



Facultad de Ingeniería en
Ciencias de la Computación y Telecomunicaciones
U.A.G.R.M.



Somos ingeniería!

UNIDAD I

MODELOS DE REPRESENTACION DE ESTRUCTURAS DE DATO

Ing. Mario M. López Winnipeg

1. MODELOS DE REPRESENTACION DE ESTRUCTURAS DE DATOS

1.0 Introducción

1.1 Abstracción

1.2 Estructura de datos

1.3 Tipos de datos abstractos

1.4 Formas de Implementación

1.4.1 Modelo estático

1.4.2 Modelo dinámico

1.4.3 Modelo persistente

1.4.4 Modelo simulado

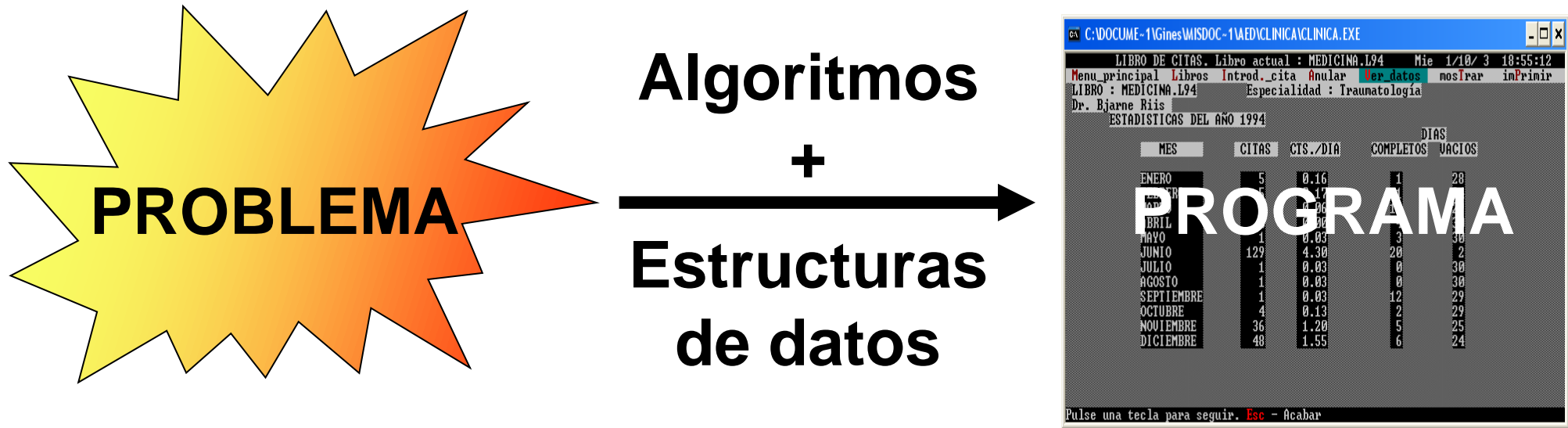
1.0 INTRODUCCION

Objetivo central

Aplicar los conceptos de estructuras de datos y sus algoritmos de manipulación, para la implementación de estructuras de datos clásicas y creación de nuevas estructuras en la solución de problemas.

1.0 INTRODUCCION

Problemas, programas, algoritmos y estructuras de datos



- **Problema:** Conjunto de hechos o circunstancias que dificultan la consecución de algún fin.
- **Algoritmo:** Conjunto de reglas finito e inambiguo.
- **Estructura de datos:** Disposición en memoria de la información.
- **Programa:** Algoritmos + Estructuras de datos.

1.1 ABSTRACCIÓN

¿QUÉ ES LA ABSTRACCIÓN DE DATOS?

- **La abstracción de datos** es una técnica o metodología que permite diseñar estructuras de datos.
 - Consiste básicamente en *representar* bajo ciertos lineamientos de formato las *características esenciales* de una estructura de datos.
 - Este proceso de diseño se *olvida de los detalles específicos de implementación* de los datos.
-

1.1 ABSTRACCION

EN QUE SE USA ABSTRACCION

- Procedimientos y funciones son **abstracciones de control**
- Los tipos definidos por el usuario son **abstracciones de datos**
- Las unidades, módulos o paquetes son abstracciones de nivel superior: **abstracciones de funcionalidades**

1.1 ABSTRACCION

CONCLUSIONES

- Es una tecnica poderosa de programacion que permite inventar o definir nuevos tipos de datos observando e identificando entidades del mundo real(objetos) ocultando datos irrelevantes para la solucion del problema.
- Gracias a esta tecnica se pueden diseñar progamas mas cortos , flexibles y legibles.
- Estos nuevos tipos de datos se conocen como **Tipos de Datos Abstractos TDA**

1.2 ESTRUCTURA DE DATOS

¿QUÉ ES ESTRUCTURA DE DATOS?

Una estructura de datos es básicamente un grupo de elementos de datos que se agrupan bajo un nombre, y que define una forma particular de almacenar y organizar datos en una computadora para que pueda ser utilizada eficientemente.

Data Structures using Reema Thareja pag 43 ultimo párrafo

1.2 ESTRUCTURA DE DATOS - CLASIFICACION

Las estructuras de datos primitivas son los tipos de datos fundamentales que son compatibles con una programación. Algunos tipos de datos básicos son enteros, reales, caracteres y booleanos. Etc.

Las estructuras de datos no primitivas son aquellas estructuras de datos que se crean utilizando datos primitivos. Los ejemplos de tales estructuras de datos incluyen listas, pilas, conjunto, etc. se pueden clasificar en dos categorías: lineal y no lineal

Lineal: Si los elementos de una estructura de datos se almacenan en un orden lineal o secuencial, entonces pertenece a la categoría lineal

No Lineal: Si los elementos de una estructura de datos no se almacenan en un orden secuencial, entonces es una estructura de datos no lineal

Data Structures using Reema Thareja pag 46

1.3 TIPO DE DATO ABSTRACTO (TDA)

- Es la representación de una entidad u objeto para facilitar su programación. Se compone de:
 - **Estructura de datos**: Es la estructura de programación que se selecciona para representar las características de la entidad modelada
 - **Funciones de Abstracción**: Son funciones que permiten hacer uso de la estructura de datos, y que esconden los detalles de dicha estructura, permitiendo un mayor nivel de abstracción.
-

1.3 TIPO DE DATO ABSTRACTO (TDA)

- Se plasma la abstracción realizada al diseñar una estructura de datos, esto pasa a ser el mapa o plano con el cual se construirá la estructura de datos y se definirán claramente las reglas en las que podrá usarse el TDA.
 - La especificación lógica de un TDA consiste de los siguientes cuatro puntos:
-

1.3 TIPO DE DATO ABSTRACTO (TDA)

I. Elementos que conformarán la estructura de datos.

Es el tipo de los datos que se guardará en la estructura.

Ejemplo:

números enteros, caracteres, fechas, registros con los datos de un empleado, etcétera.

2. Tipo de organización en que se guardarán los elementos.

Existen cuatro tipos de organización para los datos en la estructura.

- **Lineal**: Si hay una relación de uno a uno entre los elementos.
- **Jerárquica**: Si hay una relación de uno a muchos entre los elementos.
- **Red**: Si hay una relación de muchos a muchos entre los elementos.
- **Sin relación**: Si no hay relaciones entre los elementos

1.3 TIPO DE DATO ABSTRACTO (TDA)

3. Dominio de la estructura.

Este punto es opcional, y en él se describirá la capacidad de la estructura en cuanto al rango posible de datos por guardar.

4. Descripción de las operaciones de la estructura.

- Cada operación que está relacionada con la estructura debe describirse como:
 - Nombre de la operación.
 - Descripción breve de su utilidad.
 - Datos de entrada a la operación.
 - Datos que genera como salida la operación.
 - Precondición: Condición que deberá cumplirse antes de utilizar la operación para que se realice sin problemas.
 - Poscondición: Condición en que queda el TDA después de ejecutar la operación.
-

1.3 TIPO DE DATO ABSTRACTO (TDA)

- *Especificación lógica del TDA: Cadena*

- **Elementos:** todos los caracteres alfabéticos (letras mayúsculas y minúsculas), caracteres numéricos y caracteres especiales.
- **Estructura:** hay una relación lineal entre los caracteres
- **Dominio:** existen entre 0 y 80 caracteres en cada valor del TDA CADENA. El dominio serán todas aquellas secuencias de caracteres que cumplan con las reglas.

Ejemplo



1.3 TIPO DE DATO ABSTRACTO (TDA)

operación...

Ejemplo

Agregafinal

- Utilidad: Sirve para agregar un carácter al final de una cadena.
- Entrada: Cadena S y el carácter L, que se añadirá a la cadena S.
- Salida: Cadena S modificada.
- Precondición: La cantidad de caracteres en S es menor que 80.
- Poscondición: La cadena S tiene el carácter L que queda al extremo derecho de la cadena.

Llena

- Utilidad: Sirve para verificar si una cadena está llena o no.
- Entrada: .Cadena S que será verificada.
- Salida: VERDADERO si la cadena S contiene ya 80 caracteres, FALSO en caso contrario
- Precondición: Ninguna
- Poscondición: Ninguna (pues la cadena S no se modifica).

1.3 TIPO DE DATO ABSTRACTO (TDA)

En la abstracción de datos se pueden definir tres niveles de trabajo:

- 1. El nivel **lógico o abstracto** se define la estructura de datos y las operaciones relacionadas con ella. La descripción es independiente del lenguaje de programación en el se usará la estructura.
 - 2. El nivel **físico** o de implementación. En este nivel se decide el lenguaje de programación para la implementación, los tipos de datos ya definidos servirán para representarla y se implementarla
 - 3. En el nivel **aplicación** o de uso el programador usará el TDA para resolver determinada aplicación. El uso del TDA se limita a llamar las operaciones **sobre la estructura que se requiera cuidando siempre de cumplir con las reglas de cada operación especificadas en el nivel lógico.**
-

1.4 FORMAS DE IMPLEMENTACIÓN

Un *Tipo de dato abstracto* (en adelante *TDA*) es un conjunto de datos u objetos al cual se le asocian *operaciones*.

El TDA provee de una interfaz con la cual es posible realizar las operaciones permitidas, **abstrayéndose de la manera en como estén implementadas** dichas operaciones. Esto quiere decir que **un mismo TDA puede ser implementado utilizando distintas estructuras de datos** y proveer la misma funcionalidad.

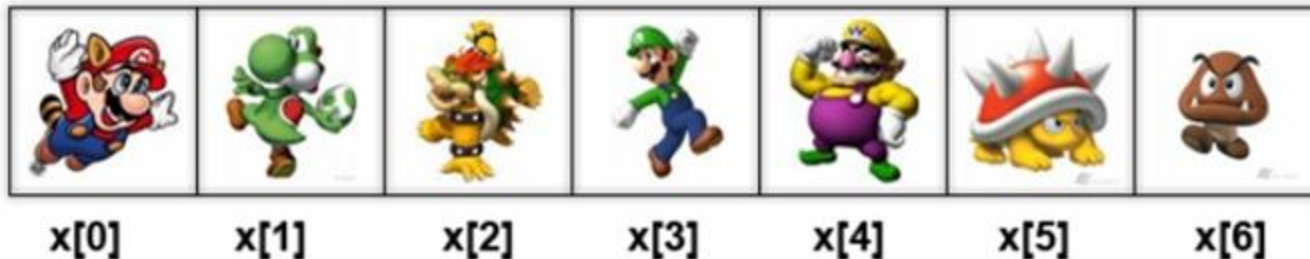
1.4.1 MODELO ESTÁTICO DE IMPLEMENTACIÓN

Arreglos

¿Que es un arreglo?

Los arreglos son una coleccion de variables del mismo tipo que se referencian utilizando un nombre comun. Un arreglo consta de posiciones de memoria contigua. La dirección más baja corresponde al primer elemento y la más alta al último. Un arreglo puede tener una o varias dimensiones. Para acceder a un elemento en particular de un arreglo se usa un índice.

struct mario x[7];



1.4.1 MODELO ESTÁTICO DE IMPLEMENTACIÓN

Arreglos

¿String?

Los llamados strings en C no existen como tal, son solo un arreglo de caracteres, los cuales tienen como única diferencia el carácter de corte en el último elemento `\0`. Deben recordar que la posición comienza desde 0 y que cada elemento `char` utiliza un byte de memoria. Si necesitan exactamente 10 letras deben declarar el arreglo con 11 direcciones de memorias asignadas.

`char x[7];`

| | | | | | | |
|-------------------|-------------------|-------------------|-------------------|-------------------|-------------------|-------------------|
| Y | A | G | A | M | I | \0 |
| <code>x[0]</code> | <code>x[1]</code> | <code>x[2]</code> | <code>x[3]</code> | <code>x[4]</code> | <code>x[5]</code> | <code>x[6]</code> |

1.4.1 MODELO ESTÁTICO DE IMPLEMENTACIÓN

Se puede implementar TDA usando simplemente datos primitivos o basicas que tenga el lenguaje


ENTEROS

CARACTERES

BITWISE

REALES

1.4.2 MODELO DINAMICO DE IMPLEMENTACIÓN

- Los apunadores pueden ser de cualquier tipo, se declaran anteponiendo un *:
 - `int *puntero; //apuntador a un entero`
 - `char *cad; //apuntador a carácter`
 - `void *puntero; //puntero a cualquier tipo de dato`
 - El operador & se utiliza para obtener la dirección de una variable.
 - En C++ se utiliza la función **new** para pedir memoria y **delete** para liberarla
 - Investigar para que se usa  en c++
-

1.4.3 MODELO PERSISTENTE DE IMPLEMENTACIÓN

Esta forma de implementación consiste en que los TDA que se implementen persistentes en el tiempo deberán usar archivos para que puedan perdurar en su estado.

1.4.4 MODELO SIMULADO DE IMPLEMENTACIÓN

Consiste en implementar la clase Smemoria que simula el comportamiento de la memoria

Constantes MAX = 20 NULO = -1

NodoM que tiene

Dato TipoDato (Entero)

Link Direccion de Memoria(Entero)

clase **CSMemoria**

Atributos

MEM arreglo(MAX) de tipo **NodoM**

libre Direccion de memoria (entero)

Metodos

Constructor

direccion **new_espacio(cant)**

Delete_espacio(dir)

poner_dato(dir , lugar, valor)

TipoDato **obtenerDato(dir, lugar)**

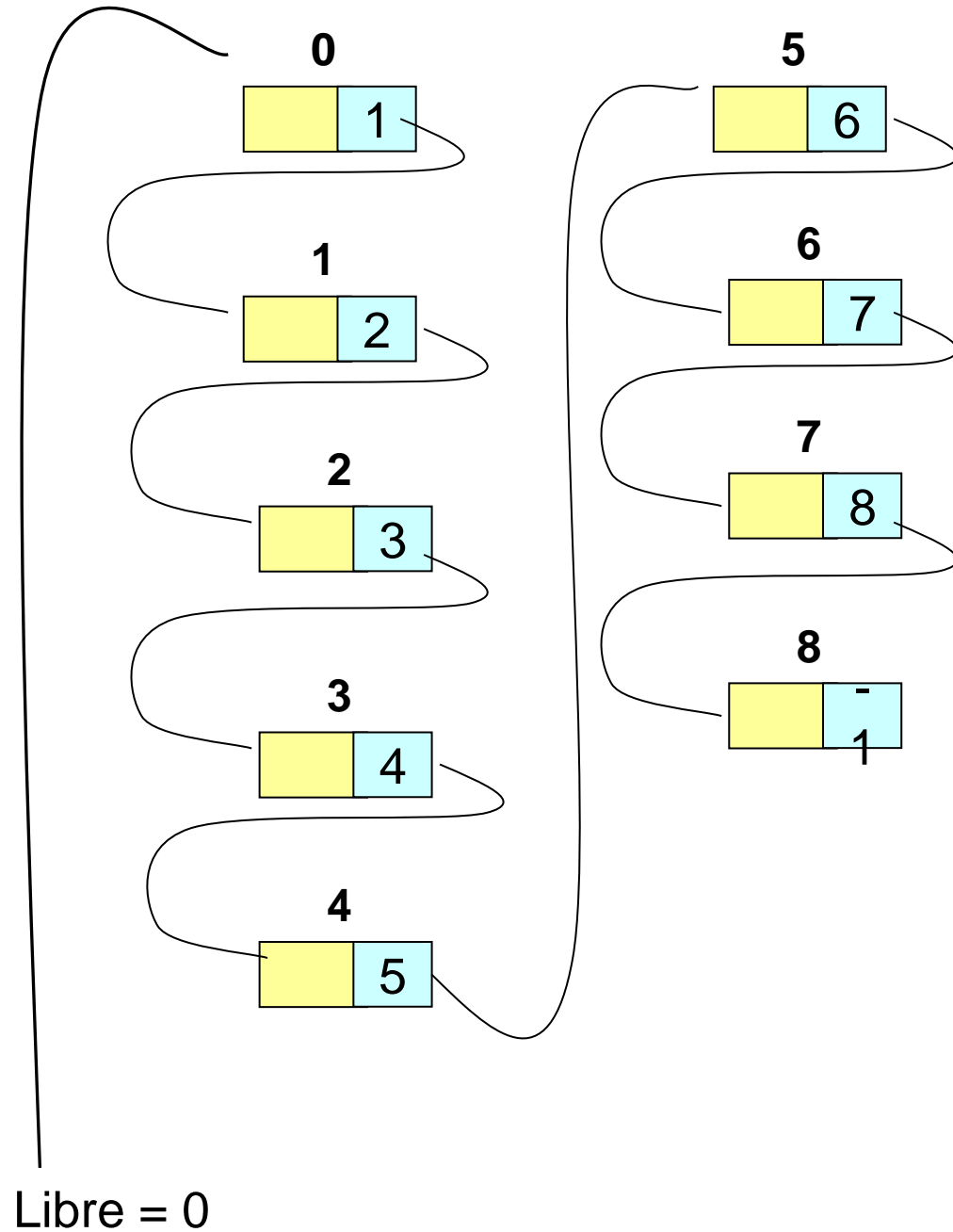
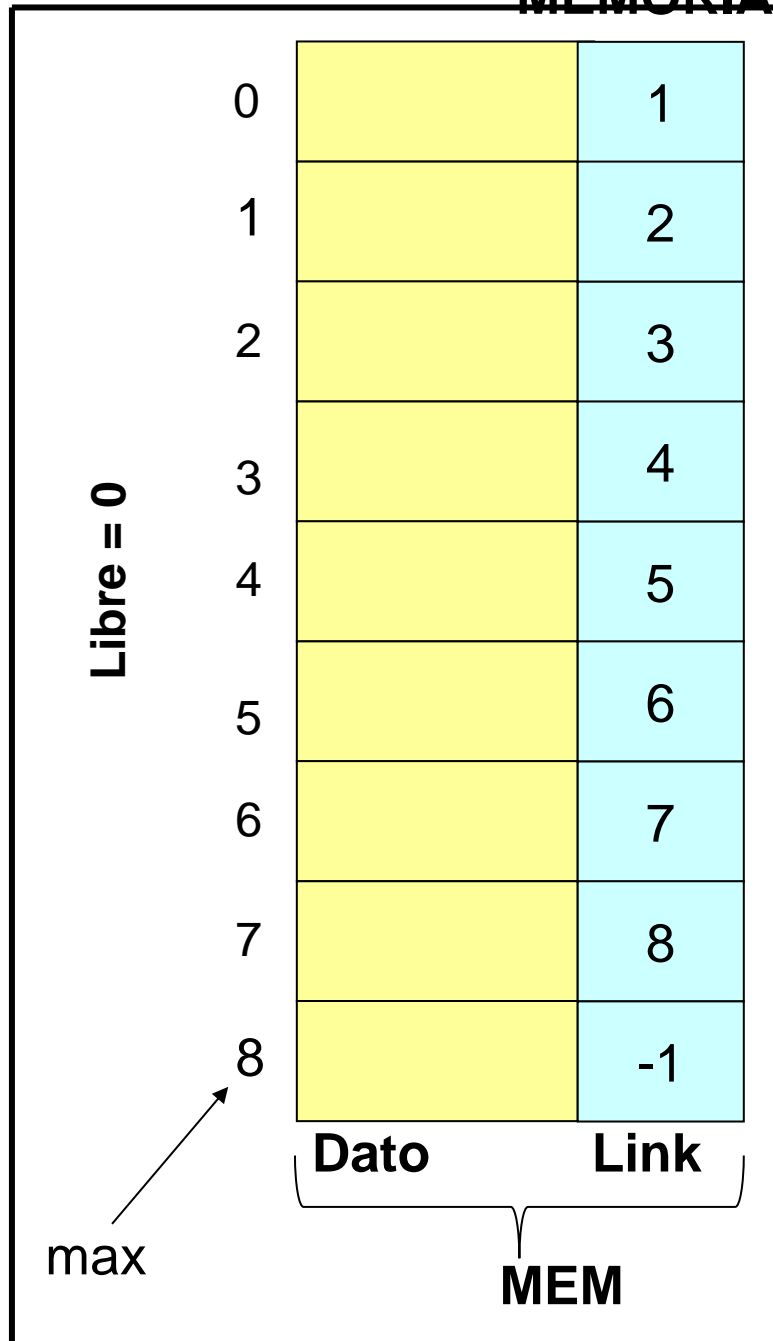
entero

Espacio_Disponible()

entero **Espacio_ocupado()**

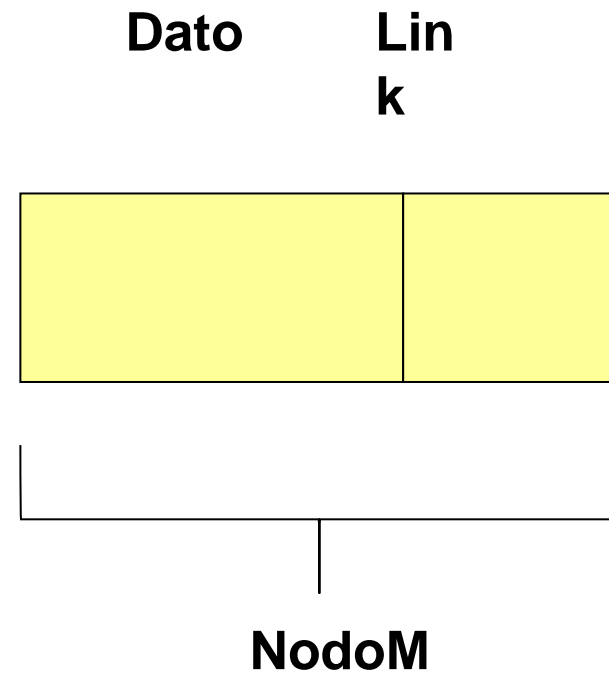
booleano **Estado(dir)**

IMPLEMENTANDO LA CLASE MEMORIA



DEFINIENDO LA ESTRUCTURA DE LA ZONA DATO

| | | | |
|-----------|---|------|------|
| Libre = 0 | 0 | | 1 |
| | 1 | | 2 |
| | 2 | | 3 |
| | 3 | | 4 |
| | 4 | | 5 |
| | 5 | | 6 |
| | 6 | | 7 |
| | 7 | | 8 |
| | 8 | | -1 |
| | | Dato | Link |
| MEM | | | |



IMPLEMENTANDO METODOS DE LA CLASE MEMORIA

CONSTRUCTOR CSMEMORIA::CREAR()

INICIO

PARA CADA **I** DESDE **0** HASTA **max**

MEM[**I**].LINK = **I**+1

MEM[**max**].LINK = **-1**

LIBRE = 0

FIN

MEM

Libre = 0

| | | |
|---|--|----|
| 0 | | 1 |
| 1 | | 2 |
| 2 | | 3 |
| 3 | | 4 |
| 4 | | 5 |
| 5 | | 6 |
| 6 | | 7 |
| 7 | | 8 |
| 8 | | -1 |

Dato

Link

IMPLEMENTANDO METODOS DE LA CLASE MEMORIA

DIRECCION CSMEMORIA::NEW_ESPACIO(CANT)

INICIO

DIR = LIBRE

D = LIBRE

PARA CADA I = 1 HASTA CANT -1

D = MEM [D].LINK

LIBRE=MEM [D].LINK

MEM [D].LINK = -1

// RETORNAR DIR

FIN

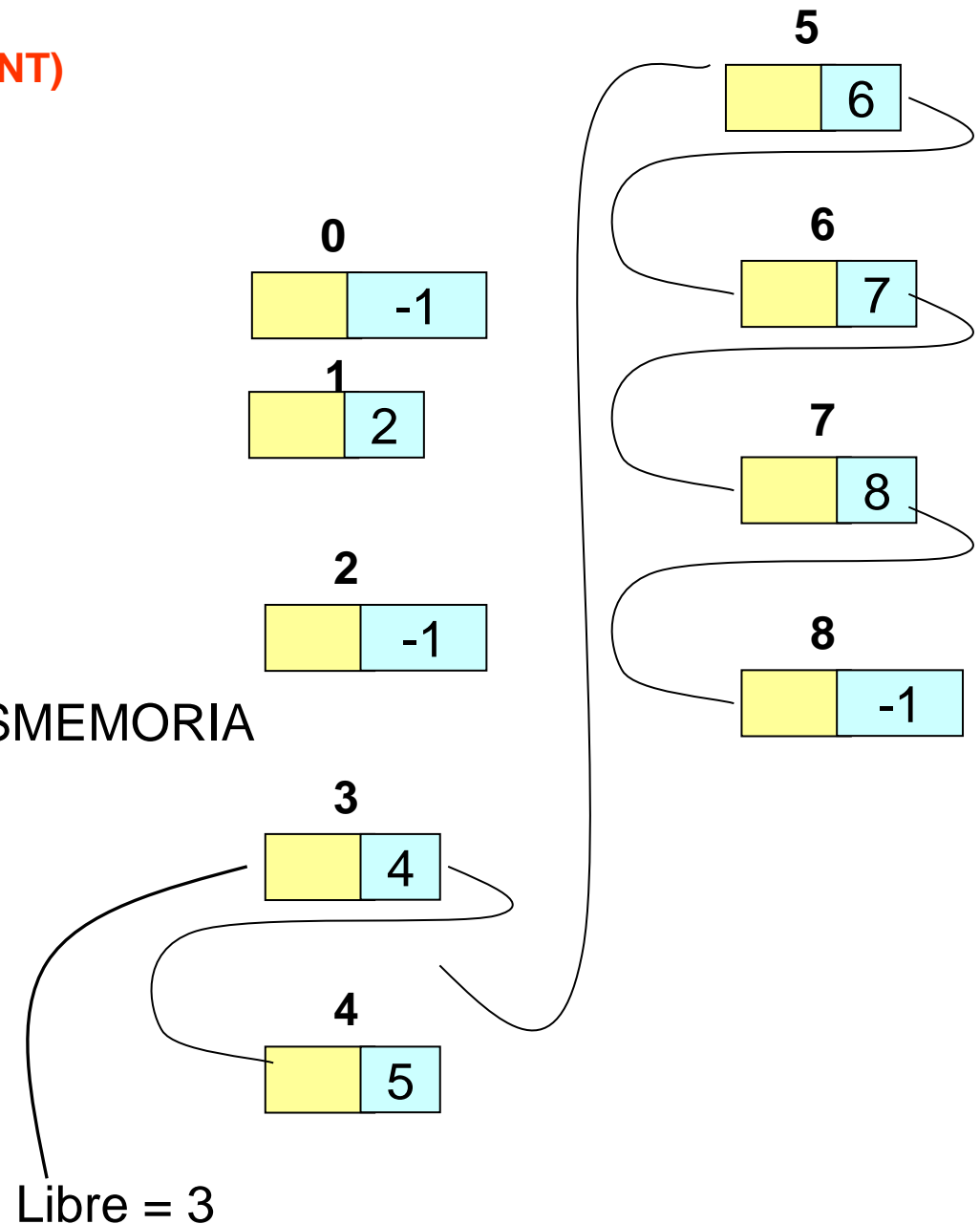
SUPOGASE EL OBJETO A DE TIPO CSMEMORIA

X=A.PEDIR_ESPACIO(1)

Y=A.PEDIR_ESPACIO(2)

X = 0

Y = 1



IMPLEMENTANDO METODOS DE LA CLASE MEMORIA

PROCEDIMIENTO CSMEMORIA::DELETE_ESPACIO(

DIR) INICIO

X = DIR

MIENTRAS MEM[X].LINK<>-1

X= MEM [X].LINK

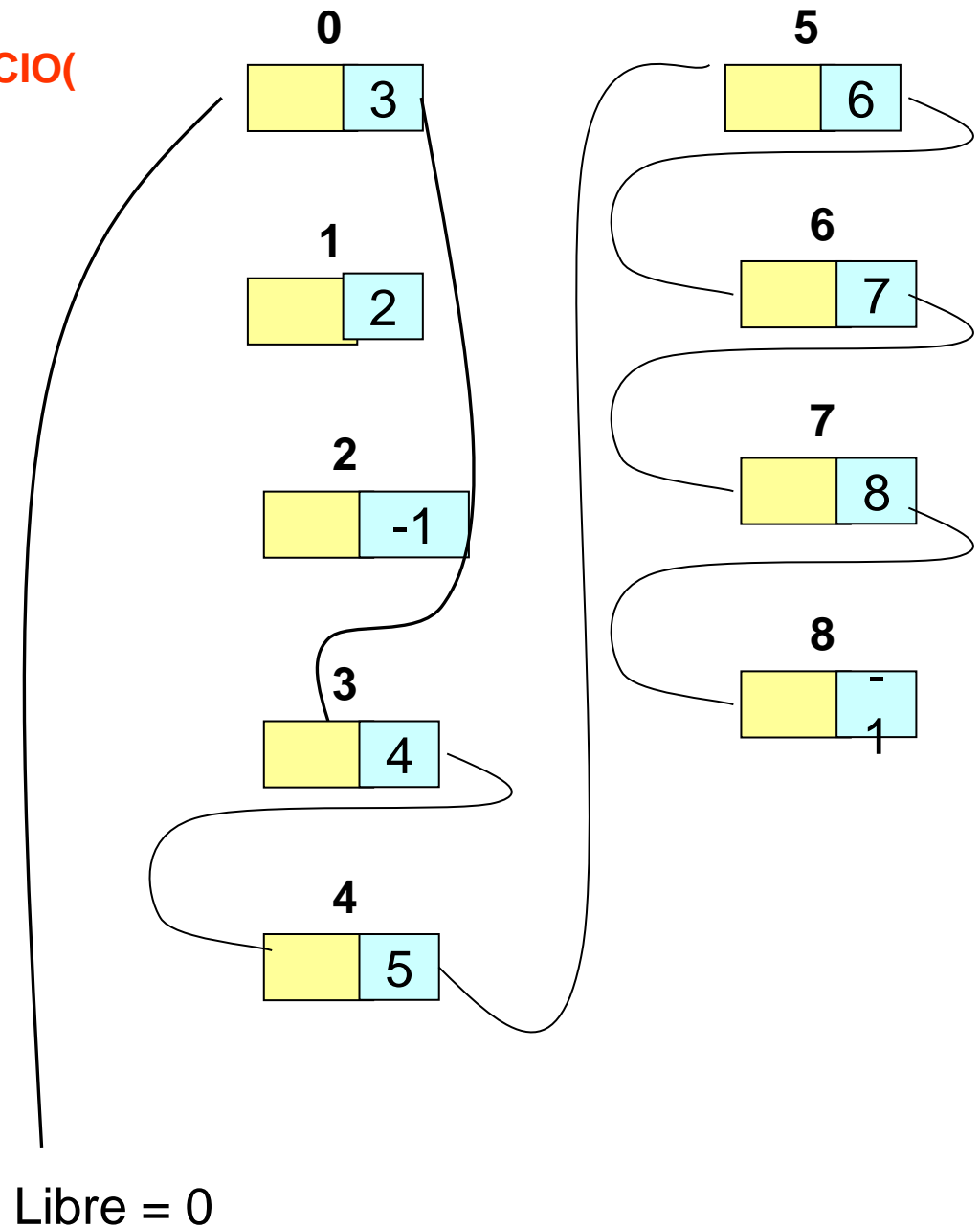
MEM [X].LINK=LIBRE

LIBRE=DIR

FIN

SUPOGASE EL OBJETO A DE TIPO
CSMEMORIA

A.LIBERAR_ESPACIO(X)



IMPLEMENTANDO METODOS DE LA CLASE MEMORIA

FUNCION CSMEMORIA::ESPACIO_DISPONIBLE()

// Cantidad de memoria disponible

INICIO

X = LIBRE

C = 0 // CONTADOR

MIENTRAS X <> -1

C=C+1

X=MEM[X].LINK

FIN

// RETORNAR C

FIN

FUNCTION CSMEMORIA::ESPACIO_OCUPADO()

// Cantidad de Memoria Ocupada

INICIO

C= (max +1) - DISPONIBLE

// RETORNAR C

FIN

FUNCION CSMEMORIA::DIRECCIONLIBRE(DIR)

// Verifica el estado de una dirección de memoria

INICIO

X = LIBRE C = FALSO // BANDERA

MIENTRAS (X <> -1) Y (C = FALSO)

SI X = DIR ENTONCES C = VERDADERO

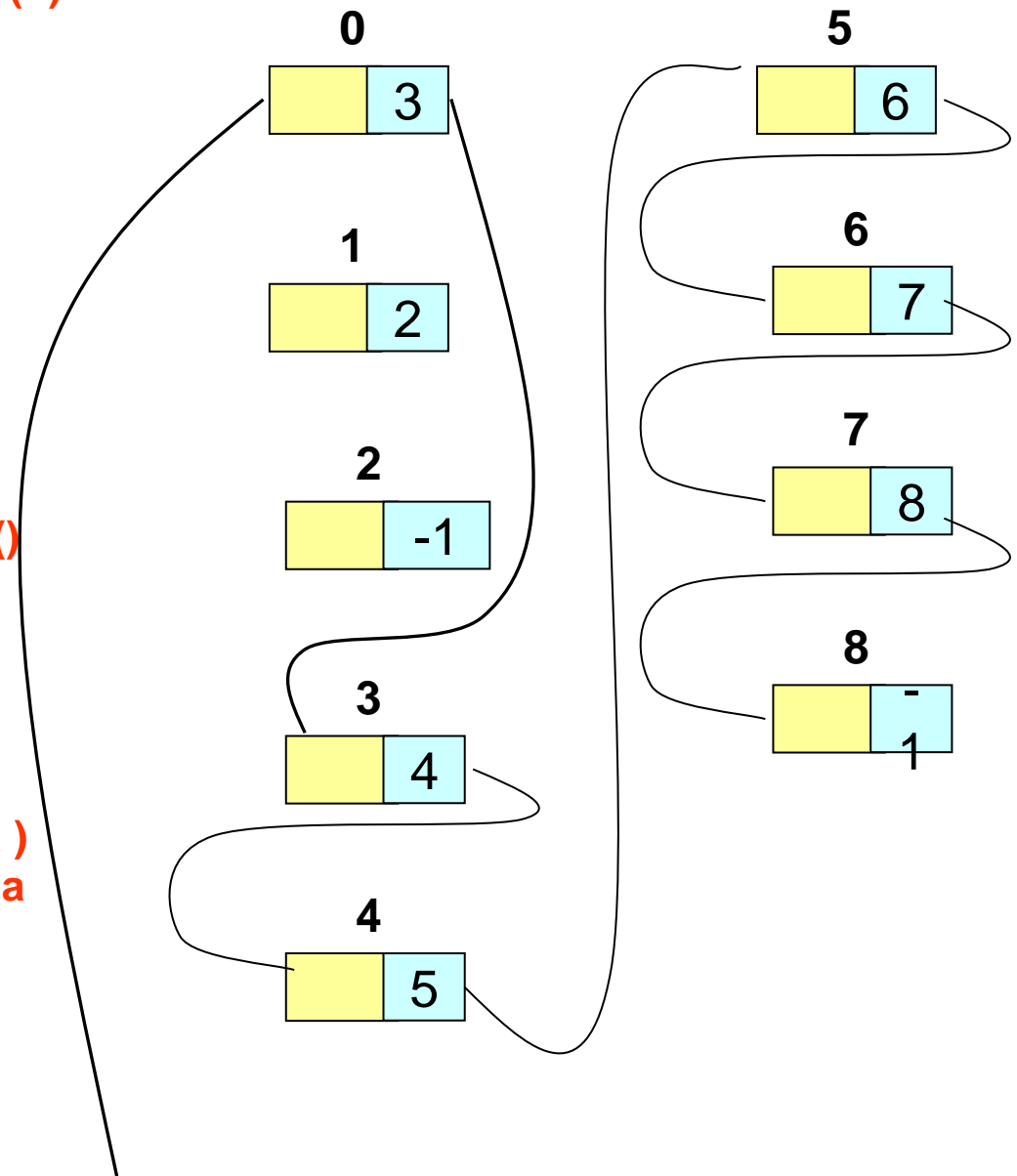
FIN SI

X=MEM[X].LINK

FIN

// RETORNAR C // VERDADERO SI DIR ESTA LIBRE Libre = 0

FIN // FALSO SI DIR ESTA OCUPADA



IMPLEMENTANDO METODOS DE LA CLASE MEMORIA

PROCEDIMIENTO CSMEMORIA::PONERDATO

(DIR , LUGAR, VALOR)

// Coloca valor en lugar a partir de la direccion dir

INICIO

Z = DIR

PARA CADA I = I HASTA LUGAR -1

Z= MEM[Z].LINK

MEM[Z].DATO = valor

FIN

// SI CONSIDERAMOS QUE A ES UN OBJETO

A.PONERDATO(Y,2,25)

FUNCION SMEMORIA::OBTENER_DATO(DIR , LUGAR)

// Cantidad de memoria disponible

INICIO

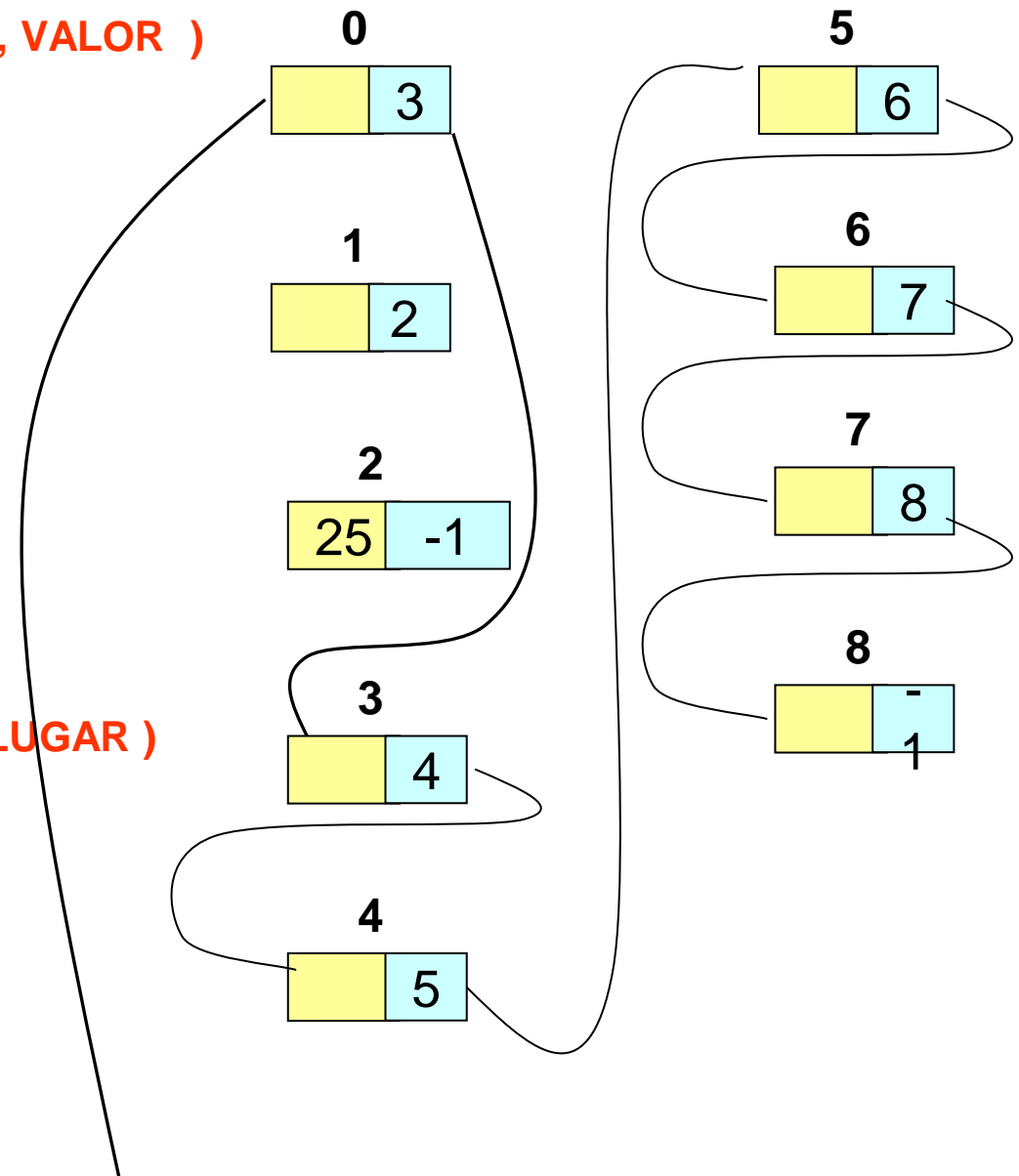
Z = DIR

PARA CADA I = I HASTA LUGAR -1

Z= MEM[Z].LINK

RETORNAR MEM[Z].DATO

FIN



Practica de Laboratorio

- 1) Crear la Clase CSMemoria
- 2) Aumentar el metodo mostrar a la clase CSmemoria
- 3) Crear proyecto en modo consola que tenga menu con 6 opciones
 - i) Crear
 - ii) Pedir Espacio
 - iii) Liberar Espacio
 - iv) Poner Dato
 - v) Mostrar memoria
 - vi) Salir



| | DIR | DATO | LINK |
|---|-----|----------|------|
| M | | + | + |
| E | | 0 | 1 |
| M | | 1 | 2 |
| O | | 2 | 3 |
| R | | 3 | 4 |
| I | | 4 | 5 |
| A | | 5 | 6 |
| | | 6 | 7 |
| | | 7 | 8 |
| | | 8 | 9 |
| | | 9 | 10 |
| | | 10 | -1 |
| | | + | + |
| | | LIBRE =0 | |