ANÁLISIS COMPARATIVO DE LENGUAJES

La educación genera confianza. La confianza genera esperanza. La esperanza genera paz.-Confucio.

M. Cs. Patricia Roggero

Universidad Nacional de San Luis
Año 2021

Tipos de datos Estructurados

Notas de Clase

Capítulos VI Programming Languages – Design and Implementation – Terrence Pratt

Capítulos V Concepts of Programming Languages – Robert Sebesta

Apunte de la cátedra.

Tipos de datos estructurados (TDE's)

Estructura de datos (objeto de datos estructurado): es un OD cuyos elementos (o componentes) son OD elementales o estructurados.

Algunos tipos de datos estructurados importantes y que veremos a continuación, son:

- Arreglos
- Registros
- Conjuntos

Las ligaduras de OD a valores, nombres y ubicaciones suelen ser más complejas que en los elementales.

Especificación de tipos de datos estructurados

La especificación suele incluir los siguientes atributos:

Número de componentes: estructura de tamaño ¿fijo o variable?

Tipo de cada componente: estructura de datos ¿homogénea o heterogénea?

Nombres usados para seleccionar las componentes: ¿subíndices, identificadores definidos por el programador o componentes particulares?

Número máximo de componentes (en estructuras de tamaño variable).

Organización de las componentes: ¿secuencia lineal o formas multidimensionales?

Operaciones sobre estructuras de datos

- La *selección (o acceso)* de componentes: ¿directa o secuencial?.
- Operaciones con *estructuras de datos completas*: algunos lenguajes proveen este tipo de operaciones, pero son limitadas.
- La *inserción/supresión* de componentes: modifican el número de componentes.
- La *creación/destrucción* de estructuras de datos.

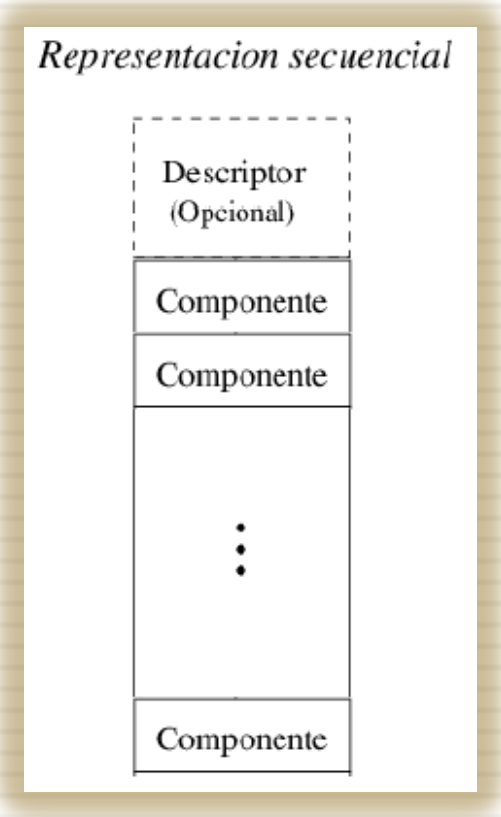
Implementación de tipos de datos estructurados

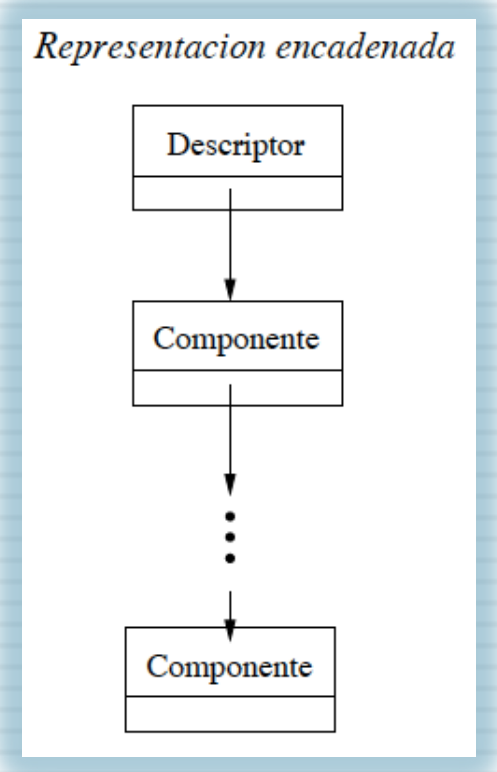
La **representación en memoria** para una estructura de datos incluye:

1. Espacio para almacenar las *componentes* de la estructura.
2. Un *descriptor* (opcional) con (algunos o todos) los atributos de la estructura.

Para la elección del almacenamiento de estas estructuras, se buscará una que permita la *selección* eficiente de las componentes y eficiencia en la *administración del almacenamiento* en la implementación provista por el lenguaje.

Implementación de tipos de datos estructurados





Secuencial → estructuras de tamaño fijo y algunas de tamaño variable con elementos homogéneos.

Encadenada → estructuras de tamaño variable.

Implementación de operaciones en tipos de datos estructurados

En general deben ser simuladas por software. Es fundamental el rol de la selección (acceso) de componentes (directa o secuencial), que está influenciada por el tipo de representación.

Con representación secuencial:

- la *selección directa* involucra un cálculo *dirección-base-más-desplazamiento (db+d)*.
- el *acceso a una secuencia* de componentes involucra:
  1. seleccionar la primera componente de la serie en base al cálculo `db+d`
  2. para avanzar a siguiente componente, sumar tamaño componente actual a su ubicación.

Implementación de tipos de datos estructurados

Con representación encadenada:

- la *selección directa* involucra seguir la cadena hasta la componente deseada.
- la *selección de una secuencia* de componentes involucra:
  1. acceder a la primera componente como antes.
  2. seguir el puntero a la siguiente componente por cada selección subsiguiente.

Administración de memoria y estructuras de datos

Durante el tiempo de vida de un OD, se pueden crear y destruir distintas ligaduras de él al lugar de almacenamiento.

Cuando los tiempos de vida de los OD y de sus ligaduras al almacenamiento no coinciden pueden surgir dos tipos de problemas: **Basura** y **Referencias desactivadas**.

Declaraciones y chequeo de tipo para estructuras de datos

Similar al de los tipos de datos elementales aunque más complejo: hay más atributos para especificar y debido a las operaciones de selección de componentes:

- Problema de la existencia de la componente seleccionada:

```
Var Ar : array [1..20, -2..6] of char
writeln(Ar[i][j])
```

- Problema del tipo de la componente seleccionada (una secuencia de selección puede definir un paso complejo en la estructura de datos hasta llegar a la componente deseada).

Arreglos unidemensionales (o vectores)

Estructuras de datos integradas por un número fijo de componentes del mismo tipo organizados como una serie lineal simple, cuyas componentes pueden ser accedidas mediante un subíndice, un entero (o valor enumerado) que indica su posición en la serie.

Atributos de un vector:

- El número de componentes.
- El tipo de datos de las componentes.
- Valores de subíndice válidos.

Ejemplos arreglos unidemensionales

En el lenguaje Pascal:

```
var B : array [1..8] of boolean
type colores = (rojo,azul,gris,verde)
var arcolores : array[colores] of char
```

En el lenguaje C:

```
float B [10];
char C[]={'#','M','9','+','E','ñ','O','@'};
```

Algunas operaciones con vectores

- Acceso (y modificación) de una componente mediante un subíndice.
- Operaciones aritméticas con vectores completos (pocos lenguajes las proveen).
- Creación y destrucción de vectores.
- No se permite la inserción y supresión de componentes.
- Modificación del contenido de las componentes.

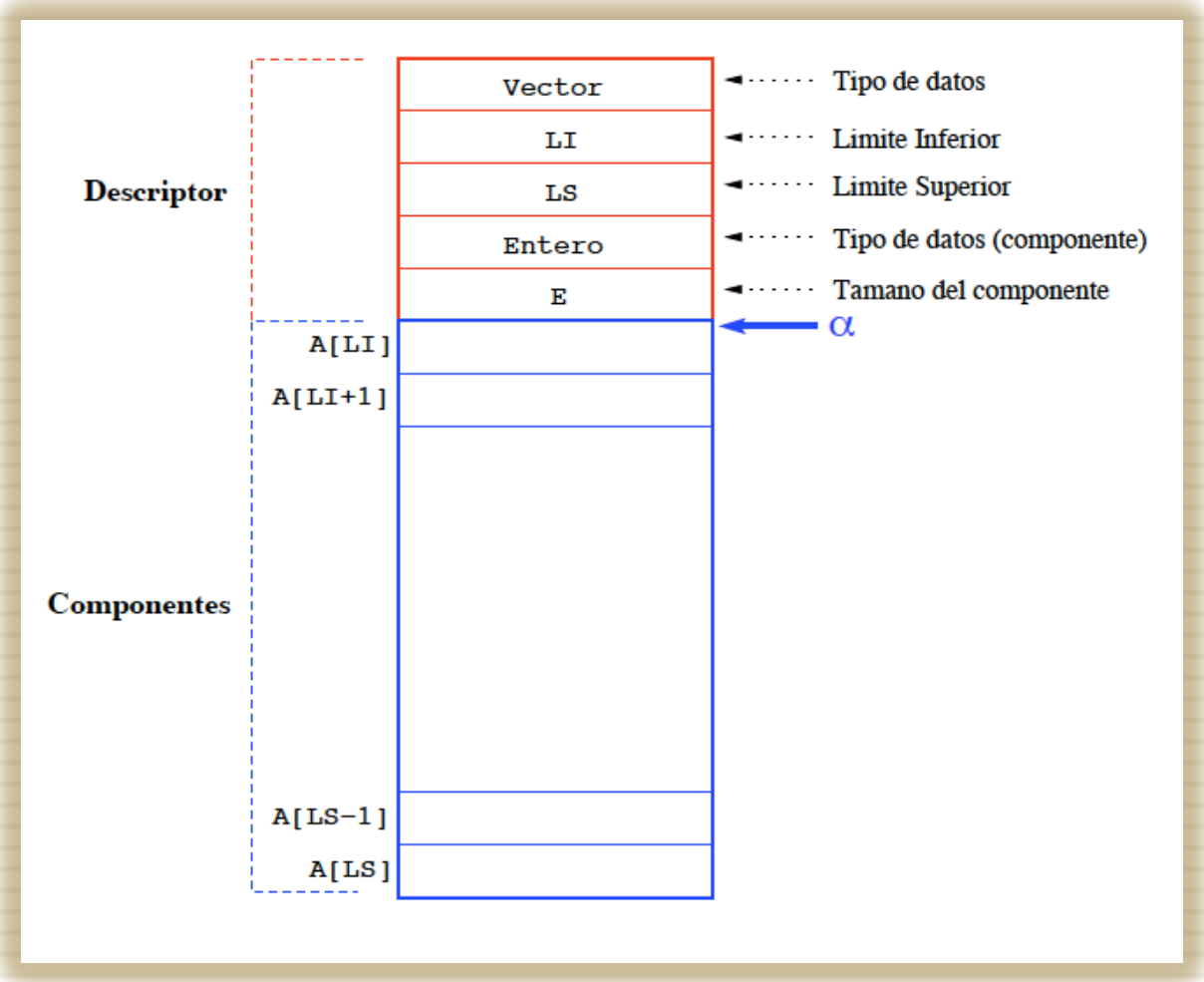
Implementación

Tanto el almacenamiento como el acceso son fáciles de implementar, razones:

- Homogeneidad de las componentes: tamaño y estructura de los elementos no cambiará.
- Tamaño fijo del arreglo: número y posición de las componentes no cambiará.

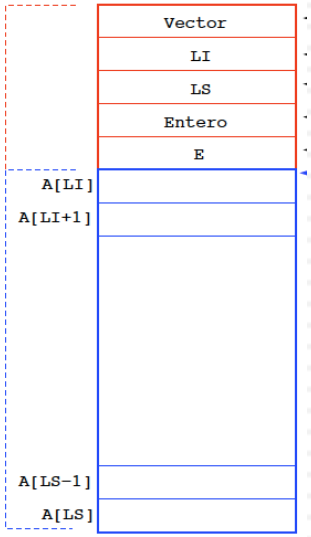
Entonces la representación secuencial es la más adecuada.

Representación en memoria de arreglos unidimensionales



Acceso a la i -ésima componente de un vector

$$dir(A[i]) = \alpha + (i - LI) \times E$$



Pero esta fórmula la podemos escribir como:

$$dir(A[i]) = (\alpha - LI \times E) + i \times E$$

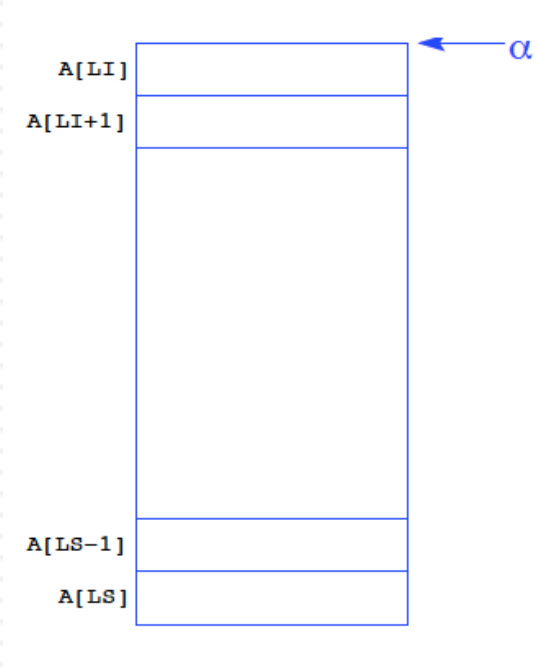
Notar que una vez que se asigna espacio $(\alpha - LI \times E)$

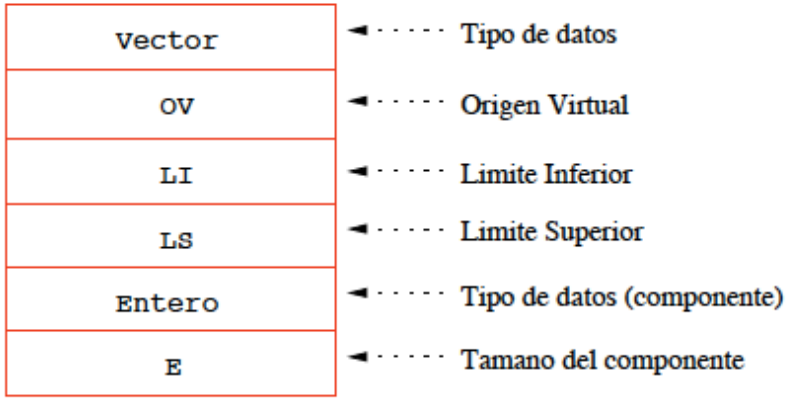
es una constante que se puede llamar K, entonces la fórmula de acceso se reduce a:

$$dir(A[i]) = K + i \times E$$

Almacenamiento separado para descriptor y componentes

También es posible almacenar el descriptor en otra ubicación de memoria, no solamente de manera contigua a las componentes.





Arreglos multidimensionales

Los arreglos de 1 dimensión que hemos visto se denominan vectores.

Los arreglos de 2 dimensiones se suele denominar matrices.
Ejemplo:

1	2	3	4
3	4	5	6
5	6	7	8

Los arreglos n-dimensionales con $n > 2$.

Implementación de arreglos bidimensionales (matrices)

Es directa si consideramos una matriz como un vector de vectores. Un arreglo tri-dimensional puede ser considerado un vector cuyas componentes son vectores de vectores, etc.

- Representación por filas (vector de filas)
- Representación por columnas (vector de columnas)

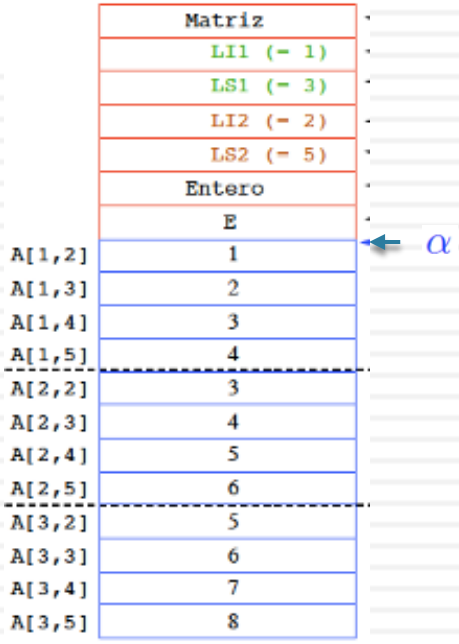
Representación por filas de una matriz

Ejemplo: var A: array [1 .. 3, 2 .. 5] of integer;

	2	3	4	5
1	1	2	3	4
2	3	4	5	6
3	5	6	7	8

Acceso a la componente A[i,j] de la matriz A

A partir de α debo "saltar" $(i - LI1)$ filas de tamaño S y $(j - LI2)$ elementos de tamaño E.



S = cantidad de componentes por fila $\times$ tamaño de componente = $(LS2 - LI2 + 1) \times E$

$$dir(A[i,j]) = \alpha + (i - LI1) \times S + (j - LI2) \times E$$

Ejemplo

$$dir(A[i,j]) = \alpha + (i - LI1) \times S + (j - LI2) \times E$$

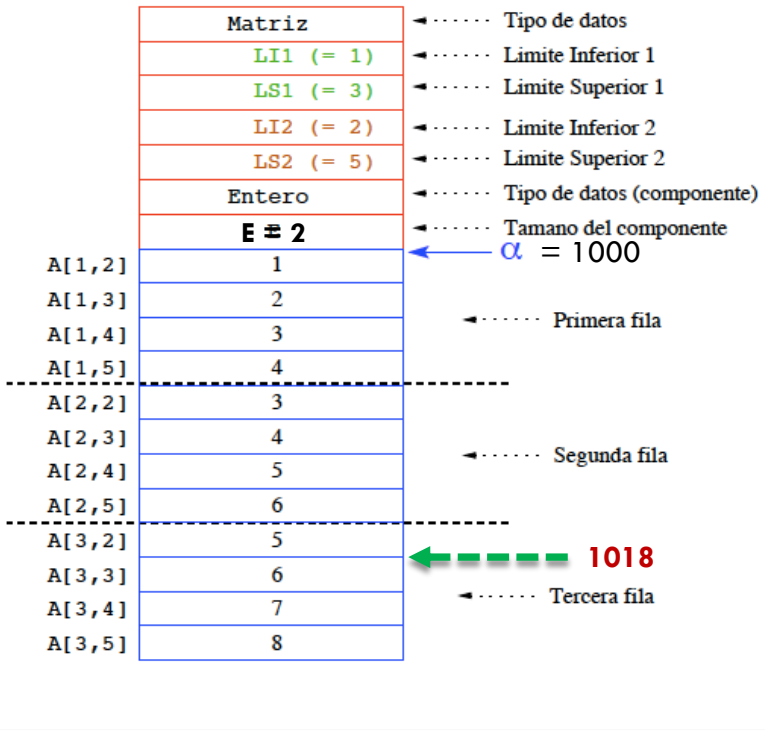
$$(LS2 - LI2 + 1) \times E$$

dir A[3,3] = 1000 + (3-1) x S + (3 - 2) x 2
= 1000 + 2 x 8 + 1 x 2
= 1000 + 18 = **1018**

S = tamaño de fila (cantidad de componentes por fila x tamaño de componente)

S = $(LS2 - LI2 + 1) \times E$

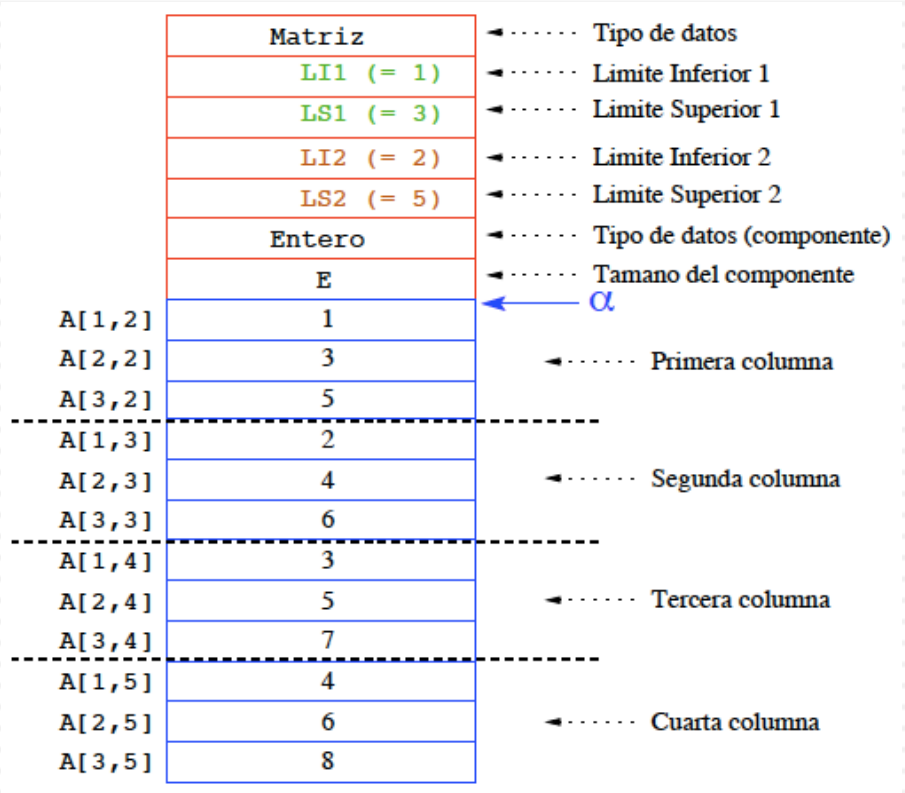
S = (5 – 2 +1) x 2 = 4 x 2 = **8**



Representación por columnas de una matriz

Ejemplo: var A: array [1 .. 3, 2 .. 5] of integer;

	2	3	4	5
1	1	2	3	4
2	3	4	5	6
3	5	6	7	8



Registros

Especificación y Sintaxis

Son estructuras de datos lineales y de longitud fija. Difieren de los arreglos en dos aspectos:

- Las componentes de los registros pueden ser heterogéneas.
- Las componentes se designan con nombres simbólicos (identificadores).

Atributos:

- El número de componentes.
- El tipo de datos de cada componente.
- El selector que se usa para nombrar cada componente.

Operaciones

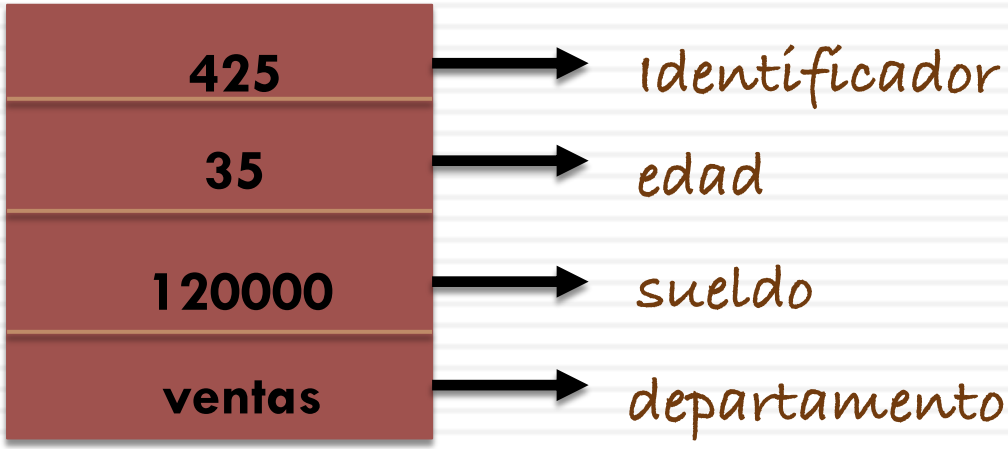
La operación básica es la selección de componentes. La manera de acceder a una componente es vía un nombre y no un valor computado.

Por lo general son pocas las operaciones sobre registros completos, la más común es la asignación de registros de estructura idéntica. Cobol.

Implementación de registros

La representación de almacenamiento consiste en un solo bloque secuencial de memoria, donde se guardan las componentes una a continuación de la otra.

Ejemplo:



Selección de una componente

La selección de componentes es sencilla debido a que los nombres de campos se conocen durante la traducción.

Además la declaración de los registros permiten determinar el tamaño de las componentes y su posición dentro del bloque de almacenamiento durante la traducción. Fórmula

de acceso para la *i*-ésima componente:

$$dir(R.i) = \alpha + \sum_{j=1}^{i-1}(tamR.j)$$

O directamente:

$$dir(R.i) = \alpha + K_i$$

Conjuntos

Un conjunto es un objeto de datos que contiene una colección no ordenada de valores distintos:

Las operaciones básicas sobre conjuntos son:

- Pertenecia
- Inserción y eliminación de valores.
- Unión, intersección y diferencia de conjuntos.

Conjuntos en Pascal

Sintaxis:

```
type TipoSet = Set of tipo;
```

Ejemplos:

```
type dia = (lu,ma,mi,ju,vi,sa,do);

     conj-caract = Set of Char;
     digitos = Set of 0..9;
     dias = Set of dia;
```

Conjuntos en Pascal

Y pueden declararse variables de tipo `Set` de la siguiente manera:

```
var laborable : dias;
    letras : conj-caract;
    conj-num : digitos;
```

La asignación de una variable de este tipo puede ser:

```
laborable := [lu, ma, mi, ju, vi];
```

Conjuntos en Swift

Crear Sets:

```
var s1 = Set(["Luz","Sol"])
```

Añadir y eliminar elementos:

```
s1.insert("Liz") //s1 es ["Luz","Sol","Liz"]
s1.remove("Sol") //s1 es ["Luz","Liz"]
s1.removeAll() //s1 es []
```

Implementación de conjuntos

Representación de conjuntos por cadenas de bits: es apropiada cuando se sabe que el tamaño del universo subyacente de valores es pequeño.

Si tenemos N elementos en el universo $e_1, e_2, .., e_n$, un conjunto de elementos elegido de entre este universo se puede representar por medio de una cadena de bits de longitud N, donde el i-ésimo elemento de la cadena es 1 si e_i está en el conjunto y 0 en caso contrario.

Esta representación es la que usa Pascal.

Implementación de conjuntos

Representación de conjuntos codificada por distribución pseudo-aleatoria (hash): se usa cuando el universo subyacente de valores posibles es grande.

Se asigna memoria al menos dos veces más de lo que se espera usar, los elementos se dispersan al azar. El truco está en guardar cada nuevo elemento de manera tal que, su presencia o ausencia se pueda determinar después, de inmediato, sin realizar una búsqueda. Esto se realiza mediante la función de hash.

Esta representación es la que usa Lisp.

¿Dudas?