

Luisino Penkera Choque **UNIDAD 3. MODELO DE REPRESENTACIÓN DE ESTRUCTURA DE DATOS****1. LA ABSTRACCIÓN**

Es una de las herramientas que nos ayuda a solucionar un problema, es un mecanismo fundamental para la comprensión de problemas y fenómenos que poseen una gran cantidad de detalles, su idea principal consiste en manejar un problema, fenómeno, objetivo, tema o idea como un concepto general sin considerar la gran cantidad de detalles que entre puedan tener. El proceso de abstracción presenta dos aspectos complementarios:

1. Destacar los aspectos (vereda) relevantes de objetivo.
2. Ignorar los aspectos irrelevantes del mismo (la irrelevancia depende del nivel de abstracción, ya que si se pasa a niveles más concretos, es posible que ciertos aspectos pasen a ser relevantes).

Veamos los diferentes tipos de abstracción que podemos encontrar en un programa:

1. **Abstracción Funcional:** (Proporcionado por el lenguaje de alto nivel) Son unidades, librerías, paquetes que determinan el comportamiento y uso de determinado módulo.

**2. Abstracción de Datos:**

- **Tipos de datos:** Proporcionado por el lenguaje de alto nivel. La representación usada es invisible al programador, al cual solo se le permite ver operaciones predefinidas para cada tipo.
- **Tipos de valores:** Del programador que facilita la definición de valores de datos más cercanos al problema que se pretende resolver.
- **TDA:** Para la definición y representación de tipos de datos "valores más operaciones" junto con sus propiedades.
- **Objetivo:** Son TDA a los que se añade propiedades de reutilización y de compartición de código.

3. **Abstracción de Control:** Son los procedimientos que al invocarlos mediante un nombre desde se destaca qué hace la función y se ignora cómo lo hace. El usuario solo necesita conocer la especificación de la abstracción (el qué) puede ignorar el resto de los detalles (el cómo).



## Luisiño Pincene Chique

Si profundizamos más el mundo de la programación y sus conceptos, existe uno de estos conceptos que no se debe confundir, ellos son:

Tipos de datos y estructura de datos

Un tipo de dato, en un lenguaje de programación, define un conjunto de valores que una determinada variable puede tomar, así como las operaciones básicas sobre dicho conjunto.

Ahora veamos cómo se van relacionando estos conceptos, los tipos de datos constituyen un primer nivel de abstracción ya que se tienen en cuenta cómo se implementan o se representa realmente la información sobre la memoria de la máquina. Para el usuario, el proceso de implementación o representación es invisible.

### 2. Estructura de datos

Comenzamos por establecer qué estructura de datos es la forma de organizar un conjunto de datos con el objetivo de facilitar su manipulación. Un dato es la mínima información que se tienen en un sistema.

Por lo que cuando se define una estructura de datos estamos definiendo la organización e interrelación de estos y un conjunto de operaciones mínimas que se pueden realizar sobre ellas como ser:

- Alta, adicionar un nuevo valor a la estructura.
- Baja, borrar, un valor de la estructura.
- Búsqueda, encontrar un determinado valor en la estructura para realizar una operación con este valor.

Otras operaciones que se pueden realizar son:

- Ordenamiento de los elementos perteneciente a la estructura.
- Aporeo, dada dos estructuras originar una nueva ordenada y que contenga a las agregadas.
- Otras, definidas de acuerdo a las características de la estructura.

De tal forma que para resolver un problema intervienen los siguientes elementos: programa, Algoritmo y Estructura de datos.

PROBLEMA


ALGORITMOS

+

ESTRUCTURA  
DE  
DATOS

PROGRAMA



Luisiña Poncero Choque 

**Problema:** Conjunto de hechos / circunstancias que dificultan la consecución de algún fin

**Algoritmo:** Conjunto de reglas finito

**Estructura de datos:** Disposición en memoria de la información

**Programa:** Algoritmo + Estructura de datos

Como ya es conocido por ustedes por lo general tenemos que tratar con los datos simples (enteros, reales, booleanos etc) que por si solo no nos dicen nada, ni nos sirven de mucho, es necesario tratar con estructuras de datos adecuadas a cada necesidad.

La estructura de datos es una colección de datos cuya organización se caracteriza por las funciones de acceso que se usan para almacenar y acceder a los elementos individuales de datos.

Cada estructura ofrece ventajas y desventajas en relación a la simplicidad y eficiencia para la realización de cada operación de esta forma la elección de la estructura de datos apropiada.

Para cada problema depende de los factores como la frecuencia y orden en que se realiza cada operación sobre los datos.

**Ejemplo:** Partiendo de la lógica de que cuando aprendemos a programar nos pide guardar el nombre, sueldo y carnet de una persona normalmente en el código declaráramos las variables nom de tipo cadena, sueldo de tipo real y carnet de tipo cadena, sin embargo el problema se pone complejo si queremos guardar la información de 150 personas lo cual nos lleva rápidamente a la imposibilidad de declarar 450 variables en nuestro código por lo que tenemos que recurrir al uso de estructuras que permitan manejar esa cantidad de información adecuadamente.

## CLASIFICACIÓN DE LAS ESTRUCTURAS DE DATOS

Las estructuras de datos se clasifican en dos consideraciones la existencia de tipos de datos básicos ya establecidos y los nuevos por crear.

- **La estructura de datos primitivas** son los tipos de datos fundamentales que son compatible con una programación. Algunos tipos de datos básicos son enteros, reales, caracteres y booleanos etc.
- **La estructura de datos no primitivas** son aquellas estructuras que se crean utilizando datos primitivos. Los ejemplos de tales estructuras de datos incluye listas, pilas, conjuntos etc. se puede clasificar en dos categorías: lineal y no lineal.

Luisiño Pericena Choque

- **Lineal:** Si los elementos de una estructura de datos se almacenan en 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.

### 3. TIPO DE DATOS ABSTRACTO

El concepto de tipo de datos abstracto (TDA, Abstract Data Types), fue propuesto por primera vez hacia 1974 por John Guttag y otros, pero no fue hasta 1975 que por primera vez Barbara Liskov lo propuso para el lenguaje CLU.

Primera mujer en obtener un doctorado en ciencias de computación de estudios unidos (Stanford 1968).

#### DEFINICIÓN

Con mucha frecuencia se utilizan los términos TDA y abstracción de datos de manera equivalente y esto debido a la similitud e independencia de ambos. Sin embargo es importante definir por separado los dos conceptos.

Como ya se mencionó, los lenguajes de programación Orientados a Objetos son lenguajes formados por diferentes métodos y funciones que son llamados como orden en el que el programa los requiere, o el usuario lo desea. La abstracción de datos consiste en ocultar las características de un objeto y abreviarlas, de manera que solamente usamos el nombre del programa objeto en nuestro programa. Esto es similar a la situación de la vida cotidiana. Cuando yo digo la palabra "perro" no necesito que me diga que hace el perro. Usted ya sabe la forma que tiene un perro y también sabe que los perros ladran.

De manera que abstraeremos todas las características de todos los perros y también sabe que los perros en un (cuervo) solo termino, al cual llamo "perro".

A esto se lo llama abstracción es un concepto útil en la programación ya que un usuario no necesita mencionar todas las características y funciones de un objeto cada vez que este se utiliza o no quieren declarados por separado en el programa y simplemente se utiliza el término abstracto ("perro") para mencionarlo.

En el ejemplo anterior, "perro" es un tipo de datos Abstracto y todo el proceso definirlo, implementarlo, eliminarlo es lo que se llama abstracción de datos.



Luisiño Pincera Choque

Vamos a pensar un ejemplo real de la programación. Supongamos que en algún lenguaje de programación Orientado a Objeto un pequeño programa saca el área de un rectángulo de las dimensiones que un usuario decida. Pensemos también que el usuario probablemente quiera saber el área de varios rectángulos. Sería muy tedioso para el programador definir la multiplicación de base por altura varias veces, el programa además de limitar al usuario a sacar un número determinado de áreas. Por ello el programador puede crear una función denominada área la cual va ser llamada el número de veces que sea necesario por el usuario y así el programador se evita problemas y el programa resulta más rápido más eficiente y de menor longitud.

Para lograr esto se crea método de longitud. Para lograr esto se crea método de área de manera separada la interfaz gráfica presenta al usuario y se estipula ahí las operaciones a realizar resolviendo el valor de la multiplicación en el método principal solamente se llama a la función área y el programa hace el resto.

El hecho de guardar todas las características y habilidades de un objeto por separado se le llama encapsulamiento y esto también un concepto importante para entender la estructura de datos. Es frecuente en el encapsulamiento y es también un concepto importante para entender la estructura de datos.

Es frecuente que el encapsulamiento sea usado como sinónimo de ocultación de información, aunque algunos creen que no es así.

### SEPARACIÓN DE LA INTERFAZ E IMPLEMENTACIÓN

Cuando se usa un programa de computación, un TDA es representado por la interfaz la cual surge como cubierta a la correspondiente implementación. Las versiones de TDA deben preocuparse por la interfaz pero no por la implementación, ya que esto puede cambiar en el tiempo y afectar a los programas que usan TDA. Esto se basa en el concepto de ocultación de información, una protección para el programa de decisiones de diseño que son objeto de cambio.

La solidez de un TDA reside en la idea de que la implementación está escondida al usuario y la interfaz es pública. Esto significa que el TDA puede ser implementado de diferentes formas pero mientras se mantenga consistente con la interfaz, los programas que lo usan no se ven afectados.

En la terminología de lenguaje Orientado a Objeto un TDA es un enlace una instancia de un TDA o clase es un objeto.

## Luisiño Pencerá Cheque

Representación interna de los tipos de datos, que además suele ser una implementación bastante alejándose de lo que se sabe que trabaja el compilador. Lo interesante es que los lenguajes actuales no van a permitir ampliar los TDA predefinidos con otros que serán definidos por el propio programador para adecuar así los tipos de datos a la necesidad de los programas.

Los TDA que nos van a interesar son aquellos que reflejan cierto comportamiento. Es decir, cierta variedad de datos estructurados de esta forma de estructurar almacenar los datos serán los que nos referimos para caracterizar cada TDA.

Notese cuando hablamos de TDA no vemos ninguna solución al tipo de los elementos sino a la forma que están dispuestos estos elementos. Solo nos interesa la estructura que aporta la información y su organización. Para determinar el comportamiento estructural basta con observar la conducta que siguen los datos.

Resumiendo podemos decir, que **falta la implementación de los operadores** como los elementos internos del TDA serán privados (acceso externo y oculto, o cualquier otro nivel).

Un TDA representa una abstracción:

- Se destacan los detalles (normalmente pocos) de la especificación (el qué).
- Se ocultan los detalles (asi siempre numerosos) de la implementación (el cómo).

### ESPECIFICACIÓN DE UN TDA

La especificación de un TDA se puede representar de dos formas las cuales son:

- Especificación no formal/informal
- Especificación formal

**Especificación formal** (usando lenguaje natural) para la especificación de un tipo de datos abstracto informal se debe seguir esquema:

1. Elementos que conforman la estructura de datos
  - ▶ Tipos de datos: Nombre del tipo
  - ▶ Valores: Descripción de los posibles valores
  - ▶ Operación: Cita de cada operación



Luisito Pericera Choque

2. Descripción de la operación de la estructura: se describe cada uno de las operaciones sobre el TDA

Nombre	de la operación
Descripción	Breve de su utilidad
Datos de entrada	a la operación
Datos	que genera como salida la operación
Precondiciones	condiciones que deben cumplirse antes de utilizar
Poscondición	condiciones en el que TDA después

Especificación formal: (usando pseudocódigo o un lenguaje de programación) y está contruida por los siguientes puntos

1. Tipo: Nombre del TDA
2. Sintaxis: formas de las operaciones  
Nombre de la operación (argumentos)  $\rightarrow$  resultado
3. Semántica: Significado de las operaciones  
Nombre de la operación (valores particulares)  $\rightarrow$  Expresión resultante

Para mayor comprensión se presenta parte del TDA cadena

### NO FORMAL

1. elemento que conformaría la estructura de datos

TDA Cadena (Valores): todos los caracteres alfabéticos, números, operaciones, crear, eliminar caracter, insertar caracter, eliminar caracteres, número caracteres

2. Descripciones de la operaciones de la estructura

Crear

- Utilidad: Sirve para inicializar la cadena en vacío
- Entrada: Cadena S que será iniciada
- Salida: Cadena S iniciada
- Precondición: Ninguna
- Poscondición: la cadena S tiene 0 caracteres

### FORMAL/SEUDOCODIGO O LENGUE

opción 1

Tipo: Cadena

Sintaxis:

Crear (S)

Añadir Caracter (S, L)

Eliminar (S)

Insertar Caracter (S, P, L)

Eliminar Caracter (S, P)

Número Caracteres (S)

Semántica

Para todo S pertenece a

cada S

Crear (crear S)  $\rightarrow$  Paso

Número caracteres (crear S)  $\rightarrow$  0

## Luisiño Poncea Choque

2. Descripción de la operación: de la estructura: describe cada uno de las operaciones sobre el TDA

Nombre de la operación	Breve descripción
Descripción	Breve descripción
Datos de entrada a la operación	
Datos	que genera como salida la operación
Precondiciones	condiciones que deben cumplirse antes de utilizar
Poscondición	condiciones a las que TDA después

Especificación formal: usando pseudocódigo o un lenguaje de programación y está centrado por los siguientes puntos:

1. Tipo: Nombre del TDA
2. Sintaxis: formas de las operaciones  
Nombre de la operación (argumentos) → resultado
3. Semántica: Significado de las operaciones  
Nombre de la operación (valores particulares) → Expresión resultante

Para mayor comprensión se presenta parte del TDA cadena

### NO FORMAL

1. elemento que conformará la estructura de datos

TDA Cadena (valores): todos los caracteres alfabéticos, números, operaciones, crear, eliminar caracter, / insertar caracter  
Eliminar caracteres, número caracteres

2. Descripciones de la operaciones de la estructura

- Crear
- Utilidad: Sirve para inicializar la cadena en vacío
- Entrada: Cadena S que será iniciada
- Salida: Cadena S iniciada
- Precondición: Ninguna
- Poscondición: la cadena S tiene 0 caracteres

### FORMAL / SEUDOCÓDIGO O LENGUE

OPCIÓN 1

Tipo: Cadena

Sintaxis:

crear (S)

Añadir Caracter (S, L)

Eliminar (S)

Insertar Caracter (S, P, L)

Eliminar Caracter (S, P)

Número Caracteres (S)

Semántica

Para todo S pertenece a cad S

Usar (crear (S)) → falso

Número caracteres (crear (S)) → 0



Aushio Ponce Chopo

**Uleno**

Utilidad: Sirve para verificar si una cadena este llena o no  
 Entrada: cadena S que viene  
 Salida: sale la verdad S == 80 Verdadero  
 S no fulgo  
 Precondición: Ninguna  
 Poscondición: Ninguna (Pues la cadena S no se modifica)

**Uleno 2**

Clase cadena {

private:

// Datos miembros de la clase char SETA;

public:

// Funciones miembros

void AñadirCaracter (Char c);

void EliminarCaracter (int i);

void ImprimirCaracter (int i, char c);

bool llena();

**4. FORMAS DE IMPLEMENTACIÓN DE LAS TDA**

Como se puede apreciar en la lectura del presente documento se ha visto que la implementación de las TDA es independiente de la Especificación de las mismas la cual ratifica

lo indicado anteriormente "Una TDA puede ser implementado de varias formas sin que esto afecte a la aplicaciones que usen el mismo" es así la cuatro formas de implementación que se detallaron

**4.1 Modelo Estático**

Esta forma de implementación se refiere al uso de recursos memoria en la implementación de una TDA que simplemente define antes de su ejecución (al menos necesariamente no lleva a usar tiempo)

**4.2 Modelo Dinámico**

En esta forma de implementación anteriormente citada cabe aclarar que la planteada en el punto es única y exclusivamente a cadenas para tener una mayor comprensión sobre los punteros

**4.3 Modelo Simulado**

A diferencia de las 3 formas de implementación anteriormente citadas cabe aclarar que el planteado es únicamente sobre los punteros por lo que se propone crear una TDA denominada GMemoria cuyo objetivo será simular el comportamiento de la memoria con el uso de vectores

**4.4 Modelo Persistente**

El modelo persistente es el tiempo en la forma mas usual que se tiene para hacer que la implementación de una TDA sea persistente con el tiempo

Luisito Pericena Cheque

Esto significa que el comportamiento podría estar extenso del uso exclusivo de la memoria trasladando las acciones de los metadatos a guardar la información en archivo.

<https://youtu.be/CNvHX50BPvQ>

Considerando que existe biografía sobre como utilizar los modelos de implementación (estático y dinámico, Dinámico) o simulación de simulaciones el modelo simulado.

### Concepto Básico

Como funciona la memoria

Todas las variables de un programa tiene asociado un lugar en la memoria del computador.

La memoria puede ser vista como un gran arreglo de bits, un bit es la unidad básica de la información que se puede representar en un computador y puede tener valores 0 o 1.

...	00010011	11011000	00101011	01111100	01010011
-----	----------	----------	----------	----------	----------

### Tipos de datos

El tamaño de un valor de tipo char es de 1 byte y el significado de los bits o bits está determinado cuando la codificación ASCII es así que definamos la variable y tipo char y luego en la memoria también es binario el carácter según el siguiente gráfico.

...	01	11011000	00101011	01111100	01010011
-----	----	----------	----------	----------	----------

Los enteros son almacenados en su representación binaria la memoria más común de representar los enteros negativos es el complemento a dos, siendo así y considerando que definamos un espacio de memoria llamado b de tipo entero y decimos b = 9 tendríamos la siguiente consideración que los valores de 'b' y 9 están en binario.

	A	B			
...	0	9	00101011	01111100	01010011



## Direcciones de memoria

Todos los bytes en la memoria tienen una dirección, que no es más que un índice consecutivo por conveniencia, las direcciones de memoria suelen escribirse en notación hexadecimal, pero no hay que preocuparse, se trata de un simple número entero.

Dirección	Valor
...	...
0xF1E568	0 0 0 1 0 0 1 1
0xF1E569	1 1 0 1 1 0 0 0
0xF1E56A	0 0 1 0 1 0 1 1
0xF1E56B	0 1 1 1 1 1 0 0
0xF1E56C	0 1 0 1 0 0 1 1
...	...

Si embargo tenemos que guardar los datos de la variable  $Z$  que tiene dos campos  $A$  de tipo carácter y  $B$  de tipo numérico, entonces nuestra gráfica debería guardar la siguiente manera si decimos  $Z.A = 'e'$  y  $Z.B = 9$  solo para efectos de implementar el TDA memoria.

Dirección	Valor	id	link
...	...	...	...
0xF1E568	'e'	a	0xF1E56B
0xF1E569	1 1 0 1 1 0 0 0		null
0xF1E56A	0 0 1 0 1 0 1 1		null
0xF1E56B	9	b	null
0xF1E56C	0 1 0 1 0 0 1 1		null

Es así que para la implementación de nuestro TDA Simemoria utilizaremos un vector donde en cada celda (dirección) guardaremos Datos, id, link para los fines siguientes:

Datos: Será el dato guardado en la memoria.

Id: Será el nombre del espacio de memoria.

Link: Será la dirección de otro espacio.

(Se establece que el simulador planteado solo manipula datos enteros en valor por que no se incrementa un campo para determinar el tipo de dato guardado.)

Tema:

Luisiño Pericena Choque

Fecha: / /

## DESCRIPCIÓN NO FORMAL DE TDA S-MEMORIA

### 1. Elemento que conformará estructura

TDA S-Memoria (Valores: todos los números enteros, Operaciones: crear, New-espacio, Delete-Espacio, Poner-dato, Obtener-dato, de-libre, Espacio-ocupado, Espacio-Disponible)

### 2. Descripción de las operaciones de la estructura

crear

Utilidad: Sirve para crear los espacios de memoria

Entrada: Espacio de memoria sin relación

Salida: Espacio de memoria relacionado entre sí

Precondición: Ninguna

Poscondición: Memoria lista para usarse

New-Espacio: Solicitar N espacios de memoria con su debida identificación

Utilidad: Gestionar con los identificadores requeridos

Entrada:

Salida: Dirección de memoria a partir de donde se pueda guardar información para los identificadores requeridos

Precondición: Memoria inicializada

Poscondición: Memoria disminuida en su capacidad de dirección libre para trabajar

Delete-Espacio

Utilidad: Liberar un espacio de memoria ocupado a fin de que retorne a forma parte de la memoria y puede ser usada en otra actividad

Entrada: Dirección inicial del espacio de memoria al ser liberado

Salida: Ninguna

Precondición: La dirección de memoria a liberar debe existir y pertenecer a las direcciones en uso

Poscondición: Memoria incrementada en su capacidad de dirección libre para trabajar

Poner-Dato

Utilidad: Asignar un valor a un espacio de memoria considerando su identificador y partir una dirección de memoria

Entrada: Dirección inicial de memoria, Identificador de memoria

Salida: Valor que se encuentra en el lugar especificado según identificador

Precondición: Dirección inicial de memoria e indicador válido

Poscondición: Ninguna



## Luisiño Pensez Chaque

### Espace - Disponible

Utilidad: Determina cuanto espacio de memoria estan libres para poder trabajar

Entrada: Ninguna

Salida: Numero de espacio de memoria libre

Precondición: Ninguna

Poscondición: Ninguna

### Espace - Ocupado

Utilidad: Determinar Cuanto espacio de memoria estan ocupados

Entrada: Ninguna

Salida: Numero de espacio de memoria ocupada

Precondición: Ninguna

Poscondición: Ninguna

### dir - libre

Utilidad: Verificación si una dirección de memoria esta libre

Entrada: Dirección de memoria

Salida: Valor booleano / verdadero o falso

Precondición: La dirección debe ser valida

Poscondición: Ninguna

## IMPLEMENTACIÓN DEL TDA MEMORIA

Constantes MAX = 20

NULO = 1

Definiendo Tipo de Datos

Nodo M

Data Tipo Data (Entero)

id Cadena (12)

Link Dirección de Memoria (Entero)

Fin Tipo

Clase CSMemoria

Atributos

MEM arreglo (MAX) de Tipo Nodo M

libre Dirección de Memoria (Entero)

Métodos

Constructor // crear

direccion nuevo espacio (cadena)

Deletar espacio (dir)

Permitir dato (dir, cadena, id, valor)

Tipo Data obtener Data (dir, lugar)

Entero Espacio Disponible ()

Entero Espacio ocupado ()

booleano dir - libre (dir)

fin def Clase

## Luzhina Patricia Choque

Construir (S.Memoria::Crear())

Inicio

Para cada  $i$  desde 0 hasta  $max$ Men  $[i]$ .Link =  $i + 1$ 

Fin Para

Men  $[max]$ .Link = -1

Libre = 0

Fin

S.Memoria::Delete Exceso (Dir)

Inicio

 $X = dir$ Mientras Men  $[X]$ .Link  $< 7 - 1$  $X = Men [X]$ .Link

Fin mientras

Men  $[X]$ .Link = Libre

Libre = Dir

Fin

Direccion S.Memoria::New Espaciado

// cadena 'a,b,c'

Inicio

cant = Numero de (cadena)

dir = libre

D = libre

Para cada  $i = 1$  hasta cant - 1Men  $[D]$ .ID = obtener\_ID(cadena, $i$ )D = Men  $[D]$ .Link

Fin Para

Libre = Men  $[D]$ .LinkMen  $[D]$ .Link = -1Men  $[D]$ .ID = obtener\_id (cadena, cant)

// retorna Dir

Fin

Numero S.Memoria::Espacio Disponible()

// cantidad de memoria disponible

Inicio

 $X = libre$  $C = 0$  // contadorMientras  $X < 7 - 1$  $C = C + 1$  $X = Men [X]$ .Link

Fin Mientras

// retorna C

Fin

Borrar S.Memoria::Dir Libre (Dir)

// Si Dir Libre Verdadero si no falso

Inicio

 $X = Libre$   $C = FALSO$  // BANDERAMIENTRAS ( $X < 7 - 1$ )  $\vee$  ( $C = FALSO$ )Si  $X = DIR$  ENTONCES  $C = VERDADERA$ 

Fin Si

 $X = Men [X]$ .Link

Fin MIENTRAS

RETORNAR C

Fin

S.Memoria::Insertar Dato

(Dir, cadena, id, VALOR)

// Formato de cadena id = 1000

Inicio

 $Z = DIR$ 

Eliminar Pecho (cadena, id)

MIENTRAS  $Z < 7 - 1$ Si MEM  $[Z]$ .ID = cadena, id

Entonces

Mem  $[Z]$ .DATO = VALOR

Fin Si

 $Z = MEM [Z]$ .Link

Fin MIENTRAS

Fin



Tipo Datos y Memoria: OBTener  
- dato, cadena, id)

Numero (memoria), Espacio-  
ocupado ( )  
// cantidad de memoria ocupada

Inicio

Z = DIR EX = FALSO

Eliminar Flecha (cadena, id)

MIENTRAS (Z < Z NULL)

SI MEME[Z].ID = CADENA.ID

ENTONCES

// RETORNAR MEME[Z].DATO

FIN SI

Z = MEME[Z].LIN

FIN MIENTRAS

FIN

Inicio

C = (Lmax + 1) = Espacio Disponible

RETORNAR C

FIN

Using namespace std;

const int MAX = 20

const int NULO = -1

```
struct NodeM {
    int dato;
    string id;
    int link;
};
```

Class Memoria {

private:

NodeM\* mem;

int libre

int numero\_id (string id);

String & Vector\_id (string cad);

Public:

Memoria();

int new\_espacio (string cadena);

void delete\_espacio (int dir);

void poner\_dato (int dir, string cadena, cadena id, int valor);

int obtener\_dato (int dir, string cadena id);

int espacio\_disponible ();

int espacio\_ocupado ();

bool dir\_libre (int dir);

void mostrar ();

};

Memoria: Memoria() {

```

num = new NodeM[MAX];
for (int i = 0; i < MAX; i++) {
    mem[i].link = i+1;
}
mem[MAX-1].link = -1;
this -> libre = 0;
}

```

int Memoria: numero-id (string id) {

```

if (id.length() == 0)
    return 0;
int cont = 0;
for (int i = 0; i < id.length(); i++) {
    if (id[i] == ' ' || ',')
        cont++;
}
return cont + 1;
}

```

String \* Memoria: vector-id (string cad) {

```

int n = numero-id (cad);
String * v = new String [n];
int c = 0;
while (cad.length() > 0) {
    int pos = cad.find_first_of(" ,");
    if (pos == -1)
        pos = cad.length();
    string id = cad.substr(0, pos);
    v[c] = id;
    c++;
    cad.erase(0, pos+1);
}
return v;
}

```

int Memoria: new-espacio (string cadena) {

```

int cant = numero-id (cadena);
int d = this -> libre;
int d = this -> libre;
string * id = vector-id (cadena);
int i = 0;
for (i = 0; i < cant-1; i++) {
    mem[id].id = id[i];
    d = min [d].link;
}

```



Luishtta Ponceiro Chapue

```

min [d].link = -1;
return dir;

```

}

```

Void Memoria::Delete_ejecucion (int dir) {

```

```

    int x = dir;
    while (min [x].link != -1)
        x = min [x].link;
    min [x].link = libre;
    libre = dir;
}

```

```

Void Memoria::Poner_Dato (int dir, string cadena, int valor)
{

```

```

    int z = dir;
    cadena_id = cadena_id.substr (z, cadena_id.length () - z);
    while (z != Nulo) {
        if (min [z].id == cadena_id) {
            min [z].dato = valor;
            break;
        }
        z = min [z].link;
    }
}

```

```

int Memoria::Obtener_dato (int dir, string cadena_id) {

```

```

    int z = dir;
    cadena_id = cadena_id.substr (z, cadena_id.length () - z);
    while (z != Nulo) {
        if (min [z].id == cadena_id)
            return min [z].dato;
        z = min [z].link;
    }
}

```

```

Int Memoria::Espacio_disponible () {

```

```

    int x = libre;
    int c = 0;
    while (x != -1) {
        c++;
        x = min [x].link;
    }
    return c;
}

```

Luisián Benítez Choque

```
int Memoria::Espacio-disponible () {
```

```
    int c = Max - Espacio-disponible ();
    return c;
}
```

```
bool Memoria::disponible (int dir) {
```

```
    int x = libre;
```

```
    bool c = false;
```

```
    while (x != -1 & c == false) {
```

```
        if (x == dir)
```

```
            c = true;
```

```
            x = mem[x].link;
```

```
    }
```

```
    return c;
```

```
}
```

```
void Memoria::Mostrar () {
```

```
    cout << "Dir. Data ID LIN" << "\n";
```

```
    cout << " + ..... + \n";
```

```
    for (int i = 0; i < Max; i++) {
```

```
        cout << i << " | " << mem[i].data << " | " << mem[i].
```

```
        id << " | " << mem[i].link << endl;
```

```
    }
```

```
    cout << " + ..... + \n";
```

```
    cout << "libre = " << libre << endl;
```

```
}
```



