadas recursivas. Si no se cumple esta condición el programa no parará nunca.

La recursión no es más que la repetición (iteración) de una serie de acciones. Esta iteración se da hasta llegar a un valor de una variable. En la recursión, esta iteración se está desarrollando mediante la llamada de la función a sí misma con un parámetro que es la variable que determina el final de la recursión, en vez de ser un bucle dentro de una función normal. Así pues, en la iteración, es la guarda del bucle la que determina cuándo acabará la repetición; en la recursión es el parámetro.

**Reglas para que un algoritmo recursivo funcione correctamente**

**a)** Debe tener un caso base, por cada llamada recursiva que se haga dentro del algoritmo debe haber un valor para el cual el algoritmo finalice sin recursión.
**b)** Todos los posibles argumentos de las llamadas recursivas se reenvían tendiendo sus valores hacia un caso base.
**c)** La función es correcta, para valores distintos del caso base.

En este sentido la recursividad es una nueva forma de ver las acciones repetitivas permitiendo que un subprograma se llame a sí mismo para resolver una versión más pequeña del problema original.

### 1.2.4.3 Ejemplos de programas recursivos

En esta sección se explicarán diversos ejemplos de algoritmos que nos permiten encontrar la solución a distintos problemas de forma recursiva. Para entender cómo funcionan se muestra en cada uno de ellos el algoritmo iterativo, lo que nos permitirá establecer una comparación entre las dos técnicas.

**Ejemplo 1:**

Calcule el factorial de 4 utilizando recursividad.

- **Solución:**

La función del factorial de un número se describe de la siguiente forma:

$$f(x) = x \cdot (x-1) \cdot (x-2) \cdot \ \ldots \ \cdot 1$$

que está definida mediante una repetición de multiplicaciones empezando en el propio valor x y en las que el siguiente multiplicador disminuye hasta llegar al valor de 1. El algoritmo recursivo de esta función es el siguiente:

```
Función Recursiva
entero Fact (entero n)
{
   si (n=0) entonces
        retornar 1
    sino
         retornar (n*Fact (n-1))
}
```

El desarrollo de este algoritmo para el cálculo del factorial del número 4 es el siguiente:

**PASO 1**

**n = 4**

**si (n=0) entonces** **4 = 0 FALSO**

**PASO 2**

**Fact (4) = 4 * Fact (3)**

```
si (n=0) entonces
     retornar 1
  sino
    retornar (n*Factorial (n-1))
```

**PASO 3**

**3 = 0 FALSO**

**si (n=0) entonces**

**PASO 4**

**Fact (3) = 3 * Fact (2)**

si (n=0) entonces
retornar 1
sino
**retornar (n*Factorial (n-1))**

**PASO 5**

**2 = 0 FALSO**

**si (n=0) entonces**

**PASO 6**

**Fact (2) = 2 * Fact (1)**

si (n=0) entonces
retornar 1
sino
**retornar (n*Factorial (n-1))**

**PASO 7**

**1 = 0 FALSO**

**si (n=0) entonces**

**PASO 8**

**Fact (1) = 1 * Fact (0)**

si (n=0) entonces
retornar 1
sino
**retornar (n*Factorial (n-1))**

**PASO 9**

**0 = 0 CIERTO**

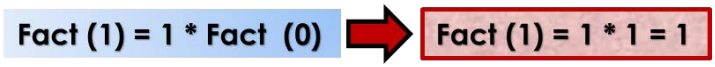
**si (n=0) entonces**

**PASO 10**

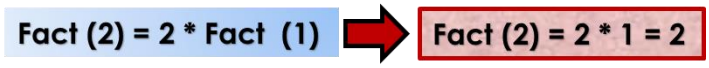
si (n=0) entonces
retornar 1

Fact (0) = 1

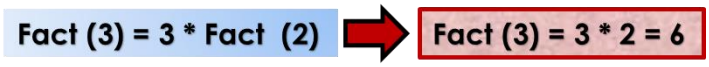
**PASO 11**



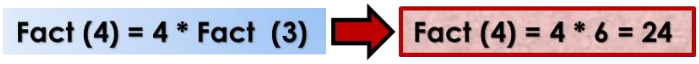

**PASO 12**




**PASO 13**




**PASO 14**




## Ejemplo 2:

Calcule la serie Fibonacci del número 6 utilizando recursividad.

- **Solución:**

La serie Fibonacci comienza con los términos 0 y 1, a partir del tercero, es la suma de los dos elementos anteriores:

```
0   +   1   =   1
1   +   1   =   2
2   +   1   =   3
3   +   2   =   5
5   +   3   =   8
...
```

Lo que corresponde a:

Fibonacci (0) = 0
Fibonacci (1) = 1
⋮
Fibonacci (n) = Fibonacci (n-1) + Fibonacci (n-2)

y la definición recursiva será:

Fibonacci (n) = n  si n = 0 o n = 1

Fibonacci (n) = Fibonacci (n-1) + Fibonacci (n-2)  si n >=2

A continuación, el programa recursivo que calcula la serie Fibonacci para el n-ésimo término de la serie:

```
entero Fib (entero n)
{
  si (n = 1) entonces
    retornar  (n, n-1)
  sino
    retornar (Fib (n-1) + Fib (n-2))
}
```

A continuación, se muestra cómo se ejecuta el programa para **n=6**.

**PASO 1**

si (n = 1) entonces

n = 6

6 = 1 FALSO

**PASO 2**

si (n = 1) entonces
    retornar (n , n-1)
sino
    retornar (Fib (n-1) + Fib (n-2))

Fib (6) = Fib (5) + Fib (4)

**PASO 3**

si (n = 1) entonces

5 = 1 FALSO

**PASO 4**

si (n = 1) entonces
    retornar (n , n-1)
sino
    retornar (Fib (n-1) + Fib (n-2))

Fib (5) = Fib (4) + Fib (3)

**PASO 5**

si (n = 1) entonces

4 = 1 FALSO

**PASO 6**

si (n = 1) entonces
    retornar (n , n-1)
sino
    retornar (Fib (n-1) + Fib (n-2))

Fib (4) = Fib (3) + Fib (2)

**PASO 7**

si (n = 1) entonces  3 = 1 FALSO

**PASO 8**

```
si (n = 1) entonces
      retornar  (n , n-1)
   sino
      retornar (Fib (n-1) + Fib (n-2))
```

Fib (3) = Fib (2) + Fib (1)

**PASO 9**

si (n = 1) entonces  2 = 1 FALSO

**PASO 10**

```
si (n = 1) entonces
      retornar  (n , n-1)
   sino
      retornar (Fib (n-1) + Fib (n-2))
```

Fib (2) = Fib (1) + Fib (0)

**PASO 11**

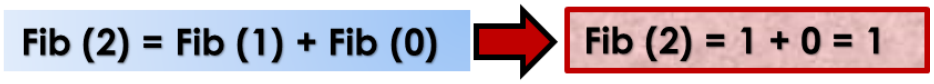
si (n = 1) entonces  1 = 1 CIERTO

**PASO 12**

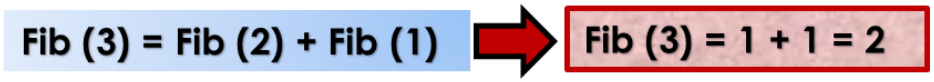
```
si (n = 1) entonces
      retornar  (n, n-1)
```

1, 0

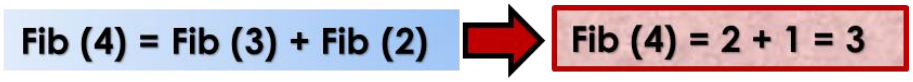
**PASO 13**



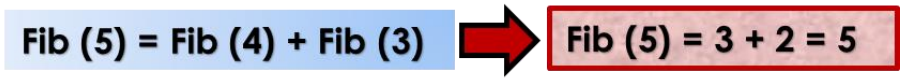

**PASO 14**




**PASO 15**




**PASO 16**




**PASO 17**


**Ejemplo 3:**

Realice una búsqueda binaria en el siguiente arreglo utilizando recursividad. El valor a buscar es: 46.

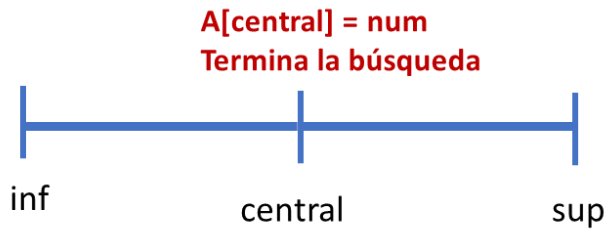
**A**

| 7 | 9 | 13 | 16 | 28 | 32 | 46 | 53 | 68 |
|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

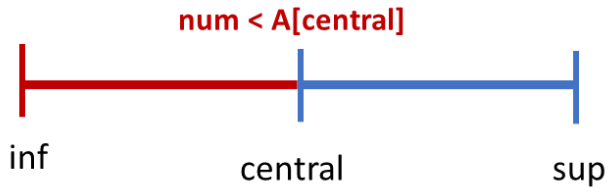
- **Solución:**

La búsqueda binaria es un método de localización de una clave especificada dentro de una lista o arreglo ordenado de n elementos que realiza una exploración de la lista hasta que se encuentra o se decide que no se encuentra en la lista. El algoritmo de búsqueda binaria se puede describir recursivamente aplicando el método divide y vencerás.
La búsqueda entre dos posiciones de un vector ordenado se puede realizar comparando el valor buscado con el elemento central de la siguiente forma:

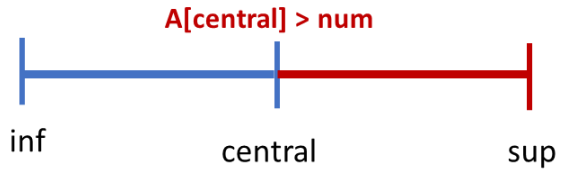
- **Si es igual, la búsqueda termina con éxito.**




- **Si es menor, la búsqueda debe continuar en el subvector izquierdo.**




- **Si es mayor, la búsqueda debe continuar en el subvector derecho.**




A continuación, el algoritmo de la función recursiva que realiza la búsqueda binaria en el arreglo y devuelve la posición en la que encuentra el número buscado si es el caso.

**Algoritmo**

```
entero busqueda(entero A[], entero inf, entero sup, entero num)
{
  entero central;

  si (inf > sup) entonces    //No encontrado
```

```
      retornar  (-1)
  sino
  {
    central = (inf + sup) / 2

    si (A[central] = num) entonces // Encontrado
      retornar (central)
    sino si (A[central] < num) entonces
          retornar busqueda(A, central+1, sup, num)
      else
          retornar busqueda(A, inf, central-1, num)
    }
}
```

A continuación, se muestra cómo se realiza la búsqueda del valor num=46.

**PASO 1**

**si (inf > sup) entonces**

num = 46
inf = 1
sup = 9
inf > sup
1 > 9 **FALSO**

**PASO 2**

si (inf > sup) entonces  //No encontrado
    retornar  (-1)
 sino
 {  **central = (inf + sup) / 2**

central = (1 + 9 ) / 2
central = 5

**A**

| 7 | 9 | 13 | 16 | 28 | 32 | 46 | 53 | 68 |
|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

**PASO 3**

sino
 {  central = (inf + sup) / 2
    **si (A[central] = num) entonces**

28 = 46 **FALSO**

**A**

| 7 | 9 | 13 | 16 | 28 | 32 | 46 | 53 | 68 |
|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

**PASO 4**

```
sino
 {    central = (inf + sup) / 2
      si (A[central] = num) entonces
            retornar (central)
      sino si (A[central] < num) entonces
```

28 < 46 **CIERTO**

**A**

| 7 | 9 | 13 | 16 | 28 | 32 | 46 | 53 | 68 |
|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

**PASO 5**

```
sino
 {    central = (inf + sup) / 2
      si (A[central] = num) entonces
            retornar (central)
      sino si (A[central] < num) entonces
                  retornar busqueda(A, central+1, sup, num)
            else
                  retornar busqueda(A, inf, central-1, num)
  }  }
```

busqueda(A, 5+1, 9, 46)
busqueda(A, 6, 9, 46)

**PASO 6**

**si (inf > sup) entonces**

num = 46
inf = 6
sup = 9
inf > sup
6 > 9 **FALSO**

**PASO 7**

```
si (inf > sup) entonces     //No encontrado
     retornar  (-1)
 sino
 {    central = (inf + sup) / 2
```

central = (6 + 9 ) / 2
central = 7

**A**

| 7 | 9 | 13 | 16 | 28 | 32 | 46 | 53 | 68 |
|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

**PASO 8**

```
sino
 {    central = (inf + sup) / 2
      si (A[central] = num) entonces
          retornar (central)
```

46 = 46 **CIERTO**

**A**

| 7 | 9 | 13 | 16 | 28 | 32 | 46 | 53 | 68 |
|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

# Capítulo II: Estructuras dinámicas de datos

## *2.1 Estructura de datos dinámicas lineales*

Las listas enlazadas son estructuras dinámicas y lineales que son utilizadas para almacenar datos que continuamente están cambiando. Estas permiten almacenar información en posiciones de memoria que no sean adyacentes. Una de las ventajas de trabajar con las listas es que se pueden obtener posiciones de memoria cuando se requieran y se pueden liberar cuando ya no sean necesarias. Esto nos permite optimizar el espacio de la memoria utilizada.

En esta sección del Capítulo II se presentan las principales definiciones y conceptos de las listas enlazadas. Se muestra la clasificación de las listas y las operaciones que se pueden realizar en cada una de ellas. Finalmente, se explican ejemplos de cada uno de estos tipos de listas.

### 2.1.1 Listas enlazadas

#### 2.1.1.1 Definición y conceptos

En las secciones anteriores de este documento se explicaron las estructuras de datos denominadas estructuras estáticas, arreglos, pilas y colas. A este tipo de estructura, se les asigna un espacio de memoria durante la compilación y éste permanece constante durante la ejecución del programa. De allí el nombre de estructuras estáticas.

En esta sección se presenta un tipo de estructura lineal y dinámica de datos, a la cual se le conoce como “Lista”. Es lineal porque a cada elemento le sigue sólo otro elemento, y dinámica porque la memoria se puede manejar de manera flexible, no es necesario reservar espacio con antelación.

**Características de la lista**

- Es una colección, de elementos homogéneos, o sea que todos los elementos son del mismo tipo.

- Hay una relación lineal entre los elementos. Para cada elemento existe un anterior y un siguiente, excepto para el primero, que no tiene anterior, y para el último, que no tiene siguiente.

- El orden de los elementos en la lista afecta a su función de acceso, por ejemplo, si la lista está ordenada de más pequeña a mayor el sucesor de cualquier elemento de la lista será mayor o igual que ese elemento.

- Se puede acceder y eliminar cualquier elemento.

- Se pueden insertar elementos en cualquier posición.

Además de ordenar las listas por su valor, hay otras formas de ordenarlas: por el tiempo en el que los elementos están en ellas, por prioridad o por importancia. La forma en que está ordenada la lista afectará a la función de acceso de ella. En sección del Capítulo II, nos referiremos a una lista ordenada por el valor de sus elementos.

En la **Figura 12** se muestran los tipos de representación de una lista:


*Figura 12. Tipos de representación de una lista. Elaboración propia.*

- **Lista secuencial:** Una representación de lista en la que el sucesor de cada elemento de la lista está implícito por la posición del elemento: los elementos se colocan en posiciones secuenciales en la estructura.

- **Lista enlazada:** Una representación de lista en la que el orden de los elementos está determinado por un campo enlace explícito en cada elemento, en vez de por su posición secuencial.

#### 2.1.1.2 Clasificación

Las listas se clasifican en: simples, doblemente enlazadas y enlazadas circulares. Cada una de ellas se describirán a continuación.

**a) Listas simples**

Una lista enlazada puede describirse como una colección de elementos, o nodos, cada uno conteniendo datos y un enlace, o puntero al siguiente nodo de la lista. Un puntero "externo" (uno que no está conectado a cualquier nodo) nos dice donde comienza la lista. En la **Figura 13** se muestra un ejemplo de una lista enlazada.

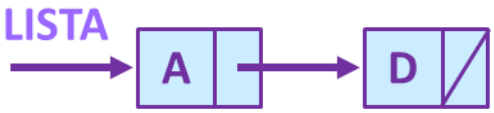



*Figura 13. Ejemplo de una lista enlazada. Elaboración propia.*

Cada **nodo** de una lista enlazada debe contener, al menos **dos campos** (como se puede observar en la **Figura 14**):

1. El primero es el campo dato, el cual contiene los datos del usuario, nos referiremos a este campo como el campo ***info***. El campo info puede contener un entero, un carácter, una cadena o un registro que contiene muchos otros campos.

2. El segundo campo contiene el puntero o enlace al siguiente nodo de la lista. Nos referiremos a este campo como el ***siguiente***.

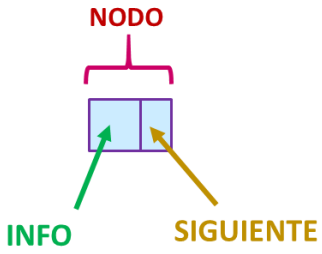



*Figura 14. Campos del nodo de una lista enlazada. Elaboración propia.*

### Notación de lista

La notación de una lista se describe en la Figura 15, mostrada a continuación:

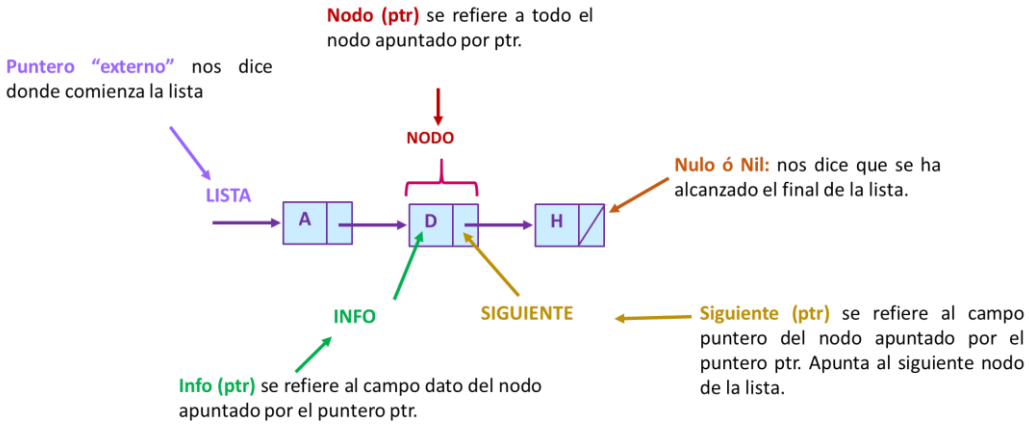



*Figura 15. Notación de una lista. Elaboración propia.*

## Operaciones sobre una lista enlazada

### Algoritmos

```
Procedimiento CrearLista
Inicio
/* Inicializa Lista al estado vacío */
        Lista = Nulo
Fin
```

El procedimiento CrearLista inicializa el puntero externo Lista a nulo lo que luego nos permitirá indicar en dónde inicia la nueva lista enlazada.

La función ListaVacía verifica si existe algún nodo en la lista.

```
Función ListaVacía (Booleano)
Inicio
/* Determina si Lista está vacía */
Si (Lista = Nulo)
    Entonces    ListaVacía = Cierto
    Sino        ListaVacía = Falso
Fin Si
Fin
```

- **Insertar un nuevo nodo**

```
Procedimiento Insertar NuevoNodo
Inicio      /* Asigna un nuevo nodo y pone nuevovalor en él */
Nuevo (Nuevonodo)
Nuevonodo.Info = Nuevovalor
Nuevonodo.Siguiente = Nulo
/* Inserta el nuevo nodo en la lista */
Si (Listavacía (Lista) = "Cierto")          /* Estamos insertando en una lista vacía */
Entonces  Lista = Nuevonodo
Sino      Si  (Nuevovalor < Lista. Info)          /* Estamos insertando en una lista existente */
                    Entonces          Nuevonodo.siguiente = Lista   /* Inserta antes del primer nodo */
                              Lista = Nuevonodo
                    Sino ptr = Lista      /* Inserta por el medio o al final de la lista */
                         LugarEncontrado = Falso
                         Mientras ((ptr.siguiente ≠ Nulo) y (LugarEncontrado = Falso)) Hacer
                                   Si (Nuevovalor >= ptr.siguiente.info)
                                        Entonces  ptr = ptr.siguiente     /* Se inserta en el medio o en el final */
                                        Sino      LugarEncontrado = Cierto
                                   Fin Si
                         Fin Mientras
                              Nuevonodo.siguiente = ptr.siguiente
                              Ptr.siguiente = Nuevonodo
          Fin Si
Fin Si
Fin
```

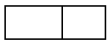
### Partes del procedimiento Insertar NuevoNodo

- Crear la estructura del nuevo nodo
- Ubicar la posición en donde se insertará el nuevo nodo
- Actualizar los enlaces del nodo o los nodos

Para la mejor comprensión de este algoritmo, se explica cada una de estas partes:

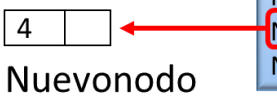
**Crear la estructura del nuevo nodo:** es la parte del algoritmo encargada de crear el nuevo nodo que va a ser insertado en la lista.

- **Crear el nuevo nodo**



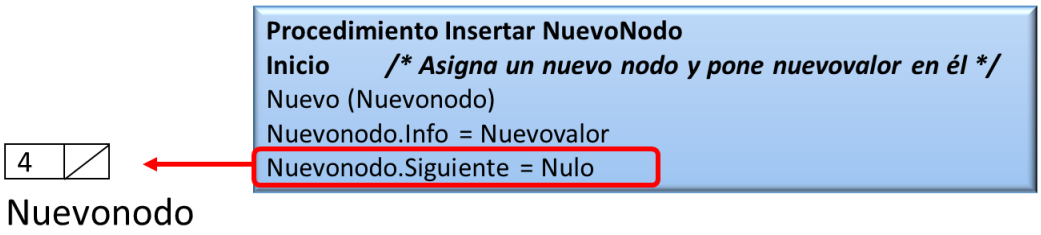

```
Procedimiento Insertar NuevoNodo
Inicio      /* Asigna un nuevo nodo y pone nuevovalor en él */
Nuevo (Nuevonodo)
Nuevonodo.Info = Nuevovalor
Nuevonodo.Siguiente = Nulo
```

- **Coloca el valor en la parte info del nodo creado**




```
Procedimiento Insertar NuevoNodo
Inicio      /* Asigna un nuevo nodo y pone nuevovalor en él */
Nuevo (Nuevonodo)
Nuevonodo.Info = Nuevovalor
Nuevonodo.Siguiente = Nulo
```

- **Coloca nulo en la parte siguiente del nuevo nodo**




**Ubicar la posición en donde se insertará el nuevo nodo:** verifica la posición en donde se colocará el nuevo nodo.

- **Inserta el nuevo nodo al inicio en una lista nueva**

```
Procedimiento Insertar NuevoNodo
Inicio      /* Asigna un nuevo nodo y pone nuevovalor en él */
Nuevo (Nuevonodo)
Nuevonodo.Info = Nuevovalor
Nuevonodo.Siguiente = Nulo
/* Inserta el nuevo nodo en la lista */
Si (Listavacía (Lista) = "Cierto")          /* Estamos insertando en una lista vacía */
```

- **Inserta el nuevo nodo antes del primer nodo en una lista ya existente**

```
Procedimiento Insertar NuevoNodo
Inicio      /* Asigna un nuevo nodo y pone nuevovalor en él */
Nuevo (Nuevonodo)
Nuevonodo.Info = Nuevovalor
Nuevonodo.Siguiente = Nulo
/* Inserta el nuevo nodo en la lista */
Si (Listavacía (Lista) = "Cierto")          /* Estamos insertando en una lista vacía */
Entonces  Lista = Nuevonodo
Sino      Si (Nuevovalor < Lista. Info)          /* Estamos insertando en una lista existente */
                  Entonces          Nuevonodo.siguiente = Lista   /* Inserta antes del primer nodo */
                                    Lista = Nuevonodo
                  Sino ptr = Lista      /* Inserta por el medio o al final de la lista */
                       LugarEncontrado = Falso
```

- **Inserta el nuevo nodo en medio o al final de una lista ya existente**

```
                  Sino ptr = Lista      /* Inserta por el medio o al final de la lista */
                       LugarEncontrado = Falso
                       Mientras ((ptr.siguiente ≠ Nulo) y (LugarEncontrado = Falso)) Hacer
                                 Si (Nuevovalor >= ptr.siguiente.info)
                                          Entonces  ptr = ptr.siguiente    /* Se inserta en el medio o en el final */
                                          Sino      LugarEncontrado = Cierto
                                 Fin Si
                       Fin Mientras
                            Nuevonodo.siguiente = ptr.siguiente
                            Ptr.siguiente = Nuevonodo
            Fin Si
Fin Si
Fin
```

**Actualizar los enlaces del nodo o los nodos:** se colocan los enlaces en los nodos que han sufrido el cambio para que se mantenga la lista enlazada.

```
                                    Fin  Si
                              Fin Mientras
                                    Nuevonodo.siguiente  = ptr.siguiente
                                    ptr.siguiente  = Nuevonodo
                        Fin Si
Fin Si
Fin
```

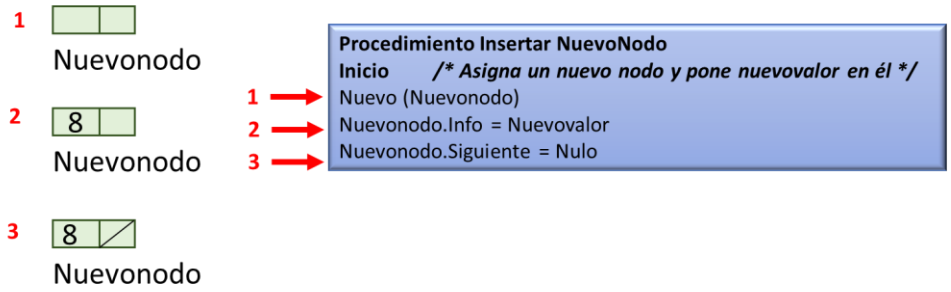
## Ejemplo:

Insertar el valor de **8** en la siguiente lista enlazada.


- **Solución:**

**PASO 1**

Crear la estructura del nuevo nodo



## PASO 2

Ubicar la posición en donde se insertará el nuevo nodo

```
Procedimiento Insertar NuevoNodo
Inicio        /* Asigna un nuevo nodo y pone nuevovalor en él */
Nuevo (Nuevonodo)
Nuevonodo.Info = Nuevovalor
Nuevonodo.Siguiente = Nulo
/* Inserta el nuevo nodo en la lista */
1 Si (Listavacía (Lista) = "Cierto")        /* Estamos insertando en una lista vacía */
```

```
Función ListaVacía (Booleano)
Inicio
/* Determina si Lista está vacía */
2 Si (Lista = Nulo)  → FALSO
        Entonces    ListaVacía = Cierto
3       Sino        ListaVacía = Falso
Fin Si
Fin
```

La lista no está vacía

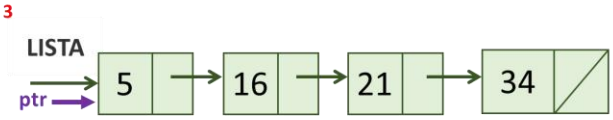
## PASO 3

Ubicar la posición en donde se insertará el nuevo nodo

2 8 < 5 FALSO

```
/* Inserta el nuevo nodo en la lista */
1 Si (Listavacía (Lista) = "Cierto")  → FALSO /* Estamos insertando en una lista vacía */
  Entonces  Lista = Nuevonodo
2 Sino  Si (Nuevovalor < Lista. Info)  → FALSO /* Estamos insertando en una lista existente */
              Entonces  Nuevonodo.siguiente = Lista      /* Inserta antes del primer nodo */
                        Lista = Nuevonodo
3             Sino ptr = Lista          /* Inserta por el medio o al final de la lista */
                   LugarEncontrado = Falso
```

3




LugarEncontrado = Falso

## PASO 4

1

Mientras ((ptr.siguiente ≠ Nulo) y (LugarEncontrado = Falso))

CIERTO  CIERTO

2

8 >= 16 FALSO

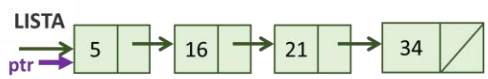
3

LugarEncontrado = Cierto

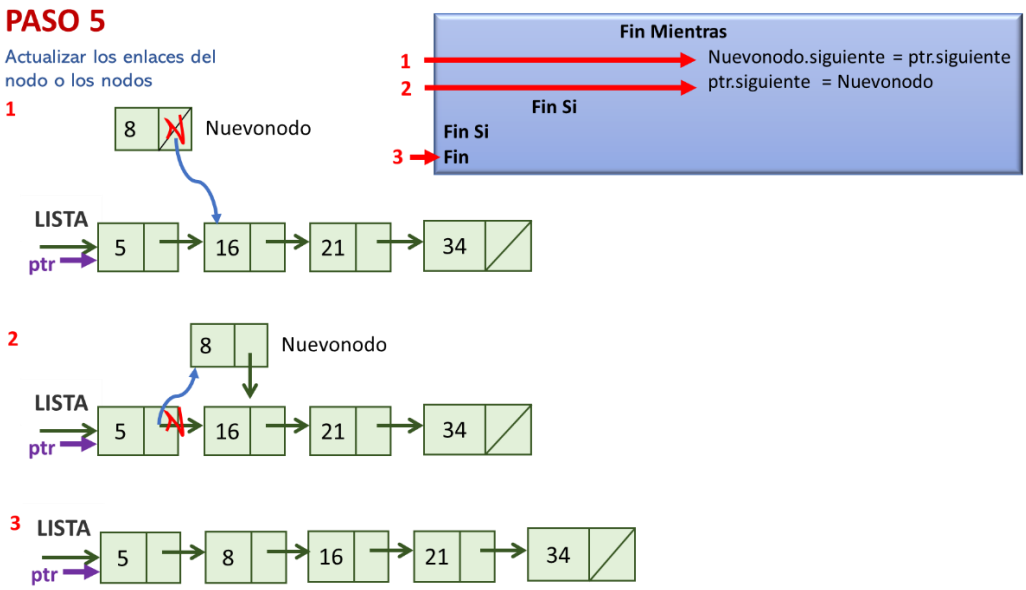
4

Mientras ((ptr.siguiente ≠ Nulo) y (LugarEncontrado = Falso))

CIERTO  FALSO




```
Sino ptr = Lista          /* Inserta por el medio o al final de la lista */
     LugarEncontrado = Falso
1    Mientras ((ptr.siguiente ≠ Nulo) y (LugarEncontrado = Falso)) Hacer
2       Si (Nuevovalor >= ptr.siguiente.info)  → FALSO
           Entonces  ptr = ptr.siguiente     /* Se inserta en el medio o en el final */
3          Sino      LugarEncontrado = Cierto
        Fin Si
     Fin Mientras
```




### Suprimir un nodo

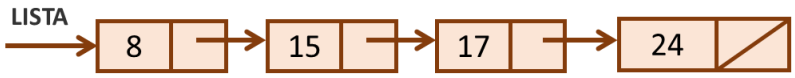
El procedimiento SuprimirNodo busca el valor a suprimir en la lista, si lo encuentra lo elimina, de otra forma, no hace nada. El algoritmo lo podríamos dividir en tres grandes acciones a realizar:

1. Inicializar los punteros que van a buscar en la lista el valor a eliminar.
2. Buscar el nodo que contiene el valor a eliminar.
3. Actualizar los enlaces para mantener la lista enlazada y elimina el nodo que contiene el valor a suprimir.

```
Procedimiento  SuprimirNodo
Inicio     /* Inicializa los punteros para buscar */
actual = Lista
anterior = Nulo
encontrado = Cierto
/* Busca el nodo que contiene el valor a suprimir */
Mientras (actual.info ≠ supvalor y encontrado = Cierto) Hacer
     Si (actual.siguiente ≠ nulo)
          Entonces   anterior = actual
                     actual = actual.siguiente
          Sino       encontrado = Falso
     Fin si
Fin Mientras
 Si (encontrado = Cierto) entonces    /* Comprueba si supvalor era el primer nodo */
          Si  (anterior = Nulo)              /* Es el primer nodo */
                Entonces  Lista = Lista.siguiente
                          Liberarnodo (actual)

                Sino      /* Quita el nodo en el medio o el final */
                          anterior.siguiente = actual.siguiente
                          Liberarnodo (actual)
          Fin si
Fin si
Fin
```

## Ejemplo:

Suprimir el valor de **15** en la siguiente lista enlazada.

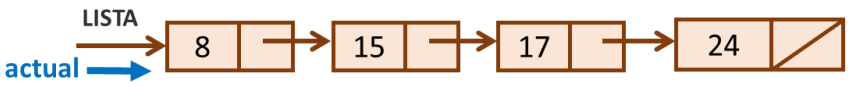



- **Solución:**

## PASO 1

**Inicializar los punteros que van a buscar en la lista el valor a eliminar**

1



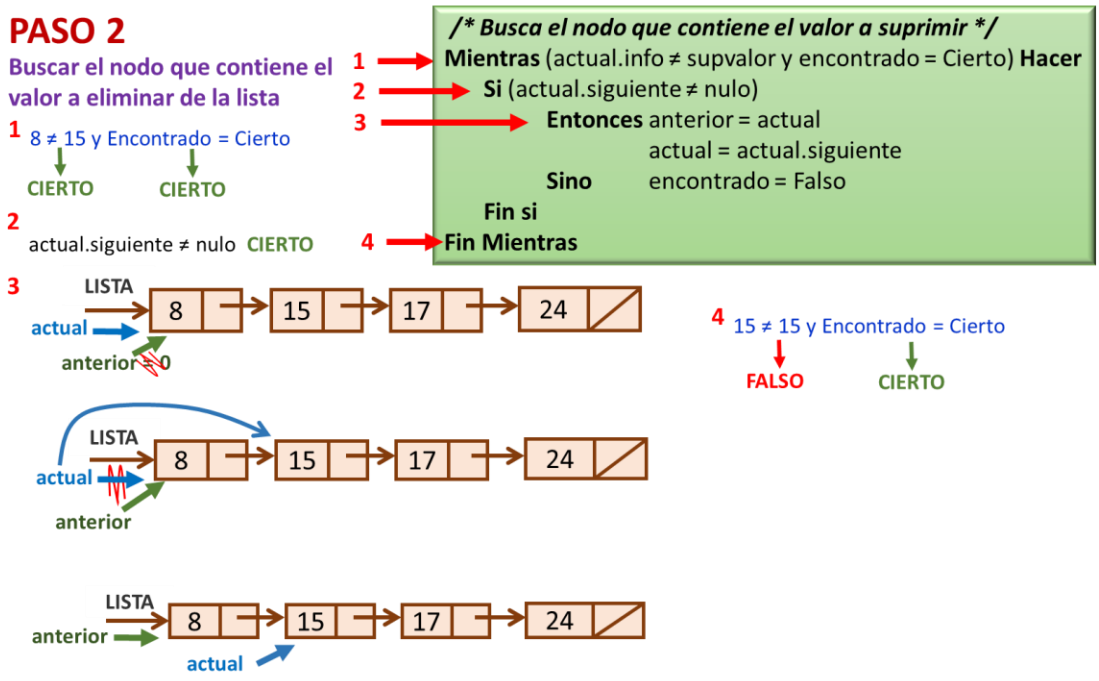

```
Procedimiento  SuprimirNodo
Inicio     /* Inicializa los punteros para buscar */
1 → actual = Lista
2 → anterior = Nulo
3 → encontrado = Cierto
```

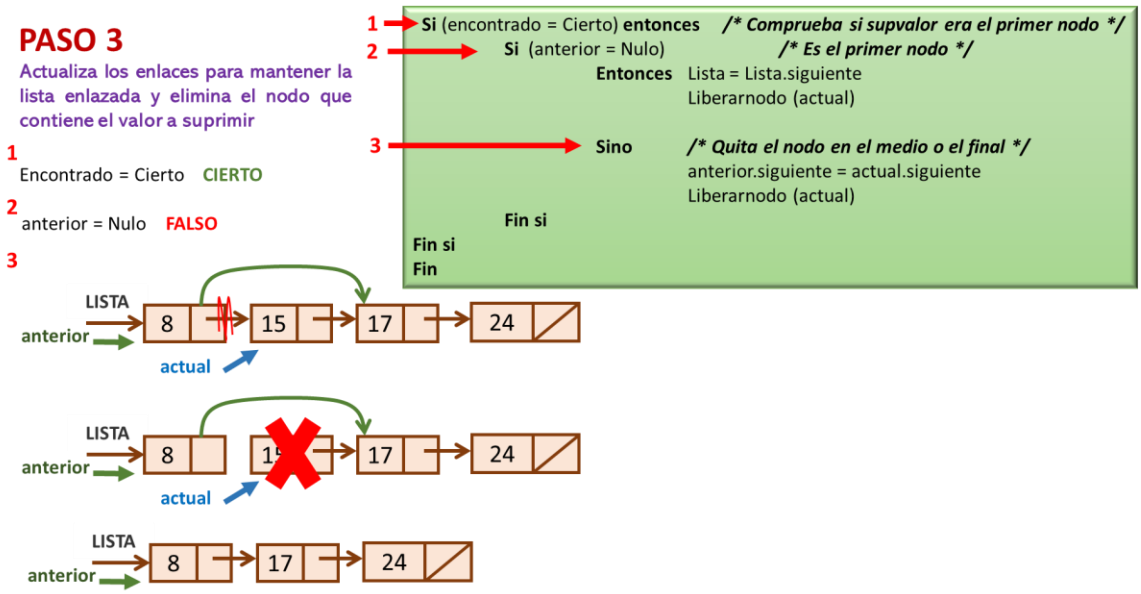
2

Anterior = 0

3

Encontrado = Cierto




## b) Listas doblemente enlazadas

Una lista doblemente enlazada (**Figura 16**) es una lista lineal en la que cada nodo tiene dos enlaces, uno al nodo siguiente, y otro al anterior. Pueden recorrerse en ambos sentidos a partir de cualquier nodo hasta que se llega a uno de los extremos de la lista.

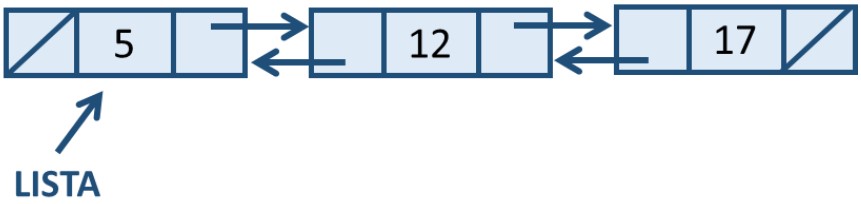



*Figura 16. Ejemplo de una lista doblemente enlazada. Elaboración propia.*

## Operaciones sobre una lista enlazada

### Algoritmos

**Procedimiento CrearLista**
Inicio
/* Inicializa Lista al estado vacío */
Lista = Nulo
Fin

**Función ListaVacía (Booleano)**
Inicio
/* Determina si Lista está vacía */
Si (Lista = Nulo)
Entonces ListaVacía = Cierto
Sino ListaVacía = Falso
Fin Si
Fin

**Procedimiento Insertar NuevoNodo**
Inicio
/* Asigna un nuevo nodo y pone nuevovalor en él */
Nuevo (Nuevonodo)
Nuevonodo.Info = Nuevovalor
Nuevonodo.Siguiente = Nulo
Nuevonodo.Anterior = Nulo

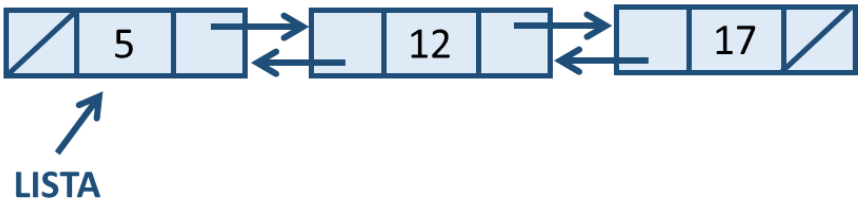
/* Inserta el nuevo nodo en la lista */

Si (Listavacía (Lista) = “Cierto”) /* Estamos insertando en una lista vacía */
Entonces Lista = Nuevonodo

```
        Sino                    /* Estamos insertando en una lista existente */
          Si   (Nuevovalor < Lista. Info)
               Entonces   /* Inserta antes del primer nodo */
                          Nuevonodo.siguiente  = Lista
                          Lista.anterior = Nuevonodo
                          Lista  = Nuevonodo
               Sino  /* Inserta por el medio o al final de la lista */
                     ptr = Lista
                     LugarEncontrado = Falso
                     Mientras ((ptr.siguiente ≠ Nulo) y (LugarEncontrado = Falso))
Hacer
                                Si (Nuevovalor >= ptr.siguiente.info)
                                     Entonces      /* Se inserta en el medio o en el final */
                                           ptr  = ptr.siguiente
                                     Sino   LugarEncontrado  = Cierto
                                Fin  Si
                           Fin Mientras
                           Nuevonodo.siguiente = ptr.siguiente
                           Nuevonodo.anterior = ptr
                           Si (ptr.siguiente ≠ Nulo)
                                Entonces   ptr.siguiente.anterior  = Nuevonodo
                           Fin si
                           ptr.siguiente = Nuevonodo
               Fin Si
Fin Si
Fin
```

**Ejemplo:**

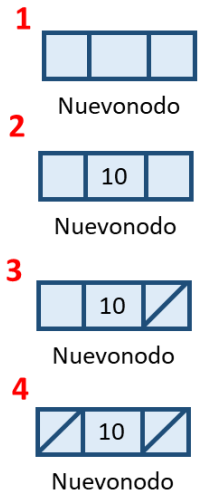
Insertar el valor de **10** en la siguiente lista doblemente enlazada.




- **Solución:**

# PASO 1

Crear la estructura del nuevo nodo




```
Procedimiento Insertar NuevoNodo
Inicio   /* Asigna un nuevo nodo y pone nuevovalor en él */
1  Nuevo (Nuevonodo)
2  Nuevonodo.Info = Nuevovalor
3  Nuevonodo.Siguiente = Nulo
4  Nuevonodo.Anterior = Nulo
```

# PASO 2

Ubicar la posición en donde se insertará el nuevo nodo

```
1  /* Inserta el nuevo nodo en la lista */
    Si (Listavacía (Lista) = "Cierto")        /* Estamos insertando en una lista vacía */
   Entonces  Lista = Nuevonodo
   Sino                                       /* Estamos insertando en una lista existente */
          Si (Nuevovalor < Lista. Info)
                Entonces  /* Inserta antes del primer nodo */
                                  Nuevonodo.siguiente = Lista
                                  Lista.anterior = Nuevonodo
                                  Lista = Nuevonodo
                Sino  /* Inserta por el medio o al final de la lista */
```

```
Función ListaVacía (Booleano)
Inicio
/* Determina si Lista está vacía */
2  Si (Lista = Nulo)  → FALSO
        Entonces        ListaVacía = Cierto
3       Sino            ListaVacía = Falso
Fin Si
Fin
```

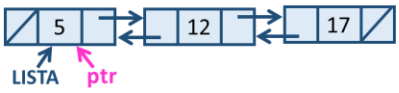
La lista no está vacía

## PASO 3

Ubicar la posición en donde se insertará el nuevo nodo

1 10 < 5 **FALSO**

2




LugarEncontrado = Falso

3 ptr.siguiente ≠ Nulo y LugarEncontrado = Falso

CIERTO CIERTO

4 10 >= 12 **FALSO**

5 LugarEncontrado = Cierto

6 ptr.siguiente ≠ Nulo y LugarEncontrado = Cierto

CIERTO **FALSO**

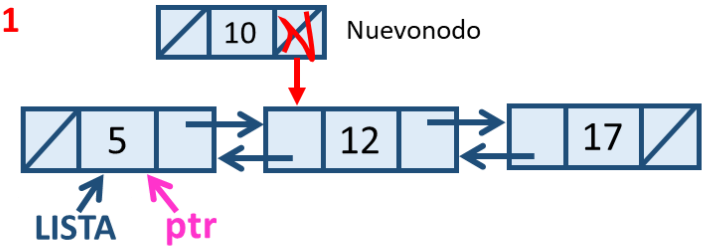
```
/* Inserta el nuevo nodo en la lista */
Si (Listavacía (Lista) = "Cierto")        /* Estamos insertando en una lista vacía */
Entonces  Lista = Nuevonodo
Sino                                      /* Estamos insertando en una lista existente */
1   Si (Nuevovalor < Lista. Info)  → FALSO
        Entonces  /* Inserta antes del primer nodo */
            Nuevonodo.siguiente = Lista
            Lista.anterior = Nuevonodo
            Lista = Nuevonodo
2       Sino /* Inserta por el medio o al final de la lista */
            ptr = Lista
            LugarEncontrado = Falso
3           Mientras ((ptr.siguiente ≠ Nulo) y (LugarEncontrado = Falso)) Hacer
4               Si (Nuevovalor >= ptr.siguiente.info)
                    Entonces    /* Se inserta en el medio o en el final */
                        ptr = ptr.siguiente
5               Sino
                        LugarEncontrado = Cierto
                Fin Si
6           Fin Mientras
```

## PASO 4

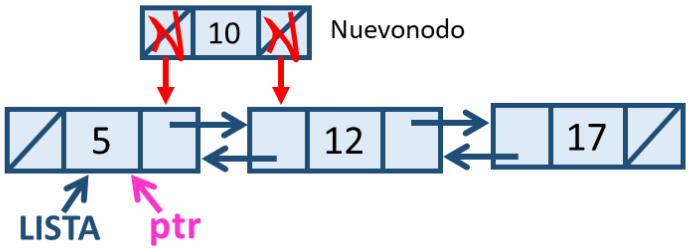
Actualizar los enlaces del nuevo nodo

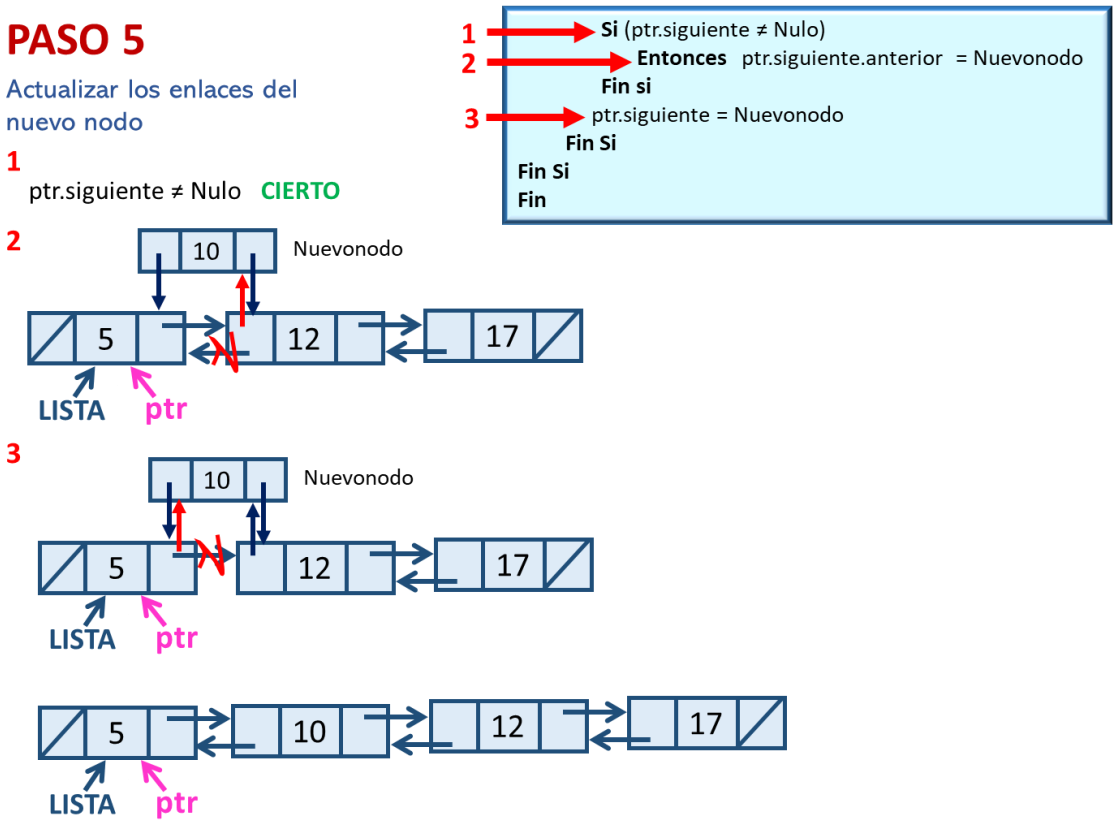
```
1  Nuevonodo.siguiente = ptr.siguiente
2  Nuevonodo.anterior = ptr
```

1




2






El procedimiento SuprimirNodo busca el valor a suprimir en la lista, si lo encuentra lo elimina de otra forma no hace nada.

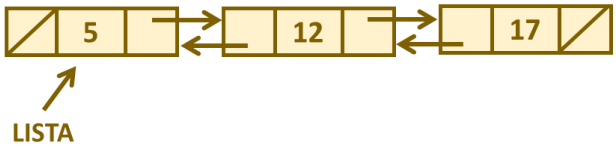
**Procedimiento SuprimirNodo**

```
Inicio
/* Inicializa los punteros para buscar */
actual = Lista
nodoanterior = Nulo
encontrado = Cierto

/* Busca el nodo que contiene el valor a suprimir */
Mientras (actual.info ≠ supvalor y encontrado = Cierto) Hacer
        Si (actual.siguiente ≠ nulo)
                Entonces    nodoanterior = actual
                            actual = actual.siguiente
                Sino    encontrado = Falso
        Fin si
Fin Mientras

Si (encontrado = Cierto) entonces
```

```
/* Comprueba si supvalor era el primer nodo */
        Si  (nodoanterior = Nulo)            /* Es el primer nodo */
                Entonces      Lista = Lista.siguiente
                              Liberarnodo (actual)

                Sino          /* Quita el nodo en el medio o el final */
                        nodoanterior.siguiente = actual.siguiente
                        Si (actual.siguiente ≠ nulo)
                                Entonces actual.siguiente.anterior = nodoanterior
                        Fin si
                        Liberarnodo (actual)
        Fin si
Fin si
Fin
```

**Ejemplo:**

Suprimir el valor de **12** en la siguiente lista doblemente enlazada.



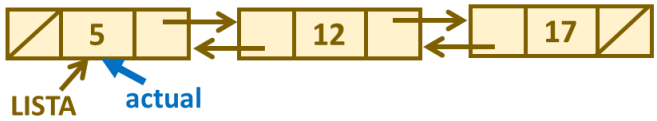

- **Solución:**

**PASO 1**

**Inicializar los punteros que van a buscar en la lista el valor a eliminar**

```
Procedimiento  SuprimirNodo
Inicio
/* Inicializa los punteros para buscar */
1  actual = Lista
2  nodoanterior = Nulo
3  encontrado = Cierto
```

1




2

nodoanterior = 0

3

encontrado = Cierto

## PASO 2

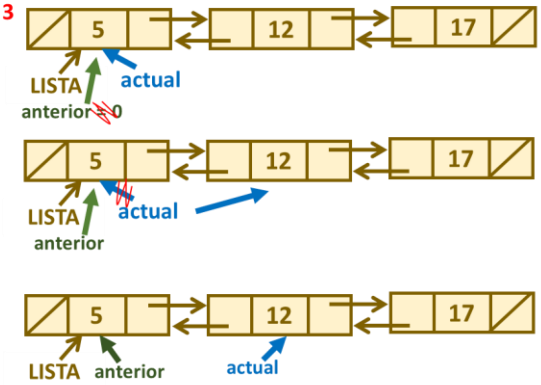
**Buscar el nodo que contiene el valor a eliminar de la lista**

1 actual.info ≠ supvalor y encontrado = Cierto

CIERTO CIERTO

2 actual.siguiente ≠ nulo CIERTO

3




```
1 /* Busca el nodo que contiene el valor a suprimir */
1 Mientras (actual.info ≠ supvalor y encontrado = Cierto) Hacer
2 Si (actual.siguiente ≠ nulo)
3       Entonces   nodoanterior = actual
                   actual = actual.siguiente
          Sino     encontrado = Falso
  Fin si
4 Fin Mientras
```

4 12 ≠ 12 y Encontrado = Cierto

FALSO CIERTO

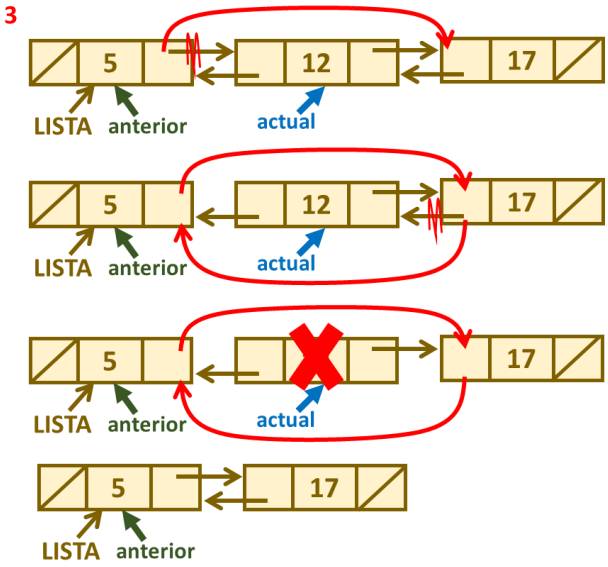
## PASO 3

**Actualiza los enlaces para mantener la lista enlazada y elimina el nodo que contiene el valor a suprimir**

1 encontrado = Cierto CIERTO

2 nodoanterior = Nulo FALSO

3




```
1 Si (encontrado = Cierto) entonces
  /* Comprueba si supvalor era el primer nodo */
2       Si (nodoanterior = Nulo)   /* Es el primer nodo */
            Entonces   Lista = Lista.siguiente
                       Liberarnodo (actual)
3       Sino   /* Quita el nodo en el medio o el final */
                nodoanterior.siguiente = actual.siguiente
                actual.siguiente.anterior = nodoanterior
                Liberarnodo (actual)
        Fin si
Fin si
Fin
```

### c) Listas enlazadas circulares

Una lista enlazada circular (**Figura 17**) es una lista lineal en la que el último nodo se enlaza con el primero, de esta forma, cada nodo siempre tiene uno anterior y uno siguiente. El enlace en este tipo de lista es similar al de las listas simplemente enlazadas, con excepción del último nodo que se enlaza con el primero.

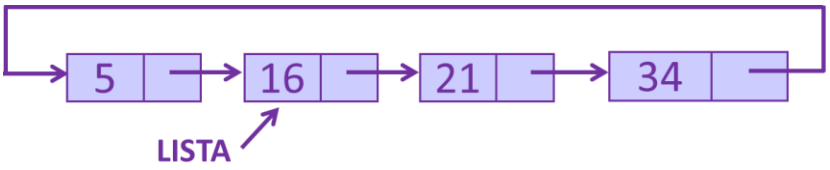



*Figura 17. Ejemplo de una lista enlazada circular. Elaboración propia.*

Algunas ventajas de estas listas son: se pueden recorrer la lista desde cualquier punto, no existe ningún elemento que apunte a nulo. Una desventaja de estas listas es que pueden llegar a crear recorridos en bucles infinitos, por ejemplo, en un proceso de búsqueda, no es tan sencillo dar por terminada la búsqueda cuando el elemento buscado no existe. Para solucionar este inconveniente, se suele resaltar algún nodo en particular que nos ayude a dar por terminada la búsqueda. Dicho nodo puede variar puede variar durante la ejecución del programa y cualquier nodo puede ser seleccionado.

Para evitar la única excepción posible, la de que la lista esté vacía, en algunas listas circulares se añade un nodo especial de cabecera. Además, este nodo cabecera ayuda en los casos de búsqueda.

#### Listas enlazadas circulares con nodos cabeza

Una lista enlazada con nodos cabeza también llamada lista enlazada con cabecera (**Figura 18**) es una lista enlazada que contiene un nodo especial, llamado nodo cabecera, al principio de la lista el cual actúa como limitador de lugar y el último nodo apunta hacia el nodo cabecera.

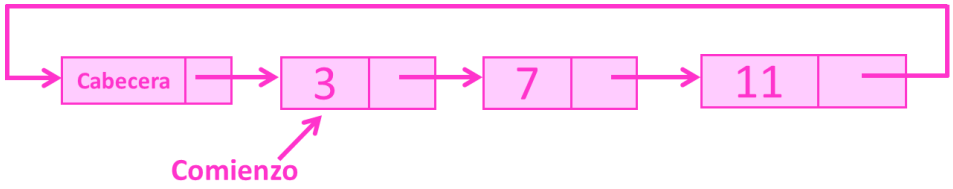


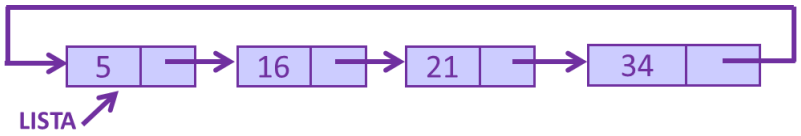

*Figura 18. Ejemplo de una lista enlazada circular con nodo cabeza. Elaboración propia.*

El nodo cabecera es un nodo que siempre estará al comienzo de una lista y es usado para simplificar el procesamiento de la lista y/o para contener información sobre la lista. El primer nodo de una lista con cabecera es siempre el siguiente al nodo cabecera y su posición será COMIENZO.

Las operaciones en listas enlazadas circulares son similares a las operaciones en listas lineales.

**Ejemplo 1:**

Insertar el valor de **3** en la siguiente lista enlazada circular.




- **Solución:**

**PASO 1**

Crear la estructura del nuevo nodo

1 Nuevonodo

2 3 Nuevonodo

3 3 Nuevonodo

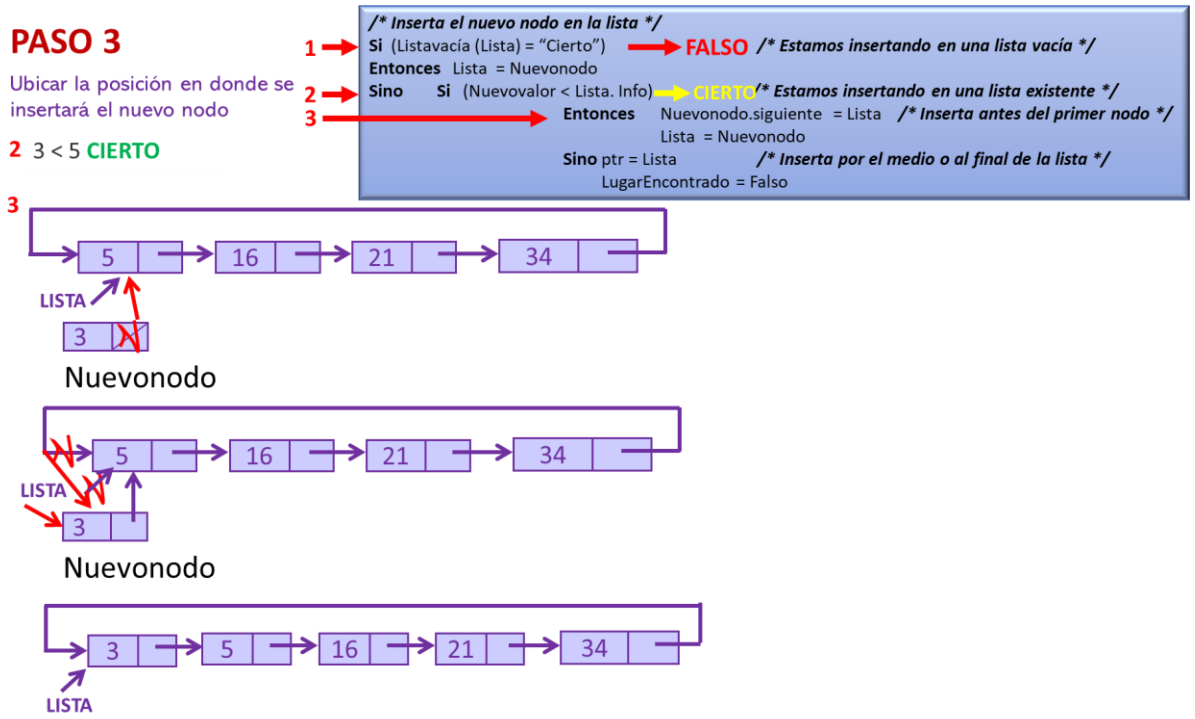
```
Procedimiento Insertar NuevoNodo
Inicio      /* Asigna un nuevo nodo y pone nuevovalor en él */
1 Nuevo (Nuevonodo)
2 Nuevonodo.Info = Nuevovalor
3 Nuevonodo.Siguiente = Nulo
```

**PASO 2**

Ubicar la posición en donde se insertará el nuevo nodo

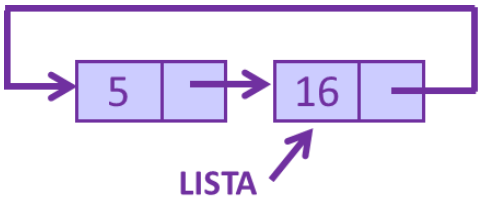
```
Procedimiento Insertar NuevoNodo
Inicio      /* Asigna un nuevo nodo y pone nuevovalor en él */
Nuevo (Nuevonodo)
Nuevonodo.Info = Nuevovalor
Nuevonodo.Siguiente = Nulo
/* Inserta el nuevo nodo en la lista */
1 Si (Listavacía (Lista) = "Cierto")        /* Estamos insertando en una lista vacía */
```

```
Función ListaVacía (Booleano)
Inicio
/* Determina si Lista está vacía */
2 Si (Lista = Nulo)  → FALSO
        Entonces        ListaVacía = Cierto
La lista no está vacía
3       Sino            ListaVacía = Falso
Fin Si
Fin
```




## Ejemplo 2:

Suprimir el valor de **16** en la siguiente lista enlazada circular.



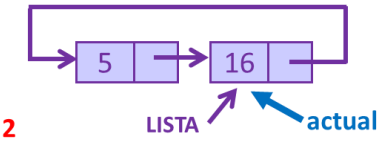

- **Solución:**

## PASO 1

**Inicializar los punteros que van a buscar en la lista el valor a eliminar**

```
Procedimiento  SuprimirNodo
Inicio     /* Inicializa los punteros para buscar */
1  actual = Lista
2  anterior = Nulo
3  encontrado = Cierto
```

1




2

anterior = 0

3

encontrado = Cierto

## PASO 2

**Buscar el nodo que contiene el valor a eliminar de la lista**

1 16 ≠ 16 y encontrado = Cierto

FALSO  CIERTO

```
1  /* Busca el nodo que contiene el valor a suprimir */
   Mientras (actual.info ≠ supvalor y encontrado = Cierto) Hacer
       Si (actual.siguiente ≠ nulo)
              Entonces anterior = actual
                       actual = actual.siguiente
              Sino     encontrado = Falso
       Fin si
   Fin Mientras
```

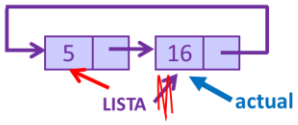
## PASO 3

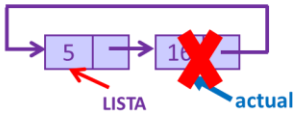
**Actualiza los enlaces para mantener la lista enlazada y elimina el nodo que contiene el valor a suprimir**

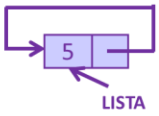
1 encontrado = Cierto CIERTO

2 anterior = Nulo CIERTO

3




```
1  Si (encontrado = Cierto) entonces   /* Comprueba si supvalor era el primer nodo */
2      Si (anterior = Nulo)                /* Es el primer nodo */
3          Entonces  Lista = Lista.siguiente
                     Liberarnodo (actual)

                 Sino      /* Quita el nodo en el medio o el final */
                           anterior.siguiente = actual.siguiente
                           Liberarnodo (actual)
           Fin si
   Fin si
   Fin
```

## *2.2 Estructuras de datos dinámicas no lineales*

En esta sección del Capítulo II se describirán las estructuras de datos dinámicas no lineales: los árboles y los grafos.

### 2.2.1 Árboles

#### 2.2.1.1 Definición y conceptos

Un árbol es una estructura de datos no-lineal y dinámica. Es no-lineal debido a que a cada elemento pueden seguirle varios elementos y es dinámica porque su estructura puede cambiar durante su ejecución.

**Árboles generales**

Un árbol general es un árbol donde cada nodo puede tener cero o más hijos (un árbol binario es un caso especializado de un árbol general). Los árboles generales se utilizan para modelar aplicaciones como los sistemas de archivos.

Se puede definir a un árbol como un conjunto finito no vacío T de elementos, llamados nodos, tales que:

- Existe un nodo raíz.
- El resto de los nodos se distribuye en un número n de subconjuntos distintos.
- Cada uno de estos subconjuntos es un subárbol del nodo raíz.

En la **Figura 18** se muestran cada uno de estos elementos:

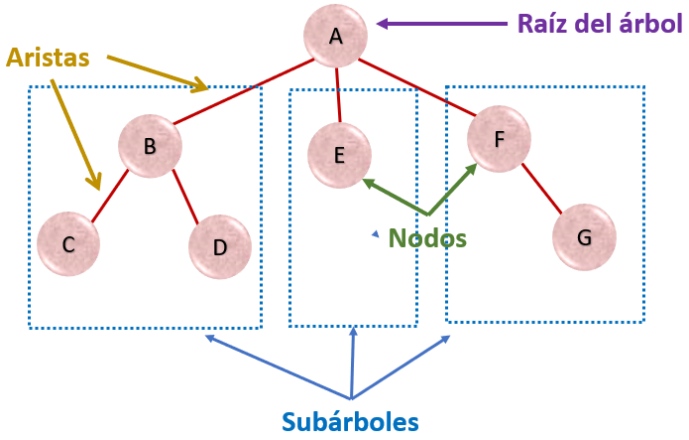



*Figura 19. Árbol general. Elaboración propia.*

**Nodos del árbol:** A, B, C, D, E, F

**Nodo raíz:** A

**Raíz:** es el primer nodo del árbol, se encuentra ubicado en la parte superior del árbol. Solo hay una raíz por árbol y una ruta desde el nodo raíz a cualquier nodo.

**Aristas o ramas:** son las líneas que unen dos nodos.

**Padre:** es el nodo que tiene hijos, es decir, nodos inferiores que se encuentran unidos al nodo superior por una arista. El único nodo que no tiene padre es el nodo raíz del árbol.

A cada nodo se le asocian uno o varios subárboles llamados descendientes o hijos.

> **Descendientes o hijos del nodo B:** C, D

**Hijo:** el nodo debajo de un nodo dado (padre) conectado por su arista hacia abajo. Cada nodo puede tener un número arbitrario de hijos.

> **Hijos del nodo A:** B, E, F. Estos a su vez conforman los subárboles del árbol general.

**Nodos hermanos:** son los sucesores o descendientes directos de un mismo nodo (hijos de un mismo padre).

> **Nodos hermanos (hijos de B):** C, D

**Hojas:** son los nodos sin hijos. También se les llama nodos terminales, nodos de grado o nodos externos.

> **Nodos hojas:** C, D, E, G

**Nodos internos:** son los nodos que tienen al menos un hijo.

> **Nodos internos:** A, B, F

En la **Figura 20** se muestra una representación gráfica de los conceptos de padre, hijo, hermanos, hojas y nodo interno.

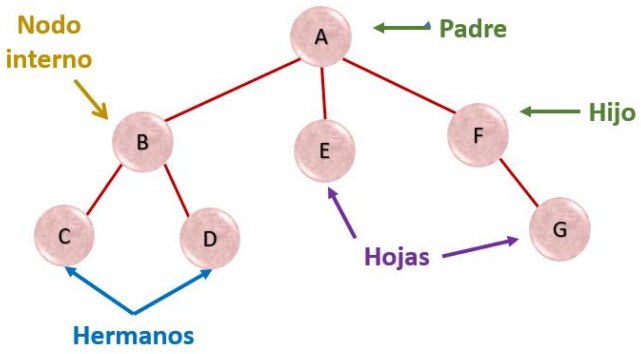



*Figura 20. Conceptos de padre, hijo, hermanos, hojas y nodo interno. Elaboración propia.*

**Camino de un nodo:** es la secuencia de aristas a través de las cuales se pasa desde el nodo raíz a un nodo.

**Rama:** Un camino que termina en una hoja.

**Antecesores o predecesores de un nodo:** son todos los nodos del camino que va desde la raíz del árbol hasta el nodo.

**Antecesores de D:** B, A

**Descendientes o sucesores de un nodo:** son aquellos nodos accesibles por un camino que comience en el nodo.

**Descendientes del nodo E:** H, I, J

En la **Figura 21** se muestra una representación gráfica de los conceptos de antecesores, descendientes, camino y rama.

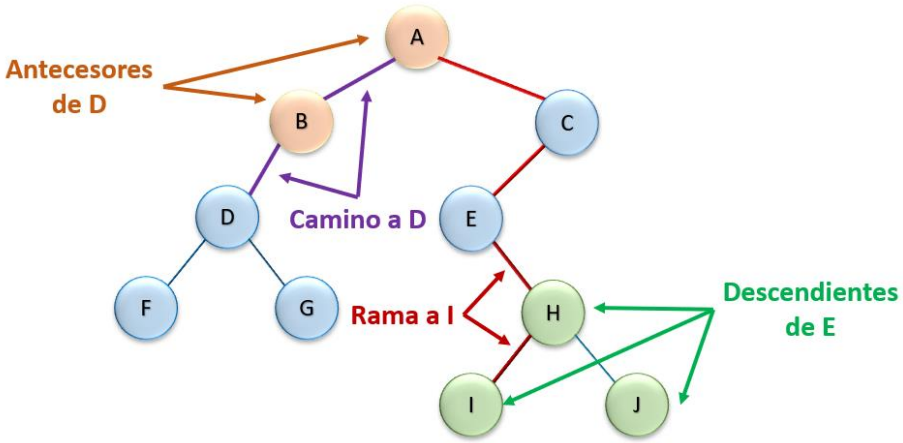



*Figura 21. Conceptos de antecesores, descendientes, camino y rama. Elaboración propia.*

**Grado de un nodo:** es el número de hijos que tiene el nodo. Así, el grado de un nodo hoja es cero. En la figura anterior el grado del nodo D es 2 y el grado del nodo E es 1.

**Grado del árbol:** es el mayor grado de sus nodos. Por ejemplo, el árbol binario es de grado 2 porque cada nodo tiene como mucho dos descendientes directos.

**Niveles:** es la distancia desde la raíz en la que se encuentra ubicado cada nodo. Cada nodo de un árbol tiene asignado un número de nivel de la siguiente forma la raíz tiene el número de nivel 0, y al resto de los nodos se le asigna un número de nivel que es mayor en 1 que el número de nivel del padre.

**Nivel de un nodo:** se refiere a la distancia del nodo desde la raíz del árbol.

En la **Figura 22** se muestra una representación gráfica de los conceptos de niveles del árbol.

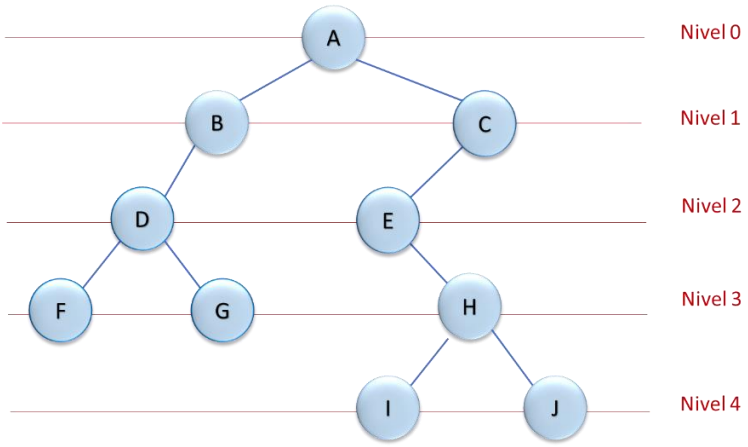



*Figura 22. Conceptos de niveles del árbol. Elaboración propia.*

**Altura de un nodo:** es la longitud del camino más largo que comienza en el nodo y termina en una hoja.

- La altura de un nodo hoja es 0.
- La altura de un nodo es igual a la mayor altura de sus hijos + 1. Por ejemplo, la altura del nodo B en la **Figura 22** es 2.

**Altura de un árbol:** es la longitud de la rama más larga del árbol más uno. Equivale a 1 más que el mayor número de nivel del árbol.

**Profundidad de un nodo:** es la longitud del camino (único) que comienza en la raíz y termina en el nodo. También se denomina nivel.

- La profundidad de la raíz es 0
- La profundidad de un nodo es igual a la profundidad de su padre + 1

## Árboles binarios

El árbol binario es un árbol en el cual cada nodo tiene como máximo dos hijos, uno a la izquierda y el otro a la derecha. En un **árbol binario de búsqueda** el hijo izquierdo, si existe, debe tener un valor menor que el valor de su padre y el hijo derecho, si existe, debe tener un valor mayor que el valor de su padre.

En la **Figura 23** se muestra un ejemplo de un árbol binario de búsqueda:

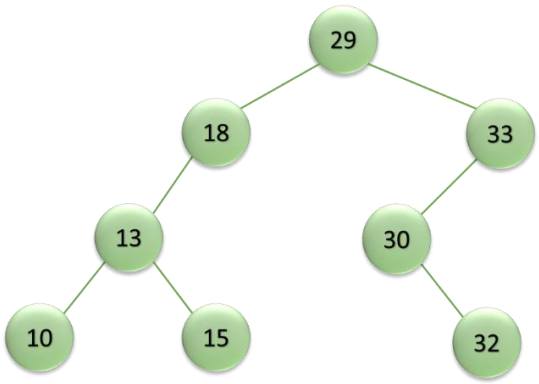



*Figura 23. Árbol binario de búsqueda. Elaboración propia.*

**Árboles binarios distintos:** dos árboles binarios son distintos cuando sus estructuras son diferentes. Un ejemplo de árboles binarios distintos se muestra en la **Figura 24**.

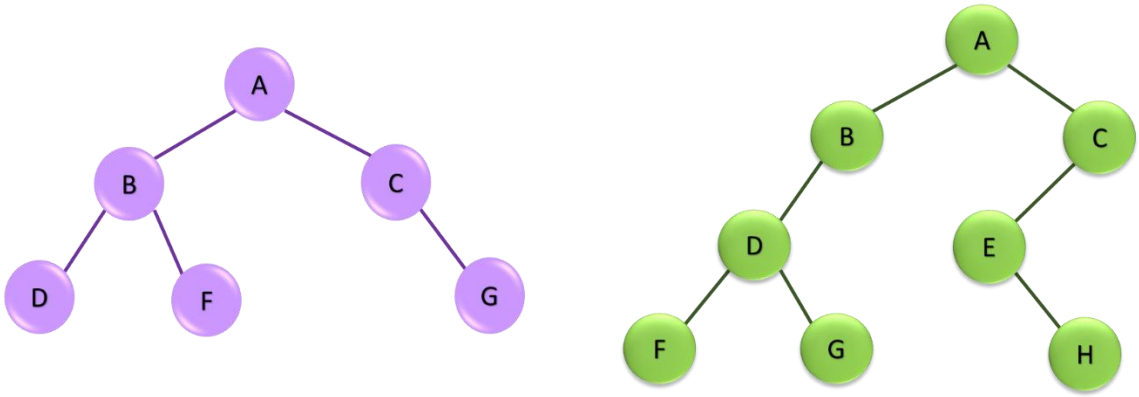



*Figura 24. Árboles binarios distintos. Elaboración propia.*

**Árboles binarios similares:** dos árboles binarios son similares si sus estructuras (forma) son idénticas pero la información que contienen sus nodos difiere entre sí. Un ejemplo de árboles binarios distintos se muestra en la **Figura 25**.

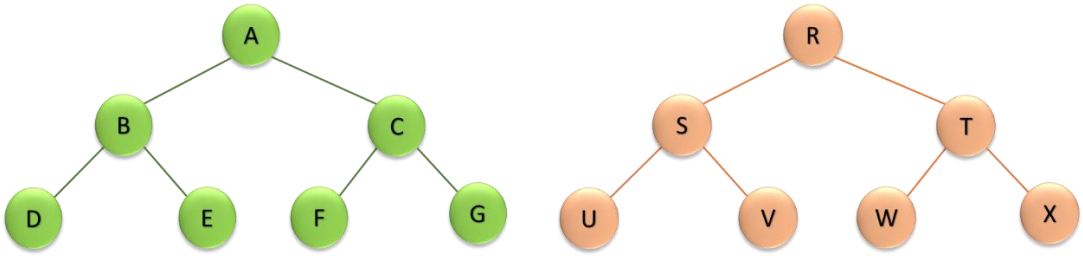



*Figura 25. Árboles binarios similares. Elaboración propia.*

**Árboles binarios equivalentes:** son aquellos que son similares y los nodos contienen la misma información. Un ejemplo de árboles binarios distintos se muestra en la **Figura 26**.

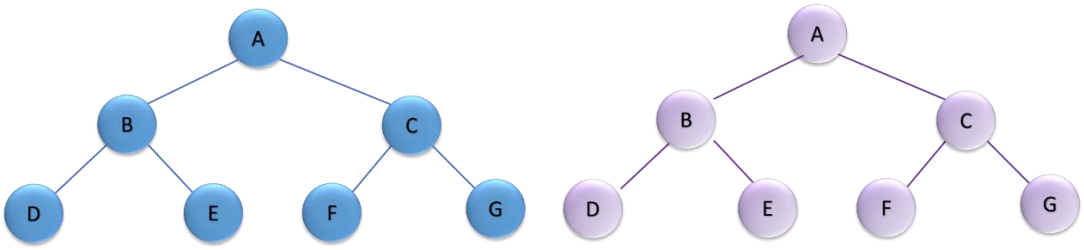



*Figura 26. Árboles binarios equivalentes. Elaboración propia.*

**Árbol binario completo:** el árbol binario es completo si todos sus niveles, excepto posiblemente el último, tienen el máximo número de nodos posibles y si todos los nodos del último nivel están situados lo más posible a la izquierda. Un ejemplo de árboles binarios distintos se muestra en la **Figura 27**.

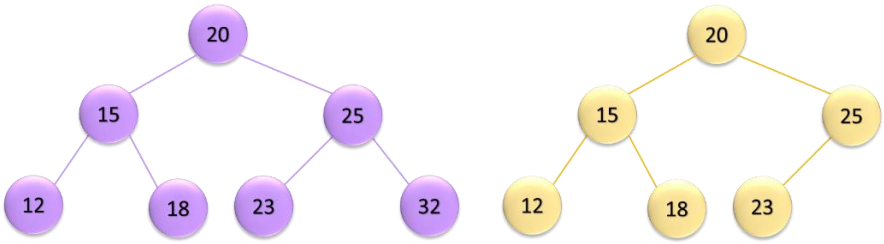



*Figura 27. Árboles binarios de búsqueda completos. Elaboración propia.*

### Árboles binarios extendidos: árboles-2

Es un árbol binario en donde el número de hijos de cada nodo es igual al grado del árbol, en este caso, es 2. Si alguno de los nodos no cumple con esta condición, entonces se le deben agregar tantos nodos especiales como se requiera para llegar a cumplirla.

Para convertir el árbol binario de la **Figura 28** a un árbol-2 añadimos a cada nodo, que no tenga dos hijos, los nodos externos necesarios para hacer cumplir dicha cantidad. Los nodos externos añadidos son representados por medio de un cuadrado.

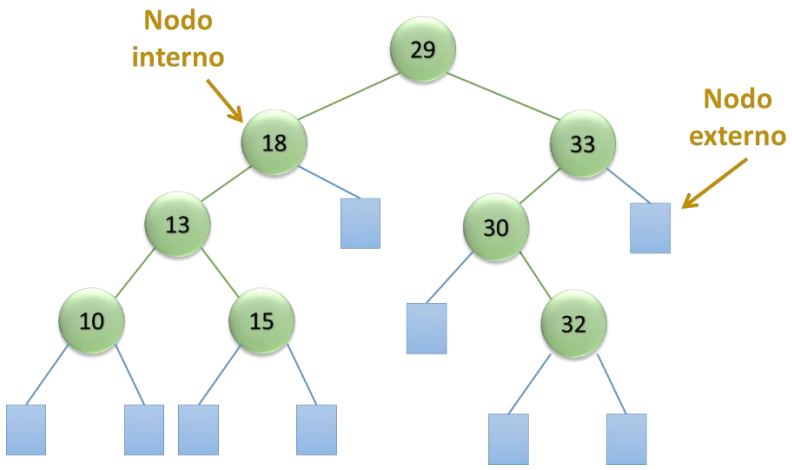



*Figura 28. Árboles binarios de búsqueda extendidos. Elaboración propia.*

### 2.2.1.2 Representación

Existen dos formas de representar un árbol binario en memoria:

- Por medio de punteros o enlazada
- Por medio de arreglos

La representación de punteros o enlazada es análoga a la forma en que se representan las listas enlazadas en memoria; la representación secuencial utiliza un arreglo simple.

#### Enlazada

Un árbol binario se puede representar en memoria de forma enlazada utilizando tres arreglos paralelos (**Figura 29**): INFO, IZQ y DER como se muestra en la figura de abajo y una variable puntero a la que llamaremos RAIZ.


*Figura 29. Arreglos paralelos usados en la representación enlazada de un árbol binario de búsqueda. Elaboración propia.*

A cada nodo N del árbol le corresponderá una posición K, tal que:

- INFO [K] contendrá los datos del nodo N.
- IZQ [K] contendrá la localización del hijo izquierdo del nodo N.
- DER [K] contendrá la localización del hijo derecho del nodo N.

La raíz contendrá la posición de la raíz R del árbol. Cuando el árbol está vacío, la RAIZ contendrá el valor nulo.

En el siguiente ejemplo se mostrará la representación enlazada en memoria de un árbol binario de búsqueda. Observe que cada nodo está dibujado con sus tres campos y que los subárboles vacíos están dibujados usando un diagonal / para las entradas nulas.

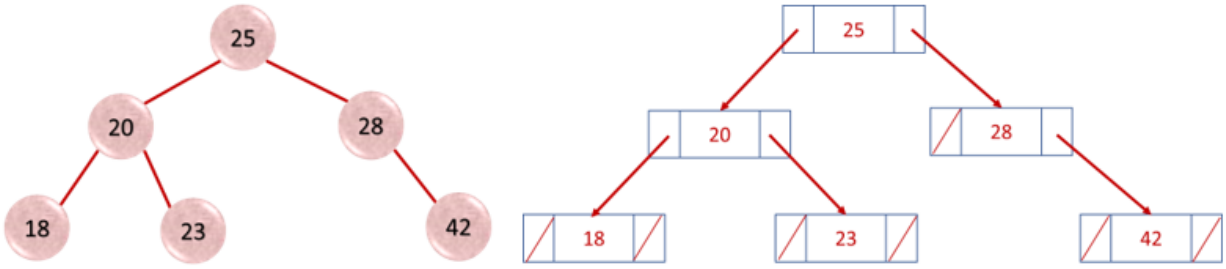



*Figura 30. Representación enlazada de un árbol binario de búsqueda. Elaboración propia.*

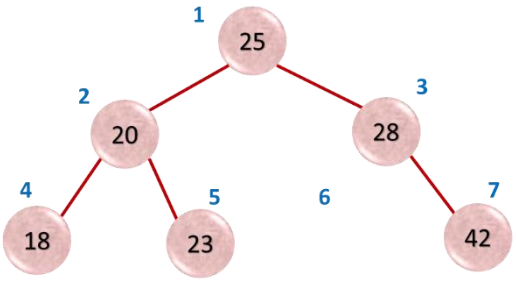
### Secuencial

Esta representación usa un arreglo lineal al que llamaremos ARBOL de la forma siguiente:

- La raíz R del árbol se guarda en la posición del arreglo ARBOL [1].
- Si un nodo N está ubicado en la posición ARBOL [K], entonces sus hijos:
  - Izquierdo está en la posición ARBOL [2*K]
  - Derecho en la posición ARBOL [2*K+1]

Se usa nulo para indicar que el árbol o un subárbol está vacío, así ARBOL [1] = NULO indica que el árbol está vacío.

En la **Figura 31** se mostrará la representación secuencial en memoria de un árbol binario de búsqueda. Observe que cada nodo está ubicado en el arreglo en la misma posición que tiene en el árbol. Para una mejor comprensión de dichas posiciones se ha colocado el número que le corresponde a cada una de ellas al lado del nodo en el árbol.




ARBOL

| POSICIÓN | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| VALOR | 25 | 20 | 28 | 18 | 23 | | 42 |

*Figura 31. Representación Secuencial de un árbol binario de búsqueda. Elaboración propia.*

### 2.2.1.3 Recorridos en un árbol binario

El recorrido en un árbol es el proceso de visitar todos los nodos de un árbol. Se clasifican los algoritmos de recorrido, dependiendo del orden en que se visitan los nodos.

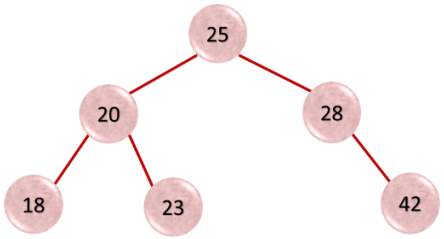
**Recorridos en un árbol binario**

- Profundidad
- Amplitud

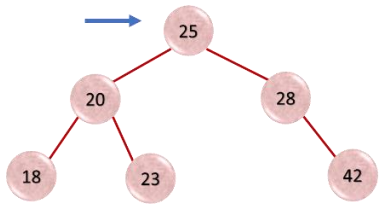
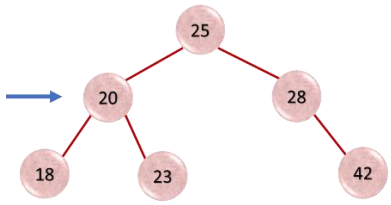
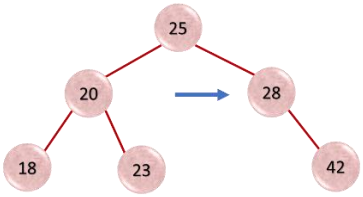
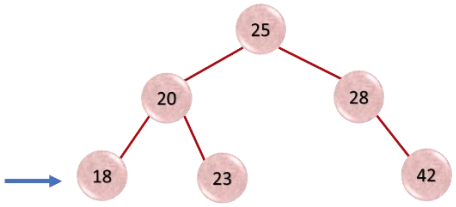
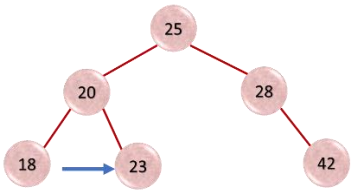
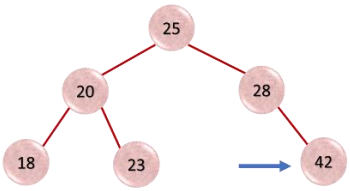
**Recorrido en amplitud:** se implementa utilizando la estructura de datos denominada cola. El recorrido empieza desde la raíz y atraviesa el árbol nivel por nivel, pasa por todos los nodos de un nivel antes de pasar a los nodos hijos.

**Ejemplo:**

Utilizando el siguiente árbol escriba el recorrido por amplitud.




- **Solución:**

<table>
<tr>
<td>PASO 1<br>  <br><br>25</td>
<td>PASO 2<br>  <br><br>25, 20</td>
</tr>
<tr>
<td>PASO 3<br>  <br><br>25, 20, 28</td>
<td>PASO 4<br>  <br><br>25, 20, 28, 18</td>
</tr>
<tr>
<td>PASO 5<br>  <br><br>25, 20, 28, 18, 23</td>
<td>PASO 6<br>  <br><br>25, 20, 28, 18, 23, 42</td>
</tr>
</table>

**Recorrido en profundidad:** se implementa utilizando la estructura de datos denominada pila. El algoritmo empieza desde la raíz y visita a todos los nodos de una sola rama del árbol. Cuando termina en esa rama, entonces hace una vuelta hacia atrás (denominado backtracking) y sigue por la otra rama.

**Recorridos en profundidad**

- Preorden
- Inorden
- Postorden

A continuación se describen los recorridos en profundidad:

- **Preorden**

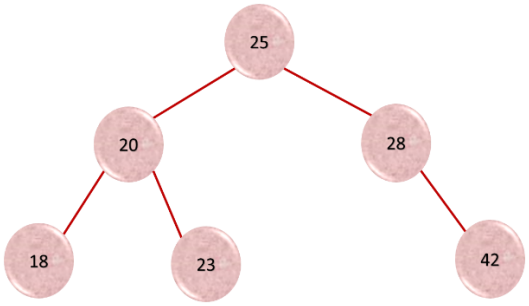
El orden del recorrido es el siguiente:

1. Raíz

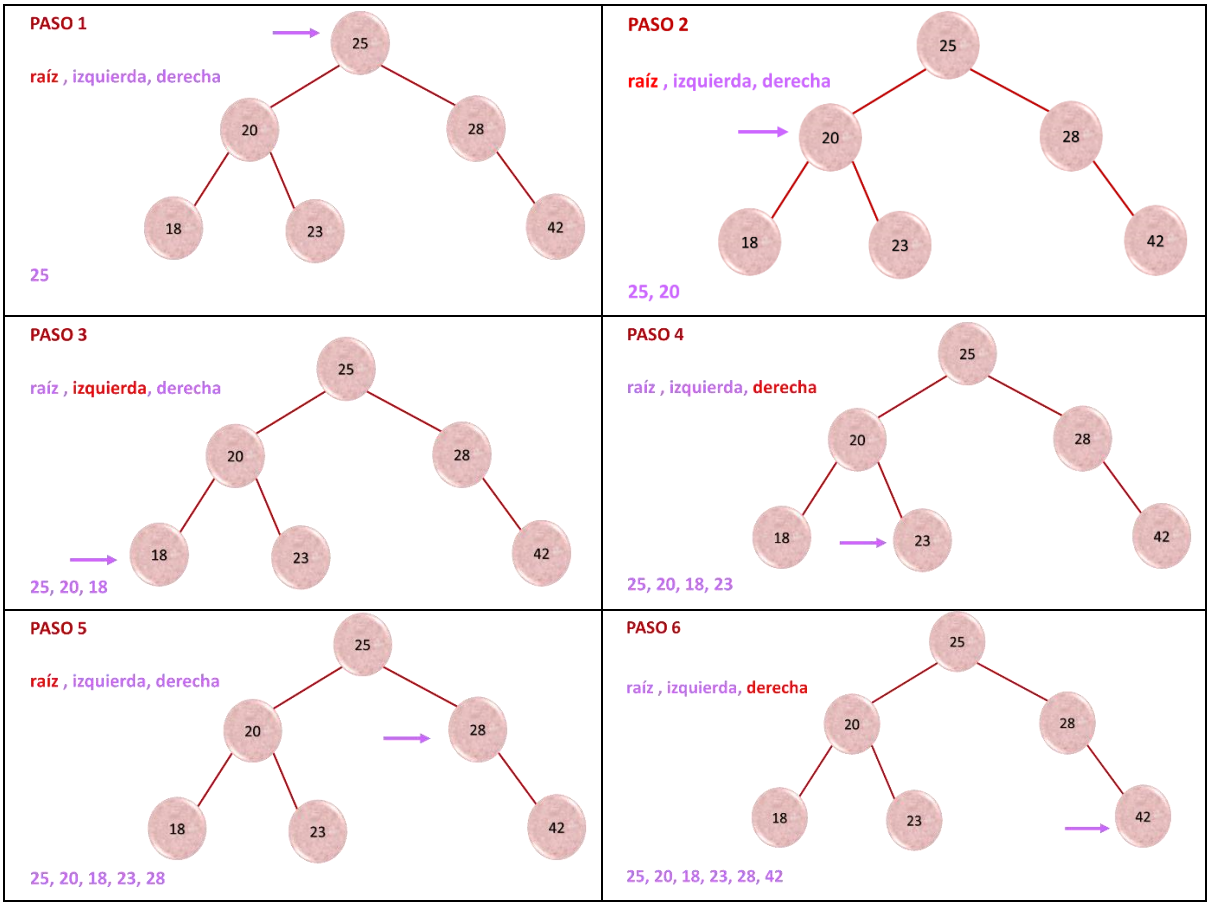
2. Subárbol izquierdo
3. Subárbol derecho

**Ejemplo:**

Utilizando el siguiente árbol escriba el recorrido por preorden.




- **Solución:**



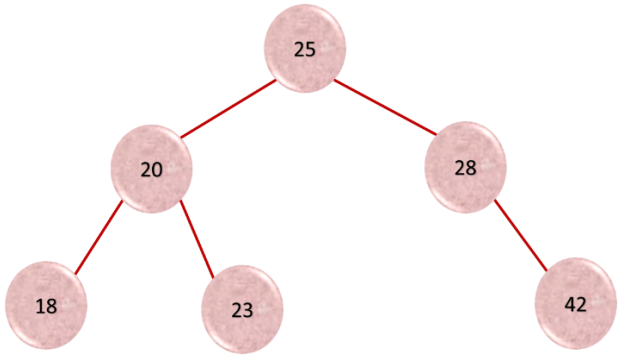

- **Inorden**

El orden del recorrido es el siguiente:

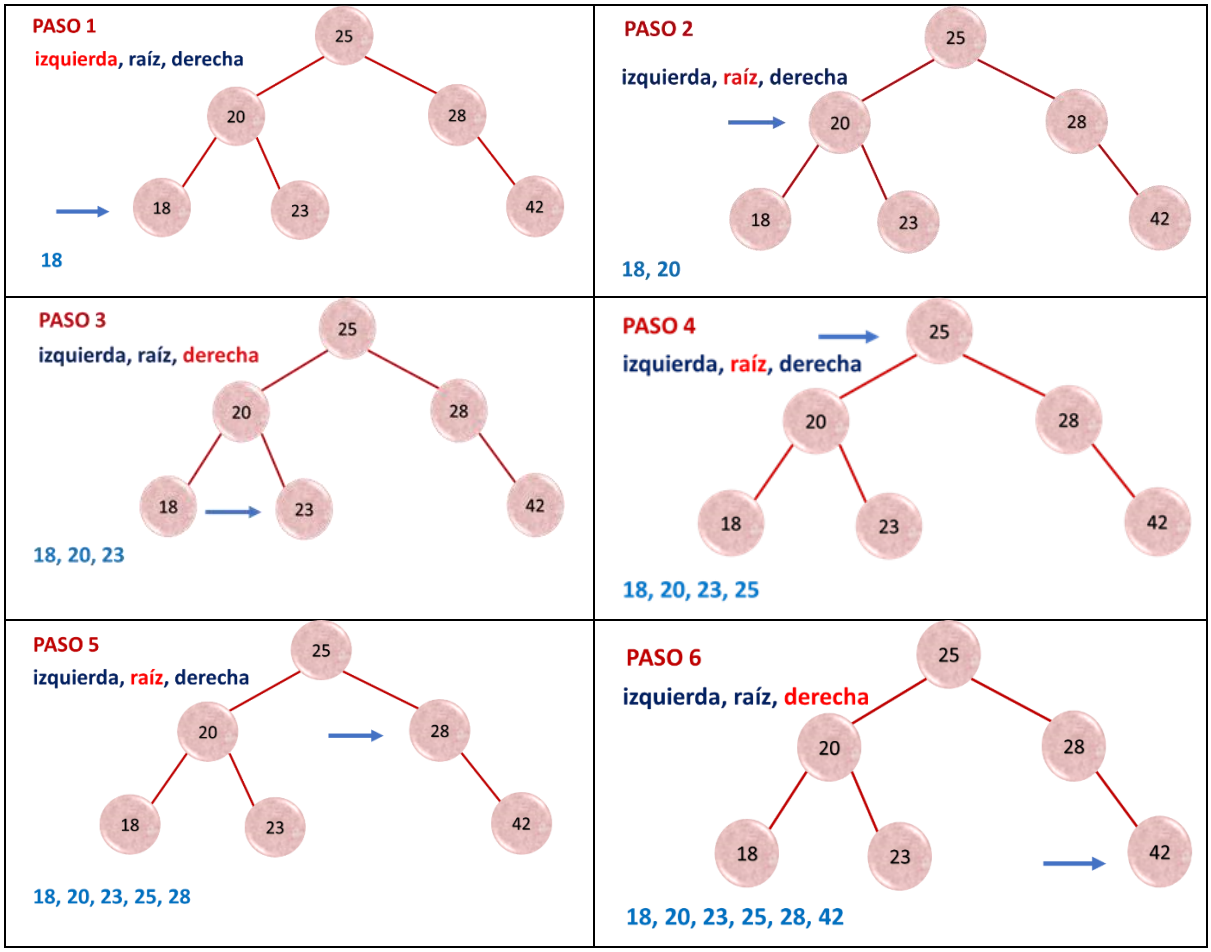
1. Subárbol izquierdo
2. Raíz
3. Subárbol derecho

**Ejemplo:**

Utilizando el siguiente árbol escriba el recorrido por inorden.




- **Solución:**



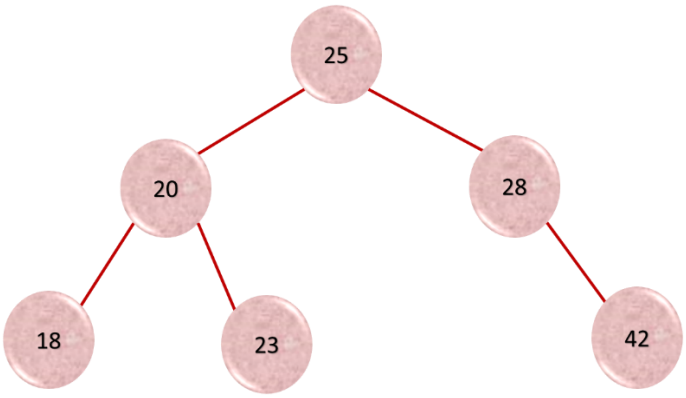
- **Postorden**

El orden del recorrido es el siguiente:

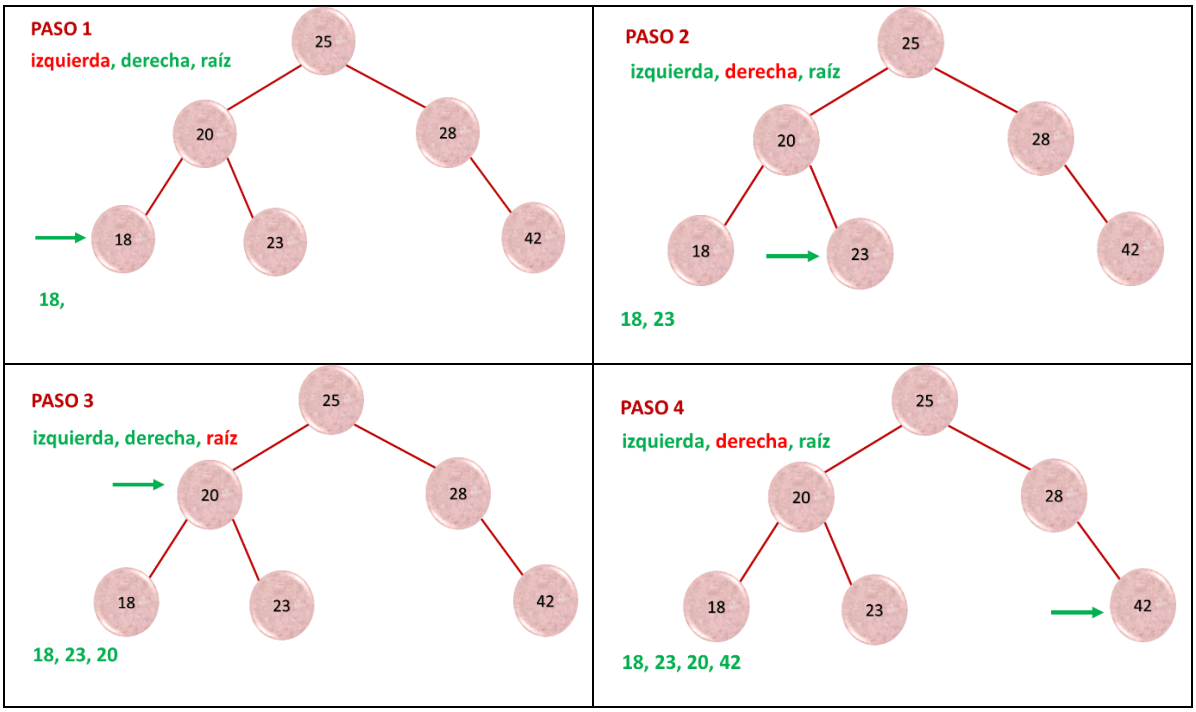
1. Subárbol izquierdo
2. Subárbol derecho
3. Raíz

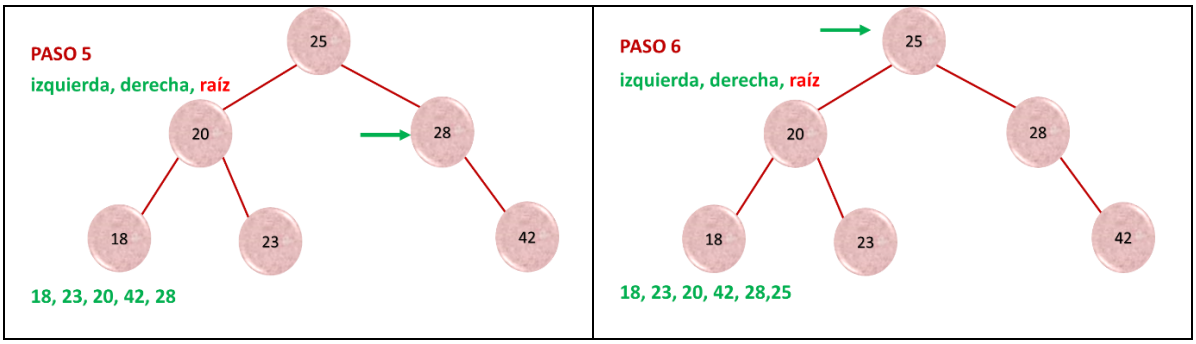
**Ejemplo:**

Utilizando el siguiente árbol escriba el recorrido por postorden.




- **Solución:**






## 2.2.2 Grafos

### 2.2.2.1 Definición y conceptos

Un grafo está formado por un conjunto de nodos (llamados también vértices) y un conjunto de líneas llamadas aristas (o arcos) que conectan los diferentes nodos.

**Definición formal de un grafo:**

**G = (V, A)**

donde

- **V(G)** es un conjunto finito, no vacío de vértices que se especifican listando los nodos en notación conjunto, es decir, dentro de paréntesis o llaves.
- **A(G)** es un conjunto de aristas (pares de vértices) que se específica listando una secuencia de aristas. Cada arista se denota escribiendo los nombres de los dos nodos que conecta entre paréntesis, con una coma entre ellos.

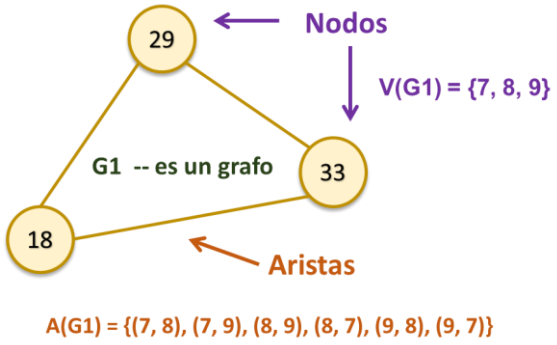



*Figura 32. Grafo no dirigido G1. Elaboración propia.*

#### Clasificación de los grafos

A continuación se describen las diferentes clasificaciones de los grafos.

**Grafo dirigido:** la dirección de la línea es indicada por el nodo que se lista primero. Un ejemplo de grafo dirigido se muestra en la **Figura 33**.

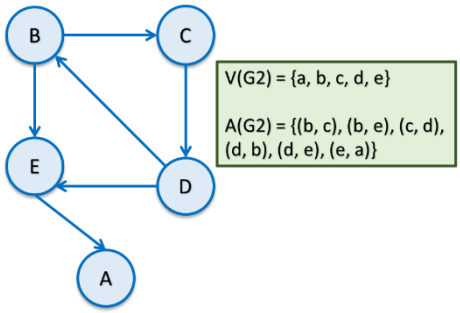



*Figura 33. Ejemplo de grafo dirigido. Elaboración propia.*

**Grafo no dirigido:** la relación entre los dos nodos es desordenada. Es decir, un nodo apunta al otro; ellos están simplemente conectados. Un ejemplo de grafo no dirigido se muestra en la **Figura 34**.

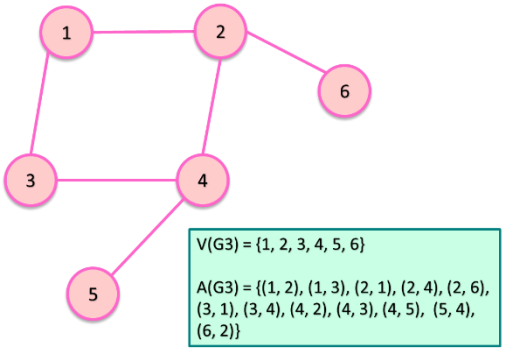



*Figura 34. Ejemplo de grafo no dirigido. Elaboración propia.*

### Tipos particulares de grafos

**Grafo acíclico:** es aquel grafo no contiene ningún ciclo simple. Un ejemplo de este tipo de grafo se muestra en la **Figura 35**.

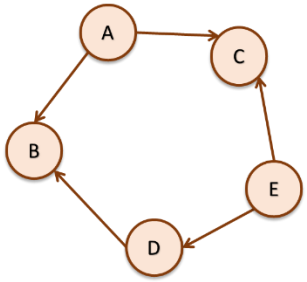



*Figura 35. Grafo acíclico. Elaboración propia.*

**Grafo cíclico:** un grafo se dice cíclico si contiene algún ciclo simple. Un ejemplo de este tipo de grafo se muestra en la **Figura 36**.

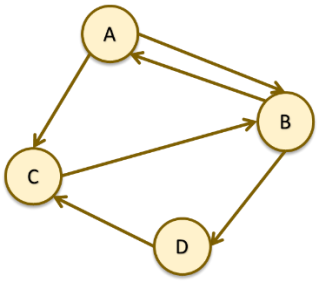



*Figura 36. Grafo cíclico. Elaboración propia.*

**Grafo regular:** es un grafo cuyos vértices tienen el mismo grado. Un ejemplo de este tipo de grafo se muestra en la **Figura 37**.

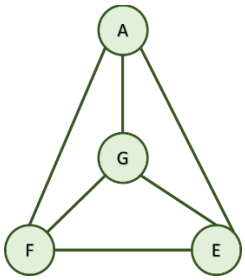



*Figura 37. Grafo regular. Elaboración propia*

**Grafo plano:** es un grafo que es posible dibujar en el plano sin que ningún par de aristas se crucen entre sí. Un ejemplo de este tipo de grafo se muestra en la **Figura 38**.

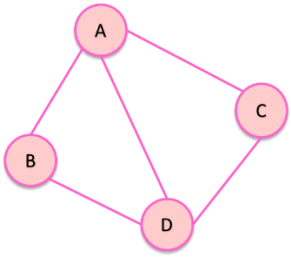



*Figura 38. Grafo plano. Elaboración propia.*

**Grafo simple o simplemente grafo:** es aquel que acepta una sola una arista uniendo dos vértices cualesquiera. Esto es equivalente a decir que una arista cualquiera es la única que une dos vértices específicos. Es la definición estándar de un grafo. Un ejemplo de este tipo de grafo se muestra en la **Figura 39**.

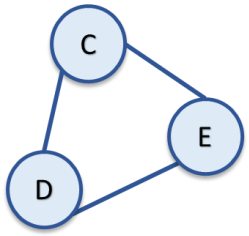



*Figura 39. Grafo simple. Elaboración propia.*

**Multigrafo:** son grafos que aceptan más de una arista entre dos vértices. Estas aristas se llaman múltiples o lazos. Los grafos simples son una subclase de esta categoría de grafos. Un ejemplo de este tipo de grafo se muestra en la **Figura 40**.

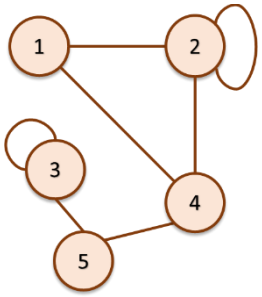



*Figura 40. Multigrafo. Elaboración propia.*

**Grafo vacío:** es el grafo cuyo conjunto de aristas es vacío. Un ejemplo de este tipo de grafo se muestra en la **Figura 41**.

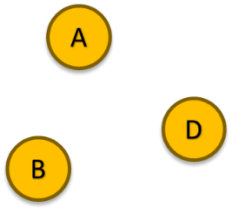



*Figura 41. Grafo vacío. Elaboración propia.*

**Grafo completo:** un grafo es completo si cada vértice tiene un grado igual a n-1, donde n es el número de vértice que compone el grafo. Además, es un grafo simple en el que cada vértice es adyacente a cualquier todo otro vértice. Un ejemplo de este tipo de grafo se muestra en la **Figura 42**.

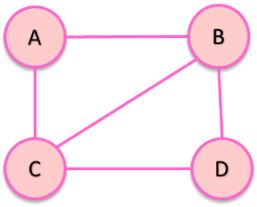



*Figura 42. Grafo completo. Elaboración propia.*

**Grafo conexo:** decimos que es un grafo conexo, si es posible formar un camino desde cualquier vértice a cualquier otro en el grafo. Un ejemplo de este tipo de grafo se muestra en la **Figura 43**.

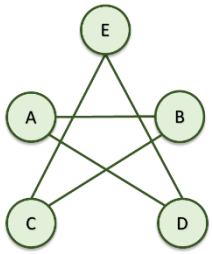



*Figura 43. Grafo conexo. Elaboración propia.*

**Grafo bipartito:** es cualquier grafo, cuyos vértices pueden ser divididos en dos conjuntos, tal que no haya aristas entre los vértices del mismo conjunto. Se ve que un grafo es bipartito si no hay ciclos de longitud impar. Un ejemplo de este tipo de grafo se muestra en la **Figura 44**.

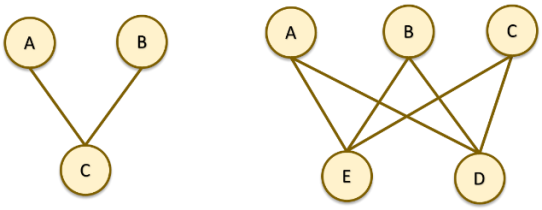



*Figura 44. Grafo bipartito. Elaboración propia.*

**Grafo denso:** un grafo denso es aquel grafo en el que el número de aristas está cercano al número de máximo de aristas. Un ejemplo de este tipo de grafo se muestra en la **Figura 45**.

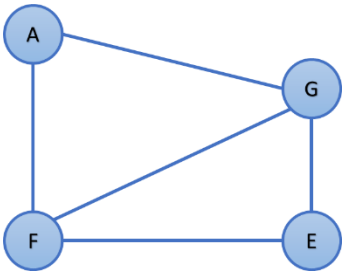



*Figura 45. Grafo denso. Elaboración propia.*

**Grafo con peso:** es un grafo en el que cada arista transporta un valor. Los grafos con pesos se usan para representar situaciones en las que el valor de la conexión entre los vértices es importante, no sólo la existencia de la conexión. Un ejemplo de este tipo de grafo se muestra en la **Figura 46**.

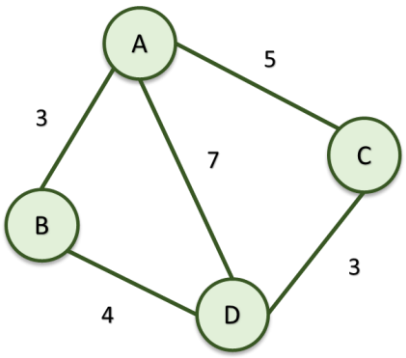



*Figura 46. Grafo no dirigido con peso en las aristas. Elaboración propia.*

### Otras definiciones

**Grado total de un vértice:** corresponde al número de aristas incidentes sobre el vértice, en donde, cada bucle lo cuenta dos veces. En base a esta definición podemos mencionar:

**Vértice fuente** es un vértice con grado de entrada cero.

**Vértice sumidero:** es un vértice con grado de salida cero.

**Vértice hoja:** es un vértice con grado uno.

**Vértice aislado:** tiene grado cero.

**Grado total de un vértice (Gr) en un grafo no dirigido:** es igual al número de aristas que tiene el vértice.

En la **Figura 47** se muestra un ejemplo del grado de un vértice en un grafo:

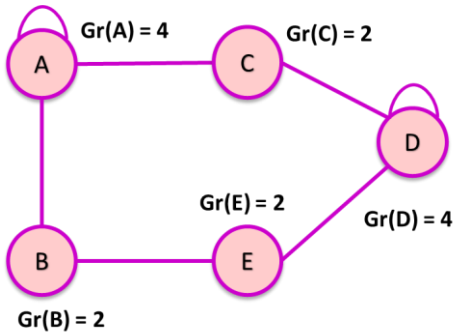



*Figura 47. Grado de un vértice en un grafo. Elaboración propia.*

En un grafo dirigido distinguimos entre grado de entrada (Ge) y grado de salida (Gs) de un vértice. A continuación, se explican cada uno de ellos.

**El grado de entrada (Ge) de un vértice en un grafo dirigido:** es el número de aristas que llegan al vértice. Un ejemplo de esto se muestra en la **Figura 48**.

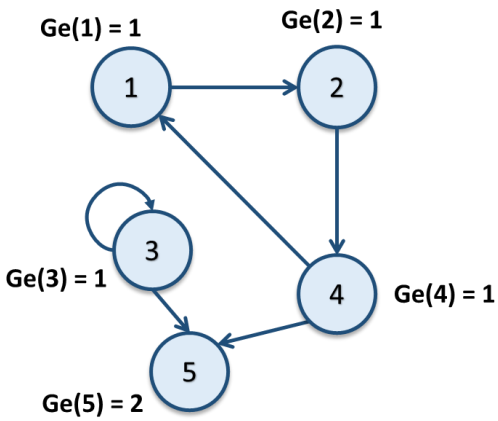



*Figura 48. Grado de entrada de un vértice en un grafo dirigido. Elaboración propia.*

**El grado de salida (Gs) de un vértice en un grafo dirigido:** corresponde al número de aristas que salen del vértice. Un ejemplo de esto se muestra en la **Figura 49**.

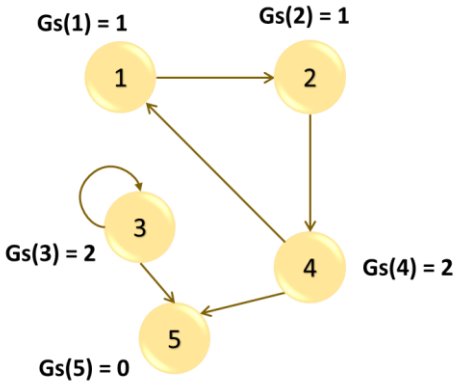



*Figura 49. Grado de salida de un vértice en un grafo dirigido. Elaboración propia.*

**El grado total de un vértice (Gr) en un grafo dirigido:** es la suma del grado entrante más el grado saliente. Un ejemplo de esto se muestra en la **Figura 50**.

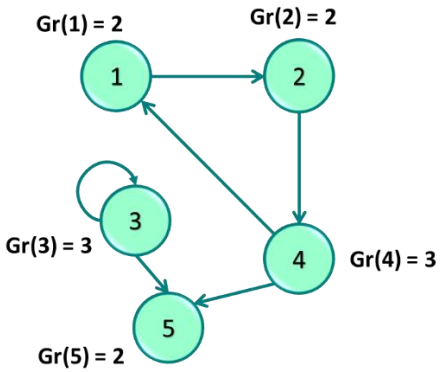



*Figura 50. Grado total de un vértice en un grafo dirigido. Elaboración propia.*

**Camino:** es una sucesión de vértices que conecta al vértice $V_i$ con el vértice $V_j$. Es decir, debe haber una secuencia ininterrumpida de aristas desde $V_i$ a través de cualquier número de nodos hasta $V_j$. Un ejemplo de esto se muestra en la **Figura 51**.

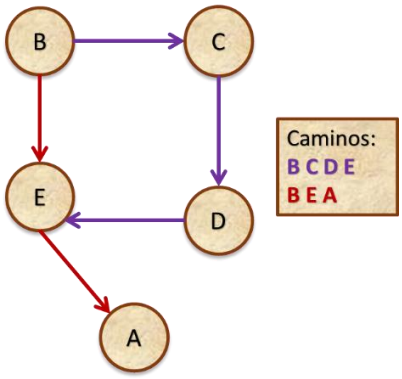



*Figura 51. Caminos en el grafo. Elaboración propia.*

### 2.2.2.2 Representación

#### Representación secuencial de grafos

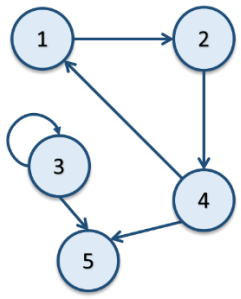
En esta sección se muestran dos formas de representar secuencialmente un grafo: la matriz de adyacencia y la matriz de caminos.

#### Matriz de adyacencia

Para un grafo con N nodos la matriz de adyacencia será una tabla con N filas y N columnas, en donde, el valor en la posición [i, j] de la tabla será 1 si existe una arista ($V_i$, $V_j$) perteneciente al conjunto de las aristas A, y 0 en otro caso. Si el grafo es un grafo con peso, la celda [i, j] contendrá el peso de la arista si la arista está en el conjunto A, y 0 en otro caso.

**Ejemplo**

Escriba la matriz de adyacencia para el siguiente grafo.

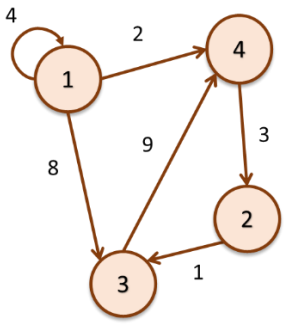



**Solución**

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | 1 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 1 | 0 |
| 3 | 0 | 0 | 1 | 0 | 1 |
| 4 | 1 | 0 | 0 | 0 | 1 |
| 5 | 0 | 0 | 0 | 0 | 0 |

**Ejemplo**

Escriba la matriz de adyacencia para el siguiente grafo con peso.




**Solución**

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 4 | 0 | 8 | 2 |
| 2 | 0 | 0 | 1 | 0 |
| 3 | 0 | 0 | 0 | 9 |
| 4 | 0 | 3 | 0 | 0 |

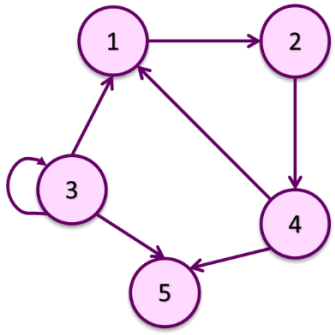
## Matriz de caminos

Sea G un grafo dirigido simple con *m* nodos $v_1$, $v_2$ … $v_m$. La **matriz de caminos** o **matriz de alcance** de G es la matriz *m*-cuadrada P= V(i, j) definida como sigue:

$$P = \begin{cases} \mathbf{1} & \text{si hay un camino desde } V_i \text{ hasta } V_j \\ \mathbf{0} & \text{en otro caso} \end{cases}$$

**Ejemplo**

Escriba la matriz de caminos para el siguiente grafo.




**Solución**

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 1 | 1 |
| 2 | 1 | 1 | 0 | 1 | 1 |
| 3 | 1 | 0 | 1 | 0 | 1 |
| 4 | 1 | 1 | 0 | 1 | 1 |
| 5 | 0 | 0 | 0 | 0 | 0 |

## Representación enlazada de grafos

Otra forma de representar los grafos es utilizando las listas enlazadas. Un ejemplo de esto se muestra en la **Figura 52**.

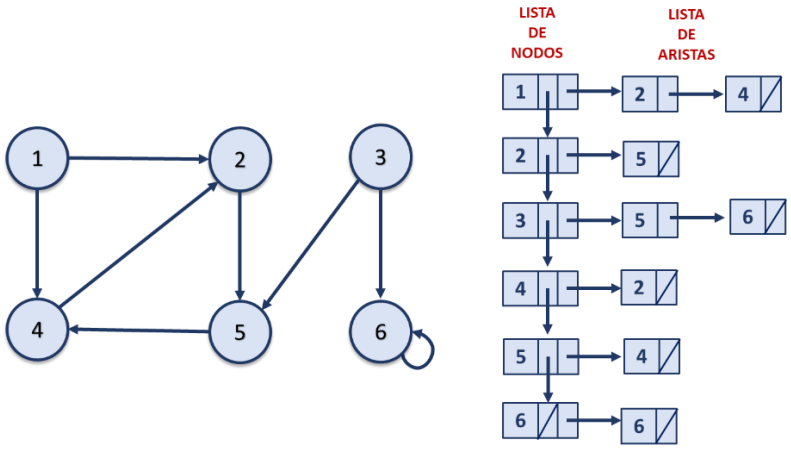



*Figura 52. Representación enlazada de grafos. Elaboración propia.*

En las listas enlazadas temenos:

- **Lista de nodos:** lista que contiene los nombres de los nodos. Está formada por tres partes (**Figura 53**):
  - **Info:** el nombre o valor clave del nodo.
  - **Sig:** puntero al siguiente nodo de la lista nodo.
  - **Ady:** puntero al primer elemento de la lista de adyacencia del nodo que se mantiene en la lista de aristas.

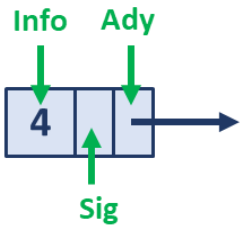



*Figura 53. Representación de las partes de la lista de nodos. Elaboración propia.*

- **Lista de aristas:** que contiene la(s) arista(s) a la(s) que el nodo está conectado. Está formada por dos partes (**Figura 54**):
  - **Dest:** campo que apunta al nodo destino de la arista.
  - **Enl:** campo que enlaza las aristas del mismo nodo inicial.


*Figura 54. Representación de las partes de la lista de aristas. Elaboración propia.*

En los grafos con peso se debe agregar un campo adicional en la lista de las aristas en donde se colocará el peso de las mismas. Esto se representa en el ejemplo mostrado en la **Figura 55**.

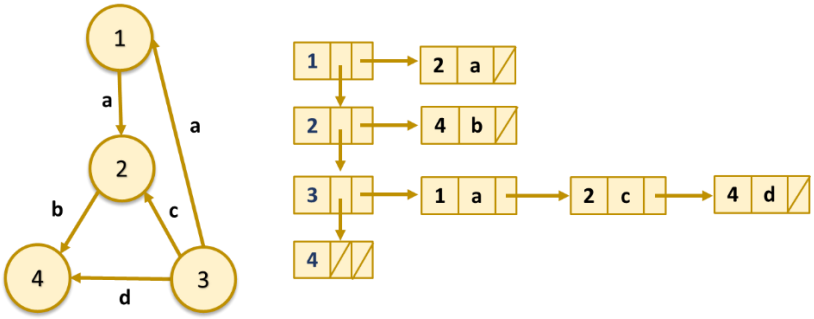



*Figura 55. Representación enlazada de grafos con peso. Elaboración propia.*

### 2.2.2.3 Recorridos

Hay dos enfoques básicos de recorridos: el recorrido en anchura y el recorrido en profundidad. A continuación, se explican cada uno de estos recorridos.

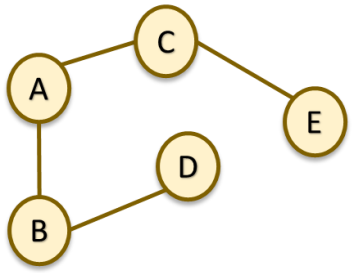
### Anchura

En este recorrido se empieza en un nodo v: primero se visita v, luego se visitan todos sus adyacentes. Luego los adyacentes de estos y así sucesivamente. El algoritmo utiliza una cola de vértices. Las operaciones básicas que se utilizarán en la cola son:

- Sacar un elemento de la cola
- Añadir a la cola sus adyacentes no visitados

**Ejemplo:**

Realizar el recorrido en anchura en el siguiente grafo. La lista de nodos inicia en el grafo con Actual = A.



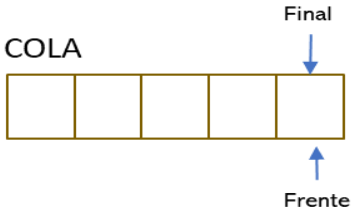

- **Solución:**

## PASO 1

Hacer un arreglo de visitados del grafo
Marcar cada nodo como no_visitado

| NODOS | A | B | C | D | E |
|---|---|---|---|---|---|
| VISITADOS | F | F | F | F | F |

Crear la Cola




## PASO 2

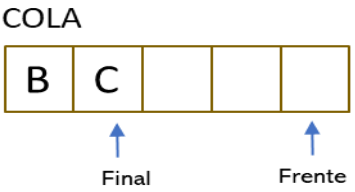
Actual = A
Marcar actual como visitado

| NODOS | A | B | C | D | E |
|---|---|---|---|---|---|
| VISITADOS | C | F | F | F | F |

## PASO 3

Marcar todos los vecinos de actual como visitados y los metemos en la Cola

| NODOS | A | B | C | D | E |
|---|---|---|---|---|---|
| VISITADOS | C | C | C | F | F |

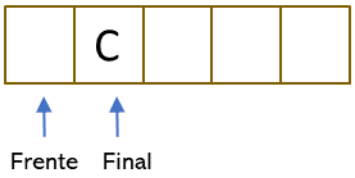



**RECORRIDO:**
**A**

## PASO 4

Sacar de la Cola y asignar a Actual

COLA

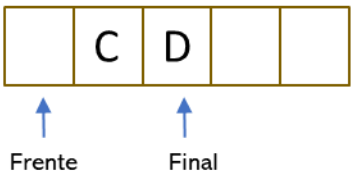



Actual = B

Marcar todos los vecinos no marcados de actual como visitados y los metemos en la Cola

| NODOS | A | B | C | D | E |
|---|---|---|---|---|---|
| VISITADOS | C | C | C | C | F |

COLA



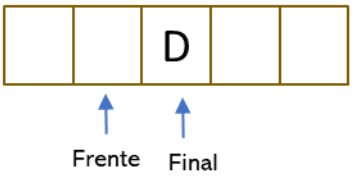

**RECORRIDO:**

A B

## PASO 5

Sacar de la Cola y asignar a Actual

COLA

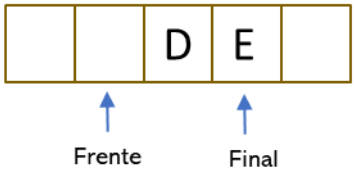



Actual = C

Marcar todos los vecinos no marcados de actual como visitados y los metemos en la Cola

| NODOS | A | B | C | D | E |
|---|---|---|---|---|---|
| VISITADOS | C | C | C | C | C |

COLA



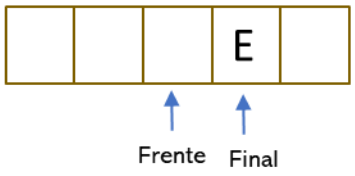

**RECORRIDO:**

**A B C**

## PASO 6

Sacar de la Cola y asignar a Actual

COLA

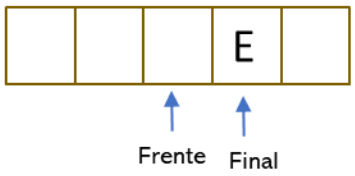



Actual = D

Marcar todos los vecinos no marcados de actual como visitados y los metemos en la Cola

| NODOS | A | B | C | D | E |
|---|---|---|---|---|---|
| VISITADOS | C | C | C | C | C |

COLA



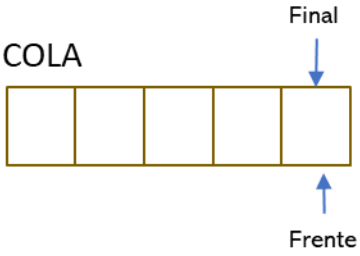

**RECORRIDO:**

**A B C D**

## PASO 7

Sacar de la Cola y asignar a Actual

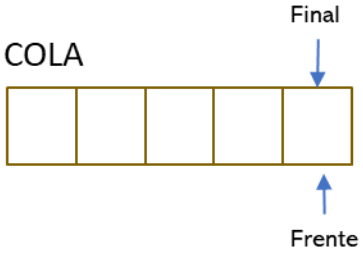
COLA




Actual = E

Marcar todos los vecinos no marcados de actual como visitados y los metemos en la Cola

| NODOS | A | B | C | D | E |
|---|---|---|---|---|---|
| VISITADOS | C | C | C | C | C |




**RECORRIDO:**

**A B C D E**

### Profundidad

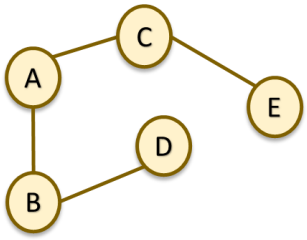
Consiste en alejarse todo lo posible del nodo origen para después empezar a visitar los nodos restantes a la vuelta.

En este tipo de recorrido hay que usar una pila para ir almacenando los nodos que nos falta visitar. Las operaciones básicas que se utilizarán en la pila son:

- Sacar un elemento de la pila.
- Añadir un elemento a la pila.

**Ejemplo:**

Realizar el recorrido en profundidad en el siguiente grafo. La lista de nodos inicia en el grafo con Actual = A.



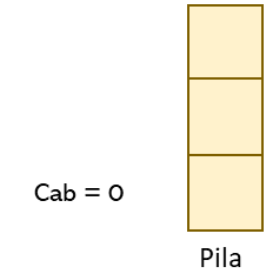

- **Solución:**

**PASO 1**

Hacer un arreglo de visitados del grafo
Marcar cada nodo como no_visitado

| NODOS | A | B | C | D | E |
|---|---|---|---|---|---|
| VISITADOS | F | F | F | F | F |

Crear la Pila



## PASO 2

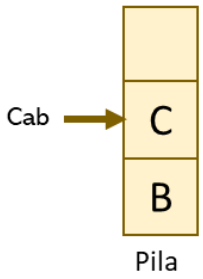
Actual = A
Marcar actual como visitado

| NODOS | A | B | C | D | E |
|---|---|---|---|---|---|
| VISITADOS | C | F | F | F | F |

## PASO 3

Marcar todos los vecinos de actual como visitados y los metemos en la Pila

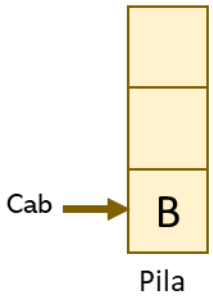
| NODOS | A | B | C | D | E |
|---|---|---|---|---|---|
| VISITADOS | C | C | C | F | F |




**RECORRIDO:**
**A**

## PASO 4

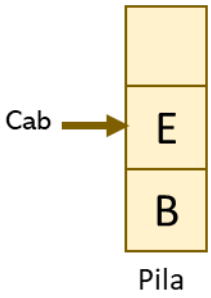
Sacar de la Pila y asignar a Actual




Actual = C

Marcar todos los vecinos no marcados de actual como visitados y los metemos en la Pila

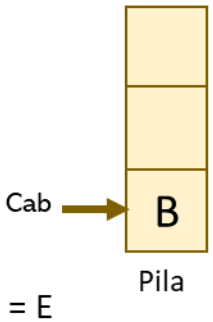
| NODOS | A | B | C | D | E |
|---|---|---|---|---|---|
| VISITADOS | C | C | C | F | C |




**RECORRIDO:**
**A C**

## PASO 5

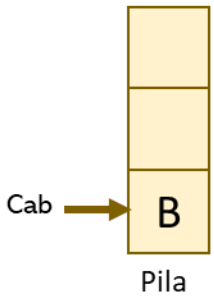
Sacar de la Pila y asignar a Actual




Actual = E

Marcar todos los vecinos no marcados de actual como visitados y los metemos en la Pila

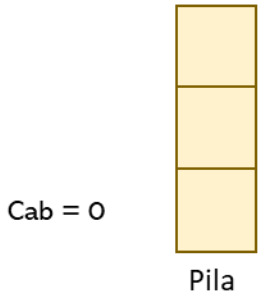
| NODOS | A | B | C | D | E |
|---|---|---|---|---|---|
| VISITADOS | C | C | C | F | C |




**RECORRIDO:**
**A C E**

## PASO 6

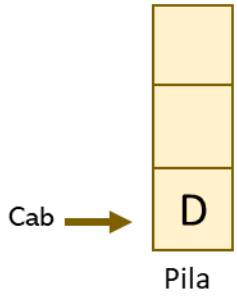
Sacar de la Pila y asignar a Actual




Actual = B

Marcar todos los vecinos no marcados de actual como visitados y los metemos en la Pila

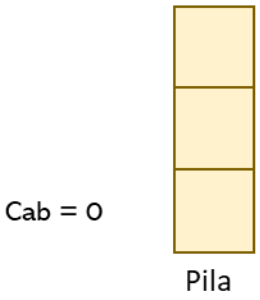
| NODOS | A | B | C | D | E |
|---|---|---|---|---|---|
| VISITADOS | C | C | C | C | C |




**RECORRIDO:**
**A C E B**

## PASO 7

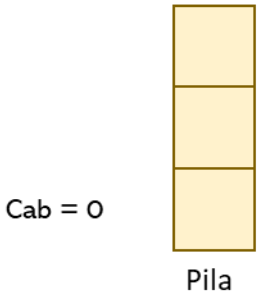
Sacar de la Pila y asignar a Actual




Actual = D

Marcar todos los vecinos no marcados de actual como visitados y los metemos en la Pila

| NODOS | A | B | C | D | E |
|---|---|---|---|---|---|
| VISITADOS | C | C | C | C | C |




**RECORRIDO:**
**A C E B D**

# Bibliografía

Estructuras de Datos con C y C++. Yedidyah Langsam, Moshe J. Augenstein, Aaron M. tenenbaum. Editorial Prentice Hall. 1997. Segunda Edición.

Estructuras de Datos. Osvaldo Cairó, Silvia Guardati. Editorial Mc Graw Hill. 2006. Tercera Edición.

Estructuras de Datos orientada a objetos Algoritmos con C++. Silvia Guardati. Editorial Pearson. 2007. Primera Edición.

Fundamentos de Programación. Algoritmos y Estructura de Datos. Aguilar, Luis Joyanes Mc Graw-Hill.

Estructura de Datos en Java. Joyanes Aguilar, Luis; Zahonero Martínez, Ignacio. Edit. McGraw.

Estructuras de Datos. Osvaldo Cairó, Silvia Guardati. Editorial Mc Graw Hill. 2006. Tercera Edición.

Algoritmos y Estructuras de Datos. Wirth, Niklaus. Prentice Hall.

Estructura de Datos Teoría y Problemas. Lipschuts, Seymour. Mc Graw Hill.

Análisis de Algoritmos y Tecnología de Grafos. Abellanas, Lodares. Macrobit.

Estructura de Datos y Organización de Archivos. Loomis, Mary E. Prentice Hall.

# Anexos 1: Pruebas Rápidas

Universidad Tecnológica de Panamá

Facultad de Ingeniería en Sistemas Computacionales

Estructuras y Representación de Datos

Prueba #1

Nombre: _____ Cédula: _____ Grupo: _____

Profesor: Dr. Carlos A. Rovetto Puntos Obtenidos: _____/

I Parte: Llene los espacios.

1. Cuatro tipos de estructuras de datos primitivas son: _____, _____, _____ y _____.
2. Clase de datos que se caracteriza por su organización y operaciones definidas sobre ella: _____.
3. Es un conjunto finito ordenado de elementos homogéneos: _____.
4. Es la unidad mínima de información significativa para alguien: _____.
5. Mencione dos tipos de estructuras de datos simple: _____ y _____.
6. Mencione tres tipos de estructuras de datos lineales: _____, _____ y _____.

II Parte: Desarrollo.

1. Mencione los dos requisitos necesarios para la implementación de cualquier estructura de datos.

Universidad Tecnológica de Panamá

Facultad de Ingeniería en Sistemas Computacionales

Estructuras y Representación de Datos

Prueba #2

Nombre: _____ Cédula: _____ Grupo: _____

Profesor: Dr. Carlos A. Rovetto Puntos Obtenidos: _____/

I Parte: Cierto y Falso.

1. _____ Las posiciones de los elementos en un arreglo se designan con índices.
2. _____ A la posición de memoria de la primera celda de un arreglo se le denomina cota inferior (CI).
3. _____ Las estructuras de datos simples se construyen a partir de estructuras de datos primitivas.
4. _____ Los grafos y árboles son estructuras de datos lineales.
5. _____ Un arreglo bidimensional se puede ver de forma lógica como una tabla con filas y columnas.

II Parte: Desarrollo.

1. Dado el siguiente arreglo, haga su correspondiente representación en memoria.

   **Dato [1…10]**

   **Tipo: Entero**

   **Base: 20**

2. Dado el siguiente arreglo:

   **Facturas (2,3)**

   **Tipo: Real**

   **Base: 100**

   Encuentre:

   a) Función de acceso por orden principal de filas.

   b) Función de acceso en Facturas (2,2).

Universidad Tecnológica de Panamá

Facultad de Ingeniería en Sistemas Computacionales

Estructuras y Representación de Datos

Prueba #3

Nombre: _____ Cédula: _____ Grupo: _____

Profesor: Dr. Carlos A. Rovetto Puntos Obtenidos: _____/

I Parte: Cierto y Falso.

1. _____ Las pilas son estructuras de datos que suelen llamarse FIFO.
2. _____ En una pila se pueden insertar y eliminar elementos sólo por uno de los extremos.
3. _____ La principal característica de las estructuras de datos lineales es que sus elementos están en secuencia.
4. _____ Existen dos tipos de expresiones que son prefija y postfija.
5. _____ Las colas son estructuras de datos que suelen llamarse LIFO.

II Parte: Desarrollo.

1. Convierta la siguiente expresión en postfija: **A – 2 * (B + 1)**

Universidad Tecnológica de Panamá

Facultad de Ingeniería en Sistemas Computacionales

Estructuras y Representación de Datos

Prueba #4

Nombre: _____ Cédula: _____ Grupo: _____

Profesor: Dr. Carlos A. Rovetto Puntos Obtenidos: _____/

I Parte: Selección múltiple.

1. Tipo de recursividad en la que un subprograma se llama a sí mismo una o más veces directamente:
   a. Condicional b. Directa c. Indirecta

2. Operación para verificar si una pila contiene algún elemento que se pueda sacar:
   a. Sacar b. PilaLlena c. PilaVacia

3. Tipo de expresión en la que los operadores se encuentran al principio de la expresión:
   a. Postfija b. Prefija c. Infija

4. Operación para quitar elementos del frente de la cola:
   a. LimpiarCola(Cola) b. SupCola(Cola, Elemento) c. ColaVacía(Cola)

5. Tipo de recursividad en la que se definen una serie de subprogramas que se llaman unos a otros:
   a. Condicional b. Directa c. Indirecta

II Parte: Desarrollo.

1. Defina qué es un registro y cómo se diferencia este de un arreglo?

Universidad Tecnológica de Panamá

Facultad de Ingeniería en Sistemas Computacionales

Estructuras y Representación de Datos

Prueba #5

Nombre: _____ Cédula: _____ Grupo: _____

Profesor: Dr. Carlos A. Rovetto Puntos Obtenidos: _____/

I Parte: Llene los espacios.

1. Existen dos tipos de representación de una lista que son: _____ y _____.
2. Las listas pueden ordenarse por su _____, _____, _____ o _____.
3. Nombre de la representación de lista en la que el orden de los elementos está determinado por un campo enlace explícito en cada elemento: _____.
4. Nombre de la representación de lista en la que el sucesor de cada elemento de la lista está implícito por la posición del elemento: _____.
5. Las listas se clasifican en: _____, _____ y _____.

II Parte: Desarrollo.

1. ¿Por qué las listas enlazadas se consideran un tipo de estructura de datos dinámicas lineales?

Universidad Tecnológica de Panamá

Facultad de Ingeniería en Sistemas Computacionales

Estructuras y Representación de Datos

Prueba #6

Nombre: _____ Cédula: _____ Grupo: _____

Profesor: Dr. Carlos A. Rovetto Puntos Obtenidos: _____/

I Parte: Cierto y Falso.

1. _____ En una lista doblemente enlazada el último nodo se enlaza con el primero.
2. _____ Durante la compilación, a una lista se le asigna un espacio de memoria que varía durante la ejecución del programa.
3. _____ Una lista enlazada circular es aquella en la que cada nodo tiene dos enlaces, uno al nodo siguiente y otro al anterior.
4. _____ La forma en que se ordena una lista afectará su función de acceso.
5. _____ Una lista enlazada se define como una colección de elementos homogéneos y entre los que hay una relación lineal.

II Parte: Desarrollo.

1. Mencione dos características de las listas enlazadas.

Universidad Tecnológica de Panamá

Facultad de Ingeniería en Sistemas Computacionales

Estructuras y Representación de Datos

Prueba #7

Nombre: _____ Cédula: _____ Grupo: _____

Profesor: Dr. Carlos A. Rovetto Puntos Obtenidos: _____/

I Parte: Selección múltiple.

1. Tipo de árbol en el que el hijo izquierdo, si existe, debe tener un valor menor que el valor de su padre y el hijo derecho, si existe, debe tener un valor mayor que el valor de su padre:
   a. Árbol binario b. Árbol binario de búsqueda c. Árbol general

2. Dos árboles binarios que son similares y contienen la misma información se denominan:
   a. Árboles binarios similares
   b. Árboles binarios completos
   c. Árboles binarios equivalentes

3. Un árbol es una estructura de datos de tipo:
   a. Lineal y dinámica b. Lineal y no dinámica c. No lineal y dinámica

II Parte: Llene los espacios con el concepto correcto.

1. Es el primer nodo del árbol: _____.
2. Secuencia de aristas a través de las cuales se pasa desde el nodo raíz a un nodo: _____.
3. Camino que termina en una hoja: _____.
4. Distancia desde la raíz en la que se encuentra ubicado cada nodo: _____.
5. Longitud del camino más largo que comienza en el nodo y termina en una hoja: _____.
6. Longitud del camino único que comienza en la raíz y termina en el nodo: _____.

Universidad Tecnológica de Panamá

Facultad de Ingeniería en Sistemas Computacionales

Estructuras y Representación de Datos

Prueba #8

Nombre: _____ Cédula: _____ Grupo: _____

Profesor: Dr. Carlos A. Rovetto Puntos Obtenidos: _____/

I Parte: Llene los espacios.

1. Un grafo _____ es aquel en el que es posible formar un camino desde cualquier vértice a cualquier otro en el grafo.
2. Un grafo _____ es aquel cuyos vértices tienen el mismo grado.
3. Un grafo _____ es aquel que contiene algún ciclo simple.
4. Un grafo _____ es aquel en el que la relación entre los nodos es desordenada.
5. Un grafo _____ es aquel en el que cada arista transporta un valor.
6. Un grafo se puede representar secuencialmente de dos formas: _____ y _____.

II Parte: Desarrollo.

1. Mencione los dos tipos de recorridos en un grafo y en qué consiste cada uno.

# Anexos 2: Presentaciones


1

# Capítulo I: Estructura de datos fundamentales y lineales

2

## Capítulo I: Estructura de datos fundamentales y lineales

Existen dos conceptos que son importantes conocer antes de iniciar este capítulo, estos son la definición de datos y de estructura de datos. El dominio de estas definiciones, les permitirán comprender la forma en la cual se desarrollarán los problemas relacionados con los mismos.

a) **Datos:** Es la mínima unidad de información significativa para alguien. Es la materia prima para la obtención de la información.
b) **Estructura de datos:** Es una clase de datos que se puede caracterizar por su organización y operaciones definidas sobre de ella. Algunas veces a estas estructuras se les llama tipos de datos.

3

- **Estructuras lógicas de datos:** En un programa, cada variable pertenece a alguna estructura de datos la cual determina el conjunto de operaciones válidas para ella.
- **Estructuras primitivas y simples:** Las estructuras primitivas no están compuestas por otras estructuras de datos, por ejemplo: enteros, booleanos y caracteres.
- **Estructuras lineales y no-lineales:** Las estructuras de datos simples se pueden combinar de varias maneras para formar estructuras más complejas.

Las estructuras de datos se clasifican de la siguiente manera:

4

## 1.1. Estructura de datos primitivas y simples

**1. Primitivas:** A continuación se describen los tipos de estructuras de datos primitivas:

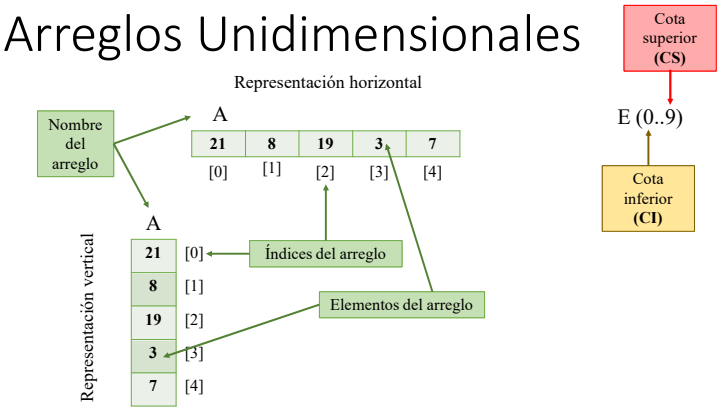
- **Enteros:** Un entero es un miembro del siguiente conjunto de números: {..., -(n+1), -n, ..., -2, -1, 0, 1, 2, ...., n, (n+1), ...} .
- **Reales:** Son aquellos valores que poseen dígitos enteros y decimales. Pueden ser valores positivos o negativos. Por ejemplo 12.47, 0.12, $6x\ 10^{-3}$, $7x\ 10^{5}$.
- **Carácter:** Es un elemento tomado de un conjunto de símbolos, en el cual se incluyen dígitos, los caracteres del alfabeto y algunos caracteres especiales.
- **Booleanos:** A esta estructura de datos primitiva también se le llama lógico. Un dato booleano es un elemento que puede tener uno de dos valores: verdadero ó falso. Los tres operadores booleanos básicos son not (no), and (y) y or(o).

**2. Simples:** Los distintos tipos de estructuras de datos simples se describen a continuación:

- **Cadenas:** Una cadena (string en inglés) de caracteres es una sucesión de caracteres que se encuentran delimitados por una comilla o dobles comillas, según el tipo de lenguaje de programación.
- **Arreglos:** Un arreglo es un conjunto finito ordenado de elementos homogéneos. La propiedad de ordenación significa que es posible identificar el primero, segundo, tercero, ... y el enésimo elemento del arreglo.
  - **a. Arreglos unidimensionales:** La forma más simple de un arreglo es el arreglo unidimensional, conocido como vector.

5

## Arreglos Unidimensionales




6

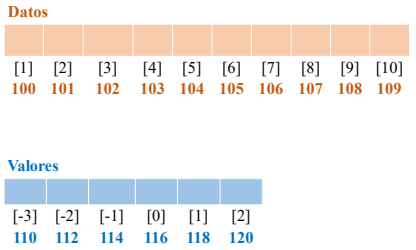
## Arreglo Unidimensional

En el caso de un arreglo unidimensional, las sentencias de declaración le dicen al compilador cuántas celdas se necesitan para representar el arreglo. El nombre del arreglo se asocia entonces con unas características del mismo. Estas características incluyen:

- La cota superior del rango índice (**CS**)
- La cota inferior del rango índice (**CI**)
- La posición de memoria de la primera celda del arreglo llamada la dirección base del arreglo (Base)
- El número de posiciones de memoria necesitadas para cada elemento del arreglo (**Tamaño**)

7

## Arreglos Unidimensionales




8

## Las fórmulas para utilizar son las siguientes:

Para calcular el **número de celdas** necesitadas:

**Celdas = (CS - CI + 1) * tamaño**

El **tamaño** de los elementos es:

**Entero = 1**

**Real = 2**

**Carácter = 1**

La **función de acceso** que nos da la posición de un elemento en un arreglo unidimensional asociado con la expresión índice es:

**Dirección (índice) = Base + Desplazamiento por índice**

En donde,

**Desplazamiento por índice = (índice - CI) * tamaño**

Así la función de acceso quedará de la siguiente forma:

**Dirección (índice) = Base + [(índice - CI) * tamaño]**

9

# Arreglos Bidimensionales

Es un tipo de datos estructurado formado por una colección finita, de tamaño fijo de elementos homogéneos ordenados. Un par de índices específica el elemento deseado dando su posición relativa.

Un arreglo bidimensional es una forma natural de representar datos que puede verse de forma lógica como una tabla con filas y columnas

10

# Arreglos Bidimensionales

***Tabla***

| | 1 | 2 | 3 |
|---|---|---|---|
| 1 | (1,1) | (1,2) | (1,3) |
| 2 | (2,1) | (2,2) | (2,3) |

Elementos de la tabla

Arreglo ventas almacenado por orden principal de fila

| [1,1] | [1,2] | [2,1] | [2,2] | [3,1] | [3,2] |
|---|---|---|---|---|---|
| 200 | 202 | 204 | 206 | 208 | 210 |

Arreglo ventas almacenado por orden principal de columna

| [1,1] | [2,1] | [3,1] | [1,2] | [2,2] | [3,2] |
|---|---|---|---|---|---|
| 200 | 202 | 204 | 206 | 208 | 210 |

11

## Las fórmulas para utilizar son las siguientes:

Para calcular el número de celdas necesitadas

***# Celdas = [(CS fila - CI fila + 1) * (CS columna - CI columna + 1)] * tamaño***

El tamaño de los elementos es:

**Entero = 1**

**Real = 2**

**Carácter = 1**

La función de acceso que nos da la posición de un elemento en un arreglo bidimensional asociado con la expresión índice (fila, columna). El arreglo puede estar ordenado en una de las siguientes formas:

**Ordenado en orden principal por fila es:**

***Dirección [elemento (fila, columna)] = Base + tamaño * [(CS columna - CI Columna + 1) * (fila - CI fila) + (columna - CI columna)]***

**Ordenado en orden principal por columna es:**

***Dirección [elemento (fila, columna)] = Base + tamaño * [(columna - CI columna) * (CS fila - CI fila + 1) + (fila - CI fila)]***

12

# Registros

Un registro es una colección finita y ordenada de elementos, posiblemente heterogéneos, que se tratan como una unidad. Un registro se distingue de un arreglo en el hecho de que todos los elementos de un arreglo deben tener la misma estructura, a diferencia de los elementos componentes del registro que pueden tener diferentes estructuras de datos. Un registro se menciona algunas veces sólo como una estructura.

**Campo:** El acceso se hace directamente mediante un conjunto de selectores de campos con nombres.

Sintaxis de la función de acceso: ***Variable registro.selector de campo.***

13

# Registro - Empleado


Ejemplo del registro empleado.

| Campo | Nombre | Cédula | Dirección | Teléfono | Salario |
|---|---|---|---|---|---|
| **Longitud** | 30 | 1 | 15 | 1 | 2 |
| **Dirección del campo** | 200 | 230 | 231 | 246 | 247 |

14

# Estructura de datos lineales y recursividad

Las estructuras lineales de datos se caracterizan porque sus elementos están en secuencia, relacionados en forma lineal, uno luego del otro. Cada elemento de la estructura puede estar conformado por uno o varios subelementos o campos que pueden pertenecer a cualquier tipo de dato, pero que normalmente son tipos básicos.

1. **Pila:** Una pila es una lista de elementos en la que se pueden insertar y eliminar elementos sólo por uno de los extremos. Como consecuencia, los elementos de una pila serán eliminados en orden inverso al que se insertaron. Es decir, el último elemento que se metió a la pila será el primero en salir de ella. Debido al orden en que se insertan y eliminan los elementos en una pila, a ésta también se le conoce como estructura LIFO (Last In, First Out: último en entrar, primero en salir).

15

# OPERACIONES SOBRE PILAS

16

- ***Meter:*** operación que añade un elemento por la cabeza de la pila.
- ***Sacar:*** operación que toma el elemento de la cabeza de la pila y lo quita.
- ***LimpiarPila:*** operación para crear una pila vacía (una pila que no contiene elementos).
- ***PilaVacía:*** operación para verificar si una pila contiene algún elemento que podamos sacar.
- ***PilaLlena:*** operación para verificar si una pila está llena antes de meter un elemento.

17

```
Procedimiento LimpiarPila
Inicio
        Pila[cabeza] = 0
Fin
```


18

```
Función PilaLlena
Inicio
Si (Pila[cabeza] = maxPila)
   Entonces imprimir Pila llena
   Sino imprimir la pila no está llena
Fin si
Fin
```

19

```
Función PilaVacía
Inicio
Si (Pila[cabeza] = 0)
    Entonces imprimir la pila está vacía
    Sino imprimir la pila no está vacía
Fin si
Fin
```

20

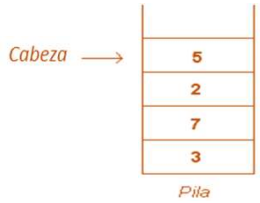
**Procedimiento Meter**

Inicio

Pila[cabeza] = Pila[cabeza] + 1

Pila[cabeza] = Nuevoelemento

Fin




21

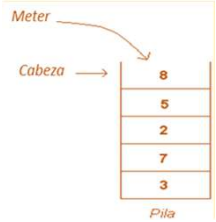
**Procedimiento Meter**

Inicio

Pila[cabeza] = Pila[cabeza] + 1

Pila[cabeza] = Nuevoelemento

Fin




22

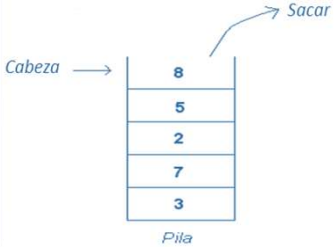
**Procedimiento Sacar**

Inicio

Elementosacado = Pila[Pila[cabeza] ]

Pila[cabeza] = Pila[cabeza] - 1

Fin




23

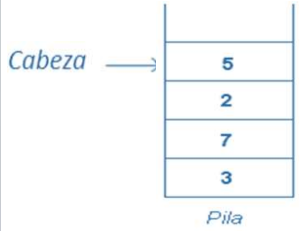
**Procedimiento Sacar**

Inicio

Elementosacado = Pila[Pila[cabeza] ]

Pila[cabeza] = Pila[cabeza] - 1

Fin




24

# IMPLEMENTACIÓN DE PILA

25

**EJEMPLO**

Limpiarpila ( Pila ) maxpila =5

X = 1 Y = 3

Z = Y * 2

Meter (Pila, Z)

Z = Y + X

Meter (Pila, Z)

Y = Z * 3 + X

Meter (Pila, Y)

Imprimir ( Pila Contiene)

Mientras no-pilavacia (Pila)
hacer

Sacar (Pila , Y)

Imprimir ( Y )

Fin – Mientras

26

## PASO 1

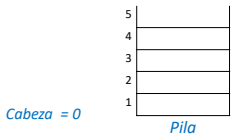
Limpiarpila ( Pila ) maxpila =5

**Procedimiento LimpiarPila**

Inicio

Pila[cabeza] = 0

Fin




27

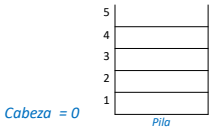
## PASO 2

Limpiarpila ( Pila ) maxpila =5

X = 1 Y = 3
Z = Y * 2

Z = 3 * 2 = 6

| X | Y | Z |
|---|---|---|
| 1 | 3 | 6 |




28

## PASO 3

Limpiarpila ( Pila ) maxpila =5

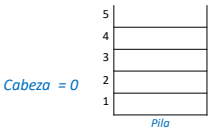
X = 1 Y = 3

Z = Y * 2

Meter (Pila, Z)

| X | Y | Z |
|---|---|---|
| 1 | 3 | 6 |

**Antes de meter un elemento a la pila hay que verificar que la misma no esté llena**




**Función PilaLlena**
Inicio
**Si** (Pila[cabeza] = maxPila)
**Entonces** imprimir Pila llena
**Sino** imprimir la pila no está llena
**Fin si**
Fin

*Cabeza = 5*

0 = 5 **FALSO**

29

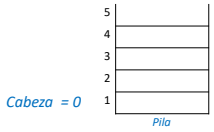
## PASO 3

Limpiarpila ( Pila ) maxpila =5

X = 1 Y = 3

Z = Y * 2

Meter (Pila, Z)




**Procedimiento Meter**
Inicio
Pila[cabeza] = Pila[cabeza] + 1
Pila[cabeza] = Nuevoelemento
Fin

| X | Y | Z |
|---|---|---|
| 1 | 3 | 6 |


30

## PASO 3

Limpiarpila ( Pila ) maxpila =5

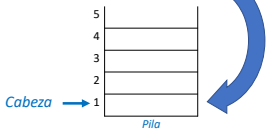
X = 1 Y = 3

Z = Y * 2

Meter (Pila, Z)

**Procedimiento Meter**
Inicio
Pila[cabeza] = Pila[cabeza] + 1
Pila[cabeza] = Nuevoelemento
Fin

| X | Y | Z |
|---|---|---|
| 1 | 3 | 6 |




31

## PASO 3

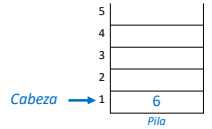
Limpiarpila ( Pila ) maxpila =5

X = 1 Y = 3

Z = Y * 2

Meter (Pila, Z)

| X | Y | Z |
|---|---|---|
| 1 | 3 | 6 |




32

## PASO 4

Limpiarpila ( Pila ) maxpila =5

X = 1 Y = 3

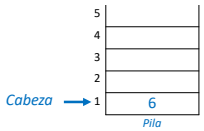
Z = Y * 2

Meter (Pila, Z)

**Z = Y + X**

**Y = 3 + 1 = 4**

| X | Y | Z |
|---|---|---|
| 1 | 3 | 6 |
| | | 4 |




33

## PASO 5

Limpiarpila ( Pila ) maxpila =5

X = 1 Y = 3

Z = Y * 2

Meter (Pila, Z)

Z = Y + X

**Meter (Pila, Z)**


***Cabeza = 5***

***1 = 5*** ***FALSO***

La pila no está llena

Podemos meter el nuevo elemento

***Cabeza = 1 + 1 = 2***

34

## PASO 5

Limpiarpila ( Pila ) maxpila =5

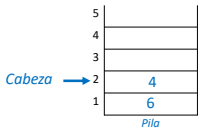
X = 1 Y = 3

Z = Y * 2

Meter (Pila, Z)

Z = Y + X

**Meter (Pila, Z)**




35

## PASO 6

Limpiarpila ( Pila ) maxpila =5

X = 1 Y = 3

Z = Y * 2

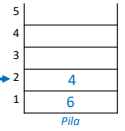
Meter (Pila, Z)

Z = Y + X

Meter (Pila, Z)

**Y = Z * 3 + X** **Y = 4 * 3 + 1 = 13**

| X | Y | Z |
|---|---|---|
| 1 | 3 | 6 |
| | | 4 |
| | 13 | |

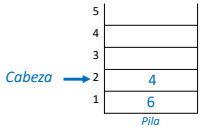



36

## PASO 7

Limpiarpila ( Pila ) maxpila =5
X = 1 Y = 3
Z = Y * 2
Meter (Pila, Z)
Z = Y + X
Meter (Pila, Z)
Y = Z * 3 + X
**Meter (Pila, Y)**

***Cabeza = 5***
***2 = 5 FALSO***
La pila no está llena



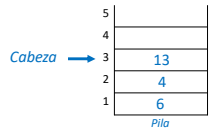

| X | Y | Z |
|---|---|---|
| 1 | 3 | 6 |
| | | 4 |
| | 13 | |

Podemos meter el nuevo elemento
***Cabeza = 2 + 1 = 3***


37

## PASO 8

Limpiarpila ( Pila ) maxpila =5
X = 1 Y = 3
Z = Y * 2
Meter (Pila, Z)
Z = Y + X
Meter (Pila, Z)
Y = Z * 3 + X
Meter (Pila, Y)
**Imprimir ( Pila Contiene)**




**Pila Contiene**

**Antes de sacar un elemento a la pila hay que verificar que la misma no esté vacía**

38

## PASO 9

Limpiarpila ( Pila ) maxpila =5
X = 1 Y = 3
Z = Y * 2
Meter (Pila, Z)
Z = Y + X
Meter (Pila, Z)
Y = Z * 3 + X
Meter (Pila, Y)
Imprimir ( Pila Contiene)
**Mientras no-pilavacia (Pila) hacer**


**Función PilaVacía**
Inicio
**Si** (Pila[cabeza] = 0)
**Entonces** imprimir la pila está vacía
**Sino** imprimir la pila no está vacía
**Fin si**
Fin

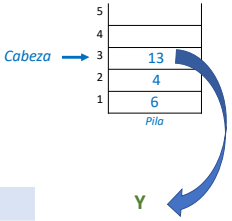
***Cabeza = 0***
***3 = 0 FALSO***
La pila no está vacía
Podemos sacar el elemento de la pila

39

## PASO 10

Limpiarpila ( Pila ) maxpila =5
X = 1 Y = 3
Z = Y * 2
Meter (Pila, Z)
Z = Y + X
Meter (Pila, Z)
Y = Z * 3 + X
Meter (Pila, Y)
Imprimir ( Pila Contiene)
Mientras no-pilavacia (Pila) hacer
**Sacar (Pila , Y)**

**Procedimiento Sacar**
Inicio
Elementosacado = Pila[Pila[cabeza] ]
Pila[cabeza] = Pila[cabeza] - 1
Fin



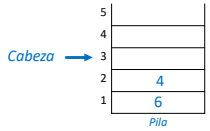

**Pila Contiene**

40

## PASO 10

Limpiarpila ( Pila ) maxpila =5
X = 1 Y = 3
Z = Y * 2
Meter (Pila, Z)
Z = Y + X
Meter (Pila, Z)
Y = Z * 3 + X
Meter (Pila, Y)
Imprimir ( Pila Contiene)
Mientras no-pilavacia (Pila) hacer
Sacar (Pila , Y)

**Procedimiento Sacar**
Inicio
Elementosacado = Pila[Pila[cabeza] ]
Pila[cabeza] = Pila[cabeza] - 1
Fin




| Pila Contiene |
|---|
| |

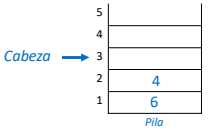
Y
13

41

## PASO 10

Limpiarpila ( Pila ) maxpila =5
X = 1 Y = 3
Z = Y * 2
Meter (Pila, Z)
Z = Y + X
Meter (Pila, Z)
Y = Z * 3 + X
Meter (Pila, Y)
Imprimir ( Pila Contiene)
Mientras no-pilavacia (Pila) hacer
Sacar (Pila , Y)

**Procedimiento Sacar**
Inicio
Elementosacado = Pila[Pila[cabeza] ]
Pila[cabeza] = Pila[cabeza] - 1
Fin




*Cabeza = 3 - 1 = 2*

| Pila Contiene |
|---|
| |

Y
13

42

## PASO 10

Limpiarpila ( Pila ) maxpila =5
X = 1 Y = 3
Z = Y * 2
Meter (Pila, Z)
Z = Y + X
Meter (Pila, Z)
Y = Z * 3 + X
Meter (Pila, Y)
Imprimir ( Pila Contiene)
Mientras no-pilavacia (Pila) hacer
Sacar (Pila , Y)

**Procedimiento Sacar**
Inicio
Elementosacado = Pila[Pila[cabeza] ]
Pila[cabeza] = Pila[cabeza] - 1
Fin

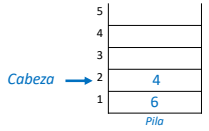

| Pila Contiene |
|---|
| |

Y
13

43

## PASO 11

Limpiarpila ( Pila ) maxpila =5
X = 1 Y = 3
Z = Y * 2
Meter (Pila, Z)
Z = Y + X
Meter (Pila, Z)
Y = Z * 3 + X
Meter (Pila, Y)
Imprimir ( Pila Contiene)
Mientras no-pilavacia (Pila) hacer
Sacar (Pila , Y)
Imprimir ( Y )
Fin – Mientras




| Pila Contiene |
|---|
| 13 |

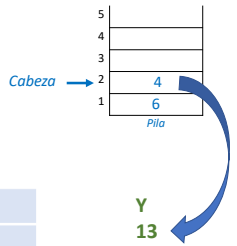
Y
13

44

## PASO 12

Limpiarpila ( Pila ) maxpila =5
X = 1 Y = 3
Z = Y * 2
Meter (Pila, Z)
Z = Y + X
Meter (Pila, Z)
Y = Z * 3 + X
Meter (Pila, Y)
Imprimir ( Pila Contiene)
Mientras no-pilavacia (Pila) hacer

*2 = 0* *FALSO*
La pila no está vacía
Podemos sacar el elemento de la pila



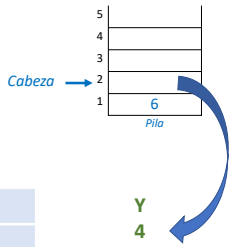

| Pila Contiene |
|---|
| 13 |

Y
13

45

## PASO 13

Limpiarpila ( Pila ) maxpila =5
X = 1 Y = 3
Z = Y * 2
Meter (Pila, Z)
Z = Y + X
Meter (Pila, Z)
Y = Z * 3 + X
Meter (Pila, Y)
Imprimir ( Pila Contiene)
Mientras no-pilavacia (Pila) hacer
Sacar (Pila , Y)



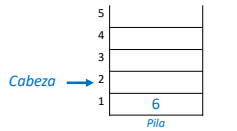

| Pila Contiene |
|---|
| 13 |

Y
4

46

## PASO 13

Limpiarpila ( Pila ) maxpila =5
X = 1 Y = 3
Z = Y * 2
Meter (Pila, Z)
Z = Y + X
Meter (Pila, Z)
Y = Z * 3 + X
Meter (Pila, Y)
Imprimir ( Pila Contiene)
Mientras no-pilavacia (Pila) hacer
Sacar (Pila , Y)




*Cabeza = 2 - 1 = 1*

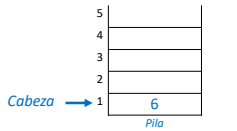
| Pila Contiene |
|---|
| 13 |

Y
4

47

## PASO 13

Limpiarpila ( Pila ) maxpila =5
X = 1 Y = 3
Z = Y * 2
Meter (Pila, Z)
Z = Y + X
Meter (Pila, Z)
Y = Z * 3 + X
Meter (Pila, Y)
Imprimir ( Pila Contiene)
Mientras no-pilavacia (Pila) hacer
Sacar (Pila , Y)



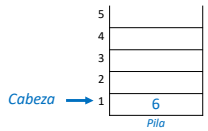

| Pila Contiene |
|---|
| 13 |

Y
4

48

## PASO 14

Limpiarpila ( Pila ) maxpila =5
X = 1 Y = 3
Z = Y * 2
Meter (Pila, Z)
Z = Y + X
Meter (Pila, Z)
Y = Z * 3 + X
Meter (Pila, Y)
Imprimir ( Pila Contiene)
Mientras no-pilavacia (Pila) hacer
Sacar (Pila , Y)
**Imprimir ( Y )**




| Pila Contiene |
|---|
| 13 |
| 4 |

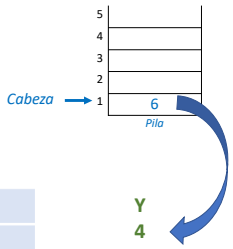
Y
4

49

## PASO 15

Limpiarpila ( Pila ) maxpila =5
X = 1 Y = 3
Z = Y * 2
Meter (Pila, Z)
Z = Y + X
Meter (Pila, Z)
Y = Z * 3 + X
Meter (Pila, Y)
Imprimir ( Pila Contiene)
**Mientras no-pilavacia (Pila) hacer**

*1 = 0* ***FALSO***
La pila no está vacía
Podemos sacar el elemento de la pila



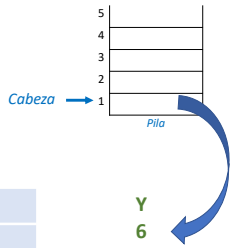

| Pila Contiene |
|---|
| 13 |
| 4 |

Y
4

50

## PASO 16

Limpiarpila ( Pila ) maxpila =5
X = 1 Y = 3
Z = Y * 2
Meter (Pila, Z)
Z = Y + X
Meter (Pila, Z)
Y = Z * 3 + X
Meter (Pila, Y)
Imprimir ( Pila Contiene)
Mientras no-pilavacia (Pila) hacer
**Sacar (Pila , Y)**



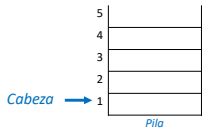

| Pila Contiene |
|---|
| 13 |
| 4 |

Y
6

51

## PASO 16

Limpiarpila ( Pila ) maxpila =5
X = 1 Y = 3
Z = Y * 2
Meter (Pila, Z)
Z = Y + X
Meter (Pila, Z)
Y = Z * 3 + X
Meter (Pila, Y)
Imprimir ( Pila Contiene)
Mientras no-pilavacia (Pila) hacer
**Sacar (Pila , Y)**




*Cabeza = 1 - 1 = 0*

| Pila Contiene |
|---|
| 13 |
| 4 |

Y
6

52

## PASO 16

```
Limpiarpila ( Pila )   maxpila =5
X = 1     Y = 3
Z = Y * 2
Meter  (Pila, Z)
Z = Y + X
Meter  (Pila, Z)
Y = Z * 3 + X
Meter  (Pila, Y)
Imprimir ( Pila Contiene)
Mientras   no-pilavacia (Pila)  hacer
  Sacar (Pila ,  Y)
```

*Cabeza = 0*


| Pila Contiene |
|---|
| 13 |
| 4 |

Y
6

53

## PASO 17

```
Limpiarpila ( Pila )   maxpila =5
X = 1     Y = 3
Z = Y * 2
Meter  (Pila, Z)
Z = Y + X
Meter  (Pila, Z)
Y = Z * 3 + X
Meter  (Pila, Y)
Imprimir ( Pila Contiene)
Mientras   no-pilavacia (Pila)  hacer
  Sacar (Pila ,  Y)
  Imprimir ( Y )
Fin – Mientras
```

*Cabeza = 0*


| Pila Contiene |
|---|
| 13 |
| 4 |
| 6 |

Y
6

54

## PASO 18

```
Limpiarpila ( Pila )   maxpila =5
X = 1     Y = 3
Z = Y * 2
Meter  (Pila, Z)
Z = Y + X
Meter  (Pila, Z)
Y = Z * 3 + X
Meter  (Pila, Y)
Imprimir ( Pila Contiene)
Mientras   no-pilavacia (Pila)  hacer
  Sacar (Pila ,  Y)
  Imprimir ( Y )
Fin – Mientras
```

*0 = 0* *CIERTO*
La pila está vacía

*Cabeza = 0*


| Pila Contiene |
|---|
| 13 |
| 4 |
| 6 |

Y
6

55

## Evaluación de expresiones

Una de las implementaciones más comunes de las pilas es en la evaluación de expresiones aritméticas formadas por operandos y operadores. En donde, los operandos son variables que pueden tomar valores enteros o reales. Por otro lado, los operadores son aquellos con los que estamos acostumbrados a trabajar: suma, resta, división, multiplicación y exponenciación y, en donde, el orden de ejecución depende del orden de precedencia que existe entre ellos.

56

## a) Conversión de una expresión infija a prefija o postfija

La diferencia al evaluar una expresión infija a prefija o postfija es la forma en la cual se escribe la expresión, esto es, debemos recordar que en una expresión prefija los operadores se encuentran al principio de la expresión mientras que en una expresión postfija los operadores se encuentran al final de la expresión.

Existen tres tipos expresiones:

- Expresión infija: ((A + 2) * (B + 4)) – 1
- Expresión prefija: - * + A 2 + B 4  1
- Expresión postfija: A 2 + B 4 + * 1 -

El orden de los operandos y el operador en cada una de estas expresiones es el siguiente:

- expresión infija: operando1 operador operando2
- expresión prefija: operador operando1 operando2
- expresión postfija: operando1 operando2 operador

57

## CONVERSIÓN DE EXPRESIONES INFIJAS A PREFIJAS O POSTFIJAS

58

1. Repetir hasta que no haya caracteres en la expresión de entrada

2. Leer un carácter de la expresión.
   - Si es un operando se coloca en la pila de operandos
   - Si es un operador se coloca en la pila de operadores

3. Si es un paréntesis:
   - Si es izquierdo se mete en la pila de operandos
   - Si es derecho se sacan de la pila de operandos todos los operandos hasta encontrar el primer paréntesis izquierdo. También se saca el operador de la pila de operadores y se evalúa la expresión en función de la conversión solicitada, esto es, para convertir a:
     - **expresión prefija:** operador operando1  operando2
     - **expresión postfija:** operando1  operando2  operador

59

La evaluación también se debe realizar si no hay paréntesis izquierdo y ha entrado un operando en la pila de operandos seguido de: ***un operador en la pila de operadores y otro operando en la pila de operandos.***

Se debe recordar que el **primer operando** que se saca de la pila de operandos corresponde al **operando2** mientras que el siguiente operando que se saca de la pila de operandos corresponde al operando1.

60

## EJEMPLO:

## CONVERSIÓN DE UNA EXPRESIÓN DE INFIJA A PREFIJA

Convierta la siguiente expresión infija en Prefija

**A + 3 * (B - 2)**

61

### PASO 1

**A + 3 * (B - 2)**

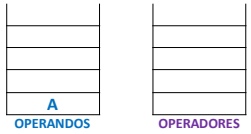
Leer un carácter de la expresión de entrada

**Caracteres leídos: A**

Es un operando, se coloca en la pila de operandos

Hay un operador en la pila de operadores

FALSO




62

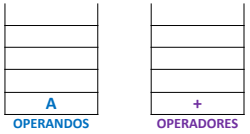
### PASO 2

**A + 3 * (B - 2)**

Leer un carácter de la expresión

**Caracteres leídos: A +**

Es un operador, se coloca en la pila de operadores




63

### PASO 3

**A + 3 * (B - 2)**

Leer un carácter de la expresión

**Caracteres leídos: A + 3**

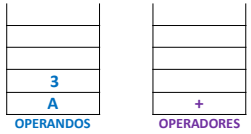
Es un operando, se coloca en la pila de operandos

Hay un operador en la pila de operadores

CIERTO

Hay dos o más operandos en la pila de operandos

CIERTO



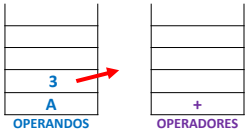

64

## PASO 4

Sacar los dos últimos operandos de la pila de operandos

**A + 3 * (B - 2)**

**Caracteres leídos: A + 3**




OPERANDO 2: **3**

65

## PASO 5

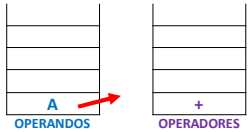
Sacar los dos últimos operandos de la pila de operandos

operando1 es igual al paréntesis izquierdo

➡ FALSO

**A + 3 * (B - 2)**

**Caracteres leídos: A + 3**




OPERANDO 2: **3**

OPERANDO 1: **A**

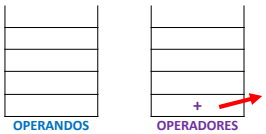
66

## PASO 6

Sacar el operador de la pila de operadores y evaluar la expresión

**A + 3 * (B - 2)**

**Caracteres leídos: A + 3**




OPERANDO 2: **3**

OPERANDO 1: **A**

OPERADOR: **+**

*Expresión prefija: operador operando1 operando2*

***+A3***

67

## PASO 7

Meter el resultado a la pila de operandos

**A + 3 * (B - 2)**

**Caracteres leídos: A + 3**


68

## PASO 8

Leer un carácter de la expresión de entrada

Es un operador, se coloca en la pila de operadores

Leer un carácter de la expresión de entrada

Si es un paréntesis izquierdo: se mete en la pila de operandos

CIERTO

Leer un carácter de la expresión de entrada

Es un operando se coloca en la pila de operandos

**A + 3 * (B - 2)**

**Caracteres leídos: A + 3 * ( B**


69

## PASO 9

Hay un operador en la pila de operadores

CIERTO

Hay dos o más operandos en la pila de operandos

CIERTO

**A + 3 * (B - 2)**

**Caracteres leídos: A + 3 * ( B**


70

## PASO 10

Sacar los dos últimos operandos de la pila de operandos

Si el operando1 es igual al paréntesis izquierdo

CIERTO

**A + 3 * (B - 2)**

**Caracteres leídos: A + 3 * ( B**


OPERANDO 2: B

OPERANDO 1: (

71

## PASO 11

Meter los operandos en la pila, en el mismo orden en que fueron sacados de la pila

**A + 3 * (B - 2)**

**Caracteres leídos: A + 3 * ( B**


72

## PASO 12

**A + 3 * (B - 2)**

**Caracteres leídos: A + 3 * ( B – 2**

Leer un carácter de la expresión de entrada

Es un operador, se coloca en la pila de operadores

Leer un carácter de la expresión de entrada

Es un operando se coloca en la pila de operandos

Hay un operador en la pila de operadores ➡ CIERTO

Hay dos o más operandos en la pila de operandos ➡ CIERTO

| OPERANDOS | OPERADORES |
|---|---|
| 2 | |
| B | |
| ( | - |
| +A3 | * |

73

## PASO 13

**A + 3 * (B - 2)**

**Caracteres leídos: A + 3 * ( B – 2**

Sacar los dos últimos operandos de la pila de operandos

¿operando1 es igual al paréntesis izquierdo? ➡ FALSO

Sacar el operador de la pila de operadores y evaluar la expresión

Meter el resultado a la pila de operandos

| OPERANDOS | OPERADORES |
|---|---|
| ( | - |
| +A3 | * |

OPERANDO 2: 2

OPERANDO 1: B

OPERADOR: -

*Expresión prefija: operador operando1 operando2*

***-B2***

74

## PASO 14

**A + 3 * (B - 2)**

**Caracteres leídos: A + 3 * ( B – 2 )**

Leer un carácter de la expresión de entrada

¿Es un paréntesis derecho? ➡ CIERTO

Sacar los dos últimos operandos de la pila de operandos

| OPERANDOS | OPERADORES |
|---|---|
| -B2 | |
| ( | |
| +A3 | * |

75

## PASO 15

**A + 3 * (B - 2)**

**Caracteres leídos: A + 3 * ( B – 2 )**

Si el operando1 es igual al paréntesis izquierdo ➡ CIERTO

Meter el operando2 en la pila de operandos

| OPERANDOS | OPERADORES |
|---|---|
| +A3 | * |

OPERANDO 2: -B2

OPERANDO 1: (

76

## PASO 16

Hay un operador en la pila de operadores

CIERTO

Hay dos o más operandos en la pila de operandos

CIERTO

Sacar los dos últimos operandos de la pila de operandos

**A + 3 * (B - 2)**

**Caracteres leídos: A + 3 * ( B – 2 )**


77

## PASO 17

Sacar los dos últimos operandos de la pila de operandos

¿operando1 es igual al paréntesis izquierdo?

FALSO

Sacar el operador de la pila de operadores y evaluar la expresión

Meter el resultado a la pila de operandos

No hay más caracteres en la expresión de entrada

**A + 3 * (B - 2)**

**Caracteres leídos: A + 3 * ( B – 2 )**


OPERANDO 2: -B2

OPERANDO 1: +A3

OPERADOR: *

*Expresión prefija: operador operando1 operando2*

***+A3-B2***

78

# Postfija

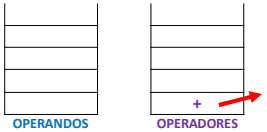
79

## PASO 6

Sacar el operador de la pila de operadores y evaluar la expresión

**A + 3 * (B - 2)**

**Caracteres leídos: A + 3**




OPERANDO 2: 3

OPERANDO 1: A

OPERADOR: +

*Expresión postfija: operando1 operando2 operador*

***A3+***

80

## PASO 7

Meter el resultado a la pila de operandos

**A + 3 * (B - 2)**

**Caracteres leídos: A + 3**


81

## PASO 8

Leer un carácter de la expresión de entrada

Es un operador, se coloca en la pila de operadores

Leer un carácter de la expresión de entrada

Si es un paréntesis izquierdo: se mete en la pila de operandos

CIERTO

Leer un carácter de la expresión de entrada

Es un operando se coloca en la pila de operandos

**A + 3 * (B - 2)**

**Caracteres leídos: A + 3 * ( B**


82

## PASO 9

Hay un operador en la pila de operadores

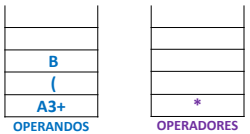
CIERTO

Hay dos o más operandos en la pila de operandos

CIERTO

**A + 3 * (B - 2)**

**Caracteres leídos: A + 3 * ( B**




83

## PASO 10

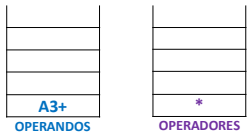
Sacar los dos últimos operandos de la pila de operandos

Si el operando1 es igual al paréntesis izquierdo

CIERTO

**A + 3 * (B - 2)**

**Caracteres leídos: A + 3 * ( B**




OPERANDO 2: B

OPERANDO 1: (

84

## PASO 11

Meter los operandos en la pila, en el mismo orden en que fueron sacados de la pila

**A + 3 * (B - 2)**

**Caracteres leídos: A + 3 * ( B**


85

## PASO 12

Leer un carácter de la expresión de entrada

Es un operador, se coloca en la pila de operadores

Leer un carácter de la expresión de entrada

Es un operando se coloca en la pila de operandos

Hay un operador en la pila de operadores ➡ CIERTO

Hay dos o más operandos en la pila de operandos ➡ CIERTO

**A + 3 * (B - 2)**

**Caracteres leídos: A + 3 * ( B – 2**


86

## PASO 13

Sacar los dos últimos operandos de la pila de operandos

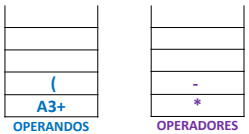
¿operando1 es igual al paréntesis izquierdo? ➡ FALSO

Sacar el operador de la pila de operadores y evaluar la expresión

Meter el resultado a la pila de operandos

**A + 3 * (B - 2)**

**Caracteres leídos: A + 3 * ( B – 2**




OPERANDO 2: **2**

OPERANDO 1: **B**

OPERADOR: **-**

*Expresión postfija: operando1 operando2 operador*

***B2 -***

87

## PASO 14

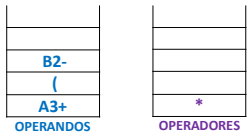
Leer un carácter de la expresión de entrada

¿Es un paréntesis derecho? ➡ CIERTO

Sacar los dos últimos operandos de la pila de operandos

**A + 3 * (B - 2)**

**Caracteres leídos: A + 3 * ( B – 2 )**




88

## PASO 15

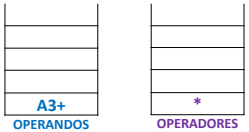
Si el operando1 es igual al paréntesis izquierdo

CIERTO

Meter el operando2 en la pila de operandos

**A + 3 * (B - 2)**

**Caracteres leídos: A + 3 * ( B – 2 )**




OPERANDO 2: B2-

OPERANDO 1: (

89

## PASO 16

Hay un operador en la pila de operadores

CIERTO

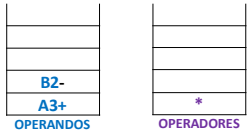
Hay dos o más operandos en la pila de operandos

CIERTO

Sacar los dos últimos operandos de la pila de operandos

**A + 3 * (B - 2)**

**Caracteres leídos: A + 3 * ( B – 2 )**




90

## PASO 17

Sacar los dos últimos operandos de la pila de operandos

¿operando1 es igual al paréntesis izquierdo?

FALSO

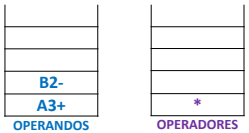
Sacar el operador de la pila de operadores y evaluar la expresión

Meter el resultado a la pila de operandos

No hay más caracteres en la expresión de entrada

**A + 3 * (B - 2)**

**Caracteres leídos: A + 3 * ( B – 2 )**




OPERANDO 2: B2-

OPERANDO 1: A3+

OPERADOR: *

*Expresión postfija: operando1 operando2 operador*

***A3+B2-****

91

## b) Evaluación de una expresión prefija

92

**Pasos para la evaluación de expresiones prefijas.**

- Leer un carácter de la expresión hasta que no haya caracteres en la expresión de entrada

1.1. Si es un operador se coloca en la pila de operadores. **Ir al paso 1.**

1.2. Si es un operando se coloca en la pila de operandos, verificar

1. **Si hay dos operandos precedidos por un operador, entonces:** sacar los dos últimos operandos de la pila de operandos y el operador de la pila de operadores. Evaluar la expresión de la siguiente forma: operando1 operador operando2. Meter el resultado a la pila de operandos. **Ir al paso 1.**
1. **Sino: Ir al paso 1.**

93

## EJEMPLO:

## EVALUACIÓN DE UNA EXPRESIÓN PREFIJA

Evaluar la siguiente expresión prefija

- * 4 5 + 10 8

94

## PASO 1

Leer un carácter de la expresión de entrada

Es un operador, se coloca en la pila de operadores

Leer un carácter de la expresión de entrada

Es un operador, se coloca en la pila de operadores

Leer un carácter de la expresión de entrada

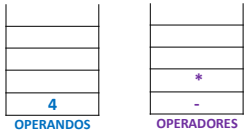
Es un operando, se coloca en la pila de operandos

¿Hay dos operandos precedidos de un operador?

FALSO

**- * 4 5 + 10 8**

**Caracteres leídos: - * 4**




95

## PASO 2

Leer un carácter de la expresión de entrada

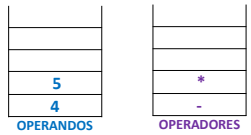
Es un operando, se coloca en la pila de operandos

¿Hay dos operandos precedidos de un operador?

CIERTO

**- * 4 5 + 10 8**

**Caracteres leídos: - * 4 5**



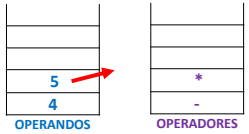

96

## PASO 3

Sacar los dos últimos operandos de la pila de operandos

**- * 4 5 + 10 8**

**Caracteres leídos: - * 4 5**




OPERANDO 2: 5

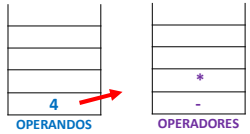
97

## PASO 4

Sacar los dos últimos operandos de la pila de operandos

**- * 4 5 + 10 8**

**Caracteres leídos: - * 4 5**




OPERANDO 2: 5

OPERANDO 1: 4

98

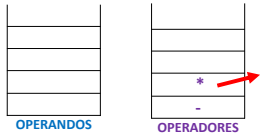
## PASO 5

Sacar el operador de la pila de operadores

Evaluar la expresión

**- * 4 5 + 10 8**

**Caracteres leídos: - * 4 5**




OPERANDO 2: 5

OPERANDO 1: 4

OPERADOR: *

*Expresión prefija: operando1 operador operando2*

***4 * 5 = 20***

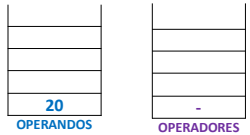
99

## PASO 6

Meter el resultado a la pila de operandos

**- * 4 5 + 10 8**

**Caracteres leídos: - * 4 5**




100

## PASO 7

Leer un carácter de la expresión de entrada

Es un operador, se coloca en la pila de operadores

Leer un carácter de la expresión de entrada

Es un operando, se coloca en la pila de operandos

¿Hay dos operandos precedidos de un operador?

FALSO

**- * 4 5 + 10 8**

**Caracteres leídos: - * 4 5 + 10**


101

## PASO 8

Leer un carácter de la expresión de entrada

Es un operando, se coloca en la pila de operandos

¿Hay dos operandos precedidos de un operador?

CIERTO

**- * 4 5 + 10 8**

**Caracteres leídos: - * 4 5 + 10 8**


102

## PASO 9

Sacar los dos últimos operandos de la pila de operandos

**- * 4 5 + 10 8**

**Caracteres leídos: - * 4 5 + 10 8**


OPERANDO 2: **8**

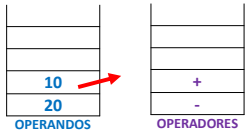
103

## PASO 10

Sacar los dos últimos operandos de la pila de operandos

**- * 4 5 + 10 8**

**Caracteres leídos: - * 4 5 + 10 8**




OPERANDO 2: **8**

OPERANDO 1: **10**

104

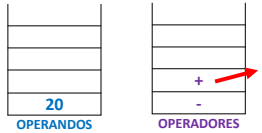
## PASO 11

Sacar el operador de la pila de operadores

Evaluar la expresión

**- * 4 5 + 10 8**

**Caracteres leídos: - * 4 5 + 10 8**




OPERANDO 2: 8

OPERANDO 1: 10

OPERADOR: +

*Expresión prefija: operando1 operador operando2*

***10 + 8 = 18***

105

## PASO 12

Meter el resultado a la pila de operandos

¿Hay dos operandos precedidos de un operador?

CIERTO

**- * 4 5 + 10 8**

**Caracteres leídos: - * 4 5 + 10 8**

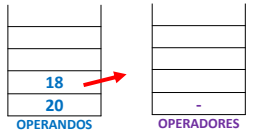

106

## PASO 13

Sacar los dos últimos operandos de la pila de operandos

**- * 4 5 + 10 8**

**Caracteres leídos: - * 4 5 + 10 8**




OPERANDO 2: 18

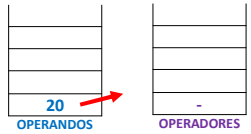
107

## PASO 14

Sacar los dos últimos operandos de la pila de operandos

**- * 4 5 + 10 8**

**Caracteres leídos: - * 4 5 + 10 8**




OPERANDO 2: 18

OPERANDO 1: 20

108

**PASO 15**

Sacar el operador de la pila de operadores

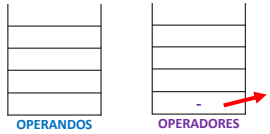
Evaluar la expresión

Meter el resultado a la pila de operandos

No hay más caracteres en la expresión de entrada

**- * 4 5 + 10 8**

**Caracteres leídos: - * 4 5 + 10 8**




OPERANDO 2: 18

OPERANDO 1: 20

OPERADOR: -

*Expresión prefija: operando1 operador operando2*

***20 - 18 = 2***

109

## c) Evaluación de una expresión postfija

110

**Pasos para la evaluación de expresiones postfijas.**

- Leer un carácter de la expresión hasta que no haya caracteres en la expresión de entrada
  - Si es un operando se coloca en la pila de operandos. Ir al paso 1.
  - Si es un operador se coloca en la pila de operadores, verificar
- Si hay dos operandos seguidos de un operador, entonces: sacar los dos últimos operandos de la pila de operandos y el operador de la pila de operadores. Evaluar la expresión de la siguiente forma: operando1 operador operando2. Meter el resultado a la pila de operandos. Ir al paso 1.

1. Sino: Ir al paso 1.

111

## EJEMPLO:

## EVALUACIÓN DE UNA EXPRESIÓN POSTFIJA

Evaluar la siguiente expresión postfija

**4 5 * 10 8 + -**

112

## PASO 1

Leer un carácter de la expresión de entrada

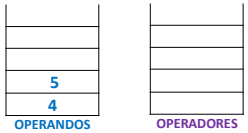
Es un operando, se coloca en la pila de operandos

Leer un carácter de la expresión de entrada

Es un operando, se coloca en la pila de operandos

**4 5 * 10 8 + -**

**Caracteres leídos: 4 5**




113

## PASO 2

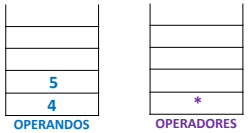
Leer un carácter de la expresión de entrada

Es un operador, se coloca en la pila de operadores

¿Hay dos operandos seguidos de un operador? ➡ CIERTO

**4 5 * 10 8 + -**

**Caracteres leídos: 4 5 ***



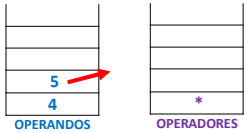

114

## PASO 3

Sacar los dos últimos operandos de la pila de operandos

**4 5 * 10 8 + -**

**Caracteres leídos: 4 5 ***




OPERANDO 2: 5

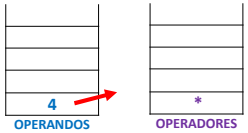
115

## PASO 4

Sacar los dos últimos operandos de la pila de operandos

**4 5 * 10 8 + -**

**Caracteres leídos: 4 5 ***




OPERANDO 2: 5

OPERANDO 1: 4

116

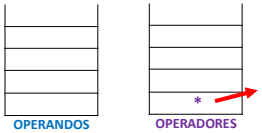
## PASO 5

Sacar el operador de la pila de operadores

Evaluar la expresión

**4 5 * 10 8 + -**

**Caracteres leídos: 4 5 ***




OPERANDO 2: 5

OPERANDO 1: 4

OPERADOR: *

*Expresión prefija: operando1 operador operando2*

***4 * 5 = 20***

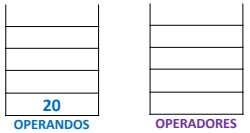
117

## PASO 6

Meter el resultado a la pila de operandos

**4 5 * 10 8 + -**

**Caracteres leídos: 4 5 ***




118

## PASO 7

Leer un carácter de la expresión de entrada

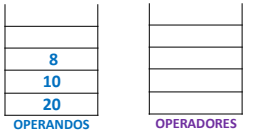
Es un operando, se coloca en la pila de operandos

Leer un carácter de la expresión de entrada

Es un operando, se coloca en la pila de operandos

**4 5 * 10 8 + -**

**Caracteres leídos: 4 5 * 10 8**




119

## PASO 8

Leer un carácter de la expresión de entrada

Es un operador, se coloca en la pila de operadores

¿Hay dos operandos seguidos de un operador? CIERTO

**4 5 * 10 8 + -**

**Caracteres leídos: 4 5 * 10 8 +**


120

## PASO 9

Sacar los dos últimos operandos de la pila de operandos

**4 5 * 10 8 + -**

**Caracteres leídos: 4 5 * 10 8 +**


OPERANDO 2: 8

121

## PASO 10

Sacar los dos últimos operandos de la pila de operandos

**4 5 * 10 8 + -**

**Caracteres leídos: 4 5 * 10 8 +**


OPERANDO 2: 8

OPERANDO 1: 10

122

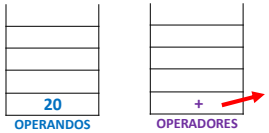
## PASO 11

Sacar el operador de la pila de operadores

Evaluar la expresión

**4 5 * 10 8 + -**

**Caracteres leídos: 4 5 * 10 8 +**




OPERANDO 2: 8

OPERANDO 1: 10

OPERADOR: +

*Expresión prefija: operando1 operador operando2*

***10 + 8 = 18***

123

## PASO 12

Meter el resultado a la pila de operandos

**4 5 * 10 8 + -**

**Caracteres leídos: 4 5 * 10 8 +**


124

## PASO 13

Leer un carácter de la expresión de entrada

Es un operador, se coloca en la pila de operadores

¿Hay dos operandos seguidos de un operador? CIERTO

**4 5 * 10 8 + -**

**Caracteres leídos: 4 5 * 10 8 + -**


125

## PASO 14

Sacar los dos últimos operandos de la pila de operandos

**4 5 * 10 8 + -**

**Caracteres leídos: 4 5 * 10 8 + -**


OPERANDO 2: 18

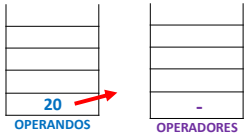
126

## PASO 15

Sacar los dos últimos operandos de la pila de operandos

**4 5 * 10 8 + -**

**Caracteres leídos: 4 5 * 10 8 + -**




OPERANDO 2: 18

OPERANDO 1: 20

127

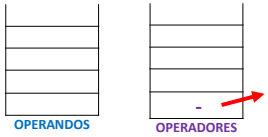
## PASO 16

Sacar el operador de la pila de operadores

Evaluar la expresión

**4 5 * 10 8 + -**

**Caracteres leídos: 4 5 * 10 8 + -**




OPERANDO 2: 18

OPERANDO 1: 20

OPERADOR: -

*Expresión prefija: operando1 operador operando2*

***10 - 18 = 2***

128

**PASO 17**

Sacar el operador de la pila de operadores

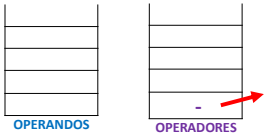
Evaluar la expresión

Meter el resultado a la pila de operandos

No hay más caracteres en la expresión de entrada

**4 5 * 10 8 + -**

**Caracteres leídos: 4 5 * 10 8 + -**




OPERANDO 2: **18**

OPERANDO 1: **20**

OPERADOR: **-**

*Expresión prefija: operando1 operador operando2*

***10 - 18 = 2***

129

## Colas

Una cola es un grupo ordenado de elementos homogéneos en el que los nuevos elementos se añaden por un extremo de la cola (el final) y se quitan por el otro extremo de la cola (el frente).

Una cola es una estructura "primero en entrar, primero en salir", se suele llamar FIFO (First Input First Output).

A diferencia de las pilas, una cola está abierta en ambos extremos. Un extremo siempre se usa para insertar datos (poner en cola) y el otro se usa para eliminar datos (quitar de cola).


130

## EJEMPLO DE COLA

**SE QUITA POR EL FRENTE**

**SE AÑADE POR EL FINAL**


131

# OPERACIONES SOBRE COLA

132

***InsCola(Cola, NuevoElemento):*** operación para añadir nuevos elementos al final de la cola.

***SupCola. SupCola(Cola, Elemento):*** operación para quitar elementos del frente de la cola

***ColaVacía(Cola):*** operación para verificar si la cola está vacía.

***ColaLlena (Cola):*** operación para verificar si la cola está llena.

***LimpiarCola(Cola):*** operación para inicializar una cola a un estado vacío.

133

# IMPLEMENTACIÓN DE COLA

134

```
Procedimiento LimpiarCola
Inicio
        Cola.Frente = MaxCola
        Cola.Final = MaxCola
Fin
```

```
Función ColaVacía
Inicio
Si (Cola.Final = Cola.Frente)
    Entonces Imprimir ("La cola está vacía")
    Sino Imprimir ("La cola no está vacía")
Fin si
Fin
```

135

```
Función ColaLlena
/* Encuentra la siguiente posición disponible del
final */
Inicio
Si (Cola.Final = MaxCola)
        Entonces SigFinal = 1
        Sino SigFinal = Cola.Final + 1
Fin si

/* La cola está llena si la siguiente posición
disponible del final es igual al frente de la cola */

Si (SigFinal = Cola.Frente)
        Entonces Imprimir ("La cola está llena")
        Sino Imprimir ("La cola no está llena")
Fin si
Fin
```

136

**Procedimiento InsCola**
/* Añade NuevoElemento al final de la cola.
Supone que la cola no está llena */

Inicio
Si (Cola.Final = MaxCola)
Entonces Cola.Final = 1
Sino Cola.Final = Cola.Final + 1
Fin si

/* Añade un nuevo elemento al final de la cola */

Cola.Elementos[Cola.Final] = NuevoElemento
Fin

**Procedimiento SupCola**
/* Quita el elemento frente de Cola y lo devuelve en ElemSuprimido. Supone que la cola no está vacía */

Inicio
Si (Cola.Frente = MaxCola)
Entonces Cola.Frente = 1
Sino Cola.Frente = Cola.Frente + 1
Fin si

/* Asigna el elemento frente de la cola a ElemSuprimido */

SupColaElemento =
Cola.Elementos[Cola.Frente]
Fin

137

# EJEMPLO

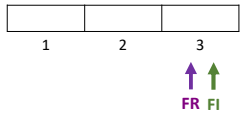
Limpiarcola (Cola) maxcola =3
X = 0 Y = 1
R = X + Y
Mientras (R < 10) Y (No Colallena)
Inscola (Cola, R)
X = Y
Y = R
R = X + Y
Fin – Mientras
Imprimir ( Cola Contiene)
Mientras no-colavacia (Cola ) hacer
Supcola (Cola, Y)
Imprimir ( Y )
Fin – Mientras

138

## PASO 1

Limpiarcola (Cola) maxcola =3

**Procedimiento LimpiarCola**
Inicio
Cola.Frente = MaxCola
Cola.Final = MaxCola
Fin




139

## PASO 2

Limpiarcola (Cola) maxcola =5
X = 0 Y = 1
R = X + Y

| X | Y | R |
|---|---|---|
| 0 | 1 | 1 |

140

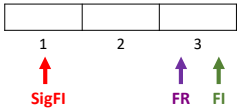
## PASO 3

Limpiarcola (Cola) maxcola =5
X = 0 Y = 1
R = X + Y
Mientras (R < 10) Y (No Colallena)

CIERTO CIERTO

*VERIFICAMOS QUE LA COLA NO ESTÉ LLENA*

| X | Y | R |
|---|---|---|
| 0 | 1 | 1 |




SigFI ≠ FR
LA COLA NO ESTÁ LLENA

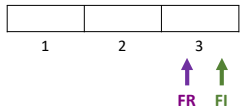
```
Función ColaLlena
/* Encuentra la siguiente posición disponible del final */
Inicio
1  Si (Cola.Final = MaxCola) → CIERTO
2      Entonces SigFinal = 1
       Sino SigFinal = Cola.Final + 1
Fin si

/* La cola está llena si la siguiente posición disponible del final
es igual al frente de la cola */

3  Si (SigFinal = Cola.Frente) → FALSO
       Entonces Imprimir ("La cola está llena")
4      Sino Imprimir ("La cola no está llena")
Fin si
Fin
```

141

## PASO 4

Limpiarcola (Cola) maxcola =5
X = 0 Y = 1 R = X + Y
Mientras (R < 10) Y (No Colallena)
Inscola (Cola, R)

| X | Y | R |
|---|---|---|
| 0 | 1 | 1 |



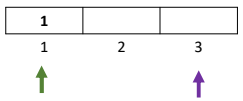

```
Procedimiento InsCola
/* Añade NuevoElemento al final de la cola. Supone que la
cola no está llena */

Inicio
Si (Cola.Final = MaxCola) → CIERTO
        Entonces Cola.Final = 1
        Sino Cola.Final = Cola.Final + 1
Fin si

/* Añade un nuevo elemento al final de la cola */

Cola.Elementos[Cola.Final] = NuevoElemento
Fin
```

142

## PASO 4

Limpiarcola (Cola) maxcola =5
X = 0 Y = 1 R = X + Y
Mientras (R < 10) Y (No Colallena)
Inscola (Cola, R)

| X | Y | R |
|---|---|---|
| 0 | 1 | 1 |



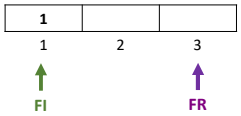

```
Procedimiento InsCola
/* Añade NuevoElemento al final de la cola. Supone que la
cola no está llena */

Inicio
Si (Cola.Final = MaxCola)
1       Entonces Cola.Final = 1
        Sino Cola.Final = Cola.Final + 1
Fin si

/* Añade un nuevo elemento al final de la cola */

2  Cola.Elementos[Cola.Final] = NuevoElemento
Fin
```

143

## PASO 5

Limpiarcola (Cola) maxcola =5
X = 0 Y = 1 R = X + Y
Mientras (R < 10) Y (No Colallena)
Inscola (Cola, R)
X = Y
Y = R
R = X + Y
Fin – Mientras

| X | Y | R |
|---|---|---|
| 0 | 1 | 1 |
| 1 | 1 | 2 |




144

## PASO 6

Limpiarcola (Cola) maxcola =5
X = 0 Y = 1 R = X + Y
Mientras (R < 10) Y (No Colallena)

CIERTO CIERTO

*VERIFICAMOS QUE LA COLA NO ESTÉ LLENA*

| 1 | | |
|---|---|---|
| 1 | 2 | 3 |

FI SigFI FR

SigFI ≠ FR

LA COLA NO ESTÁ LLENA

| X | Y | R |
|---|---|---|
| 0 | 1 | 1 |
| 1 | 1 | 2 |

**Función ColaLlena**
/* Encuentra la siguiente posición disponible del final */
Inicio
1 → Si (Cola.Final = MaxCola) → FALSO
Entonces SigFinal = 1
2 → Sino SigFinal = Cola.Final + 1
Fin si

/* La cola está llena si la siguiente posición disponible del final es igual al frente de la cola */

3 → Si (SigFinal = Cola.Frente) → FALSO
Entonces Imprimir ("La cola está llena")
4 → Sino Imprimir ("La cola no está llena")
Fin si
Fin

145

## PASO 7

Limpiarcola (Cola) maxcola =5
X = 0 Y = 1 R = X + Y
Mientras (R < 10) Y (No Colallena)
Inscola (Cola, R)

| 1 | 2 | |
|---|---|---|
| 1 | 2 | 3 |

FI FR

| X | Y | R |
|---|---|---|
| 0 | 1 | 1 |
| 1 | 1 | 2 |

**Procedimiento InsCola**
/* Añade NuevoElemento al final de la cola. Supone que la cola no está llena */

Inicio
1 → Si (Cola.Final = MaxCola) → FALSO
Entonces Cola.Final = 1
2 → Sino Cola.Final = Cola.Final + 1
Fin si

/* Añade un nuevo elemento al final de la cola */

3 → Cola.Elementos[Cola.Final] = NuevoElemento
Fin

146

## PASO 8

Limpiarcola (Cola) maxcola =5
X = 0 Y = 1 R = X + Y
Mientras (R < 10) Y (No Colallena)
Inscola (Cola, R)
X = Y
Y = R
R = X + Y
Fin – Mientras

| X | Y | R |
|---|---|---|
| 0 | 1 | 1 |
| 1 | 1 | 2 |
| 1 | 2 | 3 |

| 1 | 2 | |
|---|---|---|
| 1 | 2 | 3 |

FI FR

147

## PASO 9

Limpiarcola (Cola) maxcola =5
X = 0 Y = 1 R = X + Y
Mientras (R < 10) Y (No Colallena)

CIERTO FALSO

| 1 | 2 | |
|---|---|---|
| 1 | 2 | 3 |

FI SigFI FR

SigFI = FR

LA COLA ESTÁ LLENA

| X | Y | R |
|---|---|---|
| 0 | 1 | 1 |
| 1 | 1 | 2 |
| 1 | 2 | 3 |

**Función ColaLlena**
/* Encuentra la siguiente posición disponible del final */
Inicio
1 → Si (Cola.Final = MaxCola) → FALSO
Entonces SigFinal = 1
2 → Sino SigFinal = Cola.Final + 1
Fin si

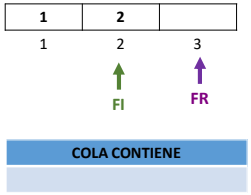
/* La cola está llena si la siguiente posición disponible del final es igual al frente de la cola */

3 → Si (SigFinal = Cola.Frente) → CIERTO
4 → Entonces Imprimir ("La cola está llena")
Sino Imprimir ("La cola no está llena")
Fin si
Fin

148

## PASO 10

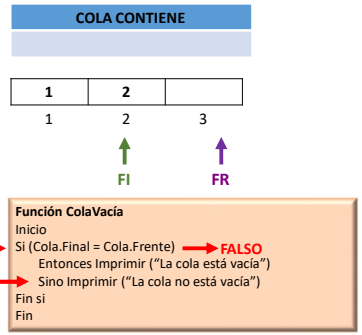
Limpiarcola (Cola) maxcola =3
X = 0 Y = 1 R = X + Y
Mientras (R < 10) Y (No Colallena)
Inicio
Inscola (Cola, R)
X = Y
Y = R
R = X + Y
Fin – Mientras
**Imprimir ( Cola Contiene)**




149

## PASO 11

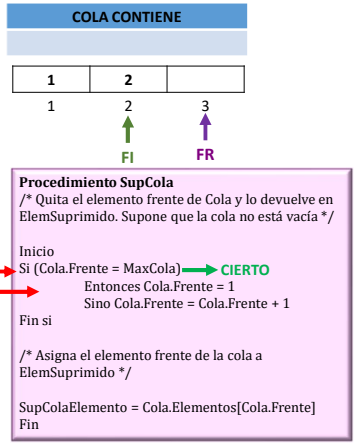
Limpiarcola (Cola) maxcola =3
X = 0 Y = 1 R = X + Y
Mientras (R < 10) Y (No Colallena)
Inicio
Inscola (Cola, R)
X = Y
Y = R
R = X + Y
Fin – Mientras
Imprimir ( Cola Contiene)
**Mientras no-colavacia (Cola ) hacer**
FI ≠ FR
LA COLA NO ESTÁ VACÍA




150

## PASO 12

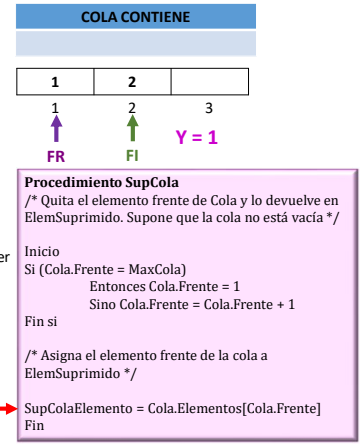
Limpiarcola (Cola) maxcola =3
X = 0 Y = 1 R = X + Y
Mientras (R < 10) Y (No Colallena)
Inicio
Inscola (Cola, R)
X = Y
Y = R
R = X + Y
Fin – Mientras
Imprimir ( Cola Contiene)
Mientras no-colavacia (Cola ) hacer
**Supcola (Cola, Y)**




151

## PASO 12

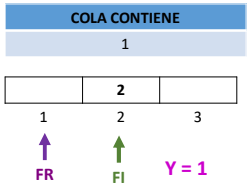
Limpiarcola (Cola) maxcola =3
X = 0 Y = 1 R = X + Y
Mientras (R < 10) Y (No Colallena)
Inicio
Inscola (Cola, R)
X = Y
Y = R
R = X + Y
Fin – Mientras
Imprimir ( Cola Contiene)
Mientras no-colavacia (Cola ) hacer
**Supcola (Cola, Y)**




152

## PASO 13

```
Limpiarcola (Cola)   maxcola =3
X = 0      Y = 1    R = X + Y
Mientras   (R < 10) Y (No Colallena)
Inicio
        Inscola (Cola, R)
        X = Y
        Y = R
        R = X + Y
Fin – Mientras
Imprimir ( Cola Contiene)
Mientras   no-colavacia (Cola )  hacer
        Supcola (Cola, Y)
        Imprimir ( Y )
Fin – Mientras
```



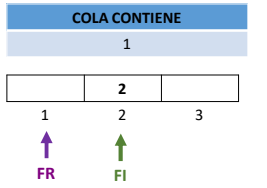

153

## PASO 14

```
Limpiarcola (Cola)   maxcola =3
X = 0      Y = 1    R = X + Y
Mientras   (R < 10) Y (No Colallena)
Inicio
        Inscola (Cola, R)
        X = Y
        Y = R
        R = X + Y
Fin – Mientras
Imprimir ( Cola Contiene)
Mientras   no-colavacia (Cola )  hacer
```

FI ≠ FR

LA COLA NO ESTÁ VACÍA



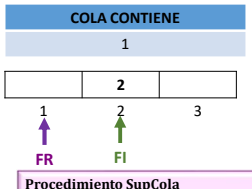

**Función ColaVacía**

```
Inicio
Si (Cola.Final = Cola.Frente)   FALSO
        Entonces Imprimir ("La cola está vacía")
        Sino Imprimir ("La cola no está vacía")
Fin si
Fin
```

154

## PASO 15

```
Limpiarcola (Cola)   maxcola =3
X = 0      Y = 1    R = X + Y
Mientras   (R < 10) Y (No Colallena)
Inicio
        Inscola (Cola, R)
        X = Y
        Y = R
        R = X + Y
Fin – Mientras
Imprimir ( Cola Contiene)
Mientras   no-colavacia (Cola )  hacer
        Supcola (Cola, Y)
```



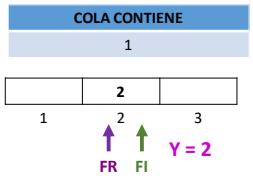

**Procedimiento SupCola**

```
/* Quita el elemento frente de Cola y lo devuelve en
ElemSuprimido. Supone que la cola no está vacía */

Inicio
Si (Cola.Frente = MaxCola)   FALSO
        Entonces Cola.Frente = 1
        Sino Cola.Frente = Cola.Frente + 1
Fin si

/* Asigna el elemento frente de la cola a
ElemSuprimido */

SupColaElemento = Cola.Elementos[Cola.Frente]
Fin
```

155

## PASO 15

```
Limpiarcola (Cola)   maxcola =3
X = 0      Y = 1    R = X + Y
Mientras   (R < 10) Y (No Colallena)
Inicio
        Inscola (Cola, R)
        X = Y
        Y = R
        R = X + Y
Fin – Mientras
Imprimir ( Cola Contiene)
Mientras   no-colavacia (Cola )  hacer
        Supcola (Cola, Y)
```




**Procedimiento SupCola**

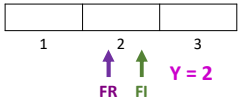
```
/* Quita el elemento frente de Cola y lo devuelve en
ElemSuprimido. Supone que la cola no está vacía */

Inicio
Si (Cola.Frente = MaxCola)
        Entonces Cola.Frente = 1
        Sino Cola.Frente = Cola.Frente + 1
Fin si

/* Asigna el elemento frente de la cola a
ElemSuprimido */

SupColaElemento = Cola.Elementos[Cola.Frente]
Fin
```

156

## PASO 16

```
Limpiarcola (Cola)   maxcola =3
X = 0      Y = 1     R = X + Y
Mientras   (R < 10) Y (No Colallena)
Inicio
        Inscola (Cola, R)
        X = Y
        Y = R
        R = X + Y
Fin – Mientras
Imprimir ( Cola Contiene)
Mientras   no-colavacia (Cola )  hacer
        Supcola (Cola, Y)
        Imprimir ( Y )
Fin – Mientras
```

| COLA CONTIENE |
|---|
| 1 |
| 2 |




157

## PASO 17

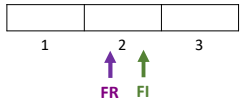
```
Limpiarcola (Cola)   maxcola =3
X = 0      Y = 1     R = X + Y
Mientras   (R < 10) Y (No Colallena)
Inicio
        Inscola (Cola, R)
        X = Y
        Y = R
        R = X + Y
Fin – Mientras
Imprimir ( Cola Contiene)
Mientras   no-colavacia (Cola )  hacer
        Supcola (Cola, Y)
        Imprimir ( Y )
Fin – Mientras
```

1 → 2 → FI = FR

LA COLA ESTÁ VACÍA

**FIN**

| COLA CONTIENE |
|---|
| 1 |
| 2 |



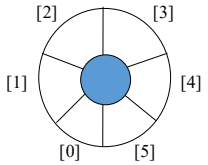

**Función ColaVacía**
Inicio
Si (Cola.Final = Cola.Frente) → CIERTO
  Entonces Imprimir ("La cola está vacía")
  Sino Imprimir ("La cola no está vacía")
Fin si
Fin

158

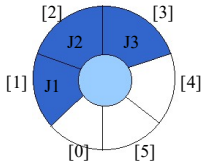
# COLA CIRCULAR

159

## ¿ESTÁ VACÍA LA COLA?



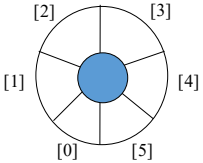

**FR = 0**
**FI = 0**




**FR = 0**
**FI = 3**

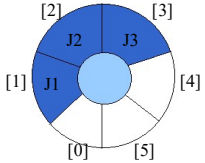
160

## ¿ESTÁ VACÍA LA COLA?




FR = 0
FI = 0

SI



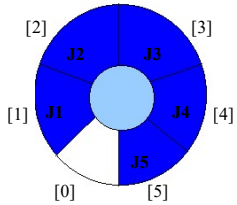

FR = 0
FI = 3

NO

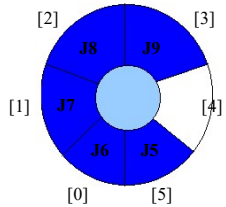
```
Función ColaVacía
Inicio
Si (Cola.Final = Cola.Frente)
     Entonces Imprimir ("La cola está vacía")
     Sino Imprimir ("La cola no está vacía")
Fin si
Fin
```

161

## ¿ESTÁ LLENA LA COLA?



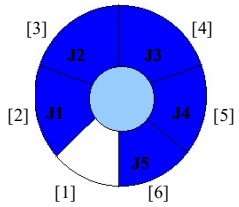

FR =0
FI = 5




FR =4
FI =3

162

## ¿ESTÁ LLENA LA COLA?



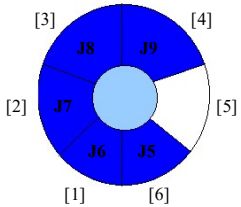

FR = 1
FI = 6

SI

```
Función ColaLlena
/* Encuentra la siguiente posición disponible del
final */
Inicio
Si (Cola.Final = MaxCola)
          Entonces SigFinal = 1
          Sino SigFinal = Cola.Final + 1
Fin si

/* La cola está llena si la siguiente posición
disponible del final es igual al frente de la cola */

Si (SigFinal = Cola.Frente)
          Entonces Imprimir ("La cola está llena")
          Sino Imprimir ("La cola no está llena")
Fin si
Fin
```

163

## ¿ESTÁ LLENA LA COLA?




FR = 5
FI = 4

SI

```
Función ColaLlena
/* Encuentra la siguiente posición disponible del
final */
Inicio
Si (Cola.Final = MaxCola)
          Entonces SigFinal = 1
          Sino SigFinal = Cola.Final + 1
Fin si

/* La cola está llena si la siguiente posición
disponible del final es igual al frente de la cola */

Si (SigFinal = Cola.Frente)
          Entonces Imprimir ("La cola está llena")
          Sino Imprimir ("La cola no está llena")
Fin si
Fin
```

164

## Recursividad

- En las secciones anteriores de este capítulo, hemos visto formas de estructurar mediante funciones datos y programas. Estas funciones se resolvían por sí mismas o podían llamar a otras funciones que terminaban solucionando el problema. Sin embargo, hay veces en que una función sólo se puede resolver volviéndose a llamar a sí mismas. Este tipo de funciones son conocidas como funciones recursivas.
- Esta técnica puede ser utilizada en lugar de la iteración dando excelentes resultados en especial en programación. Las soluciones generadas para los problemas de gran complejidad, en esta técnica, son bien estructuradas.

**Tipos de recursividad**

- **Directa:** un subprograma se llama a sí mismo una o más veces directamente.
- **Indirecta:** se definen una serie de subprogramas que se llaman unos a otros.

165

## Procedimiento recursivo

Un algoritmo recursivo consta de una parte recursiva, otra iterativa o no recursiva y una condición de terminación. La parte recursiva y la condición de terminación siempre existen. En cambio, la parte no recursiva puede coincidir con la condición de terminación.

Algo muy importante para tener en cuenta cuando se usa la recursividad es que es necesario asegurarnos que llega un momento en que no hacemos más llamadas recursivas. Si no se cumple esta condición el programa no parará nunca.

**Reglas para que un algoritmo recursivo funcione correctamente**

a) Debe tener un caso base, por cada llamada recursiva que se haga dentro del algoritmo debe haber un valor para el cual el algoritmo finalice sin recursión.

b) Todos los posibles argumentos de las llamadas recursivas se reenvían tendiendo sus valores hacia un caso base.

c) La función es correcta, para valores distintos del caso base.

166

# Capítulo II: Estructuras dinámicas de datos

167

## Estructura de datos dinámicas lineales

Las listas enlazadas son estructuras dinámicas y lineales que son utilizadas para almacenar datos que continuamente están cambiando. Estas permiten almacenar información en posiciones de memoria que no sean adyacentes. Una de las ventajas de trabajar con las listas es que se pueden obtener posiciones de memoria cuando se requieran y se pueden liberar cuando ya no sean necesarias. Esto nos permite optimizar el espacio de la memoria utilizada.

168

## Listas enlazadas

A este tipo de estructura, se les asigna un espacio de memoria durante la compilación y éste permanece constante durante la ejecución del programa. De allí el nombre de estructuras estáticas.

En esta sección se presenta un tipo de estructura lineal y dinámica de datos, a la cual se le conoce como "Lista". Es lineal porque a cada elemento le sigue sólo otro elemento, y dinámica porque la memoria se puede manejar de manera flexible, no es necesario reservar espacio con antelación.

**Características de la lista**

- Es una colección, de elementos homogéneos, o sea que todos los elementos son del mismo tipo.
- Hay una relación lineal entre los elementos. Para cada elemento existe un anterior y un siguiente, excepto para el primero, que no tiene anterior, y para el último, que no tiene siguiente.
- El orden de los elementos en la lista afecta a su función de acceso, por ejemplo, si la lista está ordenada de más pequeña a mayor el sucesor de cualquier elemento de la lista será mayor o igual que ese elemento.
- Se puede acceder y eliminar cualquier elemento.
- Se pueden insertar elementos en cualquier posición.

169

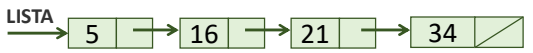
## Tipos de Líneas


- **Lista secuencial:** Una representación de lista en la que el sucesor de cada elemento de la lista está implícito por la posición del elemento: los elementos se colocan en posiciones secuenciales en la estructura.
- **Lista enlazada:** Una representación de lista en la que el orden de los elementos está determinado por un campo enlace explícito en cada elemento, en vez de por su posición secuencial.

170

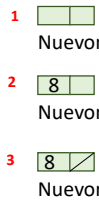
## Ejemplo de Lista

171




**PASO 1**

Crear la estructura del nuevo nodo




```
Procedimiento Insertar NuevoNodo
Inicio      /* Asigna un nuevo nodo y pone nuevovalor en él */
1  Nuevo (Nuevonodo)
2  Nuevonodo.Info = Nuevovalor
3  Nuevonodo.Siguiente = Nulo
```

172

## PASO 2

Ubicar la posición en donde se insertará el nuevo nodo

```
Procedimiento Insertar NuevoNodo
Inicio        /* Asigna un nuevo nodo y pone nuevovalor en él */
Nuevo (Nuevonodo)
Nuevonodo.Info = Nuevovalor
Nuevonodo.Siguiente = Nulo
/* Inserta el nuevo nodo en la lista */
Si (Listavacía (Lista) = "Cierto")          /* Estamos insertando en una lista vacía */
```

```
Función ListaVacía (Booleano)
Inicio
/* Determina si Lista está vacía */
Si (Lista = Nulo)  → FALSO
    Entonces        ListaVacía = Cierto
    Sino            ListaVacía = Falso
Fin Si
Fin
```

La lista no está vacía

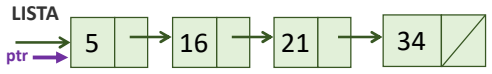
173

## PASO 3

Ubicar la posición en donde se insertará el nuevo nodo

2 8 < 5 FALSO

```
/* Inserta el nuevo nodo en la lista */
Si (Listavacía (Lista) = "Cierto")  → FALSO  /* Estamos insertando en una lista vacía */
Entonces  Lista = Nuevonodo
Sino  Si (Nuevovalor < Lista. Info)  → FALSO /* Estamos insertando en una lista existente */
        Entonces  Nuevonodo.siguiente = Lista        /* Inserta antes del primer nodo */
                  Lista = Nuevonodo
        Sino ptr = Lista                /* Inserta por el medio o al final de la lista */
             LugarEncontrado = Falso
```




LugarEncontrado = Falso

174

## PASO 4

1 Mientras ((ptr.siguiente ≠ Nulo) y (LugarEncontrado = Falso))

CIERTO  CIERTO

2 8 >= 16 FALSO

3 LugarEncontrado = Cierto

4 Mientras ((ptr.siguiente ≠ Nulo) y (LugarEncontrado = Falso))

CIERTO  FALSO


```
Sino ptr = Lista              /* Inserta por el medio o al final de la lista */
     LugarEncontrado = Falso
1    Mientras ((ptr.siguiente ≠ Nulo) y (LugarEncontrado = Falso)) Hacer
2      Si (Nuevovalor >= ptr.siguiente.info)  → FALSO
         Entonces  ptr = ptr.siguiente      /* Se inserta en el medio o en el final */
3        Sino      LugarEncontrado = Cierto
       Fin Si
     Fin Mientras
```

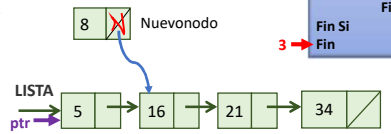
175

## PASO 5

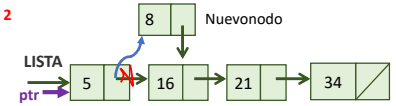
Actualizar los enlaces del nodo o los nodos

```
Fin Mientras
1  Nuevonodo.siguiente = ptr.siguiente
2  ptr.siguiente = Nuevonodo
   Fin Si
Fin Si
3  Fin
```

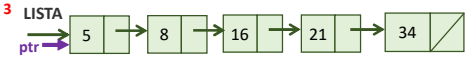
1




2

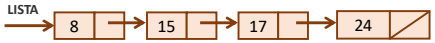



3




176

# Suprimir un Nodo



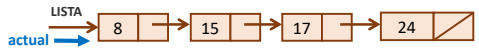

177

## PASO 1

Inicializar los punteros que van a buscar en la lista el valor a eliminar

```
Procedimiento SuprimirNodo
Inicio    /* Inicializa los punteros para buscar */
1  actual = Lista
2  anterior = Nulo
3  encontrado = Cierto
```

1




2

Anterior = 0

3

Encontrado = Cierto

178

## PASO 2

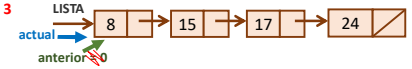
Buscar el nodo que contiene el valor a eliminar de la lista

```
/* Busca el nodo que contiene el valor a suprimir */
1  Mientras (actual.info ≠ supvalor y encontrado = Cierto) Hacer
2     Si (actual.siguiente ≠ nulo)
3        Entonces anterior = actual
                  actual = actual.siguiente
         Sino     encontrado = Falso
      Fin si
4  Fin Mientras
```

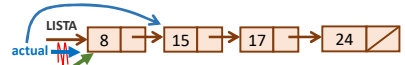
1 8 ≠ 15 y Encontrado = Cierto → CIERTO CIERTO

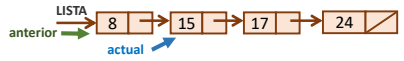
2 actual.siguiente ≠ nulo CIERTO

3




4 15 ≠ 15 y Encontrado = Cierto → FALSO CIERTO




179

## PASO 3

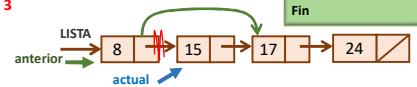
Actualiza los enlaces para mantener la lista enlazada y elimina el nodo que contiene el valor a suprimir

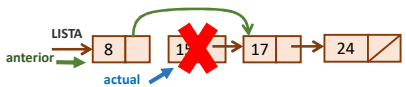
```
1  Si (encontrado = Cierto) entonces   /* Comprueba si supvalor era el primer nodo */
2     Si (anterior = Nulo)                    /* Es el primer nodo */
         Entonces  Lista = Lista.siguiente
                   Liberarnodo (actual)
3     Sino                                /* Quita el nodo en el medio o el final */
         anterior.siguiente = actual.siguiente
         Liberarnodo (actual)
      Fin si
Fin si
Fin
```

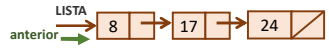
1 Encontrado = Cierto CIERTO

2 anterior = Nulo FALSO

3

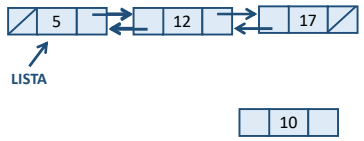



180

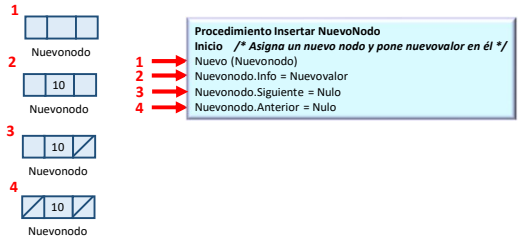
## Listas doblemente enlazadas




181

## PASO 1

Crear la estructura del nuevo nodo




182

## PASO 2

Ubicar la posición en donde se insertará el nuevo nodo

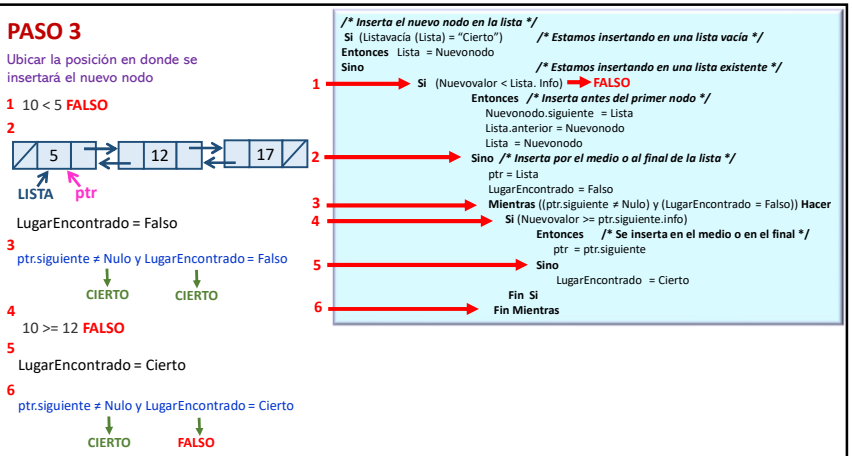
```
1  /* Inserta el nuevo nodo en la lista */
   Si (Listavacía (Lista) = "Cierto")      /* Estamos insertando en una lista vacía */
   Entonces  Lista = Nuevonodo
   Sino                                    /* Estamos insertando en una lista existente */
        Si (Nuevovalor < Lista. Info)
             Entonces  /* Inserta antes del primer nodo */
                  Nuevonodo.siguiente = Lista
                  Lista.anterior = Nuevonodo
                  Lista = Nuevonodo
             Sino  /* Inserta por el medio o al final de la lista */

2  Función ListaVacía (Booleano)
   Inicio
   /* Determina si Lista está vacía */
   Si (Lista = Nulo)  → FALSO
        Entonces   ListaVacía = Cierto
3       Sino       ListaVacía = Falso
   Fin Si
   Fin
```

La lista no está vacía

183

## PASO 3

Ubicar la posición en donde se insertará el nuevo nodo

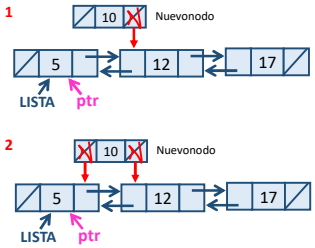



184

**PASO 4**

Actualizar los enlaces del nuevo nodo

1 → Nuevonodo.siguiente = ptr.siguiente
2 → Nuevonodo.anterior = ptr



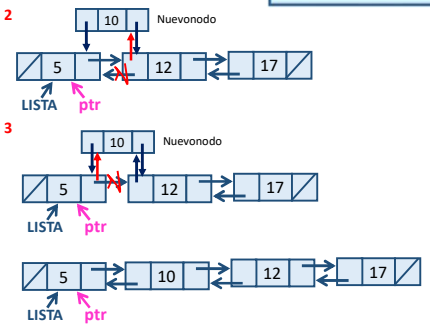

185

**PASO 5**

Actualizar los enlaces del nuevo nodo

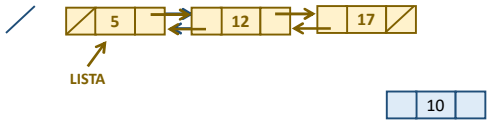
1 → Si (ptr.siguiente ≠ Nulo)
2 → Entonces ptr.siguiente.anterior = Nuevonodo
Fin si
3 → ptr.siguiente = Nuevonodo
Fin Si
Fin Si
Fin

1 ptr.siguiente ≠ Nulo CIERTO




186

## Procedimiento SuprimirNodo




187

**PASO 1**

Inicializar los punteros que van a buscar en la lista el valor a eliminar

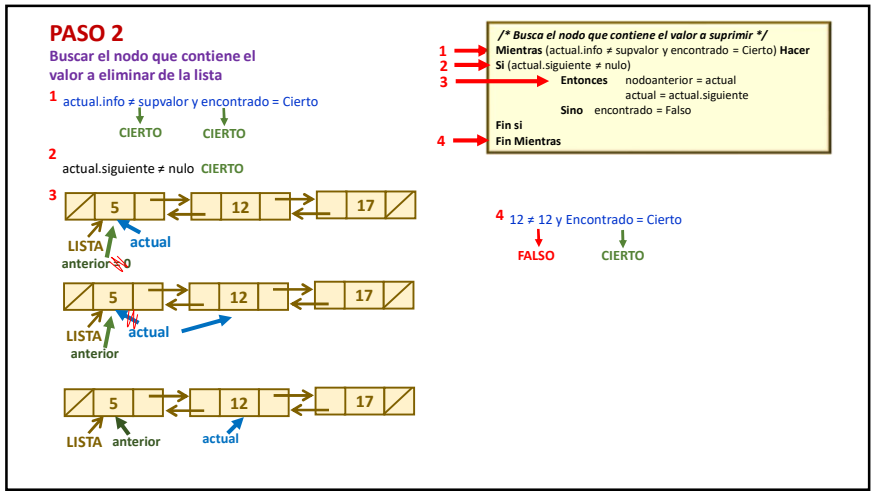
```
Procedimiento SuprimirNodo
Inicio
/* Inicializa los punteros para buscar */
1 → actual = Lista
2 → nodoanterior = Nulo
3 → encontrado = Cierto
```

1

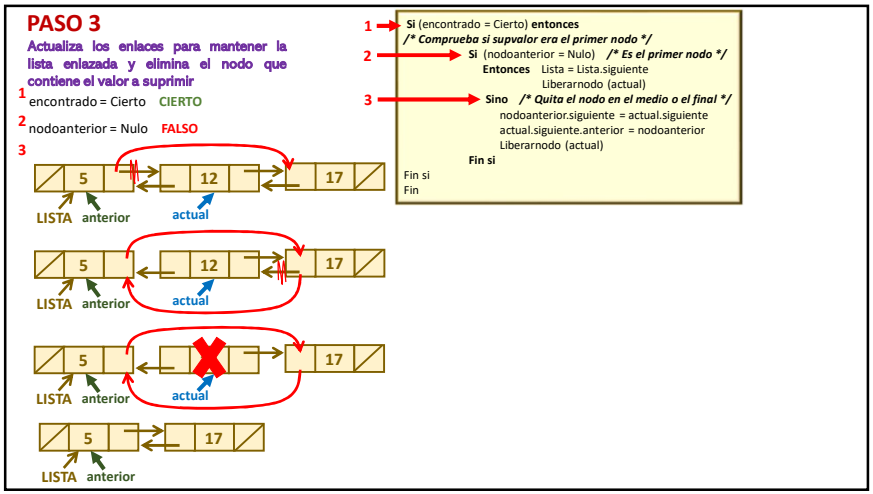

2 nodoanterior = 0

3 encontrado = Cierto

188



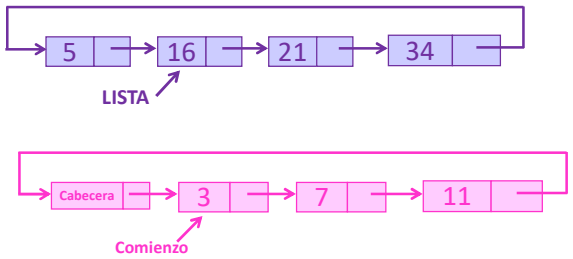

189



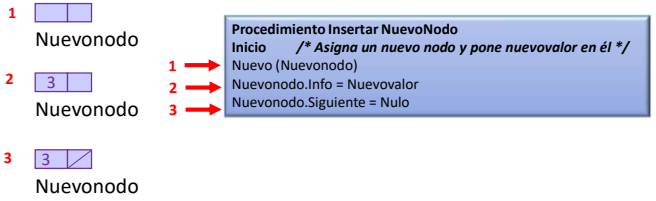

190

## Listas enlazadas circulares

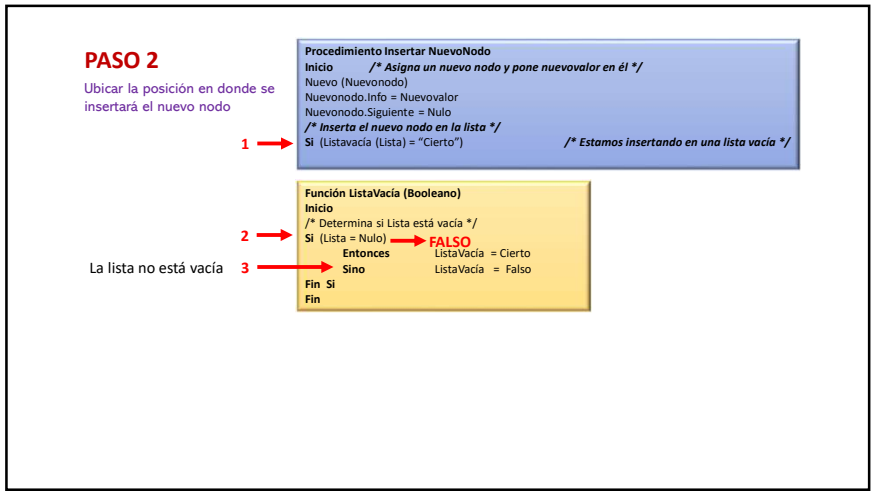
Es una lista lineal en la que el último nodo se enlaza con el primero, de esta forma, cada nodo siempre tiene uno anterior y uno siguiente. El enlace en este tipo de lista es similar al de las listas simplemente enlazadas, con excepción del último nodo que se enlaza con el primero.



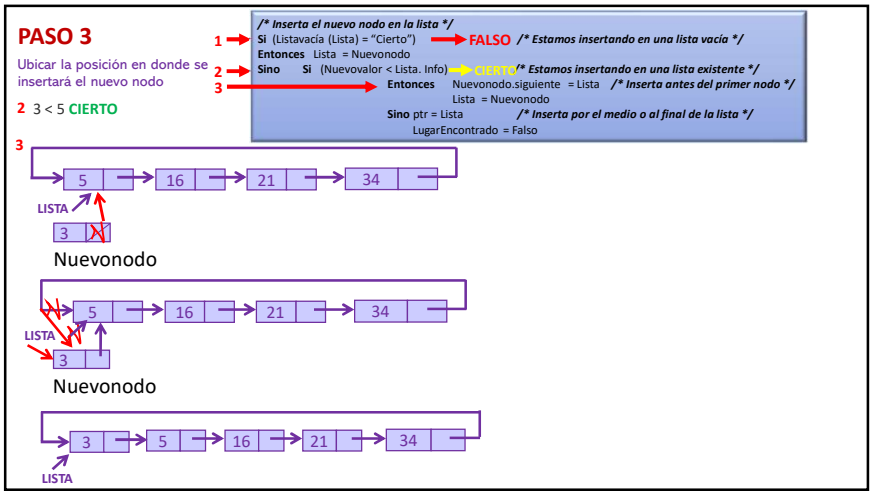

191



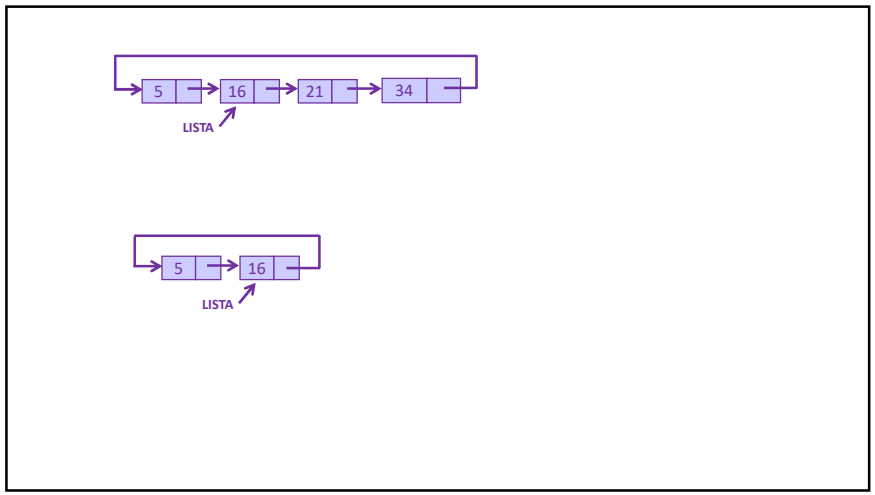

192




193

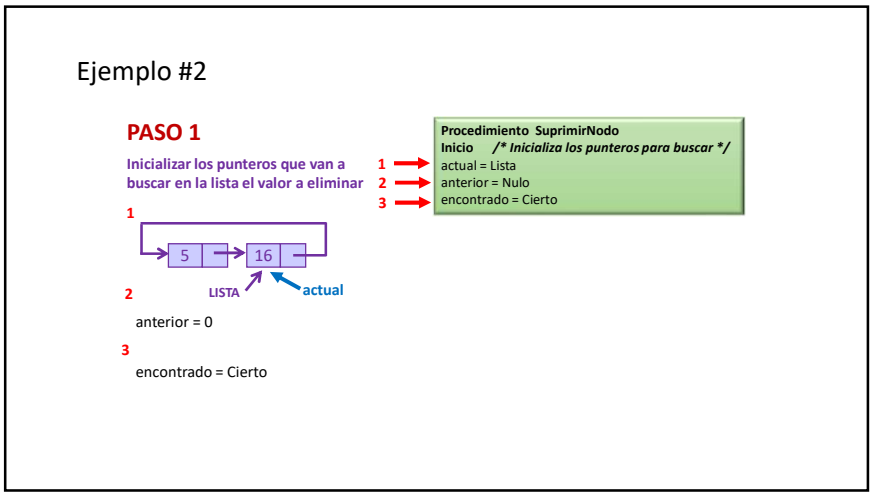



194




195

Ejemplo #2




196

**PASO 2**

Buscar el nodo que contiene el valor a eliminar de la lista

1 16 ≠ 16 y encontrado = Cierto

FALSO CIERTO

```
1 → /* Busca el nodo que contiene el valor a suprimir */
    Mientras (actual.info ≠ supvalor y encontrado = Cierto) Hacer
        Si (actual.siguiente ≠ nulo)
            Entonces anterior = actual
                     actual = actual.siguiente
            Sino     encontrado = Falso
        Fin si
    Fin Mientras
```

197

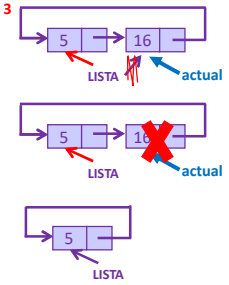
**PASO 3**

Actualiza los enlaces para mantener la lista enlazada y elimina el nodo que contiene el valor a suprimir

1 encontrado = Cierto CIERTO

2 anterior = Nulo CIERTO

3



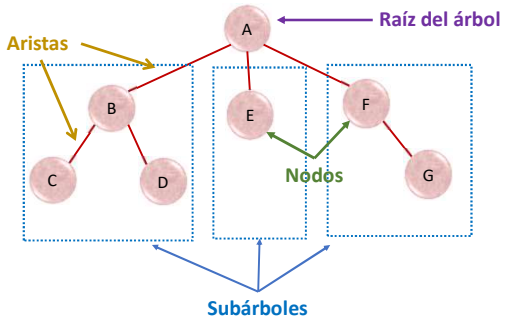

```
1 → Si (encontrado = Cierto) entonces   /* Comprueba si supvalor era el primer nodo */
2 →     Si (anterior = Nulo)            /* Es el primer nodo */
3 →         Entonces Lista = Lista.siguiente
                     Liberarnodo (actual)
            Sino     /* Quita el nodo en el medio o el final */
                     anterior.siguiente = actual.siguiente
                     Liberarnodo (actual)
        Fin si
    Fin si
    Fin
```

198

## Estructuras de datos dinámicas no lineales

1. **Árboles:** Un árbol es una estructura de datos no-lineal y dinámica. Es no-lineal debido a que a cada elemento pueden seguirle varios elementos y es dinámica porque su estructura puede cambiar durante su ejecución.
   - **Árboles generales:** Un árbol general es un árbol donde cada nodo puede tener cero o más hijos (un árbol binario es un caso especializado de un árbol general). Los árboles generales se utilizan para modelar aplicaciones como los sistemas de archivos

199



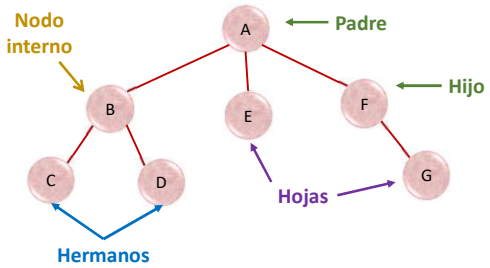

**Nodos del árbol:** A, B, C, D, E, F

**Nodo raíz:** A

**Raíz:** es el primer nodo del árbol, se encuentra ubicado en la parte superior del árbol. Solo hay una raíz por árbol y una ruta desde el nodo raíz a cualquier nodo.

**Aristas o ramas:** son las líneas que unen dos nodos.

200




**Padre:** es el nodo que tiene hijos, es decir, nodos inferiores que se encuentran unidos al nodo superior por una arista. El único nodo que no tiene padre es el nodo raíz del árbol.

A cada nodo se le asocian uno o varios subárboles llamados descendientes o hijos.

**Descendientes o hijos del nodo B:** C, D

**Hijo:** el nodo debajo de un nodo dado (padre) conectado por su arista hacia abajo. Cada nodo puede tener un número arbitrario de hijos.

**Hijos del nodo A:** B, E, F. Estos a su vez conforman los subárboles del árbol general.

**Nodos hermanos:** son los sucesores o descendientes directos de un mismo nodo (hijos de un mismo padre).

**Nodos hermanos (hijos de B):** C, D

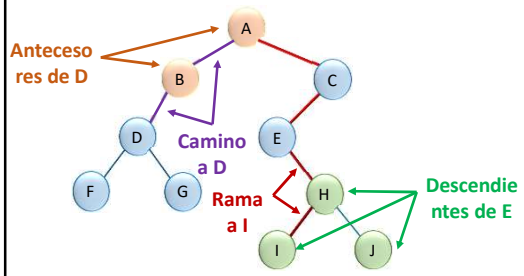
**Hojas:** son los nodos sin hijos. También se les llama nodos terminales, nodos de grado o nodos externos.

**Nodos hojas:** C, D, E, G

**Nodos internos:** son los nodos que tienen al menos un hijo.

**Nodos internos:** A, B, F

201




**Camino de un nodo:** es la secuencia de aristas a través de las cuales se pasa desde el nodo raíz a un nodo.

**Rama:** Un camino que termina en una hoja.

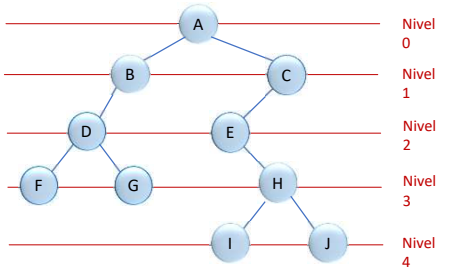
**Antecesores o predecesores de un nodo:** son todos los nodos del camino que va desde la raíz del árbol hasta el nodo.

**Antecesores de D:** B, A

**Descendientes o sucesores de un nodo:** son aquellos nodos accesibles por un camino que comience en el nodo.

**Descendientes del nodo E:** H, I, J

202



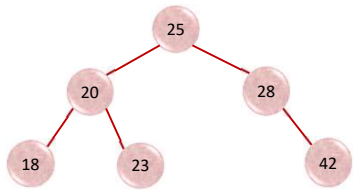

**Grado de un nodo:** es el número de hijos que tiene el nodo. Así, el grado de un nodo hoja es cero. En la figura anterior el grado del nodo D es 2 y el grado del nodo E es 1.

**Grado del árbol:** es el mayor grado de sus nodos. Por ejemplo, el árbol binario es de grado 2 porque cada nodo tiene como mucho dos descendientes directos.

**Niveles:** es la distancia desde la raíz en la que se encuentra ubicado cada nodo. Cada nodo de un árbol tiene asignado un número de nivel de la siguiente forma la raíz tiene el número de nivel 0, y al resto de los nodos se le asigna un número de nivel que es mayor en 1 que el número de nivel del padre.

**Nivel de un nodo:** se refiere a la distancia del nodo desde la raíz del árbol.

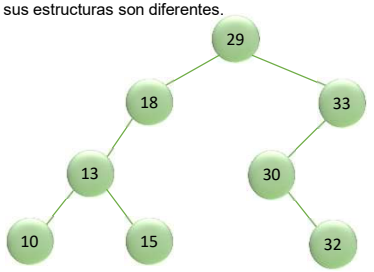
203



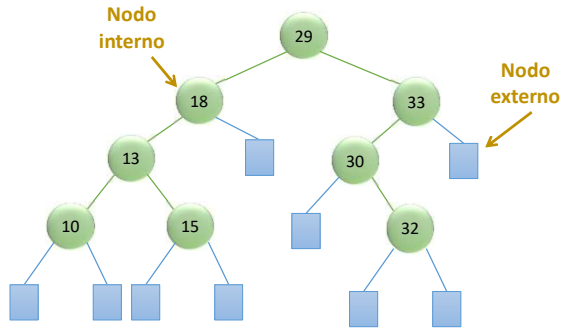

- **Árboles binarios**

El árbol binario es un árbol en el cual cada nodo tiene como máximo dos hijos, uno a la izquierda y el otro a la derecha. En un **árbol binario de búsqueda** el hijo izquierdo, si existe, debe tener un valor menor que el valor de su padre y el hijo derecho, si existe, debe tener un valor mayor que el valor de su padre.

**Árboles binarios distintos:** dos árboles binarios son distintos cuando sus estructuras son diferentes.



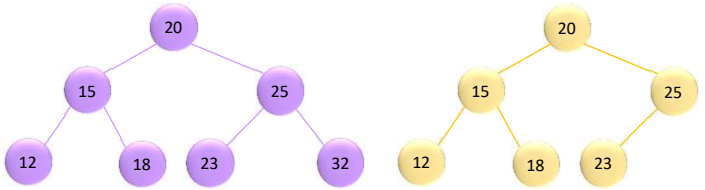

204




- **Árboles binarios extendidos: árboles-2**

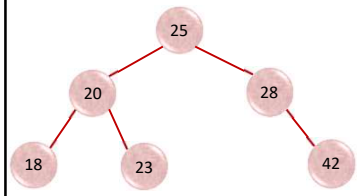
Es un árbol binario en donde el número de hijos de cada nodo es igual al grado del árbol, en este caso, es 2. Si alguno de los nodos no cumple con esta condición, entonces se le deben agregar tantos nodos especiales como se requiera para llegar a cumplirla.

205




**Árbol binario completo:** el árbol binario es completo si todos sus niveles, excepto posiblemente el último, tienen el máximo número de nodos posibles y si todos los nodos del último nivel están situados lo más posible a la izquierda.

206





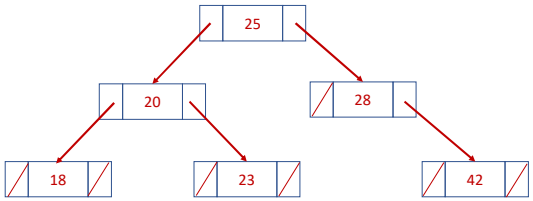
En el siguiente ejemplo se mostrará la representación enlazada en memoria de un árbol binario de búsqueda. Observe que cada nodo está dibujado con sus tres campos y que los subárboles vacíos están dibujados usando un diagonal / para las entradas nulas.

- **Enlazada**

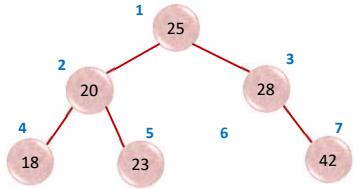
Un árbol binario se puede representar en memoria de forma enlazada utilizando tres arreglos paralelos INFO, IZQ y DER como se muestra en la figura de abajo y una variable puntero a la que llamaremos RAIZ.

A cada nodo N del árbol le corresponderá una posición K, tal que:

- INFO [K] contendrá los datos del nodo N.
- IZQ [K] contendrá la localización del hijo izquierdo del nodo N.
- DER [K] contendrá la localización del hijo derecho del nodo N.




207




ARBOL

| POSICIÓN | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| VALOR | 25 | 20 | 28 | 18 | 23 | | 42 |

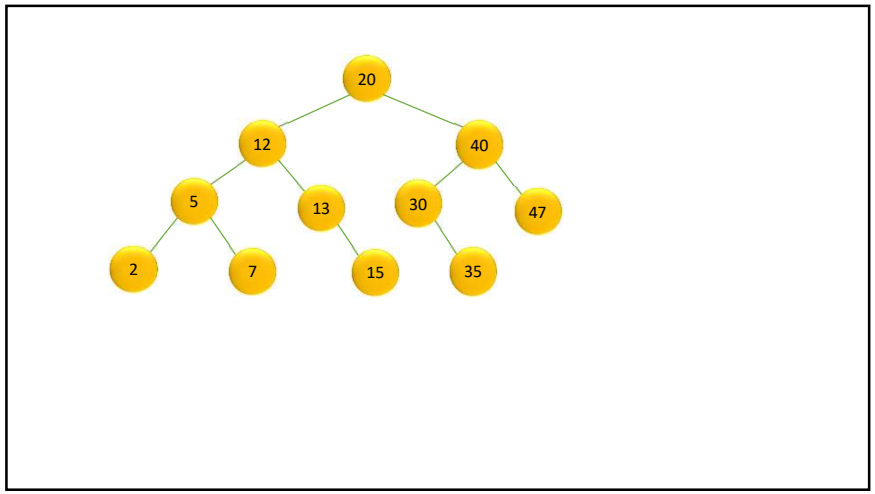
- **Secuencial**

Esta representación usa un arreglo lineal al que llamaremos ARBOL de la forma siguiente:

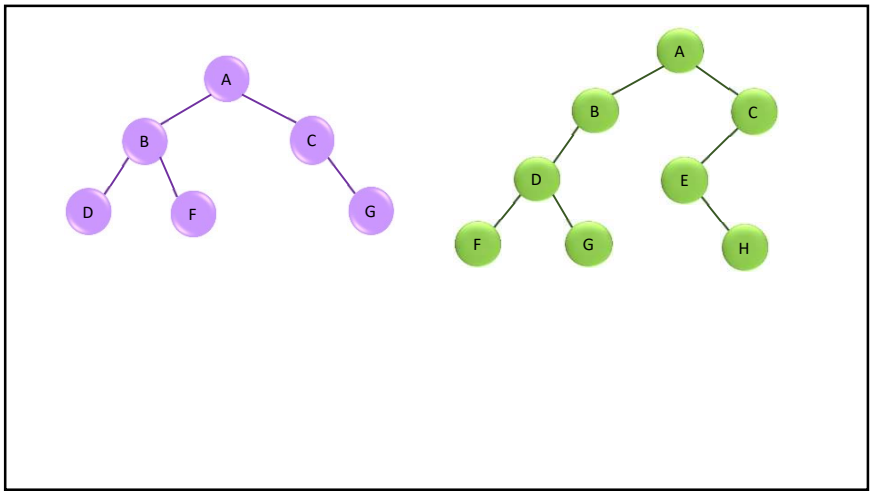
- La raíz R del árbol se guarda en la posición del arreglo ARBOL [1].
- Si un nodo N está ubicado en la posición ARBOL [K], entonces sus hijos:
  - Izquierdo está en la posición ARBOL [2*K]
  - Derecho en la posición ARBOL [2*K+1]

Se usa nulo para indicar que el árbol o un subárbol está vacío, así ARBOL [1] = NULO indica que el árbol está vacío.

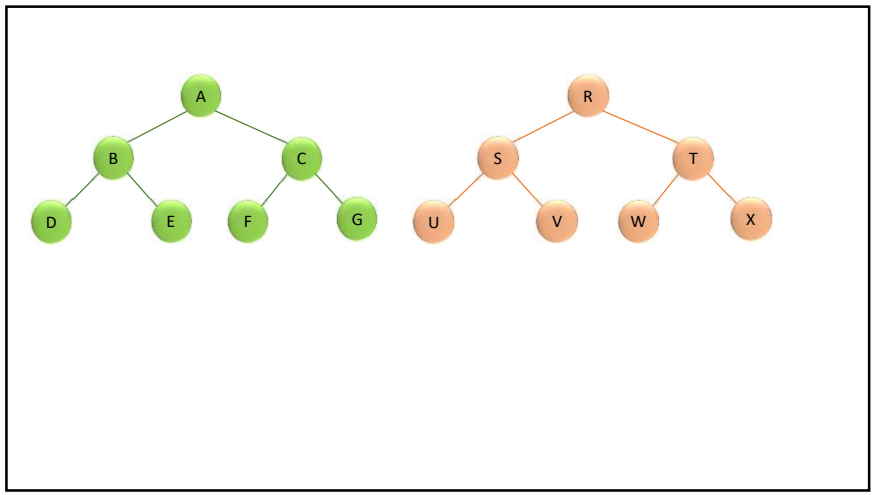
208



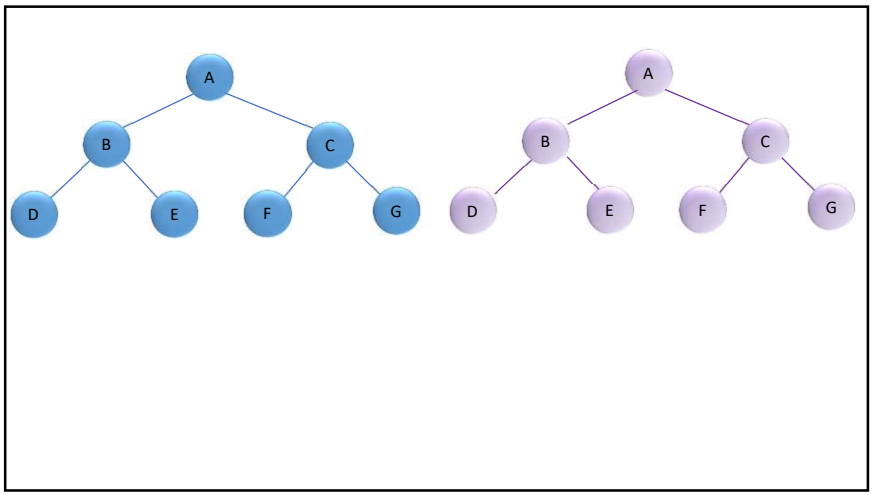

209



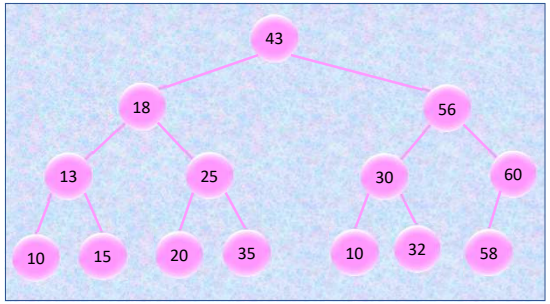

210



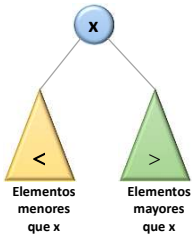

211




212




213

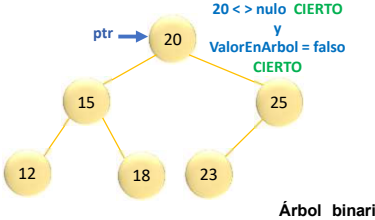



214

**Paso 1**

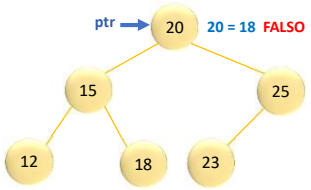
ptr ← Raízarbol
ValorEnArbol ← Falso

**Mientras (ptr < > nulo) y (ValorEnArbol = falso) hacer**




**Paso 2**

**Si (ptr.info = ClaveNueva)**



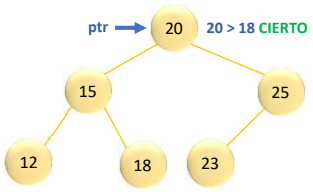

**Árbol binario completo:** el árbol binario es completo si todos sus niveles, excepto posiblemente el último, tienen el máximo número de nodos posibles y si todos los nodos del último nivel están situados lo más posible a la izquierda.

**BUSQUEDA**

215

**Paso 3**

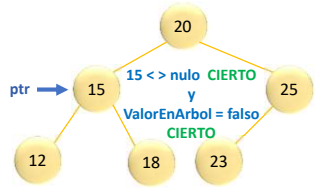
Si (ptr.info > ClaveNueva)




**BUSQUEDA**

**Paso 4**

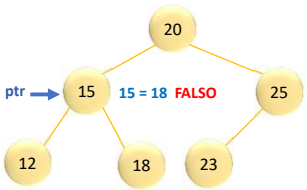
**Mientras (ptr < > nulo) y (ValorEnArbol = falso) hacer**




216

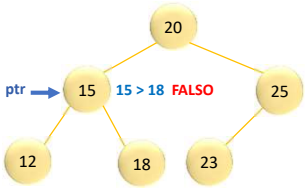
Paso 5

Si (ptr.info = ClaveNueva)




Paso 6

Si (ptr.info > ClaveNueva)

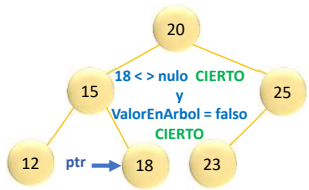



BUSQUEDA

217

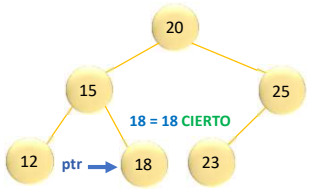
Paso 7

Mientras (ptr < > nulo) y (ValorEnArbol = falso) hacer




Paso 8

Si (ptr.info = ClaveNueva)



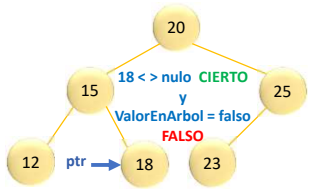

ValorEnArbol ← cierto

BUSQUEDA

218

Paso 9

Mientras (ptr < > nulo) y (ValorEnArbol = falso) hacer




Si (ValorEnArbol = cierto) ➡ CIERTO

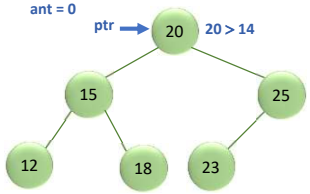
El valor: 18 ha sido encontrado

BUSQUEDA

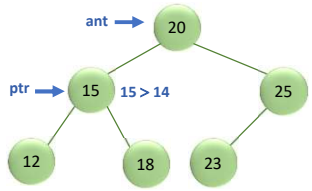
219

Paso 1

ant = 0

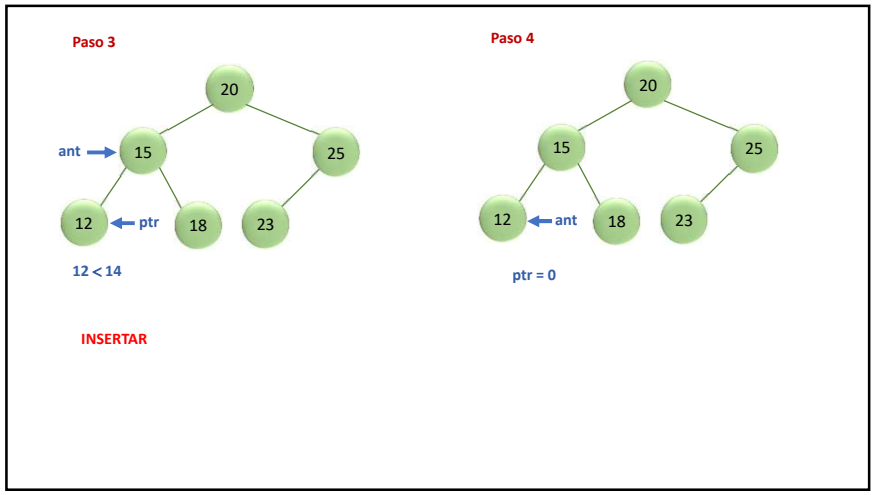



Paso 2

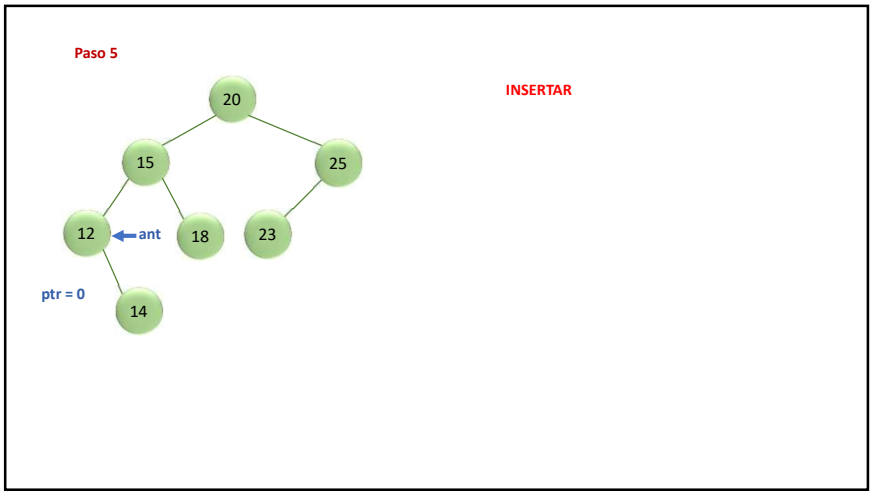



INSERTAR

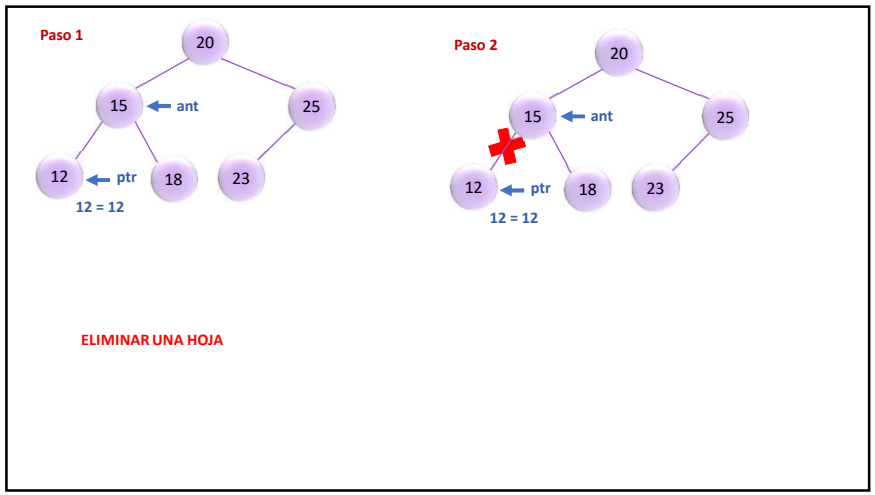
220



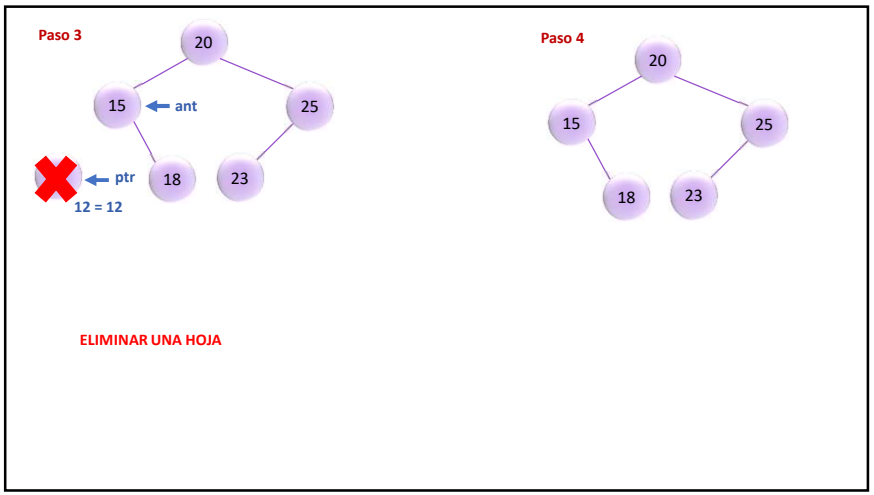

221



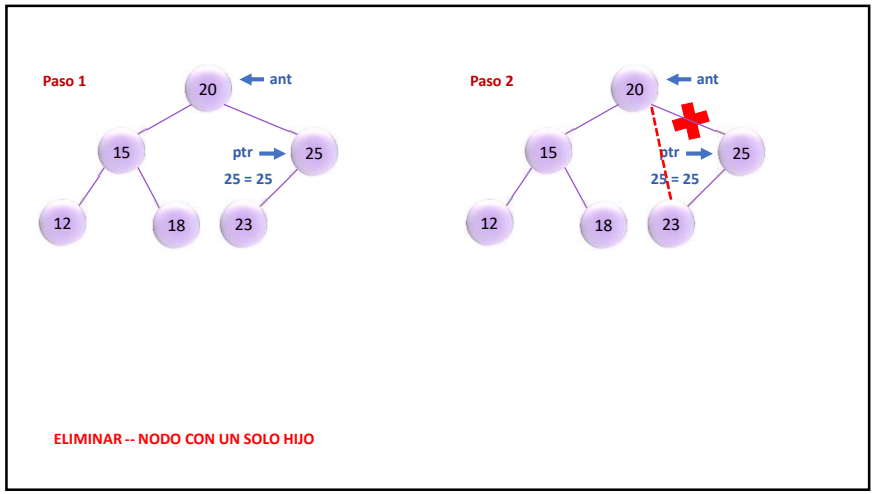

222



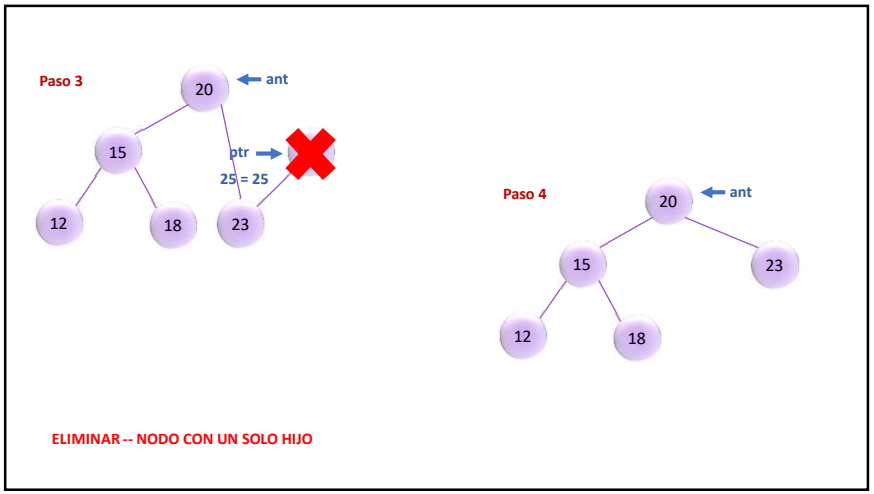

223



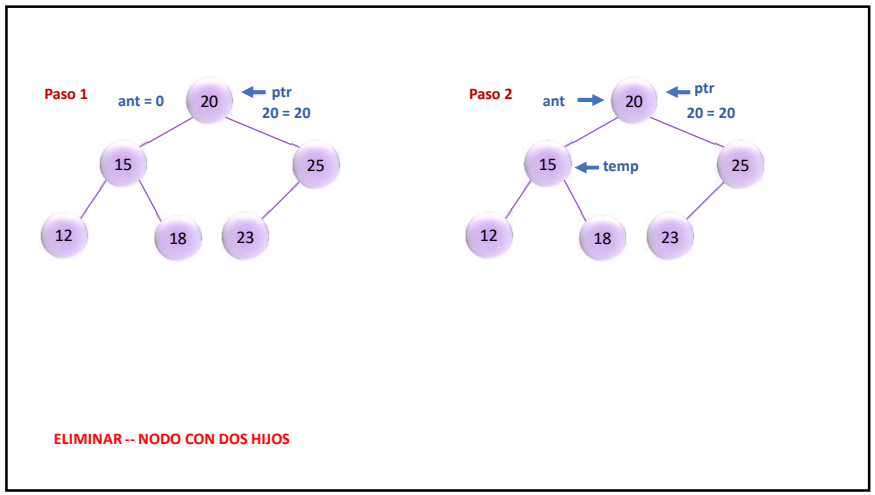

224



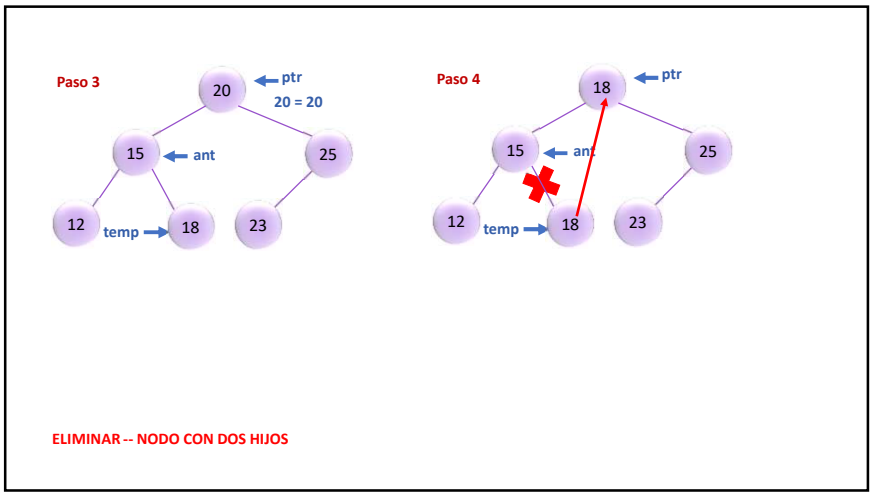

225



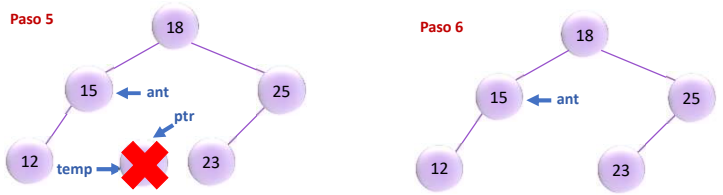

226




227

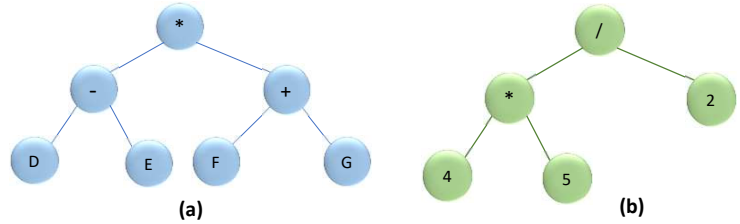



228



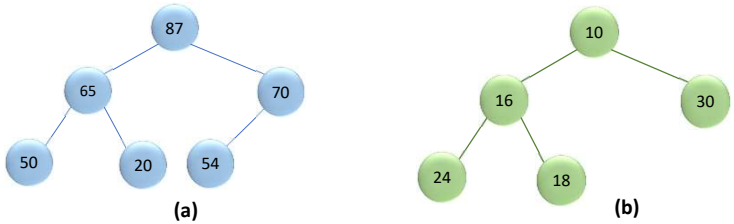

ELIMINAR -- NODO CON DOS HIJOS

229



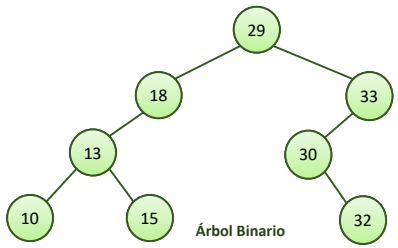

ËJEMPLOS ARBOLES BINARIOS DE EXPRESION

230



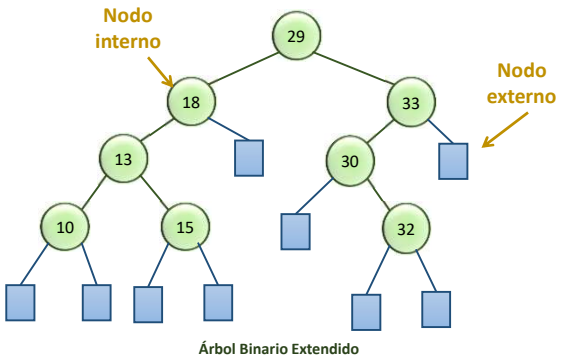

ËJEMPLOS ARBOLES EN MONTON MIN MAX

231




EJEMPLOS ARBOLES EXTENDIDOS

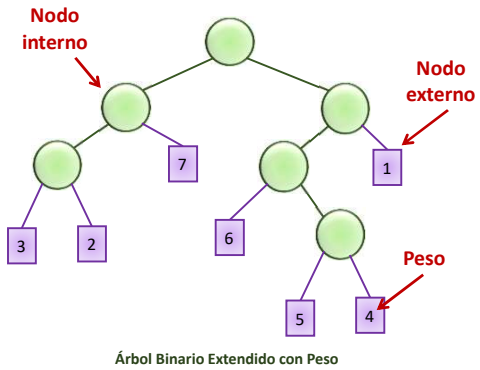
232




Árbol Binario Extendido

EJEMPLOS ARBOLES EXTENDIDOS

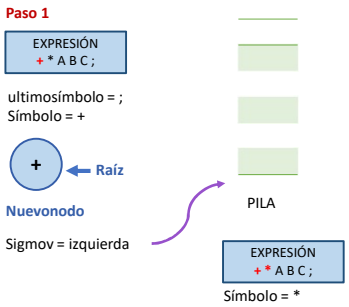
233



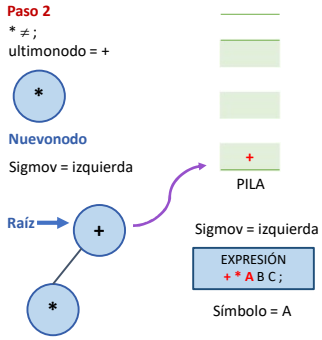

Árbol Binario Extendido con Peso

EJEMPLOS ARBOLES EXTENDIDOS

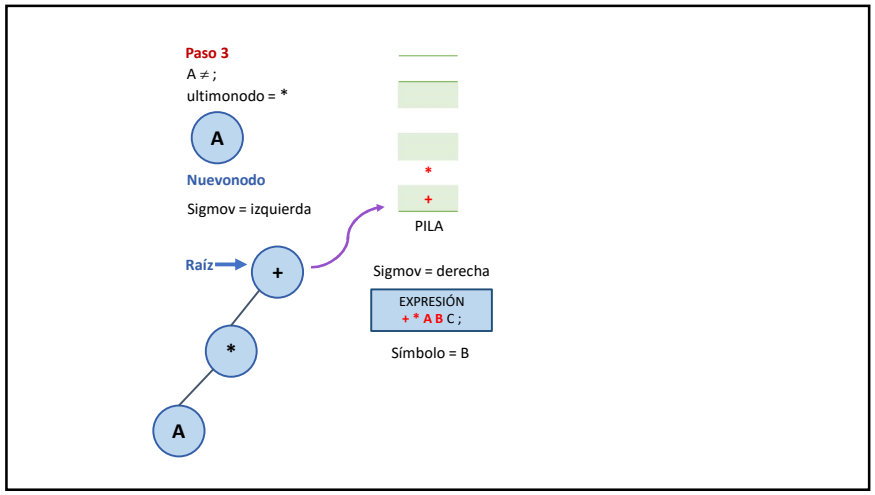
234



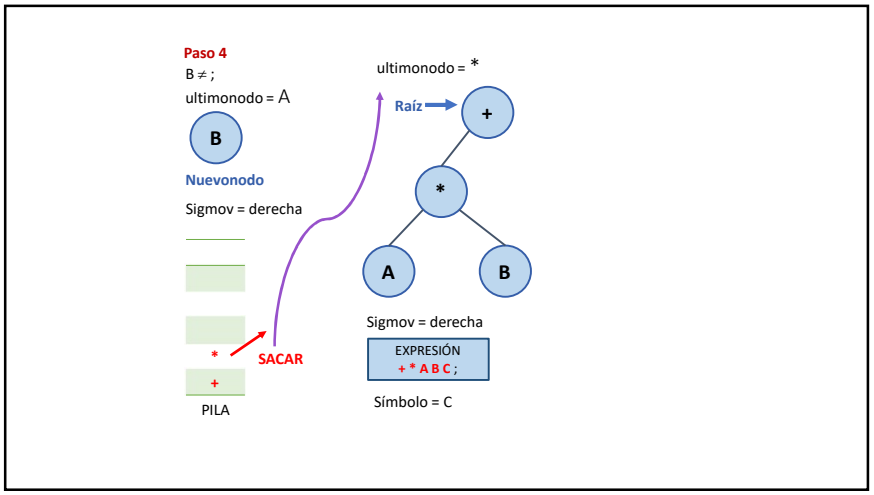

235



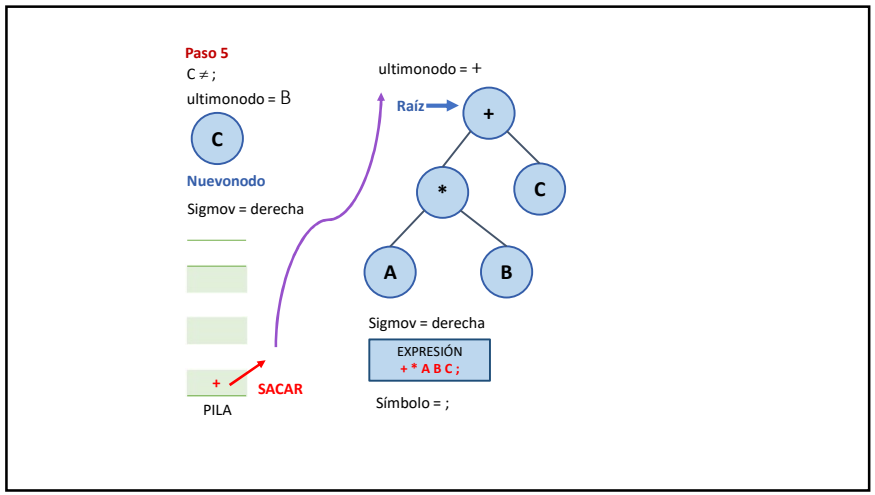

236



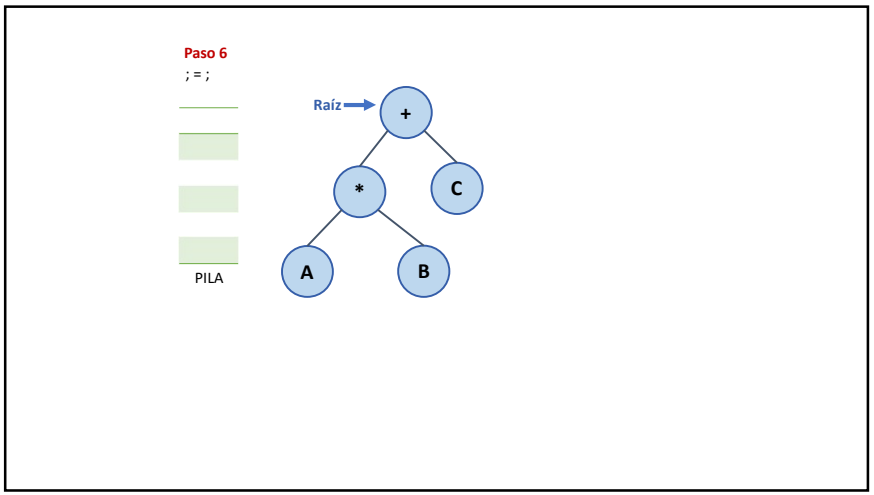

237




238




239




240

## Grafos

Un grafo está formado por un conjunto de nodos (llamados también vértices) y un conjunto de líneas llamadas aristas (o arcos) que conectan los diferentes nodos.

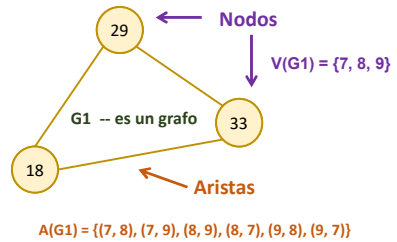
- **Definición formal de un grafo:**

  **G = (V, A)**

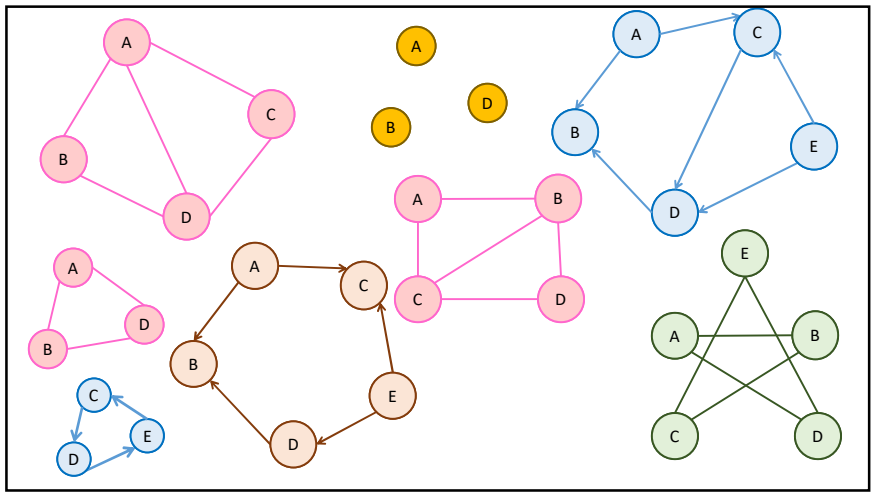
Donde:

- **V(G)** es un conjunto finito, no vacío de vértices que se especifican listando los nodos en notación conjunto, es decir, dentro de paréntesis o llaves.
- **A(G)** es un conjunto de aristas (pares de vértices) que se específica listando una secuencia de aristas. Cada arista se denota escribiendo los nombres de los dos nodos que conecta entre paréntesis, con una coma entre ellos.

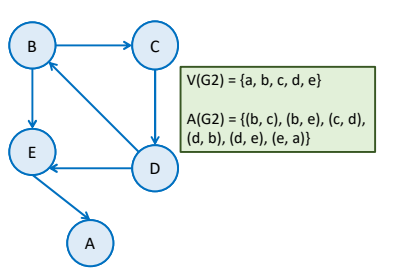
241

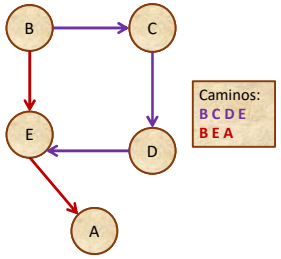



242



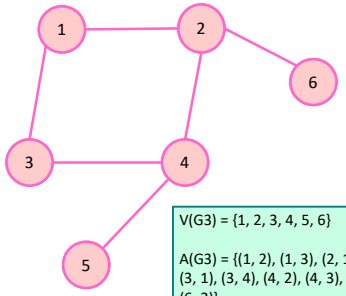
243




**Grafo dirigido:** la dirección de la línea es indicada por el nodo que se lista primero.

244

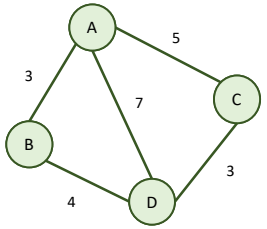



**Grafo no dirigido:** la relación entre los dos nodos es desordenada. Es decir, un nodo apunta al otro; ellos están simplemente conectados.

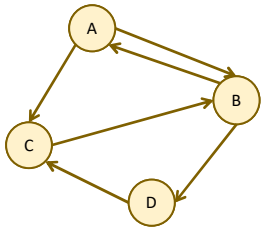
V(G3) = {1, 2, 3, 4, 5, 6}

A(G3) = {(1, 2), (1, 3), (2, 1), (2, 4), (2, 6), (3, 1), (3, 4), (4, 2), (4, 3), (4, 5), (5, 4), (6, 2)}

245



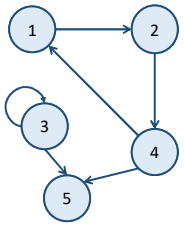

Grafo no dirigido con peso en las aristas




**Grafo acíclico:** es aquel grafo no contiene ningún ciclo simple.

**Grafo cíclico:** un grafo se dice cíclico si contiene algún ciclo simple

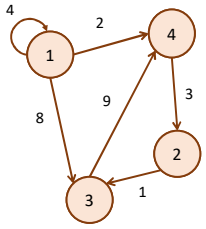
246




| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | 1 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 1 | 0 |
| 3 | 0 | 0 | 1 | 0 | 1 |
| 4 | 1 | 0 | 0 | 0 | 1 |
| 5 | 0 | 0 | 0 | 0 | 0 |

**Multigrafo:** son grafos que aceptan más de una arista entre dos vértices. Estas aristas se llaman múltiples o lazos. Los grafos simples son una subclase de esta categoría de grafos.

247

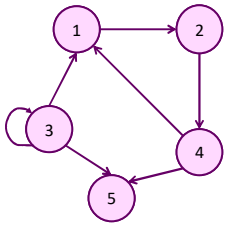



| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 4 | 0 | 8 | 2 |
| 2 | 0 | 0 | 1 | 0 |
| 3 | 0 | 0 | 0 | 9 |
| 4 | 0 | 3 | 0 | 0 |

**Ejemplo**

Escriba la matriz de adyacencia para el siguiente grafo con peso.

248

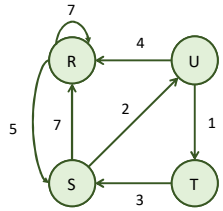



| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 1 | 1 |
| 2 | 1 | 1 | 0 | 1 | 1 |
| 3 | 1 | 0 | 1 | 0 | 1 |
| 4 | 1 | 1 | 0 | 1 | 1 |
| 5 | 0 | 0 | 0 | 0 | 0 |

**Ejemplo**

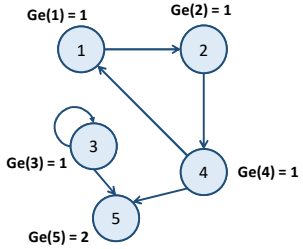
Escriba la matriz de caminos para el siguiente grafo.

249

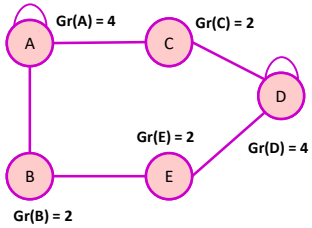



| | R | S | T | U |
|---|---|---|---|---|
| R | 7 | 5 | 0 | 0 |
| S | 7 | 0 | 0 | 2 |
| T | 0 | 3 | 0 | 0 |
| U | 4 | 0 | 1 | 0 |

250




**Multigrafo:** son grafos que aceptan más de una arista entre dos vértices. Estas aristas se llaman múltiples o lazos. Los grafos simples son una subclase de esta categoría de grafos.



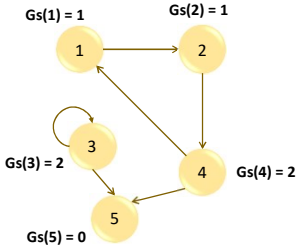

- **Otras definiciones**

**Grado total de un vértice:** corresponde al número de aristas incidentes sobre el vértice, en donde, cada bucle lo cuenta dos veces. En base a esta definición podemos mencionar:

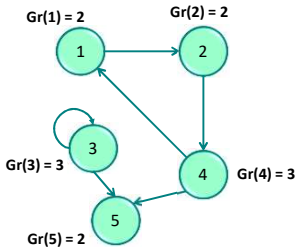
- **Vértice fuente** es un vértice con grado de entrada cero.
- **Vértice sumidero:** es un vértice con grado de salida cero.
- **Vértice hoja:** es un vértice con grado uno.
- **Vértice aislado:** tiene grado cero.

**Grado total de un vértice (Gr) en un grafo no dirigido:** es igual al número de aristas que tiene el vértice.

251

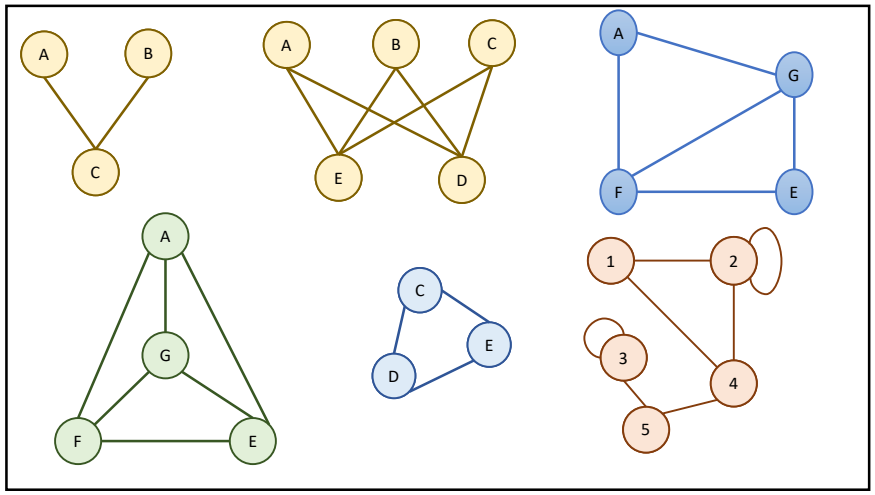



**El grado de entrada (Ge) de un vértice en un grafo dirigido:** es el número de aristas que llegan al vértice




**El grado total de un vértice (Gr) en un grafo dirigido:** es la suma del grado entrante más el grado saliente.

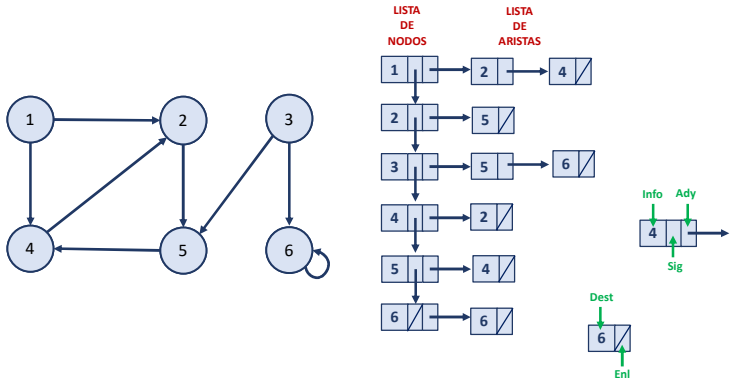
252



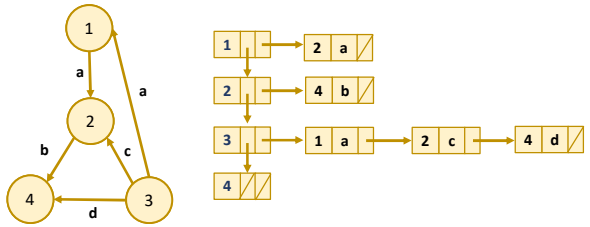

253

**Representación enlazada de grafos**

Otra forma de representar los grafos es utilizando las listas enlazadas.



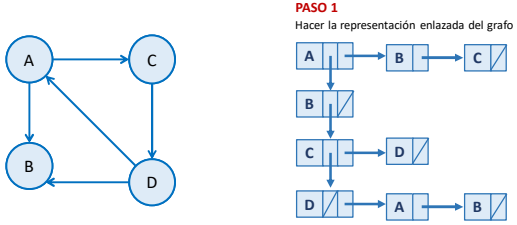

254




- **Lista de aristas:** que contiene la(s) arista(s) a la(s) que el nodo está conectado. Está formada por dos partes (**Figura 54**):
  - **Dest:** campo que apunta al nodo destino de la arista.
  - **Enl:** campo que enlaza las aristas del mismo nodo inicial.

En los grafos con peso se debe agregar un campo adicional en la lista de las aristas en donde se colocará el peso de las mismas

255




**Representación enlazada de grafos**

Otra forma de representar los grafos es utilizando las listas enlazadas.

256