Universidad Nacional Autónoma de México
Facultad de Ingeniería
Estructuras de datos y algoritmos
Tema 1:
ESTRUCTURA DE DATOS

1 Estructura de datos

Objetivo: Resolver problemas de almacenamiento, recuperación y ordenamiento de datos y las técnicas de representación más eficientes, utilizando las estructuras para representarlos.

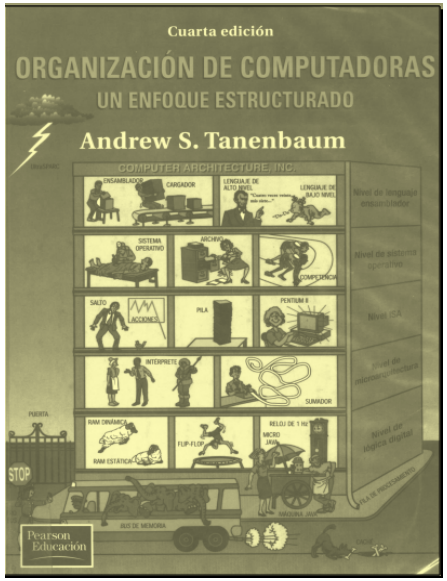
1 Estructura de datos

1.1 Representación de datos en memoria.
- 1.1.1 Tipos primitivos.
- 1.1.2 Arreglos.
- 1.1.3 Apuntadores.
- 1.1.4 Tipo de dato abstracto.

1.2 Administración del almacenamiento en tiempo de ejecución.

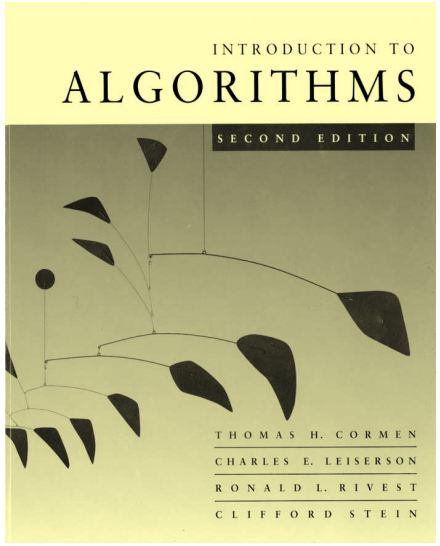
1.3 Estructura de datos compuestos.
- 1.2.2 Pila: almacenamiento contiguo y ligado, y operaciones.
- 1.2.3 Cola: almacenamiento contiguo y ligado, y operaciones.
- 1.2.4 Cola doble: almacenamiento contiguo y ligado, y operaciones.
- 1.2.5 Listas circular: almacenamiento contiguo y ligado, y operaciones.
- 1.2.6 Listas doblemente ligadas: almacenamiento contiguo y ligado, y operaciones.

Bibliografía



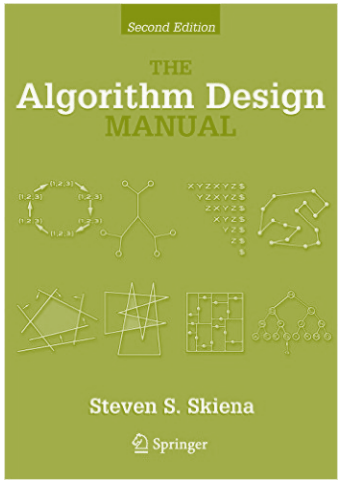
§Organización de computadoras, un enfoque estructurado. Andrew S. Tanenbaum, Prentice Hall.

Bibliografía



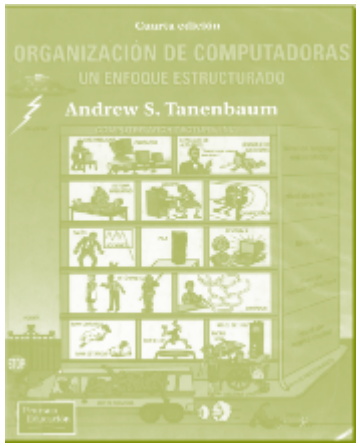
Introduction to Algorithms. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, McGraw-Hill.

Bibliografía



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

Bibliografía



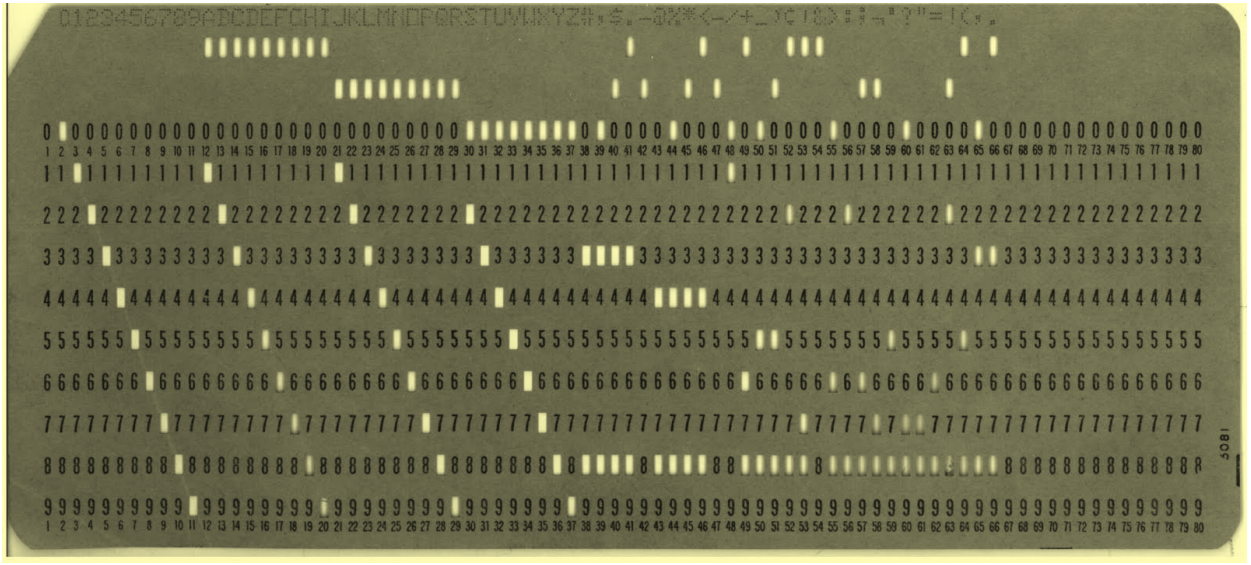
Organización de computadoras, un enfoque estructurado. Andrew S. Tanenbaum, Prentice Hall.

Bibliografía

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

Licencia GPL de GNU

```
/*
*
* 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
* E-mail: jorge.a.solano@hotmail.com
*
*/
```



1.1 Representación de datos en memoria.

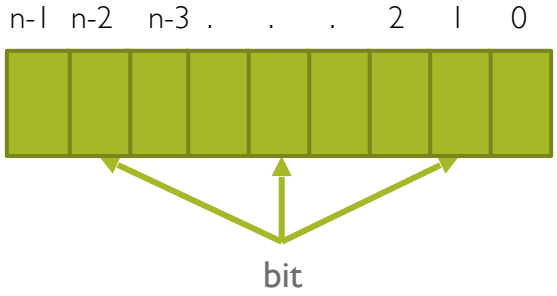
1.1 Representación de datos en memoria.

Cuando se requiere conservar información por un lapso de tiempo indefinido, ésta se almacena dentro de la memoria secundaria.

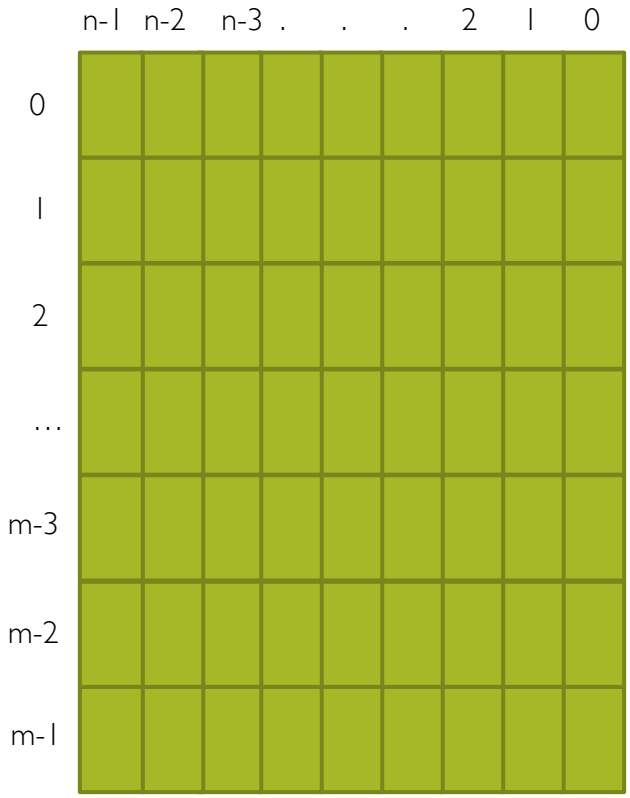
Cuando se desea crear, consultar o modificar la información almacenada se realiza dentro de la memoria principal.

En ambos casos, la información se representa en su forma más simple, es decir, en forma de bits (BInary digiTS), los cuales solo pueden tomar dos valores: 0 ó 1.

La longitud de una palabra de computadora corresponde al número de bits que la componen. En la mayoría de los equipos de cómputo esta longitud oscila entre 8 y 64 bits.



Un bit puede representar dos estados distintos (2^1). Una palabra de computadora de longitud n puede representar $2^n - 1$ estados distintos.



La memoria principal está constituida por un conjunto de *m* palabras de longitud *n*. A cada palabra de computadora se le asocia una dirección de memoria.

La representación de información en la memoria se realiza mediante un número finito de bits, dependiendo del tipo de dato que se desee almacenar.

El número palabras que se requieren en la memoria para almacenar un tipo de dato dependen de la longitud de palabra que maneja la computadora.

Medida	2^n	Número de bits	Equivalencia
Byte	2^3	8	1 byte
Kilobyte	2^{10}	1 024	1024 bytes
Megabyte	2^{20}	1 048 576	1024 KB
Gigabyte	2^{30}	1 073 741 824	1024 MB
Terabyte	2^{40}	1 099 511 627 776	1024 GB
Petabyte	2^{50}	1 125 899 906 842 624	1024 TB
Exabyte	2^{60}	1 152 921 504 606 846 976	1024 PB
Zetabyte	2^{70}	1 180 591 620 717 411 303 424	1024 EB
Yottabyte	2^{80}	1 208 925 819 614 629 174 706 176	1024 ZB
Brontobyte	2^{90}	1 237 940 039 285 380 274 899 124 224	1024 YB
Geopbyte	2^{100}	1 267 650 600 228 229 401 496 703 205 376	1024 BB

Tabla 1. Medidas de almacenamiento de información.

1.1.1 Tipos primitivos.

Los lenguajes de programación más próximos a la arquitectura de la computadora (hardware) se denominan lenguajes de bajo nivel.

Los lenguajes de bajo nivel son códigos dependientes del procesador, es decir, el programa que se realiza con este tipo de lenguajes no se pueden migrar o utilizar en otras arquitecturas.

Los lenguajes de programación más próximos al lenguaje del usuario se denominan lenguajes de alto nivel.

Un lenguaje de programación de alto nivel maneja representaciones de, por lo menos, tres tipos de datos: caracteres, números enteros y números reales.

Representación en memoria: Tipo de dato carácter.

En memoria, un carácter ocupa 8 bits y se representa en sistema binario dependiendo del tipo de codificación que esté ocupando la máquina. Las representaciones de datos más utilizadas son código ASCII, código EBCDIC y UNICODE.

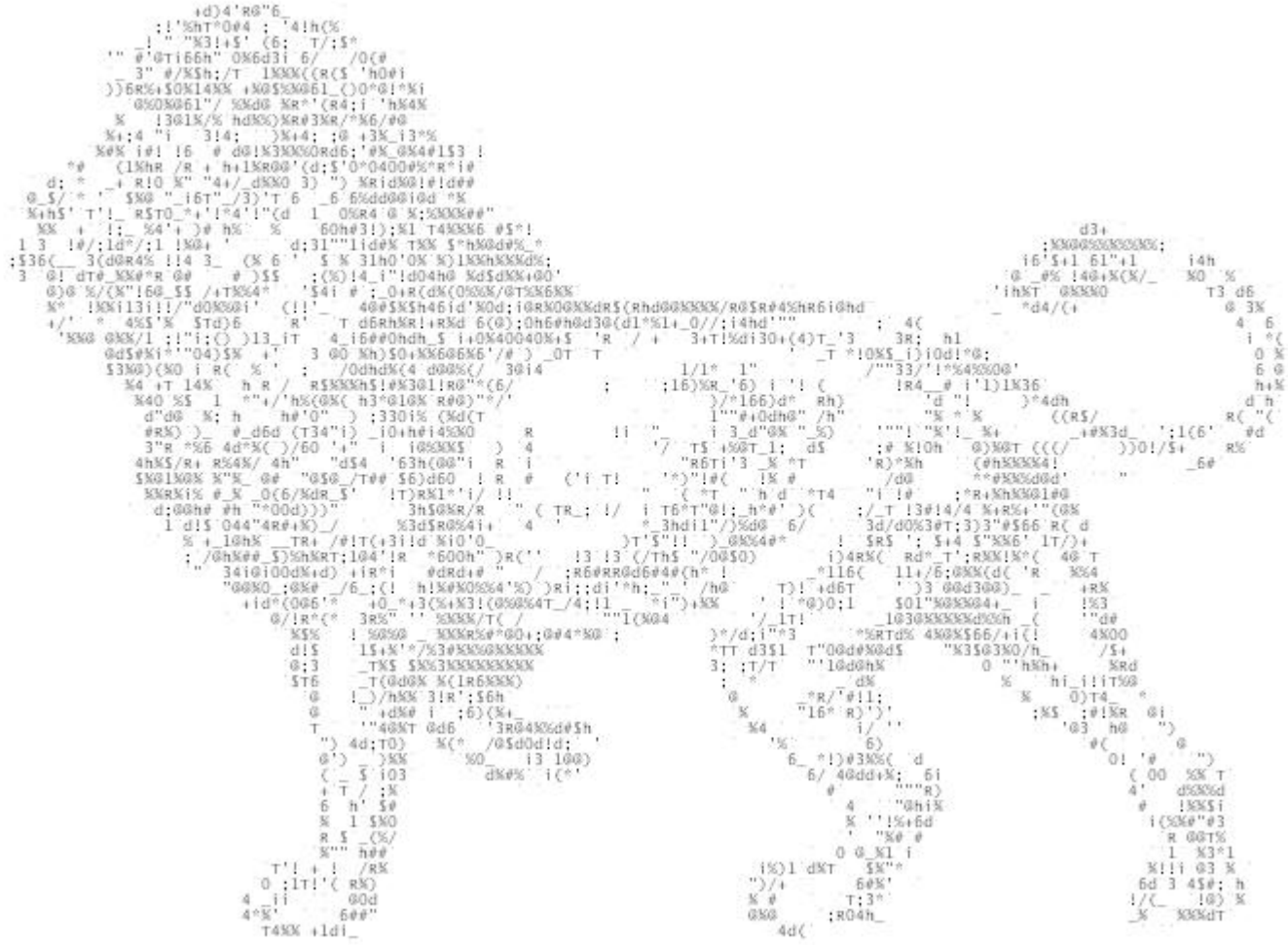
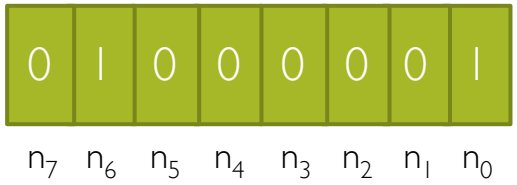


Figura 1. Tabla de código ASCII (Enlace de la imagen)

Ejemplo 1

En código ASCII, el carácter 'A' se representa por el código 41_{16}. Este número en binario se representa por los dígitos 0100 0001. Su representación en memoria es la siguiente:



Ejemplo 1

```
// Almacenamiento de carácter en memoria

#include <stdio.h>

int main(){
    char caracterA = 'A';
    printf("La codificación del carácter A\n");
       printf("en este sistema en hexadecimal es: %x\n",
caracterA);
    return 0;
}
```

Representación en memoria: Tipo de dato entero.

Dependiendo del lenguaje de programación y del tipo de dato que se declare, los números enteros pueden ocupar 8, 16, 32 ó 64 bits en memoria, donde el primer bit de izquierda a derecha (el bit más significativo) está reservado para el signo y los restantes definen la capacidad de almacenamiento del número.

Un número entero que ocupa 32 bits se distribuye en la memoria de la siguiente manera:

1 bit para el signo del número
31 bits para el número

+/-	n_{30}	n_{29}	n_{28}	n_{27}	n_{26}	n_{25}	n_{24}	n_{23}	n_{22}	n_{21}	n_{20}	n_{19}	n_{18}	n_{17}	n_{16}
n_{15}	n_{14}	n_{13}	n_{12}	n_{11}	n_{10}	n_9	n_8	n_7	n_6	n_5	n_4	n_3	n_2	n_1	n_0

Ejemplo 2

Convertir el número $28{,}345_{10}$ a binario y mostrar su representación en memoria en 32 bits.

$$28\,345_{10} = 110111010111001_2$$

28 345	2
14 172	1
7 086	0
3 543	0
1 771	1
885	1
442	1
221	0
110	1
55	0
27	1
13	1
6	1
3	0
1	1
0	1

Ejemplo 2

La representación en memoria del número $28{,}345_{10}$ es:

$$28{,}345_{10} = 1101110101111001_2$$

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	1	0	1	1	1	0	1	0	1	1	1	0	0	1

Es importante tener en cuenta que un número entero puede superar la capacidad de almacenamiento reservada para el tipo de dato y ello provoca pérdida de información (parcial o total).

Complemento aritmético.

Dependiendo de la arquitectura de la computadora (lógica del procesador), un dato entero puede ser almacenado en memoria con su complemento aritmético (cuando el número es negativo).

El completo aritmético (ar) de un número real se refiere a la cantidad que le falta a dicho número para ser igual a una unidad del orden inmediato superior.

Complemento a la base

El complemento aritmético ar (o complemento a la base) de un número real se obtiene a partir de la siguiente fórmula:

$$ar = r^n - |N|$$

donde:

ar: complemento aritmético de un número real base r.
r: es la base del número.
n: número de dígitos de la parte entera del número.
N: el número dado.

Ejemplo 3

Dado el número 789_{10}, obtener su complemento aritmético a la base ar (Complemento a10).

$$ar = r^n - |N|$$
$$ar = 10^3 - |789|$$
$$ar = 1000 - 789 = 211$$
$$ar = 211_{10}$$

Ejemplo 4

Obtener el resultado de restar 7895_{10} con 4999_{10}.

$$\begin{array}{r} 7\ 8\ 9\ 5 \\ -\ 1\ 1\ 1 \\ \hline 4\ 9\ 9\ 9 \\ \hline 2\ 8\ 9\ 6 \end{array}$$

Ejemplo 5

Obtener el resultado de restar 1001_2 con 1111_2.

$$
\begin{array}{rcccccccccc}
 & \ldots & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\
- & \ldots & 1 & 1 & 1 & 1 & 1 & 1 & 1 & & \\
\hline
 & \ldots & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
\hline
 & \ldots & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0
\end{array}
$$

El resultado de la resta es negativo y, por tanto, está dado en complemento a la base (complemento a2). El complemento del complemento obtiene el número en magnitud y signo:

$$n = (ar)r$$

Ejemplo 5

Para obtener la magnitud y signo del número anterior se debe obtener su complemento aritmético (complemento a2), es decir:

$$n = (ar)r$$
$$ar = r^n - |N|$$
$$ar = 2^6 - |111010|$$

$$\begin{array}{rcccccccc} & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ - & 1 & 1 & 1 & 1 & 1 & & \\ & 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ \hline & 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{array}$$

Ejemplo 5

Para un número que está dado en complemento el bit de signo es parte del número, por lo tanto, en el ejemplo anterior se puede afirmar que el número es negativo y, por ende, la magnitud y signo del número 111010_2 es -110_2.

Ejercicio 1

Obtener el complemento aritmético de los siguientes números:

Línea	Base 2	Base 10
1	11100101	84 356
2	11110000	61 912
3	11000011	70 007
4	11100001	49 234
5	10000111	81 765
6	11111000	34 963
7	10001111	52 124
8	11101111	25 684

Complemento a la base disminuida

El complemento aritmético menos uno (ar-1 o complemento a la base disminuida) de un número real se calcula con base en la siguiente fórmula:

$$ar\text{-}1 = r^{n} - r^{-m} - |N|$$

donde:

ar-1: complemento aritmético de un número real base r.
r: es la base del número.
n: número de dígitos de la parte entera del número.
m: número de dígitos de la parte fraccionaria del núm.
N: el número dado.

Ejemplo 6

Dado el número 789_{10}, obtener su complemento aritmético a la base disminuida ar-1 (Complemento a10-1).

$$ar = r^n - r^m - |N|$$
$$ar = 10^3 - 10^{-0} - |789|$$
$$ar = 1000 - 1 - 789 = 210$$
$$ar = 210_{10}$$

Ejemplo 7

Dado el número 1001.012, obtener su complemento aritmético a la base ar-1(Complemento a2-1).

$$ar\text{-}1 = r^{n} - r^{-m} - |n|$$
$$ar\text{-}1 = 2^{4} - 2^{-2} - |1001.01|$$
$$ar\text{-}1 = 10000 - 0.01 - 1001.01$$
$$ar\text{-}1 = 10000.0 - 1001.1$$
$$ar\text{-}1 = 00110.1$$
$$ar\text{-}1 = 110.1_{2}$$

Ejercicio 2

Obtener el complemento aritmético a la base disminuida de los siguientes números:

Línea	Base 2	Base 10
1	10000111	49 234
2	11111000	21 816
3	10001111	57 362
4	11101111	39 349
5	11100101	93 428
6	11110000	71 912
7	11000011	60 001
8	11100001	89 258

Ejemplo 8

Realizar siguiente operación 123 – 98 en magnitud y signo, en complemento a la base y en complemento a la base disminuida.

Magnitud y signo

```
    1 2 3
  – 1 1
      9 8
  -------
    0 2 5
```

Complemento ar

```
    1 2 3
  +
    9 0 2
  -------
    0 2 5
```

Complemento ar-1

```
    1 2 3
  +
    9 0 1
  -------
    0 2 4
  +     1
  -------
    0 2 5
```

Ejemplo 9

Realizar siguiente operación 123 – 125 en magnitud y signo, en complemento a la base y en complemento a la base disminuida.

Magnitud y signo

```
  1 2 3
–
  1 2 5
-------
– 0 0 2
```

Complemento ar

```
  1 2 3
+
  8 7 5
-------
  9 9 8
-------
– 0 0 2
```

Complemento ar-1

```
  1 2 3
+
  8 7 4
-------
  9 9 7
+     1
-------
  9 9 8
-------
– 0 0 2
```

Ejemplo 10

Se tiene un tipo de dato que ocupa 3 bits en memoria, por tanto, solo es posible almacenar 2^2 datos diferentes, ya que un bit está reservado para el bit de signo.

			Magnitud y signo	Complemento a2	Complemento a2-1
0	1	1	3	3	3
0	1	0	2	2	2
0	0	1	1	1	1
0	0	0	0	0	0
1	0	0	-0	-4	-3
1	0	1	-1	-3	-2
1	1	0	-2	-2	-1
1	1	1	-3	-1	-0

Ejemplo 11

El número entero -1101110101111001_2 se desea representar en memoria en complemento ar.

Para representar el complemento del número es necesario expresarlo con el número máximo de elementos que es posible almacenar en memoria, es decir, para un tipo de datos de que ocupa 32 bits en memoria, el número quedaría expresado de la siguiente manera:

$$-00000000000000001101110101111001_2$$

Ejemplo 11

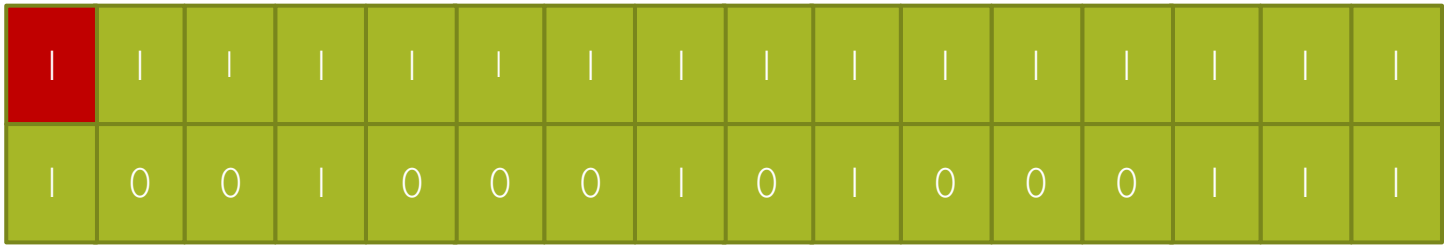
Una vez que se posee el número representado en la totalidad de bits a almacenar, se procede a obtener su complemento aritmético:

$$-00000000000000001101110101110011_2 =$$
$$11111111111111110010001010001111_2$$

Ejemplo 11

Por lo tanto, la representación en memoria del número entero -1101110101110011_2 en complemento aritmético es:

$$1111111111111111001000101000111_2$$



Representación en memoria: Tipo de dato real.

En la práctica se está constantemente trabajando con números reales, sin embargo, estos números no siempre se pueden representar de manera exacta debido al tamaño máximo de bits que puede almacenar un tipo de dato, por lo tanto, los números necesariamente son redondeados o truncados, generando un error matemático.

Para almacenar en memoria un número real la computadora puede hacer uso de la notación de punto flotante normalizado o la notación de punto flotante no normalizada.

Notación científica no normalizada

Un número A puede ser expresado en notación científica no normalizada, es decir, el número se expresa como una potencia de 10:

$$A = C \times 10^{n}$$

Donde

C: número entero mayor o igual a uno ($C \in Z$).
n: es un número entero positivo, negativo o cero ($n \in Z$).

Ejemplo 12

Se desea expresar el número 836.238_{10} en la forma de punto flotante no normalizada, por tanto:

$$836.238_{10} = 836238 \times 10^{-3}.$$

NOTA: Cuando el punto decimal se recorre hacia la derecha del número, el exponente es negativo.

Notación científica normalizada

Un número A puede ser expresado en notación científica normalizada, es decir, el número se expresa como una potencia de 10:

$$A = C \times 10^{n}$$

Donde

C: número menor a uno ($C \in R$).
n: es un número entero positivo, negativo o cero ($n \in Z$).

Ejemplo 13

Se desea expresar el número 836.238_{10} en la forma de punto flotante normalizada, por tanto:

$$836.238_{10} = 0.836238 \times 10^{3}.$$

NOTA: Cuando el punto decimal se recorre hacia la izquierda del número, el exponente es positivo.

En este curso se trabajará con la notación científica normalizada (punto flotante normalizada).

Los número binarios también pueden ser representados en notación científica normalizada (notación de punto flotante normalizada) de la siguiente manera:

$$A = M \times 2^n$$

Donde

n es un entero positivo, negativo o cero (expresado en binario).
M es la mantisa (dígitos significativos del número), que debe ser menor a 1.

Ejemplo 14

Expresar los números 11111.01_2 y -0.0000001110110_2 en su forma científica normalizada.

$$A = M \times 2^n$$

$$11111.01_2 = 0.1111101_2 \times 2^{101}$$

$$-0.0000001110110_2 = -0.1110110_2 \times 2^{-110}$$

Un número real de precisión sencilla (flotante) ocupa 32 bits (4 bytes) en memoria y se distribuyen de la siguiente manera:

1 bit para el signo de la mantisa
1 bit para el signo del exponente
7 bits para el exponente entero (en binario)
23 bits para la mantisa (en binario)

$\pm$	$\pm$	e_6	e_5	e_4	e_3	e_2	e_1	e_0	m_0	m_1	m_2	m_3	m_4	m_5	m_6
m_7	m_8	m_9	m_{10}	m_{11}	m_{12}	m_{13}	m_{14}	m_{15}	m_{16}	m_{17}	m_{18}	m_{19}	m_{20}	m_{21}	m_{22}

Ejemplo 15

Convertir el número 31.25_{10} a sistema binario y dibujar su representación en memoria.

$$31.25_{10} = 0.3125_{10} \times 10^2$$

$$0.3125_{10} \times 10^2 = 0.3125_{10} \times 100_{10}$$

Ejemplo 15

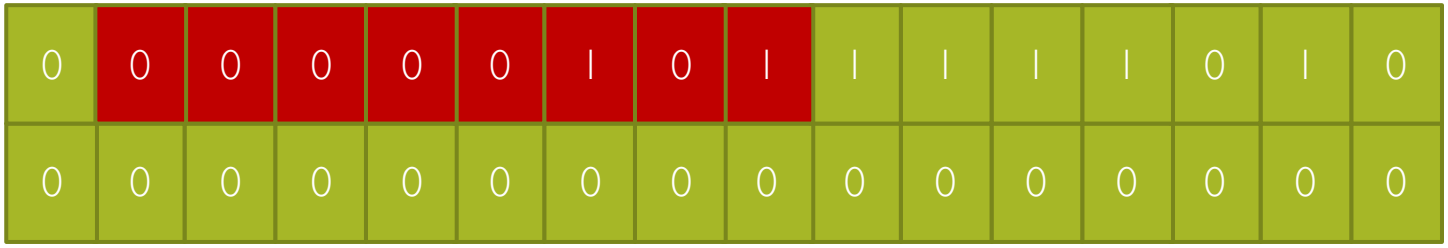
0.3125	* 2
0.6250	0
0.2500	1
0.5000	0
0.0000	1

100	/ 2
50	0
25	0
12	1
6	0
3	0
1	1
0	1

$$31.25_{10} = 0.01011_2 \times 1100100_2 = 11111.01_2$$
$$11111.01_2 = 0.1111101_2 \times 2^{101}$$

Ejemplo 15

$$0.1111101_2 \times 2^{101}$$



Ejemplo 16

Representar el número en memoria -0.000721619 para un tipo de datos que ocupa de 32 bits.

Primero se convierte el número de decimal a binario:

$$-0.000721619_{10} = -0.00000000001011110100100101100000011_{2}$$

Ejemplo 16

Después se convierte el número a su forma científica normalizada:

$$-0.10111101001001011000000011_2 \times 2^{-1010}$$

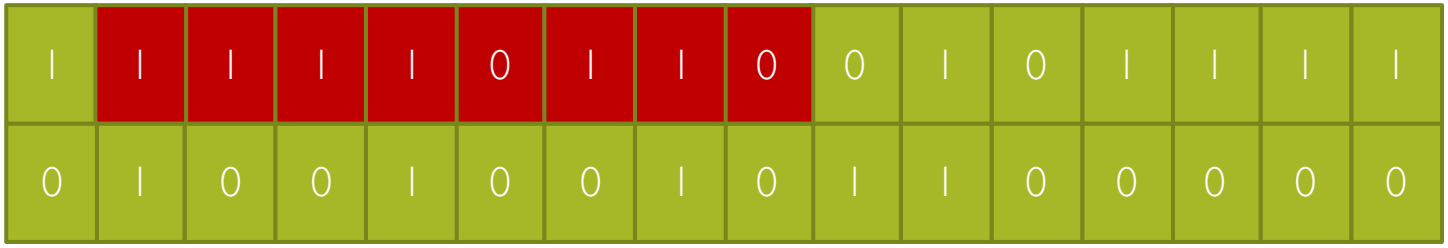
Como el exponente es negativo, se debe pasar a complemento a la base (a2) a 8 cifras ya que el signo es parte del número (se puede aplicar complemento ar-1 y, al resultado, sumarle 1):

$$-00001010_2 = 11110101_2 + 1_2 = 11110110_2$$

Ejemplo 16

Por lo tanto, la representación en memoria del número real en memoria es:

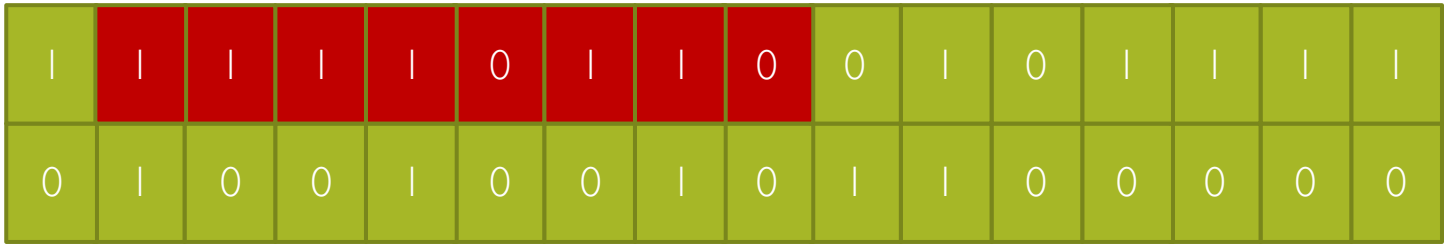
$$-0.101111010010010110000000\,11_2 \times 2^{11110110}$$



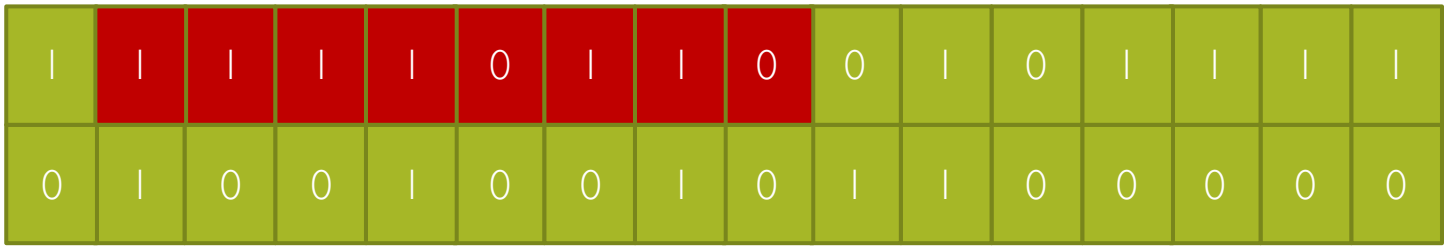
Nótese que existe pérdida de información.

Ejemplo 17

Obtener el número almacenado a partir de la siguiente representación en memoria:



Ejemplo 17



Lo primero que se debe realizar es la extracción de los dígitos, es decir, se sabe que el primer dígito de la memoria corresponde al signo de la mantisa, los siguientes 8 bits corresponden al exponente del número y el resto de bits corresponden a la mantisa. Así mismo, se sabe que el primer dígito de la mantisa no se escribe y que se está representado en su forma de punto flotante normalizada, por tanto, la extracción del número queda de la siguiente manera:

$$-0.10101111010010010110000 \times 2^{11110110}$$

Ejemplo 17

Debido a que el bit de signo del exponente es negativo, se sabe que el número está en complemento, por tanto, hay que obtener el valor en magnitud y signo:

$$11110110_2 = -1010_2$$

Por lo tanto, el número almacenado es el siguiente:

$$-\ 0.10101111010010010110000 \times 2^{-1010}$$

Ejemplo 17

Para convertir el número de base 2 a base 10 se eleva a la enésima posición los bits que están encendidos (los unos) de la mantisa, se suman todos y se multiplican por la base elevada al exponente, es decir:

$$- 0.10101111010010010110000 \times 2^{-1010}$$

$$-(2^{-1} + 2^{-3} + 2^{-5} + 2^{-6} + 2^{-7} + 2^{-8} + 2^{-10} + 2^{-13} + 2^{-16} + 2^{-18} + 2^{-19}) \times 2^{-10}$$

Ejemplo 17

```
// Almacenamiento de número real en memoria

#include<stdio.h>

int main(){
    float num = -0.000721619548;
    printf("%.12f", num);
}
```

Tarea 1

Se quiere almacenar el resultado de multiplicar 35.34 x 7.23 en un lenguaje de programación para un tipo de dato de 16 bits. Dibujar su representación en memoria. Considerar 2 bits para los signos, 5 bits para el exponente y 9 bits para la mantisa.

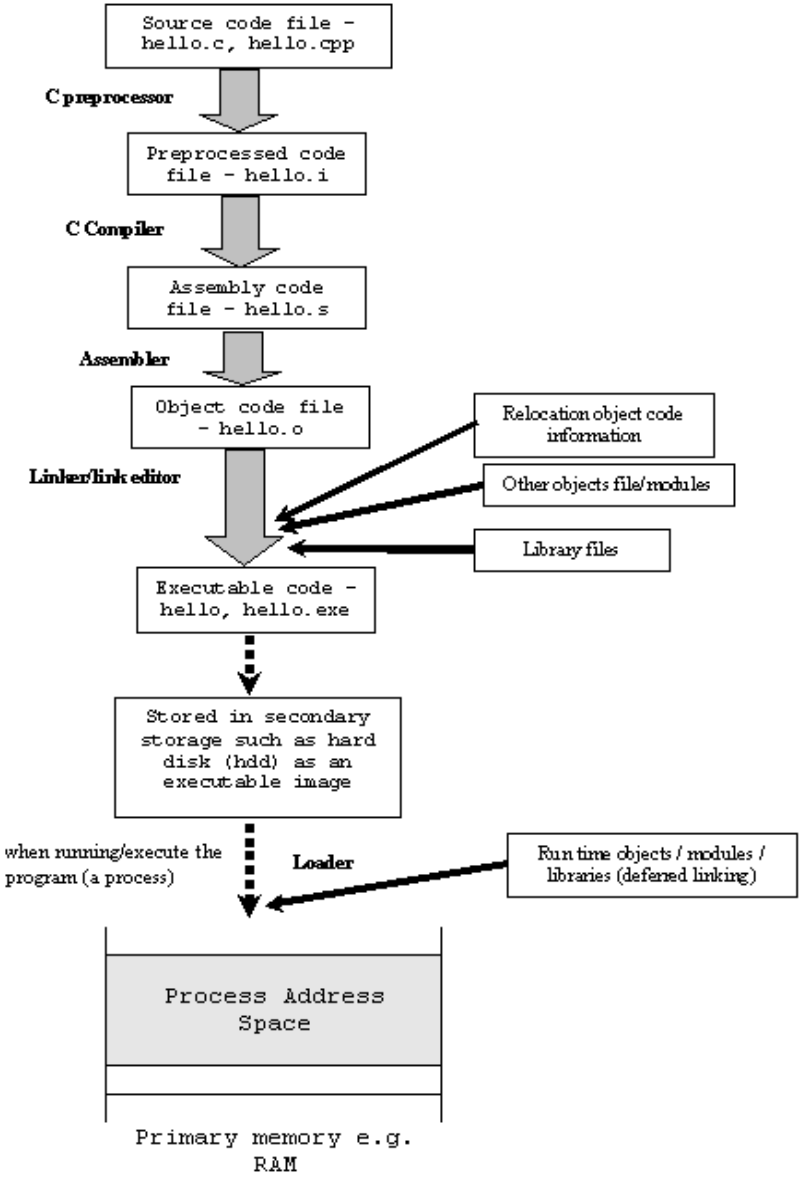
"There is no programming languaje – no matter how structured – that will prevent programmers from making bad programs."

Larry Flon
(He worked at USC's Information Sciences Institute)

Lenguaje C

El compilador de C sigue cuatro etapas para pasar el código fuente a código ejecutable, estas etapas están compuestas por:

- Preprocesador: antes de iniciar el proceso de compilación, se sustituyen o interpretan las directivas especiales del preprocesador, las que inician con #.
- Compilación: proceso por el cual un programa escrito en un lenguaje es traducido a código objeto.
- Ensamblado: traduce el código objeto a código máquina.
- Ligador: si el código requiere otro archivo fuente, éste se liga y se genera el archivo ejecutable.


Source code file - hello.c, hello.cpp
C preprocessor
Preprocessed code file - hello.i
C Compiler
Assembly code file - hello.s
Assembler
Object code file - hello.o
Linker/link editor
Relocation object code information
Other objects file/modules
Library files
Executable code - hello, hello.exe
Stored in secondary storage such as hard disk (hdd) as an executable image
when running/execute the program (a process)
Loader
Run time objects / modules / libraries (deferred linking)
Process Address Space
Primary memory e.g. RAM

Lenguaje C maneja diferentes tipos de datos enteros y de punto flotante, además de caracteres. Los tipos de datos básicos son:

- Caracteres: codificación definida por la máquina.
- Enteros: números sin punto decimal.
- Reales: números fraccionarios de precisión normal y de doble precisión.

Variables en lenguaje C: tipo carácter.

Tipo	Bits	Valor mínimo	Valor máximo
signed char	8	-128	127
unsigned char	8	0	255

Si se omite el clasificador, por defecto se considera "signed".

Variables en lenguaje C: tipo entero.

Tipo	Bits	Valor mínimo	Valor máximo
signed short	16	-32 768	32 767
unsigned short	16	0	65 535
signed int	32	-2 147 483 648	2 147 483 647
unsigned int	32	0	4 294 967 295
signed long	32	-2 147 483 648	2 147 483 647
unsigned long	32	0	4 294 967 295
enum	16	-32 768	32 767

Si se omite el clasificador, por defecto se considera "signed".

Variables en lenguaje C: tipo real.

Tipo	Bits	Valor mínimo	Valor máximo
float	32	$3.4E^{-38}$	3.4E38
double	64	$1.7E^{-308}$	$1.7E^{-308}$
long double	96	$3.4E^{-4932}$	$3.4E^{-4932}$

Las variables reales siempre poseen signo.

“If you lie to the compiler, it will get its revenge.”

Henry Spencer
(A Canadian computer programmer, he wrote "regex",
a widely used software library for regular expressions)

Ejemplo 18

```
// Tamaño de almacenamiento de datos en memoria

#include<stdio.h>

int main(){
    printf("%d\n", sizeof(char));
    printf("%d\n", sizeof(short));
    printf("%d\n", sizeof(int));
    printf("%d\n", sizeof(long));
    printf("%d\n", sizeof(float));
    printf("%d\n", sizeof(double));
    printf("%d\n", sizeof(long double));
}
```

Cada tipo de dato posee un especificador para el formato de impresión:

Tipo de dato	Especificador de formato
Entero	%d, %i, %o, %x
Real	%f, %lf, %e, %g
Carácter	%c, %d, %i, %o, %x
Cadena de caracteres	%s

Secuencias de escape

Las secuencias de escape están constituidas por dos caracteres aunque representan uno solo:

\a	carácter de alarma
\b	retroceso
\f	avance de hoja
\n	salto de línea
\r	regreso de carro
\t	tabulador horizontal
\v	tabulador vertical
'\0'	carácter nulo

Ejemplo 19

```
// Secuencias de escape

#include<stdio.h>

int main(){
    long a = 1234567890;
    double be = 8.123456789;
    char ce = 'c';
    char de[] = "Hola";

    printf("\a %d \t %i \t %o \t %x \n", a,a,a,a);
    getchar();
    printf("\a %f \t %lf \t %e \t %g \n", be,be,be,be);
    getchar();
        printf("\a  %c  \t  %d  \t  %i  \t  %o  \t  %x  \n",
ce,ce,ce,ce,ce);
    getchar();
    printf("\a %s", de);
}
```

"You might not think that programmers are artists, but programming is an extremely creative profession. It's logic-based creativity."

John Romero
(Is an American director, designer, programmer, and developer in the video game industry.)

© Jorge A. Solano

1.1.2 Arreglos.

Se denomina arreglo a un conjunto de datos, contiguos o ligados, que son del mismo tipo. Un arreglo posee un tamaño fijo definido al momento de crearse.

A cada localidad de un arreglo (elemento) se le asocia un número (índice). Es posible crear arreglos unidimensionales y multidimensionales.

Un arreglo contiguo es aquel que se declara en tiempo de compilación y permanece estático durante toda la ejecución del programa, 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 memoria dinámica (tema 1.3).

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).

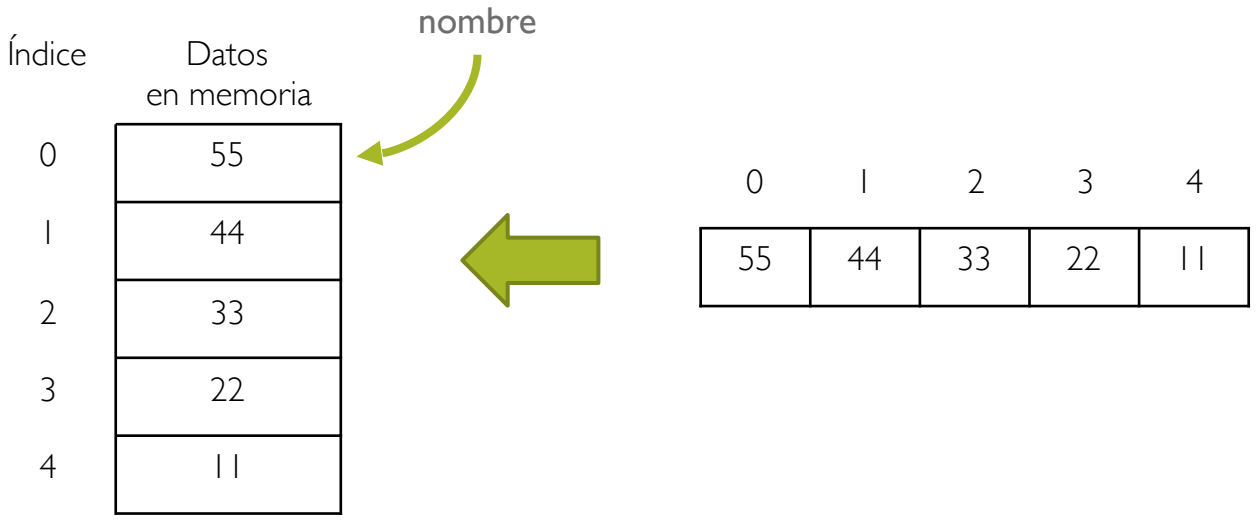
Los arreglos multidimensionales están constituidos por localidades de memoria (ya sea contiguas o ligadas) ordenadas bajo un mismo nombre y que pueden tener varios niveles (varias dimensiones) que van desde el plano (dos dimensiones) hasta la enésima dimensión.

En lenguaje C, la primera localidad de un arreglo corresponde al índice 0 y la última corresponde al índice n-1, donde n es el tamaño de cada dimensión.

La sintaxis para declarar arreglos en lenguaje C está determinada por el tipo de dato (el valor que se puede almacenar en el arreglo), el nombre del arreglo (el identificador con el que se puede acceder a cada elemento), las dimensiones (determinadas mediante los corchetes que abren y cierran) y el tamaño de cada dimensión (número entero que se define dentro de los corchetes de cada dimensión), es decir:

tipo_dato nombre_Arreglo [tam][tam][tam]…[tam]

Representación en memoria de un arreglo unidimensional.



La fórmula para acceder a una localidad de memoria está dada por la posición del inicio del arreglo (nombre) más el índice del elemento.

Ejemplo 20

```
//    Almacenamiento    en    memoria    de    arreglo
unidimensional

#include<stdio.h>

int main(){
    short nums[] = {5,4,3,2,1}, cont;

    for (cont = 0; cont < 5 ; cont++)
        printf("%x\n",&nums[cont]);

    return 0;
}
```

Representación en memoria de un arreglo bidimensional.

	0	1	2
0	99	88	77
1	66	55	44
2	33	22	11
3	999	555	111

Dato	Índice
99	0
88	1
77	2
66	3
55	4
44	5
33	6
22	7
11	8
999	9
555	10
111	11

nombre

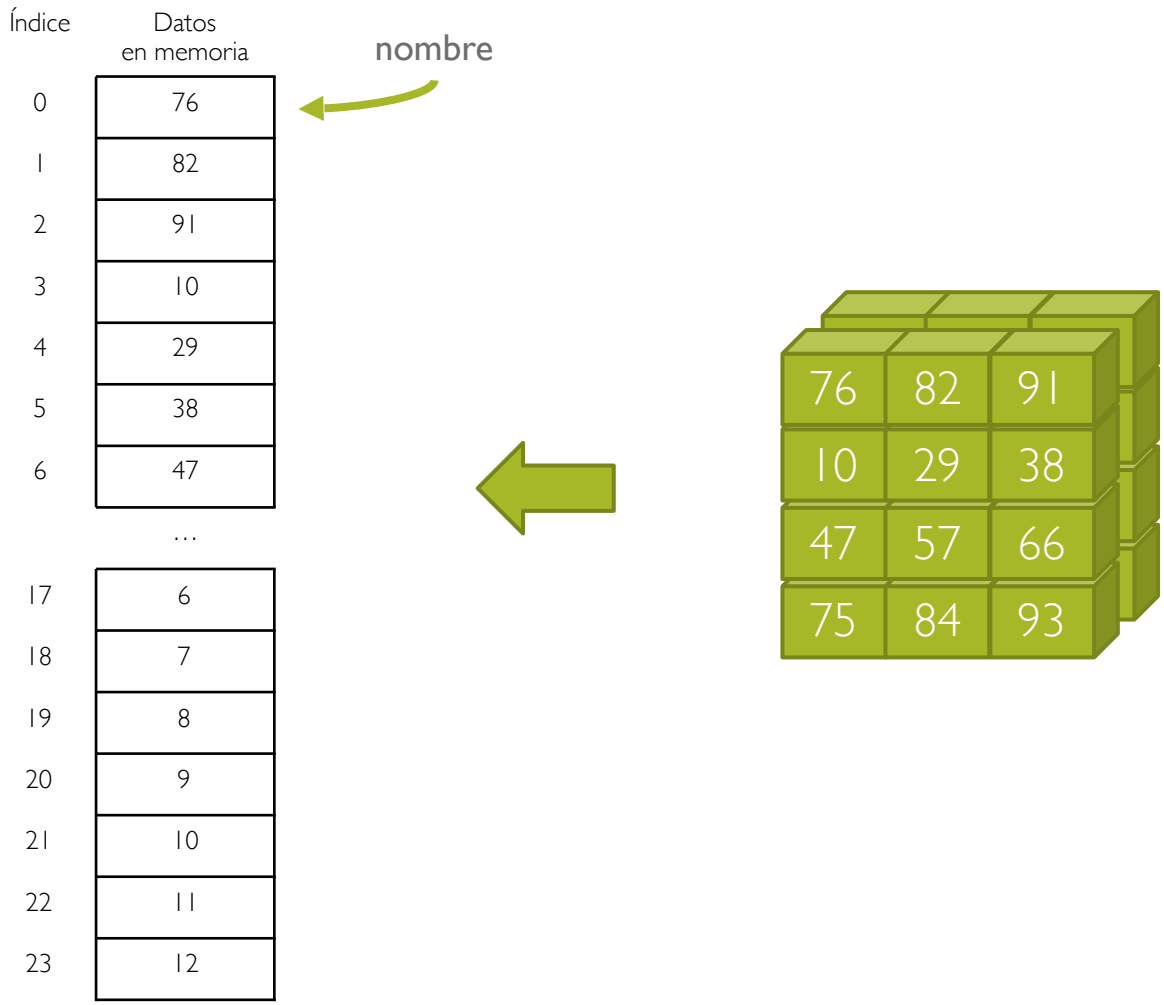
¿Cuál es la fórmula para acceder al un elemento en la memoria que se encuentra en las coordenadas [renglón, columna] de una matriz de MxN?

¿(renglón*N) + columna?

Ejemplo 21

```
// Almacenamiento en memoria de arreglo bidimensional

#include<stdio.h>

int main(){
    short renglon, columna;
          short   nums[4][3]   =   {{99,88,77},{66,55,44},{33,22,11},
{999,555,111}};
    printf("Arreglo bidimensional\n");

    for (renglon = 0; renglon < 4 ; renglon++){
        for (columna = 0; columna < 3 ; columna++){
            printf("%x\t",&nums[renglon][columna]);
        }
        printf("\n");
    }

    return 0;
}
```

Representación en memoria de un arreglo tridimensional.



Tarea 2

¿Cuál es la fórmula para acceder en la memoria a un elemento de un arreglo tridimensional que se encuentra en las coordenadas [renglón, columna, plano] de una matriz de MxNxL?

Crear un programa que genere una matriz tridimensional de MxNxL y que permita acceder a un elemento a través de sus coordenadas [renglón, columna, plano]. Las dimensiones y las coordenadas las ingresa el usuario. Dibujar la matriz resultante y el elemento seleccionado.

"Weeks of coding can save you hours of planning." - Unknown

@CodeWisdom

APLICACIONES DE ARREGLOS.

Práctica 1

© Jorge A. Solano

1. Aplicaciones de arreglos.

Crear una aplicación que permita manejar mínimo 5 elementos en un carrito de compra, siguiendo las siguientes especificaciones:

- La función principal sólo debe tener una llamada a una función menú. El menú debe contener las opciones: mostrar elementos de la tienda, mostrar carrito, agregar elemento al carrito, eliminar elemento del carrito y salir de la aplicación.
- Todas las opciones del menú deben estar programadas en funciones separadas (de preferencia en archivos diferentes).

© Jorge A. Solano

1.1.3 Apuntadores.

Un apuntador es una variable que contiene la dirección de memoria de una variable.

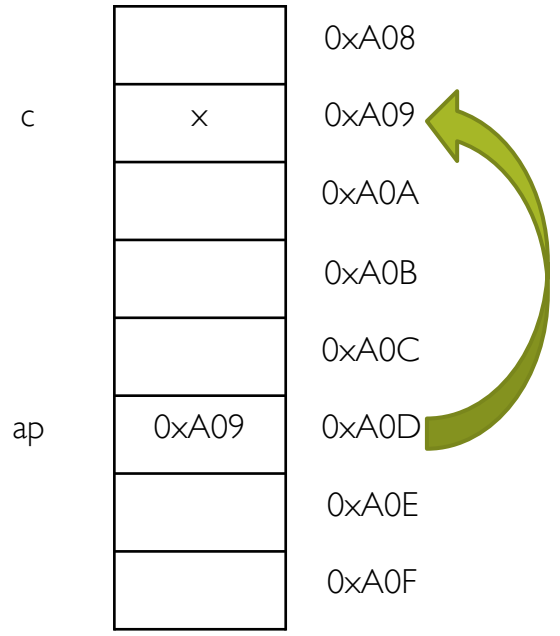
Los apuntadores se utilizan debido a que algunas veces son la única forma de expresar una operación, y debido a que, generalmente, llevan un código compacto y eficiente.

Los apuntadores también pueden emplearse para obtener claridad y simplicidad.

Ejemplo 22

Asignación de un apuntador a una variable (localidad de memoria).

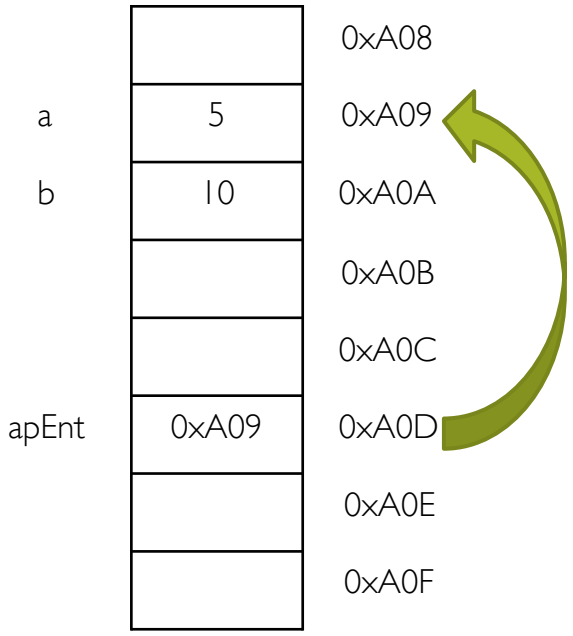
```
#include <stdio.h>

int main(){
    char c = 'x';
    char *ap;
    ap = &c;
    printf("&c = %x\n", &c);
    printf("ap -> %x\n", ap);
    printf("&ap = %x\n", &ap);
    return 0;
}
```



Ejemplo 23

Operaciones de un apuntador.

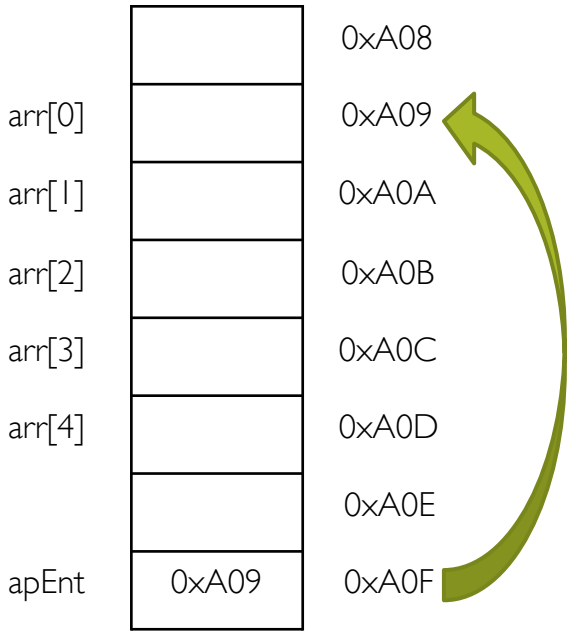
```
#include<stdio.h>

int main () {
    short a = 5, b = 10;
    short *apEnt;
    apEnt = &a;
    b = *apEnt;      // b = 5
    b = *apEnt+1;    // b = 6
    *apEnt = 0;      // a = 0
    return 0;
}
```



Ejemplo 24

Apuntador a arreglo.

```
#include<stdio.h>

int main () {
    short arr[5], *apArr;
    apArr = &arr[0];
    printf("%x\n",&arr);
    printf("%x\n",apArr);
    return 0;
}
```



Aritmética de direcciones en los apuntadores.

Si 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.

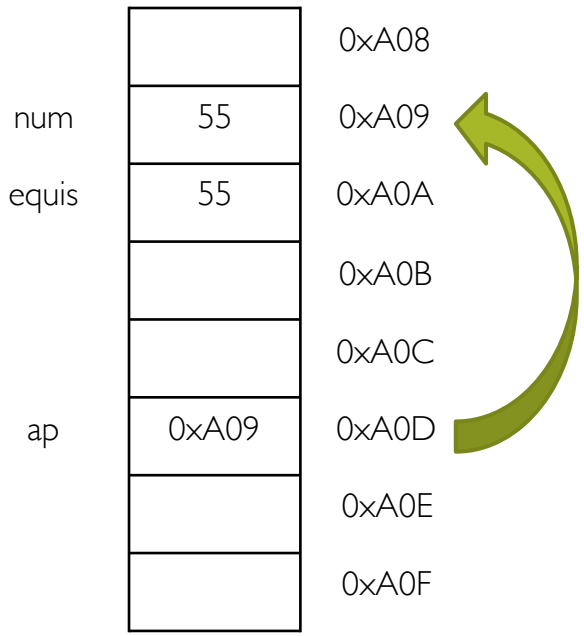
Parámetros de una función

En su declaración, una función define el tipo y el número de parámetros que recibe. Una función puede recibir variables por valor o por referencia según sean definidos.

Ejemplo 25

Cuando se reciben variables por valor se envía una copia del valor original, de tal manera que si se modifica el contenido de la variable, el valor original no se verá afectado.

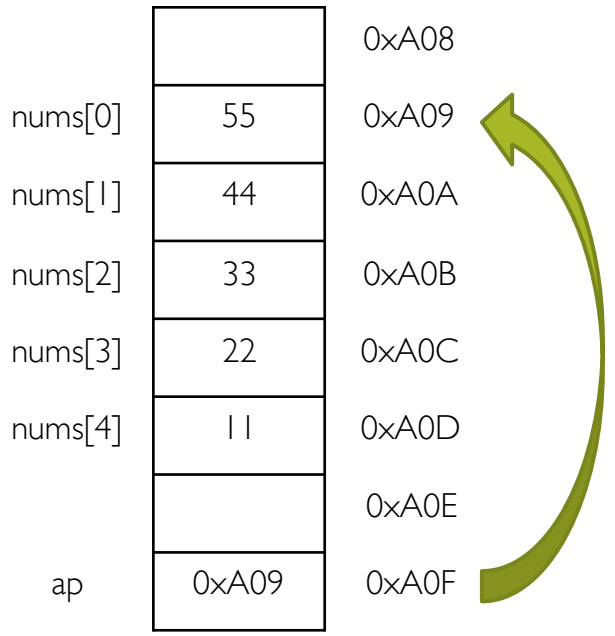
```
#include<stdio.h>

void pasarValor(int);

int main(){
    int num = 55, *ap;
    ap = &num;
    printf("Pasar valor: %d\n", *ap);
    pasarValor(*ap);
    printf("Valor final: %d\n", *ap);
    return 0;
}

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



Ejemplo 26

Cuando se reciben variables por referencia se envía un apuntador del valor original y, por ende, se está trabajando con dicho valor todo el tiempo.

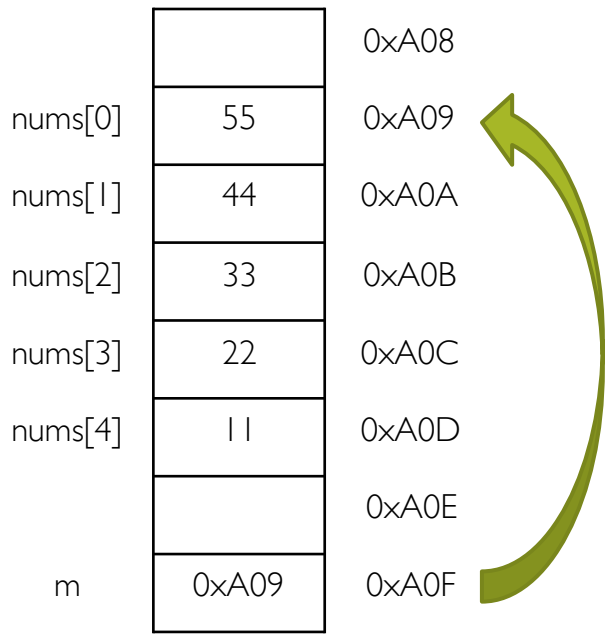
```
#include<stdio.h>

void pasarReferencia(int *);

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

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



Ejemplo 27

Si una función define como argumento un arreglo, en automático se puede considerar un paso por referencia.

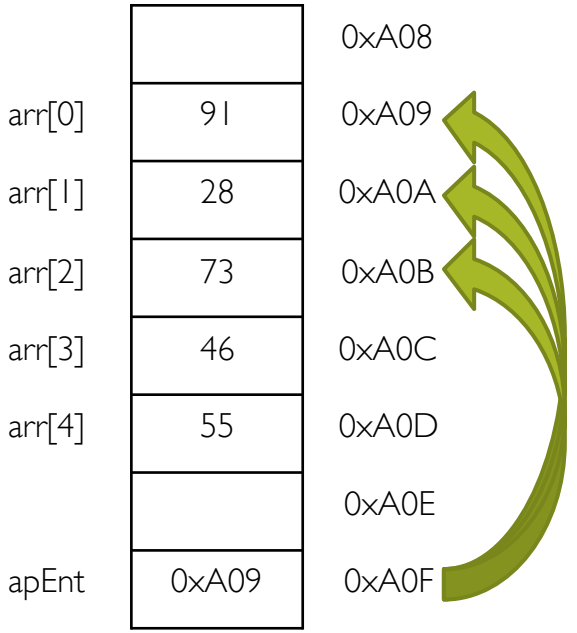
```
#include<stdio.h>

void pasarReferencia(int m[]);

int main(){
    int nums[] = {55,44,33,22,11}, cont;
    pasarReferencia(nums);
    for (cont=0 ; cont<5 ; cont++){
        printf("%i",nums[cont]);
    }
    return 0;
}

void pasarReferencia(int m[]){
    int cont;
    for (cont=0 ; cont<5 ; cont++){
        m[cont]=cont;
    }
}
```



Ejemplo 28

Aritmética de direcciones en un apuntador.

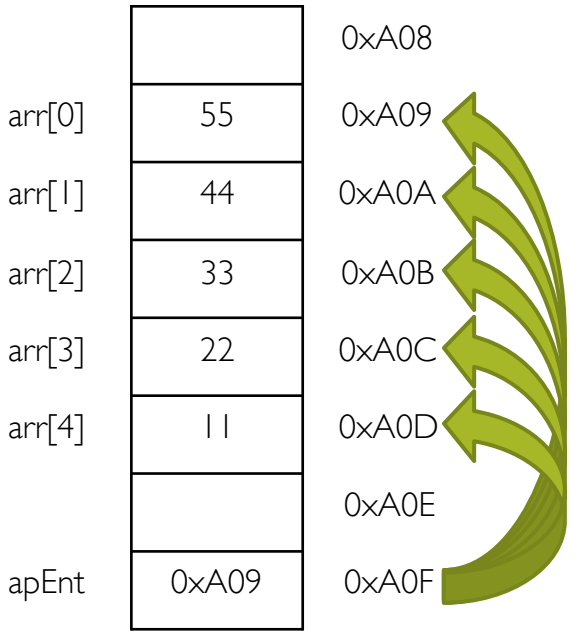
```
#include<stdio.h>

int main () {
    short arr[5] = {91,28,73,46,55};
    short *apArr;
    apArr = arr;
    printf("%i\n",*apArr);
    printf("%i\n",*(apArr+1));
    printf("%i\n",*(apArr+2));
    return 0;
}
```



Ejemplo 29

Almacenamiento en memoria de arreglo unidimensional.

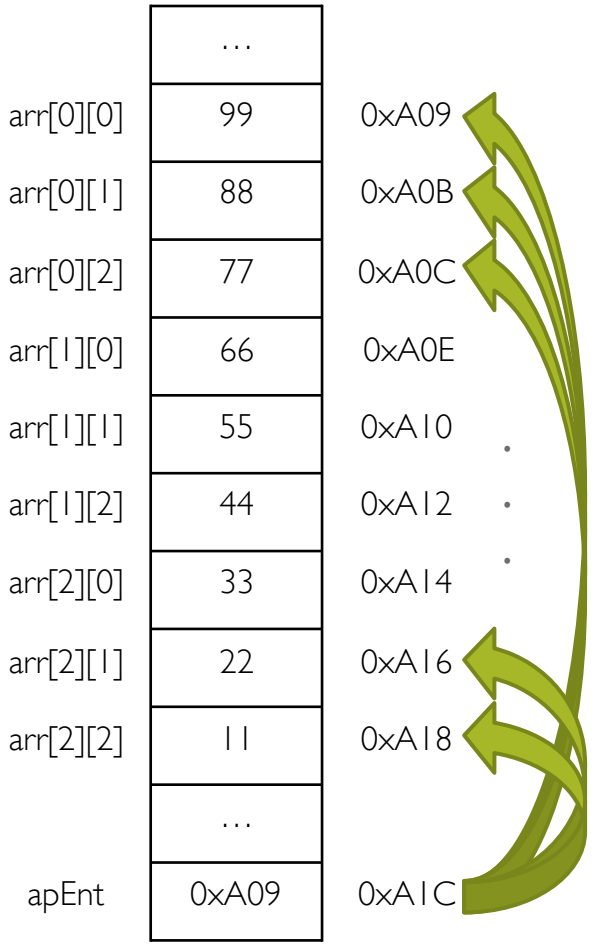
```
#include<stdio.h>

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;
}
```



Ejemplo 30

Almacenamiento en memoria de arreglo bidimensional.

```
#include<stdio.h>

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;
}
```



Ejemplo 31

Apuntador a void

```
#include <stdio.h>

void ver(void *, int);

int main() {
    char a='b';
    int x=3;
    double y=4.5;
    char *cad="hola";
    ver(&a, 0);
    ver(&x, 2);
    ver(&y, 1);
    ver(cad, 3);
    getchar();
    return 0;
}
```

```
void ver(void *p, int d) {
    switch(d) {
        case 0:
            printf("%c\n",*(char *)p);
            break;
        case 1:
            printf("%d\n",*(double *)p);
            break;
        case 2:
            printf("%ld\n",*(int *)p);
            break;
        case 3:
            printf("%s\n",(char *)p);
            break;
        default:
            printf("Error ");
    }
}
```

Ejemplo 32

Apuntador de apuntadores (arreglo de apuntadores).

```
#include <stdio.h>

char *nombre_mes(int);

int main(void){
    char *mes = NULL;
    int n; puts("Introduzca el número del mes deseado");
    if (scanf("%d", &n) != 1){
        printf("Carácter inválido\n");
        return;
    }
    mes = nombre_mes(n);
    printf("El mes seleccionado es:\n%s\n", mes);
    return 0;
}
```

Ejemplo 32

Apuntador de apuntadores (arreglo de apuntadores).

```
char *nombre_mes(int numero){
    static char *mes[] = {
        "Mes invAlido",
        "Enero", "Febrero", "Marzo",
        "Abril", "Mayo", "Junio",
        "Julio", "Agosto", "Septiembre",
        "Octubre", "Noviembre", "Diciembre" };

    return (numero < 1 || numero > 12) ? mes[0] : mes[numero];
}
```

Apuntador a función

Un apuntador a función es una función que apunta a otra función. La firma para declarar un apuntador a función es la siguiente:

```
TipoDatoRetorno (*apFunc)([parámetros]);
```

Para que una función pueda apuntar a la dirección de memoria de otra función es necesario tener la declaración de la definición de la función, es decir:

```
int funcion(int);
```

La asignación de una función apuntador se realiza de la siguiente manera:

```
apFunc = &funcion;
```

Ejemplo 33

Apuntador a función

```
#include <stdio.h>

int suma(int,int);

int main(){
    int (*apSuma)(int,int)=&suma;
    printf("Dirección memoria suma: %x\n",&suma);
    printf("Dirección memoria apSuma: %x\n",&apSuma);
    printf("apSuma apunta a: %x\n",*apSuma);
    printf("apSuma(6,4) -> %d\n",apSuma(6,4));
    printf("suma(6,4) -> %d\n",suma(6,4));
}

int suma(int a,int b) {
    return a+b;
}
```

"Code is like a humor. When you have to explain it, it's bad."

Cory House
(Author @pluralsight, Speaker, Software Architect,
@microsoft MVP, @JSJabber panelist #JavaScript)

© Jorge A. Solano

APLICACIONES DE APUNTADORES.

Práctica 2

© Jorge A. Solano

2. Aplicaciones de apuntadores.

Crear una aplicación que permita manejar mínimo 5 elementos en un carrito de compra, siguiendo las siguientes especificaciones:

- La función principal sólo debe tener una llamada a una función menú. El menú debe contener las opciones: mostrar elementos de la tienda, mostrar carrito, agregar elemento al carrito, eliminar elemento del carrito y salir de la aplicación.
- Todas las opciones del menú deben estar programadas en funciones separadas y en archivos diferentes.
- Dentro del código todos los accesos a arreglos se deben realizar utilizando apuntadores (el código no debe tener ningún paréntesis cuadrado).

© Jorge A. Solano

1.1.4 Tipo de dato abstracto.

Las estructuras de datos están compuestas por nodos, objetos o elementos, estos nodos son un tipo dato abstracto, es decir, puede contener cualquier cantidad y tipo de información:

- Información de una película (Netflix).
- Información de un video (Youtube).
- Información de una pista (iTunes).
- Información de un documento a imprimir.

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 estructuras (struct).

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 por variables de diferente tipo, que pueden ser tratadas como una unidad.

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.

Los datos que pueden contener una estructura son:

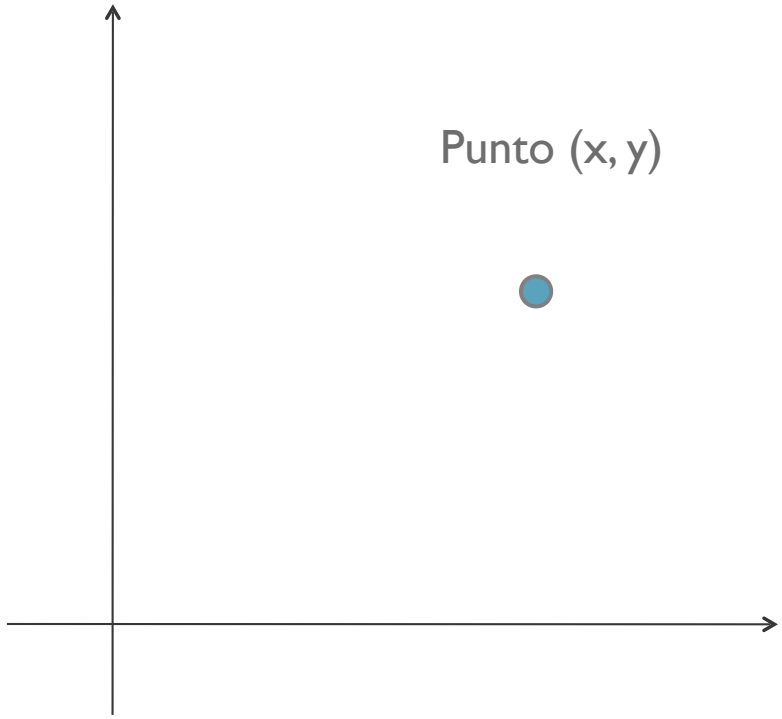
Simples:

- **Caracteres**
- **Números**
- **Enteros**
- **Punto flotante**

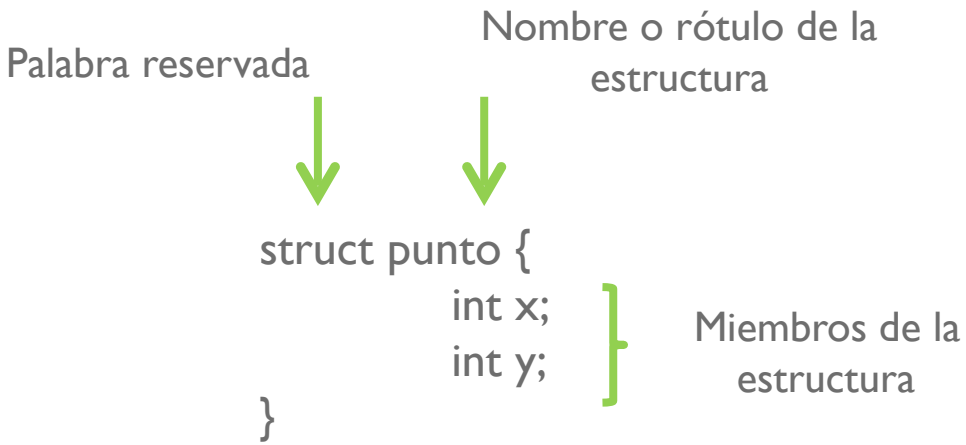
Compuestos:

- **Arreglos**
- **Estructuras**

Modelar un punto en el espacio.



int x = a;
Int y = b;




```
struct punto p = { 5, 3 };
```

```
printf("x = %d, y = %d", p.x, p.y);
```

Declaraciones válidas de una estructura en lenguaje C:

```
struct {
          int x;
          int y;
} punto;
```

```
struct {
          int x, y;
} punto;
```

```
struct { int x, y; } punto;
```

```
#include<stdio.h>

struct punto {
          int x, y;
};

void main() {
          struct punto p = {5, 3};
          printf("x = %d, y = %d", p.x, p.y);
}
```

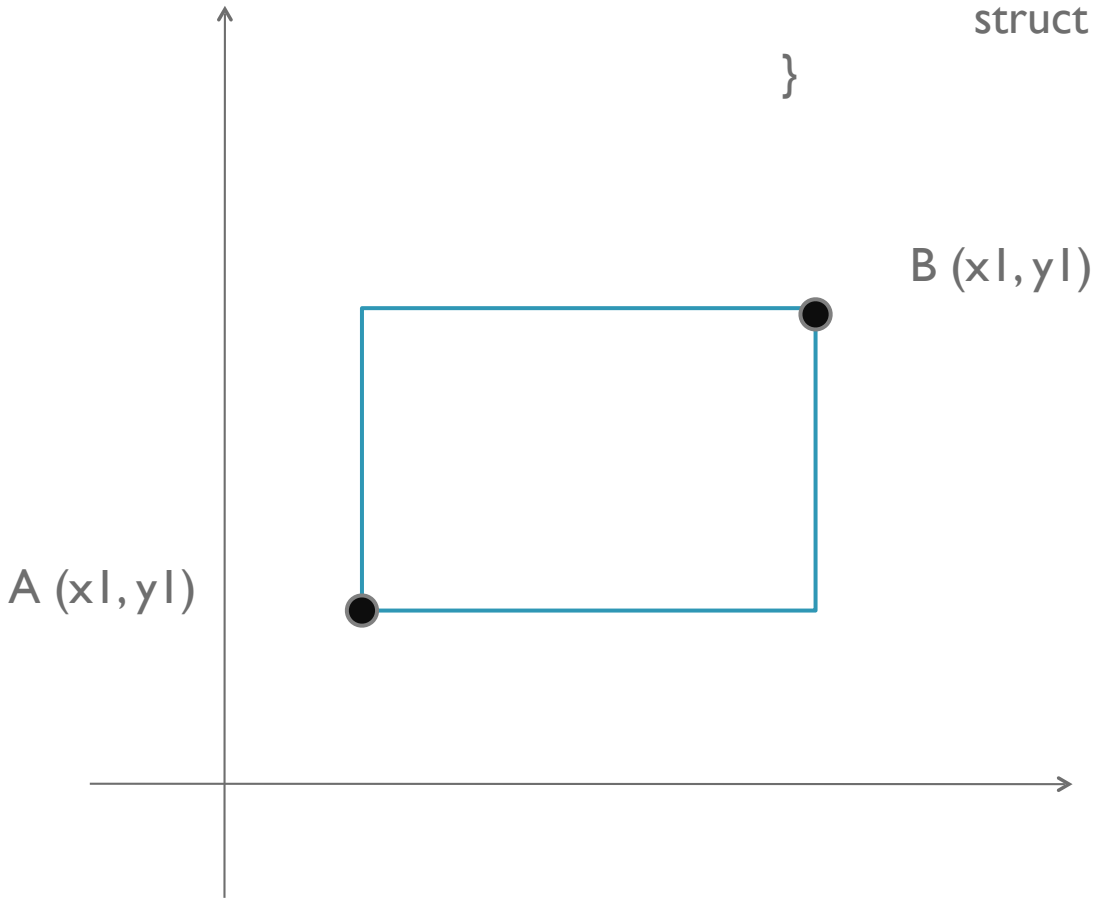
```
#include<stdio.h>

struct alumno{
          int num_cuenta;
          char nombre [50];
};

void main() {
          struct alumno a;
          printf("Introduce tu nombre: ");
          scanf("%s",  a.nombre);
          printf("Introduce tu número de cuenta: ");
          scanf("%d",  &a.num_cuenta);
          printf("Alumno: %s\n Número de cuenta: %d\n",
                    a.nombre, a.num_cuenta);
}
```

```
#include<stdio.h>

struct alumno{
          int num_cuenta;
          char nombre [50];
};
struct alumno a;

void main() {
          printf("Introduce tu nombre: ");
          scanf("%s",  a.nombre);
          printf("Introduce tu número de cuenta: ");
          scanf("%d",  &a.num_cuenta);
          printf("Alumno: %s\n Número de cuenta: %d\n",
                    a.nombre, a.num_cuenta);
}
```

```
#include<stdio.h>

struct alumno{
          int num_cuenta;
          char nombre [50];
} a;

void main() {
          printf("Introduce tu nombre: ");
          scanf("%s",  a.nombre);
          printf("Introduce tu número de cuenta: ");
          scanf("%d",  &a.num_cuenta);
          printf("Alumno: %s\n Número de cuenta: %d\n",
                    a.nombre, a.num_cuenta);
}
```

Estructuras anidadas

```
struct rectangulo {
        struct punto a;
        struct punto b;
}
```



```
#include<stdio.h>

struct punto {
          int x, y;
};
struct  rectangulo {
          struct punto a, b;
};
int main() {
          struct rectangulo r;
          struct punto uno = {5,3}, dos = {8,6};
          r.a = uno;
          r.b = dos;
          printf("Punto a: x = %d, y = %d\n", r.a.x, r.a.y);
          printf("Punto b: x = %d, y = %d\n", r.b.x, r.b.y);
          return 4;
}
```

Una función puede recibir como parámetros estructuras de datos.

```
int ptoEnRect (struct punto a, struct rectangulo y) {
        if ((y.a.x < a.x && a.x < y.b.x) &&
                (y.a.y < a.y && a.y < y.b.y) {
                return 1;
        } else {
                return 0;
        }
}
```

Una función puede regresar como valor de retorno una estructura de datos.

```
struct punto crearPunto (int x, int y) {
        struct punto temporal;

        temporal.x = x;
        temporal.y = y;

        return temporal;
}
```

Una función también puede recibir como parámetros y regresar como valor de retorno estructuras de datos.

```
struct punto sumarPuntos (struct punto a, struct punto b) {
        a.x += b.x;
        a.y += b.y;
        return a;
}
```

Cuando se van a pasar estructuras de datos grandes hacia una función, es mejor realizarlo mediante apuntadores.

```
void main() {
        struct rectangulo r, *apRect;
        apRect = &r;
        printf("Punto a = (%d, %d)",
                (*apRect).a.x, (*apRect).a.y);
}
```

```
printf("Punto a = (%d, %d)", (*apRect)->a.x, (*apRect)->a.y);
```

Modelar un punto en el espacio.

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

Ejemplo 34

Estructura nodo

```
#include<stdio.h>

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

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

Ejemplo 34

Estructura nodo

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

Ejemplo 34

Estructura nodo

```
struct nodo llenarDatosPelicula(char *nombre,
          char *genero, short anio,
          short numDirectores, char *directores[10]){
    struct nodo 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;
}
```

Ejemplo 34

Estructura nodo

```
void imprimirDatosPelicula(struct nodo 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]);
    }
}
```

"Just because you've implemented something doesn't mean you understand it."

Brian Cantwell Smith
(A scholar in the fields of cognitive science, computer science, information studies, and philosophy, especially ontology.)

© Jorge A. Solano

TIPO DE DATO ABSTRACTO.

Práctica 3

© Jorge A. Solano

3. Tipo de dato abstracto.

Crear una aplicación que permita manejar mínimo 5 elementos (nodos) en un carrito de compra, siguiendo las siguientes especificaciones:

- La función principal sólo debe tener una llamada a una función menú. El menú debe contener las opciones: mostrar elementos de la tienda, mostrar carrito, agregar elemento al carrito, eliminar elemento del carrito y salir de la aplicación.
- Todas las opciones del menú deben estar programadas en funciones separadas y en archivos diferentes.
- Dentro del código se debe tener un arreglo de nodos que maneje la información de cada artículo (nombre, precio, cantidad de productos en almacen, cantidad de productos en el carrito).

© Jorge A. Solano

1.2 Administración del almacenamiento en tiempo de ejecución.

1.2 Administración del almacenamiento en tiempo de ejecución

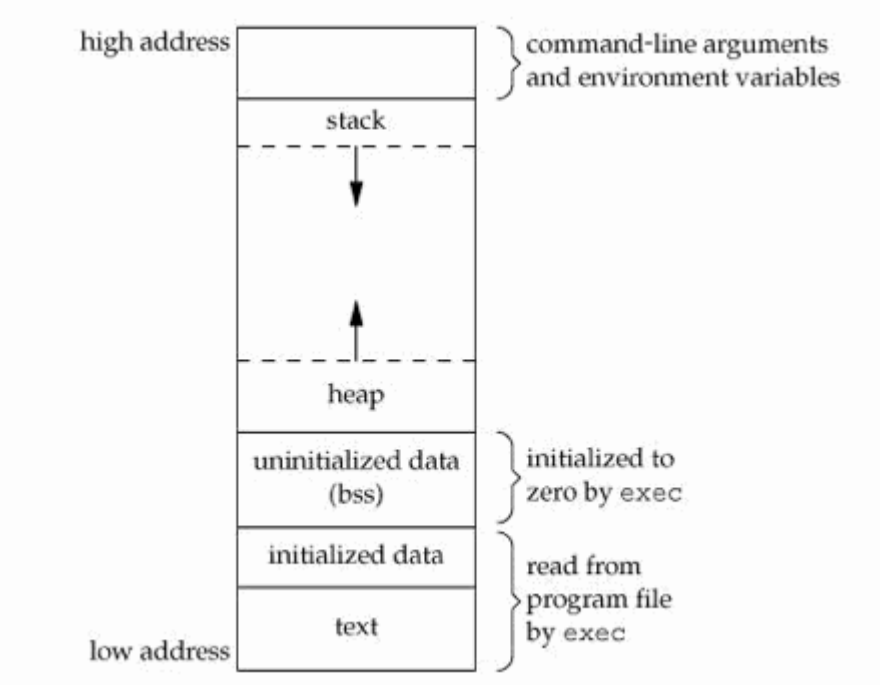
La memoria dinámica se refiere al espacio de almacenamiento que se reserva en tiempo de ejecución, debido a que su tamaño puede variar durante la ejecución del programa.

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.

Dentro de la memoria RAM, la memoria reservada de forma dinámica está alojada en el heap o almacenamiento libre y la memoria estática (arreglos) 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.

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).



El segmento de texto es el segmento de memoria donde el código de la aplicación existe mientras se encuentra en ejecución. Mientras la aplicación se está ejecutando las funciones, procedimientos y métodos se encuentra en esa sección de la memoria. Este segmento es controlado por el Sistema Operativo, pero es importante saber que existe.

Cuando la 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.

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 (LIFO).

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.

Función malloc.

En lenguaje C, la función malloc reserva 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).

Función free.

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.

Ejemplo 35

Memoria dinámica: 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;
}
```

Función calloc.

La función calloc funciona de manera similar a la función malloc pero, además de reservar memoria, inicializa la memoria reservada con 0, por tanto, es utilizada comúnmente para arreglos unidimensionales y multidimensionales. 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.

Ejemplo 36

Memoria dinámica: 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;
}
```

Función 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).

Ejemplo 37

Memoria dinámica: REALLOC

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

int main (){
    int *arreglo, 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");
```

Ejemplo 37

Memoria dinámica: REALLOC

```
        printf("Aumentando el tamaño del conjunto al doble.\n");
        num *= 2;
        int *arreglo2 = (int *)realloc (arreglo,num*sizeof(int));
        if (arreglo2 != NULL) {
            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 (arreglo2);
        }
        free (arreglo);
    }
    return 0;
}
```

Tratar de utilizar un apuntador cuyo bloque de memoria ha sido liberado con la función free() puede ser peligroso, el comportamiento del programa queda indefinido: puede terminar de forma inesperada, sobrescribir otros datos y provocar problemas de seguridad.

Para evitar problemas con apuntadores utilizando memoria dinámica se recomienda que se establezca su valor directamente a NULL.

"Programming is not about typing, it's about thinking."

Rich Hickey
(Rich Hickey is the creator of the Clojure language.
Clojure is a dialect of the Lisp programming language.)

ALMACENAMIENTO EN TIEMPO DE EJECUCIÓN.

Práctica 4

© Jorge A. Solano

4. Almacenamiento en tiempo de ejecución.

Crear una aplicación que permita manejar un carrito de compras electrónico con las siguientes especificaciones:

- La tienda debe tener un mínimo de 10 artículos.
- Cada uno de los artículos se deben almacenar como una sola entidad (nodo).
- Todo se programa en funciones y en archivos distintos. El método principal solo debe contener una llamada a la función menú que permita ver los artículos de la tienda, agregar un artículo al carrito de compra, ver los artículos del carrito de compra y generar la cuenta del carrito de compra.
- El carrito de compra debe ser de tamaño variable, es decir, se reserva espacio sobre demanda y en tiempo de ejecución. El recorrido de los artículos del carrito de compra se debe realizar accediendo directamente a la memoria, a través de apuntadores.

Kernel de linux.

```
#include <linux/module.h>     /* Needed by all modules */
#include <linux/kernel.h>     /* Needed for KERN_INFO */
#include <linux/jiffies.h>

int init_module(void) {
          int n = 5;
          printk(KERN_INFO "Hello world 1.\n");
          unsigned long j, stamp_1, stamp_half,stamp_n;
          j = jiffies;

          stamp_1 = j + HZ;
          stamp_half= j + HZ/2;
          stamp_n = j + n * HZ / 1000;

          printk(KERN_INFO "Currenttime in jiffies=%lu \n",j);
          printk(KERN_INFO "Currenttime in ms = %d \n", jiffies_to_msecs(j));
          printk(KERN_INFO "1 second in the future= %d \n",
                                        jiffies_to_msecs(stamp_1));
          printk(KERN_INFO "half a second= %d \n", jiffies_to_msecs(stamp_half));
          printk(KERN_INFO "n milliseconds = %d \n", jiffies_to_msecs(stamp_n));

          return 0;
}
```

1.3 Estructura de datos compuestos.

1.3 Estructura de datos compuestos

Los conjuntos son tan fundamentales para las ciencias de la computación como lo son para las matemáticas.

Los conjuntos manipulados por algoritmos pueden crecer, reducirse o cambiar en el tiempo, por lo tanto, se les denomina conjuntos dinámicos.

Dentro de los conjuntos dinámicos se pueden realizar diversas operaciones que se pueden agrupar en dos categorías: consultas y modificaciones.

Las consultas simplemente regresan información del conjunto. Las modificaciones pueden cambiar los elementos del conjunto.

La siguiente es una lista de operaciones típicas dentro de un conjunto de elementos:

- buscar (S, k): Dado un conjunto S, la consulta regresa el elemento x que coincida con conj[x] = k o nulo si no se encuentra coincidencia.

- insertar (S, x): es una operación de modificación que incrementa el conjunto S con el elemento x.

- eliminar (S, x): es una operación de modificación tal que dado un elemento x del conjunto S, se quita el elemento x del conjunto S.

- mínimo (S): es una consulta dentro de un conjunto ordenado de elementos S que devuelve el elemento menor.

- máximo (S): es una consulta dentro de un conjunto ordenado de elementos S que devuelve el elemento mayor.

- sucesor (S, x): es una consulta que busca un elemento x dentro de un conjunto ordenado S y devuelve el elemento mayor a x o nulo si el elemento x es el mayor.

- predecesor (S, x): es una consulta que busca un elemento x dentro de un conjunto ordenado S y devuelve el elemento menor a x o nulo si el elemento x es el menor.

El tiempo que toma ejecutar una operación en un conjunto S es proporcional al tamaño de entrada del conjunto (instancias de entrada).

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 que se van a analizar en este curso se denominan estructuras lineales debido a que los elementos ocupan lugares sucesivos en la estructura y cada uno de ellos tiene un único sucesor y un único predecesor.

1.3.1 Pila: almacenamiento contiguo y ligado, y operaciones.

La pila (o stack) es una estructura de datos lineal y dinámica, en la cual el elemento obtenido a través de la operación ELIMINAR (POP) está predefinido.

La pila implementa la política last-in, first-out (LIFO), esto es, el último elemento que se agregó es el primer que se elimina.

Las operaciones que se pueden realizar sobre una pila son INSERTAR (que es llamada PUSH) y ELIMINAR (que es llamada POP). Estos nombres hacen alusión a los apilamientos físicos, por ejemplo, de libros en un escritorio o platos en un restaurante, 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).

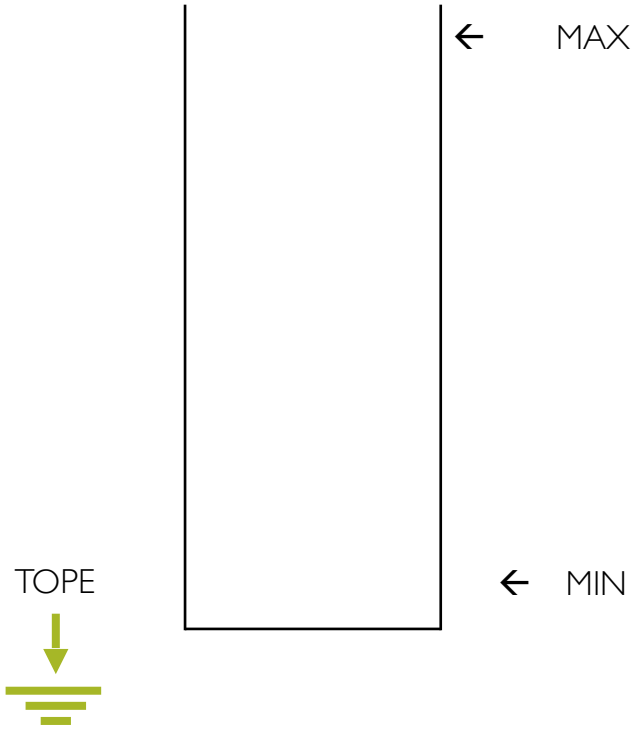
En la pila el único elemento accesible es el que está hasta arriba (tope).

La pila es una estructura de datos lineal ya que cada nodo de la pila tiene un único predecesor y un único sucesor.

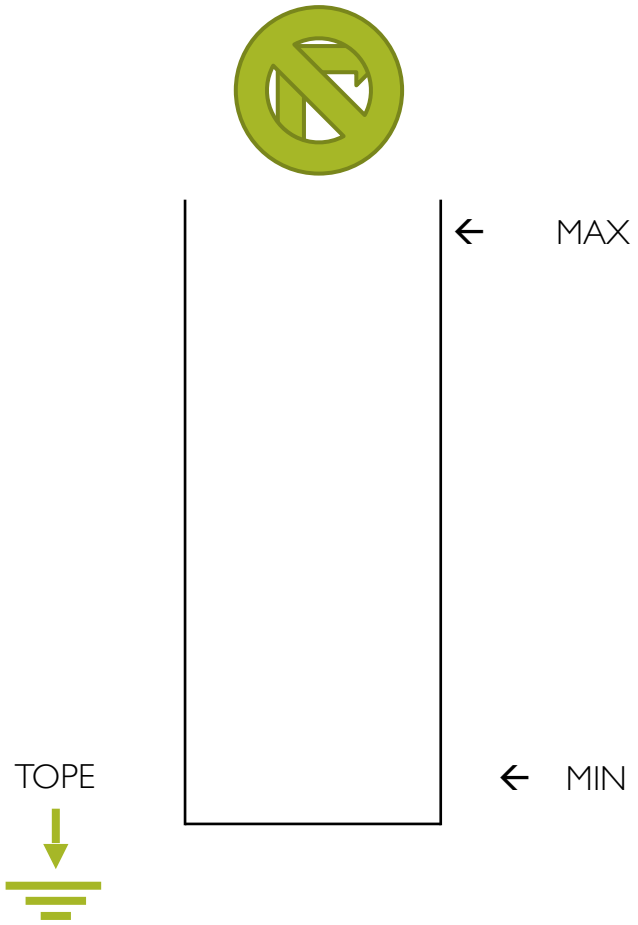
Para poder diseñar un programa que defina el comportamiento de una PILA se deben considerar 3 casos para ambas operaciones (push y pop):

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

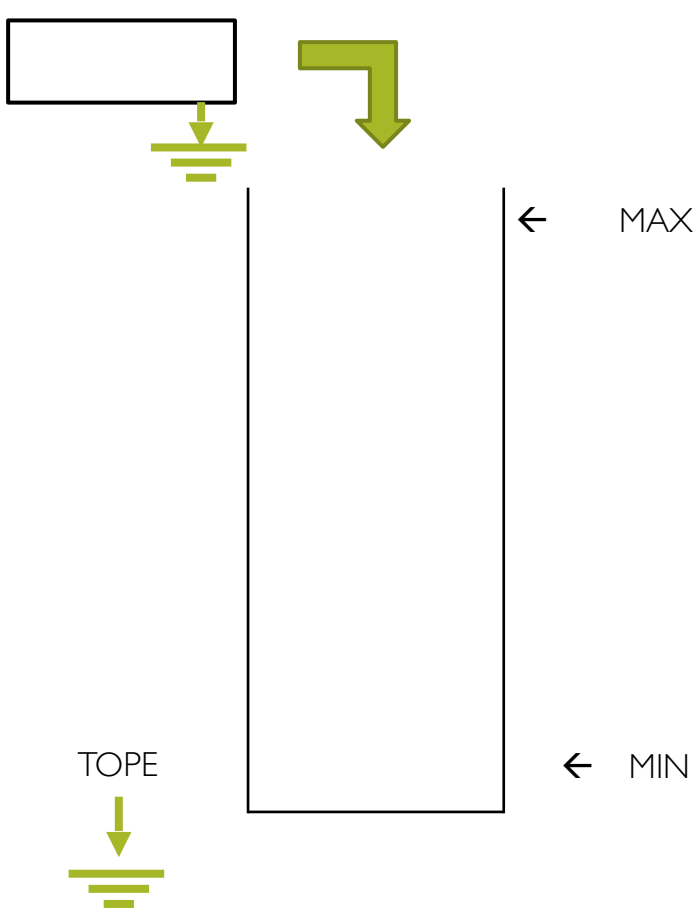
Pila vacía.



Pila vacía.

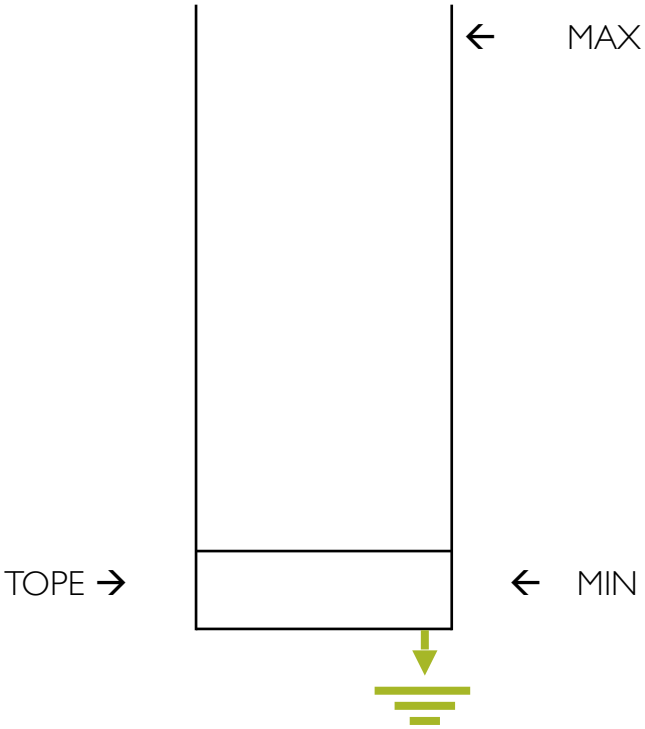


Pila vacía.

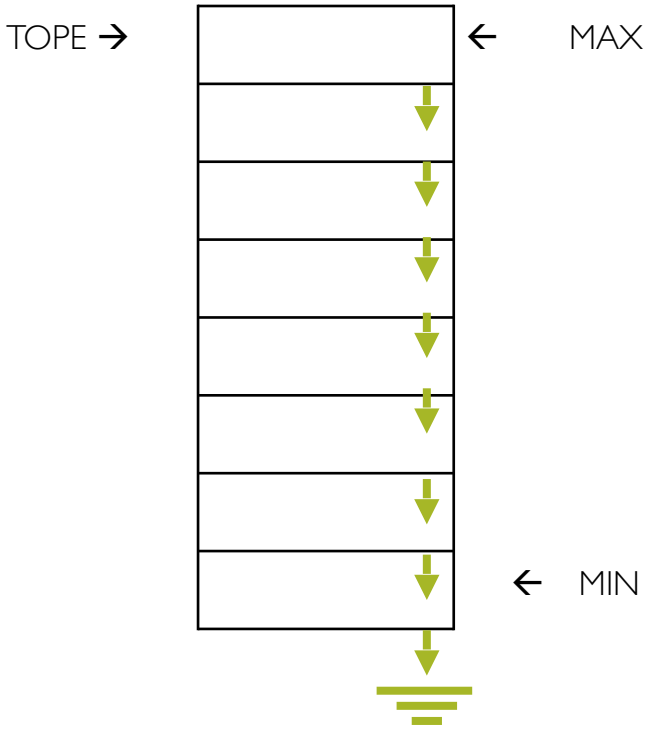


Pila vacía.

PUSH

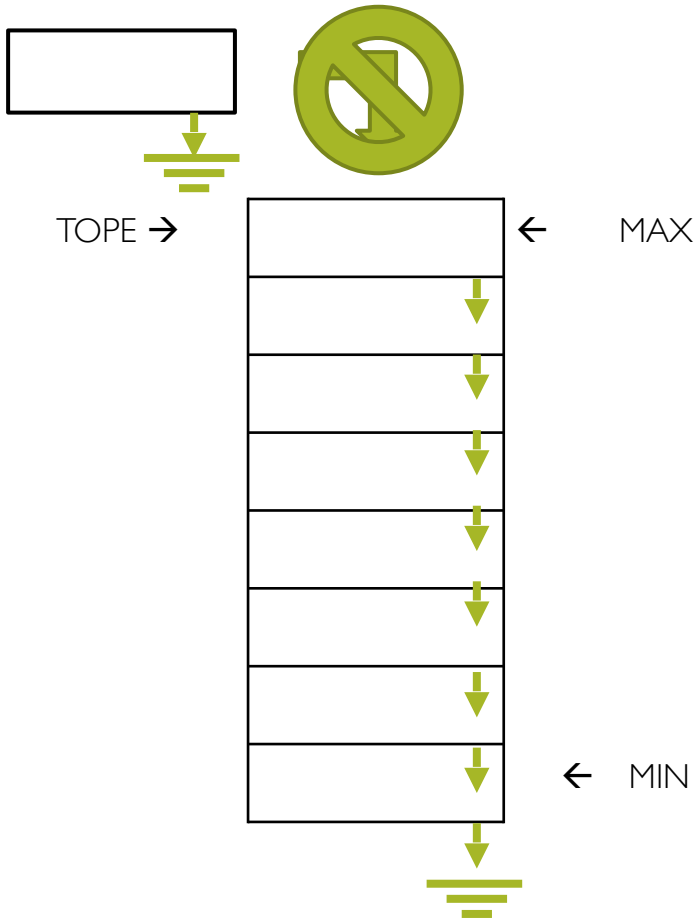


Pila llena.



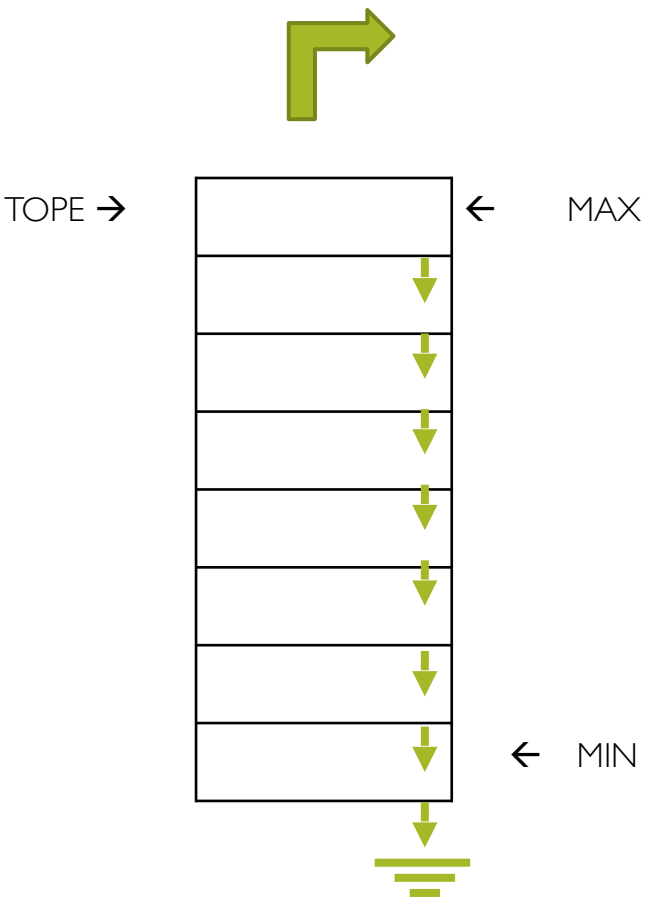
Pila llena.

PUSH

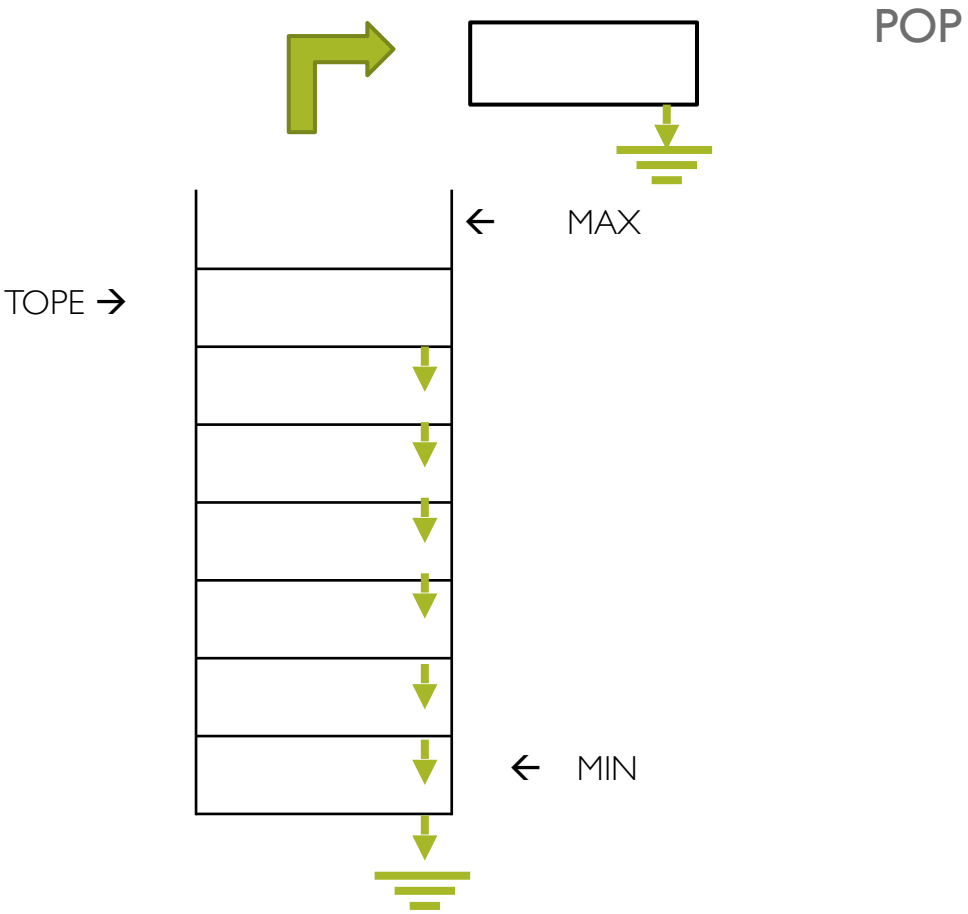


Pila llena.

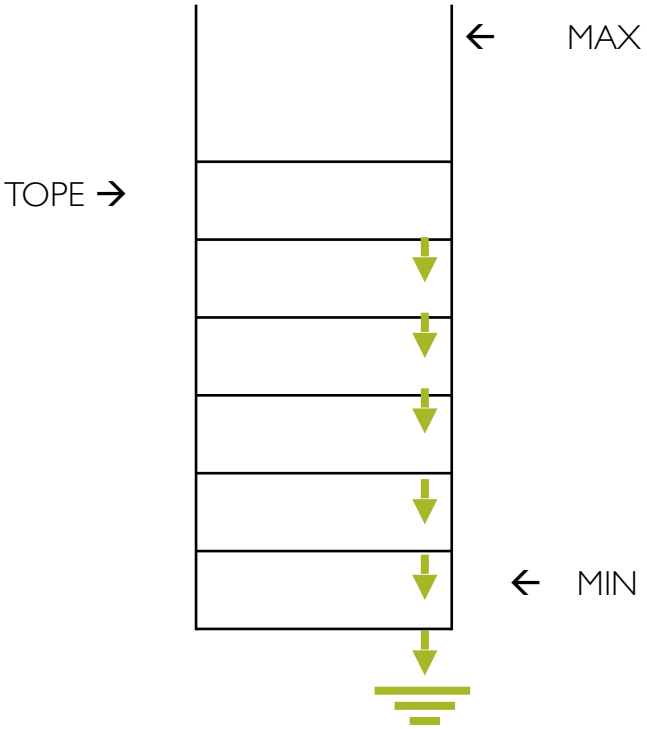
POP



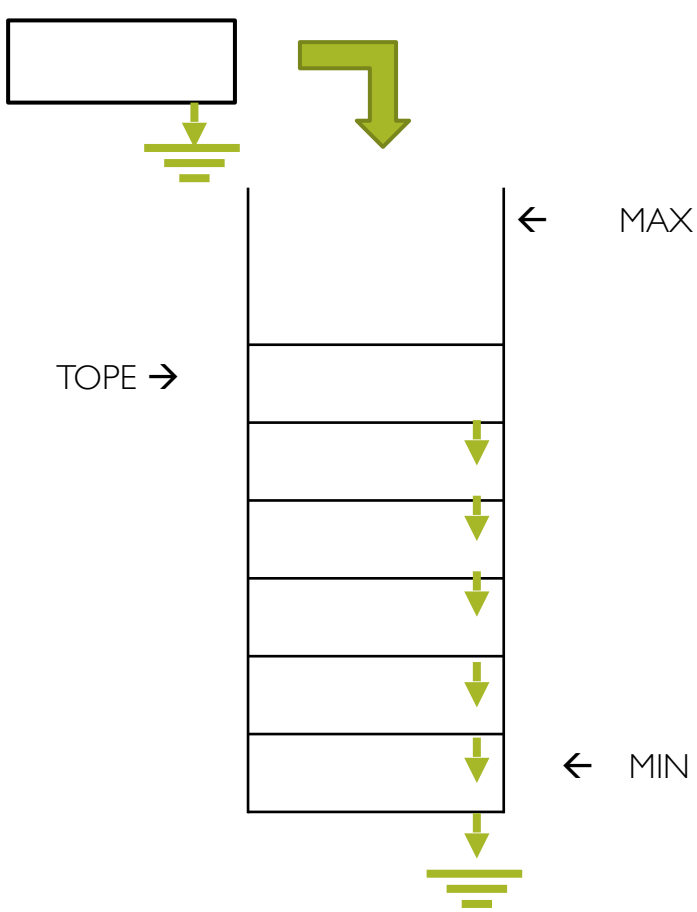
Pila llena.



Pila con elementos.

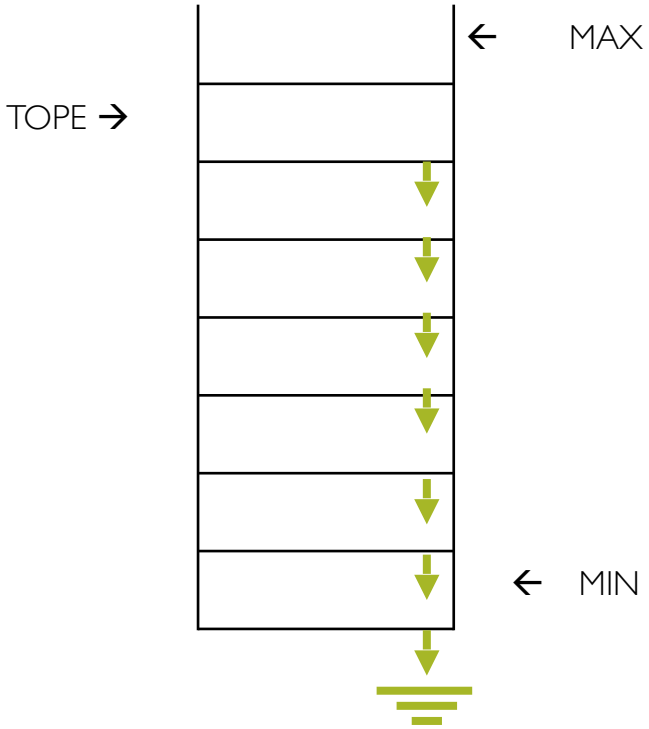


Pila con elementos.



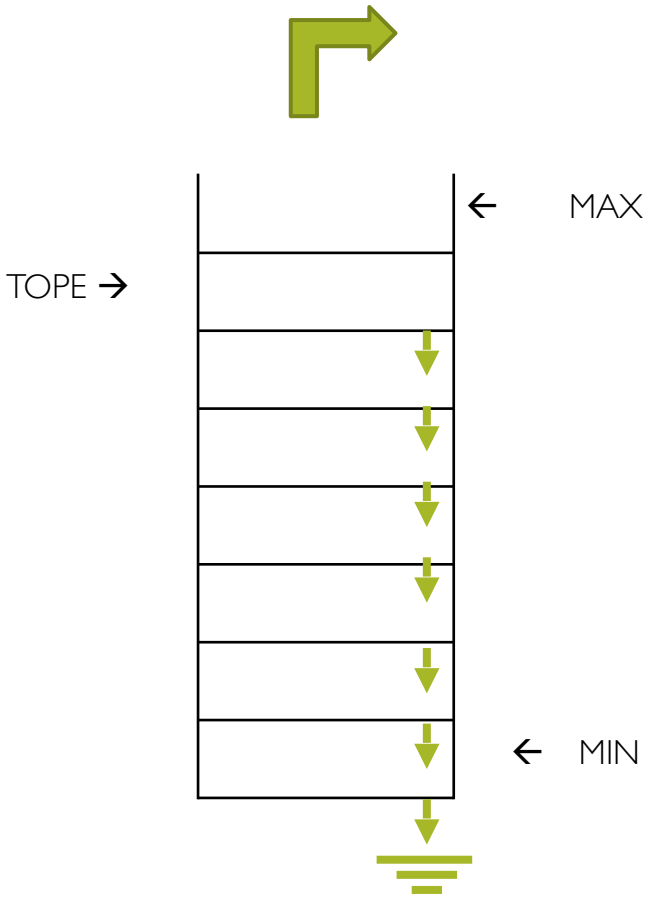
Pila con elementos.

PUSH

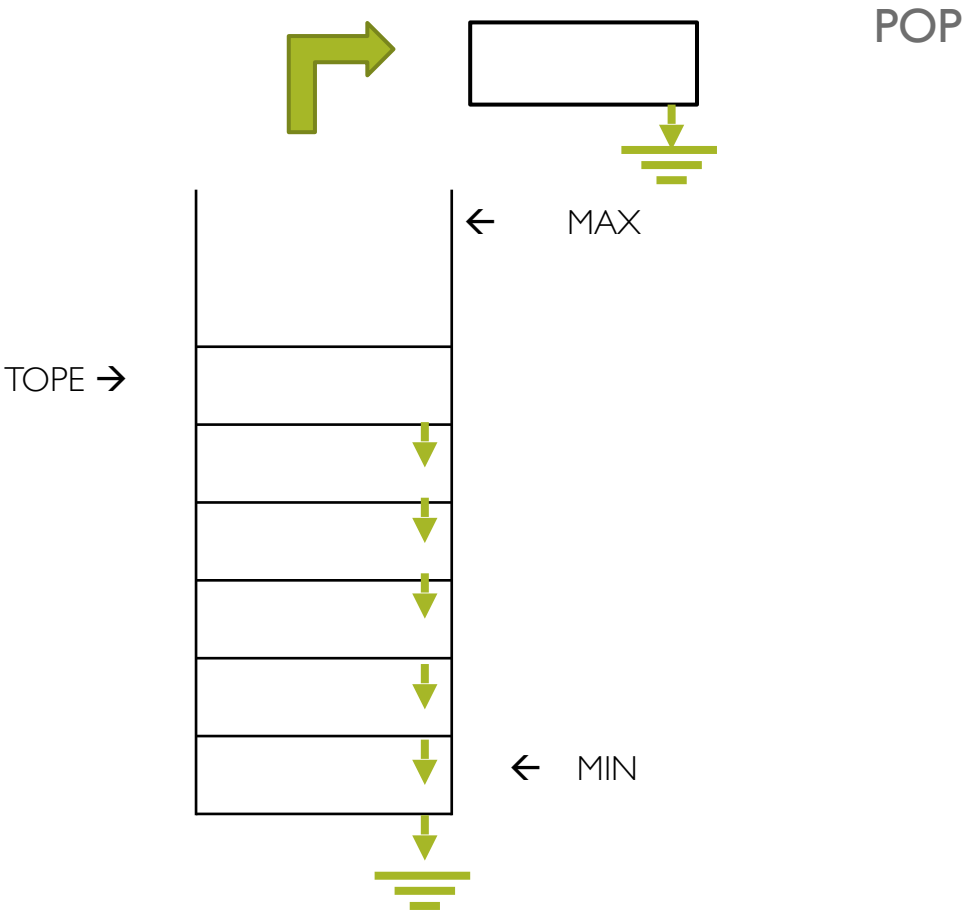


Pila con elementos.

POP



Pila con elementos.



¿Aplicaciones?

Figura 2. Pirámides de Teotihuacán.

Implementar la estructura de datos PILA en lenguaje C. La PILA debe permitir manipular nodos con las operaciones básicas PUSH y POP.

Implementar una función MOSTRAR para ver los elementos de la estructura de datos PILA.

"What is programming?... Some people call it a science, some people call it an art, some people call it a skil or trade."

Charles Simonyi
(Is a Hungarian-born American computer businessman.)

© Jorge A. Solano

1.3.2 Cola: almacenamiento contiguo y ligado, y operaciones.

La cola (o queue) es una estructura de datos lineal y dinámica, en la cual el elemento obtenido a través de la operación ELIMINAR (DEQUEUE) está predefinido.

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.

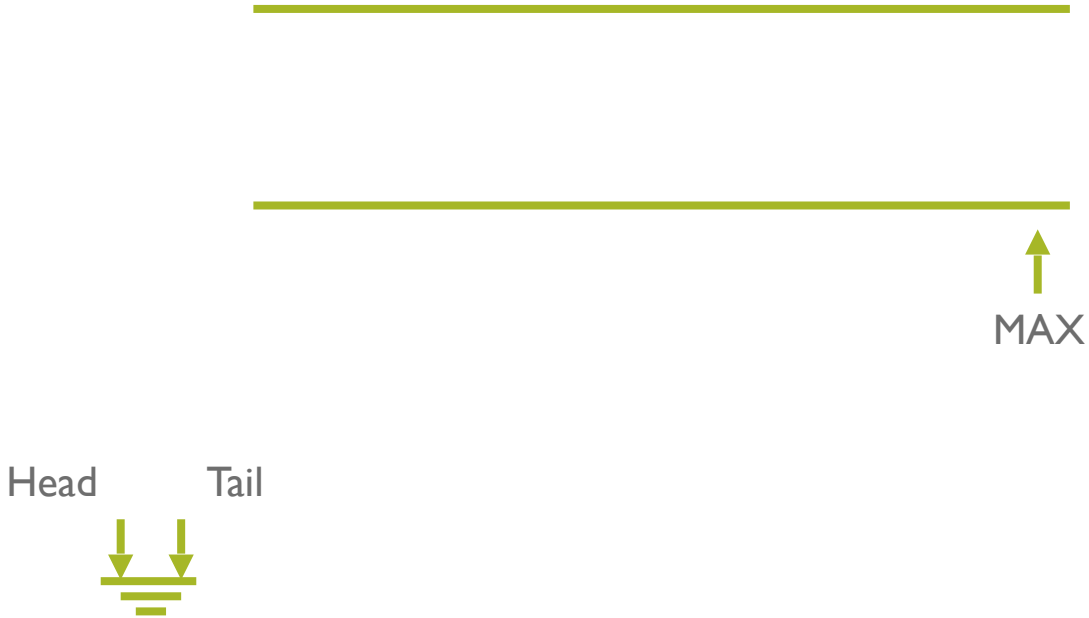
La operación de INSERTAR también se le llama ENCOLAR y la operación de ELIMINAR también se le llama DESENCOLAR.

La cola posee dos referencias, una al inicio (head) y otra al final (tail) de la estructura. Cuando un elemento es encolado, éste toma el lugar al final de la cola (tail). El elemento a desencolar siempre es el que está al inicio de la cola (head).

Para poder diseñar un programa 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.



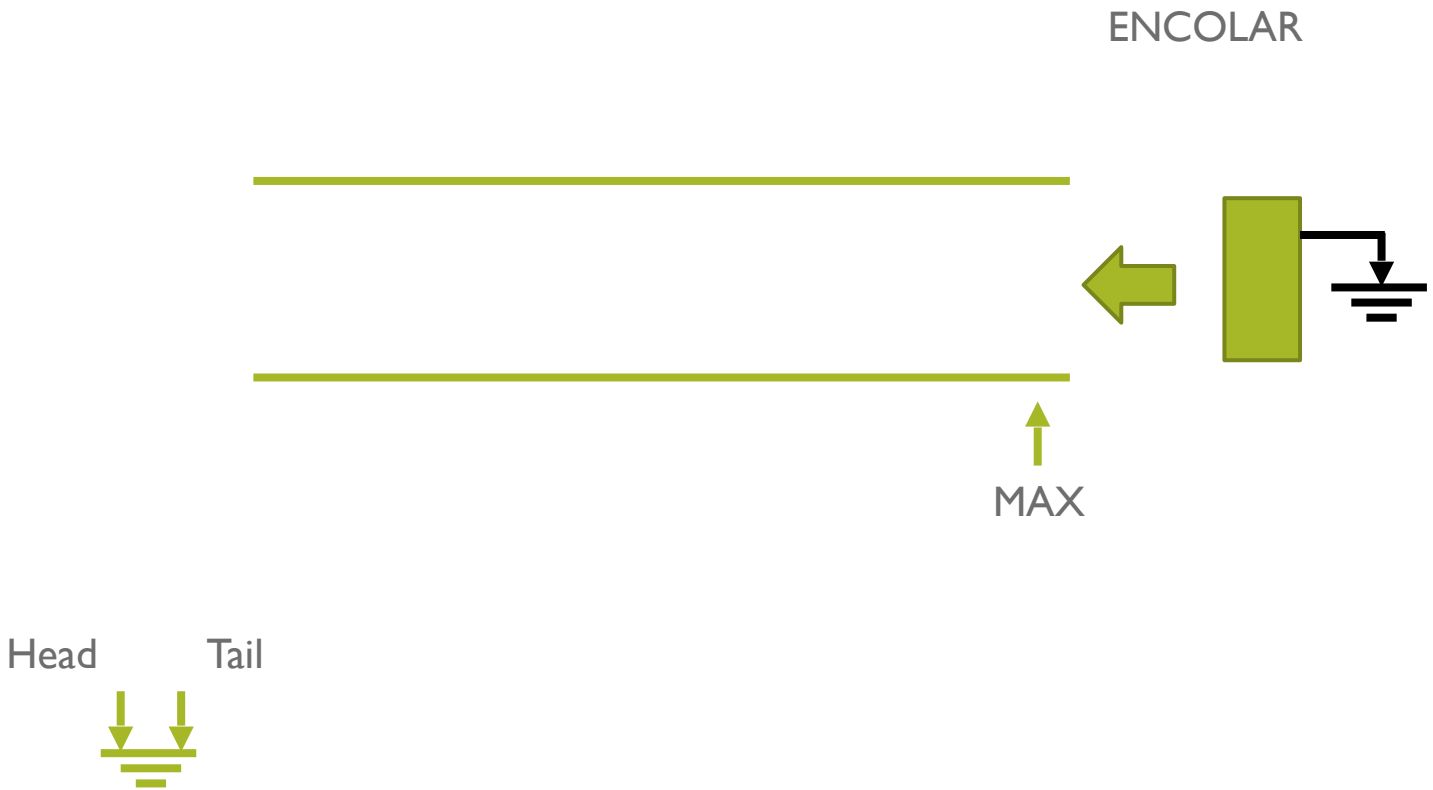
Cola vacía.

DESENCOLAR

MAX

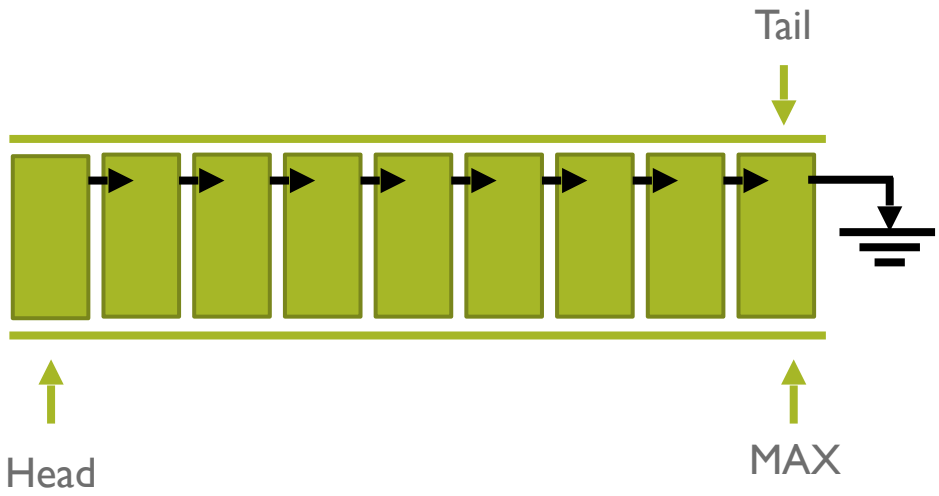
Head Tail

Cola vacía.

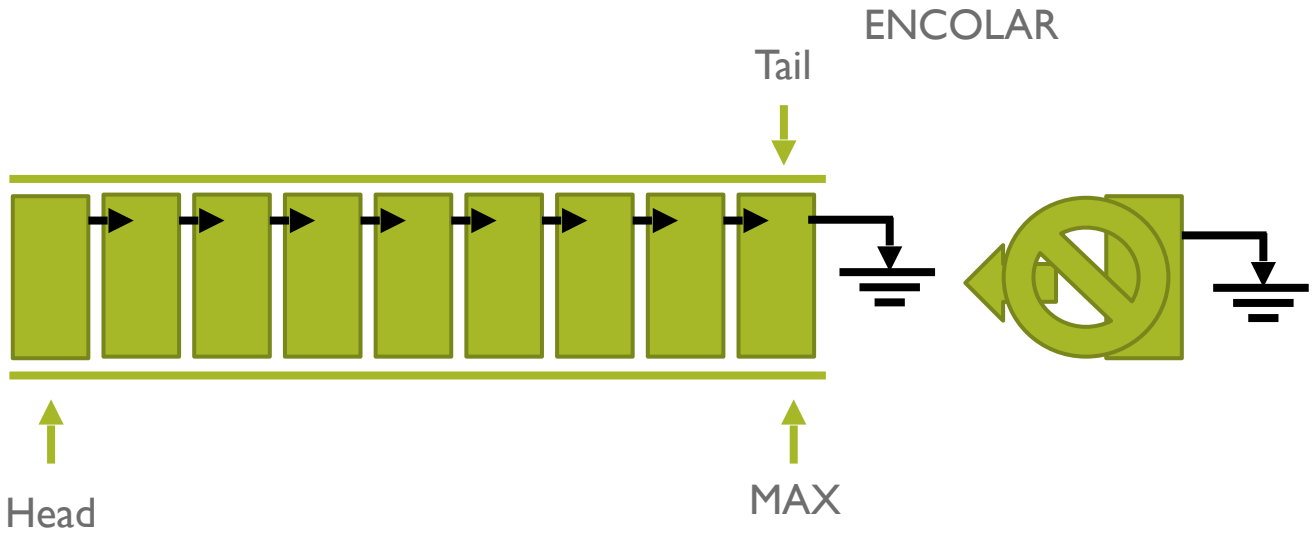


Cola vacía.

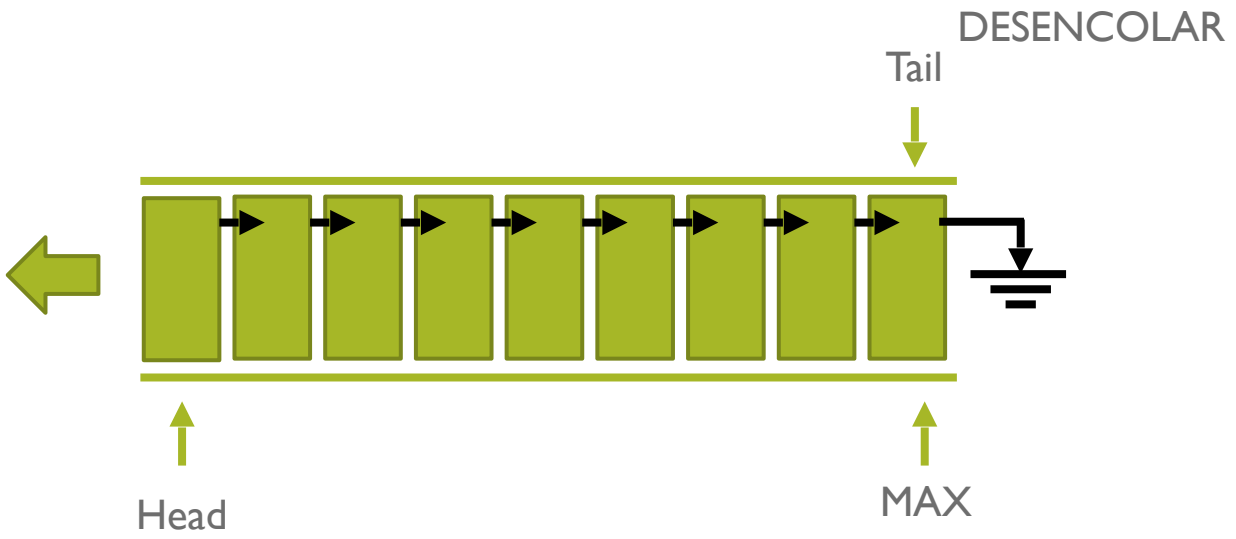
Cola llena.



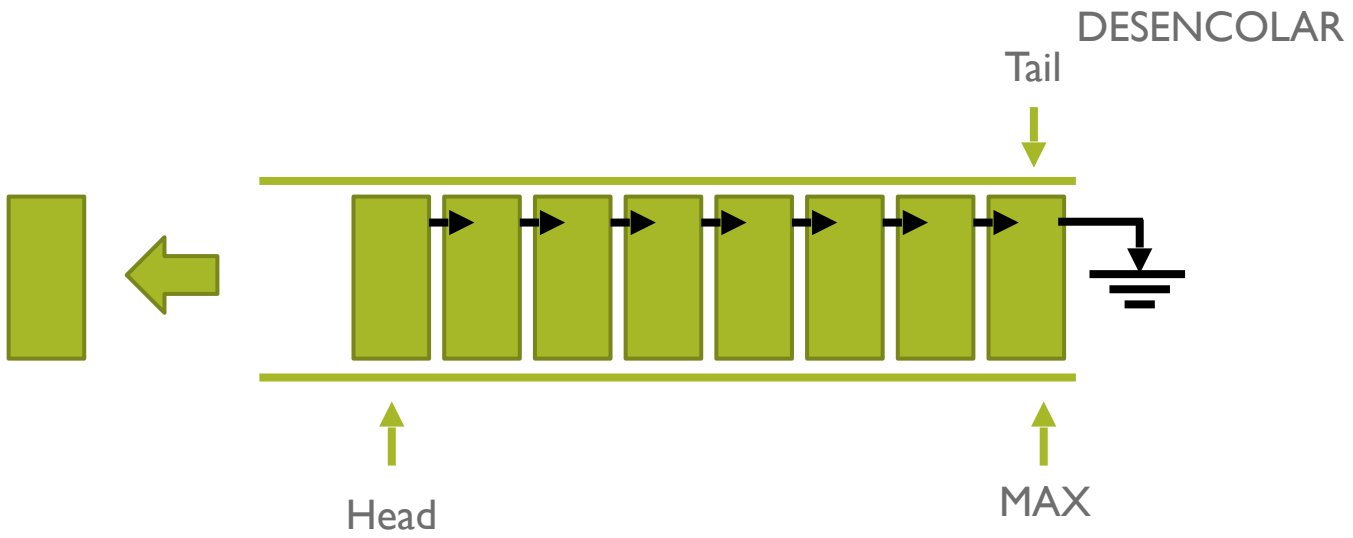
Cola llena.



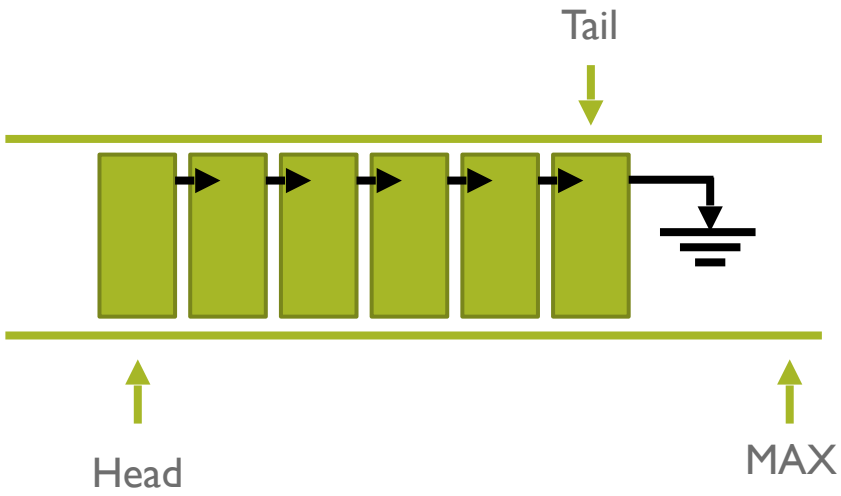
Cola llena.



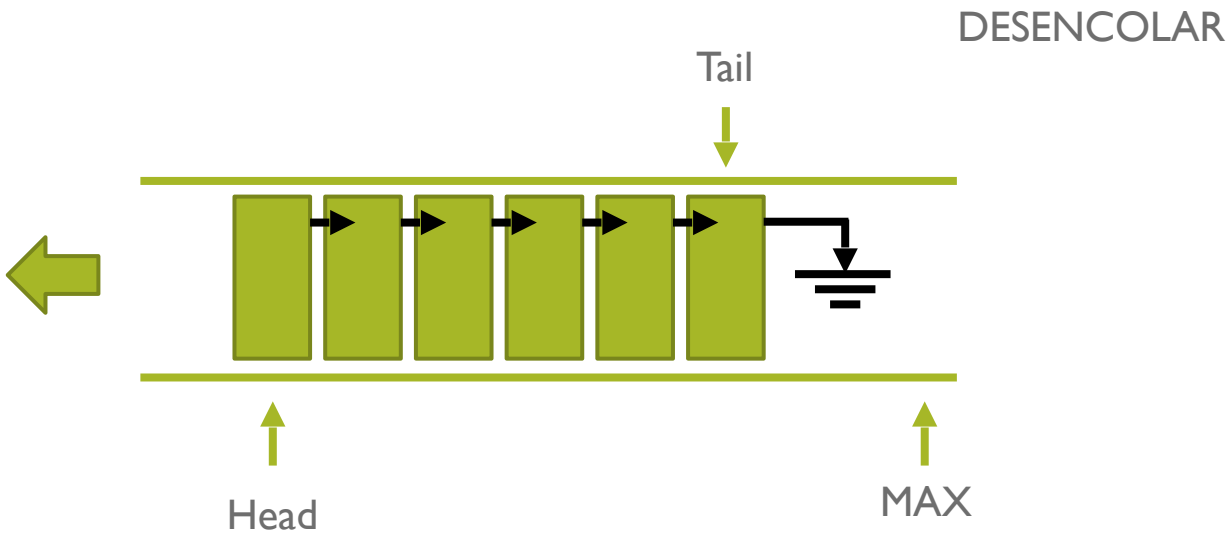
Cola llena.



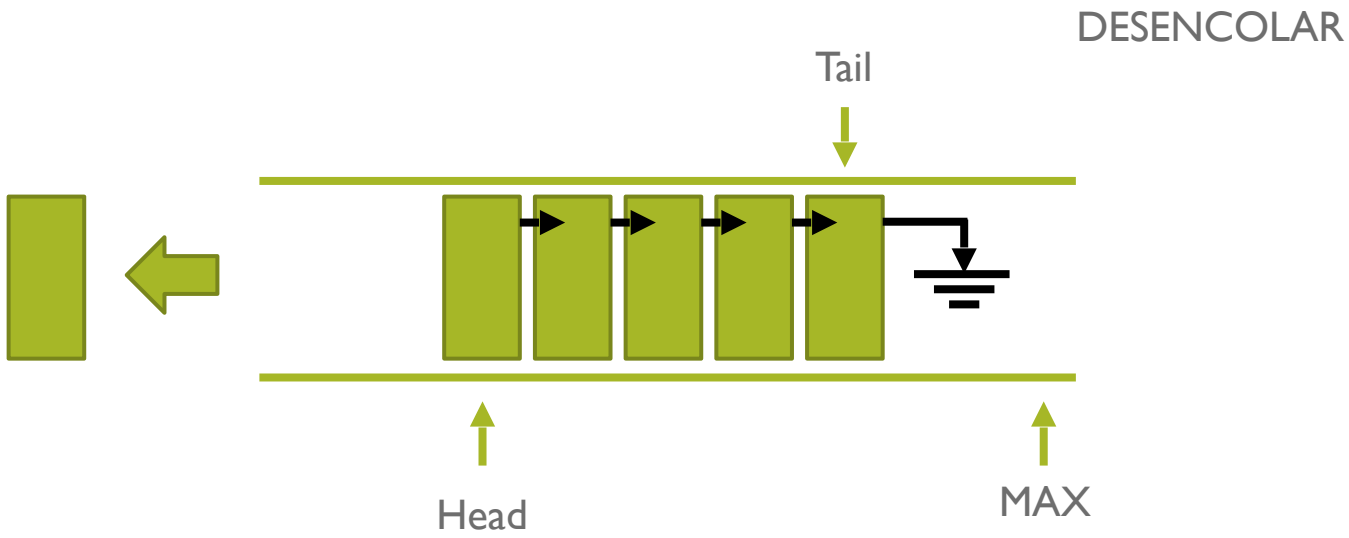
Cola con elementos.



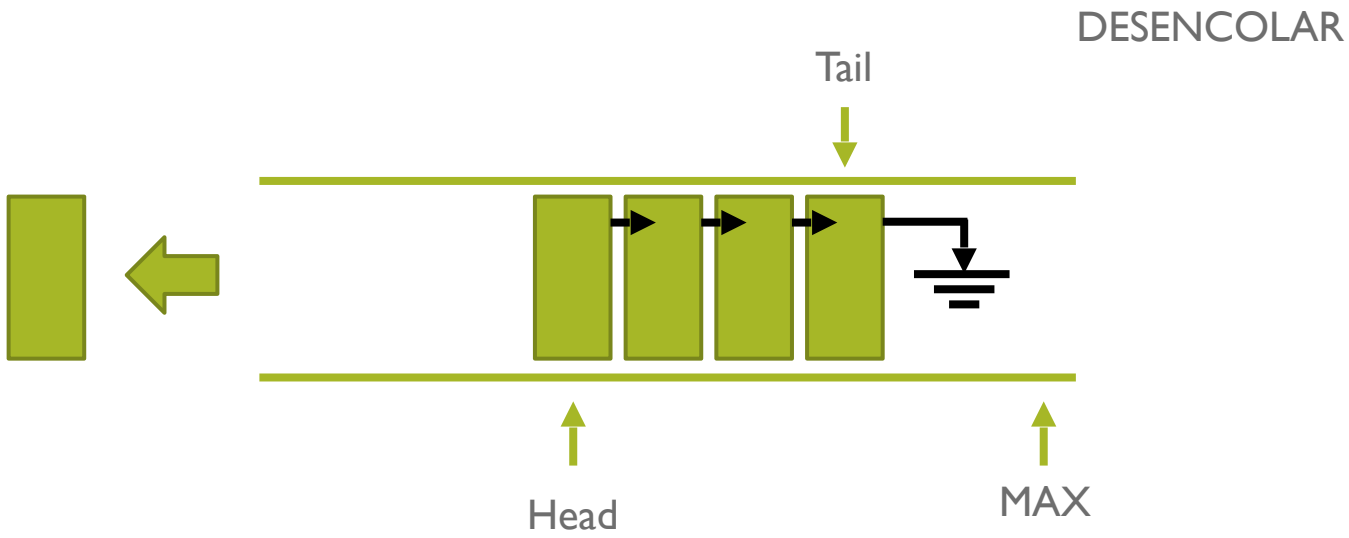
Cola con elementos.



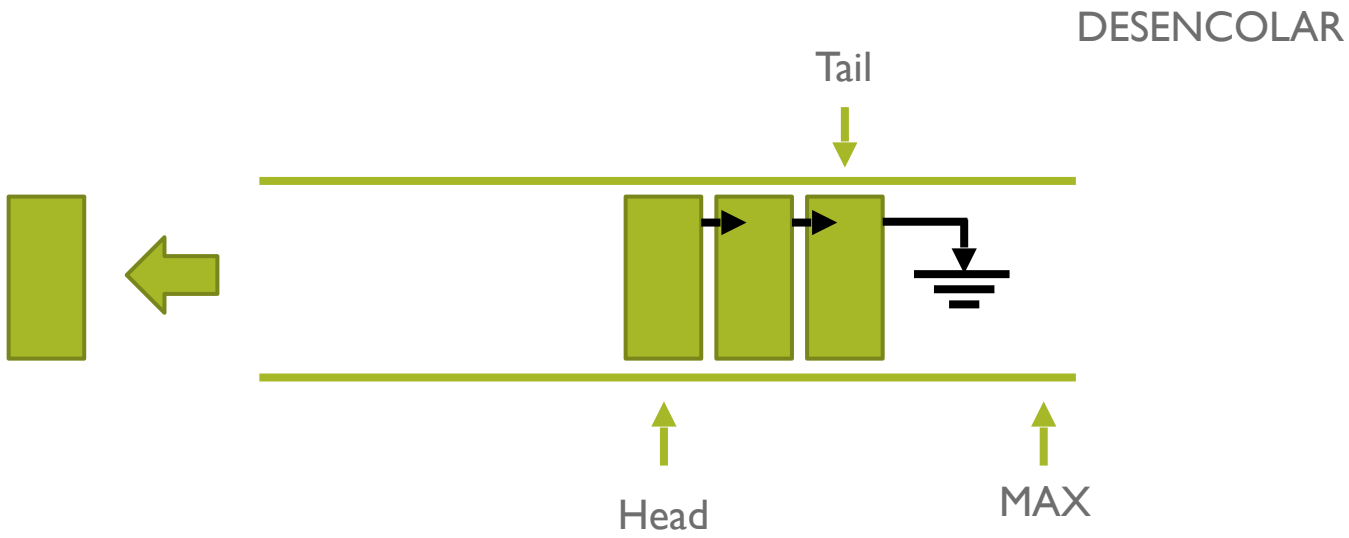
Cola con elementos.



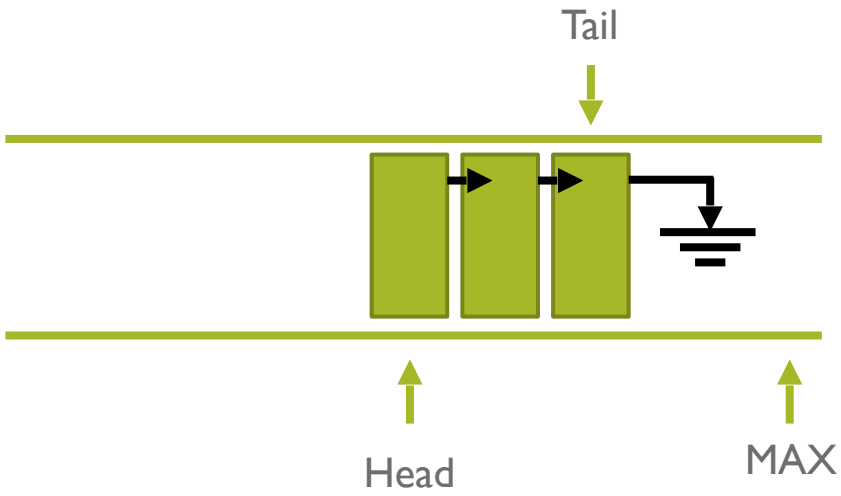
Cola con elementos.



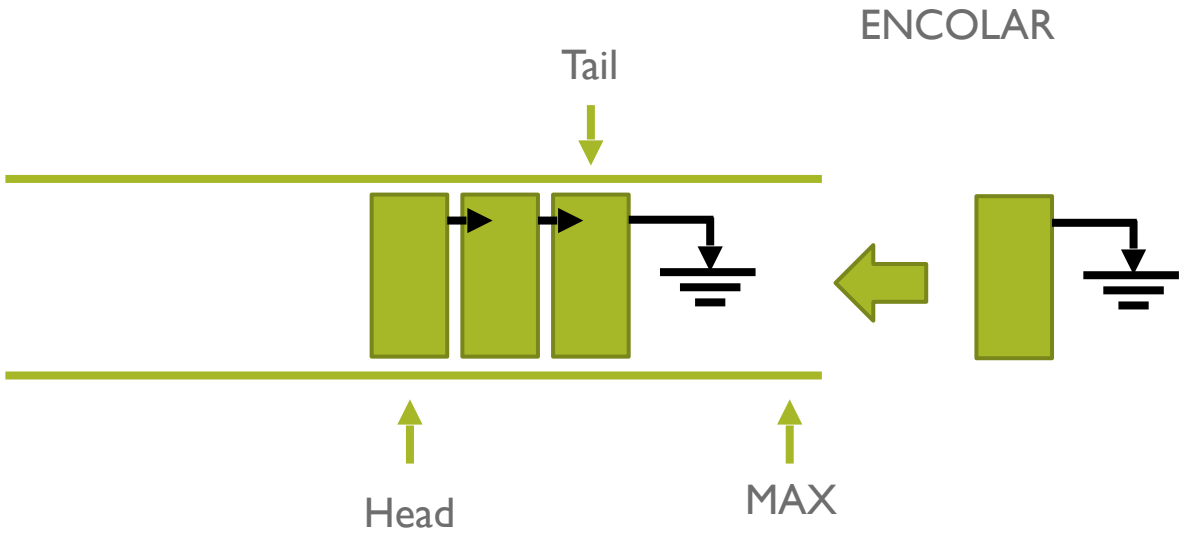
Cola con elementos.



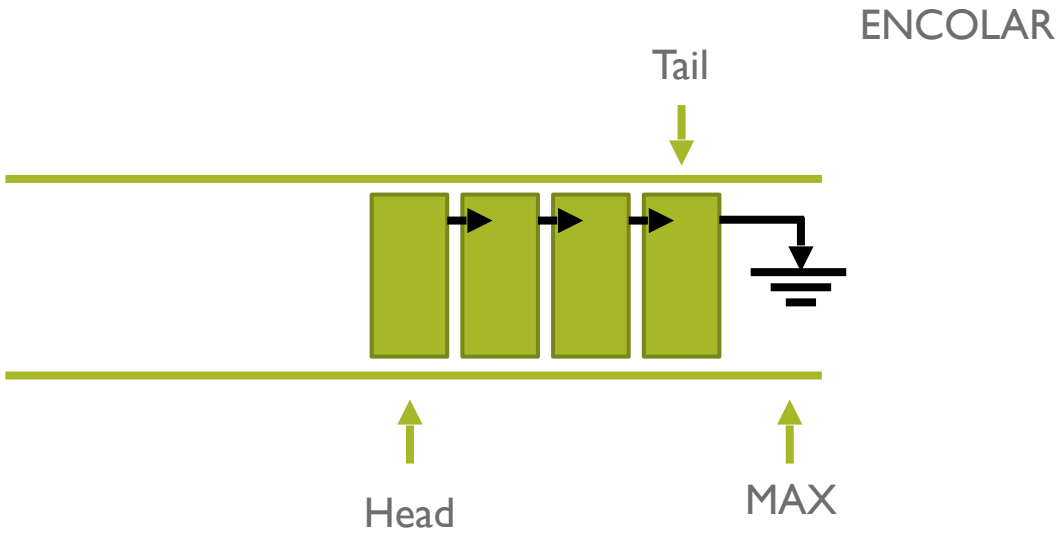
Cola con elementos.



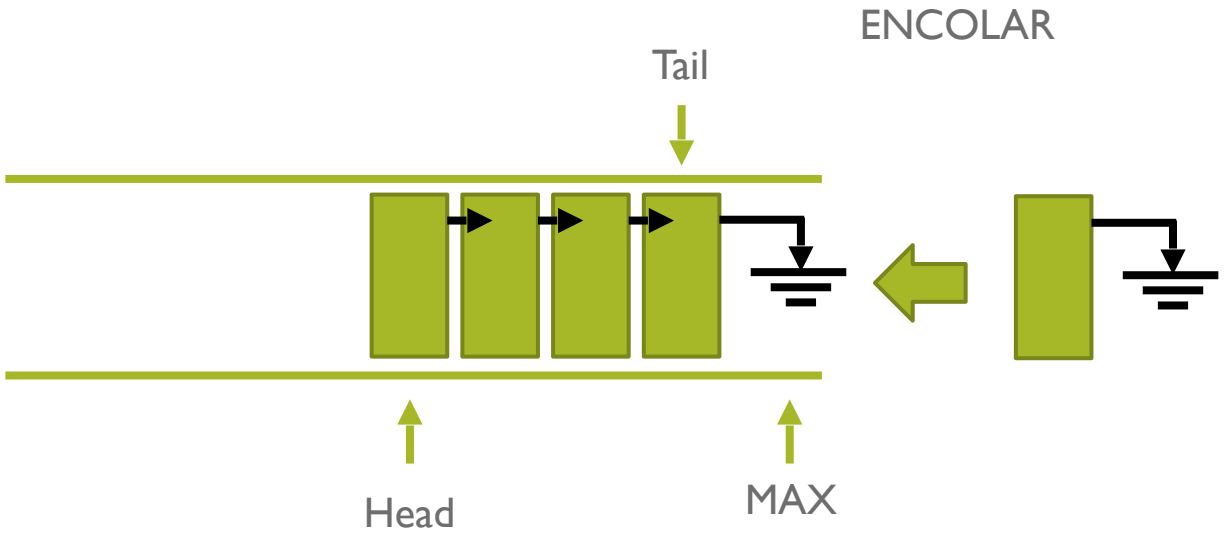
Cola con elementos.



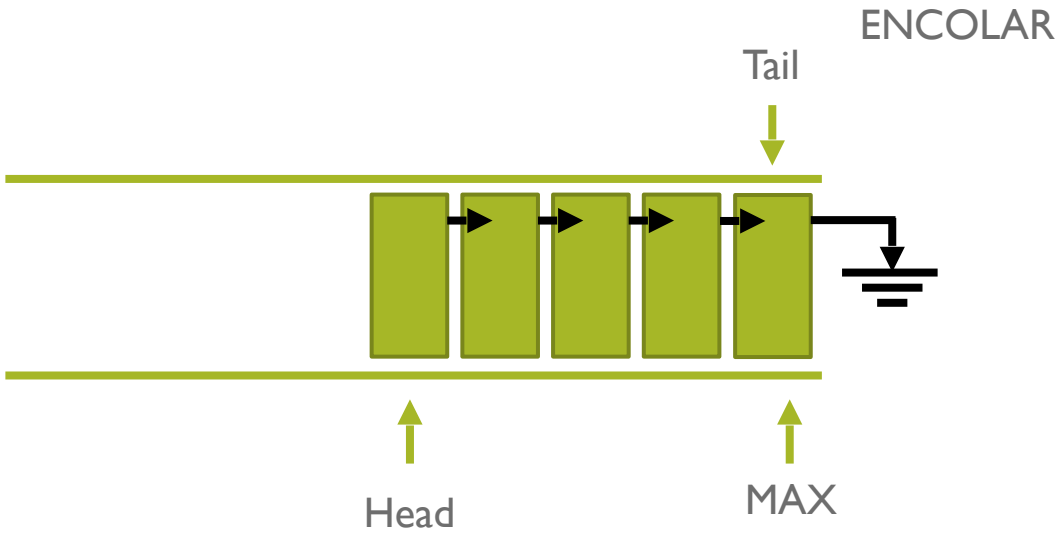
Cola con elementos.



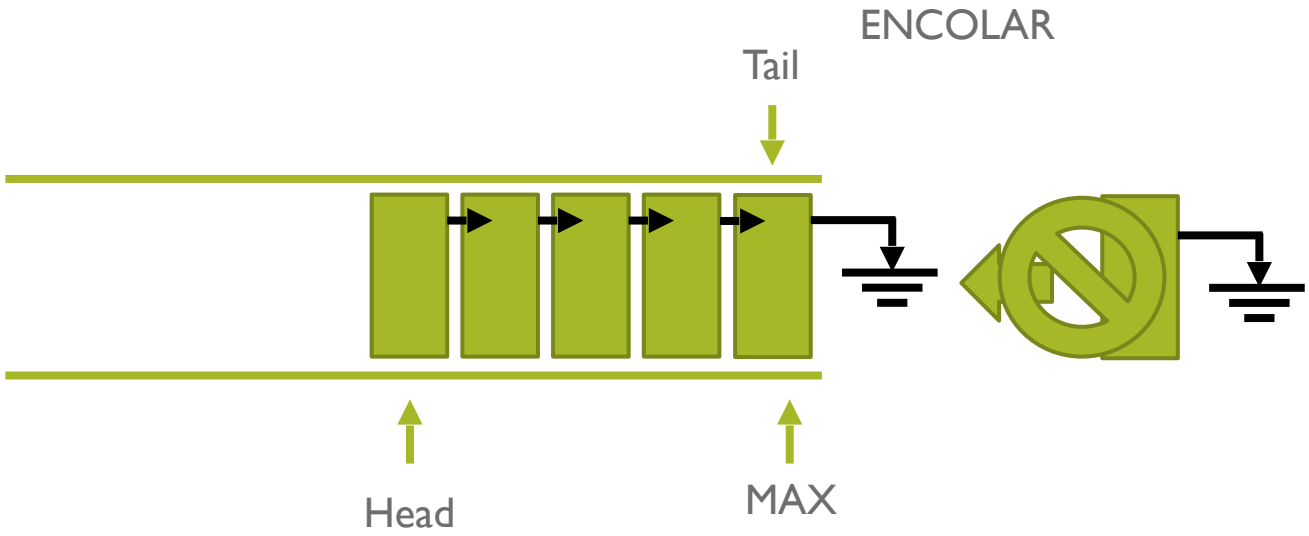
Cola con elementos.



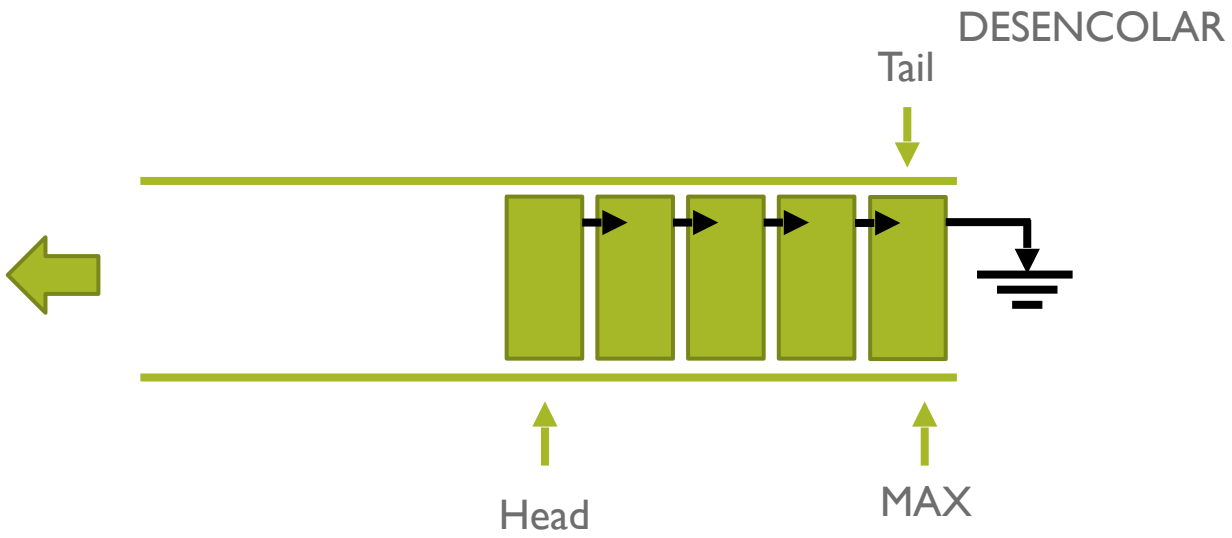
Cola con elementos.



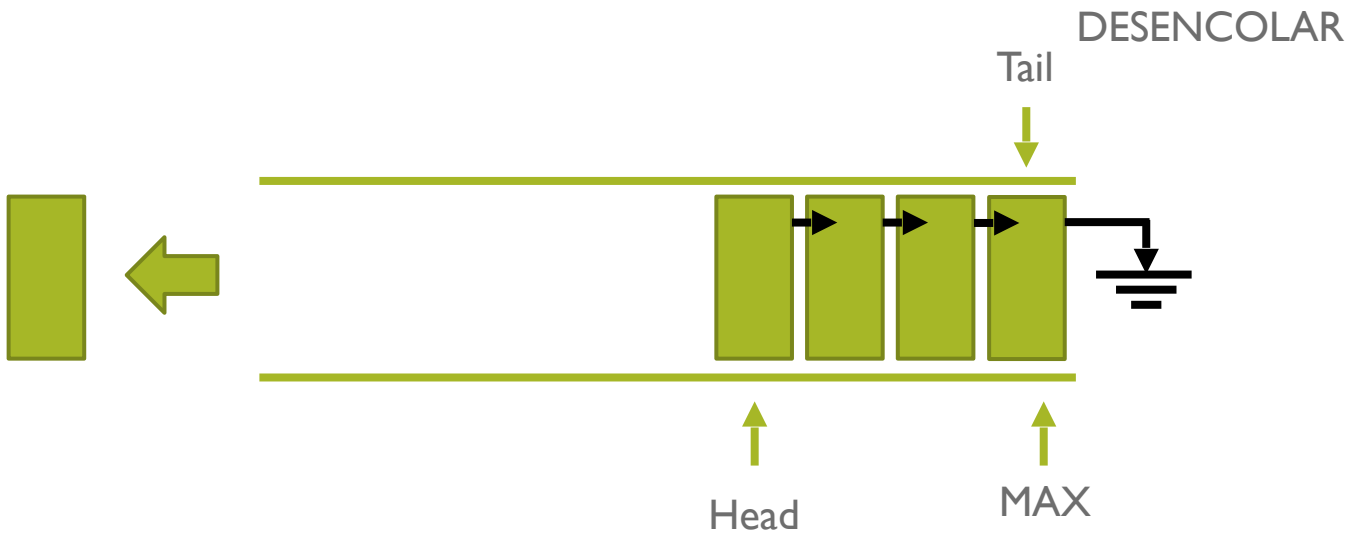
Cola con elementos.



Cola con elementos.



Cola con elementos.



¿Aplicaciones?

Figura 3. Tránsito de vehículos en LA.

Implementar la estructura de datos COLA en lenguaje C. La COLA debe permitir manipular nodos con las operaciones básicas ENCOLAR (ENQUEUE) y DESENCOLAR (DEQUEUE).

Implementar una función MOSTRAR para ver los elementos de la estructura de datos COLA.

"Weeks of coding can save you hours of planning." - Unknown

@CodeWisdom

© Jorge A. Solano

ESTRUCTURAS DE DATOS LINEALES: PILA Y COLA.

Práctica 5

© Jorge A. Solano

5. Estructuras de datos lineales: Pila y cola.

Crear una aplicación tipo estructura de datos PILA que permita apilar y desapilar nodos.

Crear una aplicación tipo estructura de datos COLA que permita encolar y desencolar nodos.

Especificaciones:

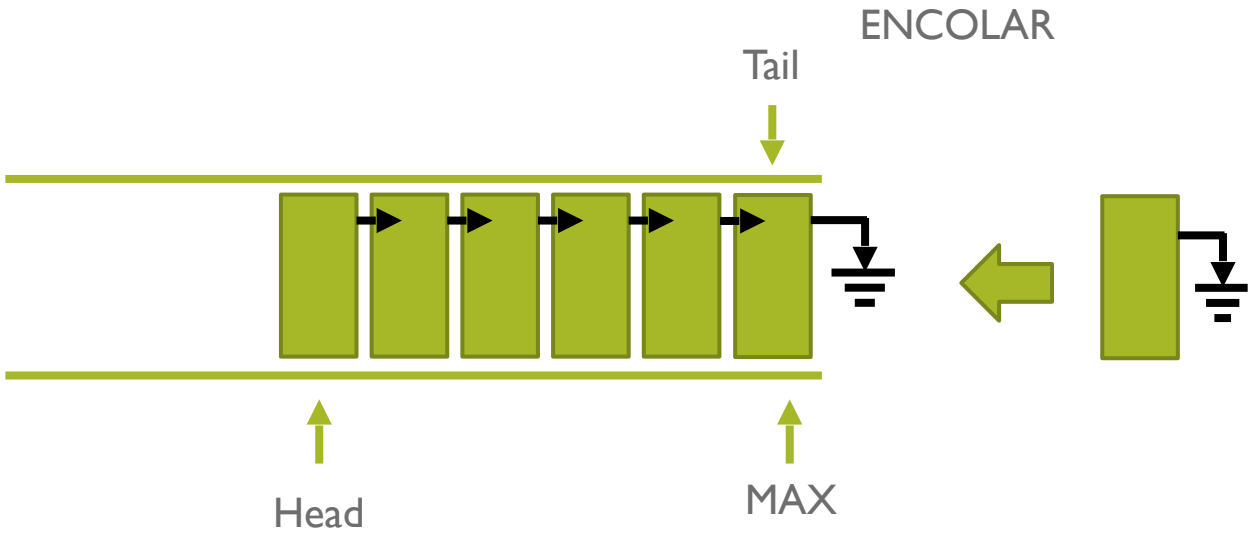
- La función main de las aplicaciones solo debe tener una llamada a otra función llamada menú, la cual debe tener las opciones descritas para cada Estructura de datos solicitada.
- Todas las operaciones, incluyendo el menú, se programan en funciones distintas y en archivos distintos.

© Jorge A. Solano

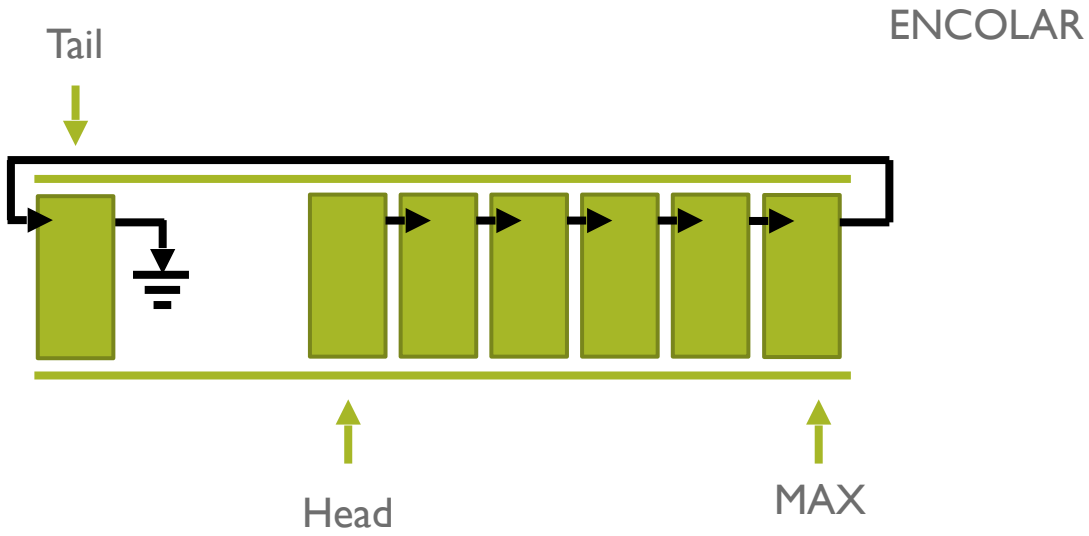
Cola circular.

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.

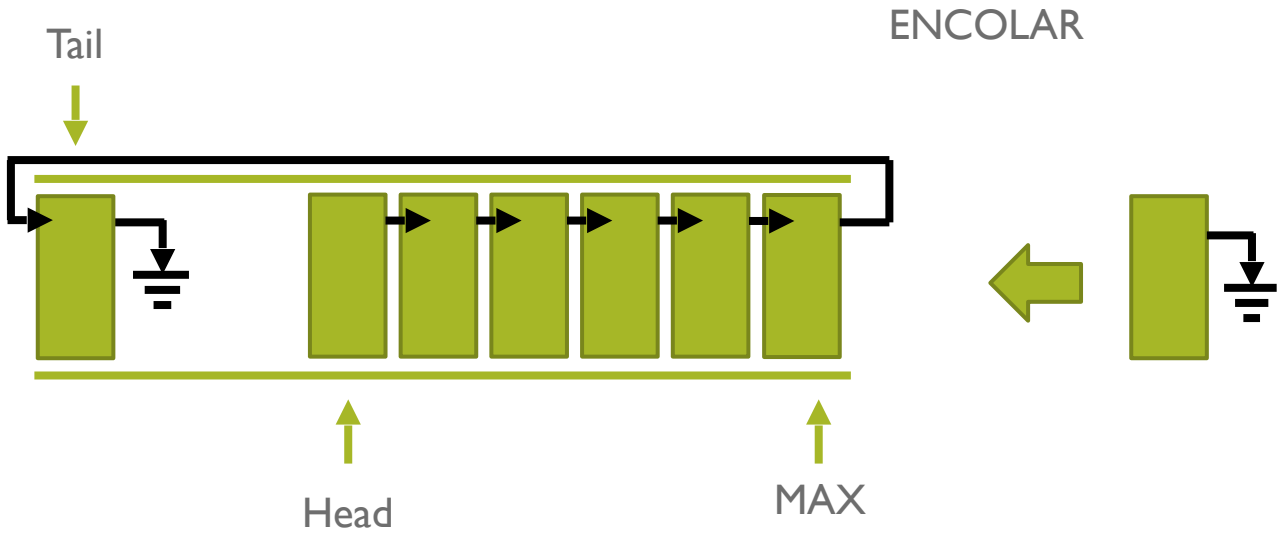
Cola con elementos.



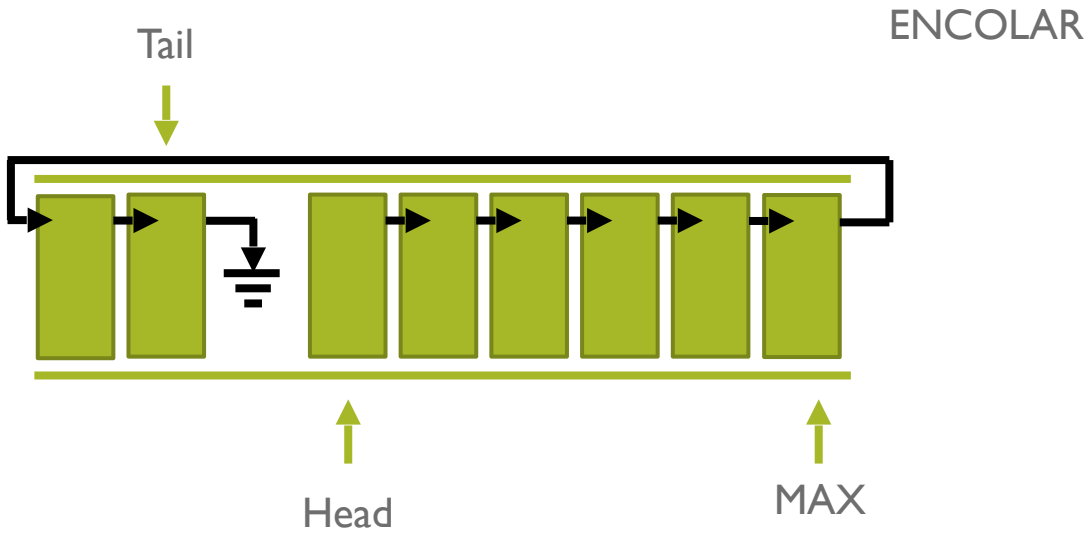
Cola con elementos.



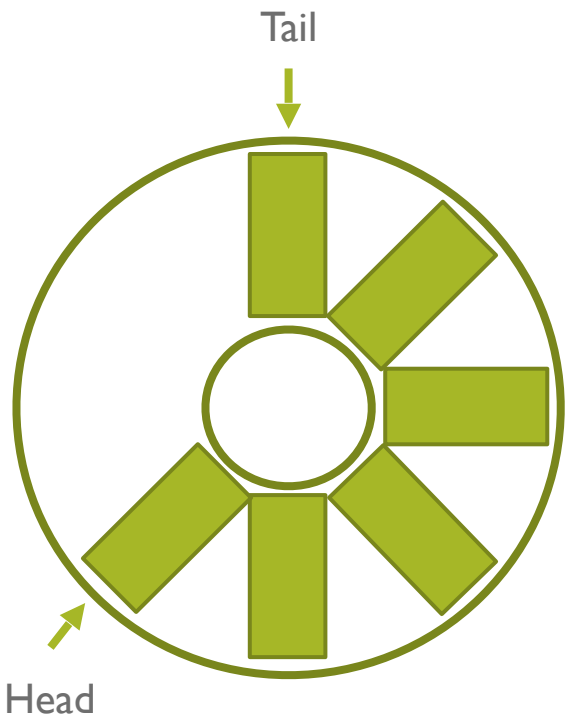
Cola con elementos.



Cola con elementos.

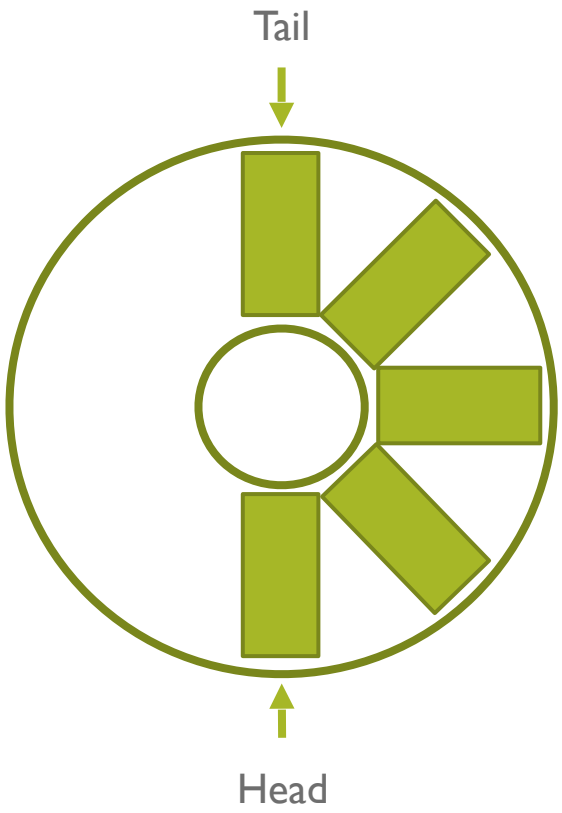


Cola con elementos.



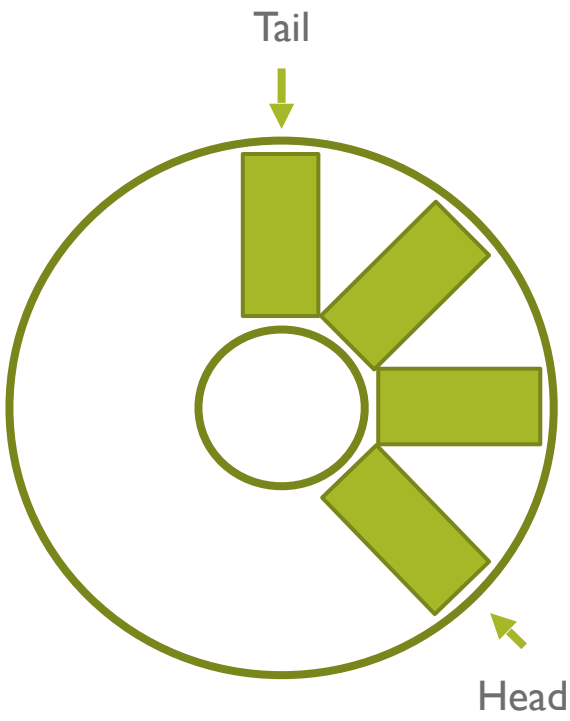
DESENCOLAR

Cola con elementos.



DESENCOLAR

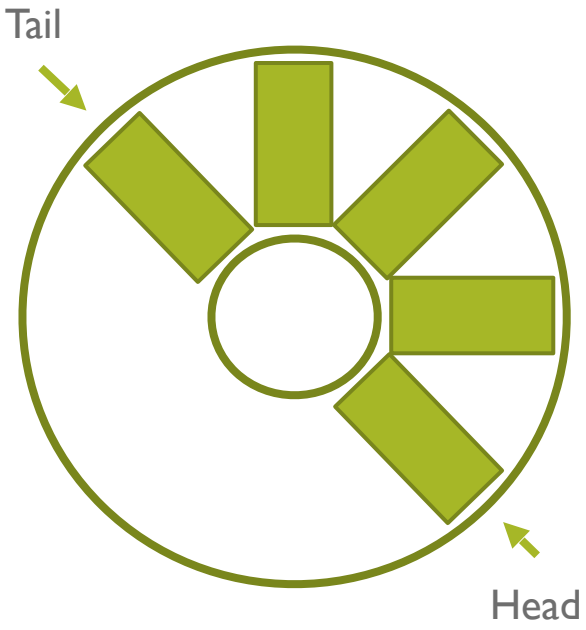
Cola con elementos.



DESENCOLAR

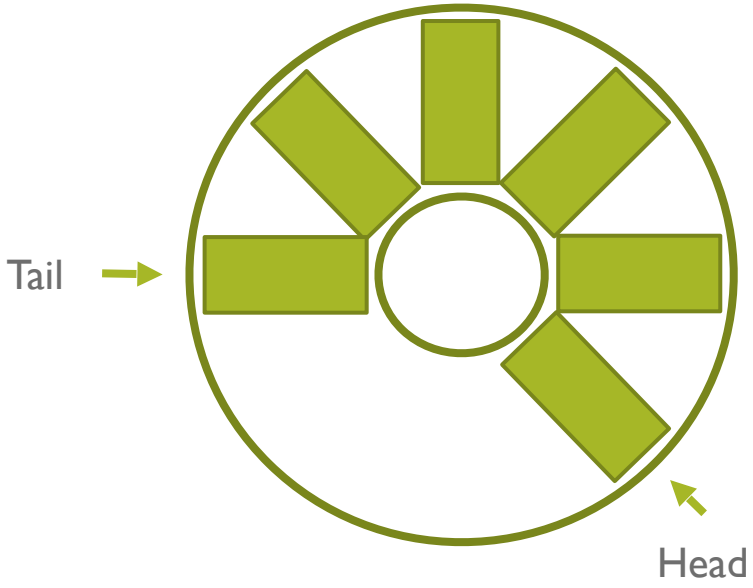
Cola con elementos.

ENCOLAR



Cola con elementos.

ENCOLAR



¿Aplicaciones?

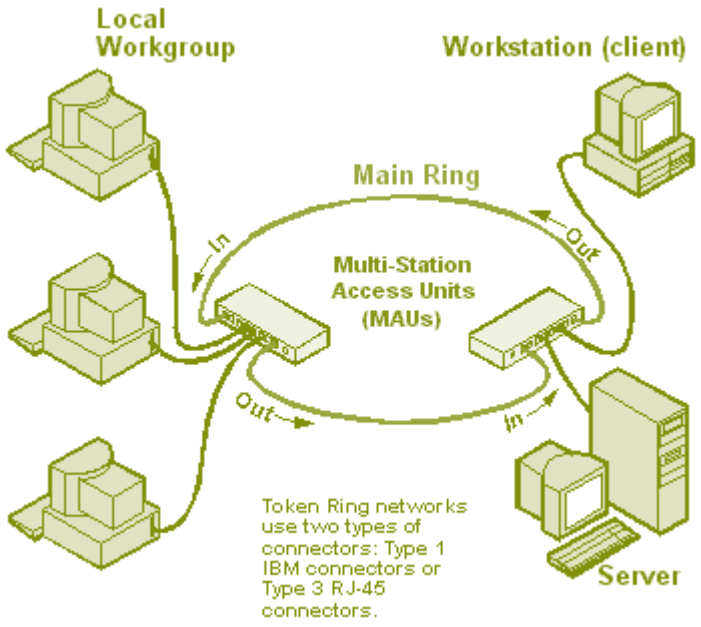


Figura 4. Token ring.

Implementar la estructura de datos COLA CIRCULAR en lenguaje C. La COLA CIRCULAR debe permitir manipular nodos con las operaciones básicas ENCOLAR (ENQUEUE) y DESENCOLAR (DEQUEUE).

Implementar una función MOSTRAR para ver los elementos de la estructura de datos COLA CIRCULAR.

"The most important single aspect of software development is to be clear about what you are trying to build."

Bjarne Stroustrup
(A Danish computer scientist, who is most notable for the creation and development of the widely used C++ programming language.)

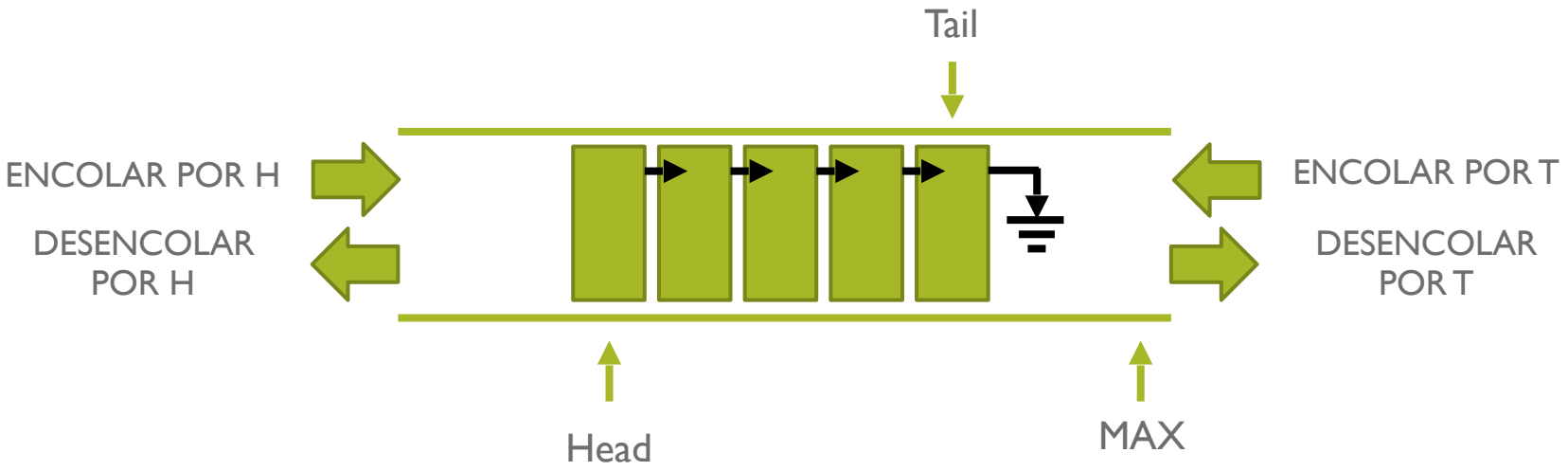
1.3.3 Cola doble: almacenamiento contiguo y ligado, y operaciones.

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

Las operaciones que se pueden realizar dentro de una cola doble son:

- ENCOLAR POR H
- DESENCOLAR POR H
- ENCOLAR POR T
- DESENCOLAR POR T

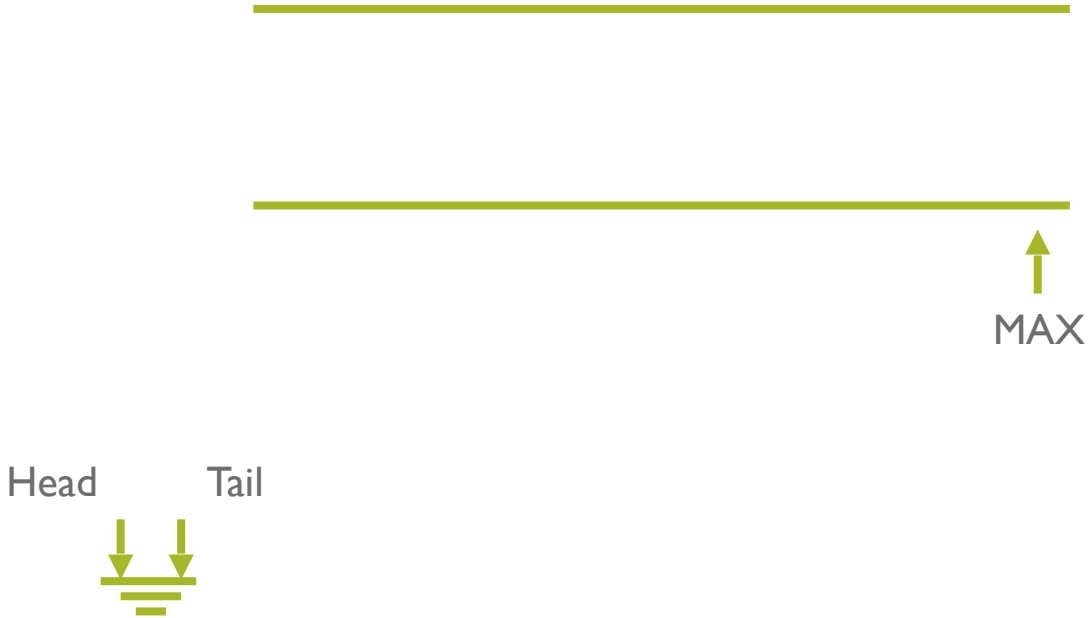
Cola doble.


Tail
ENCOLAR POR H
DESENCOLAR POR H
ENCOLAR POR T
DESENCOLAR POR T
Head
MAX

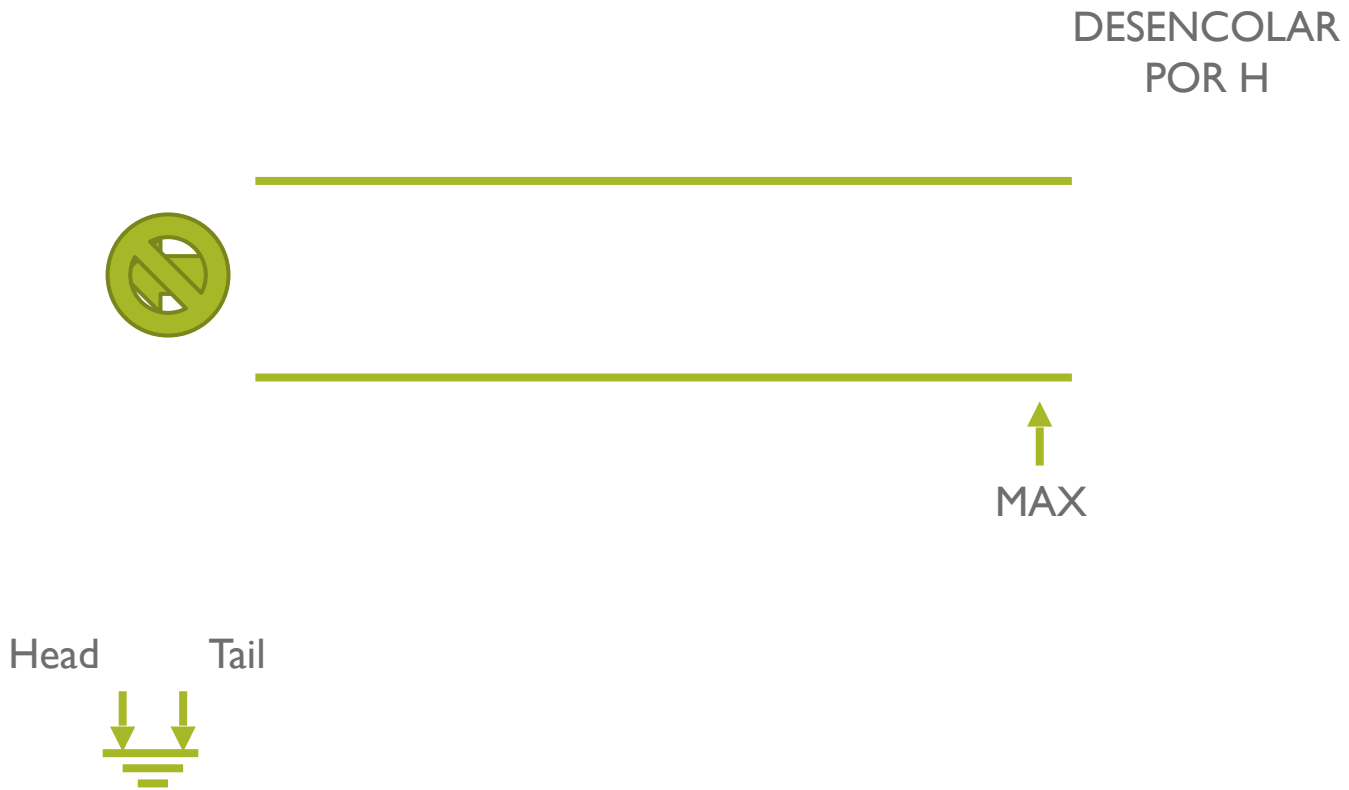
Para poder diseñar un programa que defina el comportamiento de una COLA DOBLE se deben considerar 3 casos para las 4 operaciones (INSERTAR y ELIMINAR tanto por T como por H):

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

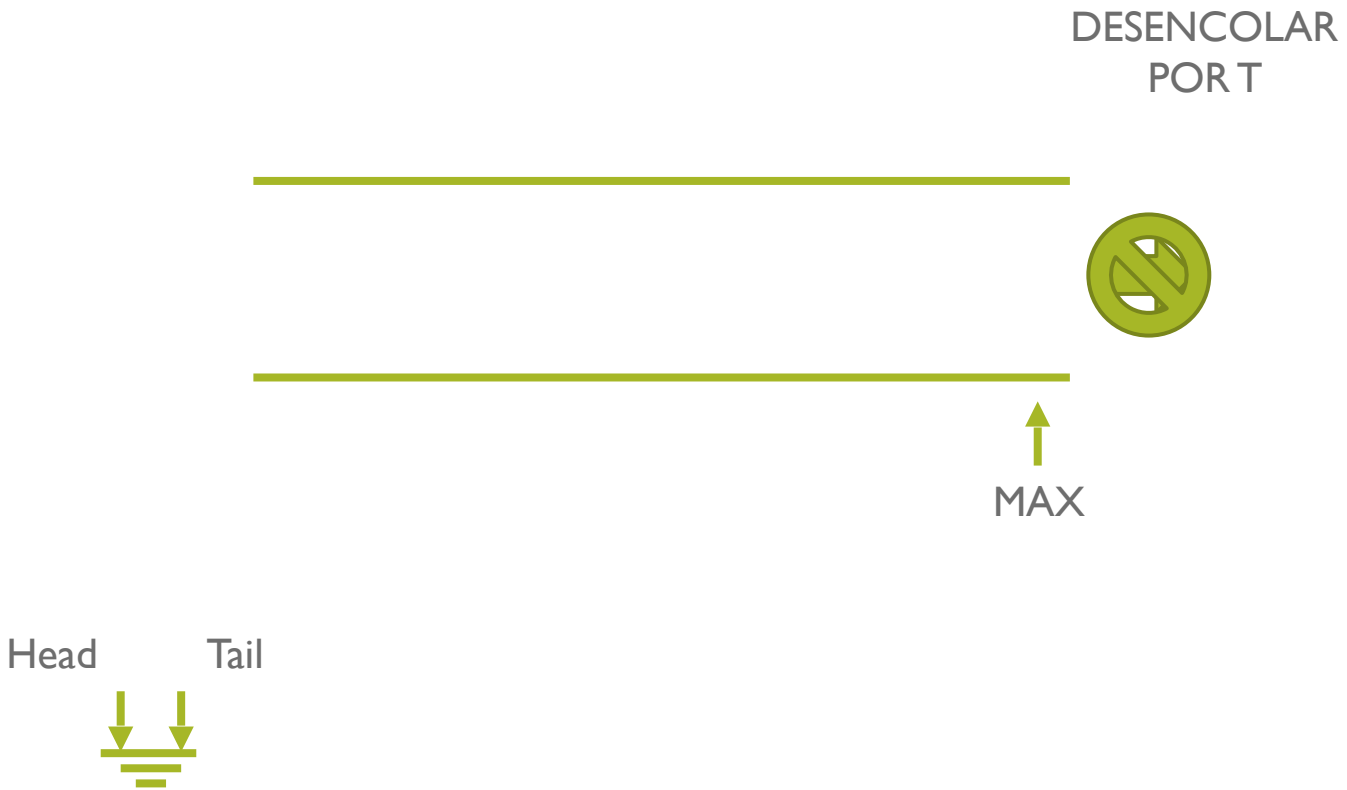
Cola doble vacía.



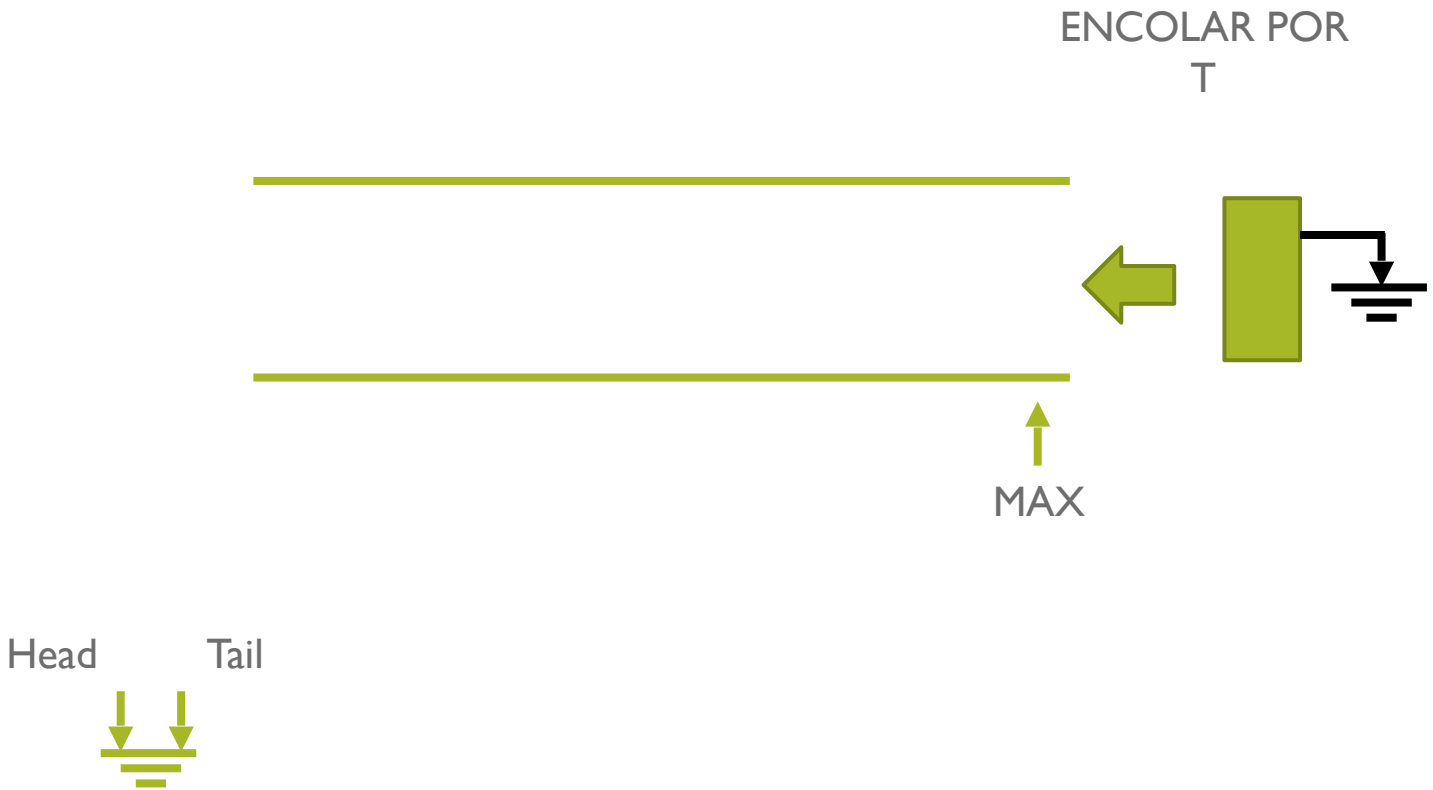
Cola doble vacía.



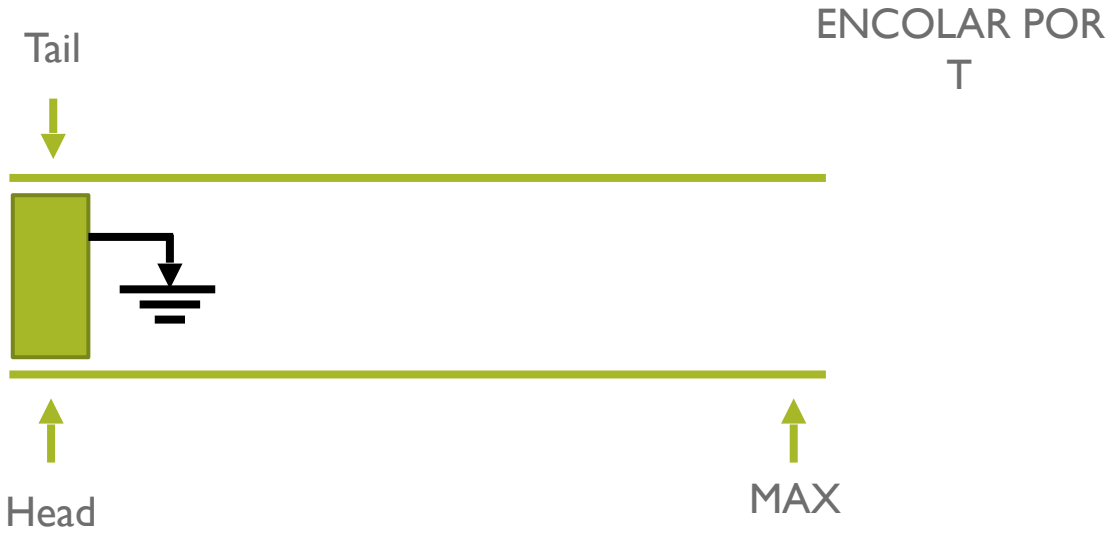
Cola doble vacía.



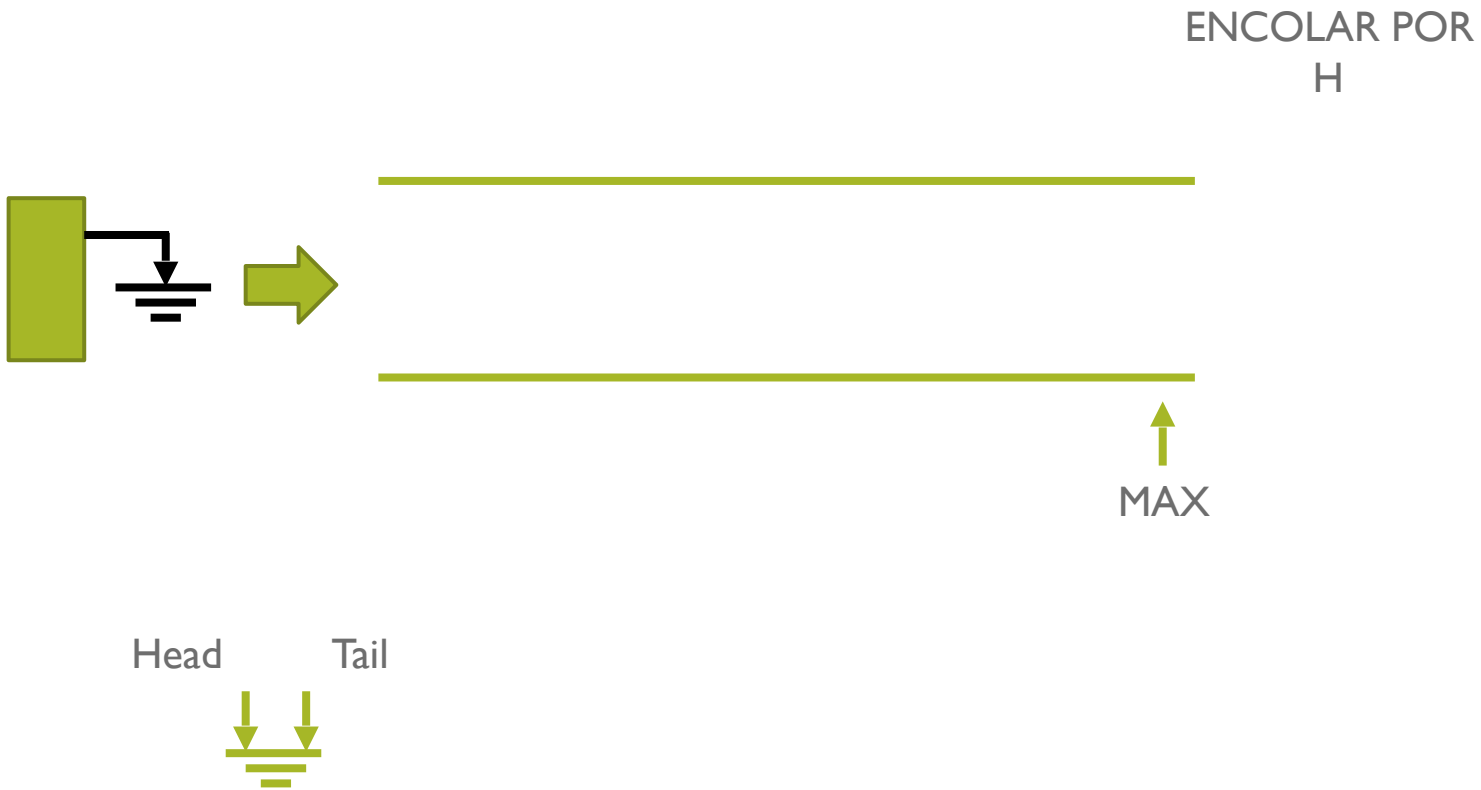
Cola doble vacía.



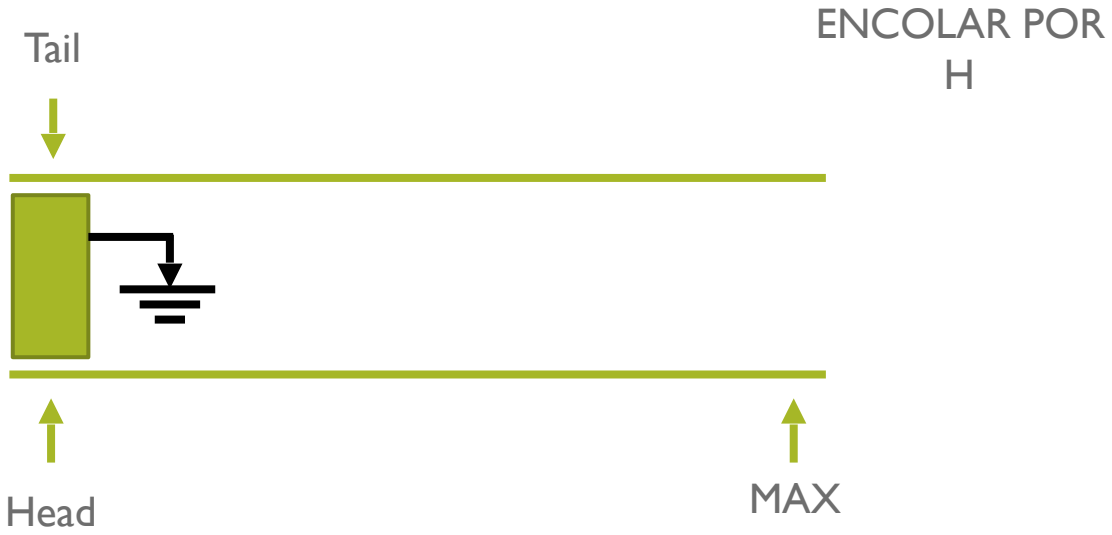
Cola doble vacía.



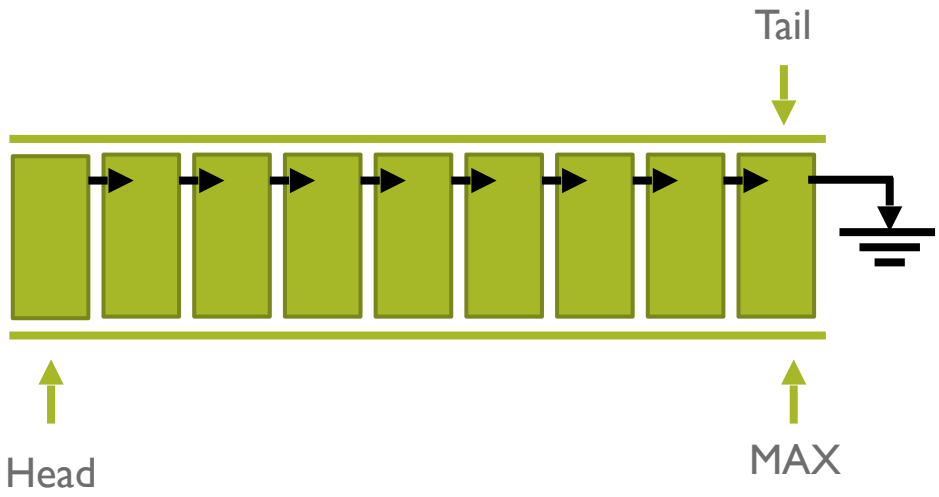
Cola doble vacía.



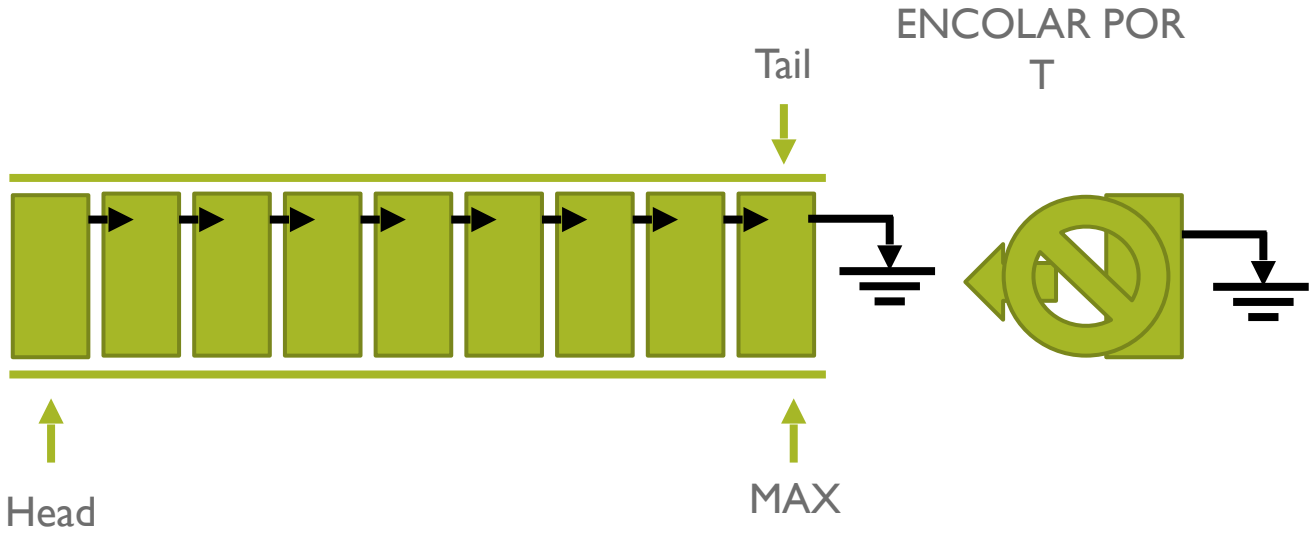
Cola doble vacía.



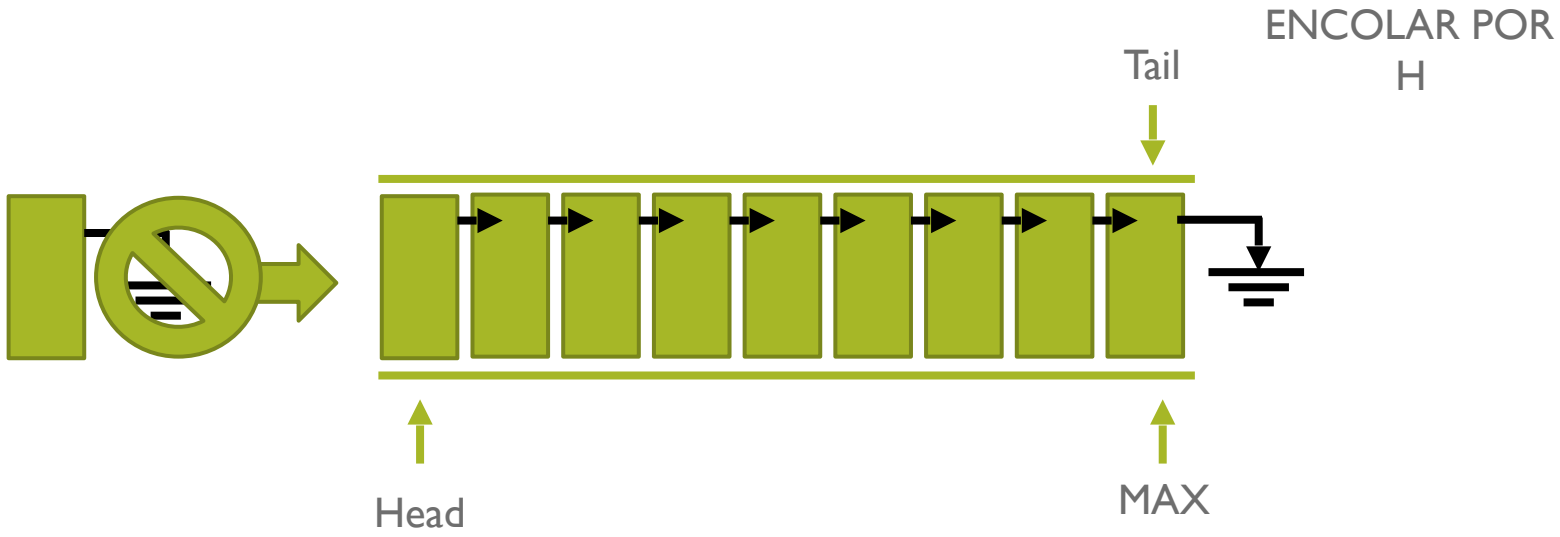
Cola doble llena.



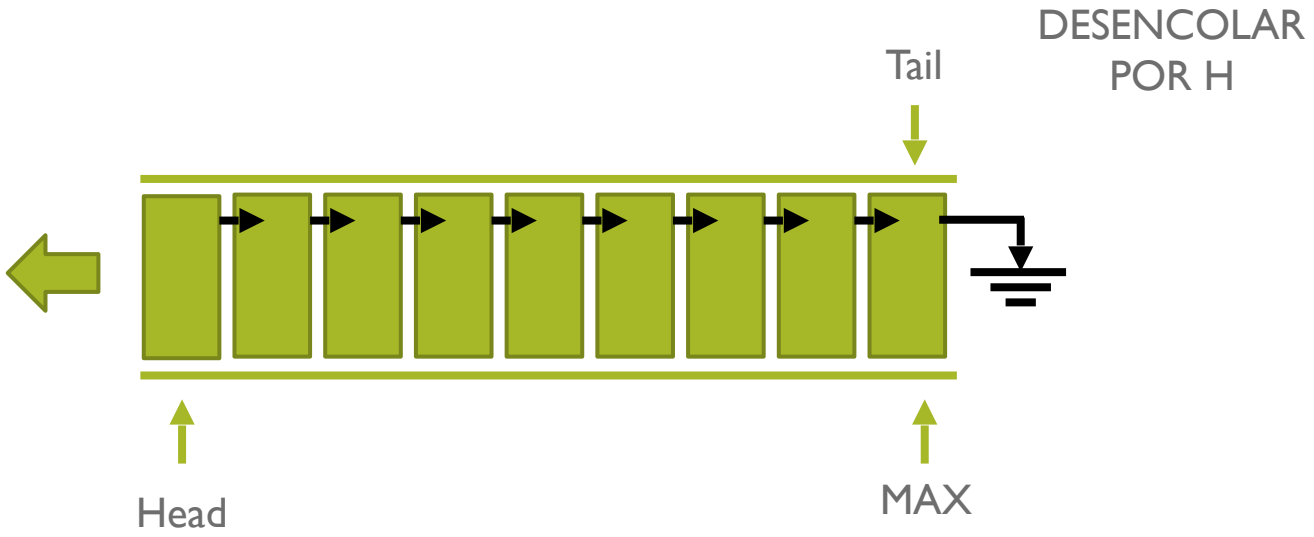
Cola doble llena.



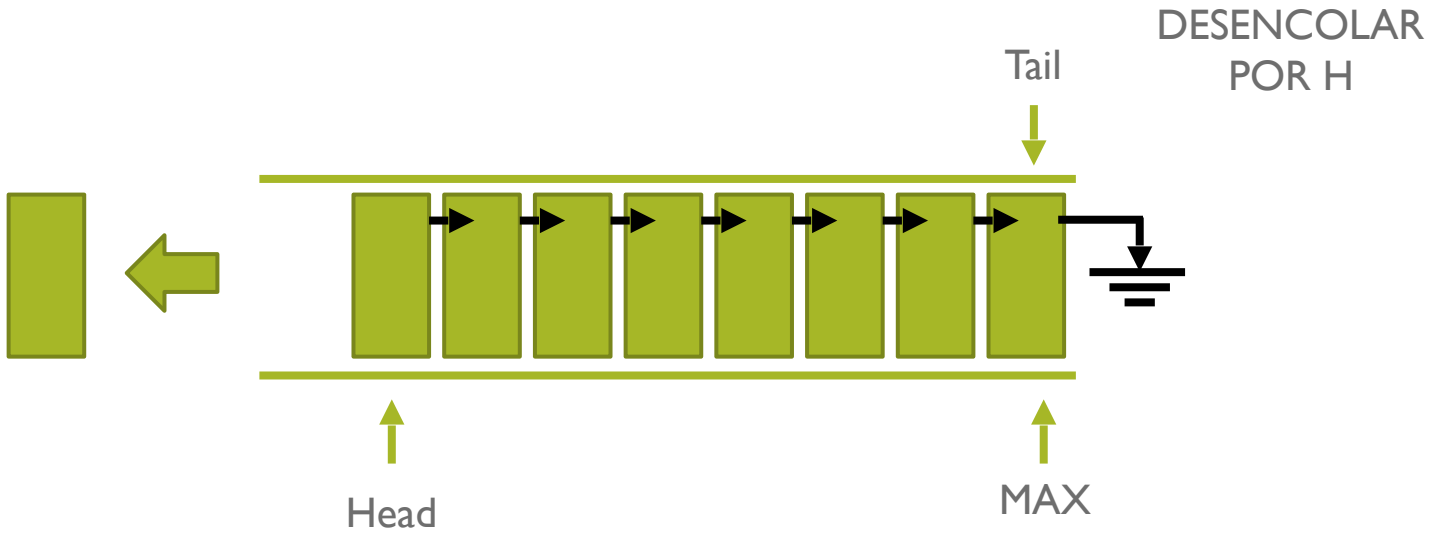
Cola doble llena.



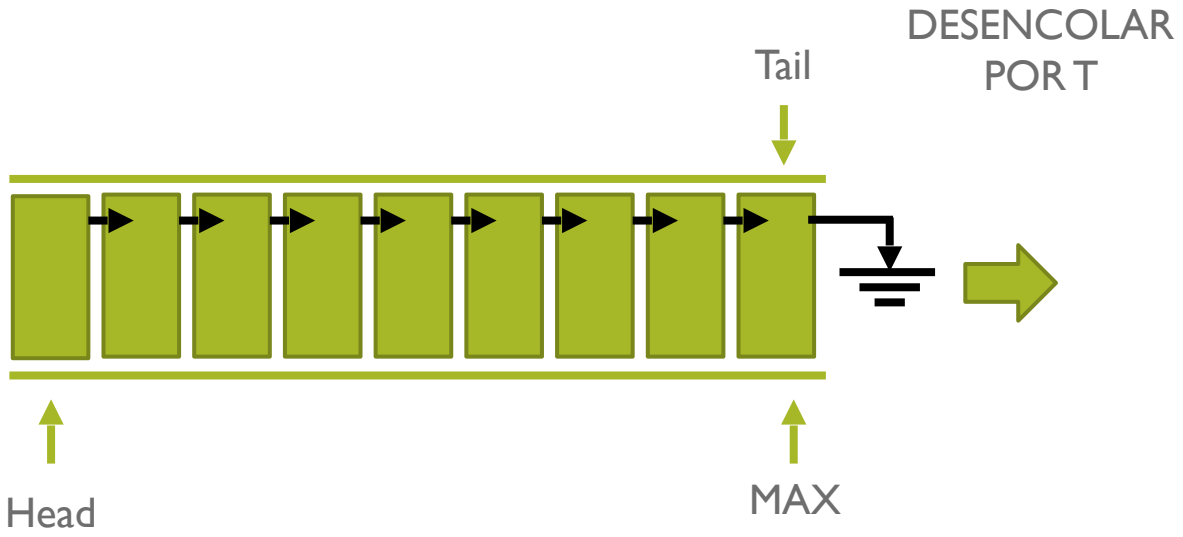
Cola doble llena.



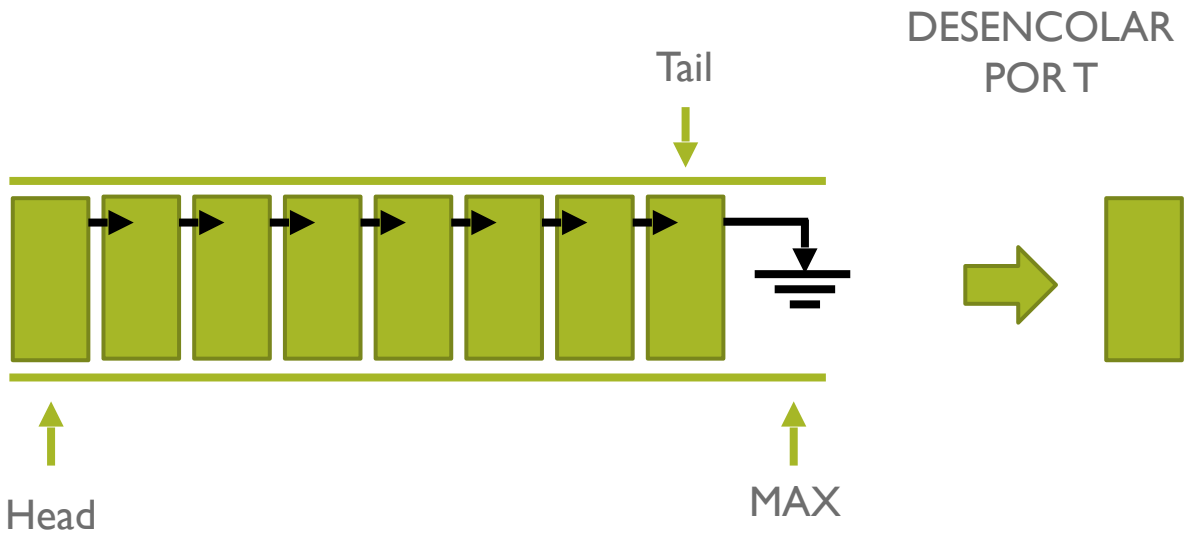
Cola doble llena.



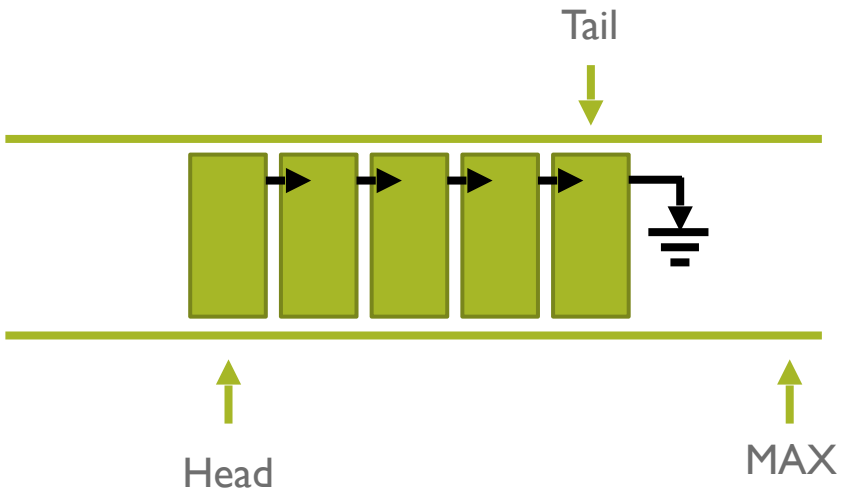
Cola doble llena.



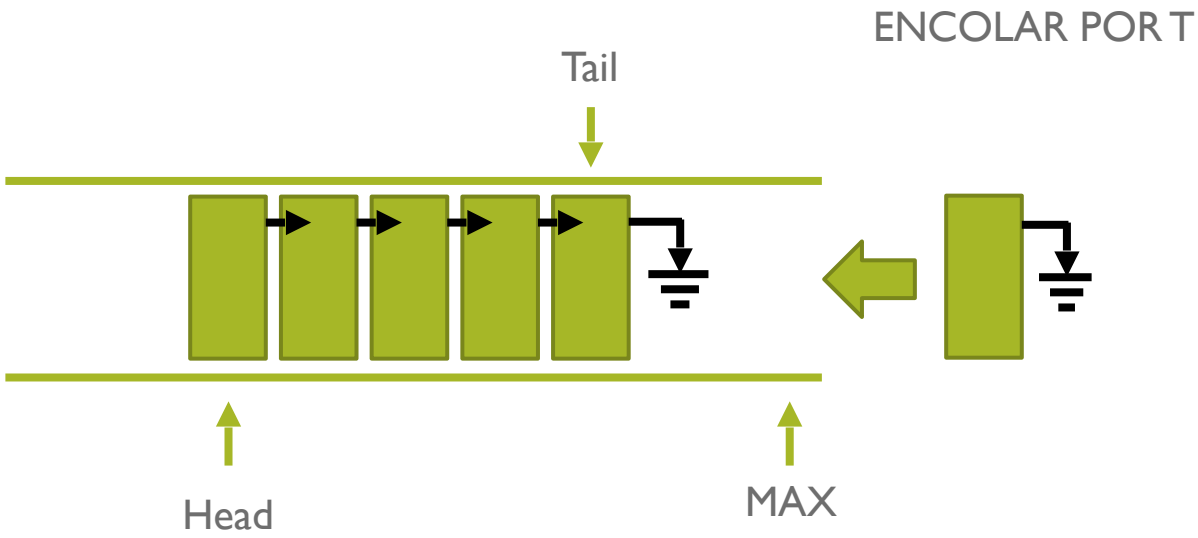
Cola doble llena.



Cola doble con elementos.

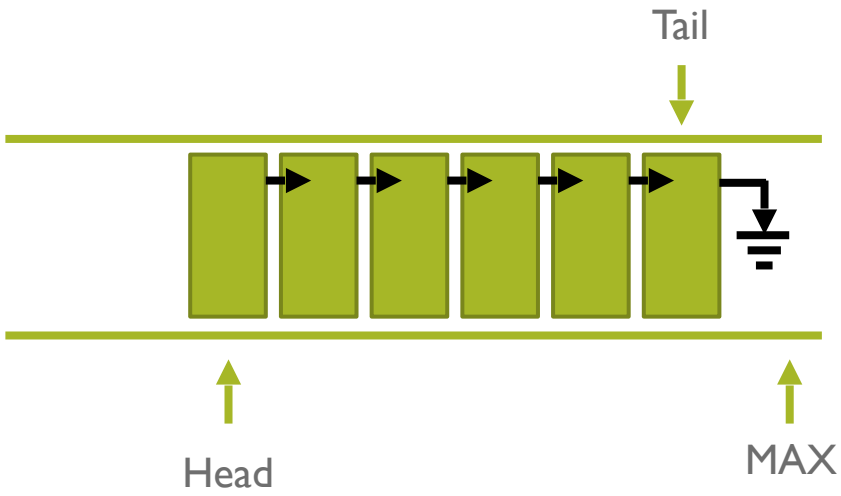


Cola doble con elementos.



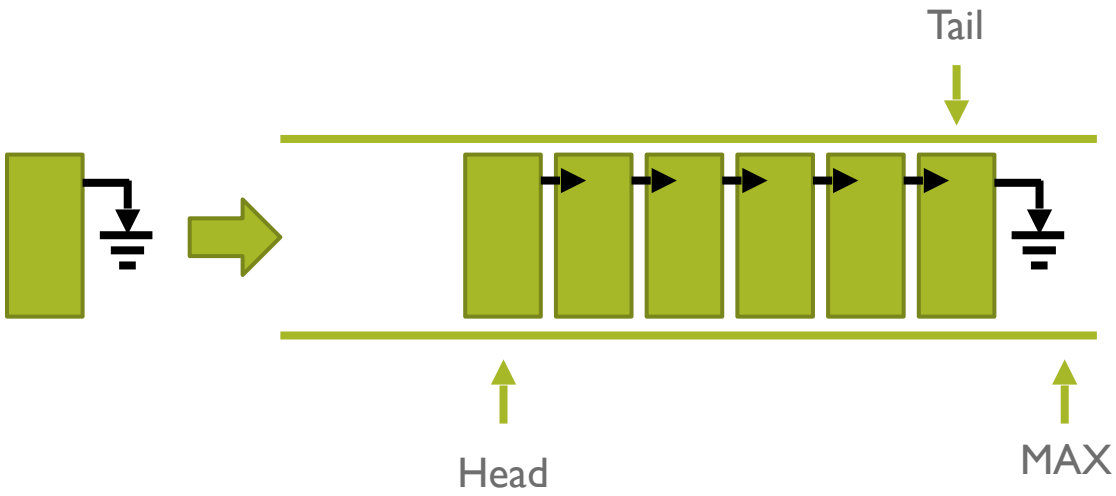
Cola doble con elementos.

ENCOLAR POR T



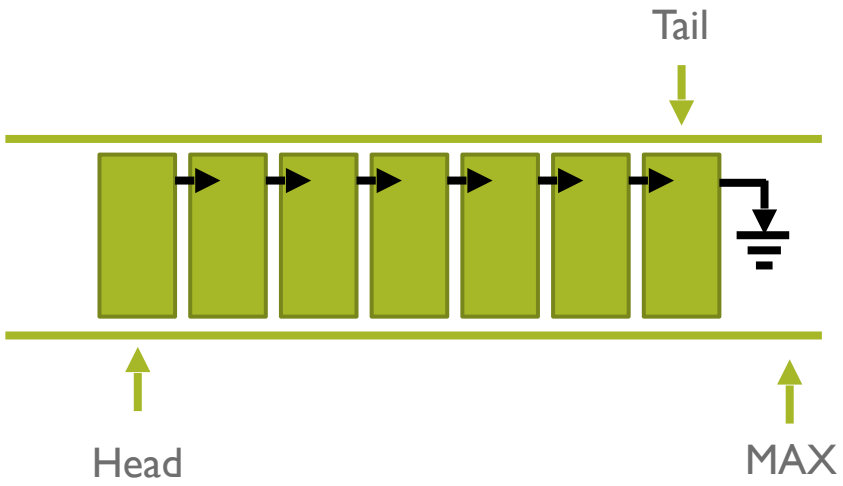
Cola doble con elementos.

ENCOLAR POR H

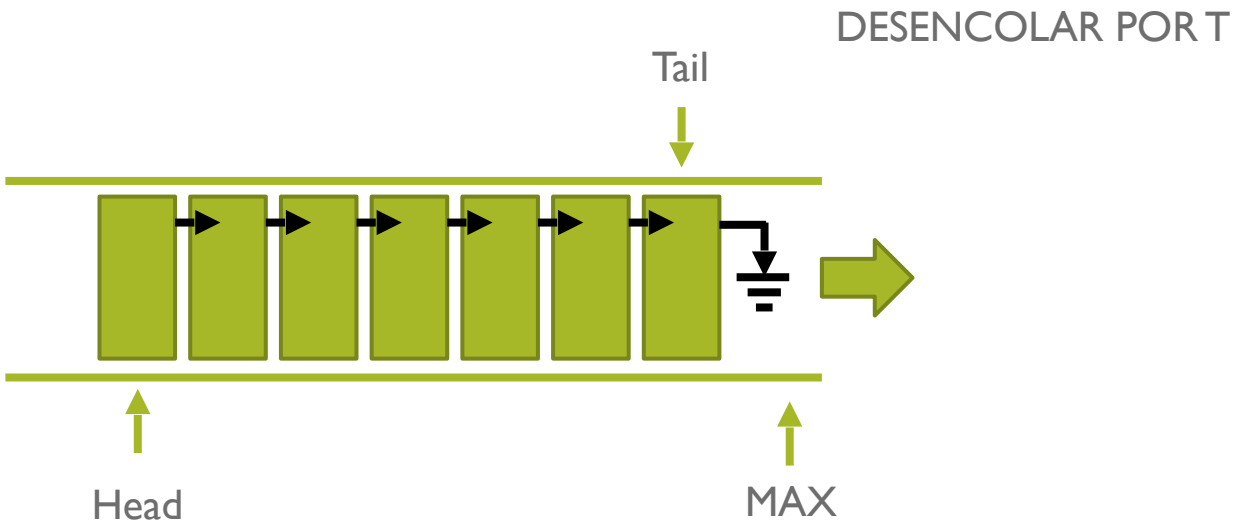


Cola doble con elementos.

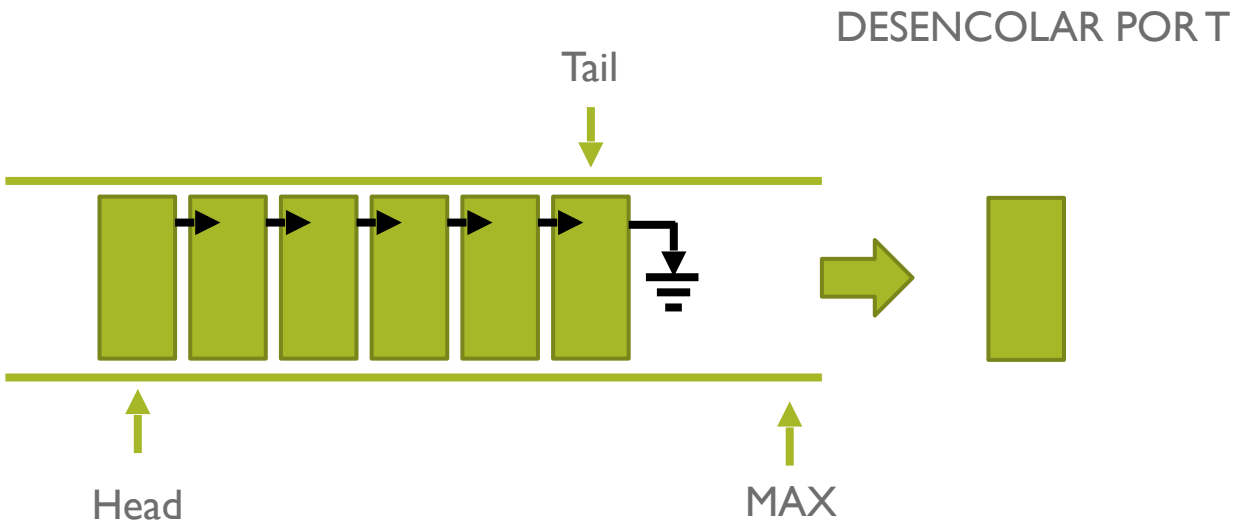
ENCOLAR POR H



Cola doble con elementos.

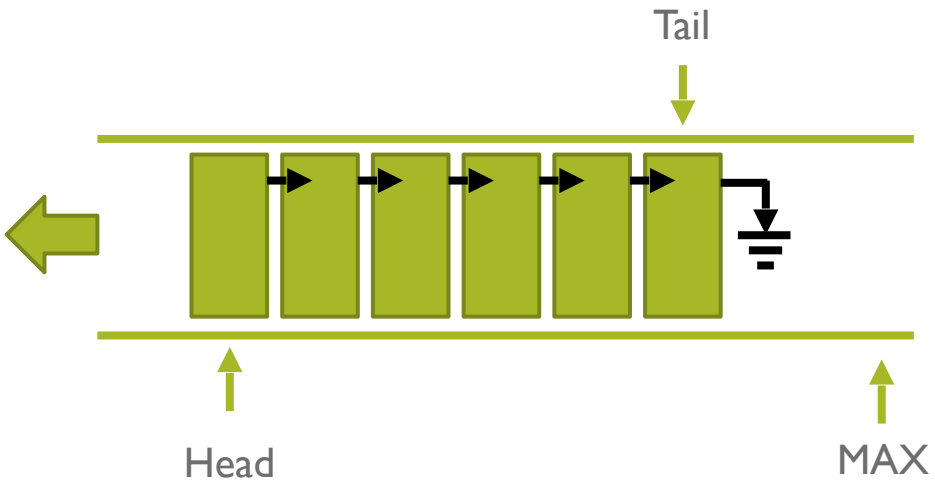


Cola doble con elementos.



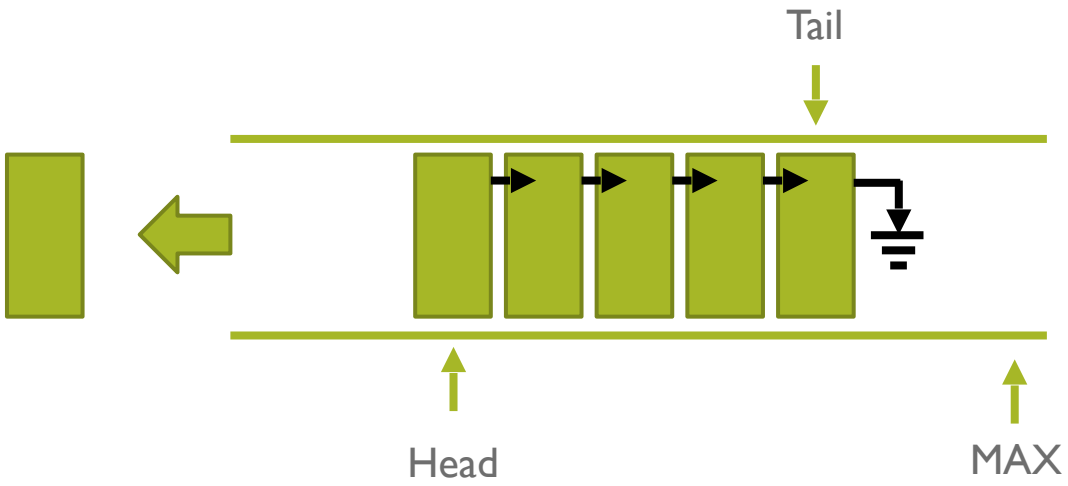
Cola doble con elementos.

DESENCOLAR POR H



Cola doble con elementos.

DESENCOLAR POR H



¿Aplicaciones?

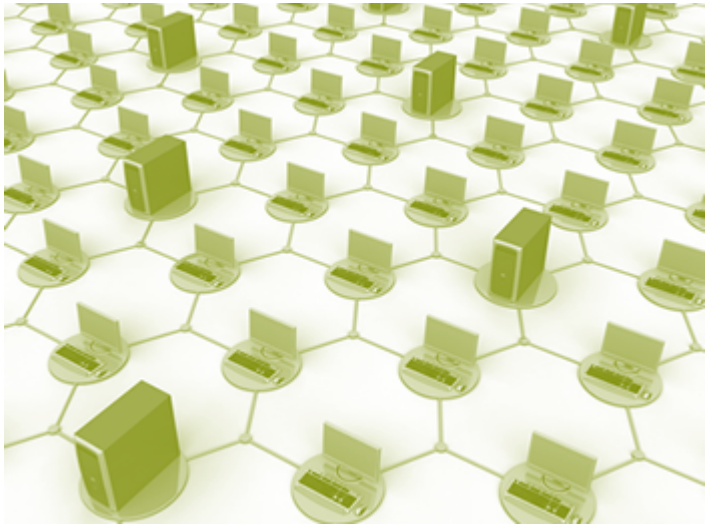


Figura 5. Transmisión de paquetes a través de la red.

Implementar la estructura de datos COLA DOBLE en lenguaje C. La COLA DOBLE debe permitir manipular nodos con las operaciones básicas ENCOLAR POR H, DESENCOLAR POR H, ENCOLAR POR T y DESENCOLAR POR T.

Implementar una función MOSTRAR para ver los elementos de la estructura de datos COLA DOBLE.

"No one in the brief history of computing has ever written a piece of perfect software. It's unlikely that you'll be the first."

Andy Hunt
(A writer of books on software development.)

ESTRUCTURAS DE DATOS LINEALES: COLA CIRCULAR Y COLA DOBLE.

Práctica 6

© Jorge A. Solano

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

Crear una aplicación tipo estructura de datos COLA CIRCULAR que permita encolar y desencolar nodos.

Crear una aplicación tipo estructura de datos COLA DOBLE que permita encolar y desencolar nodos tanto por head como por tail.

Especificaciones:

- La función main de las aplicaciones solo debe tener una llamada a otra función llamada menú, la cual debe tener las opciones descritas para cada Estructura de datos solicitadas.
- Todas las operaciones, incluyendo el menú, se programan en funciones distintas y en archivos distintos.

© Jorge A. Solano

1.3.4 Lista circular: almacenamiento contiguo y ligado, y operaciones.

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.

Una lista puede tener varias formas: puede ser simplemente ligada o doblemente ligada, puede estar ordenada o no, puede ser circular o no. Así mismo, las operaciones básicas dentro de una lista son BUSCAR, INSERTAR Y BORRAR.

Lista ligada.

Una lista ligada (o lista simple o lista simplemente ligada) está constituida por un conjunto de nodos alineados de manera lineal 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.

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).

Cada objeto debe contener una llave, la cuál debe fungir como un identificador dentro del conjunto de elementos. Así mismo, el objeto puede contener alguna otra información adicional.

Dado un elemento x en una lista, next[x] apuntaría al sucesor de x en la lista ligada. Si next[x] = NULL, el elemento x no tiene sucesor y, por ende, es el último elemento (o tail) de la lista. El atributo head[L] apunta al primer elemento de la lista. Si head[L] = x se puede afirmar que x es el primer elemento de la lista. Si head[L] = NULL entonces la lista está vacía.

Una lista posee 3 operaciones básicas para manipular los nodos que contiene: Insertar, buscar y borrar.

La operación insertar agrega un nuevo nodo al final de la lista, regresa verdadero (si lo pudo insertar) o falso (si no lo pudo insertar). La operación buscar recorre la lista comparando un parámetro predefinido del nodo, regresa el nodo (si lo encontró) o nulo (si no lo encontró). La función borrar permite eliminar de la lista el nodo deseado, regresar verdadero (si lo pudo eliminar) o falso (si no lo pudo eliminar).

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

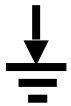
- Estructura vacía (caso extremo).
- Estructura con elemento(s) (caso base).

BUSCAR (L, k)

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.

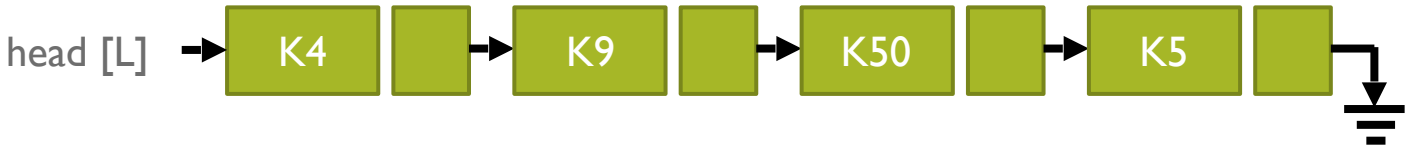
Lista simple vacía.

Head



Lista simple con elementos.

BUSCAR (L, k)

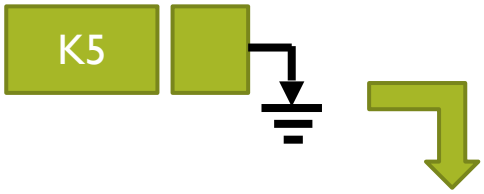


Si se ejecuta BUSCAR (L, K5), la función regresa una referencia al cuarto elemento de la lista. Si se ejecuta la función BUSCAR(L, K12), la función regresa NULO.

INSERTAR (L, x)

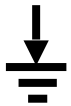
Dado un nodo x que contenga una llave k previamente establecida, el algoritmo agrega x al inicio de la lista.

Lista simple vacía.



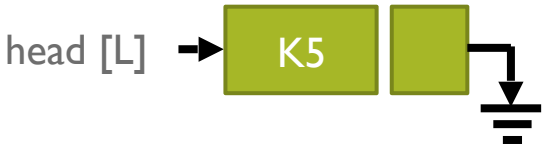
INSERTAR (L, x)

Head



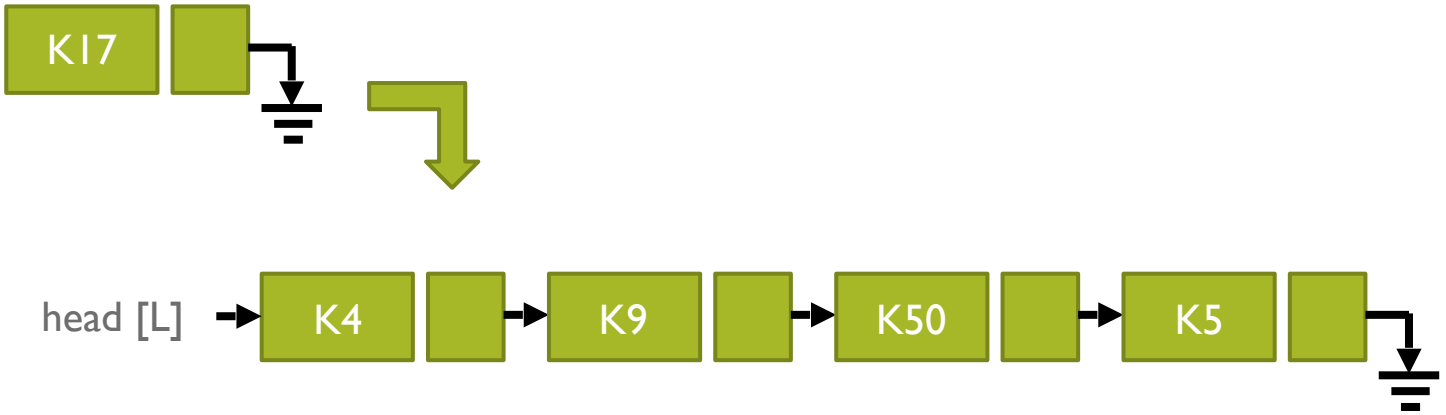
Lista simple vacía.

INSERTAR (L, x)



Lista simple con elementos.

INSERTAR (L, x)



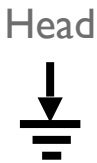
Lista simple con elementos.

INSERTAR (L, x)

BORRAR (L, x)

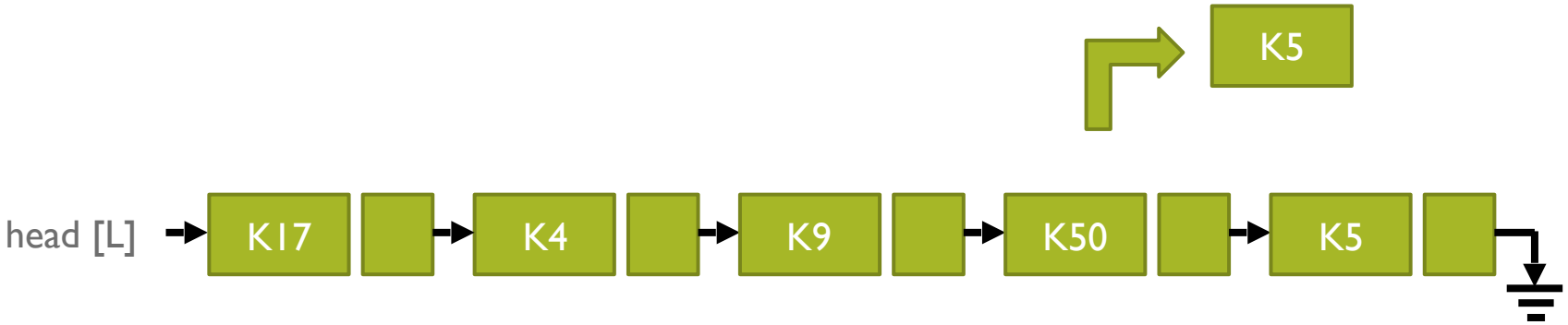
El algoritmo elimina el elemento x de la lista L. Para eliminar un elemento de la lista primero es necesario saber la ubicación del nodo a eliminar, por lo tanto, primero se debe realizar una búsqueda del elemento.

Lista simple vacía.



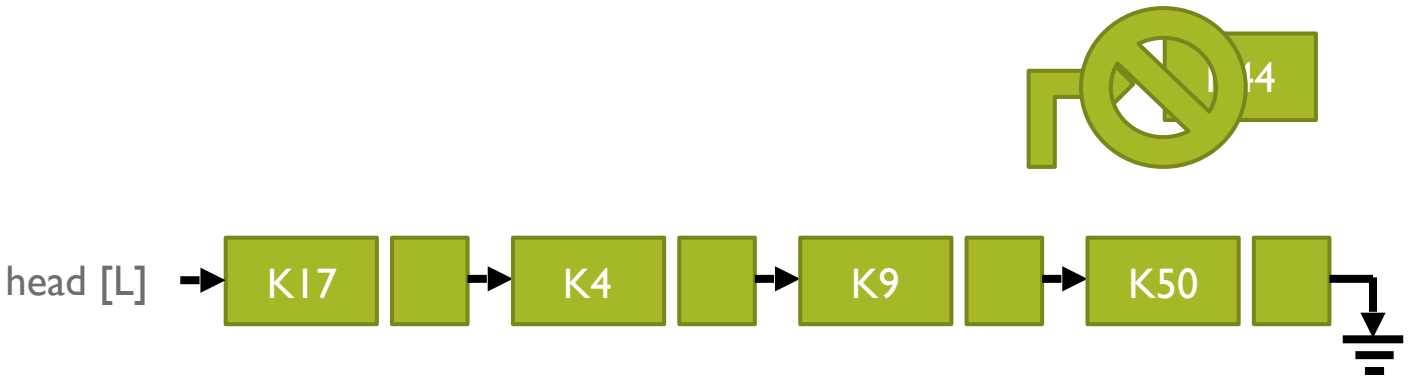
Lista simple con elementos.

BORRAR (L, x)



Lista simple con elementos.

BORRAR (L, x)



¿Aplicaciones?

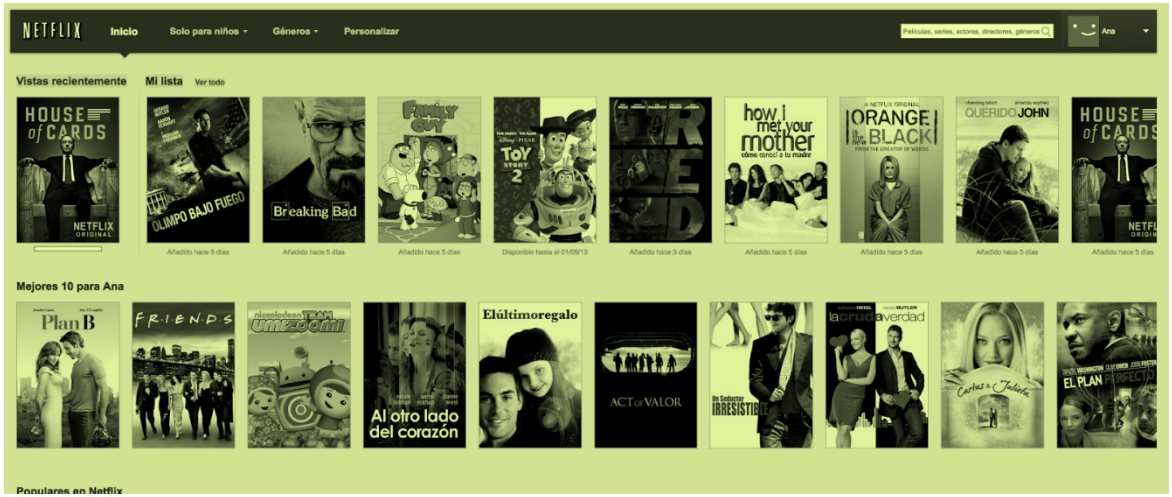


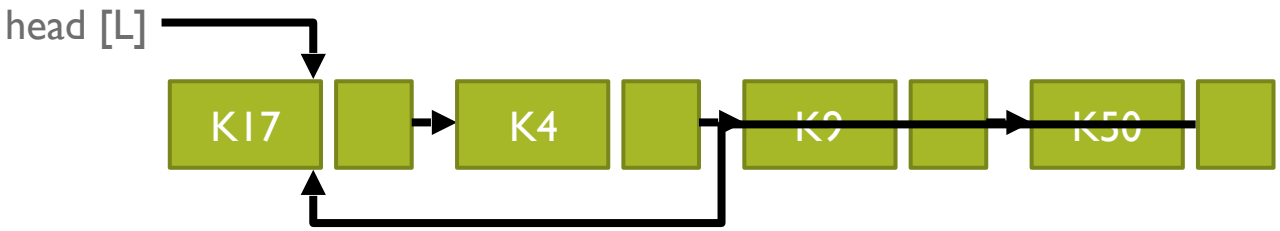
Figura 6. Funcionalidad de My list de Netflix.

Implementar la estructura de datos LISTA SIMPLEMENTE LIGADA en lenguaje C. La LISTA SIMPLEMENTE LIGADA debe permitir manipular nodos con las operaciones básicas BUSCAR, INSERTAR Y BORRAR, con base en una llave por cada nodo.

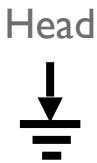
Implementar una función MOSTRAR para ver los elementos de la estructura de datos LISTA SIMPLEMENTE LIGADA.

Lista circular

Una lista circular es una lista simplemente ligada donde el apuntador del elemento que se encuentra al final de la lista tail[L] apunta al primer elemento de la lista head[L].

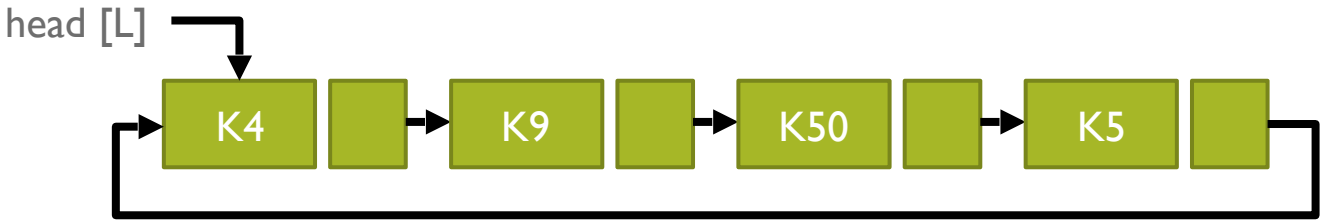


Lista circular vacía.



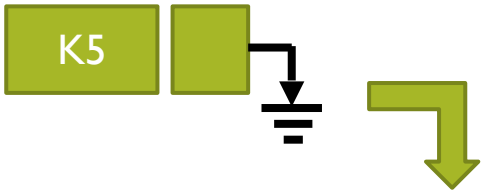
Lista circular con elementos.

BUSCAR (L, k)



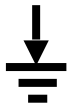
Si se ejecuta BUSCAR (L, K5), la función regresa una referencia al cuarto elemento de la lista. Si se ejecuta la función BUSCAR(L, K12), la función regresa NULO.

Lista circular vacía.



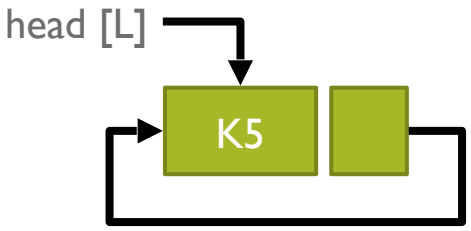
INSERTAR (L, x)

Head



Lista circular vacía.

INSERTAR (L, x)

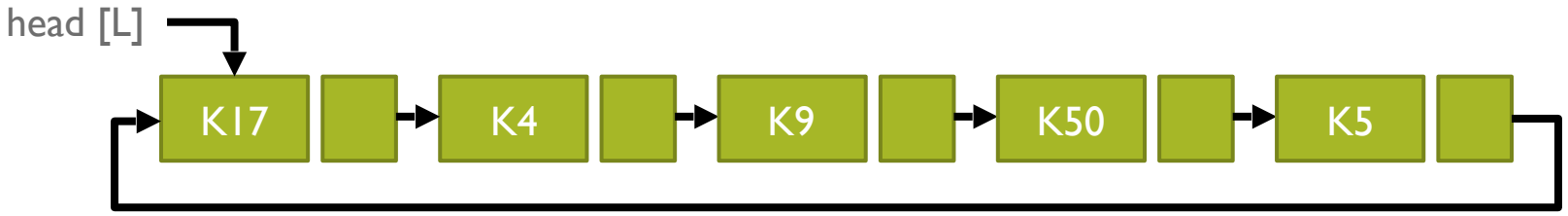


Lista circular con elementos.

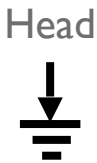
INSERTAR (L, x)

Lista circular con elementos.

INSERTAR (L, x)

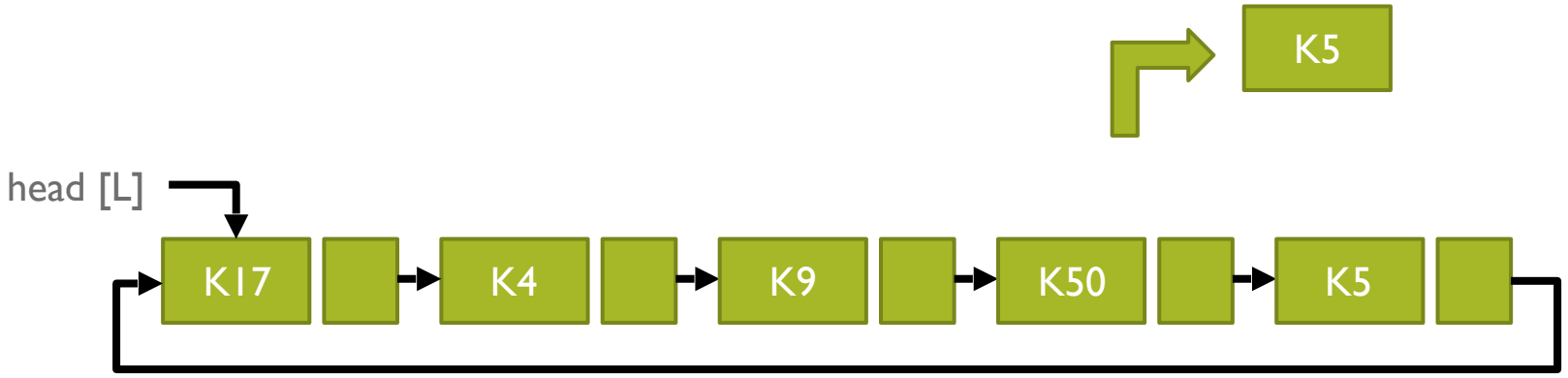


Lista circular vacía.



Lista circular con elementos.

BORRAR (L, x)

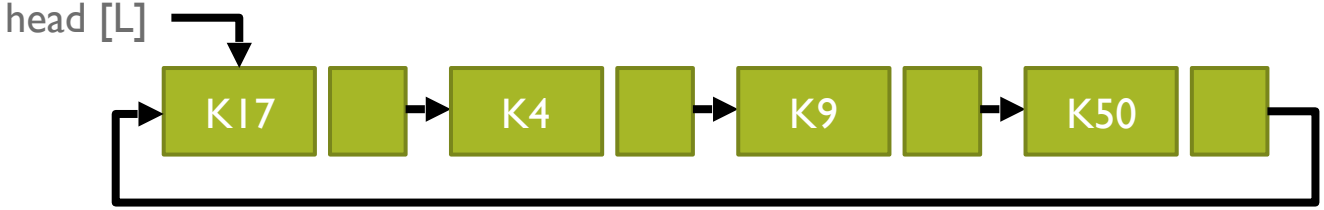


Lista circular con elementos.

BORRAR (L, x)

Lista circular con elementos.

BORRAR (L, x)



¿Aplicaciones?

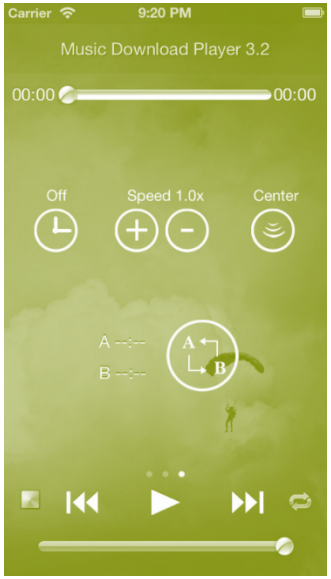


Figura 7. Lista de reproducción.

Implementar la estructura de datos LISTA SIMPLEMENTE LIGADA CIRCULAR en lenguaje C. La LISTA SIMPLEMENTE LIGADA CIRCULAR debe permitir manipular nodos con las operaciones básicas BUSCAR, INSERTAR Y BORRAR, con base en una llave por cada nodo.

Implementar una función MOSTRAR para ver los elementos de la estructura de datos LISTA SIMPLEMENTE LIGADA CIRCULAR.

"If it doesn't work, it doesn't matter how fast it doesn't work."

Mich Ravera
(Software Developer and Publisher, Bridge Director & Player, Private Pilot)

© Jorge A. Solano

ESTRUCTURAS DE DATOS LINEALES: LISTA SIMPLE Y LISTA CIRCULAR.

Práctica 7

© Jorge A. Solano

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

Crear una aplicación tipo estructura de datos LISTA SIMPLE que permita agregar, buscar y eliminar nodos.

Crear una aplicación tipo estructura de datos LISTA CIRCULAR que permita agregar, buscar y eliminar nodos.

Especificaciones:

- La función main de las aplicaciones solo debe tener una llamada a otra función llamada menú, la cual debe tener las opciones descritas para cada Estructura de datos solicitadas.
- Todas las operaciones, incluyendo el menú, se programan en funciones distintas y en archivos distintos.
- Todos los nodos que se agreguen a las listas se leen desde archivo de texto.

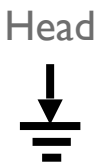
1.3.5 Listas doblemente ligadas: almacenamiento contiguo y ligado, y operaciones.

Una lista doblemente ligada tiene un apuntador a su sucesor (el siguiente nodo de la lista) y otro apuntador a su predecesor (el nodo anterior de la lista).

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, next[x] apunta al sucesor de x y prev[x] apunta al predecesor de x. Si prev[x] = NULL, el elemento x no tiene predecesor y, por ende, es el primer elemento (o head) de la lista. Si next[x] = NULL, el elemento x no tiene sucesor y, por ende, es el último elemento (o tail) de la lista. El atributo head[L] apunta al primer elemento de la lista, si head[L] = NULL entonces se puede afirmar que la lista está vacía.

Lista doblemente ligada vacía.



Lista doblemente ligada con elementos.

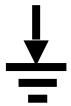
BUSCAR (L, k)

Si se ejecuta BUSCAR (L, K5), la función regresa una referencia al tercer elemento de la lista. Si se ejecuta la función BUSCAR(L, K12), la función regresa NULL. La búsqueda se puede iniciar en cualquier nodo y se puede recorrer la lista en cualquier dirección.

Lista doblemente ligada vacía.

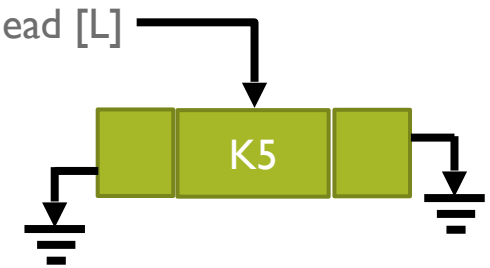
INSERTAR (L, x)

Head



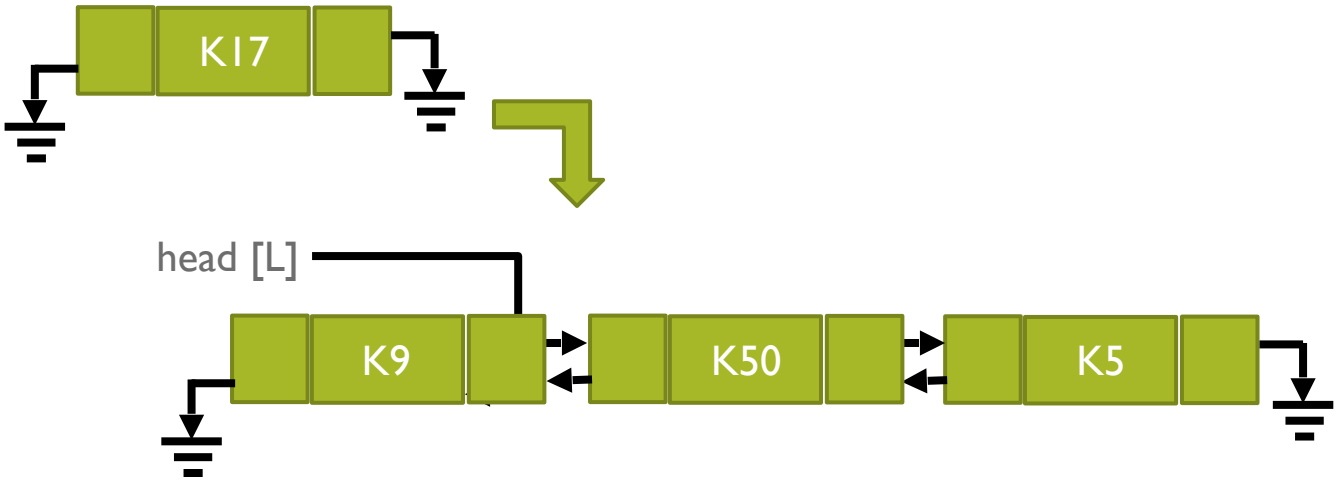
Lista doblemente ligada vacía.

INSERTAR (L, x)



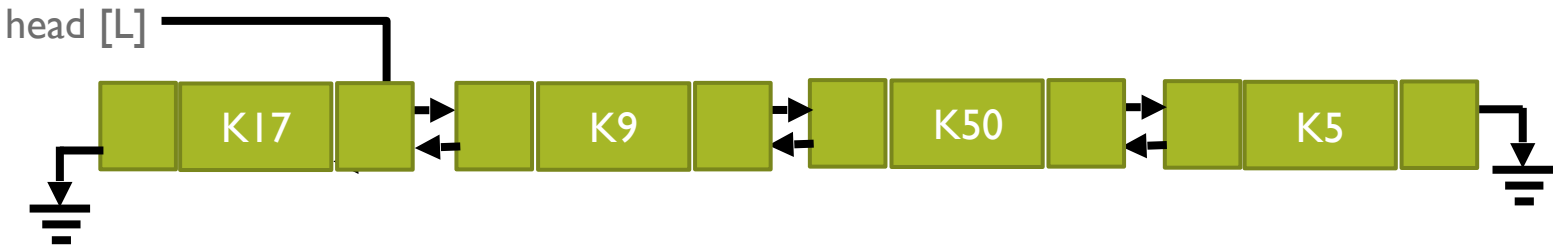
Lista doblemente ligada con elementos.

INSERTAR (L, x)

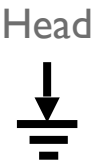


Lista doblemente ligada con elementos.

INSERTAR (L, x)

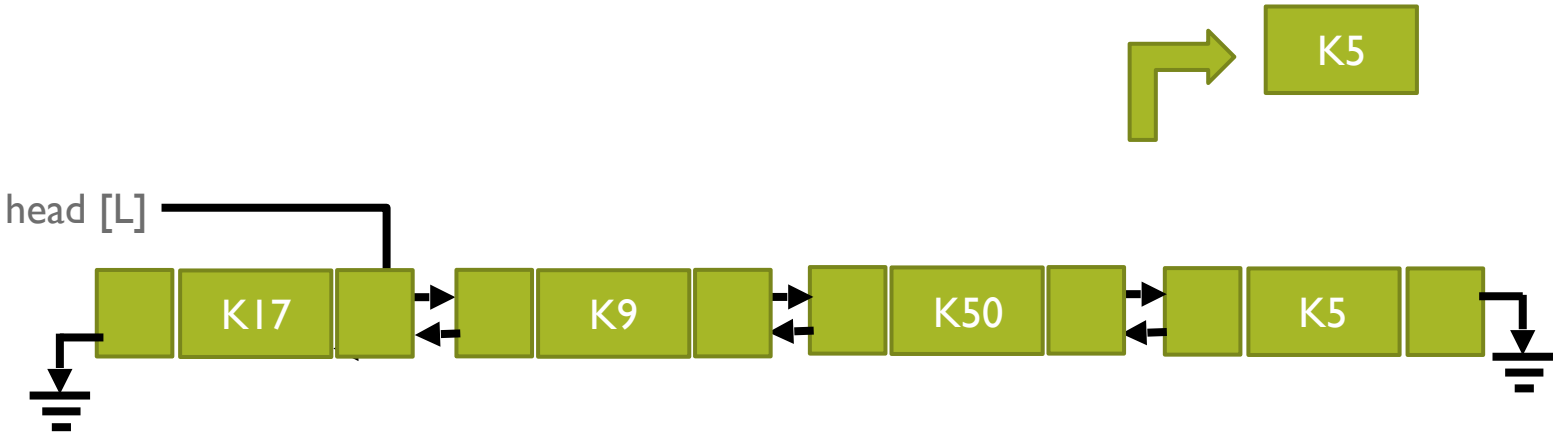


Lista doblemente ligada vacía.



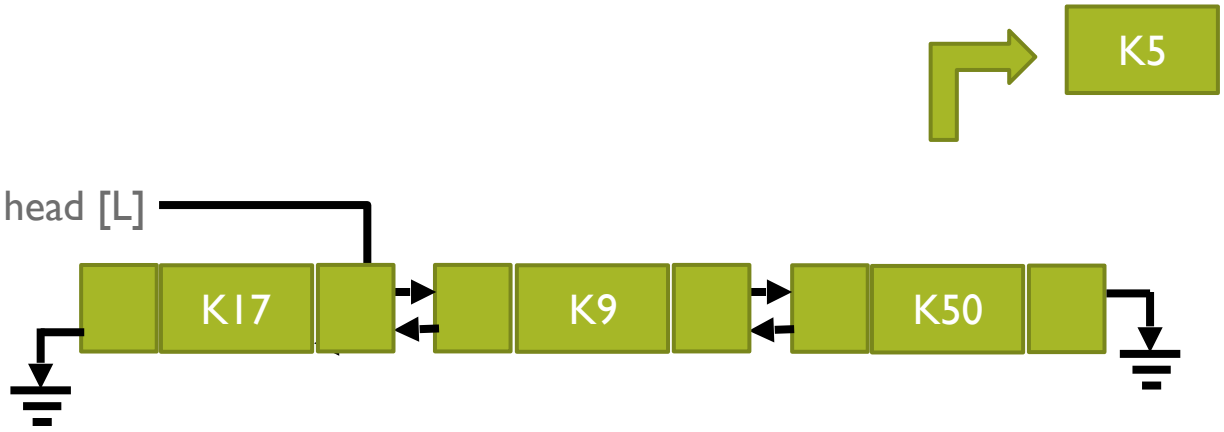
Lista doblemente ligada con elementos.

BORRAR (L, x)



Lista doblemente ligada con elementos.

BORRAR (L, x)



Lista doblemente ligada con elementos.

BORRAR (L, x)

¿Aplicaciones?

Figura 8. Batman LEGO.

Implementar la estructura de datos LISTA DOBLEMENTE LIGADA en lenguaje C. La LISTA DOBLEMENTE LIGADA debe permitir manipular nodos con las operaciones básicas BUSCAR, INSERTAR Y BORRAR, con base en una llave por cada nodo.

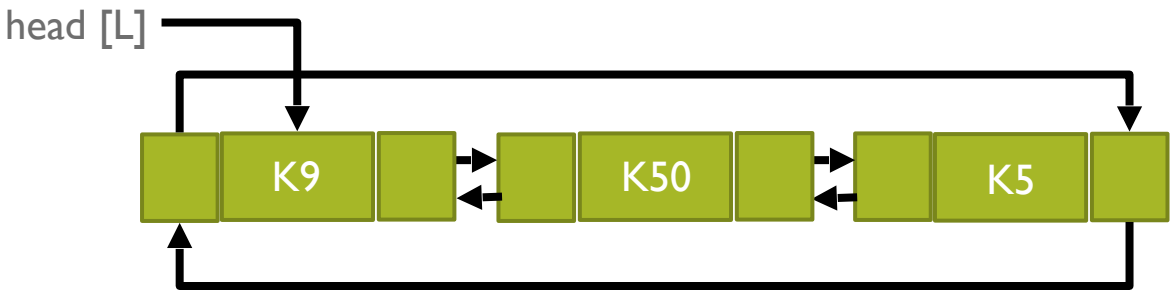
Implementar una función MOSTRAR para ver los elementos de la estructura de datos LISTA DOBLEMENTE LIGADA.

"Testing can show the presence of errors, but no their absence."

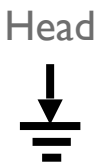
E. W. Dijkstra
(Was a Dutch systems scientist, programmer, software engineer, science essayist, and early pioneer in computing science.)

Lista doblemente ligada circular

Es una lista doblemente ligada que tiene la característica de que la referencia siguiente del último elemento apunta al primer elemento y la referencia previa del primer elemento apunta al último.

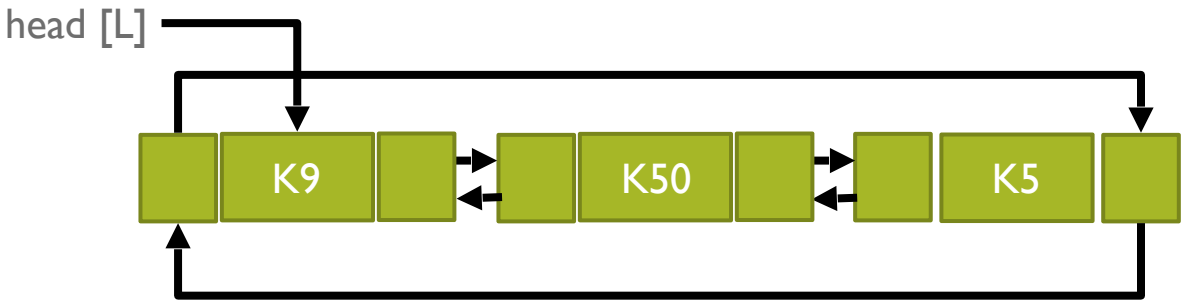


Lista doblemente ligada circular vacía.



Lista doblemente ligada circular con elementos.

BUSCAR (L, k)

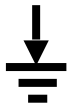


Si se ejecuta BUSCAR (L, K5), la función regresa una referencia al tercer elemento de la lista. Si se ejecuta la función BUSCAR(L, K12), la función regresa NULL.La búsqueda se puede iniciar en cualquier nodo y se puede recorrer la lista en cualquier dirección.

Lista doblemente ligada circular vacía.

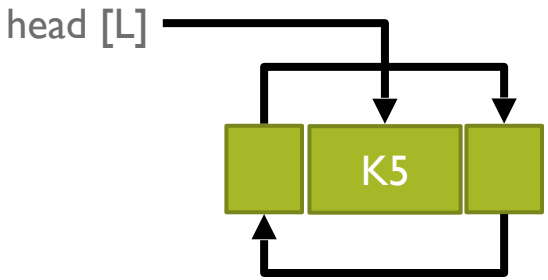
INSERTAR (L, x)

Head



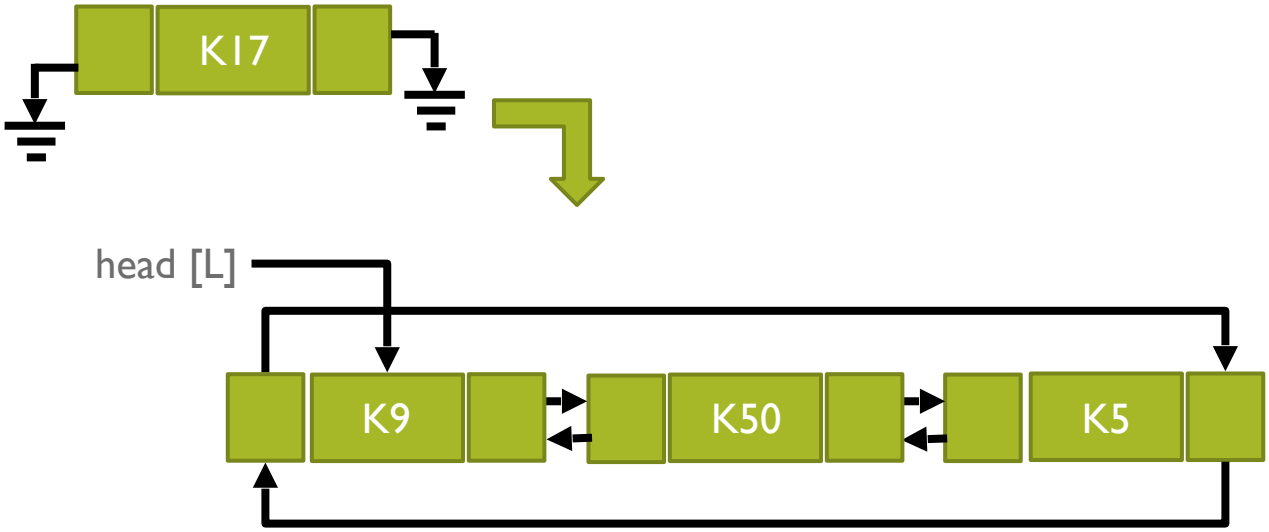
Lista doblemente ligada circular vacía.

INSERTAR (L, x)



Lista doblemente ligada circular con elementos.

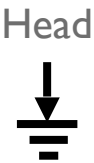
INSERTAR (L, x)



Lista doblemente ligada circular con elementos.

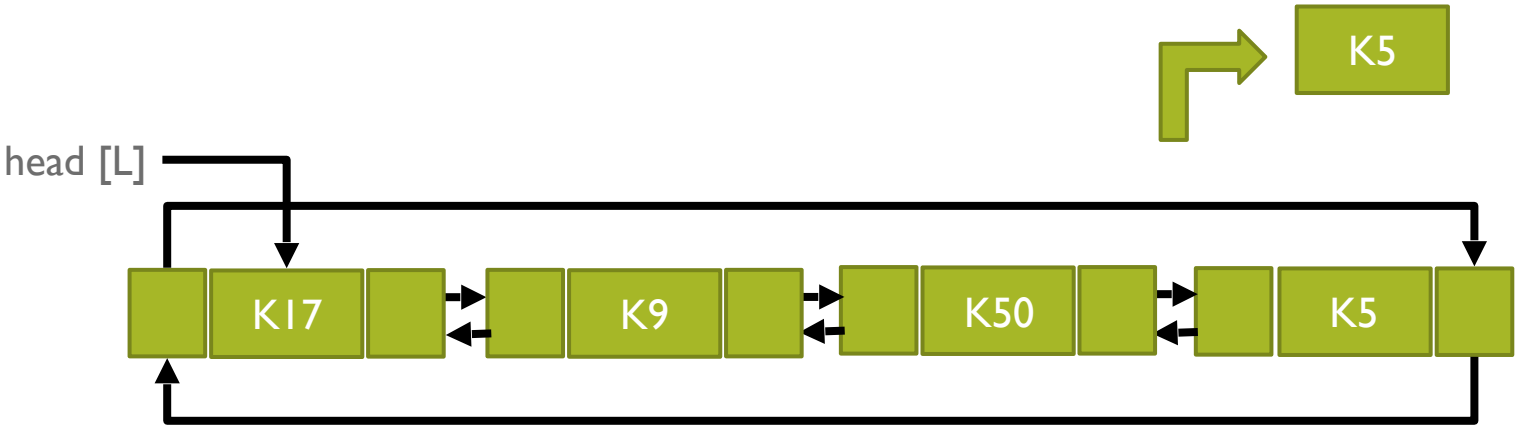
INSERTAR (L, x)

Lista doblemente ligada circular vacía.



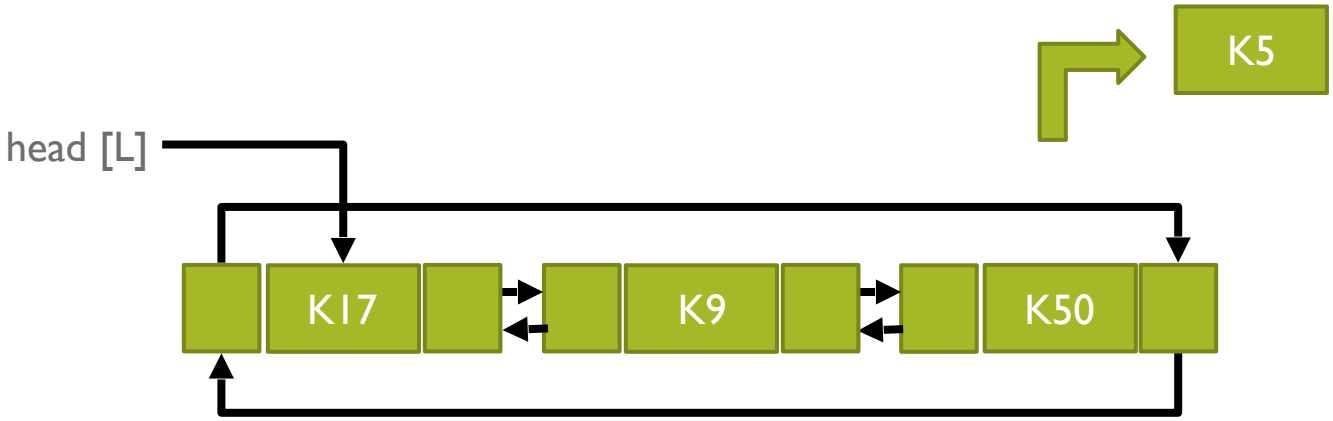
Lista doblemente ligada circular con elementos.

BORRAR (L, x)



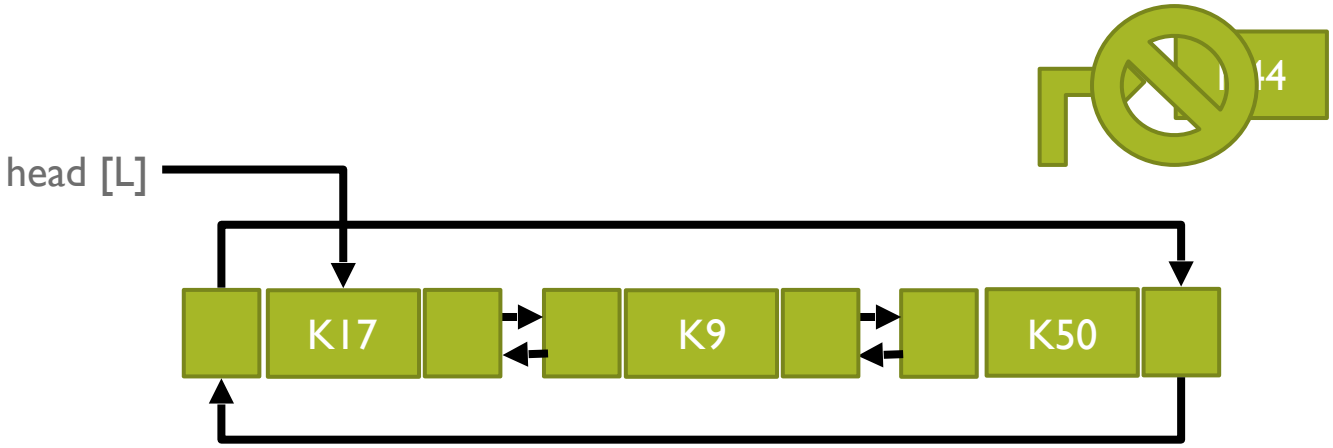
Lista doblemente ligada circular con elementos.

BORRAR (L, x)



Lista doblemente ligada circular con elementos.

BORRAR (L, x)



¿Aplicaciones?

Figura 9. Resident evil.

Implementar la estructura de datos LISTA DOBLEMENTE LIGADA CIRCULAR en lenguaje C. La LISTA DOBLEMENTE LIGADA CIRCULAR debe permitir manipular nodos con las operaciones básicas BUSCAR, INSERTAR Y BORRAR, con base en una llave por cada nodo.

Implementar una función MOSTRAR para ver los elementos de la estructura de datos LISTA DOBLEMENTE LIGADA CIRCULAR.

"Beware of bugs in the above code; I have only proved it correct, not tried it."

Donald Knuth
(American computer scientist, mathematician, and professor emeritus at Stanford University.)

ESTRUCTURAS DE DATOS LINEALES: LISTA DOBLEMENTE LIGADA Y LISTA DOBLEMENTE LIGADA CIRCULAR.

Práctica 8

© Jorge A. Solano

8. Estructuras de datos lineales: Lista doblemente ligada y lista doblemente ligada circular.

Crear una aplicación tipo estructura de datos LISTA DOBLEMENTE LIGADA que permita agregar, buscar y eliminar nodos.

Crear una aplicación tipo estructura de datos LISTA DOBLEMENTE LIGADA CIRCULAR que permita agregar, buscar y eliminar nodos.

Especificaciones:

- La función main de las aplicaciones solo debe tener una llamada a otra función llamada menú, la cual debe tener las opciones descritas para cada Estructura de datos solicitadas.
- Todas las operaciones, incluyendo el menú, se programan en funciones distintas y en archivos distintos.
- Todos los nodos que se agreguen a las listas se leen desde archivo de texto.

Dentro de las estructuras de datos lineales se pueden identificar fortalezas y debilidades.

Estructura	Ventajas	Desventajas
Arreglo	Rápida inserción. Acceso rápido a la información si se conoce el índice.	Búsqueda lenta. Eliminación lenta. Tamaño fijo.
Pila	Provee acceso directo al último elemento insertado.	Acceso lento a otros elementos.
Cola	Provee acceso directo al primer elemento insertado.	Acceso lento a otros elementos.
Lista ligada	Rápida inserción. Rápida eliminación.	Búsqueda lenta.

"First do it, then do it right, then do it better."

Addy Osmani
(Is an engineering manager at Google working on Chrome.)

1 Estructura de datos

Objetivo: Resolver problemas de almacenamiento, recuperación y ordenamiento de datos y las técnicas de representación más eficientes, utilizando las estructuras para representarlos.

1.1 Representación de datos en memoria.

- 1.1.1 Tipos primitivos.
- 1.1.2 Arreglos.
- 1.1.3 Apuntadores.
- 1.1.4 Tipo de dato abstracto.

1.2 Administración del almacenamiento en tiempo de ejecución.

1.3 Estructura de datos compuestos.

- 1.2.2 Pila: almacenamiento contiguo y ligado, y operaciones.
- 1.2.3 Cola: almacenamiento contiguo y ligado, y operaciones.
- 1.2.4 Cola doble: almacenamiento contiguo y ligado, y operaciones.
- 1.2.5 Listas circular: almacenamiento contiguo y ligado, y operaciones.
- 1.2.6 Listas doblemente ligadas: almacenamiento contiguo y ligado, y operaciones.