

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código: Versión: Página: Sección ISO: Fecha de emisión	MADO-19 02 1/165 8.3 25 de enero de 2019
Facultad de Ingeniería	Área/Departamento: Laboratorio de computación salas A y B		
La impresión de este documento es una copia no controlada			

# **Manual de prácticas del laboratorio de Estructuras de datos y algoritmos I**

Elaborado por:	Revisado por:	Autorizado por:	Vigente desde:
Jorge A. Solano	Laura Sandoval Montaño	Alejandro Velázquez Mena	25 de enero de 2019

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código:	MADO-19
		Versión:	01
		Página	2/165
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

## Índice de prácticas

No	Nombre	Página
1	<a href="#">Aplicaciones de arreglos</a>	4
2	<a href="#">Aplicaciones de apuntadores</a>	13
3	<a href="#">Tipo de dato abstracto</a>	26
4	<a href="#">Almacenamiento en tiempo de ejecución</a>	34
5	<a href="#">Estructuras de datos lineales: Pila y cola</a>	43
6	<a href="#">Estructuras de datos lineales: Cola circular y cola doble</a>	59
7	<a href="#">Estructuras de datos lineales: Lista simple y lista circular</a>	80
8	<a href="#">Estructuras de datos lineales: Lista doblemente ligada y doblemente ligada circular</a>	91
9	<a href="#">Introducción a Python (I)</a>	105
10	<a href="#">Introducción a Python (II)</a>	120
11	<a href="#">Estrategias para la construcción de algoritmos</a>	133
12	<a href="#">Recursividad</a>	152

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código: MADO-19
		Versión: 01
		Página 3/165
		Sección ISO 8.3
		Fecha de emisión 20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B
La impresión de este documento es una copia no controlada		

# Guía práctica de estudio 01: Aplicaciones de arreglos

---



*Elaborado por:*  
 M.C. Edgar E. García Cano  
 Ing. Jorge A. Solano Gálvez

*Autorizado por:*  
 M.C. Alejandro Velázquez Mena

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código: MADO-19
		Versión: 01
		Página 4/165
		Sección ISO 8.3
		Fecha de emisión 20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B
		La impresión de este documento es una copia no controlada

## Guía práctica de estudio 01: Aplicaciones de arreglos

### **Objetivo:**

Utilizar arreglos unidimensionales y multidimensionales para dar solución a problemas computacionales.

### **Actividades:**

- Crear arreglos unidimensionales.
- Crear arreglos multidimensionales.

### **Introducción**

Un arreglo es un conjunto de datos contiguos del mismo tipo con un tamaño fijo definido al momento de crearse. Pueden ser unidimensionales o multidimensionales.

A cada elemento (dato) del arreglo se le asocia una posición particular. Para acceder a los elementos de un arreglo es necesario utilizar un índice. En lenguaje C, el índice de cada dimensión inicia en 0 y termina en n-1, donde n es el tamaño de la dimensión.

### **Licencia GPL de GNU**

El software presente en esta guía práctica es libre bajo la licencia GPL de GNU, es decir, se puede modificar y distribuir mientras se mantenga la licencia GPL.

```
/*
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
```

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código: MADO-19 Versión: 01 Página 5/165 Sección ISO 8.3 Fecha de emisión 20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B
La impresión de este documento es una copia no controlada		

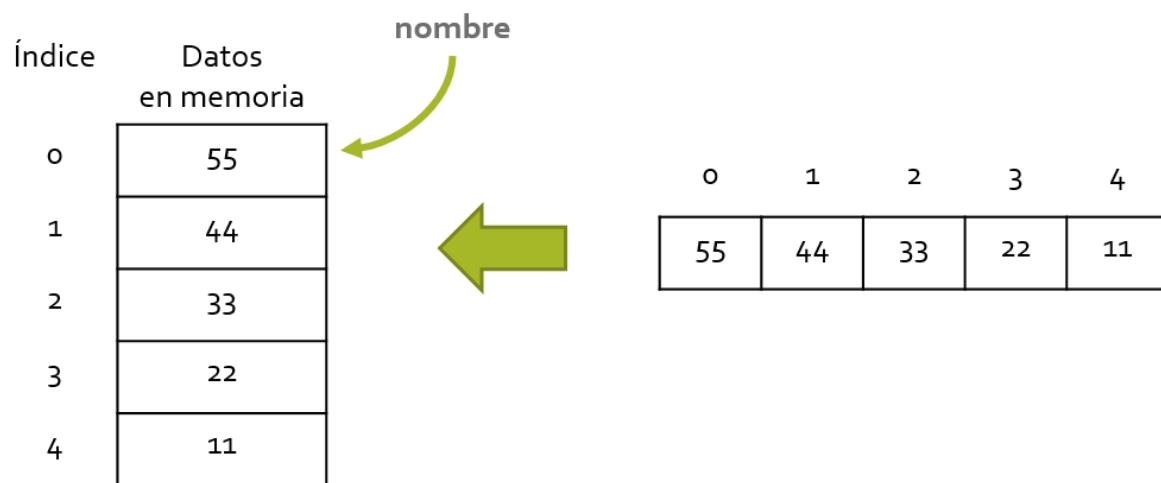
\* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the  
\* GNU General Public License for more details.  
\*  
\* You should have received a copy of the GNU General Public License  
\* along with this program. If not, see <<http://www.gnu.org/licenses/>>.  
\*  
\* Author: Jorge A. Solano  
\*/

## Arreglos contiguos o ligados

Un arreglo contiguo es aquel que se crea desde el inicio del programa y permanece estático durante toda la ejecución del mismo, es decir, no se puede redimensionar.

Un arreglo ligado es aquel que se declara en tiempo de ejecución y bajo demanda, por lo tanto, es posible incrementar su tamaño durante la ejecución del programa, utilizando de manera más eficiente la memoria. Para crear un arreglo ligado se debe utilizar lo que se conoce como memoria dinámica.

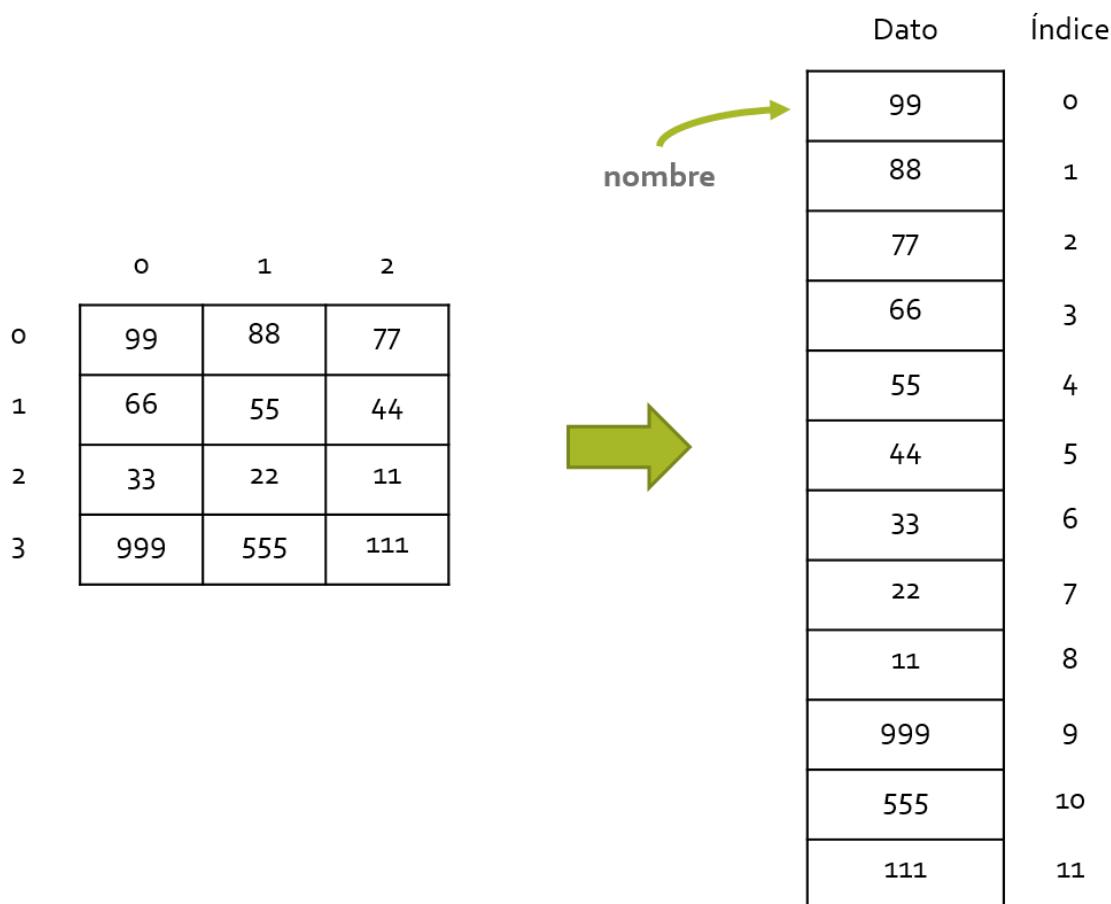
Los arreglos unidimensionales están constituidos por localidades de memoria (ya sea contiguas o ligadas) ordenadas bajo un mismo nombre y sobre un solo nivel (una dimensión).



	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código: MADO-19 Versión: 01 Página: 6/165 Sección ISO: 8.3 Fecha de emisión: 20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B
La impresión de este documento es una copia no controlada		

**Figura 1.** Representación en memoria de un arreglo unidimensional.

Los arreglos multidimensionales están constituidos por localidades de memoria (ya sea contiguas o ligadas) ordenadas bajo un mismo nombre y que pueden tener varios niveles (varias dimensiones) que van desde el plano (2 dimensiones) hasta la enésima dimensión.



**Figura 2.** Representación en memoria de un arreglo bidimensional.

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código: MADO-19 Versión: 01 Página 7/165 Sección ISO 8.3 Fecha de emisión 20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B
La impresión de este documento es una copia no controlada		

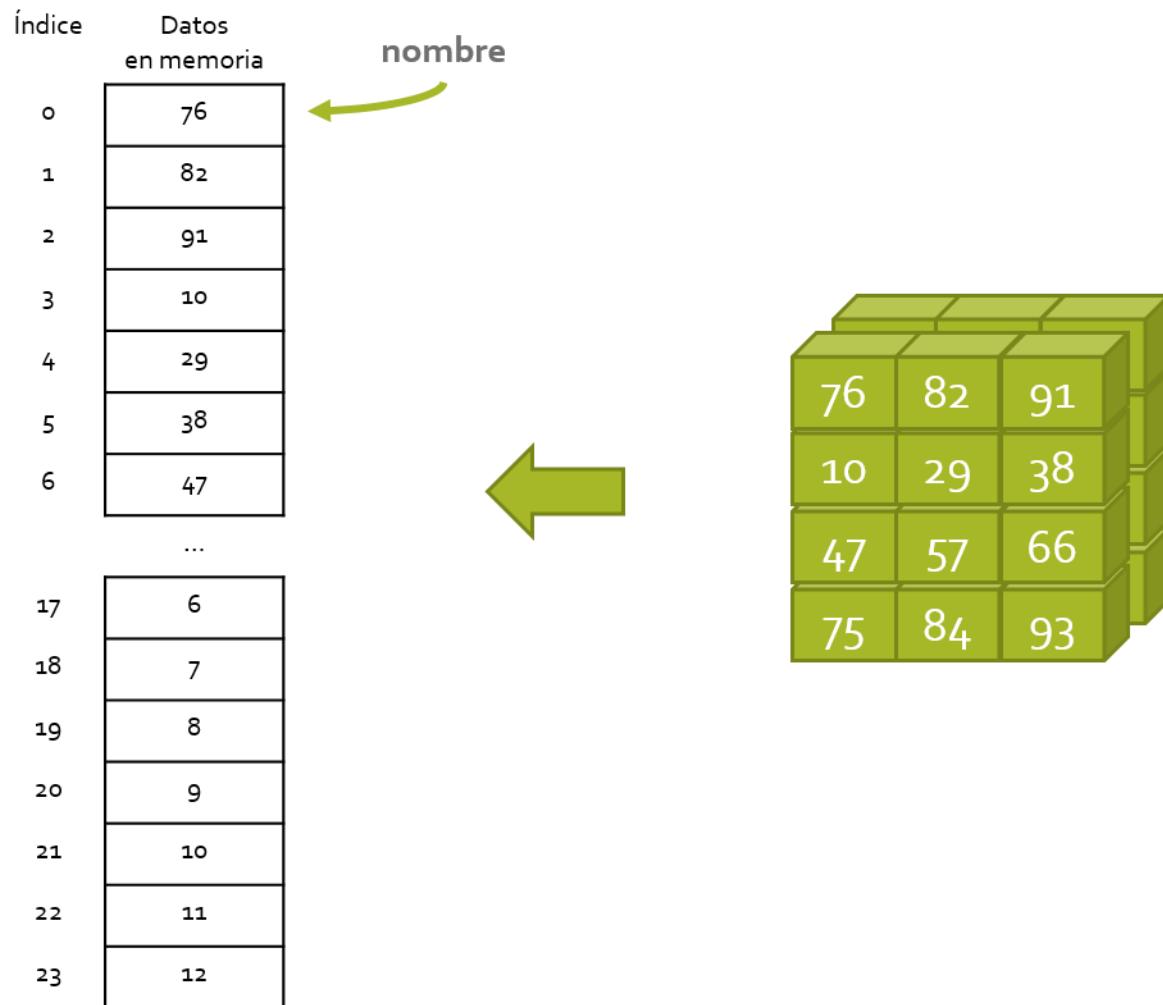


Figura 3. Representación en memoria de un arreglo tridimensional.

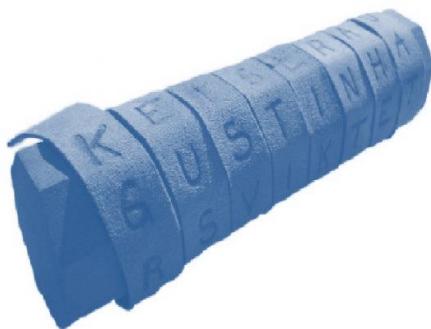
	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código:	MADO-19
		Versión:	01
		Página	8/165
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

## Aplicaciones de arreglos

Los arreglos son una herramienta indispensable a la hora de realizar aplicaciones computacionales. Si se quiere programar un juego de mesa (como ajedrez o scrabble), llevar el control de calificaciones de un grupo de alumnos, implementar estructuras de datos, optimizar operaciones matemáticas, etc., se utilizan necesariamente arreglos.

### La escíptala espartana

Uno de los primeros métodos criptográficos conocidos proviene de Esparta, Grecia. El método consiste en enrollar una tira de escritura a lo largo de un palo llamado escíptala y escribir sobre la tira una vez enrollada. Al desenrollar el mensaje resulta ininteligible a menos que se posea una escíptala similar a la que se usó para crear el mensaje.



**Figura 4.** Forma de la escíptala espartana y la tira de escritura.



## Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I

Código:	MADO-19
Versión:	01
Página	9/165
Sección ISO	8.3
Fecha de emisión	20 de enero de 2017

Facultad de Ingeniería

Área/Departamento:  
Laboratorio de computación salas A y B

La impresión de este documento es una copia no controlada

Código (la escíitala espartana)

```
#include<stdio.h>

/*
    Programa que realiza la implementación de la escíitala espartana
    Para cifrar y descifrar.
*/
void crearMensaje();           
void descifrarMensaje();

int main(){
    short opcion=0;

    while (1){
        printf("\n\t*** ESCÍITALA ESPARTANA ***\n");
        printf("¿Qué desea realizar?\n");
        printf("1) Crear mensaje cifrado.\n");
        printf("2) Descifrar mensaje.\n");
        printf("3) Salir.\n");
        scanf("%d", &opcion);
        switch(opcion){
            case 1:
                crearMensaje();
                break;
            case 2:
                descifrarMensaje();
                break;
            case 3:
                return 0;
            default:
                printf("Opción no válida.\n");
        }
    }
    return 0;
}

void crearMensaje(){
    int ren, col, i, j, k=0;
    printf("Ingresar el tamaño de la escíitala:\n");
    printf("\nRenglones:");
    scanf("%i",&ren);
    printf("\nColumnas:");
    scanf("%i",&col);
```



## Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I

Código:	MADO-19
Versión:	01
Página	10/165
Sección ISO	8.3
Fecha de emisión	20 de enero de 2017

Facultad de Ingeniería

Área/Departamento:  
Laboratorio de computación salas A y B

La impresión de este documento es una copia no controlada

```
char escitala[ren][col];
char texto[ren*col];

printf("Escriba el texto a cifrar:\n");
scanf("%s", texto);

for (i=0 ; i<ren ; i++)
    for (j=0 ; j<col ; j++)
        escitala[i][j] = texto[k++];

printf("El texto en la tira queda de la siguiente manera:\n");
for (i=0 ; i<col ; i++)
    for (j=0 ; j<ren ; j++)
        printf("%c", escitala[j][i]);

printf("\n");
}

void descifrarMensaje(){
    int ren, col, i, j, k=0;
    printf("Ingresar el tamaño de la escitala:\n");
    printf("\nRenglones:");
    scanf("%i",&ren);
    printf("\nColumnas:");
    scanf("%i",&col);

    char escitala[ren][col];
    char texto[ren*col];

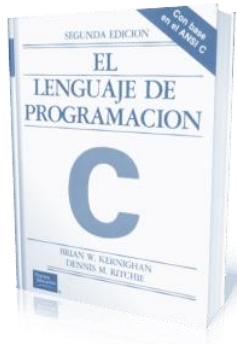
    printf("Escriba el texto a descifrar:\n");
    scanf("%s", texto);

    for (i=0 ; i<col ; i++)
        for (j=0 ; j<ren ; j++)
            escitala[j][i] = texto[k++];

    printf("El texto descifrado es:\n");
    for (i=0 ; i<ren ; i++)
        for (j=0 ; j<col ; j++)
            printf("%c", escitala[i][j]);
}
```

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código:	MADO-19
		Versión:	01
		Página	11/165
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

## Bibliografía



El lenguaje de programación C. Brian W. Kernighan, Dennis M. Ritchie, segunda edición, USA, Pearson Educación 1991.

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código: MADO-19
		Versión: 01
		Página 12/165
		Sección ISO 8.3
		Fecha de emisión 20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B
La impresión de este documento es una copia no controlada		

## Guía práctica de estudio 02: Aplicaciones de apuntadores

---



*Elaborado por:*  
 M.C. Edgar E. García Cano  
 Ing. Jorge A. Solano Gálvez

*Autorizado por:*  
 M.C. Alejandro Velázquez Mena

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código:	MADO-19
		Versión:	01
		Página	13/165
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

## **Guía práctica de estudio 02: Aplicaciones de apuntadores**

### **Objetivo:**

Utilizar apuntadores en lenguaje C para acceder a las localidades de memoria tanto de datos primitivos como de arreglos.

### **Actividades:**

- Crear apuntadores.
- Leer y modificar datos a través de apuntadores.

### **Introducción**

Un apuntador es una variable que contiene la dirección de memoria de otra variable. Los apuntadores se utilizan para dar claridad y simplicidad a las operaciones a nivel de memoria.

Lenguaje C es un lenguaje de alto nivel porque permite programar a bajo nivel. La programación a bajo nivel se refiere a la manipulación de los recursos físicos de un equipo computacional. Los apuntadores permiten manipular de manera directa las localidades de memoria RAM de la computadora.

### **Licencia GPL de GNU**

El software presente en esta guía práctica es libre bajo la licencia GPL de GNU, es decir, se puede modificar y distribuir mientras se mantenga la licencia GPL.

```
/*
 *
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 */
```

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código:	MADO-19
		Versión:	01
		Página	14/165
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

```

*
* This program is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with this program. If not, see <http://www.gnu.org/licenses/>.
*
* Author: Jorge A. Solano
*
*/

```

## Apuntadores

Un apuntador es una variable que contiene la dirección de memoria de otra variable, es decir, hace referencia a la localidad de memoria de otra variable. Debido a que los apuntadores trabajan directamente con la memoria, a través de ellos se accede con rapidez a la información almacenada.

Para declarar un apuntador se debe definir el tipo de dato y el nombre de la variable apuntador precedida de un asterisco (\*). Una variable de tipo apuntador debe tener el mismo tipo de dato de la variable a la que va a apuntar:

TipoDeDatos \*apuntador, variable;

Para asignarle un valor al apuntador, se debe acceder a la localidad de memoria de la variable a través de un ampersand (&):

apuntador = &variable;

Los apuntadores solo deben apuntar a variables del mismo tipo de dato con el que fueron declarados.

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código:	MADO-19
		Versión:	01
		Página	15/165
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería	Área/Departamento: Laboratorio de computación salas A y B		
La impresión de este documento es una copia no controlada			

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código: MADO-19 Versión: 01 Página 16/165 Sección ISO 8.3 Fecha de emisión 20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B
La impresión de este documento es una copia no controlada		

### Código (apuntador a carácter)

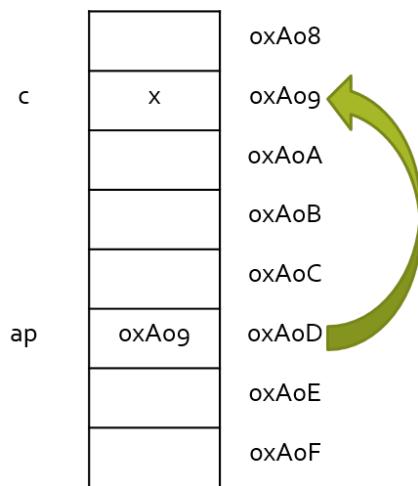
```
#include <stdio.h>

/*
Este programa crea un apuntador de tipo carácter.
*/

int main () {
    char *ap, c;
    c = 'x';
    ap = &c;

    // imprime el carácter de la localidad a la que apunta
    printf("Carácter: %c\n",*ap);
    // imprime el código ASCII de la localidad a la que apunta
    printf("Código ASCII: %d\n",*ap);
    // imprime la dirección de memoria de la localidad a la que apunta
    printf("Dirección de memoria: %d\n",ap);

    return 0;
}
```



**Figura 1.** Apuntador a carácter.

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código: MADO-19 Versión: 01 Página 17/165 Sección ISO 8.3 Fecha de emisión 20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B
La impresión de este documento es una copia no controlada		

### Código (apuntador a entero)

```
#include <stdio.h>

/*
Este programa crea un apuntador de tipo entero
y modifica la información a través del mismo.
*/

int main () {
    short a = 5, b = 10;
    short *apEnt;
    apEnt = &a;
    // imprime el valor entero de a
    printf("a = %i\n", a);
    b = *apEnt;      // b = 5
    // imprime el valor de lo que apunta apEnt
    printf("b = %i /*apEnt\n", b);
    b = *apEnt+1;   // b = 6
    // imprime el valor de lo que apunta apEnt + 1
    printf("b = %i /*apEnt+1\n", b);
    *apEnt = 0;      // a = 0
    // le asigna el valor de 0 a la variable al que apunta apEnt
    printf("a = %i /*apEnt = 0\n", a);

    return 0;
}
```

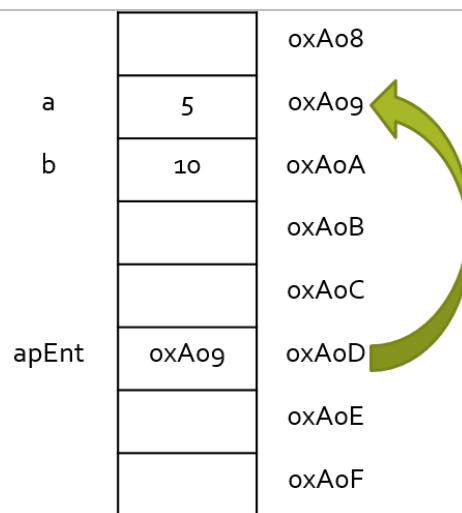


Figura 2. Apuntador a entero.

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código: MADO-19 Versión: 01 Página 18/165 Sección ISO 8.3 Fecha de emisión 20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B
La impresión de este documento es una copia no controlada		

### Código (apuntador a arreglo)

```
#include <stdio.h>

/*
Este programa crea un apuntador de tipo entero
que apunta al inicio de un arreglo.
*/

int main () {
    short arr[5], *apArr;
    apArr = &arr[0];
    // imprime la dirección de memoria del arreglo en la posición [0]
    printf("Dirección del arreglo en la primera posición: %x\n",&arr[0]);
    // imprime la dirección de memoria del arreglo
    // (el nombre del arreglo es un apuntador)
    printf("Dirección del arreglo: %x\n",&arr);
    // imprime la dirección de memoria del apuntador apArr
    printf("Dirección del apuntador: %x\n",apArr);

    return 0;
}
```

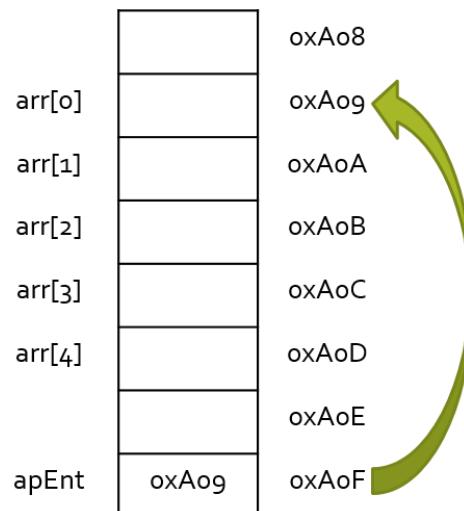


Figura 3. Apuntador a arreglo.

Como se puede observar en el código anterior, el nombre del arreglo es, en sí, un apuntador a la primera localidad del mismo.

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código:	MADO-19
		Versión:	01
		Página	19/165
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

## Parámetros de las funciones

Dentro de los parámetros o argumentos que define una función, se pueden trabajar con los valores o con las referencias de las variables. Cuando se trabaja con los valores (lo que se conoce como paso de variables por valor) en realidad se envía una copia del valor original a la función de tal manera que, si ésta modifica el contenido de la variable, el valor original no se verá afectado. Por otro lado, cuando se trabaja con las referencias (lo que se conoce como paso de variables por referencia) en realidad se envía un apuntador hacia el valor original y, por ende, en realidad se está trabajando con dicho valor todo el tiempo.

Código (Paso de variables por valor y por referencia)

```
#include<stdio.h>

void pasarValor(int);
void pasarReferencia(int *);

int main(){
    int nums[] = {55,44,33,22,11};
    int *ap, cont;
    ap = nums;
    printf("Pasar valor: %d\n", *ap);
    pasarValor(*ap);
    printf("Pasar referencia: %d\n", *ap);
    pasarReferencia(ap);
    printf("Valor final: %d\n", *ap);
    return 0;
}

void pasarValor(int equis){
    printf("%d\n", equis);
    equis = 128;
    printf("%d\n", equis);
}

void pasarReferencia(int *equis){
    printf("%d\n", *equis);
    *equis = 128;
    printf("%d\n", *equis);
}
```

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código:	MADO-19
		Versión:	01
		Página	20/165
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Debido a que el nombre de un arreglo es, en realidad, un apuntador a la primera localidad del arreglo, cuando se envía un arreglo como argumento de una función, dentro de la función se está trabajando directamente con el arreglo original.

## Aplicaciones de apuntadores

Trabajar a nivel de memoria genera muchas ventajas, entre ellas la rapidez y la sencillez para manipular los datos. Dentro de las aplicaciones más útiles de los apuntadores se encuentran las que tienen que ver con arreglos.

Como ya se mencionó, el acceso a un arreglo se puede hacer mediante un índice a cada localidad del mismo. Sin embargo, otra forma de recorrer un arreglo es mediante un apuntador, haciendo más eficiente el acceso a los datos por la rapidez que proporciona éste, esto debido al concepto aritmética de apuntadores.

Dentro de la aritmética de los apuntadores, suponiendo que apArr es un apuntador a algún elemento de un arreglo, entonces:

- apArr++: Incrementa apArr para apuntar a la siguiente localidad de memoria.
- apArr+=i: Incrementa apArr para apuntar a la i-ésima localidad de memoria a partir del valor inicial de apArr.

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código: MADO-19 Versión: 01 Página: 21/165 Sección ISO: 8.3 Fecha de emisión: 20 de enero de 2017
Facultad de Ingeniería	Area/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada		

### Código (aritmética de direcciones)

```
#include<stdio.h>

/*
    Se imprimen 3 valores de un arreglo a través
    de aritmética de direcciones.
*/
int main () {
    short arr[5] = {91,28,73,46,55};
    short *apArr;
    apArr = arr;
    // apunta al inicio del arreglo
    printf("*apArr = %i\n",*apArr);
    // suma una localidad al inicio del arreglo e imprime su valor
    printf("*(apArr+1) = %i\n",*(apArr+1));
    printf("*(apArr+2) = %i\n",*(apArr+2));

    return 0;
}
```

### Código (arreglo unidimensional)

```
#include<stdio.h>

/*
    Se recorre un arreglo unidimensional a través de un apuntador
*/
int main(){
    short nums[] = {55,44,33,22,11};
    short *ap, cont;
    ap = nums;

    for (cont = 0; cont < 5 ; cont++)
        printf("%x\n", (ap+cont));

    return 0;
}
```

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código:	MADO-19
		Versión:	01
		Página	22/165
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

### Código (arreglo bidimensional)

```
#include<stdio.h>

/*
   Se recorre un arreglo bidimensional a través de un apuntador
*/

int main(){
    int *ap, indice;
    int nums[3][3] = {{99,88,77},
                      {66,55,44},
                      {33,22,11}};
    ap = nums;
    for (indice = 0; indice < 9 ; indice++){
        if ((indice%3)==0)
            printf("\n");
        printf("%x\t", (ap+indice));
    }
    return 0;
}
```

### El cifrado César

En el siglo I antes de Cristo, Julio César el célebre militar y político, usó éste cifrado para enviar órdenes a sus generales en los campos de batalla. El método consiste en desplazar el abecedario 3 posiciones, es decir, en lugar de iniciar en la letra A, el abecedario inicia en la letra D.

En claro	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Cifrado	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C

De tal manera, si se quiere enviar el mensaje en claro RETIRADA, el mensaje cifrado será UHWLUDGD.



## Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I

Código:	MADO-19
Versión:	01
Página	23/165
Sección ISO	8.3
Fecha de emisión	20 de enero de 2017

Facultad de Ingeniería

Área/Departamento:  
Laboratorio de computación salas A y B

La impresión de este documento es una copia no controlada

### Código (Cifrado César)

```
#include<stdio.h>

/*
    Programa que realiza la implementación del cifrado César
*/

#define TAM_PALABRA 20
#define TAM_ABC 26

char abecedarioEnClaro[TAM_ABC] =
{'A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q','R','S','T',
,'U','V','W','X','Y','Z'};
char abecedarioCifrado[TAM_ABC] =
{'D','E','F','G','H','I','J','K','L','M','N','O','P','Q','R','S','T','U','V','W',
,'X','Y','Z','A','B','C'};

void cifrar(char *textoEnClaro);
void descifrar(char *textoCifrado);

int main(){
    short opcion = 0, contador;
    char palabra[TAM_PALABRA];

    while (1){
        printf("\n\t*** CIFRADO CÉSAR ***\n");
        for (contador=0 ; contador<26; contador++)
            printf("%c", *(abecedarioEnClaro+contador));
        printf("\n");
        for (contador=0 ; contador<26; contador++)
            printf("%c", *(abecedarioCifrado+contador));
        printf("\nElegir una opción:\n");
        printf("1) Cifrar\n");
        printf("2) Descifrar.\n");
        printf("3) Salir.\n");
        scanf("%d", &opcion);

        switch(opcion){
            case 1:
                printf("Ingresar la palabra a cifrar (en mayúsculas): ");
                scanf("%s", palabra);
                cifrar(palabra);
                break;
            case 2:
                printf("Ingresar la palabra a descifrar (en mayúsculas): ");
                scanf("%s", palabra);
        }
    }
}
```



## Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I

Código:	MADO-19
Versión:	01
Página	24/165
Sección ISO	8.3
Fecha de emisión	20 de enero de 2017

Facultad de Ingeniería

Área/Departamento:  
Laboratorio de computación salas A y B

La impresión de este documento es una copia no controlada

```
        descifrar(palabra);
        break;
    case 3:
        return 0;
    default:
        printf("Opción no válida.");
    }
}

return 0;
}

void cifrar(char *textoEnClaro){
    printf("El texto %s cifrado es: ", textoEnClaro);
    int contadorAbecedario, contadorPalabra, indice = 0;
    for (contadorPalabra=0 ; contadorPalabra<textoEnClaro[contadorPalabra] ; contadorPalabra++)
        for (contadorAbecedario=0 ; contadorAbecedario<TAM_ABC ;
            contadorAbecedario++)
            if (abecedarioEnClaro[contadorAbecedario] ==
                textoEnClaro[contadorPalabra]){
                printf("%c", abecedarioCifrado[contadorAbecedario]);
                contadorAbecedario = 26;
            }

    printf("\n");
}

void descifrar(char *textoCifrado){
    printf("El texto %s descifrado es: ", textoCifrado);
    int contadorAbecedario, contadorPalabra, indice = 0;
    for (contadorPalabra=0 ; contadorPalabra<textoCifrado[contadorPalabra] ; contadorPalabra++)
        for (contadorAbecedario=0 ; contadorAbecedario<TAM_ABC ;
            contadorAbecedario++)
            if (abecedarioCifrado[contadorAbecedario] ==
                textoCifrado[contadorPalabra]){
                printf("%c", abecedarioEnClaro[contadorAbecedario]);
                contadorAbecedario = 26;
            }

    printf("\n");
}
```

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código:	MADO-19
		Versión:	01
		Página	25/165
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

## Bibliografía



El lenguaje de programación C. Brian W. Kernighan, Dennis M. Ritchie, segunda edición, USA, Pearson Educación 1991.

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código: MADO-19
		Versión: 01
		Página 26/165
		Sección ISO 8.3
		Fecha de emisión 20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B
La impresión de este documento es una copia no controlada		

## Guía práctica de estudio 03. Tipo de dato abstracto

---



*Elaborado por:*  
 M.C. Edgar E. García Cano  
 Ing. Jorge A. Solano Gálvez

*Autorizado por:*  
 M.C. Alejandro Velázquez Mena

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código:	MADO-19
		Versión:	01
		Página	27/165
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería	Área/Departamento: Laboratorio de computación salas A y B		
La impresión de este documento es una copia no controlada			

## **Guía práctica de estudio 03. Tipo de dato abstracto**

### **Objetivo:**

Utilizarás estructuras en lenguaje C para modelar tipos de dato abstracto e implementarlos en las estructuras de datos lineales.

### **Actividades:**

- Crear estructuras en lenguaje C.
- Crear tipos de datos utilizando estructuras.

### **Introducción**

Un tipo de dato abstracto (TDA) es un conjunto de datos u objetos creado de manera personalizada por un programador para un fin específico. Un TDA es una abstracción que permite modelar las características de un elemento en particular.

Un tipo de dato abstracto se puede manipular de forma similar a los tipos de datos que están predefinidos dentro del lenguaje de programación, encapsulando más información, según se requiera.

La implementación de un tipo de dato abstracto depende directamente del lenguaje de programación que se utilice. En lenguaje C los tipos de dato abstracto se crean mediante las estructuras (struct).

### **Licencia GPL de GNU**

El software presente en esta guía práctica es libre bajo la licencia GPL de GNU, es decir, se puede modificar y distribuir mientras se mantenga la licencia GPL.

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código:	MADO-19
		Versión:	01
		Página	28/165
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

```
/*
 *
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see <http://www.gnu.org/licenses/>.
 *
 * Author: Jorge A. Solano
 *
 */
```

## Estructuras en lenguaje C

Una estructura es una colección de una o más variables, de iguales o diferentes tipos, agrupadas bajo un solo nombre, es decir, es un tipo de dato compuesto que permite almacenar un conjunto de datos de diferente tipo (agrupar un grupo de variables relacionadas entre sí) que pueden ser tratadas como una unidad (bajo un mismo nombre).

Las estructuras pueden contener tipos de datos simples y tipos de datos compuestos. Los tipos de datos simples (o primitivos) son: carácter, números enteros o números de punto flotante. Los tipos de datos compuestos son: los arreglos y las estructuras.

Por lo tanto, los tipos de datos abstractos (TDA) en lenguaje C se pueden crear a través de una estructura.

Cada ente u objeto es una abstracción de un elemento y, por ende, se puede modelar a través de una estructura en lenguaje C: una película, un video, una pista musical, un documento a imprimir, las armas de un juego, etc.

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código:	MADO-19
		Versión:	01
		Página	29/165
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

La sintaxis para crear estructuras en lenguaje C está definida por la palabra reservada *struct*, seguida del nombre de la estructura y, entre llaves, se definen el número y tipo de variables que definan al nodo (abstracción), es decir:

```
struct nodo {
    tipoDato elemento1;
    tipoDato elemento2;
    ...
    tipoDato elementoN;
};
```

Para crear una variable de un tipo de dato abstracto, se debe especificar el nombre de la estructura y el nombre de la variable, es decir:

```
struct nodo elemento;
```

donde *elemento* es el nombre de la variable con la que se puede acceder a los datos definidos dentro de la estructura.

### Código (Nodo película)

```
#include<stdio.h>

struct pelicula{
    char *nombre;
    char *genero;
    short anio;
    short numDirectores;
    char *directores[10];
};

void imprimirDatosPelicula(struct pelicula);
struct nodo llenarDatosPelicula(char *, char *, short , short , char *[10]);
```

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código:	MADO-19
		Versión:	01
		Página	30/165
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

```

int main(){
    char *directores[10];
    directores[0] = "Lana Wachowski";
    directores[1] = "Andy Wachowski";
    struct pelicula matrix = llenarDatosPelicula("The matrix", "Ciencia
ficción", 1999, 2, directores);
    imprimirDatosPelicula(matrix);
    return 0;
}

struct nodo llenarDatosPelicula(char *nombre, char *genero, short anio,
short numDirectores, char *directores[10]){
    struct pelicula movie;
    movie.nombre = nombre;
    movie.genero = genero;
    movie.anio = anio;
    movie.numDirectores = numDirectores;
    int cont = 0;
    for ( ; cont < movie.numDirectores ; cont++){
        movie.directores[cont] = directores[cont];
    }
    return movie;
}

void imprimirDatosPelicula(struct pelicula movie){
    printf("PELICULA: %s\n", movie.nombre);
    printf("GENERO: %s\n", movie.genero);
    printf("ANIO: %d\n", movie.anio);
    printf("DIRECTOR(ES) :\n");
    int cont = 0;
    for ( ; cont < movie.numDirectores ; cont++){
        printf("%s\n", movie.directores[cont]);
    }
}

```

En el ejemplo anterior se puede observar el modelado de algunas de las características que puede tener una película.



## Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I

Código:	MADO-19
Versión:	01
Página	31/165
Sección ISO	8.3
Fecha de emisión	20 de enero de 2017

Facultad de Ingeniería

Área/Departamento:  
Laboratorio de computación salas A y B

La impresión de este documento es una copia no controlada

### Código (Pila de películas)

```
#include<stdio.h>

#define TAM 2
#define NUM_DIR 2

struct pelicula{
    char nombre[20];
    char genero[20];
    short anio;
    short numDirectores;
    char directores[NUM_DIR][20];
};

void llenarArreglo(struct pelicula *);
void imprimirArreglo(struct pelicula *);

int main(){
    struct pelicula arreglo[TAM];
    llenarArreglo (arreglo);
    imprimirArreglo (arreglo);
    return 0;
}

void llenarArreglo(struct pelicula arreglo [TAM]){
    int iesimo, enesimo;
    for (iesimo=0 ; iesimo<TAM ; iesimo++) {
        struct pelicula movie;
        printf("##### Película %d #####\n", iesimo+1);
        printf("Ingrese nombre película:");
        setbuf(stdin, NULL);
        scanf("%s", movie.nombre);
        getchar();
        printf("Ingrese género película:");
        setbuf(stdin, NULL);
        scanf("%s", movie.genero);
        getchar();
        printf("Ingrese año película:");
        setbuf(stdin, NULL);
        scanf("%d", &movie.anio);
        movie.numDirectores = NUM_DIR;
        for (enesimo=0 ; enesimo<NUM_DIR ; enesimo++) {
            printf("Ingrese director %d:", enesimo+1);
            setbuf(stdin, NULL);
            scanf("%s", movie.directores[enesimo]);
            getchar();
        }
    }
}
```



## Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I

Código:	MADO-19
Versión:	01
Página	32/165
Sección ISO	8.3
Fecha de emisión	20 de enero de 2017

Facultad de Ingeniería

Área/Departamento:  
Laboratorio de computación salas A y B

La impresión de este documento es una copia no controlada

```
        }
        arreglo[iesimo] = movie;
    }

void imprimirArreglo(struct pelicula arreglo [TAM]){
    int iesimo, enesimo;
    printf("##### Contenido del arreglo #####\n");
    for (iesimo=TAM-1 ; iesimo>=0 ; iesimo--) {
        printf("##### Película %d #####\n", iesimo+1);
        printf("PELÍCULA: %s\n", arreglo[iesimo].nombre);
        printf("GÉNERO: %s\n", arreglo[iesimo].genero);
        printf("AÑO: %d\n", arreglo[iesimo].anio);
        printf("DIRECTOR(ES):\n");
        for (enesimo=0 ; enesimo<arreglo[iesimo].numDirectores ;
enesimo++) {
            printf("%s\n", arreglo[iesimo].directores[enesimo]);
        }
    }
}
```

Como se puede observar en el ejemplo anterior, un tipo de dato abstracto permite modelar cualquier elemento (película en este caso) para ser manipulado como una unidad dentro de una estructura de datos. Los elementos (estructuras en lenguaje C) pueden estar compuestos por tipos de datos simples (enteros, caracteres o reales) o por datos compuestos (arreglos y otras estructuras en lenguaje C), haciendo que el modelado de elementos sea tan específico como se requiera.

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código:	MADO-19
		Versión:	01
		Página	33/165
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

## Bibliografía



El lenguaje de programación C. Brian W. Kernighan, Dennis M. Ritchie, segunda edición, USA, Pearson Educación 1991.

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código: MADO-19
		Versión: 01
		Página 34/165
		Sección ISO 8.3
		Fecha de emisión 20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B
La impresión de este documento es una copia no controlada		

# Guía práctica de estudio 04.

## Almacenamiento en tiempo de ejecución

---



*Elaborado por:*  
 M.C. Edgar E. García Cano  
 Ing. Jorge A. Solano Gálvez

*Autorizado por:*  
 M.C. Alejandro Velázquez Mena

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código:	MADO-19
		Versión:	01
		Página	35/165
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

## **Guía práctica de estudio 04. Almacenamiento en tiempo de ejecución.**

### **Objetivo:**

Utilizarás funciones en lenguaje C que permiten reservar y almacenar información de manera dinámica (en tiempo de ejecución).

### **Actividades:**

- Utilizar funciones para reservar memoria dinámica en lenguaje C.
- Almacenar información en los elementos reservados con memoria dinámica.

### **Introducción**

La memoria dinámica se refiere al espacio de almacenamiento que se reserva en tiempo de ejecución, debido a que su tamaño puede variar durante la ejecución del programa.

El uso de memoria dinámica es necesario cuando a priori no se conoce el número de datos y/o elementos que se van a manejar.

### **Licencia GPL de GNU**

El software presente en esta guía práctica es libre bajo la licencia GPL de GNU, es decir, se puede modificar y distribuir mientras se mantenga la licencia GPL.

```
/*
 *
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the

```

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código: MADO-19 Versión: 01 Página 36/165 Sección ISO 8.3 Fecha de emisión 20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B
La impresión de este documento es una copia no controlada		

```

* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with this program. If not, see <http://www.gnu.org/licenses/>.
*
* Author: Jorge A. Solano
*
*/

```

## Memoria dinámica

Dentro de la memoria RAM, la memoria reservada de forma dinámica está alojada en el heap o almacenamiento libre y la memoria estática (como los arreglos o las variables primitivas) en el stack o pila.

La pila generalmente es una zona muy limitada. El heap, en cambio, en principio podría estar limitado por la cantidad de memoria disponible durante la ejecución del programa y el máximo de memoria que el sistema operativo permita direccionar a un proceso.

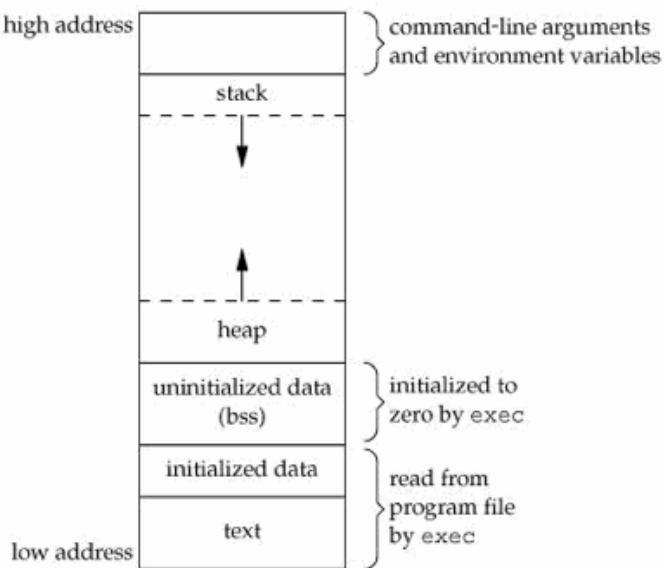


Figura 1. Regiones de la memoria RAM

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código:	MADO-19
		Versión:	01
		Página	37/165
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

El montículo o heap (también llamado segmento de datos) es un medio de almacenamiento con más capacidad que la pila y que puede almacenar datos durante toda la ejecución de las funciones. Las variables globales y estáticas viven en el heap mientras la aplicación se esté ejecutando.

Para acceder a cualquier dato almacenado dentro del heap se debe tener una referencia o apuntador en la pila.

La memoria que se define de manera explícita (estática) tiene una duración fija, que se reserva (al iniciar el programa) y libera (al terminar la ejecución) de forma automática. La memoria dinámica se reserva y libera de forma explícita.

El almacenamiento dinámico puede afectar el rendimiento de una aplicación debido a que se llevan a cabo arduas tareas en tiempo de ejecución: buscar un bloque de memoria libre y almacenar el tamaño de la memoria asignada.

Lenguaje C permite el almacenamiento de memoria en tiempo de ejecución a través de tres funciones: malloc, calloc y realloc.

### **malloc**

La función malloc permite reservar un bloque de memoria de manera dinámica y devuelve un apuntador tipo void. Su sintaxis es la siguiente:

```
void *malloc(size_t size);
```

La función recibe como parámetro el número de bytes que se desean reservar. En caso de que no sea posible reservar el espacio en memoria, se devuelve el valor nulo (NULL).

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código: MADO-19 Versión: 01 Página 38/165 Sección ISO 8.3 Fecha de emisión 20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B
La impresión de este documento es una copia no controlada		

## free

El almacenamiento en tiempo de ejecución se debe realizar de manera explícita, es decir, desde la aplicación se debe enviar la orden para reservar memoria. Así mismo, cuando la memoria ya no se utilice o cuando se termine el programa se debe liberar la memoria reservada. La función free permite liberar memoria que se reservó de manera dinámica. Su sintaxis es la siguiente:

```
void free(void *ptr);
```

El parámetro ptr es el apuntador al inicio de la memoria que se desea liberar. Si el apuntador es nulo la función no hace nada.

## Código (malloc)

```
#include <stdio.h>
#include <stdlib.h>

int main (){
    int *arreglo, num, cont;
    printf("¿Cuantos elementos tiene el conjunto?\n");
    scanf("%d",&num);
    arreglo = (int *)malloc (num * sizeof(int));
    if (arreglo!=NULL) {
        printf("Vector reservado:\n\t[");
        for (cont=0 ; cont<num ; cont++){
            printf("\t%d",*(arreglo+cont));
        }
        printf("\t]\n");
        printf("Se libera el espacio reservado.\n");
        free(arreglo);
    }
    return 0;
}
```

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código:	MADO-19
		Versión:	01
		Página	39/165
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

## calloc

La función calloc funciona de manera similar a la función malloc pero, además de reservar memoria en tiempo real, inicializa la memoria reservada con 0. Su sintaxis es la siguiente:

```
void *calloc (size_t nelem, size_t size);
```

El parámetro nelem indica el número de elementos que se van a reservar y size indica el tamaño de cada elemento.

### Código (calloc)

```
#include <stdio.h>
#include <stdlib.h>

int main (){
    int *arreglo, num, cont;
    printf("¿Cuantos elementos tiene el conjunto?\n");
    scanf("%d",&num);
    arreglo = (int *)calloc (num, sizeof(int));
    if (arreglo!=NULL) {
        printf("Vector reservado:\n\t[");
        for (cont=0 ; cont<num ; cont++){
            printf("\t%d",*(arreglo+cont));
        }
        printf("\t]\n");
        printf("Se libera el espacio reservado.\n");
        free(arreglo);
    }
    return 0;
}
```

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código: Versión: Página: Sección ISO: Fecha de emisión:	MADO-19 01 40/165 8.3 20 de enero de 2017
Facultad de Ingeniería	Área/Departamento: Laboratorio de computación salas A y B		
La impresión de este documento es una copia no controlada			

## realloc

La función realloc permite redimensionar el espacio asignado previamente de forma dinámica, es decir, permite aumentar el tamaño de la memoria reservada de manera dinámica. Su sintaxis es la siguiente:

```
void *realloc (void *ptr, size_t size);
```

Donde ptr es el apuntador que se va a redimensionar y size el nuevo tamaño, en bytes, que se desea aumentar al conjunto.

Si el apuntador que se desea redimensionar tiene el valor nulo, la función actúa como la función malloc. Si la reasignación no se pudo realizar, la función devuelve un apuntador a nulo, dejando intacto el apuntador que se pasa como parámetro (el espacio reservado previamente).

### Código (realloc)

```
#include <stdio.h>
#include <stdlib.h>

int main (){
    int *arreglo, *arreglo2, num, cont;
    printf("¿Cuántos elementos tiene el conjunto?\n");
    scanf("%d",&num);
    arreglo = (int *)malloc (num * sizeof(int));
    if (arreglo!=NULL) {
        for (cont=0 ; cont < num ; cont++){
            printf("Inserte el elemento %d del conjunto.\n",cont+1);
            scanf("%d",(arreglo+cont));
        }
        printf("Vector insertado:\n\t[");
        for (cont=0 ; cont < num ; cont++){
            printf("\t%d",*(arreglo+cont));
        }
        printf("\t]\n");
        printf("Aumentando el tamaño del conjunto al doble.\n");
        num *= 2;
        arreglo2 = (int *)realloc (arreglo,num*sizeof(int));
        if (arreglo2 != NULL) {
            free(arreglo);
            arreglo = arreglo2;
        } else {
            printf("No se pudo aumentar el tamaño del vector.\n");
        }
    } else {
        printf("No se pudo crear el vector.\n");
    }
}
```



## Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I

Código:	MADO-19
Versión:	01
Página	41/165
Sección ISO	8.3
Fecha de emisión	20 de enero de 2017

Facultad de Ingeniería

Área/Departamento:  
Laboratorio de computación salas A y B

La impresión de este documento es una copia no controlada

```
arreglo = arreglo2;
for (; cont < num ; cont++){
    printf("Inserte el elemento %d del conjunto.\n",cont+1);
    scanf("%d", (arreglo2+cont));
}
printf("Vector insertado:\n\t");
for (cont=0 ; cont < num ; cont++){
    printf("\t%d",*(arreglo2+cont));
}
printf("\t]\n");
free (arreglo);
}
return 0;
}
```

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código:	MADO-19
		Versión:	01
		Página	42/165
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

## Bibliografía



El lenguaje de programación C. Brian W. Kernighan, Dennis M. Ritchie, segunda edición, USA, Pearson Educación 1991.

Ariel Rodríguez (2010). How knowing C and C++ can help you write better iPhone apps, part 1. [Figura 1]. Consulta: Enero de 2016. Disponible en: <http://akosma.com/2010/10/11/how-knowing-c-and-c-can-help-you-write-better-iphone-apps-part-1/>

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código: MADO-19
		Versión: 01
		Página 43/165
		Sección ISO 8.3
		Fecha de emisión 20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B
La impresión de este documento es una copia no controlada		

# Guía práctica de estudio 05.

## Estructuras de datos lineales: Pila y cola.

---



***Elaborado por:***  
 M.C. Edgar E. García Cano  
 Ing. Jorge A. Solano Gálvez

***Autorizado por:***  
 M.C. Alejandro Velázquez Mena

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código:	MADO-19
		Versión:	01
		Página	44/165
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

## **Guía práctica de estudio 05. Estructuras de datos lineales: Pila y cola.**

### **Objetivo:**

Revisarás las definiciones, características, procedimientos y ejemplos de las estructuras lineales Pila y Cola, con la finalidad de que comprendas sus estructuras y puedas implementarlas.

### **Actividades:**

- Revisar definición y características de la estructura de datos pila.
- Revisar definición y características de la estructura de datos cola.
- Implementar las estructuras de datos pila y cola.

### **Introducción**

Los conjuntos (colecciones de datos) son tan fundamentales para las ciencias de la computación como lo son para las matemáticas.

Una estructura de datos consiste en una colección de nodos o registros del mismo tipo que mantienen relaciones entre sí. Un nodo es la unidad mínima de almacenamiento de información en una estructura de datos.

Las estructuras de datos lineales son aquellas en las que los elementos ocupan lugares sucesivos en la estructura y cada uno de ellos tiene un único sucesor y un único predecesor.

### **Pila**

La pila (o stack) es una estructura de datos lineal y dinámica, en la cual el elemento obtenido a través de la operación ELIMINAR está predefinido, debido a que implementa la política Last-In, First-Out (LIFO), esto es, el último elemento que se agregó es el primer que se elimina.

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código:	MADO-19
		Versión:	01
		Página	45/165
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

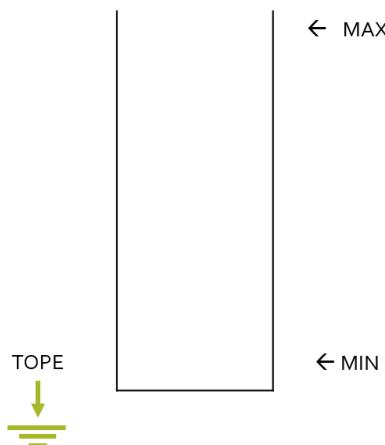
Las operaciones que se pueden realizar sobre una pila son INSERTAR (que es llamada PUSH) y ELIMINAR (que es llamada POP). Debido a la política LIFO que implementa esta estructura, el orden en el que los elementos son extraídos de la pila (POP) es inverso al orden en el que los elementos fueron insertados en la pila (PUSH). Además, el único elemento accesible de la pila es el que está hasta arriba y que se conoce como *tope* de la pila.

Para poder diseñar un algoritmo que defina el comportamiento de una pila se deben considerar 3 casos para ambas operaciones (push y pop):

- Estructura vacía (caso extremo).
- Estructura llena (caso extremo).
- Estructura con elemento(s) (caso base).

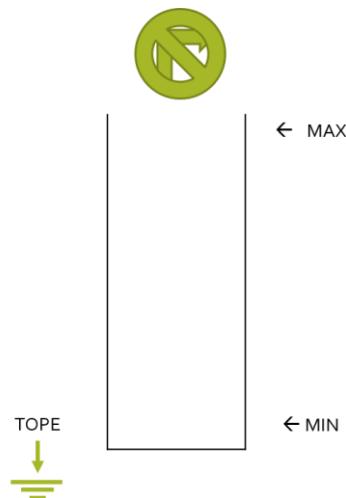
### Pila vacía

Una pila vacía no contiene elemento alguno dentro de la estructura y el *tope* de la misma apunta a *nulo*.

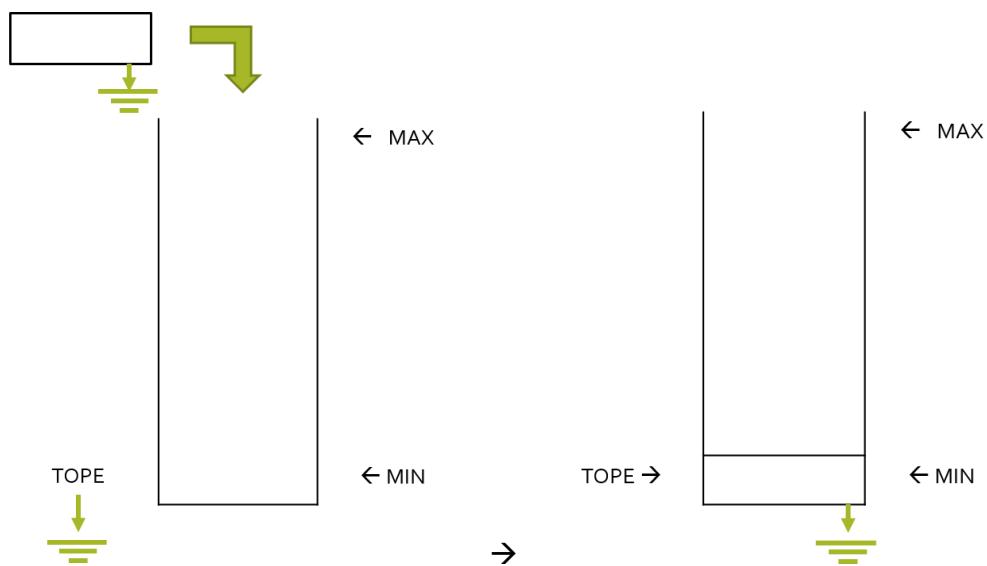


	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código: MADO-19 Versión: 01 Página 46/165 Sección ISO 8.3 Fecha de emisión 20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B
La impresión de este documento es una copia no controlada		

En una pila vacía no es posible realizar POP, debido a que la estructura no contiene información.



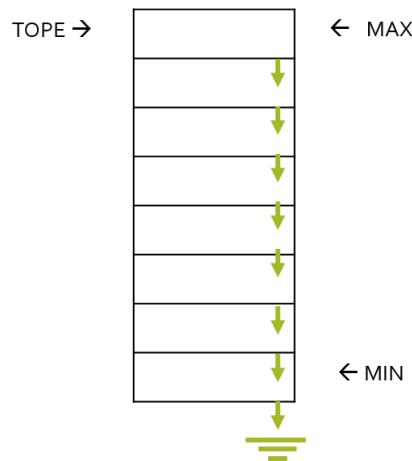
Cuando la pila está vacía sí se puede realizar PUSH, en tal caso, el nodo que entra a la estructura sería el único elemento de la pila y el *tope* apuntaría a él.



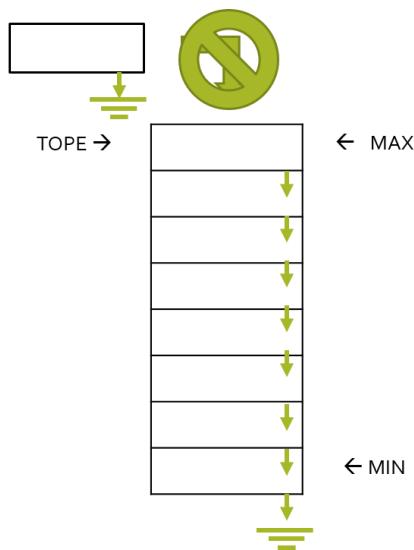
	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código: MADO-19 Versión: 01 Página 47/165 Sección ISO 8.3 Fecha de emisión 20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B
La impresión de este documento es una copia no controlada		

## Pila llena

Por definición, una estructura de datos tipo pila tiene un tamaño fijo. Cuando la pila ha almacenado el número máximo de nodos definido, se dice que la pila está llena.

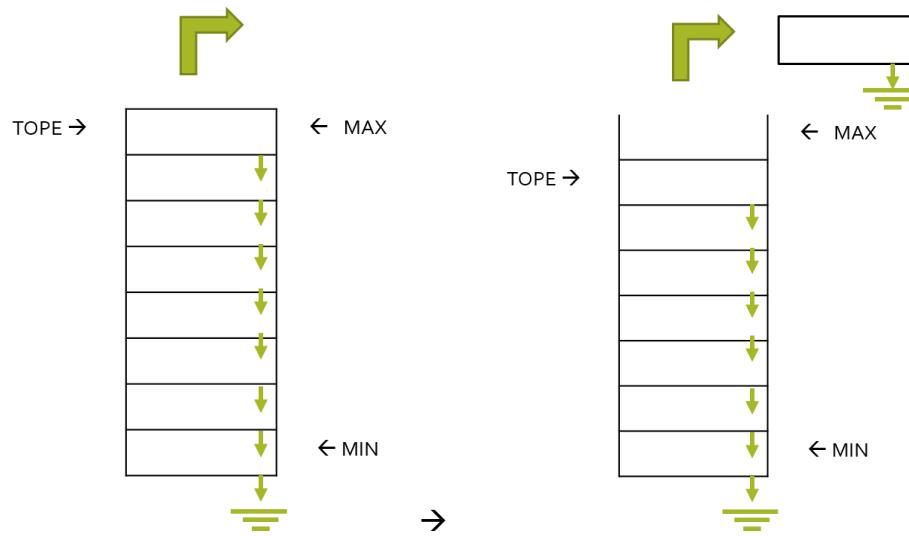


En una pila llena no es posible hacer PUSH de un nuevo elemento, ya que se ha alcanzado el tamaño máximo permitido.



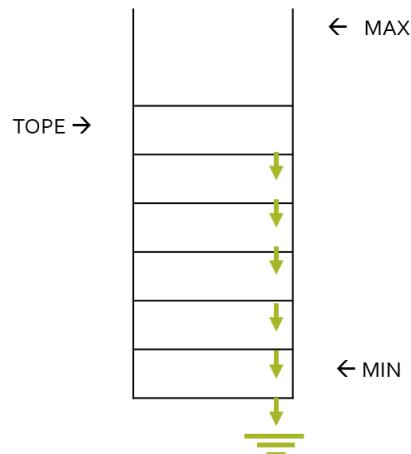
	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código: MADO-19 Versión: 01 Página 48/165 Sección ISO 8.3 Fecha de emisión 20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B
La impresión de este documento es una copia no controlada		

Cuando la pila está llena se puede hacer POP de la información contenida en la estructura. En tal caso, el *tope* apunta al elemento siguiente de la estructura.



### Pila con elementos

Una pila que contiene elementos (sin llegar a su máxima capacidad) representa el caso general.





## Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I

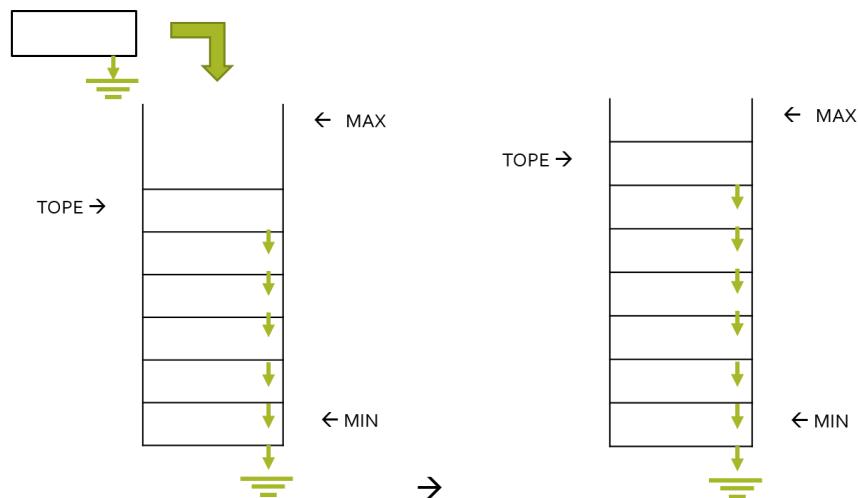
Código:	MADO-19
Versión:	01
Página	49/165
Sección ISO	8.3
Fecha de emisión	20 de enero de 2017

Facultad de Ingeniería

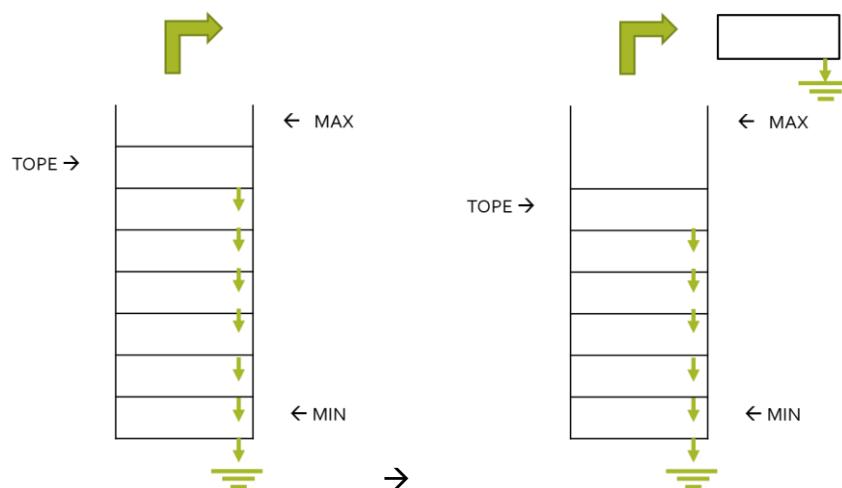
Área/Departamento:  
Laboratorio de computación salas A y B

La impresión de este documento es una copia no controlada

En una pila con elementos se pueden realizar PUSH. En tal caso, el tope apuntara al elemento que se insertó y el nuevo elemento apunta al elemento al que apuntaba *tope*.



En una pila con elementos es posible realizar POP. En tal caso, el nodo al que apunta *tope* se extrae y ahora *tope* apunta al elemento al que apuntaba éste (sucesor).



	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código: <b>MADO-19</b> Versión: <b>01</b> Página <b>50/165</b> Sección ISO <b>8.3</b> Fecha de emisión <b>20 de enero de 2017</b>
<b>Facultad de Ingeniería</b>	<b>Área/Departamento:</b> <b>Laboratorio de computación salas A y B</b>	
La impresión de este documento es una copia no controlada		

## Aplicaciones

La estructura pila tienen varias aplicaciones dentro de la ingeniería, de las más conocidas es la que se utiliza dentro de la memoria RAM de un equipo de cómputo.

La memoria de las computadoras no es un espacio uniforme, el código que se ejecuta utiliza tres diferentes segmentos de memoria: el texto (text), la pila (stack) y el montículo (heap)

Cuando una aplicación inicia, el método principal es invocado y se reserva memoria en la pila o stack. En el segmento de memoria de la pila es donde se alojan las variables requeridas por las funciones del programa. Así mismo, cada vez que se llama una función dentro del programa una sección de la pila, llamada marco o frame, se reserva y es ahí donde las variables de la nueva función son almacenadas.

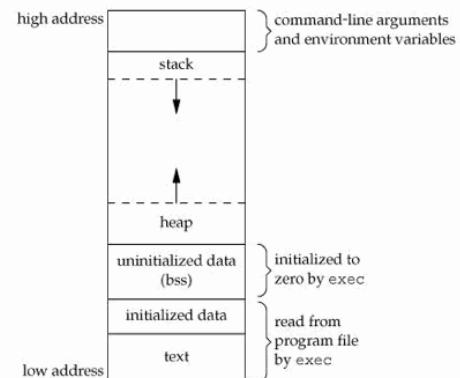


Figura 1. Regiones de la memoria RAM

Cuando una función manda llamar varias funciones, éstas generan un nuevo marco que se va creando uno encima del otro y, cuando las funciones terminan, los marcos se liberan de manera automática en orden inverso (LIFO).

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código: MADO-19 Versión: 01 Página 50/165 Sección ISO 8.3 Fecha de emisión 20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B
La impresión de este documento es una copia no controlada		

## Cola

La cola (o queue) es una estructura de datos lineal, en la cual el elemento obtenido a través de la operación ELIMINAR está predefinido y es el que se encuentra al inicio de la estructura.

La cola implementa la política First-In, First-Out (FIFO), esto es, el primer elemento que se agregó es el primero que se elimina.

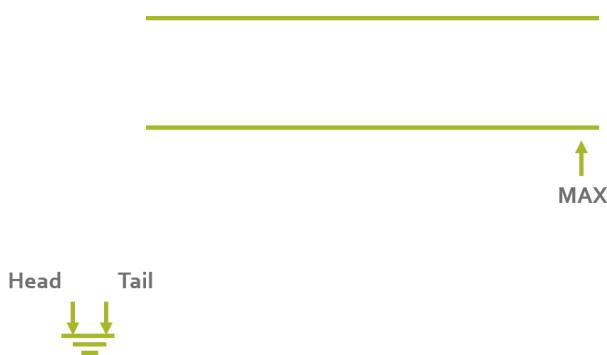
La cola es una estructura de datos de tamaño fijo y cuyas operaciones se realizan por ambos extremos; permite INSERTAR elementos al final de la estructura y permite ELIMINAR elementos por el inicio de la misma. La operación de INSERTAR también se le llama ENCOLAR y la operación de ELIMINAR también se le llama DESENCOLAR.

Para poder diseñar un algoritmo que defina el comportamiento de una COLA se deben considerar 3 casos para ambas operaciones (INSERTAR y ELIMINAR):

- Estructura vacía (caso extremo).
- Estructura llena (caso extremo).
- Estructura con elemento(s) (caso base).

### Cola vacía

La cola posee dos referencias, una al inicio (HEAD) y otra al final (TAIL) de la cola. En una cola vacía ambas referencias (HEAD y TAIL) apuntan a *nulo*.





## Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I

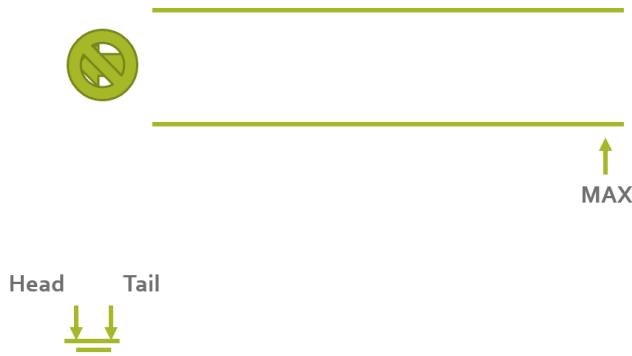
Código:	MADO-19
Versión:	01
Página	51/165
Sección ISO	8.3
Fecha de emisión	20 de enero de 2017

Facultad de Ingeniería

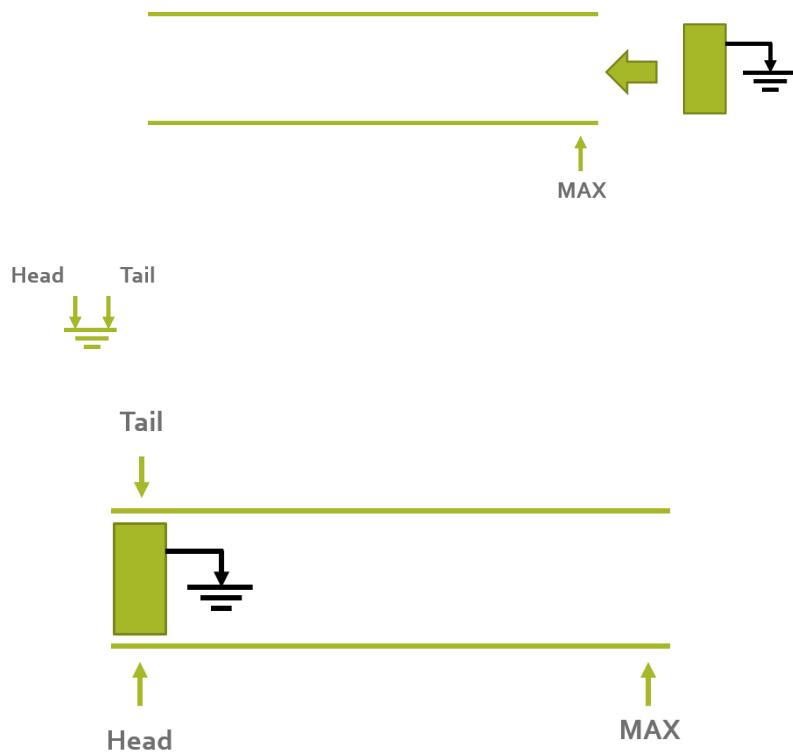
Área/Departamento:  
Laboratorio de computación salas A y B

La impresión de este documento es una copia no controlada

En una cola vacía no es posible desencolar debido a que la estructura no posee elementos.



En una cola vacía sí se pueden encolar elementos, en este caso las referencias HEAD y TAIL apuntan al mismo elemento, que es el único en la estructura.





## Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I

Código:	MADO-19
Versión:	01
Página	52/165
Sección ISO	8.3
Fecha de emisión	20 de enero de 2017

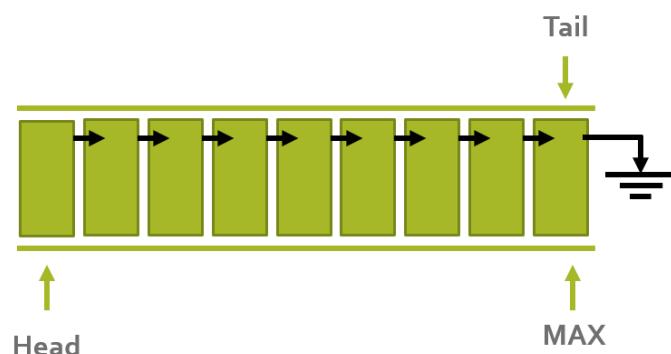
Facultad de Ingeniería

Área/Departamento:  
Laboratorio de computación salas A y B

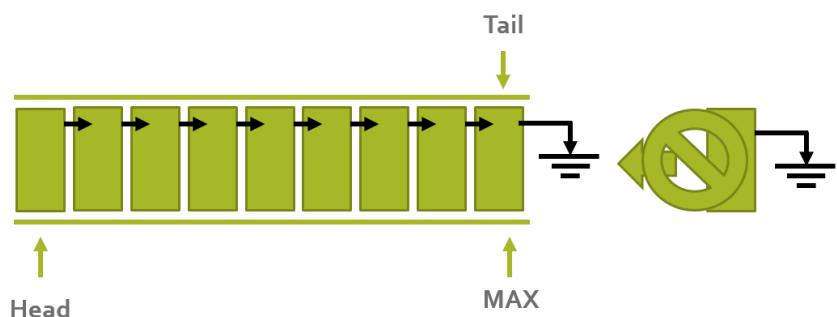
La impresión de este documento es una copia no controlada

### Cola llena

Cuando la referencia a tail de una cola llega a su máxima capacidad de almacenamiento (MAX) se dice que la cola está llena.



En una cola llena no es posible encolar más elementos.





## Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I

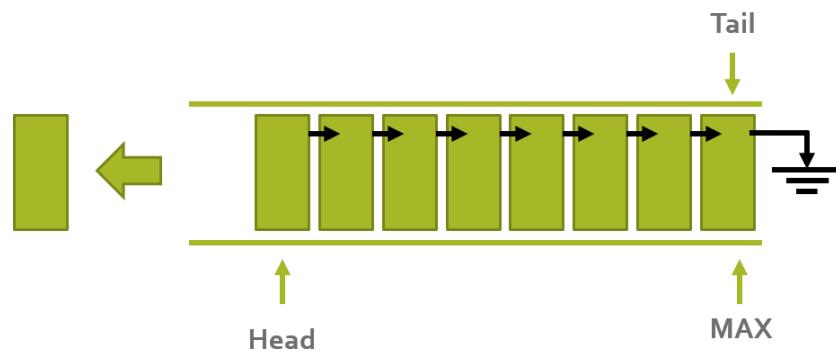
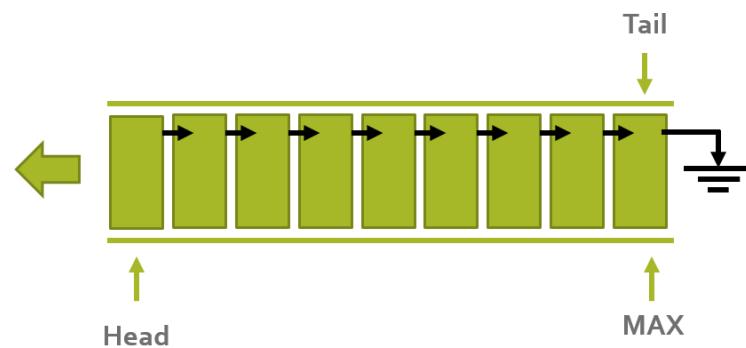
Código:	MADO-19
Versión:	01
Página	53/165
Sección ISO	8.3
Fecha de emisión	20 de enero de 2017

Facultad de Ingeniería

Área/Departamento:  
Laboratorio de computación salas A y B

La impresión de este documento es una copia no controlada

En una cola llena sí se pueden desencolar elementos, en tal caso se obtiene el elemento al que hace referencia head y esta referencia se recorre al siguiente elemento (sucesor).





## Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I

Código:	MADO-19
Versión:	01
Página	54/165
Sección ISO	8.3
Fecha de emisión	20 de enero de 2017

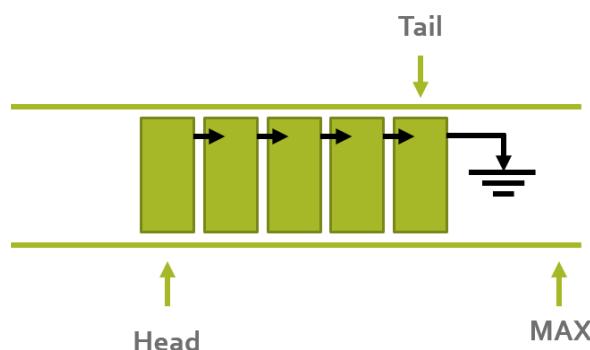
Facultad de Ingeniería

Área/Departamento:  
Laboratorio de computación salas A y B

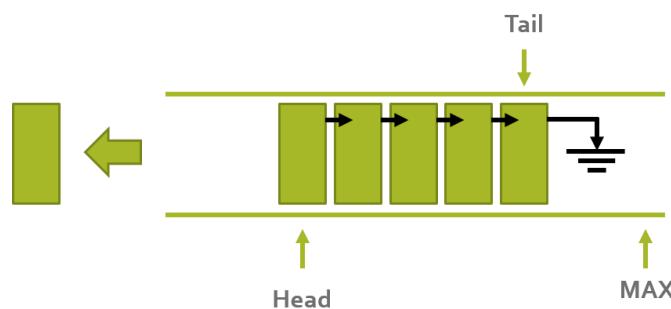
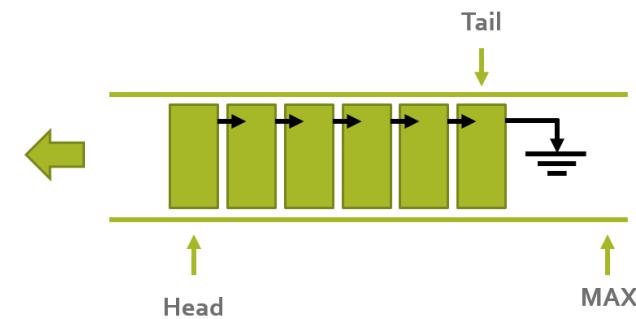
La impresión de este documento es una copia no controlada

### Cola con elementos

Una cola que contiene elementos (sin llegar a su máximo tamaño) representa el caso general de la estructura.



En una cola con elementos es posible desencolar nodos, recorriendo la referencia al inicio de la cola (HEAD) al siguiente elemento de la estructura.





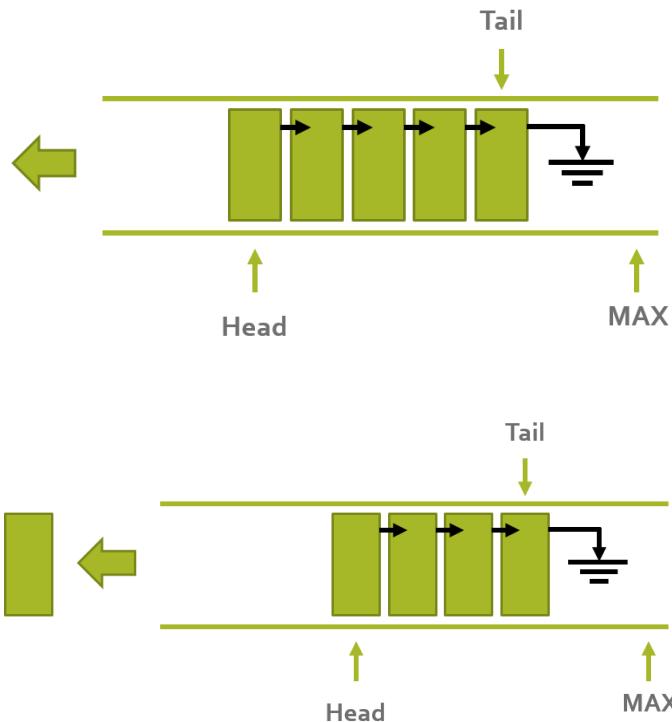
## Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I

Código:	MADO-19
Versión:	01
Página	55/165
Sección ISO	8.3
Fecha de emisión	20 de enero de 2017

Facultad de Ingeniería

Área/Departamento:  
Laboratorio de computación salas A y B

La impresión de este documento es una copia no controlada



Así mismo, se pueden encolar elementos en una cola mientras la referencia al final (TAIL) de la estructura no sea mayor al tamaño máximo de la misma. Cuando se encola un elemento, el nodo al que apunta TAIL tiene como sucesor el nuevo elemento y la referencia a TAIL apunta al nuevo elemento.



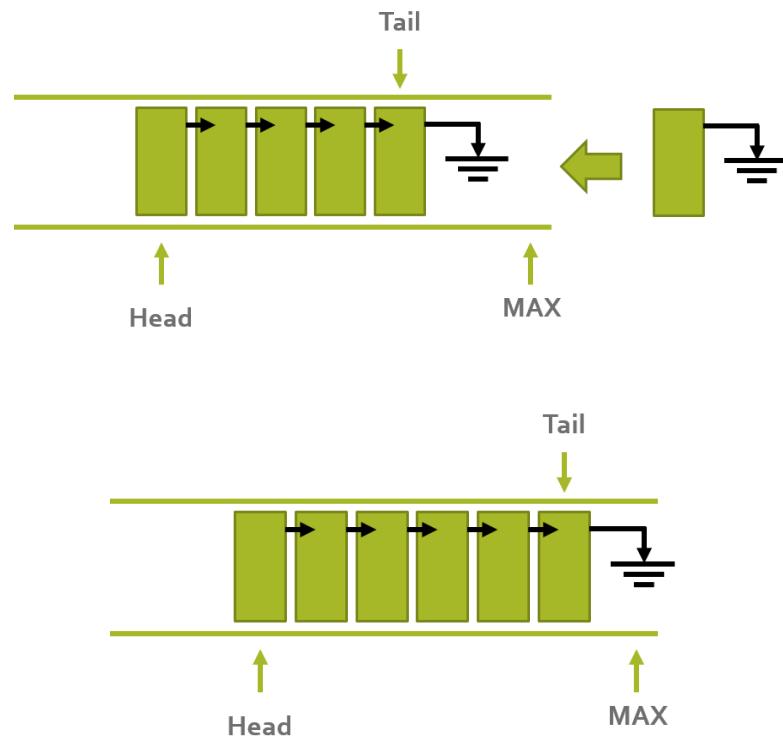
## Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I

Código:	MADO-19
Versión:	01
Página	56/165
Sección ISO	8.3
Fecha de emisión	20 de enero de 2017

Facultad de Ingeniería

Área/Departamento:  
Laboratorio de computación salas A y B

La impresión de este documento es una copia no controlada



### Aplicaciones

La aplicación más conocida de la estructura cola es la que se utiliza en la impresión de documentos.

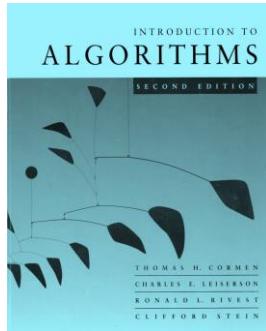
Las impresoras tienen una cantidad de memoria limitada, la cual puede ser inferior al tamaño de un documento que se desea imprimir.

La cola de impresión permite enviar documentos de gran tamaño, o varios documentos, a una impresora sin tener que esperar que se complete la impresión para seguir con la siguiente tarea. Cuando se envía un archivo a imprimir, se crea un archivo de almacenamiento intermedio en formato EMF, donde se almacena lo que se envía a la impresora y las opciones de impresión. Las impresiones se van realizando según vayan llegando los archivos (FIFO).

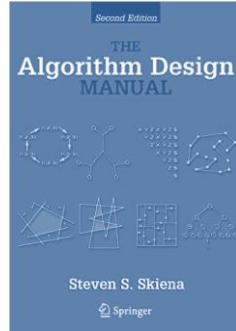
	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código:	MADO-19
		Versión:	01
		Página	57/165
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería	Área/Departamento: Laboratorio de computación salas A y B		
La impresión de este documento es una copia no controlada			

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código:	MADO-19
		Versión:	01
		Página	58/165
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

## Bibliografía



Introduction to Algorithms. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, McGraw-Hill.



The Algorithm Design Manual. Steven S. Skiena, Springer.

Ariel Rodríguez (2010). How knowing C and C++ can help you write better iPhone apps, part 1. [Figura 1]. Consulta: Enero de 2016. Disponible en: <http://akosma.com/2010/10/11/how-knowing-c-and-c-can-help-you-write-better-iphone-apps-part-1/>

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código: MADO-19
		Versión: 01
		Página 59/165
		Sección ISO 8.3
		Fecha de emisión 20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B
La impresión de este documento es una copia no controlada		

# Guía práctica de estudio 06.

## Estructuras de datos lineales: Cola circular y cola doble.

---



***Elaborado por:***  
 M.C. Edgar E. García Cano  
 Ing. Jorge A. Solano Gálvez

***Autorizado por:***  
 M.C. Alejandro Velázquez Mena

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código:	MADO-19
		Versión:	01
		Página	60/165
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

## **Guía práctica de estudio 06.**

### **Estructuras de datos lineales: Cola circular y cola doble.**

#### **Objetivo:**

Revisarás las definiciones, características, procedimientos y ejemplos de las estructuras lineales Cola circular y Cola doble, con la finalidad de que comprendas sus estructuras y puedas implementarlas.

#### **Actividades:**

- Revisar definición y características de la estructura de datos cola circular.
- Revisar definición y características de la estructura de datos cola doble.
- Implementar las estructuras de datos cola circular y cola doble.

#### **Introducción**

La cola (queue o cola simple) es una estructura de datos lineal, en la cual el elemento obtenido a través de la operación ELIMINAR está predefinido y es el que se encuentra al inicio de la misma.

Una cola simple implementa la política First-In, First-Out (FIFO), esto es, el primer elemento que se agregó es el primero que se elimina.

La cola simple es una estructura de datos de tamaño fijo y cuyas operaciones se realizan por ambos extremos; permite INSERTAR elementos al final de la estructura y permite ELIMINAR elementos por el inicio de la misma. La operación de INSERTAR también se le llama ENCOLAR y la operación de ELIMINAR también se le llama DESENCOLAR.

En una cola simple, cuando se eliminan elementos se recorre el apuntador HEAD al siguiente elemento de la estructura, dejando espacios de memoria vacíos al inicio de la misma. Existen dos mejoras de la cola simple que utilizan de manera más eficiente la memoria: la cola circular y la cola doble.

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código: Versión: Página: Sección ISO: Fecha de emisión:	MADO-19 01 61/165 8.3 20 de enero de 2017
Facultad de Ingeniería	<b>Área/Departamento:</b> Laboratorio de computación salas A y B		
La impresión de este documento es una copia no controlada			

## Cola circular

La cola circular es una mejora de la cola simple, debido a que es una estructura de datos lineal en la cual el siguiente elemento del último es, en realidad, el primero. La cola circular utiliza de manera más eficiente la memoria que una cola simple.

Debido a que una cola circular es una mejora de la cola simple, maneja las mismas operaciones para INSERTAR (ENCOLAR) y ELIMINAR (DESENCOLAR).

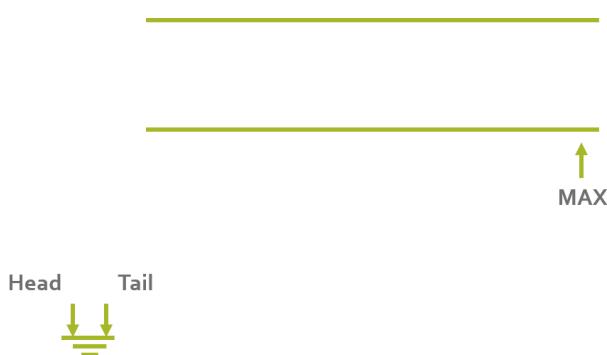
Para diseñar un algoritmo que defina el comportamiento de la cola circular es necesario considerar 3 casos para las operaciones de ENCOLAR y DESENCOLAR:

- Estructura vacía (caso extremo).
- Estructura llena (caso extremo).
- Estructura con elemento(s) (caso base).

En algoritmo de una cola circular para los casos extremos (cuando la estructura está vacía y cuando la estructura está llena) es el mismo con respecto a la cola simple, el único algoritmo que hay que volver a diseñar se presenta en el caso base, cuando la estructura tiene elementos.

### Cola circular vacía

La cola circular posee dos referencias, una al inicio (HEAD) y otra al final (TAIL) de la cola. En una cola circular vacía ambas referencias (HEAD y TAIL) apuntan a *nulo*.





## Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I

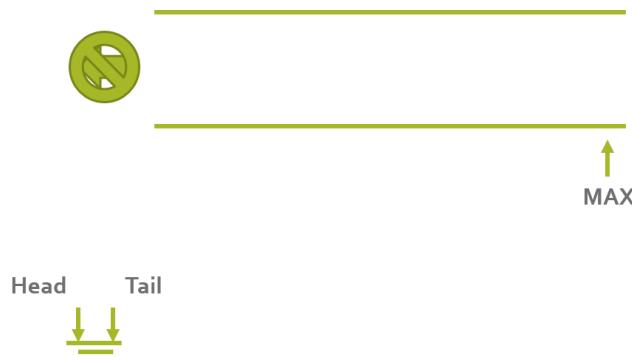
Código:	MADO-19
Versión:	01
Página	62/165
Sección ISO	8.3
Fecha de emisión	20 de enero de 2017

Facultad de Ingeniería

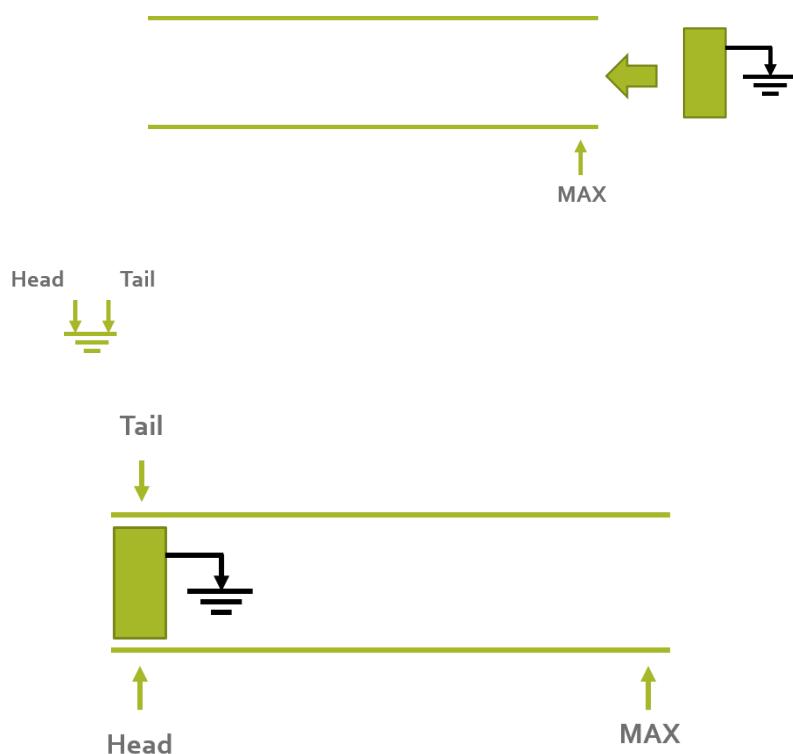
Área/Departamento:  
Laboratorio de computación salas A y B

La impresión de este documento es una copia no controlada

En una cola circular vacía no es posible desencolar debido a que la estructura no posee elementos.



En una cola circular vacía sí se pueden encolar elementos, en este caso las referencias HEAD y TAIL apuntan al mismo elemento, que es el único en la estructura.





## Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I

Código:	MADO-19
Versión:	01
Página	63/165
Sección ISO	8.3
Fecha de emisión	20 de enero de 2017

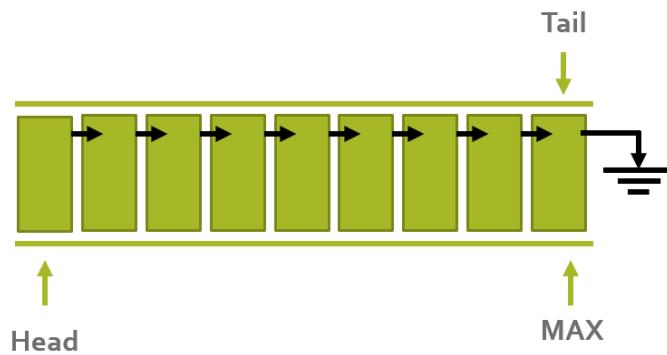
Facultad de Ingeniería

Área/Departamento:  
Laboratorio de computación salas A y B

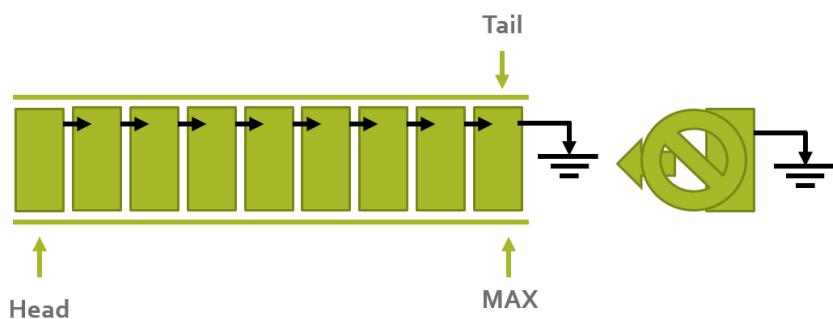
La impresión de este documento es una copia no controlada

### Cola circular llena

Cuando la referencia a TAIL de una cola llega a su máxima capacidad de almacenamiento (MAX) se dice que la cola está llena.



En una cola circular llena no es posible encolar más elementos.





## Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I

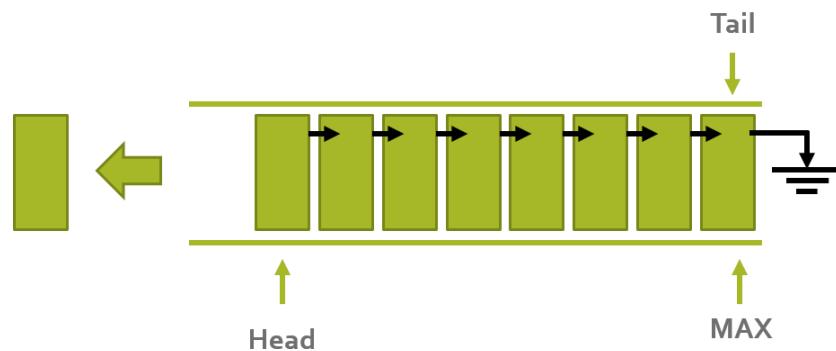
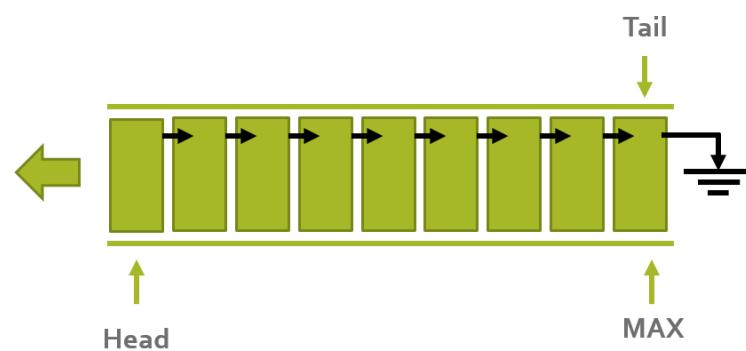
Código:	MADO-19
Versión:	01
Página	64/165
Sección ISO	8.3
Fecha de emisión	20 de enero de 2017

Facultad de Ingeniería

Área/Departamento:  
Laboratorio de computación salas A y B

La impresión de este documento es una copia no controlada

En una cola circular llena sí se pueden desencolar elementos, en tal caso se obtiene el elemento al que hace referencia HEAD y esta referencia se recorre al siguiente elemento.





## Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I

Código:	MADO-19
Versión:	01
Página	65/165
Sección ISO	8.3
Fecha de emisión	20 de enero de 2017

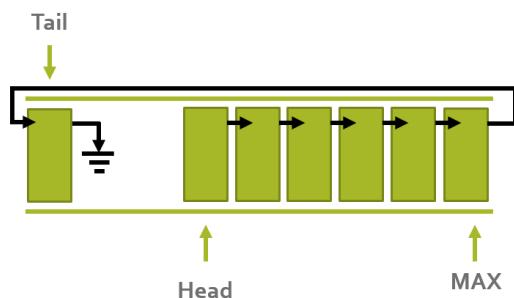
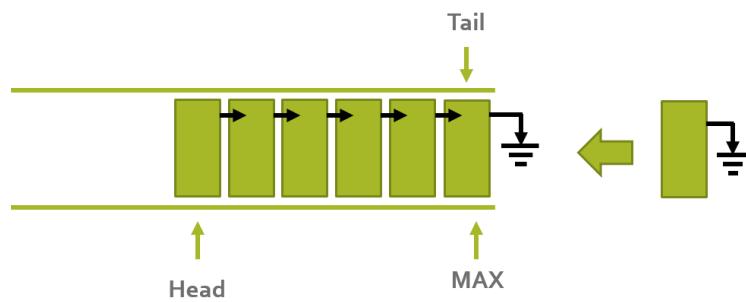
Facultad de Ingeniería

Área/Departamento:  
Laboratorio de computación salas A y B

La impresión de este documento es una copia no controlada

### Cola circular con elementos

En una cola circular con elementos, cuando se intenta insertar un nuevo elemento hay que tener en cuenta el número de los elementos dentro de la estructura y no la referencia TAIL y MAX. Por lo tanto, se debe verificar si el número de elementos que tiene la estructura es menor al número máximo de elementos definidos, si es así, existe espacio para alojar el nuevo elemento y el nuevo nodo se puede insertar.





## Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I

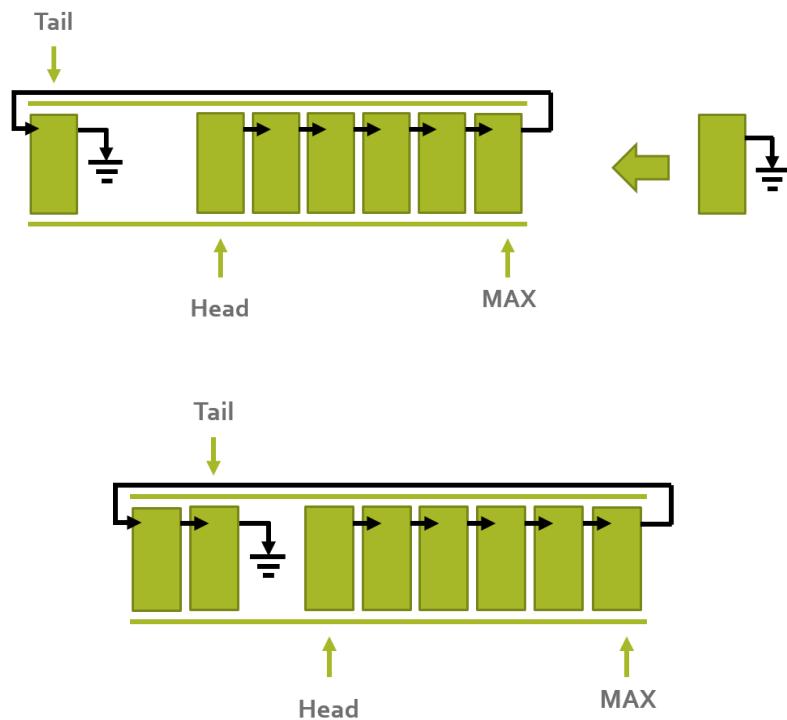
Código:	MADO-19
Versión:	01
Página	66/165
Sección ISO	8.3
Fecha de emisión	20 de enero de 2017

Facultad de Ingeniería

Área/Departamento:  
Laboratorio de computación salas A y B

La impresión de este documento es una copia no controlada

Cada vez que se desee almacenar un nuevo elemento en la estructura se debe revisar el número de elementos insertados y comparar con el número máximo de elementos que se pueden almacenar.



La posibilidad de insertar (ENCOLAR) elementos mientras se tenga espacio disponible hace más eficiente el uso de la memoria, ya que los espacios liberados cada vez que se DESENCOLA un nodo se pueden volver a utilizar, a diferencia de la cola simple.



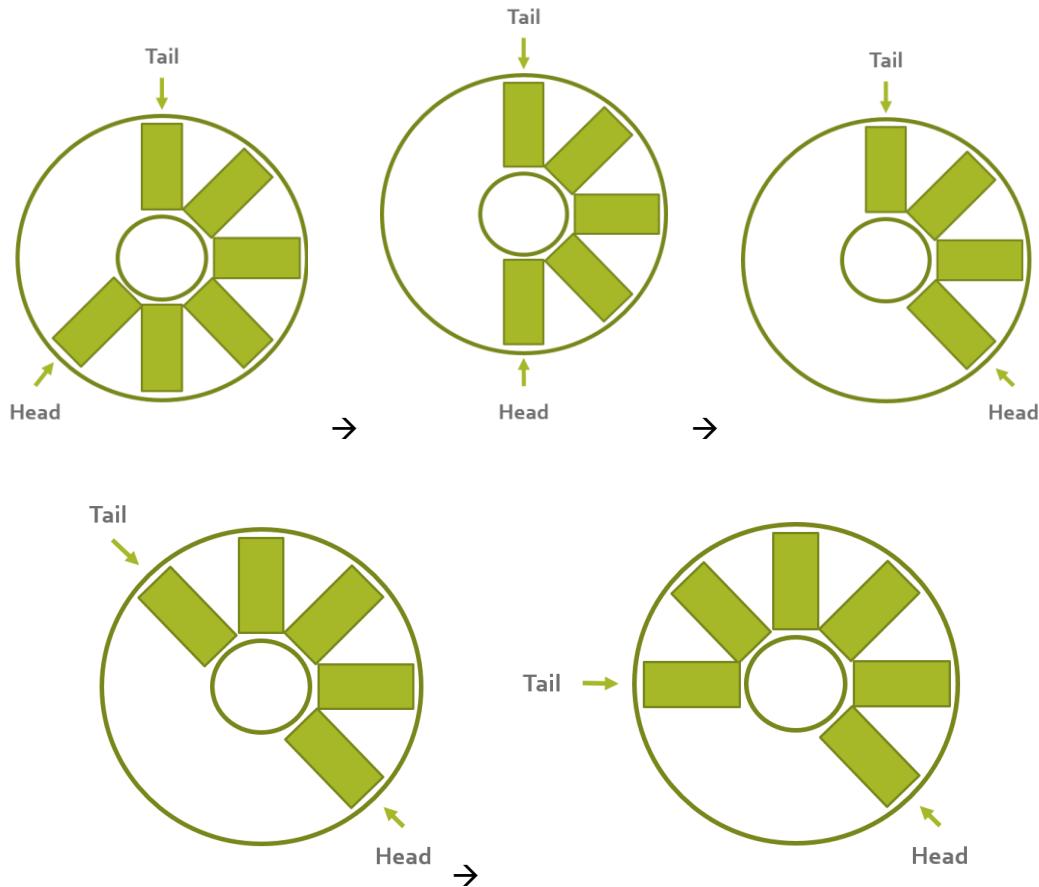
## Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I

Código:	MADO-19
Versión:	01
Página	67/165
Sección ISO	8.3
Fecha de emisión	20 de enero de 2017

Facultad de Ingeniería

Área/Departamento:  
Laboratorio de computación salas A y B

La impresión de este documento es una copia no controlada



	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código:	MADO-19
		Versión:	01
		Página	68/165
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

## Aplicación

La elección de un elemento dentro de un conjunto de datos es muy común en diversas aplicaciones, sobre todo en juegos de consola. La selección de un conjunto de elementos finitos donde a partir del último elemento se puede regresar al primero utiliza, de manera implícita, una cola circular: selección de un personaje, selección de un arma, cambios de uniformes, etc.



Figura 1. Cambio de armas en Resident evil.

En general, cuando dentro de una aplicación se puede recorrer un conjunto de elementos finito e invariable en el tiempo y el sucesor del último elemento es el primero se tiene una cola circular.



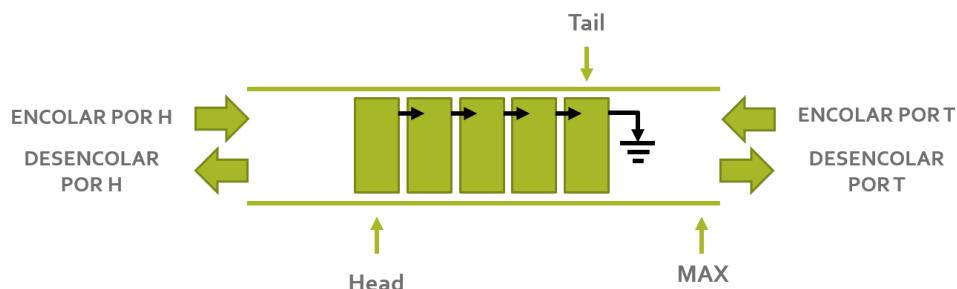
Figura 2. Elegir de uniformes en FIFA.

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código: MADO-19 Versión: 01 Página 69/165 Sección ISO 8.3 Fecha de emisión 20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B
La impresión de este documento es una copia no controlada		

## Cola doble

Una cola doble (o bcola) es una estructura de datos tipo cola simple en la cual las operaciones ENCOLAR y DESENCOLAR se pueden realizar por ambos extremos de la estructura, es decir, en una cola doble se pueden realizar las operaciones:

- ENCOLAR POR HEAD
- DESENCOLAR POR HEAD
- ENCOLAR POR TAIL
- DESENCOLAR POR TAIL



La cola doble es una mejora de una cola simple debido a que es posible realizar operaciones de inserción por ambos extremos de la estructura, permitiendo con esto utilizar el máximo espacio disponible de la estructura.

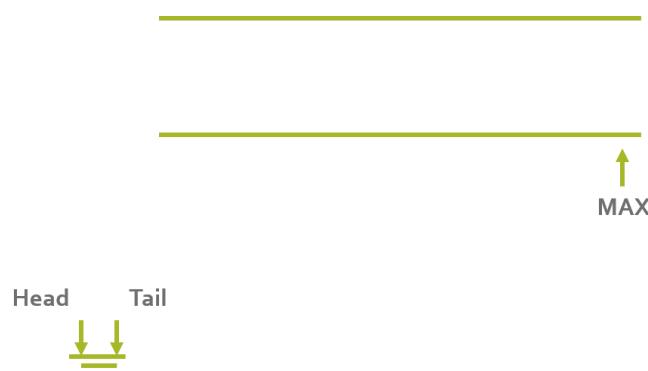
Para poder diseñar un programa que defina el comportamiento de una COLA DOBLE se deben considerar 3 casos para las 4 operaciones (INSERTAR y ELIMINAR tanto por T como por H):

- Estructura vacía (caso extremo).
- Estructura llena (caso extremo).
- Estructura con elemento(s) (caso base).

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código: MADO-19 Versión: 01 Página 70/165 Sección ISO 8.3 Fecha de emisión 20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B
La impresión de este documento es una copia no controlada		

### Cola doble vacía

La cola doble posee dos referencias, una al inicio (HEAD) y otra al final (TAIL) de la cola. En una cola doble vacía ambas referencias (HEAD y TAIL) apuntan a *nulo*.



En una cola doble vacía no es posible desencolar debido a que la estructura no posee elementos.





## Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I

Código:	MADO-19
Versión:	01
Página	71/165
Sección ISO	8.3
Fecha de emisión	20 de enero de 2017

Facultad de Ingeniería

Área/Departamento:  
Laboratorio de computación salas A y B

La impresión de este documento es una copia no controlada

DESENCOLAR  
PORT



Head      Tail

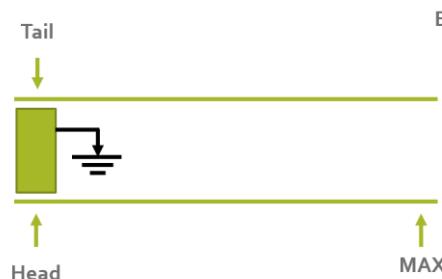
En una cola doble vacía sí se pueden encolar elementos tanto por HEAD como por TAIL, y, en este caso, las referencias HEAD y TAIL apuntan al mismo elemento, que es el único en la estructura.

ENCOLAR POR  
T



Head      Tail

ENCOLAR POR  
T





## Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I

Código:	MADO-19
Versión:	01
Página	72/165
Sección ISO	8.3
Fecha de emisión	20 de enero de 2017

Facultad de Ingeniería

Área/Departamento:  
Laboratorio de computación salas A y B

La impresión de este documento es una copia no controlada

ENCOLAR POR  
H



Head      Tail  
↓          ↓

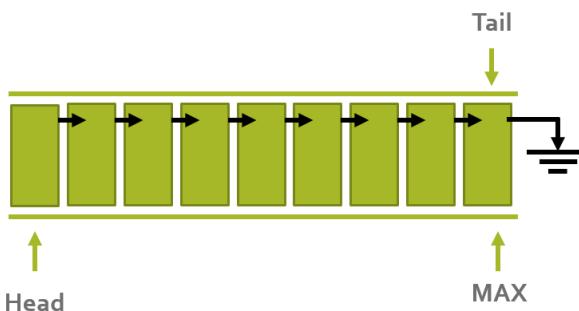


Head

Tail

### Cola doble llena

Cuando el número de elementos de la estructura es igual a la capacidad máxima de almacenamiento (MAX) se dice que la cola está llena.





## Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I

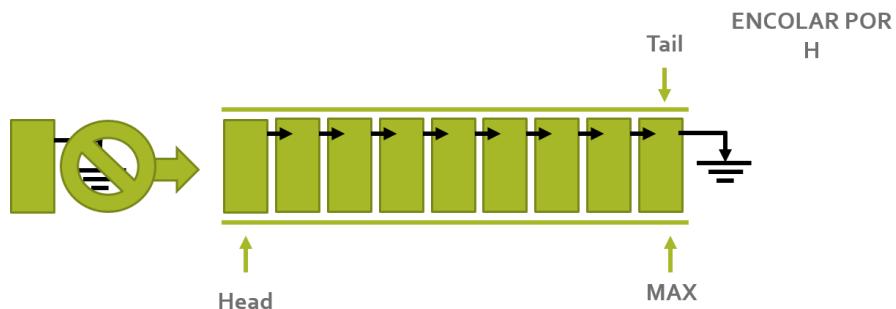
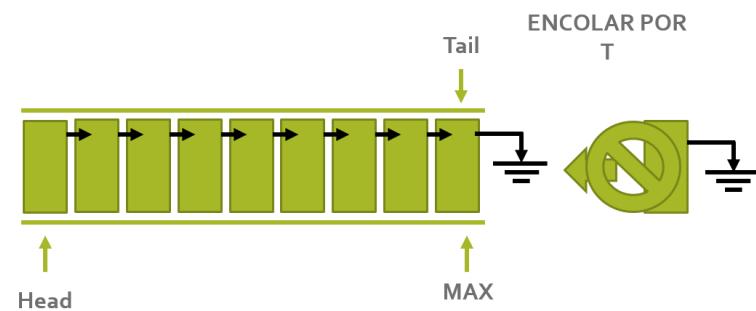
Código:	MADO-19
Versión:	01
Página	73/165
Sección ISO	8.3
Fecha de emisión	20 de enero de 2017

Facultad de Ingeniería

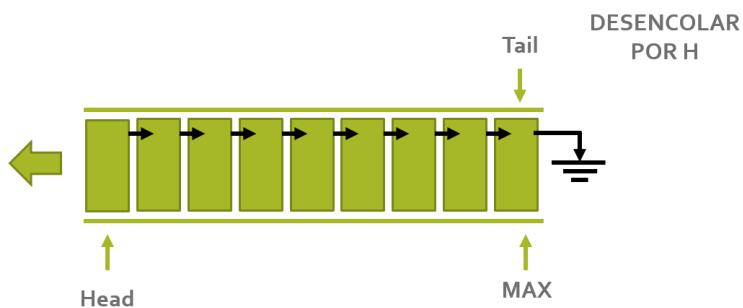
Área/Departamento:  
Laboratorio de computación salas A y B

La impresión de este documento es una copia no controlada

En una cola doble llena no es posible encolar más elementos, ni por HEAD ni por TAIL.



En una cola doble llena sí se pueden desencolar elementos tanto por HEAD como por TAIL. Cuando se desencola por el inicio de la estructura se obtiene el elemento al que hace referencia HEAD y esta referencia se recorre al siguiente elemento (sucesor).





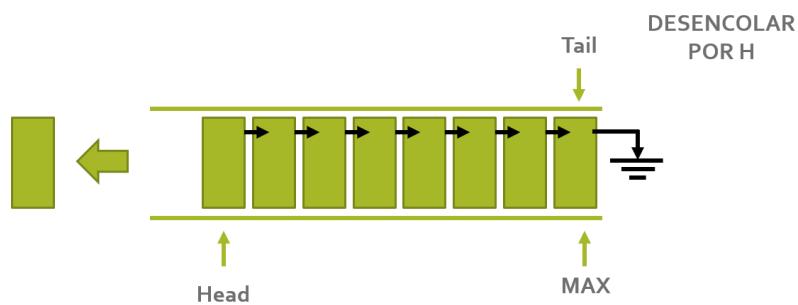
## Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I

Código:	MADO-19
Versión:	01
Página	74/165
Sección ISO	8.3
Fecha de emisión	20 de enero de 2017

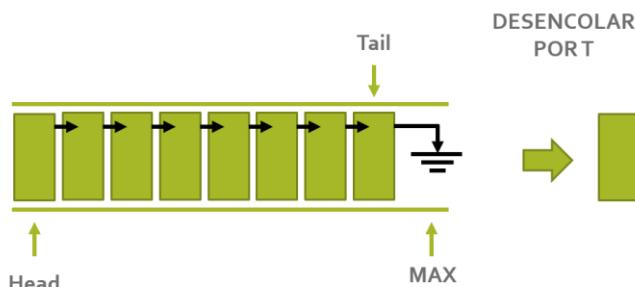
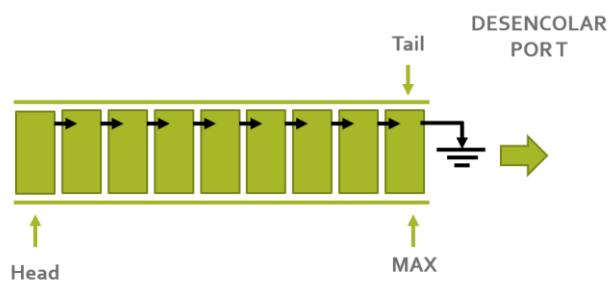
Facultad de Ingeniería

Área/Departamento:  
Laboratorio de computación salas A y B

La impresión de este documento es una copia no controlada



Cuando se desencola por el final de la estructura se obtiene el elemento al que hace referencia TAIL y esta referencia se recorre al elemento anterior (predecesor).





## Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I

Código:	MADO-19
Versión:	01
Página	75/165
Sección ISO	8.3
Fecha de emisión	20 de enero de 2017

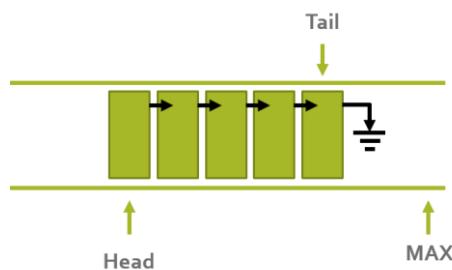
Facultad de Ingeniería

Área/Departamento:  
Laboratorio de computación salas A y B

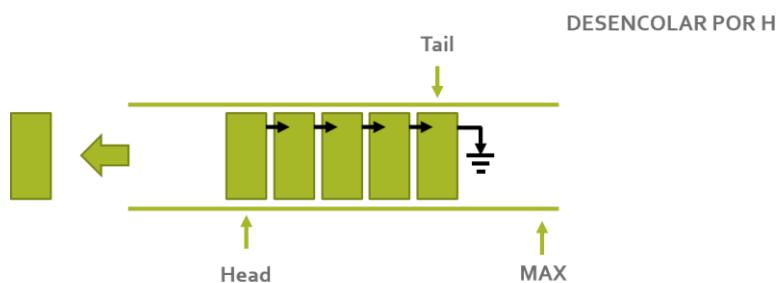
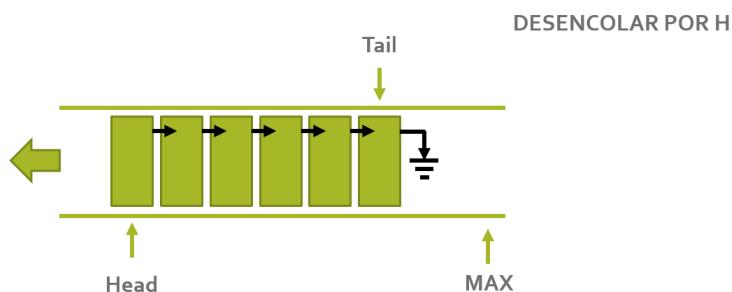
La impresión de este documento es una copia no controlada

### Cola doble con elementos

Una cola doble que contiene elementos (sin llegar a su máximo tamaño) representa el caso general de la estructura.



En una cola doble con elementos es posible desencolar nodos, tanto por HEAD como por TAIL. Cuando se desencola por el inicio de la estructura, se debe recorrer la referencia al inicio de la cola (HEAD) al siguiente elemento de la estructura (sucesor).





## Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I

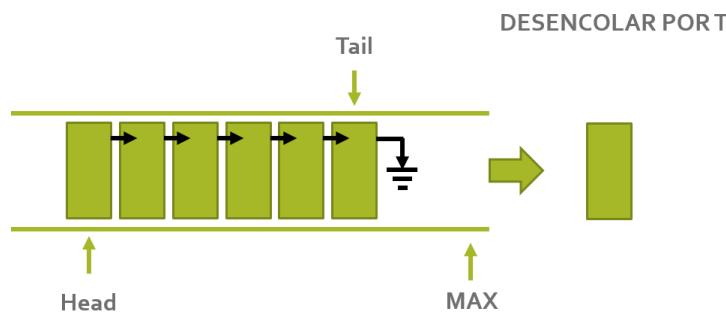
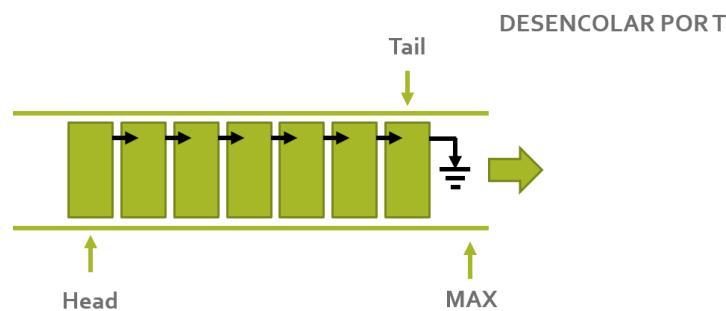
Código:	MADO-19
Versión:	01
Página	76/165
Sección ISO	8.3
Fecha de emisión	20 de enero de 2017

Facultad de Ingeniería

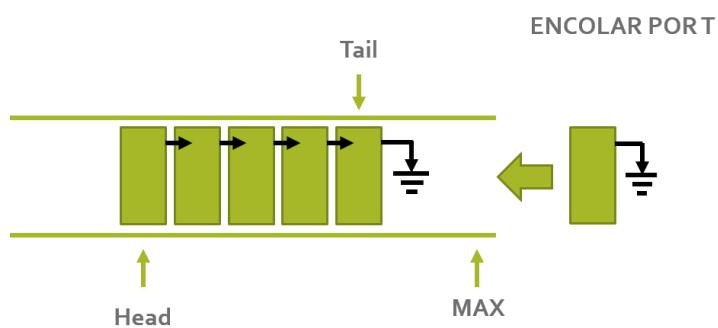
Área/Departamento:  
Laboratorio de computación salas A y B

La impresión de este documento es una copia no controlada

Cuando se desencola por el final de la estructura, se debe recorrer la referencia al final de la cola (TAIL) al elemento anterior de la estructura (predecesor).



Así mismo, se pueden encolar elementos en una cola doble mientras no se exceda la capacidad máxima de la estructura. Es posible encolar elementos tanto por HEAD como por TAIL. Cuando se encola un elemento por el final, el nodo al que apunta TAIL tiene como sucesor el nuevo nodo y la referencia a TAIL apunta al nuevo elemento.





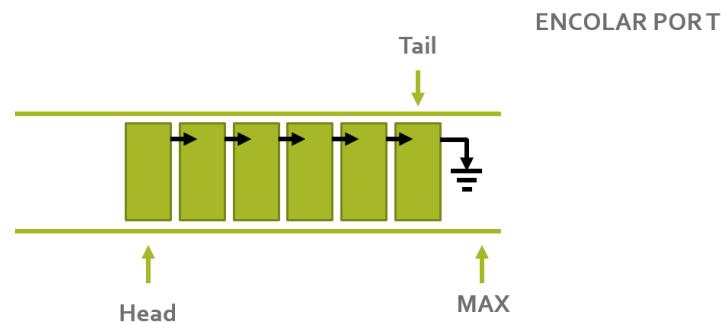
## Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I

Código:	MADO-19
Versión:	01
Página	77/165
Sección ISO	8.3
Fecha de emisión	20 de enero de 2017

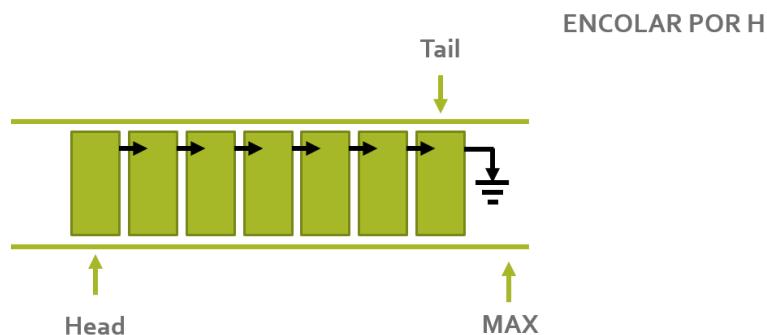
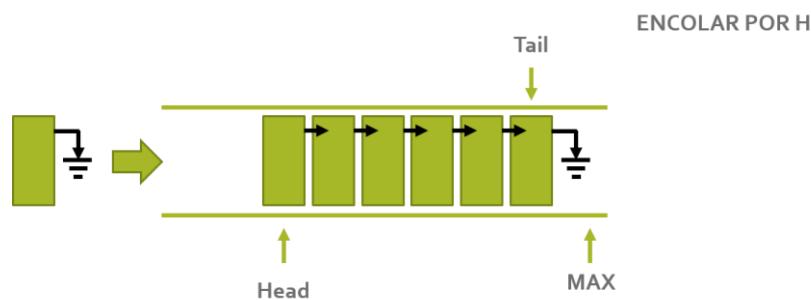
Facultad de Ingeniería

Área/Departamento:  
Laboratorio de computación salas A y B

La impresión de este documento es una copia no controlada



Cuando se encola un elemento por el inicio, el nodo al que apunta HEAD tiene como predecesor el nuevo nodo y la referencia a HEAD apunta al nuevo elemento.



	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código:	MADO-19
		Versión:	01
		Página	78/165
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería	Área/Departamento: Laboratorio de computación salas A y B		
La impresión de este documento es una copia no controlada			

## Aplicación

Una cola doble en la vida cotidiana podría verse como un abuso, es decir, formar a personas delante de la fila, a pesar de haber llegado después de los que ya están formados (y sin embargo pasa). Sin embargo, en las ciencias de la computación hay muchas aplicaciones que trabajan así, ya que hay procesos que tienen prioridad y deben ser ejecutados antes que otros procesos menos importantes.

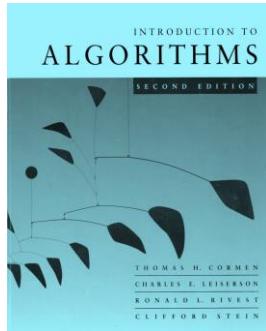
Dentro del sistema operativo no todas las aplicaciones tienen la misma exigencia en cuanto a tiempo y recursos, existen procesos que se tienen que ejecutar de manera inmediata ante algún suceso que se presente en el sistema, mientras que otros solo tengan que procesar información y puedan (y deban) esperar a que el sistema se recupere.

Todo sistema operativo define un valor de urgencia con que debe ejecutarse una aplicación, es decir, define la prioridad que tiene un proceso frente a otros que se estén ejecutando en el sistema.

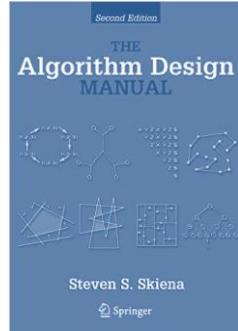
La prioridad se expresa como un número entero. Por tanto, si un proceso A tiene asignada una prioridad PR<sub>x</sub> y un proceso B tiene asignada una prioridad PR<sub>y</sub>, si PR<sub>x</sub> > PR<sub>y</sub>, el proceso A será más prioritario que el proceso B y, por ende, será el que se ejecute primero. Por otro lado, si existen varios procesos en ejecución con prioridad PR<sub>y</sub>, si llega un proceso con mayor prioridad, por ejemplo, PR<sub>x</sub>, éste último se ejecutará primero, es decir, no se encola al final si no al inicio del conjunto.

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código:	MADO-19
		Versión:	01
		Página	79/165
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

## Bibliografía



Introduction to Algorithms. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, McGraw-Hill.



The Algorithm Design Manual. Steven S. Skiena, Springer.

CAPCOM (2012). CAPCOM: RESIDENT EVIL 6 | Manual web oficial [Figura 1]. Consulta: Enero de 2016. Disponible en: <http://game.capcom.com/manual/bio6/es/page-74.html>

SergioGameplayer (2014). FIFA World Cup Brazil 2014 - Juego Completo Menús, Modos de Juego Equipos Uniformes y mas! [Figura 2]. Consulta: Enero de 2016. Disponible en: <https://www.youtube.com/watch?v=Pyu0Xp7MVJI>

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código: MADO-19
		Versión: 01
		Página 80/165
		Sección ISO 8.3
		Fecha de emisión 20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B
La impresión de este documento es una copia no controlada		

# Guía práctica de estudio 07.

## Estructuras de datos lineales: Lista simple y lista circular.

---



***Elaborado por:***  
 M.C. Edgar E. García Cano  
 Ing. Jorge A. Solano Gálvez

***Autorizado por:***  
 M.C. Alejandro Velázquez Mena

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código: MADO-19 Versión: 01 Página 81/165 Sección ISO 8.3 Fecha de emisión 20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B
La impresión de este documento es una copia no controlada		

## **Guía práctica de estudio 07. Estructuras de datos lineales: Lista simple y lista circular.**

### **Objetivo:**

Revisarás las definiciones, características, procedimientos y ejemplos de las estructuras lineales Lista simple y Lista circular, con la finalidad de que comprendas sus estructuras y puedas implementarlas.

### **Actividades:**

- Revisar definición y características de la estructura de datos lista simple.
- Revisar definición y características de la estructura de datos lista circular.
- Implementar las estructuras de datos lista simple y lista circular.

### **Introducción**

Las listas son un tipo de estructura de datos lineal y dinámica. Es lineal porque cada elemento tiene un único predecesor y un único sucesor, y es dinámica porque su tamaño no es fijo y se puede definir conforme se requiera. Las operaciones básicas dentro de una lista son BUSCAR, INSERTAR Y ELIMINAR.

### **Lista simple**

Una lista simple (también conocida como lista ligada o lista simplemente ligada) está constituida por un conjunto de nodos alineados de manera lineal (uno después de otro) y unidos entre sí por una referencia.

A diferencia de un arreglo, el cual también es un conjunto de nodos alineados de manera lineal, el orden está determinado por una referencia, no por un índice, y el tamaño no es fijo.

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código: MADO-19 Versión: 01 Página 82/165 Sección ISO 8.3 Fecha de emisión 20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B
La impresión de este documento es una copia no controlada		

La unidad básica de una lista simple es un elemento o nodo. Cada elemento de la lista es un objeto que contiene la información que se desea almacenar, así como una referencia (NEXT) al siguiente elemento (SUCESOR).



Para poder diseñar un algoritmo que defina el comportamiento de una LISTA LIGADA se deben considerar 2 casos para cada operación (BUSCAR, INSERTAR y ELIMINAR):

- Estructura vacía (caso extremo).
- Estructura con elemento(s) (caso base).

### Buscar

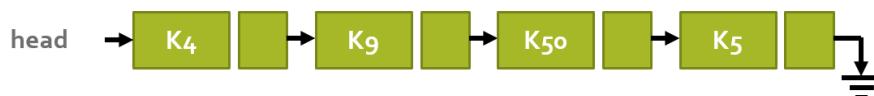
El método debe buscar el primer elemento que coincida con la llave K dentro de la lista L, a través de una búsqueda lineal simple, regresando un apuntador a dicho elemento si éste se encuentra en la lista o nulo en caso contrario.

Una lista simple vacía no contiene elementos, la referencia al inicio de la misma (head) apunta a nulo, por lo tanto, en una lista vacía no es posible buscar elementos.

Head



Una lista simple con elementos puede contener de 1 a n elementos, en tal caso, la referencia al inicio (HEAD) apunta al primer elemento de la lista. Es posible recorrer la lista a través de la referencia (NEXT) de cada nodo hasta llegar al que apunta a nulo, el cuál será el último elemento. Por lo tanto, dentro de una lista simple con elementos es posible buscar una llave K.

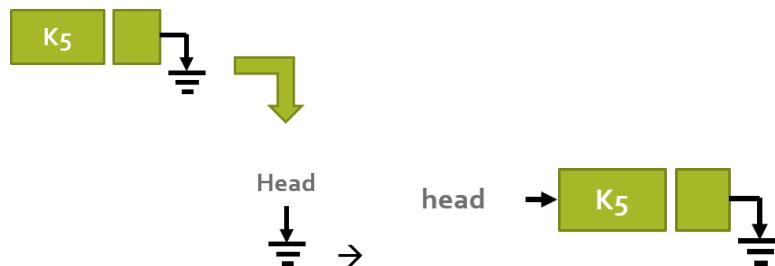


	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código: MADO-19 Versión: 01 Página 83/165 Sección ISO 8.3 Fecha de emisión 20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B
La impresión de este documento es una copia no controlada		

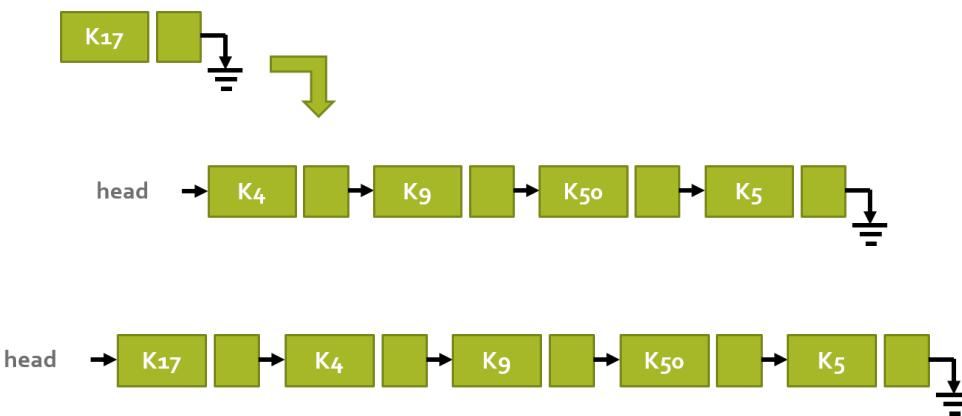
## Insertar

Dado un nodo  $x$  que contenga una llave  $k$  previamente establecida, el método INSERTAR agrega el elemento  $x$  al inicio de la lista.

Es posible insertar elementos tanto en una lista simple vacía como en una lista simple con elementos. Cuando se inserta un nuevo elemento en una lista simple vacía la referencia al inicio de la lista (HEAD) apunta al nodo insertado.



Cuando se inserta un nuevo elemento en una lista simple con elementos, la referencia del nuevo nodo (NEXT) apunta al mismo nodo al que apunta el inicio de la lista (HEAD) y ahora HEAD apunta al nuevo nodo.



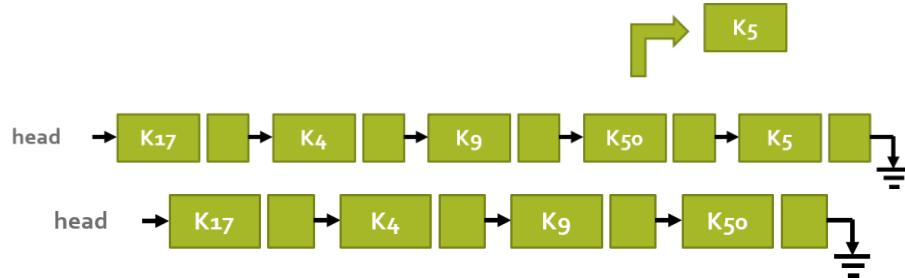
	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código: MADO-19 Versión: 01 Página: 84/165 Sección ISO: 8.3 Fecha de emisión: 20 de enero de 2017
Facultad de Ingeniería	<b>Área/Departamento:</b> <b>Laboratorio de computación salas A y B</b>	
La impresión de este documento es una copia no controlada		

Borrar

El método elimina el elemento  $x$  de la lista  $L$  (si es que éste se encuentra en la estructura). Para eliminar un elemento de la lista primero es necesario saber la ubicación del nodo a eliminar, por lo tanto, primero se debe realizar una búsqueda del elemento.

En una lista simple vacía no es posible eliminar, debido a que esta estructura no contiene elementos.

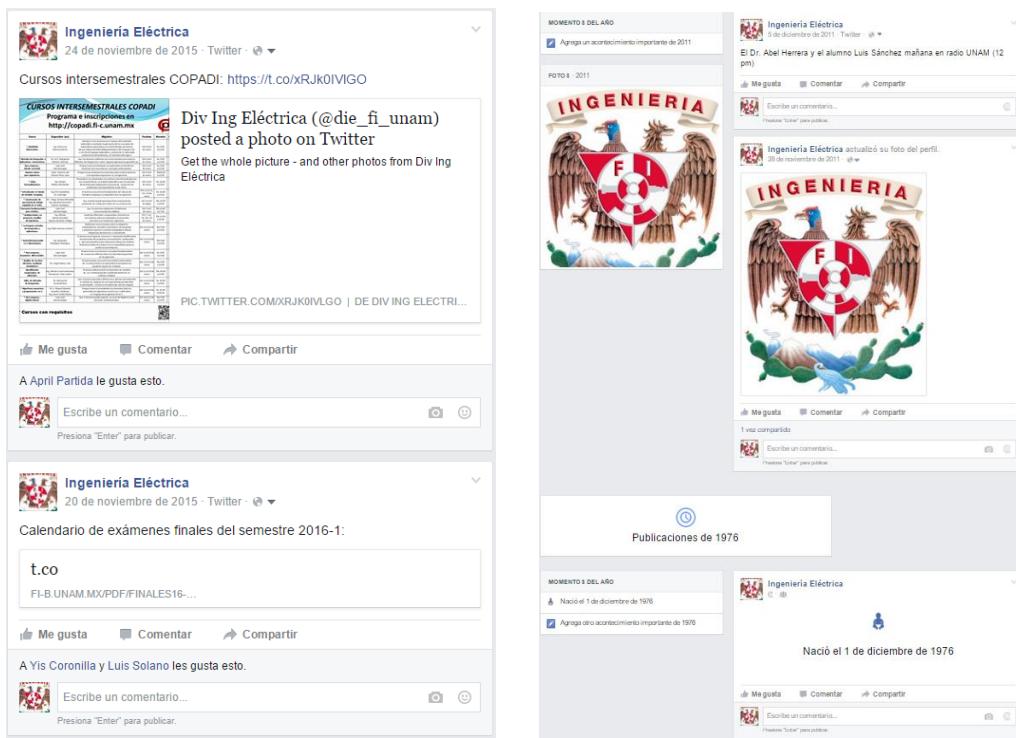
Para eliminar un nodo en una lista simple con elementos, primero se debe buscar el elemento a eliminar, una vez encontrado el nodo en la lista, se deben mover las referencias de la estructura de tal manera de que el antecesor del nodo a eliminar apunte al sucesor del mismo.



	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código: Versión: Página: Sección ISO: Fecha de emisión:	MADO-19 01 85/165 8.3 20 de enero de 2017
Facultad de Ingeniería	Área/Departamento: <b>Laboratorio de computación salas A y B</b>		
La impresión de este documento es una copia no controlada			

## Aplicación

Las listas son una de las estructuras de datos más utilizadas en las ciencias de la computación. Por ejemplo, cualquier red social utiliza una lista simple, en la que cada elemento tiene un único sucesor que sería la siguiente publicación, hasta llegar a la última.



La imagen muestra una secuencia de capturas de pantalla de Twitter. Se observan tres publicaciones:

- Publicación 1:** "Ingeniería Eléctrica" (@die\_fi\_unam) publicó una foto el 24 de noviembre de 2015. La foto muestra el escudo de la Facultad de Ingeniería UNAM. El texto adjunto dice: "Div Ing Eléctrica (@die\_fi\_unam) posted a photo on Twitter Get the whole picture - and other photos from Div Ing Eléctrica".
- Publicación 2:** "Ingeniería Eléctrica" (@die\_fi\_unam) publicó una foto el 5 de diciembre de 2011. La foto muestra el escudo de la Facultad de Ingeniería UNAM. El texto adjunto dice: "El Dr. Abel Herrera y el alumno Luis Sánchez mañana en radio UNAM (12 pm)".
- Publicación 3:** "Ingeniería Eléctrica" (@die\_fi\_unam) actualizó su foto de perfil el 20 de noviembre de 2011. La foto muestra el escudo de la Facultad de Ingeniería UNAM.

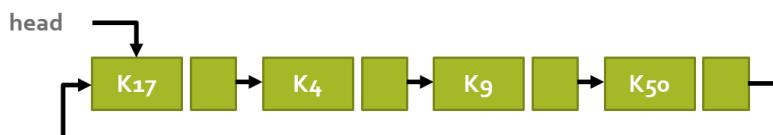
**Figura 1.** Inicio y fin de publicaciones en Twitter.

Se pueden realizar ene cantidad de publicaciones, siempre insertando por delante (TAIL) y la última publicación (TAIL) no tiene sucesor (NULO).

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código: MADO-19 Versión: 01 Página: 86/165 Sección ISO: 8.3 Fecha de emisión: 20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B
La impresión de este documento es una copia no controlada		

## Listas circulares

Una lista circular es una lista simplemente ligada modificada, donde el apuntador del elemento que se encuentra al final de la lista (TAIL) apunta al primer elemento de la lista (HEAD).



### Buscar

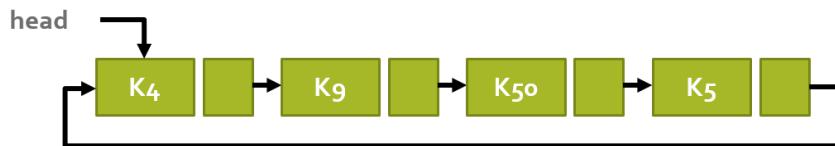
El método debe buscar el primer elemento que coincida con la llave  $K$  dentro de la lista  $L$ , a través de una búsqueda lineal simple, regresando un apuntador a dicho elemento si éste se encuentra en la lista o nulo en caso contrario.

Una lista circular vacía no contiene elementos, la referencia al inicio de la misma (HEAD) apunta a NULO, por lo tanto, en una lista vacía no es posible buscar elementos.



Una lista circular con elementos puede contener de 1 a  $n$  elementos, en tal caso, la referencia al inicio (HEAD) apunta al primer elemento de la lista y la referencia a NEXT del último elemento apunta al primer elemento. Es posible recorrer la lista a través de la referencia (NEXT) de cada nodo, hay que tener en cuenta el número de elementos de la lista, ya que el último elemento apunta al inicio de la estructura y, por tanto, se puede recorrer de manera infinita. Dentro de una lista circular con elementos es posible buscar una llave  $K$ .

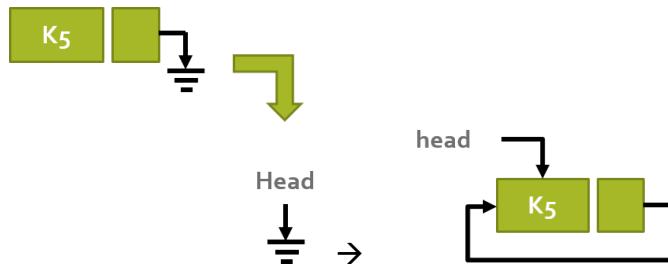
	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código: MADO-19 Versión: 01 Página 87/165 Sección ISO 8.3 Fecha de emisión 20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B
La impresión de este documento es una copia no controlada		



### Insertar

Dado un nodo  $x$  que contenga una llave  $K$  previamente establecida, el método INSERTAR agrega el elemento  $x$  al inicio de la lista.

Es posible insertar elementos tanto en una lista circular vacía como en una lista circular con elementos. Cuando se inserta un nuevo elemento en una lista circular vacía la referencia al inicio de la lista (HEAD) apunta al nodo insertado y la referencia a NEXT del nodo apunta a sí mismo.



Cuando se inserta un nuevo elemento en una lista circular con elementos, la referencia del nuevo nodo (NEXT) apunta al mismo nodo al que apunta el inicio de la lista (HEAD) y ahora HEAD apunta al nuevo nodo. Así mismo, el último nodo de la estructura (TAIL) apunta al primer elemento.



## Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I

Código:	MADO-19
Versión:	01
Página	88/165
Sección ISO	8.3
Fecha de emisión	20 de enero de 2017

Facultad de Ingeniería

Área/Departamento:  
Laboratorio de computación salas A y B

La impresión de este documento es una copia no controlada

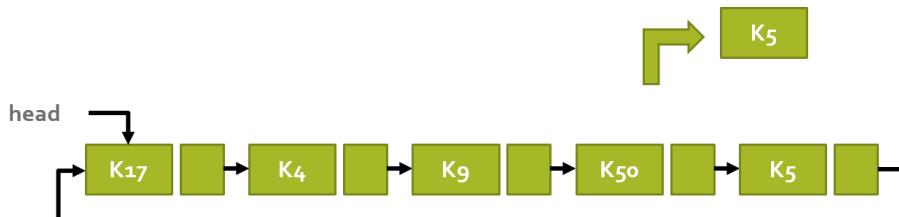


### Borrar

El método elimina el elemento  $x$  de la lista  $L$  (si es que éste se encuentra en la estructura). Para eliminar un elemento de la lista primero es necesario saber la ubicación del nodo a eliminar, por lo tanto, primero se debe realizar una búsqueda del elemento.

En una lista circular vacía no es posible eliminar, debido a que esta estructura no contiene elementos.

Para eliminar un nodo en una lista circular con elementos, primero se debe buscar el elemento a eliminar, una vez encontrado el nodo en la lista, se deben mover las referencias de la estructura de tal manera de que el antecesor del nodo a eliminar apunte al sucesor del mismo.





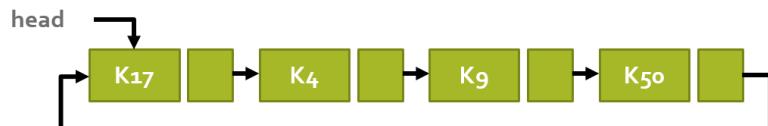
**Manual de prácticas del  
Laboratorio de Estructuras de  
datos y algoritmos I**

Código:	MADO-19
Versión:	01
Página	89/165
Sección ISO	8.3
Fecha de emisión	20 de enero de 2017

Facultad de Ingeniería

Área/Departamento:  
Laboratorio de computación salas A y B

La impresión de este documento es una copia no controlada



	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código: MADO-19 Versión: 01 Página 90/165 Sección ISO 8.3 Fecha de emisión 20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B
La impresión de este documento es una copia no controlada		

## Aplicación

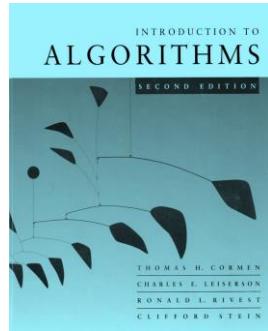
Una lista de canciones se puede reproducir de manera ordenada o de manera desordenada (aleatoriedad). Así mismo, se puede repetir la lista de reproducción de manera automática, es decir, el sucesor del último elemento de la lista es el primer elemento de la lista, lo que genera una lista circular.



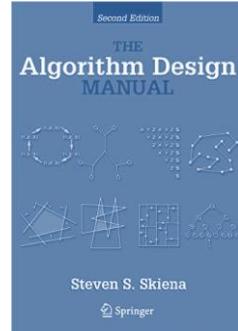
Figura 2. Repetir lista de reproducción en avs4you.

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código:	MADO-19
		Versión:	01
		Página	91/165
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

## Bibliografía



Introduction to Algorithms. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, McGraw-Hill.



The Algorithm Design Manual. Steven S. Skiena, Springer.

@die\_fi\_unam [Figura 1]. Consulta: Enero de 2016. Disponible en: [https://twitter.com/die\\_fi\\_unam](https://twitter.com/die_fi_unam)

Online Media Technologies Ltd. AVS Media Player [Figura 2]. Consulta: Enero de 2016. Disponible en: [https://twitter.com/die\\_fi\\_unam](https://twitter.com/die_fi_unam)

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código: MADO-19
		Versión: 01
		Página 92/165
		Sección ISO 8.3
		Fecha de emisión 20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B
La impresión de este documento es una copia no controlada		

# Guía práctica de estudio 08:

## Estructuras de datos lineales: Lista doblemente ligada y doblemente ligada circular.

---



*Elaborado por:*

M.C. Edgar E. García Cano  
Ing. Jorge A. Solano Gálvez

*Autorizado por:*

M.C. Alejandro Velázquez Mena

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código: MADO-19
		Versión: 01
		Página 93/165
		Sección ISO 8.3
		Fecha de emisión 20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B
La impresión de este documento es una copia no controlada		

## **Guía práctica de estudio 08: Estructuras de datos lineales: Lista doblemente ligada y lista doblemente ligada circular.**

### **Objetivo:**

Revisarás las definiciones, características, procedimientos y ejemplos de las estructuras lineales Lista doblemente ligada y Lista doblemente ligada circular, con la finalidad de que comprendas sus estructuras y puedas implementarlas.

### **Actividades:**

- Revisar definición y características de la estructura de datos lista doblemente ligada.
- Revisar definición y características de la estructura de datos lista doblemente ligada circular.
- Implementar las estructuras de datos lista doblemente ligada y lista doblemente ligada circular.

### **Introducción**

Las listas son un tipo de estructura de datos lineal y dinámica. Es lineal porque cada elemento tiene un único predecesor y un único sucesor, y es dinámica porque su tamaño no es fijo y se puede definir conforme se requiera. Las operaciones básicas dentro de una lista son BUSCAR, INSERTAR Y ELIMINAR.

### **Lista doblemente ligada**

Una lista doblemente ligada (o lista doble) está constituida por un conjunto de nodos alineados de manera lineal (uno después de otro) y unidos entre sí por dos referencias, una al sucesor (NEXT) y una al predecesor (PREV).

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código: MADO-19 Versión: 01 Página 94/165 Sección ISO 8.3 Fecha de emisión 20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B
La impresión de este documento es una copia no controlada		

La unidad básica de una lista doble es el elemento o nodo. Cada elemento de la lista es un objeto que contiene la información que se desea almacenar, así como dos referencias, una al siguiente elemento (NEXT) y otra al elemento anterior (PREV).



Dado un elemento  $x$  en una lista doble,  $\text{NEXT}[x]$  apunta al sucesor de  $x$  y  $\text{PREV}[x]$  apunta al predecesor de  $x$ . Si  $\text{PREV}[x] = \text{NULL}$ , el elemento  $x$  no tiene predecesor y, por ende, es el primer elemento (o HEAD) de la lista. Si  $\text{NEXT}[x] = \text{NULL}$ , el elemento  $x$  no tiene sucesor y, por ende, es el último elemento (o TAIL) de la lista. El atributo  $\text{HEAD}[L]$  apunta al primer elemento de la lista, si  $\text{HEAD}[L] = \text{NULL}$  entonces se puede afirmar que la lista está vacía.

Para poder diseñar un algoritmo que defina el comportamiento de una LISTA DOBLEMENTE LIGADA se deben considerar 2 casos para cada operación (buscar, insertar y eliminar):

- Estructura vacía (caso extremo).
- Estructura con elemento(s) (caso base).

### Buscar

El método debe buscar el primer elemento que coincida con la llave  $K$  dentro de la lista  $L$ , a través de una búsqueda lineal simple, regresando un apuntador a dicho elemento si éste se encuentra en la lista o nulo en caso contrario. La búsqueda se puede realizar iniciando por HEAD o iniciando por TAIL

Una lista doble vacía no contiene elementos, la referencia al inicio de la misma (HEAD) apunta a nulo, por lo tanto, en una lista vacía no es posible buscar elementos.



	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código: <b>MADO-19</b> Versión: <b>01</b> Página <b>95/165</b> Sección ISO <b>8.3</b> Fecha de emisión <b>20 de enero de 2017</b>
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B
La impresión de este documento es una copia no controlada		

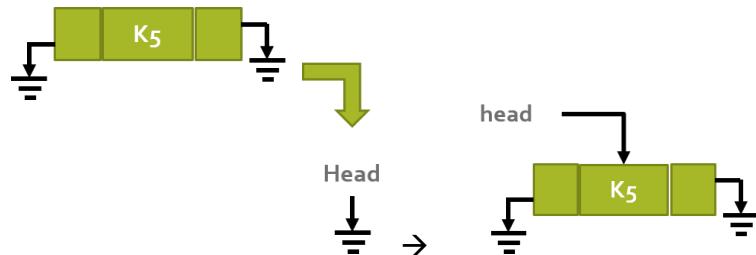
Una lista doble con elementos puede contener de 1 a n elementos, en tal caso, la referencia al inicio (HEAD) apunta al primer elemento de la lista. Es posible recorrer la lista a través de la referencia siguiente (NEXT) de cada nodo hasta llegar al que apunta a nulo, el cuál será el último elemento. Así mismo, si se posee una referencia al final de la lista (TAIL), es posible recorrer la lista a través de la referencia anterior (PREV) de cada nodo hasta llegar al que apunta a nulo, el cual será el primer elemento. Por lo tanto, dentro de una lista doble con elementos sí es posible buscar una llave K.



### Insertar

Dado un nodo  $x$  que contenga una llave  $K$  previamente establecida, el método INSERTAR agrega el elemento  $x$  al inicio de la lista.

Es posible insertar elementos tanto en una lista doble vacía como en una lista doble con elementos. Cuando se inserta un nuevo elemento en una lista doblemente ligada vacía la referencia al inicio de la lista (HEAD) apunta al nodo insertado.





## Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I

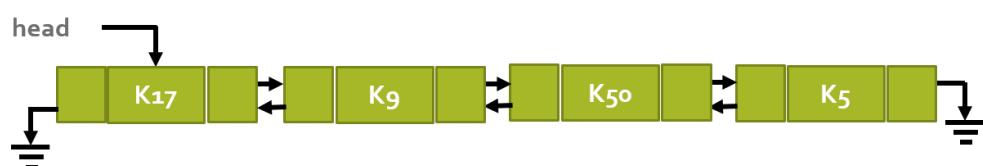
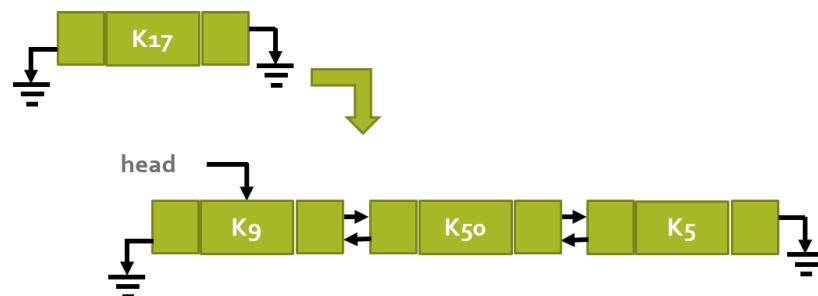
Código:	MADO-19
Versión:	01
Página	96/165
Sección ISO	8.3
Fecha de emisión	20 de enero de 2017

Facultad de Ingeniería

Área/Departamento:  
Laboratorio de computación salas A y B

La impresión de este documento es una copia no controlada

Cuando se inserta un nuevo elemento en una lista doblemente ligada con elementos, la referencia del nuevo nodo (NEXT) apunta al mismo nodo al que apunta el inicio de la lista (HEAD), la referencia anterior (PREV) del nodo siguiente (NEXT) del inicio de la lista apunta al nuevo nodo, y head también apunta al nuevo nodo.



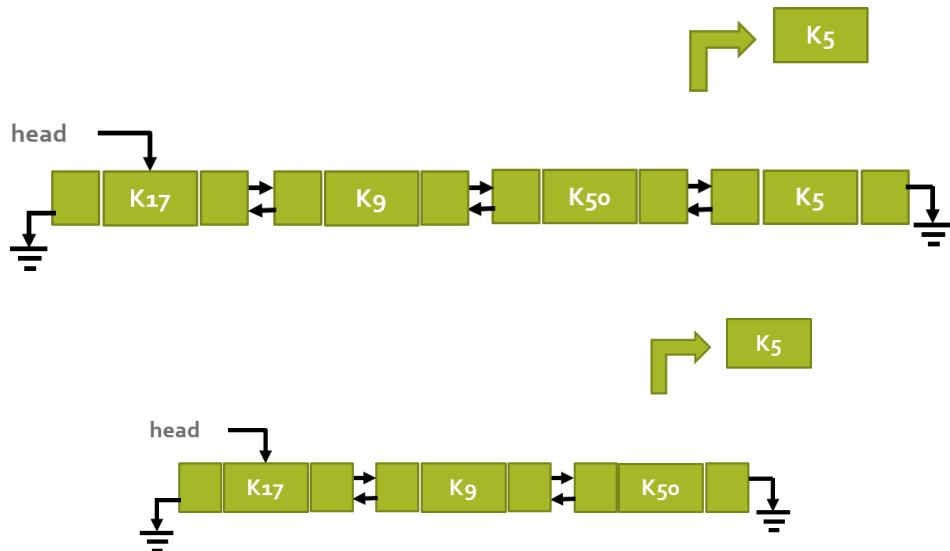
	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código: MADO-19 Versión: 01 Página 97/165 Sección ISO 8.3 Fecha de emisión 20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B
La impresión de este documento es una copia no controlada		

Borrar

El método elimina el elemento  $x$  de la lista  $L$  (si es que éste se encuentra en la estructura). Para eliminar un elemento de la lista primero es necesario saber la ubicación del nodo a eliminar, por lo tanto, primero se debe realizar una búsqueda del nodo.

En una lista doblemente ligada vacía no es posible eliminar, debido a que esta estructura no contiene elementos.

Para eliminar un nodo en una lista doblemente ligada con elementos, primero se debe buscar el elemento a eliminar, una vez encontrado el nodo en la lista, se deben mover las referencias de la estructura de tal manera de que el antecesor del nodo a eliminar apunte al sucesor del mismo y el predecesor del nodo sucesor apunte al predecesor del nodo (PREV).



	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código: <b>MADO-19</b> Versión: <b>01</b> Página <b>98/165</b> Sección ISO <b>8.3</b> Fecha de emisión <b>20 de enero de 2017</b>
<b>Facultad de Ingeniería</b>		<b>Área/Departamento:</b> <b>Laboratorio de computación salas A y B</b>
La impresión de este documento es una copia no controlada		

## Aplicación

Un manejador web de una cuenta de correo es una lista de elementos donde cada nodo representa un mensaje de correo con características particulares: asunto, remitente(s), destinatario(s), adjunto(s), mensaje, etc. Dichos nodos se relacionan entre sí con un orden específico (por fechas), es decir, el orden de inserción siempre es por enfrente (head). Además, es posible recorrer la lista de izquierda a derecha o de derecha a izquierda, es decir, de correos más recientes a correos más antiguos o viceversa. Cuando se llega a un extremo de la lista (ya sea en fechas recientes o en fechas antiguas), ya no es posible seguir recorriendo la lista.

Notificación: Diplomado: Afinación y rendimiento de base de datos vie 10 de Jul de 2015 16:00 - 21:00 (lmsr.fi.unam@gmail.com) 405 de 495 < >

Google Calendar <calendar-notification@google.com> 10/7/15 para mí

**Diplomado: Afinación y rendimiento de base de datos** [más detalles »](#)

Cuándo: vie 10 de Jul de 2015 16:00 – 21:00 Hora central: Ciudad de México  
Dónde: Laboratorio de Microsoft ([mapa](#))  
Calendario: [lmsr.fi.unam@gmail.com](mailto:lmsr.fi.unam@gmail.com)  
Quién: • Laboratorio LMSR: organizador

Invitación de Google Calendar  
Recibes este mensaje de correo electrónico en la dirección [lmsr.fi.unam@gmail.com](mailto:lmsr.fi.unam@gmail.com) de la cuenta porque estás suscrito para recibir notificaciones del calendario [lmsr.fi.unam@gmail.com](https://www.google.com/calendar). Si ya no quieres recibir estos correos, inicia sesión en <https://www.google.com/calendar> y cambia la configuración de las notificaciones para este calendario.  
Si reenvías esta invitación, es posible que cualquier destinatario pueda modificar tu respuesta de confirmación de asistencia. Más información en <https://support.google.com/calendar/answer/37135#forwarding>

Notificación: Diplomado: Afinación y rendimiento de base de datos sáb 11 de Jul de 2015 09:00 - 14:00 (lmsr.fi.unam@gmail.com) 404 de 495 < >

Google Calendar <calendar-notification@google.com> 11/7/15 para mí

**Diplomado: Afinación y rendimiento de base de datos** [más detalles »](#)

Cuándo: sáb 11 de Jul de 2015 09:00 – 14:00 Hora central: Ciudad de México  
Dónde: Laboratorio de Microsoft ([mapa](#))  
Calendario: [lmsr.fi.unam@gmail.com](mailto:lmsr.fi.unam@gmail.com)  
Quién: • Laboratorio LMSR: organizador

Invitación de Google Calendar  
Recibes este mensaje de correo electrónico en la dirección [lmsr.fi.unam@gmail.com](mailto:lmsr.fi.unam@gmail.com) de la cuenta porque estás suscrito para recibir notificaciones del calendario [lmsr.fi.unam@gmail.com](https://www.google.com/calendar). Si ya no quieres recibir estos correos, inicia sesión en <https://www.google.com/calendar> y cambia la configuración de las notificaciones para este calendario.  
Si reenvías esta invitación, es posible que cualquier destinatario pueda modificar tu respuesta de confirmación de asistencia. Más información en <https://support.google.com/calendar/answer/37135#forwarding>

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código: <b>MADO-19</b> Versión: <b>01</b> Página <b>99/165</b> Sección ISO <b>8.3</b> Fecha de emisión <b>20 de enero de 2017</b>
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B
La impresión de este documento es una copia no controlada		

Notificación: OCJP México First jue 9 de Jul de 2015 18:00 - 20:00 (lmsr.fi.unam@gmail.com) [Recibidos](#)

Google Calendar <calendar-notification@google.com> para mí

**OCJP México First**  
Cuándo jue 9 de Jul de 2015 18:00 - 20:00 Hora central: Ciudad de México  
Dónde Laboratorio de Microsoft ([mapa](#))  
Calendario [lmsr.fi.unam@gmail.com](#)  
Quién • Laboratorio LMSR organizador

más detalles

Invitación de Google Calendar  
Recibes este mensaje de correo electrónico en la dirección [lmsr.fi.unam@gmail.com](#) de la cuenta porque estás suscrito para recibir notificaciones del calendario [lmsr.fi.unam@gmail.com](#).  
Si ya no quieres recibir estos correos, inicia sesión en [https://www.google.com/calendar/](#) y cambia la configuración de las notificaciones para este Calendario.  
Si reenvías esta invitación, es posible que cualquier destinatario pueda modificar tu respuesta de confirmación de asistencia. Más información en [https://support.google.com/calendar/answer/37135#forwarding](#)

**Figura 1.** Consulta de correo electrónico vía web.

Así mismo, los manejadores de correo web separan los correos en bloques de *n-elementos*. Esta separación permite mostrar solamente los *n-elementos* a la vez, de tal manera que para ver el resto de los correos se debe pasar a la siguiente página.

201–250 de 495

Principal	Social	Promociones
<input type="checkbox"/> Google Calendar	Notificación: Arquitectura de Computadoras - C	6/10/15
<input type="checkbox"/> Google Calendar	Notificación: Programación Avanzada y Métod	6/10/15
<input type="checkbox"/> Google Calendar	Notificación: Laboratorio de computación gráfi	5/10/15
<input type="checkbox"/> Google Calendar	Notificación: Laboratorio de computación gráfi	5/10/15
<input type="checkbox"/> Google Calendar	Notificación: Diseño de Sistemas Digitales Gpc	5/10/15

151–200 de 495

Principal	Social	Promociones
<input type="checkbox"/> Google Calendar	Notificación: Programación Avanzada y Métod	27/10/15
<input type="checkbox"/> Google Calendar	Notificación: Laboratorio de computación gráfi	26/10/15
<input type="checkbox"/> Google Calendar	Notificación: Laboratorio de computación gráfi	26/10/15
<input type="checkbox"/> Google Calendar	Notificación: Diseño de Sistemas Digitales Gpc	26/10/15
<input type="checkbox"/> Google Calendar	Notificación: Diplomado: Afinación y rendimier	24/10/15

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código: MADO-19 Versión: 01 Página 100/165 Sección ISO 8.3 Fecha de emisión 20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B
La impresión de este documento es una copia no controlada		

251–300 de 495 < > ⚙

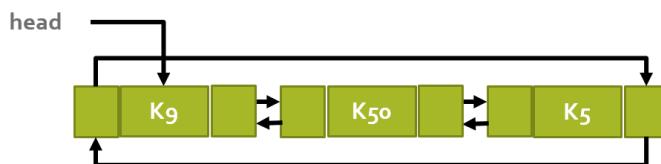
Principal	Social	Promociones	+
<input type="checkbox"/> Google Calendar	Notificación: Algoritmos y estructuras de datos	15/9/15	
<input type="checkbox"/> Google Calendar	Notificación: Arquitectura de Computadoras - C	15/9/15	
<input type="checkbox"/> Google Calendar	Notificación: Programación Avanzada y Métodos	15/9/15	
<input type="checkbox"/> Google Calendar	Notificación: Laboratorio de computación gráfi	14/9/15	
<input type="checkbox"/> Google Calendar	Notificación: Laboratorio de computación gráfi	14/9/15	

**Figura 2.** Paginación de correos en un manejador vía web.

Por lo tanto, dentro de un manejador de correo web tanto la lista de correos como la paginación de correos constituyen una lista doblemente ligada.

## **Lista doblemente ligada circular**

Una lista doblemente ligada circular (o lista doble circular) es una lista doblemente ligada modificada, donde la referencia siguiente (NEXT) del elemento que se encuentra al final de la lista (TAIL) en lugar de apuntar a nulo, apunta al primer elemento de la lista (HEAD).



### Buscar

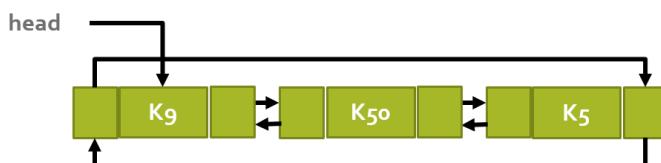
El método debe buscar el primer elemento que coincide con la llave  $K$  dentro de la lista  $L$ , a través de una búsqueda lineal simple, regresando un apuntador a dicho elemento si éste se encuentra en la lista o nulo en caso contrario.

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código: MADO-19 Versión: 01 Página 101/165 Sección ISO 8.3 Fecha de emisión 20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B
La impresión de este documento es una copia no controlada		

Una lista doble circular vacía no contiene elementos, la referencia al inicio de la misma (HEAD) apunta a nulo, por lo tanto, en una lista vacía no es posible buscar elementos.



Una lista doble circular con elementos puede contener de 1 a n elementos, en tal caso, la referencia al inicio (HEAD) apunta al primer elemento de la lista y la referencia a NEXT del último elemento apunta al primer elemento. Es posible recorrer la lista a través de la referencia al sucesor (NEXT) de cada nodo, hay que tener en cuenta el número de elementos de la lista, ya que el último elemento apunta al inicio de la estructura y, por tanto, se puede recorrer de manera infinita. Así mismo, si se posee una referencia al final de la lista (TAIL), es posible recorrer la lista a través de la referencia al predecesor (PREV) de cada nodo, hay que tener en cuenta el número de elementos de la lista, ya que el primer elemento apunta al final de la estructura y, por tanto, se puede recorrer de manera infinita. Dentro de una lista circular con elementos es posible buscar una llave K.



Insertar

Dado un nodo  $x$  que contenga una llave  $K$  previamente establecida, el método INSERTAR agrega el elemento  $x$  al inicio de la lista.



## Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I

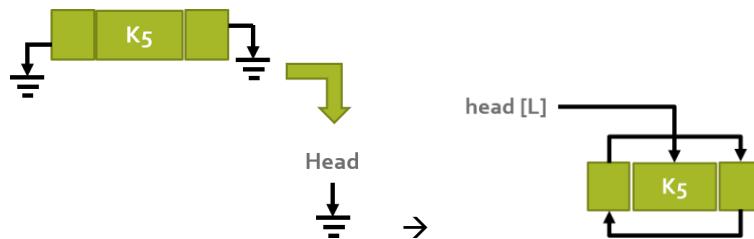
Código:	MADO-19
Versión:	01
Página	102/165
Sección ISO	8.3
Fecha de emisión	20 de enero de 2017

Facultad de Ingeniería

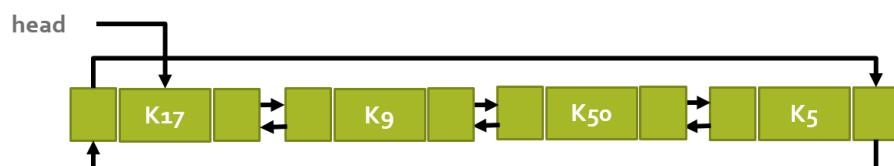
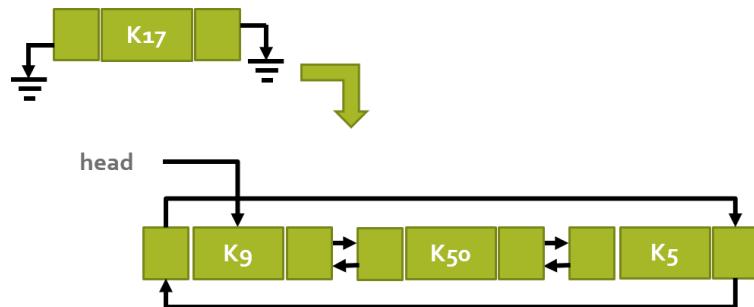
Área/Departamento:  
Laboratorio de computación salas A y B

La impresión de este documento es una copia no controlada

Es posible insertar elementos tanto en una lista doble circular vacía como en una lista doble circular con elementos. Cuando se inserta un nuevo elemento en una lista circular vacía la referencia al inicio de la lista (HEAD) apunta al nodo insertado y tanto la referencia al sucesor (NEXT) como al predecesor (PREV) del nodo apunta a sí mismo.



Cuando se inserta un nuevo elemento en una lista doble circular con elementos, el sucesor del nuevo nodo (NEXT) apunta al mismo nodo al que apunta el inicio de la lista (HEAD), la referencia al predecesor del nodo apunta al último elemento de la estructura (TAIL) y ahora HEAD apunta al nuevo nodo. Así mismo, el último nodo de la estructura (TAIL) apunta al primer elemento (nuevo nodo).





## Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I

Código:	MADO-19
Versión:	01
Página	103/165
Sección ISO	8.3
Fecha de emisión	20 de enero de 2017

Facultad de Ingeniería

Área/Departamento:  
Laboratorio de computación salas A y B

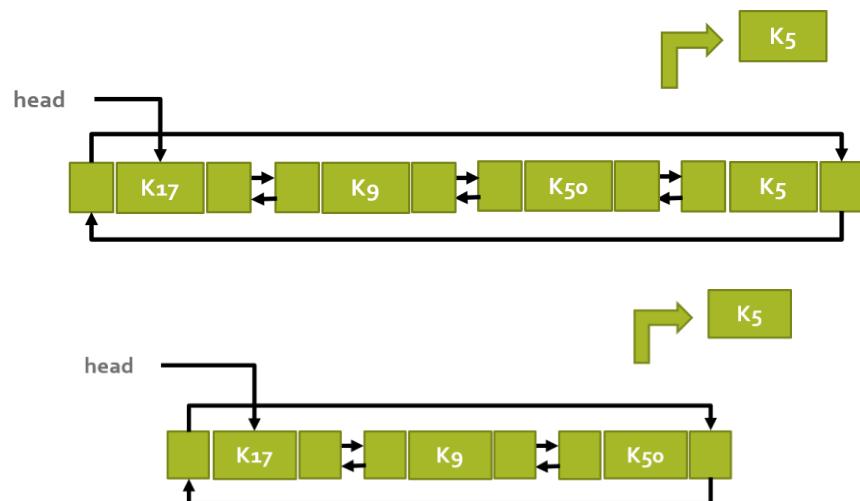
La impresión de este documento es una copia no controlada

Borrar

El método elimina el elemento  $x$  de la lista  $L$  (si es que éste se encuentra en la estructura). Para eliminar un elemento de la lista primero es necesario saber la ubicación del nodo a eliminar, por lo tanto, primero se debe realizar una búsqueda del elemento.

En una lista doble circular vacía no es posible eliminar, debido a que esta estructura no contiene elementos.

Para eliminar un nodo en una lista doble circular con elementos, primero se debe buscar el elemento a eliminar, una vez encontrado el nodo en la lista, se deben mover las referencias de la estructura de tal manera de que el antecesor del nodo a eliminar apunte al sucesor del mismo y viceversa.



	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código: MADO-19 Versión: 01 Página 104/165 Sección ISO 8.3 Fecha de emisión 20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B
La impresión de este documento es una copia no controlada		

## Aplicación

Una lista de videos en YouTube reproduce los elementos de manera lineal y secuencial, sin embargo, posee una referencia hacia el elemento siguiente (NEXT) y una referencia hacia el elemento anterior (PREV). Además, el primer elemento de la lista (HEAD) posee una referencia al siguiente elemento y una referencia al último elemento (TAIL) de la lista y viceversa. Por lo tanto, esta estructura es una lista doblemente ligada circular.

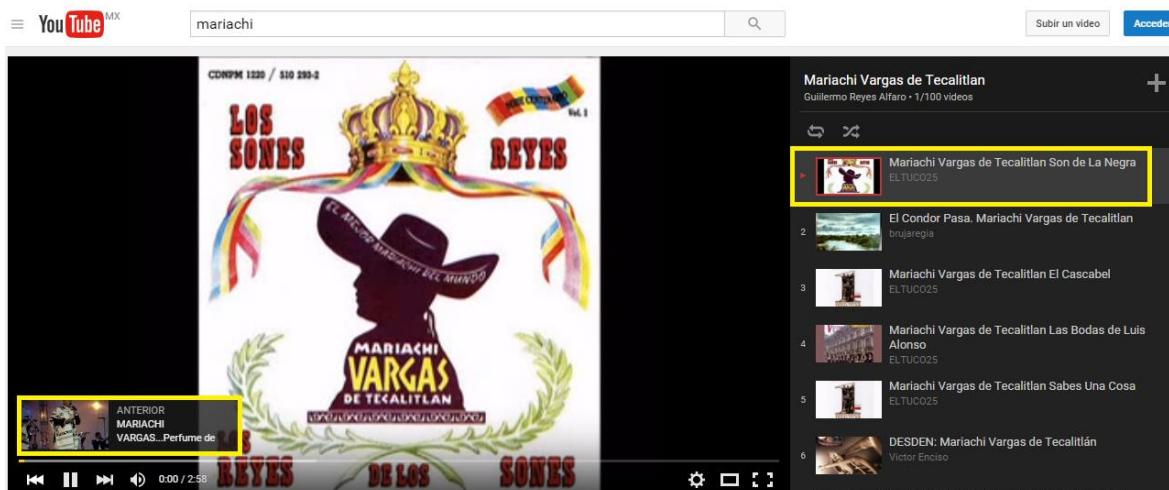
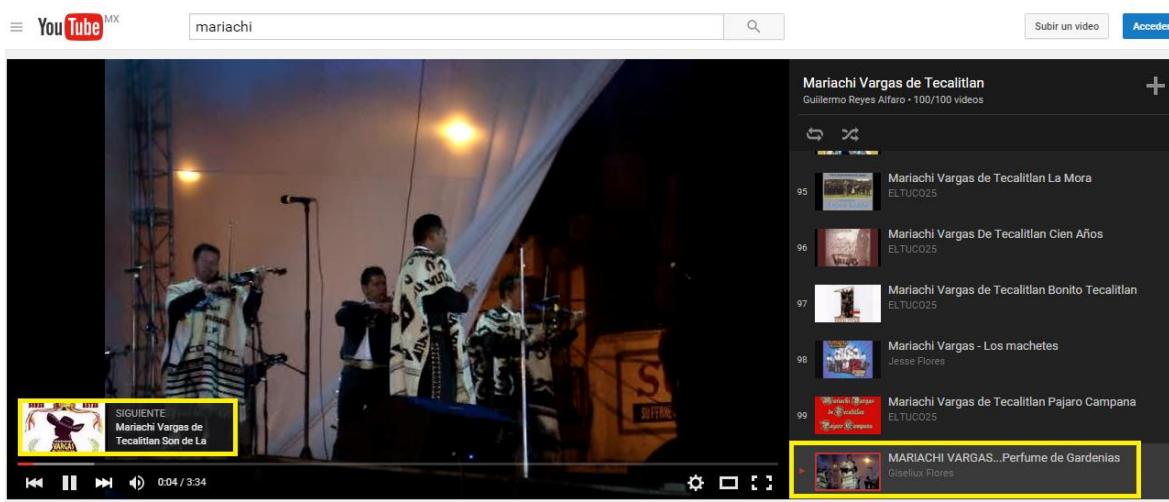


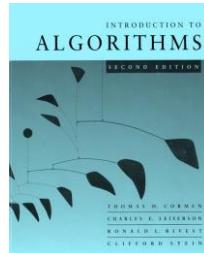
Figura 3. Primer elemento de la lista de videos.



	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código: Versión: Página: Sección ISO Fecha de emisión	MADO-19 01 105/165 8.3 20 de enero de 2017
Facultad de Ingeniería	Área/Departamento: Laboratorio de computación salas A y B		
La impresión de este documento es una copia no controlada			

**Figura 4.** Último elemento de la lista de videos.

## Bibliografía



Introduction to Algorithms. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, McGraw-Hill.



The Algorithm Design Manual. Steven S. Skiena, Springer.

Google (2016). Gmail [Figura 1 y figura 2]. Consulta: Enero de 2016. Disponible en: <https://www.mail.google.com>

Youtube (2016). Mariachi [Figura 3]. Consulta: Enero de 2016. Disponible en: <https://www.youtube.com/watch?v=qKEm19lMjuQ&list=PL84EC9ACDAF6B300C>

Youtube (2016). Mariachi [Figura 4]. Consulta: Enero de 2016. Disponible en: <https://www.youtube.com/watch?v=13dnkytPEZQ&index=100&list=PL84EC9ACDAF6B300C>

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código: MADO-19
		Versión: 01
		Página 106/165
		Sección ISO 8.3
		Fecha de emisión 20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B
La impresión de este documento es una copia no controlada		

## Guía práctica de estudio 09: Introducción a Python (I).

---



*Elaborado por:*

M.C. Edgar E. García Cano  
Ing. Jorge A. Solano Gálvez

*Autorizado por:*

M.C. Alejandro Velázquez Mena

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código:	MADO-19
		Versión:	01
		Página	107/165
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

## **Guía práctica de estudio 09: Introducción a Python (I).**

### **Objetivo:**

Aplicar las bases del lenguaje de programación Python en el ambiente de Jupyter notebook.

### **Actividades:**

- Insertar y ejecutar código en las celdas de la notebook
- Insertar texto en las celdas de la notebook
- Declarar variables
- Declarar cadenas
- Aplicar operadores
- Crear y manipular listas, tuplas y diccionarios
- Crear y ejecutar funciones

### **Repositorio de la guía:**

Jupyter Notebook GitHub:

[https://github.com/eegkno/FI\\_UNAM/blob/master/02\\_Estructuras\\_de\\_datos\\_y\\_algoritmos\\_1/P09/EDyA09\\_I.ipynb](https://github.com/eegkno/FI_UNAM/blob/master/02_Estructuras_de_datos_y_algoritmos_1/P09/EDyA09_I.ipynb)

Jupyter Notebook Visualizador:

[http://nbviewer.jupyter.org/github/eegkno/FI\\_UNAM/blob/master/02\\_Estructuras\\_de\\_datos\\_y\\_algoritmos\\_1/P09/EDyA09\\_I.ipynb](http://nbviewer.jupyter.org/github/eegkno/FI_UNAM/blob/master/02_Estructuras_de_datos_y_algoritmos_1/P09/EDyA09_I.ipynb)

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código:	MADO-19
		Versión:	01
		Página	108/165
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

## Variables y tipos

- Los nombres de las variables son alfanuméricos (a-z, A-Z, 0-9) y empiezan con una letra en minúscula.
- No se especifica el tipo de valor que una variable contiene, está implícito al momento de asignar un valor.
- No se necesita poner ; al final de cada instrucción.
- Mantener las indentaciones al momento de escribir código.

Nombres reservados en Python

and, as, assert, break, class, continue, def, del, elif, else, except, exec, finally, for, from, global, if, import, in, is, lambda, not, or, pass, print, raise, return, try, while, with, yield.

```
#Inicializando variables
x = 10      #variable de tipo entero
print(x)    #función para imprimir los valores de las variables

#Se puede utilizar comillas dobles o simples para crear una cadena
cadena = "Hola Mundo"      #variable de tipo cadena
print(cadena)
```

10  
Hola Mundo

```
#Asigna un mismo valor a tres variables
x = y = z = 10
print(x,y,z)
```

10 10 10

```
#La función type() permite conocer el tipo de una variable
type(x)
```

int

```
type(cadena)
str
```

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código:	MADO-19
		Versión:	01
		Página	109/165
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

```
#Se pueden cambiar los valores de las variables y el tipo se cambia automáticamente
x = "Hola Mundo"
cadena = 10
```

```
type(x)
```

```
str
```

```
type(cadena)
```

```
int
```

Cuando una variable tiene un valor constante, por convención, el nombre se escribe en mayúsculas.

```
SEGUNDOS_POR_DIA = 60 * 60 * 24
PI = 3.14
```

## Cadenas

Las cadenas pueden ser definidas usando comilla simple ('') o comilla doble (""). Una característica especial de las cadenas es que son inmutables, esto quiere decir que no se pueden cambiar los caracteres que contiene. El carácter \ sirve para escapar caracteres como \n o \t.

```
#Inicializando cadenas
cadena1 = 'Hola '
cadena2 = "Mundo"
print(cadena1)
print(cadena2)
concat_cadenas = cadena1 + cadena2 #Concatenación de cadenas
print(concat_cadenas)
```

```
Hola
Mundo
Hola Mundo
```

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código:	MADO-19
		Versión:	01
		Página	110/165
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

```
#Para concatenar un número y una cadena se debe usar la función str()
num_cadena = concat_cadenas +' '+ str(3) #Se agrega una cadena vacía para agregar un espacio
print(num_cadena)
```

Hola Mundo 3

Para concatenar cadenas se recomienda el uso de la función format(), en lugar del viejo estilo del operador '+!.

```
#El valor de La variable se va a imprimir en el Lugar donde se encuentre {} en La cadena
num_cadena = "{} {} {}".format(cadena1, cadena2, 3)
print(num_cadena)
```

Hola Mundo 3

Por medio de la función format, se puede cambiar el orden en que se imprimen las variables:

```
#Cuando se agrega un número dentro de {}, el valor la variable que se encuentra en esa posición
#dentro de la función format(), será impreso.
num_cadena = "Cambiando el orden: {1} {2} {0} #".format(cadena1, cadena2, 3)
print(num_cadena)
```

Cambiando el orden: Mundo 3 Hola #

Las funciones que están integradas en Python para trabajar con cadenas se pueden ver en el siguiente link <https://docs.python.org/3/library/stdtypes.html#string-methods>.

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código:	MADO-19
		Versión:	01
		Página	111/165
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

## Operadores

Aritméticos: +, -, \*, /

```
#Para el exponente se puede utilizar asterisco
print( 1 + 5 )
print( 6 * 3 )
print( 10 - 4 )
print( 100 / 50 )
print( 10 % 2 )
print( ((20 * 3) + (10 +1)) / 10 )
print( 2**2 )
```

```
6
18
6
2.0
0
7.1
4
```

Booleanos: and, not, or

```
False and True
```

Comparación: >, <, >=, <=, ==

```
print (7 < 5) #Falso
print (7 > 5) #Verdadero
print ((11 * 3)+2 == 36 - 1) #Verdadero
print ((11 * 3)+2 >= 36) #Falso
print ("curso" != "CuRsO") #Verdadero
```

Más información sobre tipos de datos y operadores se puede consultar en <https://docs.python.org/3/library/stdtypes.html>.

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código:	MADO-19
		Versión:	01
		Página	112/165
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

## Listas

- Básicamente son valores que están separados por comas dentro de paréntesis cuadrados.
- Está compuesta por cualquier cantidad y/o tipo de datos, ya sean cadenas, caracteres, números e inclusive otras listas.
- Se puede acceder a las listas por medio de índices, estos índices comienzan desde 0 hasta el número de elementos menos 1.
- Las listas son mutables.

```
#Declaracion de una lista simple
lista_diasDelMes=[31,28,31,30,31,30,31,31,30,31,30,31]

print (lista_diasDelMes)      #imprimir la lista completa
print (lista_diasDelMes[0])   #imprimir elemento 1
print (lista_diasDelMes[6])   #imprimir elemento 7
print (lista_diasDelMes[11])  #imprimir elemento 12

[31, 28, 31, 30, 31, 30, 31, 30, 31, 30, 31]
31
31
31
```

```
#Declaracion de listas anidadas

lista_numeros=[['cero', 0],['uno',1, 'UNO'], ['dos',2], ['tres', 3], ['cuatro',4], ['X',5]]

print (lista_numeros)      #imprimir lista completa

print (lista_numeros[0])   #imprime el elemento 0 de la lista
print (lista_numeros[1])   #imprime el elemento 1 de la lista

print (lista_numeros[2][0]) #imprime el primer elemento de la lista en la posicion 2
print (lista_numeros[2][1]) #imprime el segundo elemento de la lista en la posicion 2

print (lista_numeros[1][0])
print (lista_numeros[1][1])
print (lista_numeros[1][2])
```

```
[['cero', 0], ['uno', 1, 'UNO'], ['dos', 2], ['tres', 3], ['cuatro', 4], ['X', 5]]
['cero', 0]
['uno', 1, 'UNO']
dos
2
uno
1
UNO
```

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código:	MADO-19
		Versión:	01
		Página	113/165
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

```
# Se cambia el valor de uno de los elementos de la lista
lista_numeros[5][0] = "cinco"
print (lista_numeros[5])
['cinco', 5]
```

Hay otras operaciones que se pueden realizar usando listas, éstas se pueden consultar en <https://docs.python.org/3/tutorial/datastructures.html#more-on-lists>.

## Tuplas

- Son parecidas a las listas, valores separados por una coma.
- Comparadas con las listas, las tuplas no son mutables.
- Se pueden aplicar las mismas operaciones que en las listas y su ventaja es que consumen menos memoria para almacenarse.
- Se crean, ya sea utilizando paréntesis o simplemente separando los valores por comas.

```
# Declaracion de una tupla
tupla_diasDelMes=(31,28,31,30,31,30,31,31,30,31,30,31)

print (tupla_diasDelMes)      #imprimir la tupla completa
print (tupla_diasDelMes[0])   #imprimir elemento 1
print (tupla_diasDelMes[3])   #imprimir elemento 4
print (tupla_diasDelMes[1])   #imprimir elemento 2

(31, 28, 31, 30, 31, 30, 31, 31, 30, 31)
31
30
28
```

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código: MADO-19 Versión: 01 Página 114/165 Sección ISO 8.3 Fecha de emisión 20 de enero de 2017
Facultad de Ingeniería	Area/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada		

```
# Declaracion de tuplas anidadas

tupla_numeros=((('cero', 0), ('uno',1, 'UNO'), ('dos',2), ('tres', 3), ('cuatro',4), ('X',5))

print (tupla_numeros)          #imprimir tupla completa

print (tupla_numeros[0])       #imprime el elemento 0 de la tupla
print (tupla_numeros[1])       #imprime el elemento 1 de la tupla

print (tupla_numeros[2][0])    #imprime el primer elemento de la tupla en la posicion 2
print (tupla_numeros[2][1])    #imprime el segundo elemento de la tupla en la posicion 2

print (tupla_numeros[1][0])
print (tupla_numeros[1][1])
print (tupla_numeros[1][2])
```

```
(('cero', 0), ('uno', 1, 'UNO'), ('dos', 2), ('tres', 3), ('cuatro', 4), ('X', 5))
('cero', 0)
('uno', 1, 'UNO')
dos
2
uno
1
UNO
```

```
# Probando la mutabilidad de las listas vs la no mutabilidad de las tuplas
print("valor actual {}".format(lista_diasDelMes[0]))
lista_diasDelMes[0] = 50
print("valor cambiado {}".format(lista_diasDelMes[0]))
tupla_diasDelMes[0] = 50  #Esta asignación manda un error, ya que no se pueden cambiar los valores
```

```
valor actual 31
valor cambiado 50
```

```
-----
TypeError                                         Traceback (most recent call last)
<ipython-input-6-680222be056b> in <module>()
      3 lista_diasDelMes[0] = 50
      4 print("valor cambiado {}".format(lista_diasDelMes[0]))
----> 5 tupla_diasDelMes[0] = 50  #Esta asignación manda un error, ya que no se pueden cambiar los valores de las tuplas
```

```
TypeError: 'tuple' object does not support item assignment
```

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código:	MADO-19
		Versión:	01
		Página	115/165
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

## Tupla con nombre

En comparación con las tuplas anteriores, este tipo especial de tuplas permite especificar un nombre para describirla.

```
#Se debe importar la librería para hacer uso de namedtuple
from collections import namedtuple

#Se crea la tupla con nombre
#El primer argumento es el nombre de la tupla, mientras que el segundo argumento son los campos
#p es la referencia a la tupla
planeta = namedtuple('planeta', ['nombre', 'numero'])

#Se crea el planeta 1 y se agregan a la tupla los valores correspondientes a los campos
planetal = planeta('Mercurio', 1)
print(planetal)

#Se crea el planeta 2
planeta2 = planeta('Venus', 2)

#Se imprimen los valores de los campos
#Usando la referencia se llama a cada uno de sus campos
print(planetal.nombre, planetal.numero)
#Se obtienen los valores por el orden de los campos
print(planeta2[0], planeta2[1])

print('Campos de la tupla: {}'.format(planetal._fields))

planeta(nombre='Mercurio', numero=1)
Mercurio 1
Venus 2
Campos de la tupla: ('nombre', 'numero')
```

## Diccionarios

- Un diccionario se crea usando {} y consta de dos partes: llave y valor.
- Las llaves son inmutables, deben de tener un solo tipo de dato, una cadena o número. Una vez que es creado, no se puede cambiar su tipo.
- Mientras que el valor puede ser de cualquier tipo y se puede cambiar con el tiempo.
- Los elementos en un diccionario no están ordenados.

Más información sobre diccionarios en <https://docs.python.org/3/tutorial/datastructures.html#dictionaries>.

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código: MADO-19 Versión: 01 Página: 116/165 Sección ISO: 8.3 Fecha de emisión: 20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B
La impresión de este documento es una copia no controlada		

```

#Creando un diccionario
elementos = { 'hidrogeno': 1, 'helio': 2, 'carbon': 6 }

#EL momento de la impresion, pueden aparecer en diferente orden del introducido
print (elementos)

print (elementos['hidrogeno'])

{'hidrogeno': 1, 'helio': 2, 'carbon': 6}
1

#Se pueen agregar elementos al diccionario
elementos['litio'] = 3
elementos['nitrogeno'] = 8

print (elementos) #Imprimiendo todos los elementos, nótese que los elementos no están ordenados
{'hidrogeno': 1, 'helio': 2, 'carbon': 6, 'litio': 3, 'nitrogeno': 8}

#Creando un nuevo diccionario
elementos2 = {}
elementos2['H'] = {'name': 'Hydrogen', 'number': 1, 'weight': 1.00794}
elementos2['He'] = {'name': 'Helium', 'number': 2, 'weight': 4.002602}

print (elementos2)

{'H': {'name': 'Hydrogen', 'number': 1, 'weight': 1.00794}, 'He': {'name': 'Helium', 'number': 2, 'weight': 4.002602}}


#Imprimiendo los datos de un elemento del diccionario
print (elementos2['H'])
print (elementos2['H']['name'])
print (elementos2['H']['number'])
elementos2['H']['weight'] = 4.30 #Cambiando el valor de un elemento
print (elementos2['H']['weight'])

{'name': 'Hydrogen', 'number': 1, 'weight': 1.00794}
Hydrogen
1
4.3

#Agregando elementos a una llave
elementos2['H'].update({'gas noble':True})
print (elementos2['H'])

{'name': 'Hydrogen', 'number': 1, 'weight': 4.3, 'gas noble': True}

#Muestra todos los elementos del diccionario
print (elementos2.items())

#Muestra todas las llaves del diccionario
print (elementos2.keys())

dict_items([('H', {'name': 'Hydrogen', 'number': 1, 'weight': 4.3, 'gas noble': True}), ('He', {'name': 'Helium', 'number': 2, 'weight': 4.002602}))
```

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código:	MADO-19
		Versión:	01
		Página	117/165
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

## Funciones

- Una función o procedimiento sirve para empaquetar código que sirve para ser reutilizado.
- Se puede usar ese mismo código con diferentes entradas y obtener resultados o comportamiento de acuerdo con esos datos.

```
#Las funciones pueden recibir n número de parámetros, no se necesita indicar el tipo
def imprime_nombre(nombre):
    print("hola "+nombre) #Las cadenas se pueden concatenar con el +
```

```
#Llamada a la función
imprime_nombre("JJ")
```

hola JJ

```
#Definiendo una función que regresa el cuadrado de un número
def cuadrado(x):
    return x**2
```

```
x = 5
#La función format() sirve para convertir los parámetros que recibe, en cadenas; éstos valores se
#por las llaves de la cadena.
print("El cuadrado de {} es {}".format(x, cuadrado(x))) #La función cuadrado() regresa un valor
```

El cuadrado de 5 es 25

```
#Definiendo una función que regrese más de un valor
def varios(x):
    return x**2, x**3, x**4
```

```
#Los valores que regresa la función pueden ser guardado en variables separadas por ,
val1, val2, val3 = varios(2)
print("{} {} {}".format(val1, val2, val3))
```

4 8 16

```
#Función con un parámetro con un valor por defecto
def cuadrado_default(x=3):
    return x**2
```

```
#Como la función tiene un valor por default, si se manda llamar la función sin especificar el pa
#tiene por defecto
cuadrado_default()
```

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código:	MADO-19
		Versión:	01
		Página	118/165
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

**Idiom:** Cuando una función regresa más de un valor, se puede usar el operador '\_', para no guardar un valor no deseado.

```
#La función regresa tres, valores, pero sólo nos interesa el primero y el tercero
val4, _, val5 = varios(2)
print("{} {}".format(val4, val5))
```

4 16

## Variables globales

Se puede decir que el ambiente de ejecución es donde se efectúan las operaciones que componen un programa. Al momento de ejecutar un programa se crea un espacio de nombres para las variables. Hay dos tipos se espacio de nombres, el primero es el espacio global y el segundo el espacio local. Las variables que se declaran afuera de las funciones pertenecen al espacio global y no se necesita añadir un modificador para declararlas de esta manera. Por otro lado, todas las variables que se definen dentro de una función pertenecen al espacio local, estas variables sólo pueden ser reconocidas y usadas dentro de la propia función.

```
#Se crea una variable en el espacio global de nombres
vg = 'Global'
```

```
#Se crea una función que imprime la variable global
def funcion_v1():
    print(vg)
```

```
#Llamada a la función que imprime la variable global
funcion_v1()
```

```
#Imprime la variable global
print(vg)
```

Global  
Global

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código:	MADO-19
		Versión:	01
		Página	119/165
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

```
#Se crea una variable local que tiene el mismo nombre que la variable global
def funcion_v2():
    vg = "Local"
    print(vg)
```

```
#Llamada a la función
funcion_v2() #Imprime valor local

#Imprime la variable global
print(vg)
```

Local  
Global

```
#Se trata de imprimir el valor de la variable global, a diferencia de la función_v1(),
# se creó en el espacio local de la función_v3() una variable con el mismo nombre,
# por lo que se reemplaza la variable global
def funcion_v3():
    print(vg)
    vg = "Local"
    print(vg)
```

```
#Como se tiene una variable local y no se le ha asignado un valor, se genera un error
funcion_v3()
```

```
-----
UnboundLocalError                                     Traceback (most recent call last)
<ipython-input-31-375c7fdc87a4> in <module>()
      1 #Como se tiene una variable local y no se le ha asignado un valor, se genera un error
----> 2 funcion_v3()

<ipython-input-30-15325199aae4> in funcion_v3()
      3 #global
      4 def funcion_v3():
----> 5     print(vg)
      6     vg = "Local"
      7     print(vg)
```

UnboundLocalError: local variable 'vg' referenced before assignment

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código:	MADO-19
		Versión:	01
		Página	120/165
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

```
#Para resolver el problema anterior y especificar que se quiere hacer uso de la
# variable global dentro de la función funcion_v4(), se tiene que agregar la
# palabra reservada global
```

```
def funcion_v4():
    global vg
    print(vg)
    vg = "Local"
    print(vg)
```

```
#Al momento de ejecutar la función se imprime el valor que tenía asignado vg
# antes de ser modificado por la función. Después de asignar el valor, éste es impreso
funcion_v4()
```

```
#Se imprime la variable global con su valor modificado
print(vg)
```

```
Global
Local
Local
```

**NOTA:** El manejo de variables globales dentro de una función en el lenguaje Python se considera como una mala práctica, se recomienda que se pase como parámetro a la función y que se regrese un valor.

## Bibliografía

Tutorial oficial de Python: <https://docs.python.org/3/tutorial/>

Galería de notebooks: <https://wakari.io/gallery>

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código:	MADO-19
		Versión:	01
		Página	121/165
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

# Guía práctica de estudio 10: Introducción a Python (II).

---



*Elaborado por:*  
 M.C. Edgar E. García Cano  
 Ing. Jorge A. Solano Gálvez

*Autorizado por:*  
 M.C. Alejandro Velázquez Mena

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código:	MADO-19
		Versión:	01
		Página	122/165
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

## Guía práctica de estudio 10: Introducción a Python (II).

### Objetivo:

Aplicar las bases del lenguaje de programación Python en el ambiente de Jupyter notebook.

### Actividades:

- Aplicar estructuras de control selectivas
- Aplicar estructuras de control repetitivas
- Usar las bibliotecas estándar
- Generar una gráfica
- Ejecutar un programa desde la ventana de comandos
- Pedir datos al usuario al momento de ejecutar un programa

### Estructuras de control selectivas

if

La declaración IF sirve para ejecutar código dependiendo del resultado de una condición.

```
def obtenerMayor(param1,param2):
    if param1 < param2:
        print('{} es mayor que {}'.format(param2, param1))

obtenerMayor(5, 7)
7 es mayor que 5

obtenerMayor(7, 5) #No imprime nada
```

Se puede encadenar más de una condición sin tener que agregar un operador booleano.

```
x = y = z = 3
if x == y == z:
    print(True)

True
```

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código:	MADO-19
		Versión:	01
		Página	123/165
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

## if-else

Este tipo de declaraciones se usan para dar una opción en el caso de que la condición no se cumpla.

```
def obtenerMayorv2(param1,param2):
    if param1 < param2:
        return param2
    else:
        return param1

print ("El mayor es {}".format( obtenerMayorv2(4, 20) ))
El mayor es 20

print ("El mayor es {}".format( obtenerMayorv2(11, 6) ))
El mayor es 11
```

Para comparaciones simples, Python no tiene un operador ternario (`x ? True : False`), pero se puede emular con if-else:

```
def obtenerMayor_idiom(param1,param2):
    #La variable valor va a tener el valor de param2 si el if es verdadero
    #de lo contrario tendra el valor de param1
    valor = param2 if (param1 < param2) else param1
    return valor

print ("El mayor es {}".format( obtenerMayor_idiom(11, 6) ))
El mayor es 11
```

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código:	MADO-19
		Versión:	01
		Página	124/165
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

## if-elif-else

Este tipo de declaraciones sirve para generar varias casos de prueba. En otros lenguajes es similar a case o switch.

```
def numeros(num):
    if num==1:
        print ("tu numero es 1")
    elif num==2:
        print ("el numero es 2")
    elif num==3:
        print ("el numero es 3")
    elif num==4:
        print ("el numero es 4")
    else:
        print ("no hay opcion")
```

```
numeros(2)
```

```
el numero es 2
```

```
numeros(5)
```

```
no hay opcion
```

En algunos casos, se puede evitar la repetición de código del if-elif-else de la siguiente manera:

```
def numeros_idiom(num):
    #La tupla tiene las opciones válidas
    if num in (1,2,3,4):
        print("tu numero es {}".format(num))
    else:
        print ("{} no es una opcion".format(num))
```

```
numeros_idiom(2)
```

```
tu numero es 2
```

```
numeros_idiom(5)
```

```
5 no es una opcion
```

Estructura de control selectiva anidada

```
def obtenerMasGrande(a, b, c):
    if a > b:
        if a > c:
            return a
        else:
            return c
    else:
        if b > c:
            return b
        else:
            return c

print ("El mas grande es {}".format(obtenerMasGrande(7,13,1)))
```

```
El mas grande es 13
```

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código:	MADO-19
		Versión:	01
		Página	125/165
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

## Estructuras de control repetitivas

### Ciclo while

Un ciclo es la manera de ejecutar una o varias acciones repetidamente. A diferencia de las estructuras IF o IF-ELSE que sólo se ejecutan una vez. Para que el ciclo se ejecute, la condición siempre tiene que ser verdadera.

```
#Ejemplo 1
def cuenta(limite):
    i = limite
    while True:
        print(i)
        i = i -1
        if i == 0:
            break # Rompiendo el ciclo
```

```
cuenta(10)
10
9
8
7
6
5
4
3
2
1
```

```
#Ejemplo 2
def factorial(n):
    i = 2
    tmp = 1
    while i <n+1:
        tmp = tmp * i
        i = i + 1
    return tmp
```

```
print (factorial(4))
24
```

```
print (factorial(6))
720
```

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código:	MADO-19
		Versión:	01
		Página	126/165
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

## Ciclo for

Este ciclo es el más común usado en Python, se utiliza generalmente para hacer iteraciones en una lista, diccionarios y arreglos.

### Iteración en listas

```
for x in [1,2,3,4,5]:
    print(x)
```

```
1
2
3
4
5
```

```
#La función range() sirve para generar una lista
for x in range(5): #este caso es equivalente a range(0,5)
    print(x)
```

```
0
1
2
3
4
```

```
#También se puede inicializar desde números negativos
for x in range(-5,2):
    print(x)
```

```
-5
-4
-3
-2
-1
0
1
```

```
for num in ["uno", "dos", "tres", "cuatro"]:
    print(num)
```

```
uno
dos
tres
cuatro
```

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código:	MADO-19
		Versión:	01
		Página	127/165
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

## Iteración en diccionarios

```
#Creando un diccionario
elementos = { 'hidrogeno': 1, 'helio': 2, 'carbon': 6 }

for llave, valor in elementos.items():
    print(llave, " = ", valor)

helio = 2
carbon = 6
hidrogeno = 1
```

```
#Obteniendo sólo las llaves
for llave in elementos.keys():
    print(llave)

helio
carbon
hidrogeno
```

```
#Obteniendo sólo los valores
for valor in elementos.values():
    print(valor)
```

2  
6  
1

En algunos lenguajes de programación se crea un índice para iterar un conjunto de elementos (for (int i=0; i < elementos.size(); ++i)), sin embargo con Python se puede utilizar la función enumerate() en su lugar.

```
#Si se necesita iterar utilizando un índice
for idx, x in enumerate(elementos):
    print("El índice es: {} y el elemento: {}".format(idx, x))

El índice es: 0 y el elemento: hidrogeno
El índice es: 1 y el elemento: carbon
El índice es: 2 y el elemento: helio
```

Los ciclos for pueden hacer uso del else una vez que terminan de iterar, pero no funciona si se rompe el ciclo.

```
def cuenta_idiom(límite):
    for i in range(límite, 0, -1):
        print(i)
    else: #Corresponde al for, NO al IF
        print("Cuenta finalizada")
```

```
cuenta_idiom(5)

5
4
3
2
1
Cuenta finalizada
```

```
#Se rompe el ciclo y la sentencia else del for no se ejecuta
def cuenta_idiomv2(límite):
    for i in range(límite, 0, -1):
        print(i)
        if i == 3:
            break #Se rompe el ciclo
    else: #corresponde al FOR, NO al IF
        print("Cuenta finalizada")
```

```
cuenta_idiomv2(5)

5
4
3
```

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código:	MADO-19
		Versión:	01
		Página	128/165
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

## Bibliotecas

Todas las funcionalidades de Python son proporcionadas a través de bibliotecas que se encuentran en la colección de The Python Standard Library, la mayoría de estas bibliotecas son multi-plataforma.

Referencia del lenguaje: <https://docs.python.org/3/reference/index.html>

Bibliotecas estándar: <https://docs.python.org/3/library/>

```
#Para utilizar una biblioteca, ésta se debe de importar
import math

x = math.cos(math.pi)

print(x)
-1.0
```

```
#También se pueden importar todas las funciones de la biblioteca, de esta manera no se tiene que usar el prefijo
#de la biblioteca, que en el ejemplo anterior fue math
from math import *

x = cos(pi) #No se utiliza el prefijo math

print(x)
-1.0
```

```
#Otra manera es importar sólo las funciones que se necesitan
from math import cos, pi

x = cos(pi)

print(x)
-1.0
```

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código:	MADO-19
		Versión:	01
		Página	129/165
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

```
#Una vez que la biblioteca está importada, se pueden conocer las funciones que éste contiene
print(dir(math))

['__doc__', '__loader__', '__name__', '__package__', '__spec__', 'acos', 'acosh', 'asin', 'asinh', 'atan', 'atan2',
'atanh', 'ceil', 'copysign', 'cos', 'cosh', 'degrees', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs', 'factorial', 'floo
r', 'fmod', 'frexp', 'fsum', 'gamma', 'gcd', 'hypot', 'inf', 'isclose', 'isfinite', 'isnan', 'lgamma',
'log', 'log10', 'log1p', 'log2', 'modf', 'nan', 'pi', 'pow', 'radians', 'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'tr
unc']
```

```
#Para conocer cómo utilizar las funciones, se puede utilizar la función help
help(math.log)

Help on built-in function log in module math:

log(...)

    log(x[, base])

    Return the logarithm of x to the given base.
    If the base not specified, returns the natural logarithm (base e) of x.
```

```
#Se puede definir un alias para llamar a las funciones que tiene la biblioteca math.
#Esta es la forma más recomendada para importar módulos, ya que de esta manera se sabe de qué módulo proviene la función
import math as ma

x = ma.cos(ma.pi)

print(x)
-1.0
```

## Bibliotecas más usadas

NumPy (Numerical Python). Es una de las bibliotecas más populares de Python, es usado para realizar operaciones con vectores o matrices de una manera eficiente. Contiene funciones de Álgebra Lineal, transformadas de Fourier, generación de números aleatorios e integración con Fortran, C y C++.

Fuente: <http://www.numpy.org/>

SciPy (Scientific Python). Es una biblioteca hace uso de Numpy y es utilizada para hacer operaciones más avanzadas como transformadas discretas de Fourier, Álgebra Lineal, Optimización, etc.

Fuente: <http://www.scipy.org/>

Matplotlib. Esta biblioteca es usada para generar una variedad de gráficas en 2D y 3D, donde cada una de las configuraciones de la gráfica es programable. Se puede usar comando de Latex para agregar ecuaciones matemáticas a las gráficas.

Fuente: <http://matplotlib.org/>

Scikit Learn (Machine Learning). Ésta biblioteca está basada en los anteriores y contiene algoritmos de aprendizaje de máquina, reconocimiento de patrones y estadísticas para realizar clasificación, regresión, clustering, etc.

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código:	MADO-19
		Versión:	01
		Página	130/165
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Fuente: <http://scikit-learn.org/>

Pandas (Manipulación de datos). Esta biblioteca es utilizada para manipulación de datos, contiene estructuras de datos llamadas data frames que se asemejan a las hojas de cálculo y a los cuales se le puede aplicar una gran cantidad de funciones. Fuente: <http://pandas.pydata.org/>

ANEXO 1: En esta guía se explica de manera más detallada el uso de las bibliotecas Numpy y Matplotlib.

Jupyter Notebook GitHub:

[https://github.com/eegkno/FI\\_UNAM/blob/master/02\\_Estructuras\\_de\\_datos\\_y\\_algoritmos\\_1/Anexos/Anexo\\_I.ipynb](https://github.com/eegkno/FI_UNAM/blob/master/02_Estructuras_de_datos_y_algoritmos_1/Anexos/Anexo_I.ipynb)

Jupyter Notebook Visualizador:

[http://nbviewer.jupyter.org/github/eegkno/FI\\_UNAM/blob/master/02\\_Estructuras\\_de\\_datos\\_y\\_algoritmos\\_1/Anexos/Anexo\\_I.ipynb](http://nbviewer.jupyter.org/github/eegkno/FI_UNAM/blob/master/02_Estructuras_de_datos_y_algoritmos_1/Anexos/Anexo_I.ipynb)

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código: MADO-19 Versión: 01 Página 131/165 Sección ISO 8.3 Fecha de emisión 20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B
La impresión de este documento es una copia no controlada		

## Graficación

Matplotlib (<http://matplotlib.org/>) es una biblioteca usada para generar gráficas en 2D y 3D, donde cada una de las configuraciones de la gráfica es programable. En el siguiente ejemplo se mostrará la configuración básica de una gráfica.

EL API de matplotlib se encuentra en <http://matplotlib.org/api/index.html>

```
#Esta linea se ocupa para que las gráficas que se generen queden embebidas dentro de la página
%pylab inline
```

```
#Importando las bibliotecas
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
```

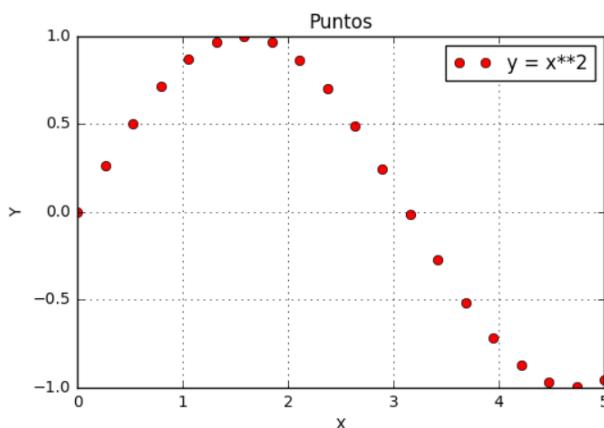
```
#Datos de entrada
x = linspace(0, 5, 20) #Generando 10 puntos entre 0 y 5
```

```
fig, ax = plt.subplots(facecolor='w', edgecolor='k')
ax.plot(x, sin(x), marker="o", color="r", linestyle='None')

ax.grid(True)
ax.set_xlabel('X') #Etiqueta del eje x
ax.set_ylabel('Y') #Etiqueta del eje y
ax.grid(True)
ax.legend(["y = x**2"])

plt.title('Puntos')
plt.show()

fig.savefig("gráfica.png") #Guardando la gráfica
```



	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código:	MADO-19
		Versión:	01
		Página	132/165
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

## Ejecución desde ventana de comandos

Todo el código que se ha visto hasta el momento puede ser guardado en archivos de texto plano con la extensión '.py'. Para ejecutarlo desde la ventana de comandos se escribe el comando:

```
python nombre_archivo.py
```

### Entrada de datos

Al igual que en otros lenguajes, también se puede pedir al usuario que introduzca ciertos datos de entrada cuando se ejecute un programa. Esto no se puede hacer desde la notebook, ya que los datos se introducen en las celdas que se van agregando a lo largo de la página, tal y como se ha venido manejando hasta ahora. Como ejemplo se va a ejecutar el archivo lectura\_datos.py desde una ventana de comandos.

```
python lectura_datos.py
```

Al momento de ejecutar el programa, se va a pedir al usuario que introduzca su nombre, esto se logra con el siguiente código:

```
#Se pide el nombre al usuario
print ("Hola, ¿cómo te llamas?")
#Se leen los datos introducidos por el usuario y se asignan a la variable nombre
nombre = input()
#Se escribe el nombre solicitado
print ("Buen día {}".format(nombre))
```

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código:	MADO-19
		Versión:	01
		Página	133/165
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Después de esto se despliega un menú donde se indican las operaciones que puede realizar el usuario, una vez que indicada la operación, se solicitan los datos necesarios para ejecutarla.

```
print ("---Calculadora---")      #Opciones para el usuario
print ("1- Sumar")
print ("2- Restar")
print ("3- Multiplicar")
print ("4- Dividir")
print ("5- Salir")
```

En la siguiente línea se solicita que el usuario especifique alguna de las operaciones, a diferencia de la primera petición, la función input() ahora tiene una cadena que se le despliega al usuario. A su vez, los datos que recibe la función input() son de tipo string, por lo que se tienen que transformar a entero con la función int() para poder realizar operaciones aritméticas.

```
op = int(input('Opcion: '))
```

## Bibliografía

Tutorial oficial de Python: <https://docs.python.org/3/tutorial/>

Galería de notebooks: <https://wakari.io/gallery>

Matplotlib: <http://matplotlib.org/>

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código: MADO-19
		Versión: 01
		Página 134/165
		Sección ISO 8.3
		Fecha de emisión 20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B
La impresión de este documento es una copia no controlada		

# Guía práctica de estudio 11: Estrategias para la construcción de algoritmos.

---



*Elaborado por:*

M.C. Edgar E. García Cano  
Ing. Jorge A. Solano Gálvez

*Autorizado por:*

M.C. Alejandro Velázquez Mena

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código:	MADO-19
		Versión:	01
		Página	135/165
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

## **Guía práctica de estudio 11: Estrategias para la construcción de algoritmos.**

### **Objetivo:**

El objetivo de esta guía es implementar, al menos, dos enfoques de diseño (estrategias) de algoritmos y analizar las implicaciones de cada uno de ellos.

### **Actividades:**

- Revisar el concepto y un ejemplo de diversas estrategias para construir algoritmos (fuerza bruta, algoritmo ávido, bottom-up, top-down, divide y vencerás, etc).

### **Conceptos a revisar en Python:**

- Escribir y leer en archivos.
- Graficar funciones usando la biblioteca Matplotlib.
- Generar listas de números aleatorios.
- Medir y graficar tiempos de ejecución.

### **Repositorio de la guía:**

Jupyter Notebook GitHub:

[https://github.com/eegkno/FI\\_UNAM/blob/master/02\\_Estructuras\\_de\\_datos\\_y\\_algoritmos\\_1/P11/EDyA11\\_I.ipynb](https://github.com/eegkno/FI_UNAM/blob/master/02_Estructuras_de_datos_y_algoritmos_1/P11/EDyA11_I.ipynb)

Jupyter Notebook Visualizador:

[http://nbviewer.jupyter.org/github/eegkno/FI\\_UNAM/blob/master/02\\_Estructuras\\_de\\_datos\\_y\\_algoritmos\\_1/P11/EDyA11\\_I.ipynb](http://nbviewer.jupyter.org/github/eegkno/FI_UNAM/blob/master/02_Estructuras_de_datos_y_algoritmos_1/P11/EDyA11_I.ipynb)

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código:	MADO-19
		Versión:	01
		Página	136/165
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

## Fuerza bruta

El objetivo de resolver problemas por medio de fuerza es bruta es hacer una búsqueda exhaustiva de todas las posibilidades que lleven a la solución del problema. Un ejemplo de esto es encontrar una contraseña haciendo una combinación exhaustiva de caracteres alfanuméricos generando cadenas de cierta longitud. La desventaja de resolver problemas por medio de esta estrategia es el tiempo que toman.

A continuación, se muestra una implementación de un buscador de contraseñas de entre 3 y 4 caracteres. Para este ejemplo se va a usar la biblioteca **string**, de ésta se van a importar los caracteres y dígitos.

También se usa la biblioteca **itertools** (<https://docs.python.org/3/library/itertools.html#>). La biblioteca **itertools** tiene una función llamada **product()** (<https://docs.python.org/3/library/itertools.html#itertools.product>) la cual se va a utilizar para realizar las combinaciones en cadenas de 3 y cuatro caracteres.

Las diferentes combinaciones generadas por el algoritmo se van a guardar en un archivo. Para guardar datos en un archivo se utiliza la función **open()**, que es para tener una referencia del archivo que se quiere abrir (<https://docs.python.org/3/tutorial/inputoutput.html#reading-and-writing-files>). Con la referencia creada se utiliza la función **write()**, que recibe la cadena que se va a escribir en el archivo. Finalmente, una vez que se termina la escritura hacia el archivo, éste se cierra con la función **close()**.

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código: MADO-19 Versión: 01 Página: 137/165 Sección ISO: 8.3 Fecha de emisión: 20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B
La impresión de este documento es una copia no controlada		

```

from string import ascii_letters , digits
from itertools import product

#Concatenar letas y dígitos en una sola cadena
caracteres = ascii_letters+digits

def buscador(con):

    #Archivo con todas las combinaciones generadas
    archivo = open("combinaciones.txt", "w")

    if 3<= len(con) <= 4:
        for i in range(3,5):
            for comb in product(caracteres, repeat = i):
                #Se utiliza join() para concatenar los caracteres regresado por la función product().
                #Como join necesita una cadena inicial para hacer la concatenación, se pone una cadena vacía
                #al principio
                prueba = "",join(comb)
                #Escribiendo al archivo cada combinación generada
                archivo.write( prueba + "\n" )
                if prueba == con:
                    print('Tu contraseña es {}'.format(prueba))
                    #Cerrando el archivo
                    archivo.close()
                    break
    else:
        print('Ingresa una contraseña que contenga de 3 a 4 caracteres')

from time import time
t0 = time()
con = 'H014'
buscador(con)
print("Tiempos de ejecución {}".format(round(time()-t0, 6)))

```

Tu contraseña es H014  
Tiempos de ejecución 15.26573

## Algoritmos ávidos (greedy)

Esta estrategia se diferencia de fuerza bruta porque va tomando una serie de decisiones en un orden específico, una vez que se ha ejecutado esa decisión, ya no se vuelve a considerar. En comparación con fuerza bruta, ésta puede ser más rápida; aunque una desventaja es que la solución que se obtiene no siempre es la más óptima.

**Tip:** En el siguiente ejemplo se va a realizar una división entre enteros, para esto se va a ocupar el operando `//`. La diferencia entre utilizar `/` y `//` es que el primer operador realiza una operación de números reales y el segundo una operación de números enteros.

$$5/2 = 2.5$$

$$5//2 = 2$$

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código: MADO-19 Versión: 01 Página: 138/165 Sección ISO: 8.3 Fecha de emisión: 20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B
La impresión de este documento es una copia no controlada		

A continuación, se muestra la implementación del problema de cambio de monedas. El problema consiste en regresar el cambio de monedas, de cierta denominación, usando el menor número de éstas. Este problema se resuelve escogiendo sucesivamente las monedas de mayor valor hasta que ya no se pueda seguir usándolas y cuando esto pasa, se utiliza la siguiente de mayor valor. La desventaja en esta solución es que, si no se da la denominación de monedas en orden de mayor a menor, se resuelve el problema, pero no de una manera óptima.

```

def cambio(cantidad, denominaciones):
    resultado = []
    while (cantidad > 0):
        if (cantidad >= denominaciones[0]):
            num = cantidad // denominaciones[0]
            cantidad = cantidad - (num * denominaciones[0])
            resultado.append([denominaciones[0], num])
            denominaciones = denominaciones[1:] #Se va consumiendo la lista de denominaciones
    return resultado

#Pruebas del algoritmo
print (cambio(1000, [500, 200, 100, 50, 20, 5, 1]))
print (cambio(500, [500, 200, 100, 50, 20, 5, 1]))
print (cambio(300, [50, 20, 5, 1]))
print (cambio(200, [5]))
print (cambio(98, [50, 20, 5, 1]))

[[500, 2]]
[[500, 1]]
[[50, 6]]
[[5, 40]]
[[50, 1], [20, 2], [5, 1], [1, 3]]

```

En el siguiente ejemplo no regresa la solución óptima, además, si no existiera la moneda de valor 1, la solución fallaría.

```

print (cambio(98, [5, 20, 1, 50]))
[[5, 19], [1, 3]]

```

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código:	MADO-19
		Versión:	01
		Página	139/165
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

## Bottom-up (programación dinámica)

El objetivo de esta estrategia es resolver un problema a partir de subproblemas que ya han sido resueltos. La solución final se forma a partir de la combinación de una o más soluciones que se guardan en una tabla, ésta previene que se vuelvan a calcular las soluciones.

Como ejemplo, se va a calcular el número n de la sucesión de Fibonacci. La sucesión de Fibonacci es una sucesión infinita de números enteros cuyos primeros dos elementos son 0 y 1, los siguientes números son calculados por la suma de los dos anteriores.

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ...

A continuación, se presenta la implementación iterativa para calcular la sucesión de Fibonacci.

```
def fibonacci_iterativo_v1(numero):
    f1=0
    f2=1
    tmp=0
    for i in range(1,numero-1):
        tmp = f1+f2
        f1=f2
        f2=tmp
    return f2
```

```
fibonacci_iterativo_v1(6)
```

5

**Tip:** En Python se puede hacer una *asignación paralela*, esto va a servir para evitar tener la variable auxiliar tmp, tal y como se muestra a continuación.

```
def fibonacci_iterativo_v2(numero):
    f1=0
    f2=1
    for i in range(1, numero-1):
        f1,f2=f2,f1+f2      #Asignación paralela
    return f2
```

```
fibonacci_iterativo_v2(6)
```

5

Una vez que se conoce como calcular la sucesión de Fibonacci, ahora vamos a aplicar la estrategia bottom-up. Partimos del hecho de que ya tenemos las soluciones para:

$$f(0) = 0$$

$$f(1) = 1$$

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código:	MADO-19
		Versión:	01
		Página	140/165
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

$$f(2) = 1$$

Estas soluciones previas son almacenadas en la tabla de soluciones f\_parciales.

$$f_{\text{parciales}} = [0, 1, 1]$$

```
def fibonacci_bottom_up(numero):
    f_parciales = [0, 1, 1] #Esta es la lista que mantiene las soluciones previamente calculadas
    while len(f_parciales) < numero:
        f_parciales.append(f_parciales[-1] + f_parciales[-2])
        print(f_parciales)
    return f_parciales[numero-1]

fibonacci_bottom_up(5)
[0, 1, 1, 2]
[0, 1, 1, 2, 3]
3
```

Como se observa en el resultado anterior, no se hace el cálculo de los primeros números, si no que se toman las soluciones ya existentes. La solución se encuentra calculando los resultados desde los primeros números (casos base), hasta llegar a n, de abajo hacia arriba.

[0, 1, 1] Datos iniciales

[0, 1, 1, 2] Primera iteración, se calcula  $n-1 = 1$ , y  $n - 2 = 1$ ;

[0, 1, 1, 2, 3] Segunda iteración, se calcula  $n-1 = 2$ , y  $n - 2 = 1$ ;

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código:	MADO-19
		Versión:	01
		Página	141/165
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

## Top-down

A diferencia de bottom-up, aquí se empiezan a hacer los cálculos de n hacia abajo. Además, se aplica una técnica llamada memorización la cual consiste en guardar los resultados previamente calculados, de tal manera que no se tengan que repetir operaciones.

Para aplicar la estrategia **top-down**, se utiliza un diccionario (memoria) el cual va a almacenar valores previamente calculados. Una vez que se realice el cálculo de algún elemento de la sucesión de Fibonacci, éste se va a almacenar ahí.

```
#Memoria inicial
memoria = {1:0, 2:1, 3:1}

def fibonacci_top_down(numero):
    if numero in memoria: #Si el número ya se encuentra calculado, se regresa el valor ya ya no se hacen más cálculos
        return memoria[numero]
    f = fibonacci_iterativo_v2(numero-1) + fibonacci_iterativo_v2(numero-2)
    memoria[numero] = f
    return memoria[numero]
```

Como se muestra en el código anterior, para obtener n, se calculan n-1 y n-2 usando la versión iterativa. La deficiencia de este algoritmo es que hay cálculos que se están repitiendo. La ventaja, es que una vez que ya se calcularon, se guardan en una memoria, que en este caso es un diccionario; en dado caso de que se necesite un valor que ya ha sido calculado, sólo regresa y ya no se realizan los cálculos.

```
fibonacci_top_down(12)
89

#Memoria después de obtener el elemento 12 de la sucesión de Fibonacci
memoria
{1: 0, 2: 1, 3: 1, 12: 89}

#Memoria después de obtener el elemento 8 de la sucesión de Fibonacci
fibonacci_top_down(8)
13

memoria
{1: 0, 2: 1, 3: 1, 8: 13, 12: 89}
```

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código:	MADO-19
		Versión:	01
		Página	142/165
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Como se muestra en la impresión de la variable memoria, que contiene los resultados previamente calculados, los nuevos valores obtenidos se agregaron a ésta. El problema con esta versión es que se siguen haciendo cálculos de más, ya que la función fibonacci\_iterativo\_v2() no tiene acceso a la variable memoria, lo que implica que tenemos que hacer modificaciones a la implementación. Por ejemplo, si se quiere calcular el elemento 5, se tiene que calcular (n-2) y (n-1), aunque algunos valores ya existen en la variable memoria no hay una manera de acceder a ellos.

$$\begin{aligned} f(5) &= \\ (n-1) &= f(4)+f(3)+f(2)+f(1) \\ (n-2) &= f(3)+f(2)+f(1) \end{aligned}$$

Ahora, se requiere que los valores ya calculados sean guardados en un archivo, de tal manera que se puedan utilizar en otro instante de tiempo. Para esto se va a hacer uso de la biblioteca pickle (<https://docs.python.org/3.5/library/pickle.html>). Los archivos que se generan con pickle están en binario, por lo que no se puede leer a simple vista la información que contienen, como se haría desde un archivo de texto plano.

```
#Se carga la biblioteca
import pickle

#Guardar variable
#No hay restricción en lo que se pone como extensión del archivo,
#generalmente se usa .p o .pickle como estandar.
archivo = open('memoria.p', 'wb')      #Se abre el archivo para escribir en modo binario
pickle.dump(memoria, archivo)          #Se guarda la variable memoria que es un diccionario
archivo.close()                         #Se cierra el archivo
```

```
#Leer variable
archivo = open('memoria.p', 'rb')        #Se abre el archivo para leer en modo binario
memoria_de_archivo = pickle.load(archivo) #Se lee la variable
archivo.close()                          #Se cierra el archivo
```

Si no se realizó un cambio en memoria, ésta variable y memoria\_de\_archivo deben contener los mismos datos.

```
memoria
{1: 0, 2: 1, 3: 1, 8: 13, 12: 89}
```

```
memoria_de_archivo
{1: 0, 2: 1, 3: 1, 8: 13, 12: 89}
```

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código:	MADO-19
		Versión:	01
		Página	143/165
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

## Incremental

Es una estrategia que consiste en implementar y probar que sea correcto de manera paulatina, ya que en cada iteración se va agregando información hasta completar la tarea.

### *Insertion sort*

Insertion sort ordena los elementos manteniendo una sublistas de números ordenados empezando por las primeras localidades de la lista. Al principio se considera que el elemento en la primera posición de la lista está ordenado. Después cada uno de los elementos de la lista se compara con la sublistas ordenada para encontrar la posición adecuada. La Figura 1 muestra la secuencia de cómo se ordena un elemento de la lista.

Tip: Las imágenes pueden ser agregadas usando etiquetas de HTML.

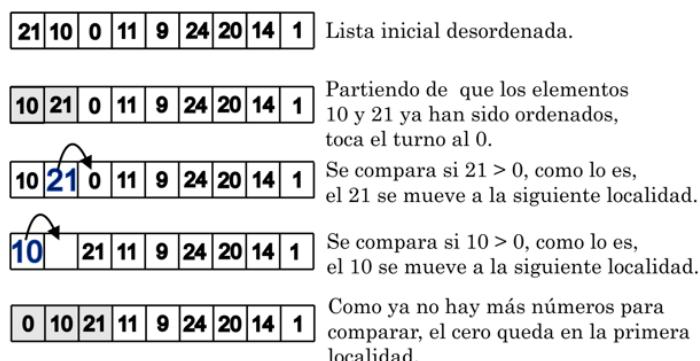


Figura 1: Pasos para ordenar el número 0.

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código:	MADO-19
		Versión:	01
		Página	144/165
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

```

def insertionSort(n_lista):
    for index in range(1,len(n_lista)):
        actual = n_lista[index]
        posicion = index
        print("valor a ordenar = {}".format(actual))
        while posicion>0 and n_lista[posicion-1]>actual:
            n_lista[posicion]=n_lista[posicion-1]
            posicion = posicion-1
        n_lista[posicion]=actual
        print(n_lista)
        print()
    return n_lista

```

```

#Datos de entrada
lista = [21, 10, 0, 11, 9, 24, 20, 14, 1]
print("lista desordenada {}".format(lista))
insertionSort(lista)
print("lista ordenada {}".format(lista))

lista desordenada [21, 10, 0, 11, 9, 24, 20, 14, 1]
valor a ordenar = 10
[10, 21, 0, 11, 9, 24, 20, 14, 1]

valor a ordenar = 0
[0, 10, 21, 11, 9, 24, 20, 14, 1]

valor a ordenar = 11
[0, 10, 11, 21, 9, 24, 20, 14, 1]

valor a ordenar = 9
[0, 9, 10, 11, 21, 24, 20, 14, 1]

valor a ordenar = 24
[0, 9, 10, 11, 21, 24, 20, 14, 1]

valor a ordenar = 20
[0, 9, 10, 11, 20, 21, 24, 14, 1]

valor a ordenar = 14
[0, 9, 10, 11, 14, 20, 21, 24, 1]

valor a ordenar = 1
[0, 1, 9, 10, 11, 14, 20, 21, 24]

lista ordenada [0, 1, 9, 10, 11, 14, 20, 21, 24]

```

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código:	MADO-19
		Versión:	01
		Página	145/165
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

## Divide y vencerás

Es una estrategia que consiste en:

- Dividir el problema en subproblemas hasta que son suficientemente simples que se pueden resolver directamente.
- Despues las soluciones son combinadas para generar la solución general del problema.

### *Quick sort*

Quicksort es un ejemplo de resolver un problema por medio de la estrategia divide y vencerás. En Quicksort se divide en dos el arreglo que va a ser ordenado y se llama recursivamente para ordenar las divisiones. La parte más importante en Quicksort es la partición de los datos. Lo primero que se necesita es escoger un valor de pivote el cual está encargado de ayudar con la partición de los datos. El objetivo de dividir los datos es mover los que se encuentran en una posición incorrecta con respecto al pivote. La siguiente figura muestra un ejemplo de cómo se ordena una lista.



## Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I

Código:	MADO-19
Versión:	01
Página	146/165
Sección ISO	8.3
Fecha de emisión	20 de enero de 2017

Facultad de Ingeniería

Área/Departamento:  
Laboratorio de computación salas A y B

La impresión de este documento es una copia no controlada

21	10	0	11	9	24	20	14	1
----	----	---	----	---	----	----	----	---



Lista inicial desordenada

21	10	0	11	9	24	20	14	1
----	----	---	----	---	----	----	----	---

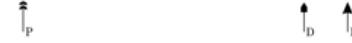


I > pivot

D < pivot

1 se cambiará de lugar de 24 y viceversa

21	10	0	11	9	1	20	14	24
----	----	---	----	---	---	----	----	----



I > pivot

D < pivot

14 se cambiará al lugar de 21 y viceversa

14	10	0	11	9	1	20	21	24
----	----	---	----	---	---	----	----	----



D > I

1 se cambiará por el 14 y se convierte en el nuevo pivote.

1	10	0	11	9	14	20	21	24
---	----	---	----	---	----	----	----	----



Se crean dos particiones y 14 es el nuevo pivote.

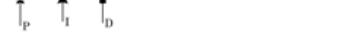
I > pivot

D < pivot

1 se cambiará por el 14 y se convierte en el nuevo pivote.

Se tiene una nueva partición de los datos

1	10	0	11	9	14	20	21	24
---	----	---	----	---	----	----	----	----



I > pivot

D < pivot

0 se cambiará al lugar de 10 y viceversa

1	0	10	11	9	14	20	21	24
---	---	----	----	---	----	----	----	----

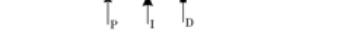


I < pivot

D > pivot

1 se cambiará al lugar de 0 y viceversa

0	1	10	11	9	14	20	21	24
---	---	----	----	---	----	----	----	----

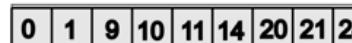


I > pivot

D < pivot

11 se cambiará al lugar de 9 y viceversa

0	1	10	9	11	14	20	21	24
---	---	----	---	----	----	----	----	----



10 se cambiará al lugar de 9 y viceversa

Lista ordenada

Figura 2: Pasos para ordenar una lista con quicksort.

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código: MADO-19 Versión: 01 Página 147/165 Sección ISO 8.3 Fecha de emisión 20 de enero de 2017
Facultad de Ingeniería	Area/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada		

```

def quicksort(lista):
    quicksort_aux(lista,0,len(lista)-1)

def quicksort_aux(lista,inicio, fin):
    if inicio < fin:
        pivot = particion(lista,inicio,fin)

        quicksort_aux(lista, inicio, pivot-1)
        quicksort_aux(lista, pivot+1, fin)

def particion(lista, inicio, fin):
    #Se asigna como pivote en número de la primera localidad
    pivot = lista[inicio]
    print("Valor del pivote {}".format(pivot))
    #Se crean dos marcadores
    izquierda = inicio+1
    derecha = fin
    print("índice izquierdo {}".format(izquierda))
    print("índice derecho {}".format(derecha))

    bandera = False
    while not bandera:
        while izquierda <= derecha and lista[izquierda] <= pivot:
            izquierda = izquierda + 1
        while lista[derecha] >= pivot and derecha >=izquierda:
            derecha = derecha -1
        if derecha < izquierda:
            bandera= True
        else:
            temp=lista[izquierda]
            lista[izquierda]=lista[derecha]
            lista[derecha]=temp

    print(lista)

    temp=lista[inicio]
    lista[inicio]=lista[derecha]
    lista[derecha]=temp
    return derecha

```

```

lista = [21, 10, 0, 11, 9, 24, 20, 14, 1]
print("lista desordenada {}".format(lista))
quicksort(lista)
print("lista ordenada {}".format(lista))

lista desordenada [21, 10, 0, 11, 9, 24, 20, 14, 1]
Valor del pivote 21
índice izquierdo 1
índice derecho 8
[21, 10, 0, 11, 9, 1, 20, 14, 24]
Valor del pivote 14
índice izquierdo 1
índice derecho 6
[14, 10, 0, 11, 9, 1, 20, 21, 24]
Valor del pivote 1
índice izquierdo 1
índice derecho 4
[1, 0, 10, 11, 9, 14, 20, 21, 24]
Valor del pivote 10
índice izquierdo 3
índice derecho 4
[0, 1, 10, 9, 11, 14, 20, 21, 24]
lista ordenada [0, 1, 9, 10, 11, 14, 20, 21, 24]

```

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código: MADO-19 Versión: 01 Página 148/165 Sección ISO 8.3 Fecha de emisión 20 de enero de 2017
Facultad de Ingeniería	Area/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada		

## Medición y gráficas de los tiempos de ejecución

```
#Importando bibliotecas
%pylab inline
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

Populating the interactive namespace from numpy and matplotlib
```

**Tip:** Las funciones en Python pueden ser guardadas en archivos individuales (**insertionSort.py**) o varias en un sólo archivo (**quickSort.py**). En el siguiente ejemplo, se agrego **\_time** al nombre de la función en los archivos.

**Tip:** En dado caso de que se quiera llamar más funciones que estén en un mismo archivo se pueden escribir los nombres de las funciones separados por nombres: \*from file\_name import función1, función2, función3\*

```
#Cargando módulos
import random
from time import time

#Cargando las funciones guardadas en los archivo
from insertionSort import insertionSort_time
#Sólo se necesita llamar a la función principal
from quickSort import quicksort_time

#Tamaños de la lista de números aleatorios a generar
datos = [ii*100 for ii in range(1,21)]

tiempo_is = [] #Lista para guardar el tiempo de ejecución de insert sort
tiempo_qs = [] #Lista para guardar el tiempo de ejecución de quick sort

for ii in datos:
    lista_is = random.sample(range(0, 10000000), ii)
    #Se hace una copia de la lista para que se ejecute el algoritmo con los mismo números
    lista_qs = lista_is.copy()

    t0 = time() #Se guarda el tiempo inicial
    insertionSort_time(lista_is)
    tiempo_is.append(round(time()-t0, 6)) #Se le resta al tiempo actual, el tiempo inicial

    t0 = time()
    quicksort_time(lista_qs)
    tiempo_qs.append(round(time()-t0, 6))
```

**NOTA:** La función `time()` regresa el tiempo en segundos (<https://docs.python.org/3/library/time.html#time.time>).

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código: MADO-19 Versión: 01 Página 149/165 Sección ISO 8.3 Fecha de emisión 20 de enero de 2017
Facultad de Ingeniería	Area/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada		

```
#Imprimiendo tiempos parciales de ejecución
print("Tiempos parciales de ejecución en INSERT SORT {} [s] \n".format(tiempo_is))
print("Tiempos parciales de ejecución en QUICK SORT {} [s]".format(tiempo_qs))
```

```
Tiempos parciales de ejecución en INSERT SORT [0.0, 0.002005, 0.005013, 0.012001, 0.014037, 0.032504, 0.026106, 0.035
0.0, 0.044623, 0.055145, 0.078246, 0.092553, 0.100029, 0.116888, 0.124838, 0.16404, 0.170012, 0.197678, 0.234017, 0.22
473] [s]
```

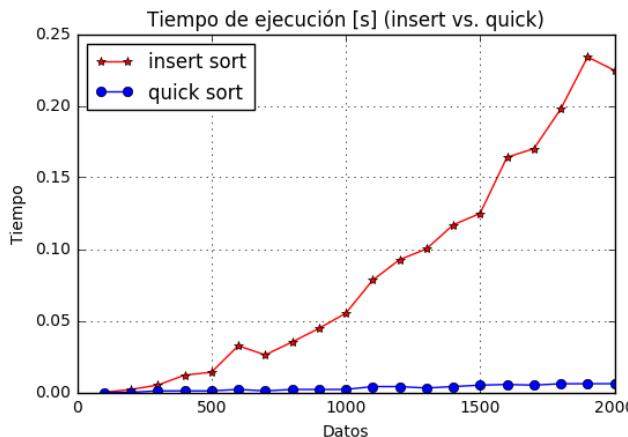
```
Tiempos parciales de ejecución en QUICK SORT [0.0, 0.0, 0.001004, 0.001004, 0.001003, 0.002035, 0.000996, 0.002005,
0.002006, 0.004008, 0.004011, 0.003008, 0.004006, 0.005014, 0.005479, 0.005006, 0.005985, 0.006016, 0.00601
6] [s]
```

```
#Imprimiendo tiempos totales de ejecución
#Para calcular el tiempo total se aplica la función sum() a las listas de tiempo
print("Tiempo total de ejecución en insert sort {} [s]".format(sum(tiempo_is)))
print("Tiempo total de ejecución en quick sort {} [s]".format(sum(tiempo_qs)))
```

```
Tiempo total de ejecución en insert sort 1.729555 [s]
Tiempo total de ejecución en quick sort 0.06060699999999999 [s]
```

```
#Generando la gráfica
fig, ax = subplots()
ax.plot(datos, tiempo_is, label="insert sort", marker="*", color="r")
ax.plot(datos, tiempo_qs, label="quick sort", marker="o", color="b")
ax.set_xlabel('Datos')
ax.set_ylabel('Tiempo')
ax.grid(True)
ax.legend(loc=2);

plt.title('Tiempo de ejecución [s] (insert vs. quick)')
plt.show()
```



	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código:	MADO-19
		Versión:	01
		Página	150/165
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

## Modelo RAM

Cuando se realiza un análisis de complejidad utilizando el modelo RAM, se debe contabilizar las veces que se ejecuta una función o un ciclo, en lugar de medir el tiempo de ejecución.

```

import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

times = 0

def insertionSort_graph(n_lista):
    global times
    for index in range(1,len(n_lista)):
        times += 1
        actual = n_lista[index]
        posicion = index
        while posicion>0 and n_lista[posicion-1]>actual:
            times += 1
            n_lista[posicion]=n_lista[posicion-1]
            posicion = posicion-1
        n_lista[posicion]=actual
    return n_lista

```

```

TAM = 101
eje_x = list(range(1,TAM,1))
eje_y = []
lista_variable = []

for num in eje_x:
    lista_variable = random.sample(range(0, 1000), num)
    times = 0
    lista_variable = insertionSort_graph(lista_variable)
    eje_y.append(times)

```



## Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I

Código:	MADO-19
Versión:	01
Página	151/165
Sección ISO	8.3
Fecha de emisión	20 de enero de 2017

Facultad de Ingeniería

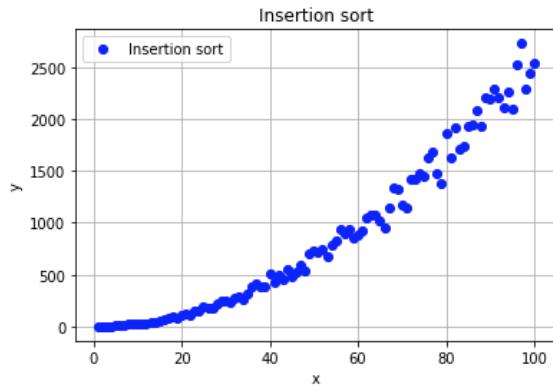
Área/Departamento:  
Laboratorio de computación salas A y B

La impresión de este documento es una copia no controlada

```
fig, ax = plt.subplots(facecolor='w', edgecolor='k')
ax.plot(eje_x, eje_y, marker="o", color="b", linestyle='None')

ax.set_xlabel('x')
ax.set_ylabel('y')
ax.grid(True)
ax.legend(["Insertion sort"])

plt.title('Insertion sort')
plt.show()
```



	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código: Versión: Página: Sección ISO Fecha de emisión	MADO-19 01 152/165 8.3 20 de enero de 2017
Facultad de Ingeniería	Área/Departamento: Laboratorio de computación salas A y B		
La impresión de este documento es una copia no controlada			

## Bibliografía

Design and analysis of algorithms; Prabhakar Gupta y Manish Varshney; PHI Learning, 2012, segunda edición.

Introduction to Algorithms, Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest y Clifford Stein; The MIT Press; 2009, tercera edición.

Problem Solving with Algorithms and Data Structures using Python; Bradley N. Miller y David L. Ranum, Franklin, Beedle & Associates; 2011, segunda edición.

<https://docs.python.org/3/library/itertools.html#>

<https://docs.python.org/3/library/itertools.html#itertools.product>

<https://docs.python.org/3/tutorial/inputoutput.html#reading-and-writing-files>

<https://docs.python.org/3.5/library/pickle.html>

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código: MADO-19
		Versión: 01
		Página 153/165
		Sección ISO 8.3
		Fecha de emisión 20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B
La impresión de este documento es una copia no controlada		

## Guía práctica de estudio 12: Recursividad.

---



*Elaborado por:*  
 M.C. Edgar E. García Cano  
 Ing. Jorge A. Solano Gálvez

*Autorizado por:*  
 M.C. Alejandro Velázquez Mena

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código:	MADO-19
		Versión:	01
		Página	154/165
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

## **Guía práctica de estudio 12: Recursividad.**

### **Objetivo:**

El objetivo de esta guía es aplicar el concepto de recursividad para la solución de problemas.

### **Actividades:**

- Revisar el concepto y las reglas de la recursividad y sus implicaciones.
- Ejecutar programas guardados en archivos desde la notebook.

### **Repositorio de la guía:**

Jupyter Notebook GitHub:

[https://github.com/eegkno/FI\\_UNAM/blob/master/02\\_Estructuras\\_de\\_datos\\_y\\_algoritmos\\_1/P13/EDyA12.ipynb](https://github.com/eegkno/FI_UNAM/blob/master/02_Estructuras_de_datos_y_algoritmos_1/P13/EDyA12.ipynb)

Jupyter Notebook Visualizador:

[http://nbviewer.jupyter.org/github/eegkno/FI\\_UNAM/blob/master/02\\_Estructuras\\_de\\_datos\\_y\\_a\\_lgoritmos\\_1/P12/EDyA12\\_II.ipynb](http://nbviewer.jupyter.org/github/eegkno/FI_UNAM/blob/master/02_Estructuras_de_datos_y_a_lgoritmos_1/P12/EDyA12_II.ipynb)

Es sumamente importante respetar las indentaciones al momento de escribir código en Python. Se recomienda usar 4 espacios por nivel de indentación, los espacios son preferidos sobre el uso de tabuladores (<https://www.python.org/dev/peps/pep-0008/#code-lay-out>).

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código:	MADO-19
		Versión:	01
		Página	155/165
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

## Recursividad

El propósito de la recursividad es dividir un problema en problemas más pequeños, de tal manera que la solución del problema se vuelva trivial. Básicamente, la recursión se puede explicar como una función que se llama así misma.

Para aplicar recursión se deben cumplir tres reglas:

- Debe de haber uno o más casos base.
- La expansión debe terminar en un caso base.
- La función se debe llamar a sí misma.

### Factorial

Uno de los ejemplos más básicos es el cálculo del factorial cuya fórmula se muestra a continuación:

$$n! = \prod_{i=1}^n p_i = 1 \times 2 \times 3 \dots \times (n - 1) \times n$$

**Tip:** En las notebooks se pueden agregar expresiones matemáticas usando Latex (<https://latex-project.org/intro.html>), tal y como se muestra en la fórmula anterior. La fórmula se tiene que encerrar usando \$\$ al principio y al final de la misma.

En el siguiente ejemplo se calcula el factorial de un número de forma iterativa usando un ciclo for

```
def factorial_no_recurutivo(numero):
    fact = 1
    #Se genera una lista que va de n a 1, el -1 indica que cada iteración se resta 1 al índice.
    for i in range(numero, 1, -1):
        fact *= i # Esto es equivalente a fact = fact * i
    return fact

factorial_no_recurutivo(5)
```

120

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código:	MADO-19
		Versión:	01
		Página	156/165
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Como se mencionó anteriormente, para resolver un problema por medio de recursividad hay que generar problemas más pequeños. Analizando la forma en que se calcula el factorial en la función pasada se tiene que:

$$5!=5\times 4\times 3\times 2\times 1$$

Si se remueve el 5 se tiene:

$$4!=4\times 3\times 2\times 1=4(4-1)!=4\times(3!)$$

Se puede afirmar que

$$4\times 3!=4\times[3(3-1)!]=4\times 3\times(2!)$$

Si se aplica esto a toda la secuencia, al final tenemos la siguiente expansión:

$$5!=5(4!)=5\times 4\times(3!)=5\times 4\times 3\times(2!)=5\times 4\times 3\times 2\times(1!)=5\times 4\times 3\times 2\times 1\times(0!)=120$$

Aplicando las reglas explicadas en un principio sobre recursividad, se puede resolver el problema del factorial por medio de recursión de la siguiente manera:

```
def factorial_recursoivo(numero):
    if numero < 2:      #El caso base es cuando numero < 2 y la función regresa 1
        return 1
    return numero * factorial_recursoivo(numero - 1) #La función se llama a sí misma

factorial_recursoivo(5)
```

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código:	MADO-19
		Versión:	01
		Página	157/165
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

De la ejecución de factorial\_recursivo() se puede observar lo siguiente:

- El caso base permite terminar la recursión.
- Conforme se va decrementando la variable numero, se aproxima al caso base. El caso base ya no necesita recursión debido a que se convirtió en la versión más simple del problema.
- La función se llama a sí misma y toma el lugar del ciclo for usado en la función factorial\_no\_recursivo().
- Cada que se llama de nuevo a la función, ésta tiene la copia de las variables locales y el valor de los parámetros.

**Tip:** En el caso de Python, hay un límite en el número de veces que se puede llamar recursivamente una función, si se excede ese límite se genera el error: maximum recursion depth exceeded in comparison. Este límite puede ser modificado, pero no es recomendable.

## Huellas de tortuga

Para el siguiente ejemplo, se va a utilizar la biblioteca turtle. Como se observa en la siguiente imagen, hay una tortuga que se desplaza en espiral. Este ejemplo ha sido tomado del tutorial de la biblioteca turtle que se puede consultar en [http://openbookproject.net/thinkcs/python/english3e/hello\\_little\\_turtles.html](http://openbookproject.net/thinkcs/python/english3e/hello_little_turtles.html).

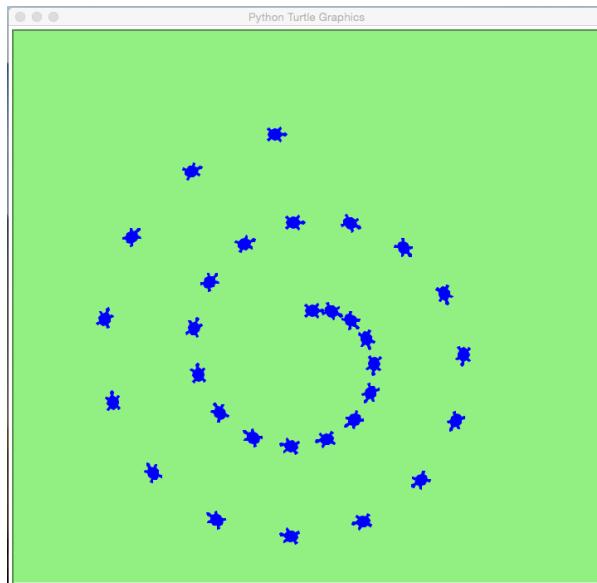
El objetivo es hacer que la tortuga deje un determinado número de huellas, cada una de las huellas se va a ir espaciando incrementalmente mientras ésta avanza. A continuación se muestra la sección de código que hace el recorrido de la tortuga.

NOTA: La siguiente sección de código no se va a ejecutar en la notebook

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código: MADO-19 Versión: 01 Página 158/165 Sección ISO 8.3 Fecha de emisión 20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B
La impresión de este documento es una copia no controlada		

#Archivo: recorrido\_no\_recursivo.py

```
for i in range(30):      #Esta determinado que se impriman 30 huellas de la tortuga
    tess.stamp()          # Huella de la tortuga
    size = size + 3        # Se incrementa el paso de la tortuga cada iteración
    tess.forward(size)     # Se mueve la tortuga hacia adelante
    tess.right(24)         # y se gira a la derecha
```



¿Cómo hacer el recorrido de la tortuga de manera recursiva? Primero se tiene que encontrar el caso base y después hacer una función que se va llame a sí misma. En esta función, el caso base es cuando se ha completado el número de huellas requerido. A continuación, se muestra el código de la función para el recorrido de la tortuga.

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código:	MADO-19
		Versión:	01
		Página	159/165
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

```
#Archivo: recorrido_recursivo.py
def recorrido_recursivo(tortuga, espacio, huellas):
    if huellas > 0:
        tortuga.stamp()
        espacio = espacio + 3
        tortuga.forward(espacio)
        tortuga.right(24)
        recorrido_recursivo(tortuga, espacio, huellas-1)
```

NOTA: El código completo de las dos versiones se encuentra guardado en los archivos `recorrido_recursivo.py` y `recorrido_no_recursivo.py`. Estos se van a ejecutar desde la notebook como si de una ventana de comandos se tratara.

Tip: En las notebooks se pueden ejecutar comandos del sistema operativo, sólo se tiene que agregar el signo de admiración antes del comando (`!comando`). Si el comando no es del sistema operativo, se despliega un aviso.

Al momento de ejecutar las siguientes instrucciones, se abre una ventana donde se muestra el desplazamiento de la tortuga. Cuando se termina de ejecutar el código, es necesario cerrar la ventana para que finalice la ejecución en la notebook.

```
#Ejecutando el código no recursivo.
!python recorrido_no_recursivo.py
```

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código:	MADO-19
		Versión:	01
		Página	160/165
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

**Tip:** Para la implementación recursiva (recorrido\_recursivo.py) se hace uso de la biblioteca argparse, esta biblioteca permite mandar datos de entrada al programa por medio de banderas, tal y como se hace con los comandos del sistema operativo.

```

ap = argparse.ArgumentParser()

#El dato de entrada se ingresa con la bandera --huellas
ap.add_argument("--huellas", required=True, help="número de huellas")

#Lo que se obtiene es un diccionario (llave:valor) , en este caso llamado args
args = vars(ap.parse_args())

# Los valores del diccionario son cadenas por lo que se tiene que
# transformar a un entero con la función int()
huellas = int(args["huellas"])

```

El código se ejecuta de la siguiente manera:

```

# Como se observa, hay un espacio después del nombre del archivo
# y un espacio después de la bandera
!python recorrido_recursivo.py --huellas 25

```

La ventaja de utilizar esta forma de mandar datos de entrada al programa, es que hace la validación por nosotros, ya que si no se especifica la bandera o se especifica un valor, se genera un mensaje de error.

```
!python recorrido_recursivo.py --huella
```

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código:	MADO-19
		Versión:	01
		Página	161/165
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

## Fibonacci

Recordando, la implementación iterativa para calcular la sucesión de Fibonacci es:

```
def fibonacci_iterativo_v2(numero):
    f1=0
    f2=1
    for i in range(1, numero-1):
        f1,f2=f2,f1+f2 #Asignación paralela
    return f2
```

Esta función se puede transformar a su versión recursiva de la siguiente manera:

```
def fibonacci_recursivo(numero):
    if numero == 1:      #Caso base
        return 0
    if numero == 2 or numero == 3:
        return 1
    return fibonacci_recursivo(numero-1) + fibonacci_recursivo(numero-2) #Llamada recursiva

fibonacci_recursivo(13)
```

Al igual que en la versión iterativa, se están repitiendo operaciones. Para calcular el elemento 5 se tiene:

$$\begin{aligned}f(5) &= \\(n-1) &= f(4)+f(3)+f(2)+f(1) \\(n-2) &= f(3)+f(2)+f(1)\end{aligned}$$

Retomando lo visto en la práctica 11, es posible mejorar la eficiencia del algoritmo si se utiliza **memorización**.

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código:	MADO-19
		Versión:	01
		Página	162/165
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

```
#Memoria inicial
memoria = {1:0, 2:1, 3:1}

def fibonacci_memo(numero):
    if numero in memoria:          #Si el número ya se encuentra calculado, se regresa el valor ya no se hacen más cálculos
        return memoria[numero]
    memoria[numero] = fibonacci_memo(numero-1) + fibonacci_memo(numero-2)
    return memoria[numero]

fibonacci_memo(13)
```

La memoria cambia después de la ejecución. En comparación con la versión iterativa de la guía 11, la función fibonacci\_memo() tiene acceso a la variable memoria, por lo que efectúa menos operaciones.

memoria:

```
{1: 0,
 2: 1,
 3: 1,
 4: 2,
 5: 3,
 6: 5,
 7: 8,
 8: 13,
 9: 21,
10: 34,
11: 55,
12: 89,
13: 144}
```

A diferencia de la versión anterior, como los resultados se están guardando en la variable memoria, el número de operaciones que se realizan es menor. Para calcular el elemento 5 con la nueva implementación se tiene:

memoria = {1:0, 2:1, 3:1}

$$\begin{aligned} f(5) &= \\ (n-1) &= f(4)+\text{memoria}(3)+\text{memoria}(2)+\text{memoria}(1) \\ (n-2) &= \text{memoria}(3) \end{aligned}$$



**Manual de prácticas del  
Laboratorio de Estructuras de  
datos y algoritmos I**

Código:	MADO-19
Versión:	01
Página	163/165
Sección ISO	8.3
Fecha de emisión	20 de enero de 2017

Facultad de Ingeniería

Área/Departamento:  
Laboratorio de computación salas A y B

La impresión de este documento es una copia no controlada

	<b>Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos I</b>	Código: MADO-19
		Versión: 01
		Página 164/165
		Sección ISO 8.3
		Fecha de emisión 20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B
		La impresión de este documento es una copia no controlada

## Desventajas de la recursividad

- A veces es complejo generar la lógica para aplicar recursión.
- Hay una limitación en el número de veces que una función puede ser llamada, tanto en memoria como en tiempo de ejecución.

## Bibliografía

Design and analysis of algorithms; Prabhakar Gupta y Manish Varshney; PHI Learning, 2012, segunda edición.

Introduction to Algorithms, Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest y Clifford Stein; The MIT Press; 2009, tercera edición.

Problem Solving with Algorithms and Data Structures using Python; Bradley N. Miller y David L. Ranum, Franklin, Beedle & Associates; 2011, segunda edición.

[http://openbookproject.net/thinkcs/python/english3e/hello\\_little\\_turtles.html](http://openbookproject.net/thinkcs/python/english3e/hello_little_turtles.html)